UNIVERSITATEA TEHNICĂ DIN TIMIŞOARA
Facultatea de Automatică şi Calculatoare

Autor:
Ing. IOAN Z. MIHU

# SOLUȚII DE IMPLEMENTARE A REȚELELOR NEURONALE PE ARHITECTURI SISTOLICE

TEZĂ DE DOCTORAT



Conducător ştiinţific:
Prof. Dr. Ing. Nicolae Budişan

1998

## Cuvânt înainte

Fiinţele vii evoluate îşi coordonează acţiunile punând în joc o cantitate considerabilă de celule nervoase, interconectate printr-un număr şi mai considerabil de conexiuni sinaptice. Acest sistem puternic interconectat de celule nervoase, este sediul unei intense activităţi electro-chimice, şi capacitatea sa de adaptare şi acţiune constituie un subiect de cercetare abordat cu deosebit interes.

Aşadar, principiul conexionismului se bazează pe faptul că activitatea colectivă a unor unităţi elementare simple, care se influenţează reciproc, poate conduce la un comportament variat şi complex.

Începând cu anii 1960, pe baza acestui principiu, au fost propuse mai multe modele care au cunoscut anumite succese în rezolvarea problemelor complexe şi insuficient fundamentate teoretic pentru a putea fi soluţionate prin metode tradiţionale.

Aceste modele fac în general apel la două faze de calcul: o fază de adaptare sau de învăţare care serveşte la stabilirea funcţiei pe care trebuie să o realizeze reţeaua neuronală pornind de la un set de exemple, apoi o fază de recunoaştere care exploatează funcţia învăţată pe un ansamblu de stimuli, în general mai vast decât setul de antrenament. Cele două faze pun în evidenţă caracteristicile de bază ale reţelelor conexioniste: învăţare din exemple şi capacitate de generalizare.

Pentru validarea acestor modele şi pentru aplicarea lor în problematica reală, inginerul face apel la mijloacele de procesare a informaţiei de care dispune. Prin urmare, simularea acestor structuri cu grad fin şi înalt de paralelism pe calculatoare clasice, dotate cu unul sau mai multe procesoare, nu permite atingerea unui nivel de performanţă suficient pentru abordarea aplicaţiilor reale.

Numai problemele care reclamă reţele neuronale de mici dimensiuni pot fi tratate cu succes pe sistemele de calcul clasice. Realizarea unor circuite VLSI, cu grad înalt de paralelism intern, specializate pentru emularea reţelelor conexioniste, şi care să poată funcţiona autonom sau integrate într-un neurosistem (calculator gazdă dotat cu un *hardware* accelerator dedicat algoritmilor neuronali), reprezintă deci un obiectiv important pentru înţelegerea şi exploatarea domeniului reţelelor neuronale artificiale.

Obiectivul acestei teze este de a propune un principiu de concepţie a circuitelor VLSI digitale, capabile să integreze un anumit număr de funcţii necesare emulării mai multor modele de reţele conexioniste. Ambele funcţii (de învăţare şi de recunoaştere) vor fi integrate pentru a permite realizarea unor eventuale sisteme (VLSI-uri) autonome.

Primul capitol descrie principalele modele neuronale în termenii resurselor revendicate şi a operaţiilor efectuate pe aceste resurse. Această etapă ne-a permis evidenţierea punctelor comune aferente acestor modele, în termeni structurali şi funcţionali (ai operaţiilor aritmetico - logice pe care le implică).

Al doilea capitol realizează o sinteză a celor mai importante realizări în domeniul neuroprocesoarelor digitale şi analogice şi stabileşte principiile de proiectare a neuroprocesoarelor sistolice.

În al treilea capitol se elaborează o arhitectură sistolică generală, dedicată reţelelor neuronale, din care vor deriva trei arhitecturi exploatabile.

În capitolul al patrulea se defineşte arhitectura sistolică optimă şi se realizează proiectarea, testarea funcţională şi evaluarea acestei arhitecturi.

Capitolul cinci este dedicat elaborării unei strategii de perfecţionare a reţelei sistolice dedicate algoritmilor conexionişti, prin *pipeline*-izarea fluxurilor de date şi de instrucţiuni în

interiorul reţelei sistolice, prin perfecţionarea structurii celulei sistolice şi prin implementarea unui mecanism performant (*pipeline*-izat) de interschimbare a matricilor sinaptice.

Capitolul şase face analiza unei propuneri alternative: Reţeaua Sistolică Liniară cu integrare de neuroni şi nu de sinapse.

Capitolul şapte este dedicat elaborării unor versiuni alternative pentru algoritmii neuronali clasici, versiuni care vor permite un grad mai înalt de paralelism la nivelul implementării *hardware* pe arhitectura sistolică anterior propusă.

Capitolul opt prezintă concluziile finale, contribuţiile originale ale autorului şi inventariază posibilele dezvoltări viitoare pornind de la studiul prezentat în această lucrare.

Am avut şansa ca pe parcursul activităţii mele profesionale şi ştiinţifice şi a pregătirii acestei teze să întâlnesc o serie de oameni deosebiţi al căror sprijin mi-a fost de mare ajutor pe un drum nu totdeauna uşor.

În primul rând aduc un omagiu reputaţilor profesori Alexandru Rogojan şi Vasile Pop, întemeietori ai şcolii timişorene de calculatoare şi dascăli care mi-au călăuzit primii paşi în acest domeniu.

Domnului profesor Nicolae Budişan îi sunt profund recunoscător pentru curajul care mi l-a insuflat în momentele cele mai dificile, pentru răbdare şi pentru încrederea cu care m-a onorat atunci când i-am devenit doctorand. Îndrumarea şi ajutorul de care m-am bucurat în toţi aceşti ani şi-au pus amprenta în mod decisiv asupra tezei pe care o prezint astăzi.

Le sunt recunoscător bunilor mei prieteni prof. Ioan Mureşan, prof. Nicolae Robu, prof. Florin Breabăn, prof. Octavian Proştean şi conf. Daniel - Ioan Curiac din cadrul şcolii timişorene de automatică pentru ajutorul şi încurajările de care m-am bucurat în toţi aceşti ani.

Nu pot să nu amintesc excelenta colaborare, căldura şi prietenia cu care am fost primit în câteva instituţii universitare de prestigiu din Belgia, unde am desfăşurat o bună parte din activitatea de cercetare prezentată în această lucrare. Le sunt profund recunoscător domnilor profesori Georges Hennuy şi Jean - Benoît Cabo de la *ECAM Bruxelles*, Dragomir Mirojevic de la *Univerité Libre Bruxelles* şi Michel Verleysen de la *Université Catolique de Louvain - la - Neuve* pentru şansa de a lucra într-un mediu de înalt profesionalism şi de a obţine rezultate care nu ar fi fost posibile fără această colaborare.

Mulţumesc de asemenea şi colegilor mei de la Universitatea "Lucian Blaga" din Sibiu care, într-un fel sau altul, m-au ajutat în munca de elaborare a acestei lucrări; tuturor le sunt profund recunoscător.

Cuvintele mele sunt prea modeste pentru a putea exprima recunoştinţa ce o datorez familiei. Părinţilor mei le sunt profund recunoscător pentru grija, atenţia şi căldura cu care mi-au îndrumat paşii în viaţă. Lor le datorez şi le dedic realizările mele de astăzi. Soţiei mele îi mulţumesc pentru fericirea deplină pe care a adus-o în viaţa mea. Iar în ceea ce priveşte pe copiii noştri, Diana şi Lucian, totul devine foarte simplu: abia prin ei totul capătă un sens.

Sibiu, 25.06.1998

Autorul

**CUPRINS**

# INTRODUCERE

Tehnicile concrete sau punctuale au fost întotdeauna inspirate de vise eterne. Zborul lui Icar, cucerirea spaţiului cosmic şi a planetelor, înţelegerea universului au constituit din totdeauna motoare puternice care au împins ştiinţa spre ungherele cele mai adânci ale cunoaşterii.

Iată că, după aspiraţia fiinţei sale spre imensitatea externă, omul încearcă descoperirea abisului său interior dând ştiinţei un nou pretext: maşinile de procesare a informaţiei nu sunt adaptabile. Ori, în sfera lumii cunoscute, omul pare a fi un rezultat al adaptării. El nu se adaptează numai din necesitate ci şi prin voinţă. În mod esenţial, el datorează această aptitudine creierului său şi, într-un sens mai larg, sistemului său nervos. Ce poate fi deci mai natural decât a-l observa, a-l înţelege şi a extrage principiile directoare pentru construcţia calculatoarelor de mâine.

Conexionismul a cunoscut în ultimii ani un interes deosebit, în principal datorită tehnicilor de investigare dezvoltate cu ajutorul informaticii tradiţionale. Dar, puterea de calcul a maşinilor tradiţionale, care au permis de altfel revenirea la principiul conexionist, a început să devină o frână în dezvoltarea acestuia.

Bazate pe arhitecturi cu grad înalt şi fin de paralelism, modelele conexioniste sunt dezvoltate de regulă pe maşini secvenţiale, vectoriale, cu grad grosier de paralelism. Ori, datorită potenţialului pus în evidenţă prin simulare, reţelele conexioniste nu pot fi pe deplin exploatate decât pe maşini care reflectă concret construcţia internă a acestora.

Trebuie deci construite maşini optimale cu un grad înalt şi fin de paralelism; acestea trebuie să fie optimale în sensul eficienţei şi a vitezei de execuţie. Scopul acestei lucrări este de a da un răspuns parţial la această problemă. Demersul nostru este canalizat pe 4 direcţii:

- studiul modelelor conexioniste;
- elaborarea unei strategii de implementare a reţelelor neuronale pe arhitecturi sistolice;
- perfecţionarea reţelelor sistolice dedicate algoritmilor neuronali;
- perfecţionarea algoritmilor neuronali dedicaţi implementării reţelelor neuronale pe reţele sistolice.

## Reţele Neuronale în Automatică

În câmpul cunoaşterii suntem în prezent martorii declanşării unor energii creatoare nebănuite prin fenomenul de fuziune al nucleelor unor discipline ştiinţifice. Considerate iniţial de sine stătătoare, pe o anumită treaptă a dezvoltării acestea îşi descoperă surprinzătoare rădăcini şi fundamentări comune, ceea ce pune în evidenţă unitatea substanţială şi structurală a lumii materiale.

Teoria sistemelor reprezintă un astfel de produs al interacţiunii interdisciplinare care, într-o viziune structuralistă integratoare adâncită până la nivelul cantitativ, reuşeşte să impună perenitatea conceptului de sistem şi a celor asociate la nivel de categorii fundamentale în analiza şi sinteza fenomenelor lumii materiale, la diverse nivele de organizare a acesteia.

Specialistul în automatică este un om cu o pregătire ştiinţifică generală, cu preocupări şi experienţă în multe ramuri bine delimitate ale ştiinţei. Pentru el sistemul poate îmbrăca cele mai variate forme fizice: electric, mecanic, biologic, economic, chimic, neuronal sau de orice altă natură. Prima sa sarcină este de a modela matematic sistemul şi de a-i extrage caracteristicile fundamentale. Pentru el prezintă mare importanţă tipul sistemului sau procesului studiat: liniar sau neliniar, discret sau continuu în raport cu timpul, concentrat sau distribuit, determinist sau stohastic.

cu stări continue sau discrete, pasiv sau activ. Automatistul va utiliza toate aceste instrumente în munca sa de concepţie şi proiectare a sistemelor automate destinate conducerii proceselor.

*Sistemele neuronale artificiale, denumite succint reţele neuronale, sunt sisteme fizice celulare capabile să dobândească, să memoreze şi să utilizeze cunoştinţe.* Cunoştinţele, stocate sub forma unui ansamblu de stări stabile sau a unor funcţii de transfer încapsulate în reţea, pot fi apelate prin aplicarea unor vectori pe intrarea reţelei. Răspunsul acesteia la stimulii aplicaţi pe intrare se obţine pe baza cunoştinţelor acumulate sau a comportamentului anterior învăţat.

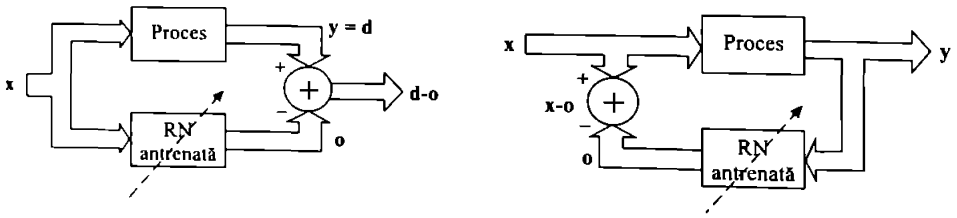
Din punctul de vedere al automatistului reţeaua neuronală este un sistem sau un subsistem destinat integrării într-un sistem. Reţeaua *feedforward* este un sistem capabil să înveţe un anumit comportament, cu alte cuvinte, să-şi însuşească o anumită funcţie de transfer. Reţelele neuronale recurente nu sunt altceva decât sisteme dinamice, continue sau discrete. Aplicarea unui vector pe intrarea reţelei înseamnă de fapt forţarea acesteia într-o anumită stare. Reţeaua va evolua apoi liber spre o stare de echilibru numită *atractor*. Fiecare atractor are propriul său set de condiţii iniţiale, care iniţiază evoluţii ale reţelei ce sfârşesc în atractorul respectiv. Acest set de condiţii iniţiale aferente unui atractor este denumit *bazin de atracţie*. Dacă atractorul este reprezentat de un punct unic în spaţiul stărilor, atunci el se numeşte atractor fix. Atractorul poate fi reprezentat şi de o secvenţă periodică de stări, caz în care el va fi denumit atractor ciclic, şi poate avea şi o structură mai complicată. Drept consecinţă, toate metodele matematice de studiu aplicate în domeniul sistemelor dinamice pot fi aplicate şi în domeniul reţelelor neuronale recurente. Criteriile de stabilitate aplicate în domeniul sistemelor dinamice continue sau discrete sunt valabile şi în domeniul reţelelor neuronale recurente. Iată deci un prim grad de apropiere, materializat într-un amplu set de similitudini care se cantonează în sfera fundamentelor teoretice.

Cel de-al doilea set de similitudini se cantonează în sfera aplicaţiilor. Reţelele neuronale sunt aplicate cu succes la modelarea, identificarea, controlul şi optimizarea proceselor; ele pot fi de asemenea utilizate în construcţia dispozitivelor de comandă aferente roboţilor industriali capabili să înveţe din experienţă.

### A. Identificarea Proceselor

În general, prin identificarea unui proces trebuie să se obţină coeficienţii ecuaţiei diferenţiale aferentă procesului respectiv sau coeficienţii funcţiei sale de transfer. Identificarea proceselor este de foarte mare importanţă în cazul sistemelor de control adaptive, deoarece parametrii procesului sunt variabili în timp şi controlul adaptiv trebuie ajustat în funcţie de variaţia parametrilor procesului.

Schema de principiu utilizată pentru identificarea funcţiei de transfer directe aferente unui proces este redată în figura I.1. a).




a) identificarea caracteristicii directe b) identificarea caracteristicii inverse

Fig. I.1. Configurarea reţelei neuronale pentru identificarea proceselor

Reţeaua neuronală neliniară recepţionează aceeaşi intrare **x** ca şi procesul identificat. În faza de antrenare a reţelei, procesul va furniza ieşirea dorită **d**. Scopul identificării constă în a găsi reţeaua neuronală care generează răspunsul **o**, identic cu răspunsul procesului **y**, pentru un set dat de vectori de intrare **x**. Pe durata fazei de identificare, norma vectorului de eroare || **d** - **o** || va fi minimizată prin ajustarea ponderilor sinaptice (învăţare). Să notăm că pot fi utilizate şi scheme mai complexe pentru identificarea proceselor. Pentru a lua în considerare dinamica proceselor, pe intrarea reţelei neuronale se va aplica un set de vectori de intrare **x** întârziaţi cu diverse cuante de timp. Întârzierile se pot realiza prin inserarea unei cascade de elemente de întârziere pe intrarea procesului.

În multe situaţii, identificarea caracteristicii inverse a procesului, oferă o alternativă viabilă pentru proiectarea sistemului de conducere aferent procesului respectiv. Pentru identificarea caracteristicii inverse (fig. I.1. b)), ieşirea procesului **y** va fi utilizată drept intrare în reţeaua neuronală. Vectorul de eroare va fi **x** - **o**, unde **x** reprezintă intrarea procesului. Învăţarea reţelei are drept scop minimizarea normei vectorului de eroare || **x** - **o** ||. După antrenare reţeaua neuronală va găzdui caracteristica inversă aferentă procesului identificat.

### B. Controlul Proceselor

În figura I.2. se prezintă un sistem de conducere *feedforward* bazat pe o reţea neuronală.

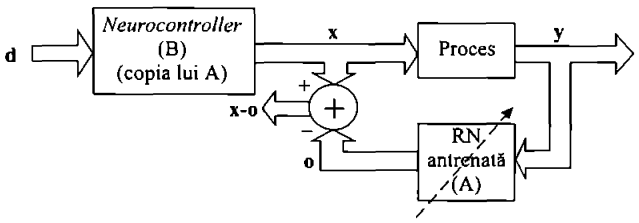



Fig. I.2. Arhitectură de conducere cu învăţarea inversei caracteristicii procesului

*Neurocontroller*-ul B este o copie exactă a reţelei neuronale A, care suportă învăţarea. Reţeaua A este astfel antrenată încât să realizeze caracteristica inversă a procesului condus. Vectorul de intrare în *neurocontroller* va fi chiar răspunsul dorit la ieşirea procesului (**d**). Răspunsul real al procesului va fi **y** şi se datorează intrării **x** generate de către *neurocontroller*-ul B. Eroarea utilizată la antrenarea reţelei neuronale va fi diferenţa dintre semnalele de ieşire aferente reţelelor A şi B (**x** - **o**). Deoarece reţeaua B urmăreşte (copiază) reţeaua A după fiecare pas de învăţare, **y** trebuie să fie identic cu **d** pentru ca eroarea să fie nulă. Să notăm că această configuraţie de conducere devine inutilizabilă atunci când caracteristica inversă aferentă procesului nu este unic definită. Această arhitectură de conducere se mai numeşte şi arhitectură cu învăţare indirectă şi are câteva avantaje evidente. Reţeaua neuronală A poate fi antrenată *on-line* în timp ce *neurocontroller*-ul realizează conducerea procesului; deci nu este necesară o fază de învăţare separată. Suplimentar, vectorii de intrare în *neurocontroller* sunt chiar vectorii doriţi la ieşirea procesului. Deci, antrenarea reţelei se va realiza chiar în regiunea de interes din domeniul vectorilor de ieşire. O astfel de învăţare, realizată în regiunea de interes, se numeşte învăţare specializată. Mai mult, reţeaua neuronală învaţă continuu şi este prin urmare adaptivă.

O arhitectură derivată din cea prezentată în fig. I.2. şi care realizează simultan conducerea procesului şi învăţarea specializată a domeniului de ieşire aferent unui proces static este prezentată în figura I.3. a). Această arhitectură posedă toate avantajele celei descrise anterior, dar utilizează o singură copie a reţelei neuronale. Obiectivul acestei arhitecturi de conducere cu învăţare specializată este de a obţine vectorul dorit **d** la ieşirea procesului, în condiţiile în care intrarea **x** care

generează această ieşire este necunoscută. Pentru procese dinamice, pe intrarea reţelei antrenate trebuie aplicat şi setul de ieşiri aferente procesului, întârziate cu diferite cuante de timp (fig. I.3. b)). Aceasta va permite reconstrucţia stării curente aferente procesului, pe baza celei mai recente istorii derulate la ieşirea acestuia. În figura I.3. b) se redă arhitectura de conducere cu învăţare specializată pentru un proces dinamic de ordinul k. În acest caz, pe intrarea *neurocontroller*-ului se vor aplica, pe lângă vectorul **d**, şi ultimii k vectori obţinuţi la ieşirea procesului. Cuantele de timp Δ plasate în buclele de reacţie, specifică timpul scurs între eşantioanele (prototipurile) aplicate la intrare în timpul învăţării; ele specifică de asemenea că procesul este controlat în timp real.

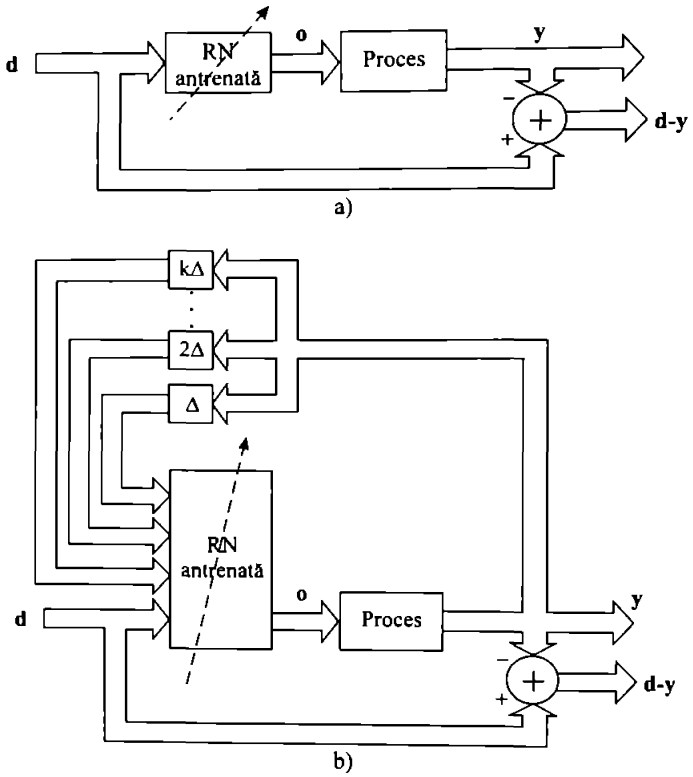



Fig. I.3. Arhitectura de conducere cu învăţare *on-line* specializată
a) procese statice b) procese dinamice

Arhitectura de conducere din fig. I.3. are anumite inconveniente care pot conduce la o învăţare lentă dacă procesul nu este cunoscut decât aproximativ. Algoritmii de învăţare supervizată (ca de exemplu EBP - *Error Back Propagation*) nu pot fi utilizaţi direct deoarece numai eroarea dintre ieşirea reală **y** şi cea dorită **d** este cunoscută utilizatorului; eroarea **d** - **y** trebuie mai întâi propagată şi transpusă la intrarea procesului (furnizată algoritmului EBP).

Să notăm că în literatura de specialitate sunt propuse şi alte arhitecturi de conducere; unele dintre acestea combină sistemele de conducere clasice cu cele bazate pe *neurocontroller*-e în ideea obţinerii unor performanţe superioare.

### C. Reţele Neuronale în Robotică

Deşi reţelele neuronale utilizate în robotică sunt, de fapt, *neurocontroller*-e, acestea sunt specializate în rezolvarea problemelor pe care le implică mişcarea braţului unui robot. În vederea obţinerii unor performanţe deosebite, cinematica joacă un rol esenţial în proiectarea şi controlul unui robot. Controlul clasic al traiectoriei braţului robotic constă în succedarea unei secvenţe programate de mişcări. Controlul unui robot revendică generarea unor semnale de comandă aplicate dispozitivelor de acţionare a încheieturilor robotului, atunci când se specifică o traiectorie dorită sau o secvenţă de poziţii dorită. Braţele robotului operează în plan sau spaţiu. Pentru a mişca braţul, coordonatele finale (x, y, z) sunt aplicate *controller*-ului şi acesta va genera unghiurile ($\theta_1$, $\theta_2$, $\theta_3$) pentru comanda motoarelor care acţionează elementele braţului. Pentru a controla mişcarea braţului spre o poziţie finală, trebuie rezolvate două probleme:

1. problema cinematicii inverse: Fiind date coordonatele carteziene, specificate fie ca punct final al traiectoriei fie ca o succesiune de puncte de pe o traiectorie, trebuie generate unghiurile, respectiv succesiunea de unghiuri, aferente încheieturilor;

2. controlul poziţiei finale: Fiind dată poziţia finală a braţului robotului, trebuie generată o secvenţă corespunzătoare de unghiuri necesară atingerii acestei poziţii.

Problemele cinematicii directe şi inverse pot fi soluţionate eficient cu ajutorul reţelelor neuronale *feedforward* multistrat. Modelarea traiectoriilor dorite precum şi transformările cinematice pot fi rezolvate cu un înalt grad de acurateţe cu ajutorul *neurocontroller*-elor. Valorile dorite la ieşire (**d**), în cadrul învăţării supervizate, pot fi furnizate fie prin măsurători geometrice fie prin rezolvarea ecuaţiilor cinematice directe sau inverse (pentru zona de lucru a robotului).

Reţelele neuronale pot fi utilizate în robotică şi pentru învăţarea unei secvenţe de transformări. Procesul poate fi privit ca o învăţare a unei secvenţe de transformări în spaţiu. Prin urmare, învăţarea unei poziţii finale poate fi privită ca o învăţare a unui control dinamic. Braţele robotului pot fi tratate ca un proces dinamic şi pot fi comandate să execute o secvenţă de paşi, cu scopul de a atinge poziţia finală dorită. Reţelele neuronale sunt capabile să memoreze diversele secvenţe de semnale de comandă necesare atingerii unor poziţii finale specificate.

Din cele prezentate până aici, se poate constata un grad înalt de întrepătrundere între domeniul automaticii şi cel al reţelelor neuronale. Aceste întrepătrunderi se cantonează atât în sfera fundamentelor teoretice cât şi în sfera aplicaţiilor. Teoria sistemelor furnizează un set larg de concepte şi de instrumente matematice necesare fundamentării teoretice a reţelelor neuronale. Pe de altă parte, reţelele neuronale rezolvă cu succes multe dintre obiectivele vizate în cadrul automaticii: modelarea şi identificarea proceselor, conducerea şi optimizarea proceselor, robotică etc. Iată de ce considerăm justificată şi oportună încadrarea acestei lucrări în sfera mai largă a sistemelor automate.

# 1. MODELE NEURONALE

Din toate timpurile, creierul a constituit un subiect de reflecţie pentru om. Încă din antichitate medicii acordau o atenţie particulară acestui organ în care ei au descoperit sursa funcţiilor de comandă. Conexionismul reprezintă astăzi versiunea tehnologică a acestei cercetări. Dacă obiectivul este ambiţios, dezvoltările actuale sunt încă sărace. Cunoaşterea neuro-psihicului a cunoscut o dezvoltare considerabilă în ultimii 15 ani, dar modelele inginereşti datează de cel puţin 50 ani. Ecartul între observaţia clinică a creierului şi modelele inginereşti este deci destul de mare.

Totuşi, cu toată sărăcia rezultatelor obţinute, impasul în care se găseşte actualmente calculul formal nu ne oferă o altă alternativă. Trebuie căutat, în toate direcţiile posibile, un nou principiu de abordare capabil să asigure uneltelor de tratare a informaţiei autonomia şi adaptabilitatea pe care încă nu le au. Conexionismul este încă mai mult speranţă decât tehnică.

## 1.1. Principiile conexionismului

Toată informatica contemporană este bazată pe principiul "maşinii *Von Neumann*". Acest concept cvasi-universal a condus la un model unic de programare a maşinilor:

- stabilirea unui model pentru fenomenul de simulat;
- transcrierea acestui model într-o secvenţă de instrucţiuni executate pe maşina sau maşinile suport.

Acest principiu operează excelent atunci când se pot extrage din realitatea observată modelele care satisfac exigenţele exprimate. Dar modelele devin tot mai complexe şi, după toate evidenţele, noţiunea de modelare este limitată. Cu tot progresul tehnologic şi arhitectural pe care-l înregistrează, calculatoarele "*Von Neumann*" nu fac faţă cerinţelor care devin tot mai complexe.

O soluţie alternativă o propune conexionismul care nu apelează la modele şi care propune o soluţie de "învăţare din exemple". Sursa de inspiraţie a conexionismului este celula nervoasă. O reţea neuronală (RN) este formată dintr-un număr mare de elemente de procesare (EP) simple, numite neuroni, conectate între ele printr-o reţea complexă de conexiuni locale. În sistemele tradiţionale (*Von Neumann*), existenţa unei unităţi de control unice şi a unui tact global conduce spre un comportament individual. Resursele implicate în transferurile de date sunt puţine şi trebuie gestionate sever. Suplimentar, transferurile de date trebuie sincronizate iar resursele eliberate la sfârşitul fiecărui transfer. La antipod, sistemele conexioniste exploatează reprezentarea distribuită a informaţiilor, asociată cu o evoluţie complet asincronă care le îndepărtează de noţiunea de control global. Elementele EP sunt numeroase precum şi legăturile (transferurile de informaţii) dintre acestea. Obţinerea unei soluţii nu va fi un proces previzibil şi cuantificabil în manieră deterministă, ci va fi mai mult efectul de manifestare a unei tendinţe pe care l-am putea considera ca o satisfacere aproximativă a soluţiei.

Calculul local simplifică complexitatea elementelor EP (neuronilor) dar creşte densitatea căilor de comunicaţie; conexiunea îşi pierde statutul de operator ideal pe care îl avea în cadrul sistemelor tradiţionale. Mii de neuroni trebuie conectaţi între ei iar legăturile vor ocupa în mod esenţial spaţiul disponibil. Reprezentând în sistemele clasice volume şi costuri aproape nule, interconexiunile par a deveni "tendonul lui Achile" în cazul reţelelor conexioniste. Este de dorit deci localizarea calculelor şi a schimburilor de date în scopul reducerii interconexiunilor la nivelul unei simple comunicaţii între procesoarele vecine.

Învăţarea în cazul RN constă în ajustarea parametrilor reţelei. Regulile şi algoritmii utilizaţi în procesul de învăţare (antrenare) înlocuiesc programarea tradiţională din cazul calculatoarelor

convenţionale. Procesarea în cadrul RN se situează undeva între inginerie şi inteligenţă artificială. Tehnicile matematice utilizate sunt inginereşti dar aplicarea unui principiu experimental adhoc se dovedeşte adesea necesară.

RN au atras atenţia oamenilor de ştiinţă dintr-un număr relativ mare de domenii şi sunt pe cale să devină un instrument de lucru în inginerie. Inginerii sunt de asemenea interesaţi în construcţia circuitelor electronice neuronale destinate maşinilor inteligente. RN sunt capabile să înveţe din experimente, să memoreze, să generalizeze şi să utilizeze cunoştinţele dobândite.

### 1.2. Scurt istoric

Geneza ideii de conexionism datează din anii 1940 şi are la bază observaţiile neuro-fiziologice ale lui *Warren McCulloch* [McC 43]. Au urmat apoi lucrările de definire a perceptronului elaborate de *Franck Rosenblatt* [ROS 58], care au pus bazele teoretice pentru dezvoltările actuale. În paralel, şi într-un alt cadru de cercetare *Bernard Windrow* propune un model foarte asemănător, modelul *Adaline*. Perceptronul original constituia ieşirea unei retine artificiale formată din 400 fotoreceptori şi era capabil să recunoască motive simple. Această muncă, iniţial artizanală, a fost profund reanalizată în lucrările teoretice ale lui *Block* [BLO 62]. Din nefericire entuziasmul lui *Rosenblatt* a fost un pic cam excesiv. El nu a sesizat posibilităţile limitate ale perceptronului; puterea acestuia, lui i se părea extraordinară. În acelaşi timp, el era de asemenea conştient de câteva dintre slăbiciunile inerente ale modelului său. El compara perceptronul cu modelul unui creier defect. Analogia era ambiţioasă dar conţinea problematica esenţială a conexionismului: se poate simula un comportament inteligent pornind de la un ansamblu de elemente simple?

Problema enunţată de *Rosenblatt* a constituit un subiect de controversă în perioada respectivă. Controversa a fost stopată brutal de lucrările lui *Minsky* şi *Papert*: "*Perceptrons*" [MIN 88]. Ceea ce îi miră încă pe cercetătorii de astăzi, este brutalitatea cu care a fost abandonat perceptronul. Opera lui *Minsky* şi *Papert* conţine abordări matematice şi vorbeşte de limitele de generalizare ale modelului. Ea arată explicit ceea ce perceptronul nu poate să facă, dar nu-l respinge.

Din nefericire, în lucrarea "*Perceptrons*", *Minsky* şi *Papert* comit aceeaşi eroare ca şi *Rosenblatt*, dar excelează de această dată prin pesimism. Ei explică faptul că perceptronul monostrat are limite şi că aceste limite pot fi depăşite dacă se construiesc perceptroni multistrat; cu condiţia determinării experimentale (manuale) a parametrilor corecţi deoarece nu este posibilă determinarea automată şi deci găsirea unui algoritm de învăţare pentru o astfel de structură. A trebuit să se aştepte anul 1986, cu lucrările lui *Rumelhart* [RUM 86] şi *Le Chun* [LeC 85] pentru a găsi un răspuns la această problemă: da este posibilă găsirea unui algoritm şi a unei reguli de învăţare pentru reţeaua de perceptroni multistrat; este algoritmul retro-propagării gradientului sau erorii (EBP - *Error Back-Propagation*). Diversele variante ale acestui algoritm sunt încă şi astăzi studiate.

Au fost propuse şi alte arhitecturi de RN. De exemplu, la începutul anilor 1970, *Kohonen* preia ideea lui *Hebb* şi a lui *Willshaw* şi propune o reţea cu 2 straturi, dintre care unul are o structură de conexiuni laterale inhibitorii recurente. Este o arhitectură care se detaşează de celelalte prin capacitatea ei de învăţare nesupervizată (fără profesor).

*J.J. Hopfield* în anii 1980, relansează interesul pentru reţelele conexioniste propunând modelul de reţea care îi poartă astăzi numele. Total rebuclată, această structură recurentă are o dinamică proprie care se încearcă a fi exploatată prin forţarea stărilor stabile (atractorilor) pe anumite valori cărora li se pot atribui anumite semnificaţii.

Astăzi cercetătorii se întrec între ei încercând să regăsească filonul inspiraţiei din neuro-fiziologie. *Herault* şi *Jutten* au propus un algoritm de separare a surselor, direct inspirat de captorii proprio-receptori. *Grossberg* lucrează în strânsă colaborare cu cercetătorii psihologi la elaborarea

modelelor ART (*Adaptive Resonance Theory*). *Mead* copiază structura retinei pentru construcția circuitelor VLSI analogice.

Domeniul este deci în plină evoluție; el trebuie să-şi regăsească sursele de inspirație originale, printr-o strânsă colaborare între biologi, ingineri şi matematicieni şi trebuie găsite rapid aplicații convingătoare pentru a-i asigura perenitatea şi pentru a evita o nouă cădere în unul sau mai multe decenii de uitare. În capitolele următoare vom defini structurile de bază aferente modelelor conexioniste precum şi diversele reguli de învățare exploatabile pe aceste structuri. Vom studia de asemenea şi modelarea matematică a algoritmilor de învățare.

### 1.3. Modele conexioniste

#### 1.3.1. Modelul McCulloch - Pitts

Modelul *McCulloch - Pitts* [McC 43] este cu certitudine cel mai vechi model şi constituie elementul de bază al rețelelor conexioniste. Acesta descrie neuronul ca pe un element sumator care realizează o sumă ponderată a intrărilor; intrările traversează conexiunile sinaptice iar coeficienții de ponderare vor fi ponderile sinaptice (fig. 1.1).

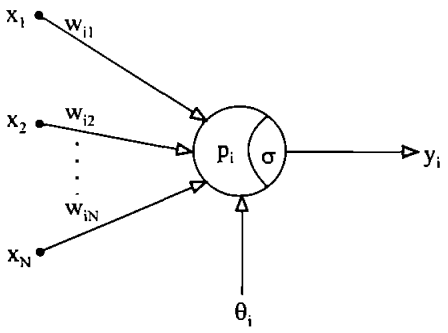



Fig. 1.1. Modelul general al neuronului

Dacă notăm:

$x_j$ - valoarea stimulului de la intrare

$w_{ij}$ - valoarea ponderii sinaptice dintre intrarea j şi neuronul i

$\theta_i$ - o valoare numită valoare de prag a neuronului i

atunci, potențialul neuronului i va fi:

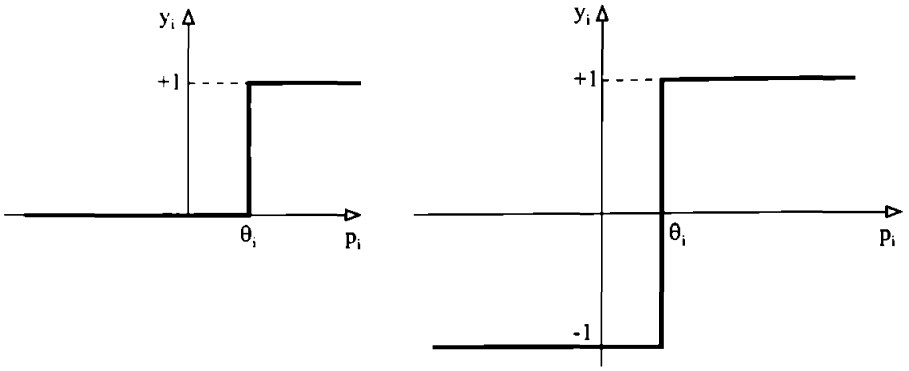
$$p_i = \sum_{j=1}^{N} w_{ij} \cdot x_j \qquad (1.1)$$

Ieşirea efectivă a neuronului se calculează comparând potențialul $p_i$ cu pragul $\theta_i$. Dacă potențialul depăşeşte valoarea de prag atunci ieşirea $y_i$ va fi +1, dacă nu va fi 0.

$$y_i = 1 \text{ dacă } p_i > \theta_i$$
$$y_i = 0 \text{ dacă } p_i \leq \theta_i$$

Funcția de transfer (activare) $\sigma$ este o funcție treaptă care generează valori binare la ieşirea neuronului. Aici funcția $\sigma$ este definită ca în fig. 1.2. a). Această funcție reprezintă o aproximare a răspunsului intrare - ieşire aferent neuronului real.

În general, se consideră că funcţia neuronală reprezintă o variaţie de frecvenţă, medie sau instantanee, în jurul valorii de răspuns aferentă neuronului real. Astfel, aceste variaţii vor fi pozitive în momentele de excitaţie şi respectiv negative în momentele de inhibiţie. Să notăm că funcţiile de transfer de tip neliniar, cum este cea prezentată în fig. 1.3, sunt destul de aproape de caracteristica intrare - ieşire a neuronului real.




a) *hard limiter* b) *sign*

Fig. 1.2. Reprezentarea funcţiei de transfer (activare)

Funcţia de transfer poate lua mai multe forme, care depind de reţea, de natura continuă sau discretă a valorilor potenţialului, de aplicaţia vizată etc. O altă variantă simplă a funcţiei de activare este funcţia *sign*, prezentată în fig. 1.2. b). Ecuaţia generală a acestei funcţii este de forma:

$$y_i(p) = \text{sign}\,(p_i - \theta_i) \tag{1.2}$$

Această funcţie generează valori binare pe ieşirea neuronului.

Adesea utilizată în modelele complexe, funcţia sigmoidală este destul de aproape de caracteristica intrare - ieşire a neuronilor reali. Forma analitică a acestei funcţii este:

a) funcţia sigmoidală unipolară:

$$\sigma(p) = \frac{1}{1 + e^{\frac{-(p-\theta)}{T}}} \tag{1.3}$$

b) funcţia sigmoidală bipolară:

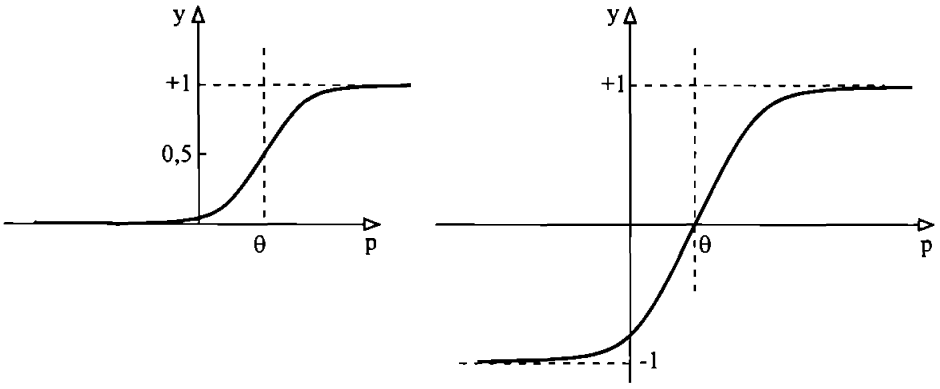
$$\sigma(p) = \frac{2}{1 + e^{\frac{-(p-\theta)}{T}}} - 1 \tag{1.4}$$

unde:

θ reprezintă valoarea de prag
T controlează panta în origine

Să notăm că în anumite lucrări se notează $\lambda = \frac{1}{T}$.

În fig. 1.3. sunt reprezentate cele 2 variante ale funcţiei sigmoidale.




a) unipolară b) bipolară

Fig. 1.3. Funcţia de activare sigmoidală

### 1.3.2. Funcţia neuronului

Funcţia unui neuron *McCulloch - Pitts* poate fi interpretată uşor în termeni geometrici. Să considerăm spaţiul $R^N$ înzestrat cu produsul scalar şi norma euclidiană. Vom considera valorile de intrare $x_1, x_2, \ldots, x_N$ ca şi componente ale vectorului **x** şi ansamblul de coeficienţi sinaptici $w_{i1}, w_{i2}, \ldots, w_{iN}$ ca şi componente ale vectorului $\mathbf{W}_i$. În acest caz potenţialul neuronului i va fi:

$$p_i = \langle \mathbf{W}_i, \mathbf{x} \rangle = \|\mathbf{W}_i\| \cdot \|\mathbf{x}\| \cdot \cos(\mathbf{W}_i, \mathbf{x}) \tag{1.5}$$

Dacă $\|\mathbf{W}_i\| = 1$ şi $\|\mathbf{x}\| = 1$ atunci:

$$p_i = \cos(\mathbf{W}_i, \mathbf{x}) \tag{1.6}$$

Funcţia principală a neuronului *McCulloch - Pitts* este deci de a determina gradul de "similaritate de direcţie" între 2 vectori. Aşadar, neuronul care va genera valoarea cea mai mare pe ieşire (cel mai mare răspuns) va determina care dintre vectorii de ponderi sinaptice corespunde cel mai bine vectorului stimul aplicat pe intrare. Funcţia de transfer va avea deci rolul de a selecta unităţile (neuronii) care răspund la un anumit criteriu, adică acelea al căror potenţial depăşeşte un anumit nivel (valoarea de prag). Procesul de selecţie este descris prin figurile 1.4 până la 1.8; se consideră o RN în care toţi neuronii au 2 intrări (spaţiul $R^2$) şi ai căror vectori de ponderi sinaptice sunt iniţial orientaţi aşa cum indică fig. 1.4.

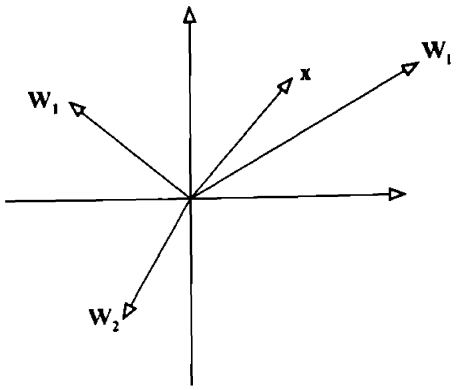



Fig. 1.4. Reprezentarea geometrică a vectorilor de intrare şi de ponderi sinaptice

Prima etapă constă în normalizarea vectorilor, inclusiv a vectorului **x** aplicat la intrare. Pentru normalizare este suficient să se calculeze (fig. 1.5):

$$\mathbf{x}' = \frac{\mathbf{x}}{\|\mathbf{x}\|} \quad \text{şi} \quad \mathbf{W}'_i = \frac{\mathbf{W}_i}{\|\mathbf{W}_i\|}$$

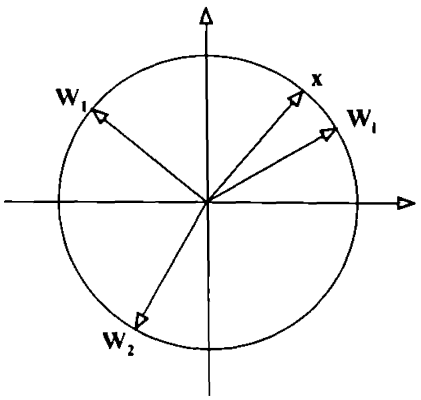



Fig. 1.5. Efectul normalizării vectorilor

Potrivit modelului *McCulloch - Pitts*, vom calcula cosinusul unghiurilor dintre vectorul aplicat la intrare şi vectorii ponderilor sinaptice. Vom obţine astfel un vector de potenţiale **p** care va conţine potenţialele tuturor neuronilor. Se aplică apoi funcţia de transfer care va avea ca efect selecţia unui subansamblu de neuroni în funcţie de valorile potenţialelor acestora. Dacă se utilizează, de exemplu, funcţia "*hard limiter*", (fig. 1.2.a)), şi pragurile $\theta_i$ sunt nule, vom clasifica neuronii aşa cum indică figura 1.6.

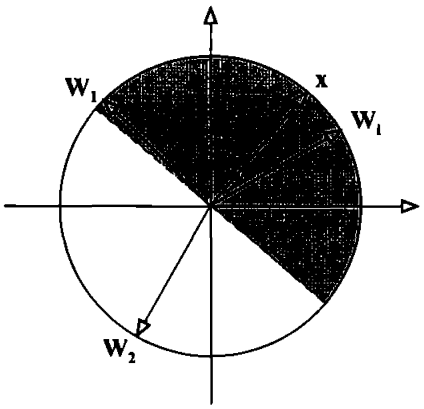



Fig. 1.6. Cele 2 zone din spaţiul ponderilor sinaptice separate prin valoarea de prag

Toţi neuronii ai căror vectori de ponderi sinaptice generează un cosinus pozitiv în raport cu vectorul de intrare **x** vor avea ieşirea egală cu 1, toţi ceilalţi vor avea ieşirea -1. În spaţiul ponderilor sinaptice se realizează o selecţie a unui subspaţiu (format din 2 cadrane) centrat pe vectorul **x**. Cu $\theta = 0$ spaţiul ponderilor sinaptice va fi împărţit în două părţi egale. Dacă se fixează o valoare de prag $\theta \neq 0$ identică pentru toţi neuronii, dimensiunea spaţiului selectat se va reduce proporţional. De exemplu, dacă se utilizează aceeaşi funcţie *hard limiter*, şi $\theta = 1/2$ vor fi selectaţi toţi neuronii pentru care $\cos(\mathbf{x}, \mathbf{W}_i) \leq 1/2$; toţi neuronii ai căror vectori sinaptici $\mathbf{W}_i$ fac un unghi cu vectorul de intrare **x** mai mic de 60° vor avea ieşirea în 1, toţi ceilalţi vor avea ieşirea în 0 (fig. 1.7).

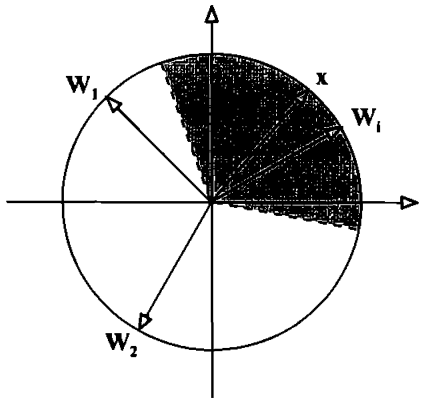



Fig. 1.7. Selecţia unui subspaţiu în funcţie de valoarea de prag

Când datele nu pot fi normalizate, reprezentarea funcţiei neuronului *McCulloch - Pitts* în coordonate polare devine mai dificilă deoarece, pe lângă direcţia vectorilor, va interveni şi norma acestora. Se poate considera funcţionarea neuronului în coordonate carteziene constatând astfel că el delimitează regiuni separate prin drepte. Să considerăm o RN elementară constituită dintr-un singur neuron cu 2 intrări (fig. 1.8).

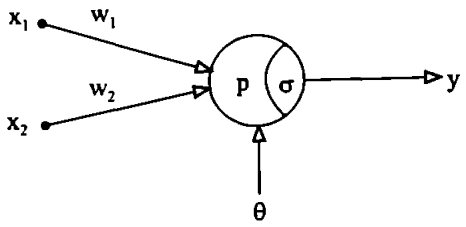



Fig. 1.8. RN elementară constituită dintr-un singur neuron

Dacă se aplică vectorul de intrare $\mathbf{x} = (x_1, x_2)$, potenţialul neuronului va fi:

$$p = w_1x_1 + w_2x_2$$

şi va fi comparat cu intrarea de prag $\theta$.

Dacă $w_1x_1 + w_2x_2 > \theta$ neuronul generează ieşirea $y = 1$, iar dacă
$w_1x_1 + w_2x_2 \leq \theta$ neuronul generează ieşirea $y = 0$.

Geometric, aceasta înseamnă că s-a divizat planul (spaţiul vectorilor stimuli) în 2 regiuni separate printr-o dreaptă de ecuaţie:

$$w_1x_1 + w_2x_2 - \theta = 0$$

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 + \frac{\theta}{w_2} \qquad (1.7)$$

Fig. 1.9 prezintă funcţia reţelei prezentate:

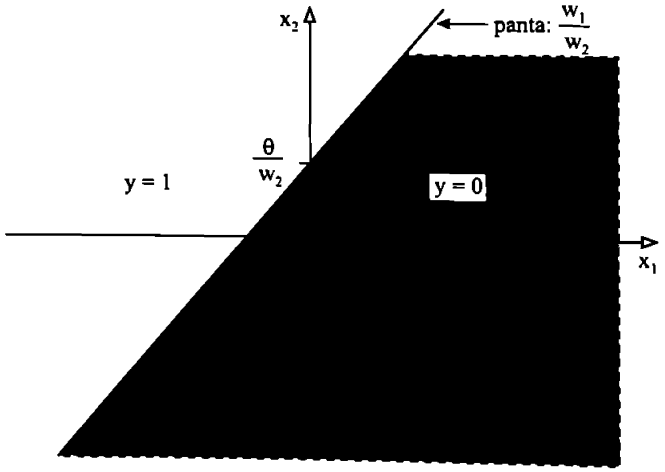



Fig. 1.9. Interpretarea geometrică a funcţiei neuronului din fig. 1.8.

Aceste interpretări geometrice sunt revelatoare în studiul reţelelor conexioniste.

### 1.3.3. Regula de învățare (modelul lui Hebb)

*Donald Hebb* este un neurofiziolog, autorul unei modelări al cărui succes se datorează în mod cert aparentei sale simplități. Studiul său [HEB 49] este axat pe ipoteza existenței unui ansamblu de celule, puternic interconectate, care se excită reciproc și care formează reprezentări ale informației în sistemul nervos. Este vorba de o reprezentare distribuită despre care *Hebb* afirmă că se regăsește atât la nivel funcțional (psihologic) cât și la nivel anatomic. Ideile sale au constituit o bogată sursă de inspirație pentru justificarea modelelor dezvoltate, printre alții, de către *Hopfield* [HOP 82], *Anderson* [AND 77] și *Grossberg* [GRO 76].

Din punct de vedere informatic, esența studiului lui *Hebb* se rezumă la un rezultat experimental calitativ, expus în cartea

*"The first stage of perception: growth and assembly":*

> *When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such A's efficiency, as one of the cells firing B, is increased.*

Această afirmație a fost intens exploatată pentru construcția modelelor conexioniste, căci ea constituie un enunț cantitativ care pune în evidență plasticitatea sinaptică. Practic, are loc o intensificare sinaptică până la nivelul unei activități comune (unite) a neuronului pre-sinaptic și post-sinaptic. Altfel spus, cauza și efectul intervin pentru modelarea mediului. Se poate astfel elabora o tabelă de adevăr pentru regula lui *Hebb*:

| Activitate pre - sinaptică | Activitate post - sinaptică | Regula lui *Hebb* |
|---|---|---|
| Activă | Activă | Intensificare + |
| Inactivă | Activă | Stare neutră |
| Activă | Inactivă | Stare neutră |
| Inactivă | Inactivă | Stare neutră |

Să remarcăm că regula lui *Hebb* nu prevede decât o intensificare sinaptică astfel încât ponderile sinaptice pot să crească și să conducă la saturația rețelei. Un amendament [RUM 86] la regula lui *Hebb* a permis ulterior centrarea semnalelor:

> *Adjust the strength of the connection between cell A and B*
> *in proportion of the product of their simultaneous activity.*

Această formulare este mai adecvată unui tratament informatic deoarece ea autorizează atât o variație pozitivă cât și una negativă a ponderilor sinaptice. Această regulă este rezumată în tabelul următor:

| Activitate pre - sinaptică | Activitate post - sinaptică | Regula de activitate sinaptică |
|---|---|---|
| Activă | Activă | Intensificare |
| Inactivă | Activă | Reducere |
| Activă | Inactivă | Reducere |
| Inactivă | Inactivă | Intensificare |

Trebuie să notăm că informația necesară calculului variației coeficientului sinaptic aferent unei anume sinapse este disponibilă local, în fiecare conexiune considerată. Variația coeficientului

sinaptic depinde doar de activitatea măsurată la cele 2 extremităţi ale sinapsei respective. În consecinţă, într-o reţea complexă, fiecare interconexiune îşi poate evalua propria variaţie şi realizează acest lucru în paralel cu celelalte interconexiuni.

Această regulă de ajustare a ponderilor sinaptice în funcţie de activitatea comună (unită) a neuronilor interconectaţi suferă totuşi de anumite limitări, care au condus la dezvoltarea unor reguli mai elaborate (de exemplu regula *Delta*). Trebuie totuşi să considerăm modelul lui *Hebb* ca stând la baza tuturor modelelor actuale, care nu fac decât o variaţie a structurii reţelei sau adaugă un "profesor" pentru procesul de învăţare [McG 87]. Din punct de vedere formal, descrierea modificării ponderilor sinaptice în funcţie de activitatea comună a neuronilor poate fi înţeleasă ca un calcul iterativ a unei măsuri de corelaţie între activitatea neuronului pre - sinaptic şi potenţialul neuronului post - sinaptic.

Reconsiderând reprezentarea neuronului din fig. 1.1, putem elabora prima regulă de ajustare a ponderilor sinaptice:

$$W_{ij}(t+1) = W_{ij}(t) + \alpha \cdot x_j(t) \cdot y_i(t) \qquad (1.8)$$

unde:

**W** - matricea ponderilor sinaptice
$x_j$ - semnalul pre - sinaptic (activitatea neuronului j)
$y_i$ - semnalul post - sinaptic (activitatea neuronului i)
t - timpul
$\alpha$ - constantă de învăţare, mică, cuprinsă în general între 0 şi 1.

Toate reţelele conexioniste actuale sunt construite pornind de la diversele variante ale neuronului elementar *McCulloch - Pitts* şi de la diversele variante ale regulii lui *Hebb* de ajustare a ponderilor sinaptice.

### 1.4 Modelul Adaline

#### 1.4.1. Introducere

Modelul *Adaline* (*ADAptive LINear Element*) reprezintă rezultatul cercetărilor lui *Bernard Windrow* şi *Marcian Hoff* [WID 60] în domeniul filtrelor adaptive. Un *Adaline* este compus dintr-un singur neuron şi dintr-un strat de interconexiuni care conţine N ponderi sinaptice (N intrări şi o ieşire) aşa cum indică fig. 1.10.

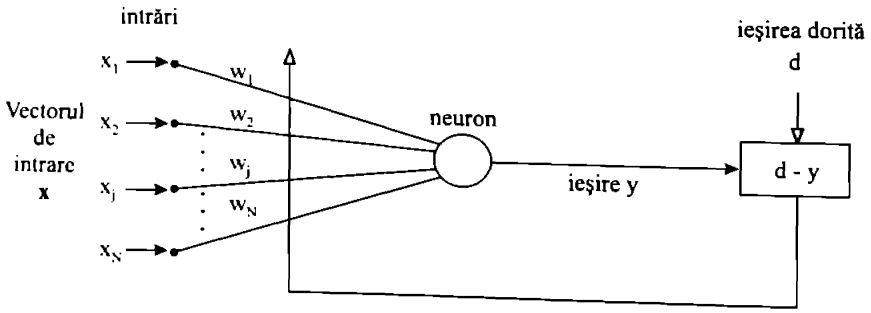



Fig. 1.10. Modelul *Adaline*

Funcţia neuronului este identică cu cea a neuronului *McCulloch - Pitts*; potenţialul se obţine printr-o însumare a intrărilor ponderate cu ponderile sinaptice aferente conexiunilor respective. Într-o primă interpretare vom considera că funcţia de transfer a neuronului este funcţia identică (ieşirea neuronului este egală cu potenţialul acestuia).

Noutatea, în cazul modelului *Adaline*, rezidă în noţiunea de învăţare sau de adaptare în sens ingineresc. Vom face presupunerea că sistemul are posibilitatea de a cunoaşte ieşirea ideală, d, asociată unui vector de intrare **x**. Devine posibilă definirea unui termen de eroare (ε) pornind de la diferenţa dintre ieşirea dorită (d) şi ieşirea calculată (y). Cunoscând această eroare, scopul devine de a adapta ponderile sinaptice astfel încât eroarea să fie redusă la un minim. Mai teoretic, ideea constă în modelarea unei mărimi (funcţii) printr-o combinaţie liniară de variabile şi de a găsi automat cei mai buni coeficienţi de ponderare pentru combinaţia liniară respectivă.

Această metodă de adaptare (învăţare) este denumită învăţare supervizată deoarece presupune existenţa unui "profesor" capabil să furnizeze ieşirea dorită (d) atunci când sistemul învaţă. Pentru a elabora un algoritm adaptiv avem la dispoziţie 3 mărimi asupra cărora se va concentra întreaga analiză:

**x** - vectorul de intrare (format din N componente reale)
**W** - vectorul ponderilor sinaptice (N componente reale)
d - ieşirea dorită furnizată de profesor (mărime scalară)

$$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_N \end{pmatrix} \qquad \mathbf{W} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{pmatrix}$$

Analiza care urmează este inspirată din lucrarea lui *B. Windrow* [WID 85] în care sunt descrise mai multe metode adaptive pentru procesarea de semnal.

### 1.4.2. Metode utilizate

Intuitiv, pentru ajustarea ponderilor sinaptice, trebuie căutată o măsură care să indice dacă ajustarea efectuată este justă sau nu. Ori, această măsură o avem deja la dispoziţie; este termenul de eroare (d-y). Dacă, la o anumită ajustare a ponderilor, eroarea creşte, atunci ajustarea nu este corectă. Dimpotrivă, dacă eroarea scade, putem considera că ajustarea ponderilor este corectă. Pentru automatizarea procesului de ajustare, trebuie determinată automat valoarea ce trebuie adăugată ponderilor pentru a produce scăderea erorii. Acest calcul poate fi realizat determinând gradientul erorii în raport cu ponderile sinaptice. Metoda care utilizează această procedură se numeşte metoda gradientului.

În cadrul metodei gradientului se determină:

- un termen de eroare pătratică;
- gradientul termenului de eroare în raport cu variaţia ponderilor sinaptice;
- regula de ajustare a ponderilor în funcţie de gradientul calculat.

Dacă termenul de eroare este un termen de eroare medie, avem de-a face cu metoda gradientului. Dacă termenul de eroare este eroarea instantanee, avem de-a face cu metoda gradientului stohastic. În cele ce urmează vom dezvolta ambele metode.

Ajustarea ponderilor se va face după următoarea regulă generală:

$$\Delta \mathbf{W}(t) = \alpha( -\text{grad}_{\mathbf{W}}\xi(t) ) \qquad (1.9)$$

unde:

$\Delta \mathbf{W}(t)$ reprezintă ajustarea vectorului **W** calculată la momentul t.
$\alpha$ este constanta de învăţare cu valoarea cuprinsă între 0 şi 1.
$\text{grad}_W \xi(t)$ este gradientul erorii $\xi$ la momentul t în raport cu ponderile sinaptice **W**.

### 1.4.3. Notaţii

Considerăm reţeaua cu N intrări şi o ieşire (fig. 1.10) care operează cu o funcţie de activare liniară. Funcţia modelului *Adaline* se rezumă la un simplu produs scalar între vectorul de intrare **x** şi vectorul ponderilor sinaptice **W** care conectează intrările la unitatea de ieşire.

Metoda coborârii pe direcţia gradientului negativ se aplică unui ansamblu de vectori de intrare aplicaţi la diferite momente de timp. Ieşirea calculată / ieşirea dorită / vectorul ponderilor sinaptice / vectorul prezentat la intrare vor fi notate cu $y(t)$ / $d(t)$ / $\mathbf{W}(t)$ / $\mathbf{x}(t)$. Ieşirea calculată va fi:

$$y(t) = \mathbf{W}^T(t) \cdot \mathbf{x}(t) \tag{1.10}$$

unde $\mathbf{W}^T$ este vectorul **W** transpus.

Pentru vectorul de intrare $\mathbf{x}(t)$ vom dispune de ieşirea dorită $d(t)$. Problema este de a ajusta ponderile **W** astfel încât, în medie, diferenţa (y - d) să fie nulă. Vom defini un termen de eroare, numit termen de eroare pătratică, cu ajutorul căruia metoda va permite calculul valorilor de ajustare a ponderilor (ajustare adaptivă).

Fixăm:

$$\varepsilon(t) = (d(t) - \mathbf{W}^T(t) \cdot \mathbf{x}(t)) \tag{1.11}$$

de unde rezultă:

$$\varepsilon^2(t) = (d(t) - \mathbf{W}^T(t) \cdot \mathbf{x}(t))^2 = d^2(t) - 2d(t)\left(\mathbf{x}^T(t) \cdot \mathbf{W}(t)\right) + \left(\mathbf{W}^T(t) \cdot \mathbf{x}(t)\right)\left(\mathbf{x}^T(t) \cdot \mathbf{W}(t)\right) \tag{1.12}$$

Relaţia (1.12) reprezintă eroarea pătratică instantanee asociată vectorului $\mathbf{x}(t)$. Dar, noi dorim să calculăm eroarea medie, adică eroarea după aplicarea întregului ansamblu de vectori $\mathbf{x}(t)$. Pentru aceasta vom face ipoteza că $\mathbf{x}(t)$, $d(t)$ şi $\varepsilon(t)$ sunt staţionare (variabile aleatoare independente şi echidistribuite). Putem utiliza media empirică sau experimentală a acestor variabile pentru a calcula eroarea medie pentru întregul ansamblu de vectori de intrare. Aceasta prezintă avantajul că permite previziunea minimului global precum şi a traseului pe suprafaţa de eroare. Definim deci:

$$\xi = E[\varepsilon^2(t)] \tag{1.13}$$

unde $\xi$ reprezintă eroarea medie pătratică.
În calculele ce urmează vom nota:

a) $$R = E[\mathbf{x}(t) \cdot \mathbf{x}^T(t)] \tag{1.14}$$

Matricea **R**, simetrică prin definiţie, se numeşte matrice de covarianţă a intrărilor (dacă acestea sunt centrate). Analiza detaliată a acesteia permite obţinerea de informaţii importante în legătură cu orientarea suprafeţei de eroare. Două elemente au o semnificaţie geometrică direct exploatabilă pentru a înţelege algoritmul: vectorii proprii ai matricii **R** definesc axele principale ale suprafeţei de

eroare şi valorile proprii ale lui **R** dau derivatele secunde ale suprafeţei de eroare după aceste axe principale.

b) $\mathbf{q} = E[d(t) \cdot \mathbf{x}(t)]$ (1.15)

Vectorul **q** reprezintă ansamblul de croscorelaţii între răspunsurile dorite şi vectorii de intrare.

Utilizând aceste notaţii, eroarea medie pătratică se va putea scrie:

$$\xi = E[\varepsilon^2(t)] = E[d^2(t)] - 2\mathbf{q}^T\mathbf{W}(t) + \mathbf{W}^T(t) \cdot R \cdot \mathbf{W}(t) \quad (1.16)$$

Fără a afecta generalitatea acestui calcul, vom utiliza ca măsură a erorii jumătatea mediei pătratice, pentru a obţine o expresie simplă pentru gradient la finele calculelor.

$$\xi = \frac{1}{2} \cdot E[\varepsilon^2(t)] \quad (1.17)$$

Pornind de la această expresie a erorii medii pătratice, putem reprezenta forma de variaţie a erorii (suprafaţa de eroare) într-un spaţiu cu N + 1 dimensiuni (primele N dimensiuni corespund celor n ponderi sinaptice $w_i$ preluate ca variabile iar pe a (N + 1) - a axă se reprezintă $\xi$). Un exemplu de suprafaţă de eroare în spaţiul tridimensional este redat în fig. 1.11. Trebuie să notăm că, în absenţa neliniarităţilor, această suprafaţă va prezenta un singur minim global care va reprezenta soluţia problemei noastre. Vom încerca să găsim o metodă iterativă care să conducă spre acest minim având ca scop final determinarea vectorului **W*** care reprezintă soluţia dorită.

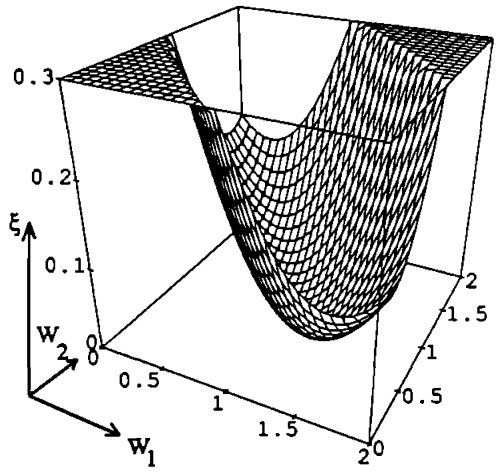



Fig. 1.11. Variaţia erorii pătratice (suprafaţa de eroare)

Această metodă a gradientului va permite ajustarea ponderilor sinaptice astfel încât să obţinem o evoluţie spre minimul global. Metoda constă în a evalua valoarea gradientului erorii într-un punct **W** dat pe suprafaţa de eroare. Acest gradient ne indică clar direcţia în care termenul de eroare creşte odată cu variaţia ponderilor sinaptice. Cum noi dorim reducerea termenului de eroare, este suficient să evoluăm în direcţie inversă.

### 1.4.4. Căutarea minimului prin metoda gradientului

Dacă se impune:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \cdot (-\text{grad}_{\mathbf{W}}\xi) = \mathbf{W}(t) - \alpha \cdot (\mathbf{R} \cdot \mathbf{W}(t) - \mathbf{q}) \qquad (1.18)$$

obţinem un algoritm iterativ în care $\mathbf{W}(t)$ converge spre $\mathbf{W}^*$, dacă $\alpha$ este mic. Valorile proprii ale matricii $(\mathbf{I} - \alpha \cdot \mathbf{R})$ trebuie să fie toate cuprinse între -1 şi +1.

Vom modifica aşadar ponderile sinaptice cu o fracţiune din gradientul calculat în punctul respectiv. Parametrul $\alpha$ controlează pasul cu care se coboară spre punctul de minim al suprafeţei de eroare.

Această metodă prezintă mai multe inconveniente. Pentru o singură ajustare a ponderilor, trebuie examinaţi toţi vectorii de intrare în cadrul unui ciclu de test; apoi se va minimiza suma erorilor pătratice. Nu este deci o metodă locală din punct de vedere temporal şi în afara unor cazuri particulare, va impune restricţii greu de respectat în cazul aplicării în timp real. Mai mult, a fost arătat că pentru a obţine o corecţie care reduce eroarea [GUI 89] trebuie respectată condiţia:

$$0 < \alpha < \alpha_{opt} \qquad \text{unde} \quad \alpha_{opt.} = \frac{\text{grad}_{\mathbf{W}}\xi}{\sum_{i} (\text{grad}_{\mathbf{W}}^{T}\xi \cdot \mathbf{x}(t))^2} \qquad (1.19)$$

Corecţia optimală revendică deci două faze succesive în cadrul fiecărui ciclu de baleiere a intrărilor:

1. acumularea vectorului gradient
2. calculul pătratelor produselor scalare între vectorul gradient şi vectorii de intrare.

În scopul eliminării acestor inconveniente, *B. Windrow* [WID 60] a propus o metodă de ajustare a ponderilor care nu ţine cont de media semnalelor ci de valoarea instantanee a acestora: metoda gradientului stohastic. După cum vom constata, această metodă conduce la acelaşi rezultat în ciuda unei convergenţe mai haotice.

### 1.4.5. Regula Delta

Principalul inconvenient al metodei anterioare rezidă din non-localizarea temporală şi spaţială a calculelor. Pentru a tinde spre soluţia optimală, trebuie cunoscută speranţa matematică a semnalelor de intrare, ceea ce ne obligă să considerăm o estimare a mediei pornind de la o medie pe termen scurt a semnalelor de intrare.

Pentru a reduce cantitatea calculelor şi pentru a propune un algoritm mult mai local, *B. Windrow* a dezvoltat o metodă de ajustare a ponderilor sinaptice în care traiectoria de coborâre spre punctul de minim se estimează la fiecare prezentare a unui vector de intrare. Convergenţa este mai complexă dar calculele nu au nevoie de media semnalelor de intrare.

Ideea revine la a considera că, pentru fiecare vector de intrare prezentat, vom genera o suprafaţă de eroare asemenea celei anterior prezentate. Pentru fiecare suprafaţă nouă, se estimează gradientul erorii în funcţie de ponderi şi se ajustează ponderile în sensul descreşterii erorii.

Vor exista atâtea suprafeţe de eroare câţi vectori se aplică la intrare şi, la fiecare aplicare, se va pointa spre un punct de minim instantaneu. Convergenţa spre minimul global teoretic este asigurată, dar traiectoria este mai haotică.

### 1.4.6. Metoda gradientului stohastic (regula Delta)

Ajustarea ponderilor, după fiecare prezentare a intrării **x**(t) se va calcula astfel:

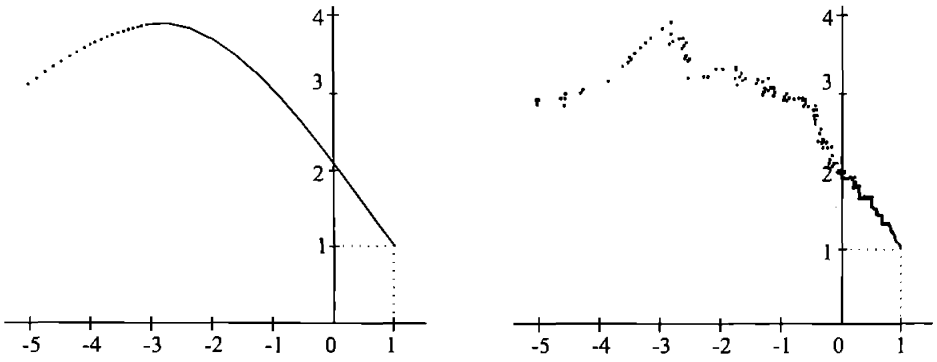
$$\mathbf{W}(t+1) = \mathbf{W}(t) - \alpha \cdot \text{grad}_{\mathbf{W}}\xi(t) \tag{1.20}$$

Regăsim astfel expresia de ajustare a ponderilor sinaptice identică cu cea prezentată în introducere:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \cdot (d(t) - y(t)) \cdot \mathbf{x}(t) \tag{1.21}$$

Şi aici parametrul $\alpha$ reprezintă constanta de învăţare al cărei rol este de a regla viteza şi stabilitatea procesului de învăţare. El poate fi menţinut constant; în anumite versiuni ale algoritmului, valoarea sa variază în funcţie de timp.

Utilizând regula *Delta*, procesul de ajustare a ponderilor spre minimul teoretic se face utilizând informaţii strict locale în timp. Derularea procesului de învăţare va comporta mai mult zgomot pe parcursul primelor iteraţii dar acest zgomot va fi compensat prin numărul de eşantioane extrase pe durata învăţării. Redăm un exemplu de convergenţă spre un minim global în figura 1.12. Punctul iniţial este $[w_1, w_2] = [-5, 3]$ şi punctul de minim care trebuie atins este $[w_1^*, w_2^*] = [1, 1]$.




a) metoda gradientului b) metoda gradientului stohastic

Fig. 1.12. Traiectoria ponderilor sinaptice în timpul învăţării

În fig. 1.12. a), traiectoria ponderilor a fost obţinută prin metoda gradientului. Această metodă revendică calculul mediei tuturor semnalelor de intrare. Observăm că traiectoria obţinută este regulată şi atinge direct punctul de minim. În figura 1.12. b) se prezintă traiectoria obţinută prin metoda gradientului stohastic. Punctul de minim este acelaşi dar traiectoria urmată este mai haotică. Ea corespunde de fapt diferitelor evaluări ale gradientului, asociate fiecărei curbe de eroare generată de fiecare vector aplicat la intrare.

### 1.4.7. Aplicaţie

Funcţia de clasificare este una dintre aplicaţiile directe ale modelului *Adaline*. Să presupunem că dorim împărţirea în 2 clase a unei mulţimi de puncte (vectori de intrare). Dacă cele 2 clase sunt liniar separabile, este posibilă utilizarea unei reţele *Adaline* pentru a construi automat clasificatorul.

În exemplul următor, care este un caz tipic, au fost generate aleator două mulţimi de puncte. Fiecare dintre acestea este centrată pe un punct de referinţă, celelalte puncte fiind generate cu o varianţă comună şi o medie distinctă.

Vom utiliza o reţea *Adaline* cu 3 intrări (coordonatele punctelor şi valoarea de prag) şi o ieşire. Ieşirea dorită este furnizată de un supervizor care indică, pentru fiecare vector de intrare căreia dintre cele 2 clase aparţine.

Printre cele 3 intrări avem una a cărei valoare este fixată permanent la 1, dar a cărei pondere sinaptică poate varia. Această intrare va juca rolul de prag pentru funcţia de activare. În interpretarea geometrică, valoarea de prag va servi la ajustarea ordonatei în origine aferentă dreptei de separare. Rezultatele simulărilor sunt redate în figura 1.13, în 3 momente succesive: la începutul simulării, după 20 şi respectiv după 100 vectori aplicaţi la intrare.

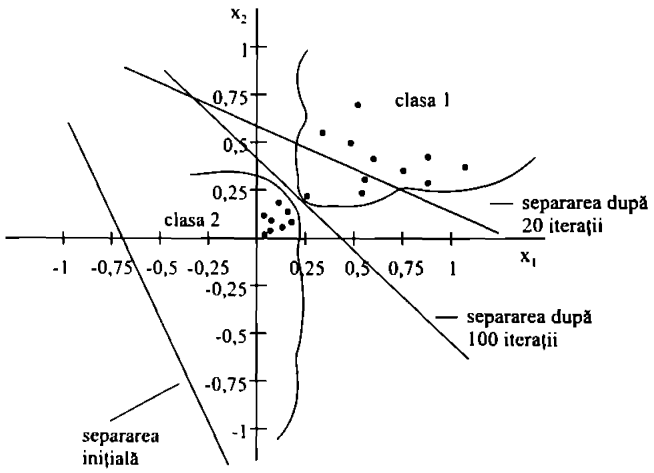



Fig. 1.13. Evoluţia dreptei de separare pe parcursul algoritmului

Valoarea erorii evoluează pe durata învăţării (fig. 1.14) şi descreşte asimptotic spre un punct de minim. Să notăm că pentru modelul *Adaline* eroarea este o valoare reală.

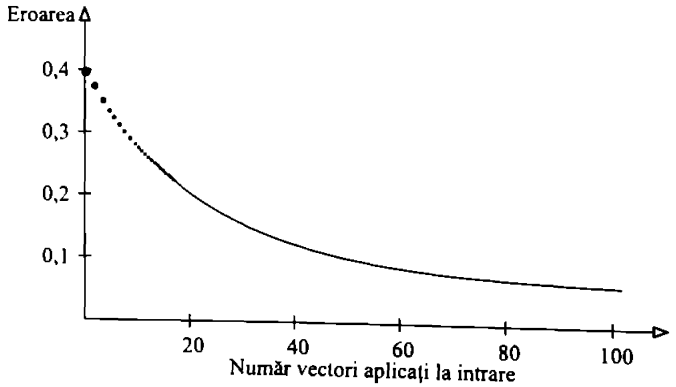



Fig. 1.14. Evoluţia erorii pe durata procesului de învăţare

În rezumat, metoda de ajustare a ponderilor sinaptice **W** decurge după următorul algoritm:

1. Inițializarea ponderilor **W** cu valori mici aleatoare
2. Aplicarea unui vector de intrare **x**(t) și a unei ieșiri dorite d(t)
3. Calculul ieșirii reale:

$$y(t) = \sum_{i=1}^{N} W_i(t) \cdot x_i(t)$$

4. Ajustarea ponderilor sinaptice după următoarea regulă:

$$\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \cdot (d(t) - y(t)) \cdot \mathbf{x}(t)$$

**W**(t) este matricea sinaptică la momentul t.
$\alpha$ este constanta de învățare, în general cuprinsă între 0 și 1.

### 1.4.8. Extensia modelului Adaline

Modelul pe care-l vom analiza poate fi utilizat pentru implementarea funcțiilor booleene. Intrările și ieșirea nu vor mai fi valori reale ci valori binare, ceea ce impune utilizarea unei funcții de transfer care transformă valorile continue în valori discrete. O primă soluție propusă constă în utilizarea unei funcții de transfer de tip "*hard limiter*", asociată cu o funcție de prag.

În articolul din 1960 [WID 60], *Windrow* utiliza un *Adaline* de acest tip pentru clasificarea modelelor; modelele exersate erau literele T, G și F. Valorile asociate la ieșire au fost: -60 pentru T, 0 pentru G și +60 pentru F. Să notăm că, în cazul *Adaline*, evaluarea erorii se face luând în considerare potențialul neuronului și nu ieșirea sa (ieșirea se obține prin aplicarea funcției de transfer asupra valorii potențialului).

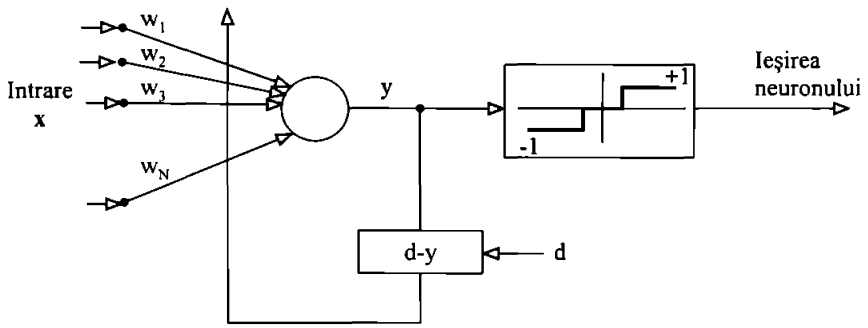



Fig. 1.15. Extensia modelului *Adaline*

### 1.4.9. Rețeaua Madaline (Many Adalines)

O rețea monostrat înzestrată cu o regulă de învățare de tip *Adaline* nu poate realiza decât o separare liniară. În problemele reale această funcție nu este suficientă. Este însă posibilă extinderea posibilităților rețelei prin adăugarea unui nou strat care va realiza o combinație de separatori liniari.

Astfel se vor putea genera operatori capabili să separe două regiuni dintre care cel puţin una este convexă sau închisă. Termenul convexă înseamnă că toate segmentele de dreaptă care unesc două puncte arbitrare situate pe marginea regiunii sunt integral conţinute în regiunea respectivă. Această separaţie poate fi realizată cu o reţea *Madaline* a cărei structură este prezentată în fig. 1.16.

Primul strat de interconexiuni va separa spaţiul de intrare prin hiperplane. Fiecare neuron din stratul intern va genera un hiperplan care separă 2 semispaţii; ieşirea lui va fi activă pentru punctele (vectorii de intrare) dintr-un semispaţiu şi inactivă pentru punctele din celălalt semispaţiu.

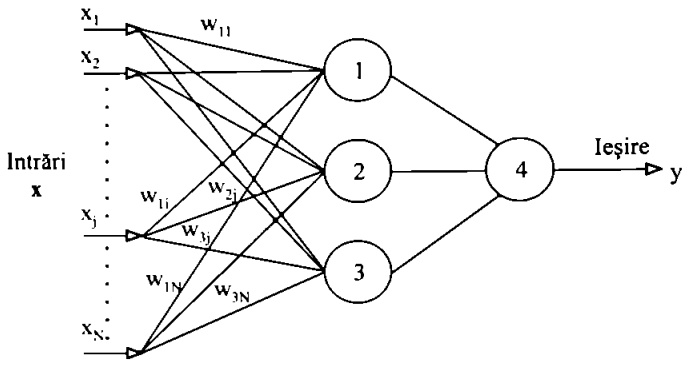
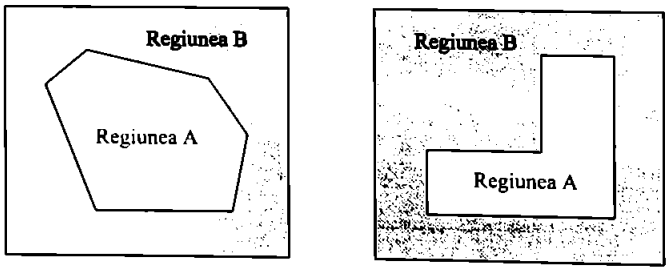



Fig. 1.16. Reţeaua *Madaline*

Al doilea strat de interconexiuni nu realizează decât o conexiune directă (neponderată). Putem considera că este un strat în care toate ponderile sinaptice sunt egale cu 1. Dacă se fixează pragul neuronului de ieşire la valoarea (P - ε), unde P reprezintă numărul de neuroni din stratul intern şi ε eroarea, atunci funcţia neuronului de ieşire va fi un ŞI logic între toate ieşirile stratului intern.

Regiunea de decizie a neuronului de ieşire va fi astfel o intersecţie de semispaţii generate în primul strat. Se va forma deci o regiune convexă care va avea exact atâtea laturi câţi neuroni există în stratul intern. În figura 1.17 se prezintă un exemplu de regiuni separabile şi un exemplu de regiuni neseparabile printr-o RN de acest tip.




a) Regiuni separabile

b) Regiuni neseparabile

Fig. 1.17. Regiuni de decizie

RN cu 2 straturi de interconexiuni, cu toate posibilităţile lor interesante, sunt neutilizabile în majoritatea problemelor reale. Trebuie deci să se asigure reţelei un grad de libertate suplimentar pentru a putea separa şi regiuni non-convexe.

### 1.4.10. Concluzii

Am studiat în acest paragraf fundamentele RN cu un singur strat de interconexiuni şi am enunţat un algoritm de învăţare care conduce la o reţea capabilă să facă doar separaţii liniare.

Descrierea intuitivă a funcţionării reţelelor cu 2 straturi de interconexiuni a arătat că este posibilă combinarea separatorilor liniari în scopul construirii unor regiuni convexe. Totuşi, a rămas deschisă problema unui algoritm care să permită, în cazul reţelelor multistrat, calcularea automată a ponderilor sinaptice (învăţarea). Această problemă a rămas fără răspuns până în anul 1985, dată la care mai mulţi autori [LeC 85], [RUM 86] au propus o soluţie bazată pe generalizarea algoritmului *Windrow - Hoff*. Vom studia acest algoritm în secţiunea consacrată Perceptronilor multistrat.

### 1.5. Modelul Perceptronului monostrat

Modelul perceptronului se bazează pe neuronul *McCulloch - Pitts* şi descrie un algoritm capabil să ajusteze ponderile sinaptice în funcţie de valorile pe care dorim să le înveţe. Termenul perceptron acoperă actualmente o clasă mult mai largă de modele decât perceptronul original introdus de *Rosenblatt* [ROS 58]. În figura 1.18 este rezumată structura perceptronului lui *Rosenblatt*.

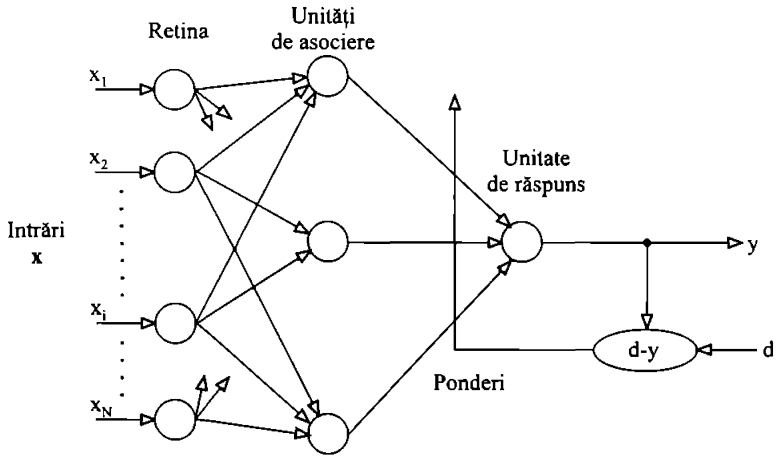



Fig. 1.18. Reprezentarea originală a perceptronului

Perceptronul este compus dintr-un prim strat de conexiuni (numite "*demons*") situat între unităţile de intrare (retina) şi unităţile de asociere. Acest prim strat nu face decât o pre - procesare fixă.

Al doilea strat va fi constituit dintr-un singur neuron, conectat la unităţile de asociere prin ponderi sinaptice variabile. Formal, acest perceptron este format dintr-un singur strat de conexiuni plastice (ajustabile). După cum am concluzionat deja în cazul modelului *McCulloch - Pitts*, această structură nu poate funcţiona decât ca separator liniar. Problema care se pune rezidă în elaborarea unui algoritm capabil să calculeze ponderile astfel încât să se realizeze o separare convenabilă a claselor.

Termenul perceptron denotă astăzi o structură mult mai simplă decât cea prezentată mai sus. El este mult mai apropiat de modelul *Adaline*, atât prin structură cât şi prin regula de învăţare.

Diferenţele apar în 2 puncte:

1. măsura erorii
2. valorile de ieşire

Aceste diferenţe au totuşi o influenţă considerabilă asupra algoritmului de învăţare. În continuare vom studia perceptronul elementar a cărui structură este prezentată în fig. 1.19. Constatăm că stratul "*demons*" a dispărut; n-a rămas decât stratul adaptiv asupra căruia se va aplica regula de învăţare.

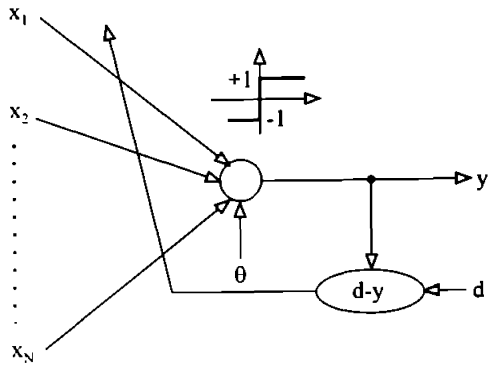



Fig. 1.19. Separator liniar în 2 clase

### 1.5.1. Principiul algoritmului

Algoritmul [ROS 58] trebuie să determine dreapta care va separa cele 2 clase. Construcţia dreptei se va realiza minimizând un termen de eroare prin metoda gradientului stohastic. Pentru aceasta, vom calcula mai întâi potenţialul neuronului pornind de la produsul scalar dintre vectorul de intrare **x** şi vectorul de ponderi sinaptice **W**, vom însuma valoarea de prag şi apoi vom aplica funcţia de transfer neliniară.

$$p = \mathbf{W}^T \cdot \mathbf{x} \tag{1.22}$$

$$y = \sigma (p + \theta) \tag{1.23}$$

Dacă vectorii de ponderi sinaptice şi respectiv intrare sunt normalizaţi, potenţialul neuronului (p) va conţine informaţii referitoare la gradul de similaritate dintre cei doi vectori (într-o interpretare geometrică unghiul dintre cei 2 vectori). În sfârşit, funcţia de transfer ($\sigma$) va indica dacă acest potenţial depăşeşte sau nu valoarea de prag. În funcţie de aceasta neuronul va răspunde cu +1 sau -1.

### 1.5.2. Algoritmul de învăţare

Algoritmul de învăţare vizează dotarea perceptronului cu capacitatea de clasificare. Această capacitate poate fi însuşită prin compararea răspunsului dorit d (care va fi +1 sau -1), furnizat sistemului de către supervizor (profesor), şi ieşirea reală y (care va fi tot +1 sau -1). Este motivul pentru care această tehnică de ajustare a ponderilor sinaptice se numeşte învăţare supervizată.

Perceptronul trebuie să separe liniar 2 clase pe care le vom nota A şi B. Pentru fiecare clasă vom asocia o ieşire dorită, d = 1 pentru clasa A şi d = -1 pentru clasa B. Valoarea reală de la ieşirea y va fi tot binară (-1 sau +1). Pentru a şti dacă vectorului de intrare **x** i se atribuie clasa corectă, este suficient să comparăm ieşirea dorită d cu ieşirea reală y:

dacă (d - y) = 0,

vectorul **x** este bine clasat. În acest caz nu se întreprinde nimic.

dacă (d - y) = 2 sau (d - y) = -2,

vectorul **x** nu este clasat corect. În acest caz trebuie modificate ponderile sinaptice în scopul minimizării erorii. Soluţia constă în a face o corecţie după următorul algoritm [BLO 62]:

1. Se iniţializează ponderile **W** cu valori aleatoare mici.
2. Se aplică o intrare **x**(t) şi ieşirea dorită asociată d(t).
3. Se calculează ieşirea reală:

   $y(t) = \sigma(\mathbf{W}(t) \cdot \mathbf{x}(t))$

   unde $\sigma$ este funcţia "*hard limiter*"

4. Se ajustează ponderile sinaptice după următoarea regulă:

   $\mathbf{W}(t+1) = \mathbf{W}(t) + \alpha \cdot (d(t) - y(t)) \cdot \mathbf{x}(t)$

   **W**(t) este matricea (vectorul) de conexiuni la momentul t
   $\alpha$ este constanta de învăţare cuprinsă între 0 şi 1.

Principalul inconvenient al acestei reguli de învăţare provine din ignorarea mărimii erorii atunci când se ajustează ponderile sinaptice (spre deosebire de cazul *Adaline*). Indiferent dacă eroarea este mică sau mare, ponderile sunt ajustate cu o valoare constantă. În general, procedura de ajustare de tip perceptron se bazează pe minimizarea unei distanţe în spaţiul ponderilor şi nu pe minimizarea unui criteriu de eroare pătratică (cum am constatat în cazul *Adaline* şi în alte proceduri mai elaborate). În general, pentru RN multistrat (capabile să realizeze funcţii complexe), se aplică criteriul minimizării erorii pătratice.

Problema care se pune acum este de a afla dacă această procedură de ajustare a ponderilor este capabilă să atingă o soluţie; cu alte cuvinte, dacă algoritmul este convergent sau nu.

### 1.5.3. Exemplu de funcţionare

Considerăm 2 mulţimi de puncte, reprezentate în planul cartezian în fig. 1.20.

Se va atribui un cod de clasă fiecăruia dintre puncte. Coordonatele punctelor reprezintă vectorii de intrare iar codul de clasă atribuit reprezintă ieşirea dorită.

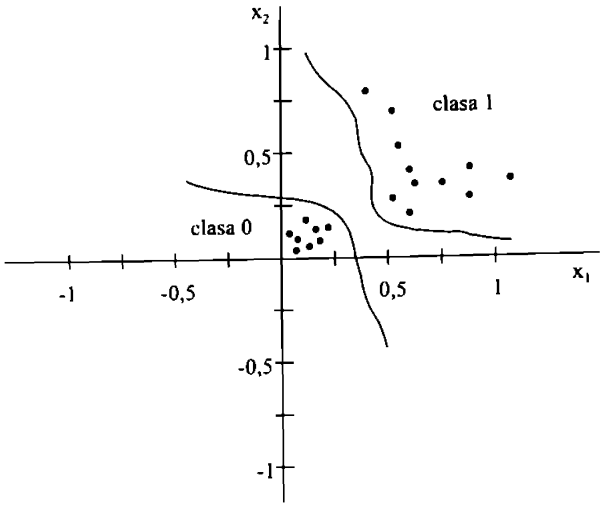



Fig. 1.20. Repartiţia punctelor ce urmează a fi clasificate

Pentru a realiza clasificarea se poate utiliza un perceptron cu 2 intrări, o valoare de prag variabilă şi o ieşire (fig. 1.21). Valoarea de prag poate fi asimilată cu o intrare de valoare fixă, +1, (polarizare) şi cu o pondere sinaptică variabilă (θ). Prin acest artificiu, calculul valorii de prag revine la calculul unei ponderi suplimentare.

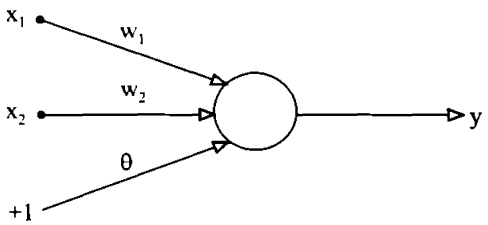



Fig. 1.21. Perceptronul utilizat pentru separarea celor 2 mulţimi de puncte

Se începe prin fixarea ponderilor sinaptice la valori mici aleatoare. Se aplică apoi perechile (vector intrare, ieşirea dorită). Se efectuează produsul scalar între vectorul de intrare aplicat ($x_1$, $x_2$, 1) şi vectorul ponderilor sinaptice ($w_1$, $w_2$, θ). Se obţine un rezultat care este transformat de funcţia de activare a neuronului (în cazul perceptronului funcţia *sign*). Reţeaua răspunde cu +1 sau -1.

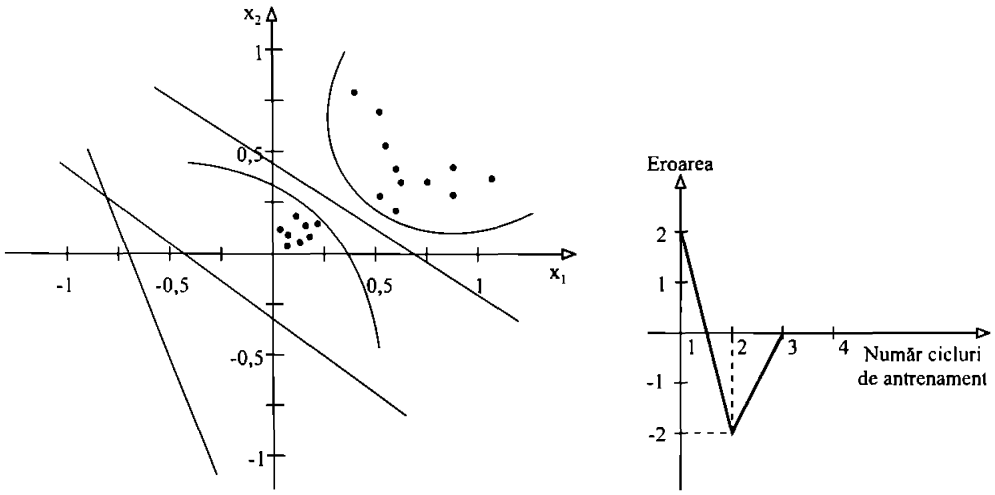
Această ieşire se compară cu ieşirea dorită şi se măsoară (calculează) diferenţa dintre ieşirea dorită şi ieşirea reală. Dacă diferenţa este nulă, ponderile sinaptice nu vor fi ajustate. Dacă diferenţa este nenulă, ponderile se ajustează în funcţie de intrare (**x**), constanta de învăţare (α) şi semnul diferenţei respective.

Această operaţie se repetă până când nu se va mai înregistra nici o diferenţă între ieşirea dorită şi cea reală (pentru toate perechile de intrare). Trebuie să subliniem că, dacă separarea liniară a celor 2 clase este posibilă, teorema convergenţei ne asigură că numărul de corecţii aplicate ponderilor sinaptice este finit.

Dreapta care va separa cele 2 clase, în momentul atingerii soluției finale, va avea ecuația:

$$x_2 = -\frac{w_1}{w_2} \cdot x_1 - \frac{\theta}{w_2} \qquad (1.24)$$

În momentul t = 0 dreapta de separare se va afla într-o poziție inițială aleatoare (fig. 1.22). Dreapta evoluează apoi, ca pantă și ca ordonată în origine, pentru a veni și a se plasa între cele 2 mulțimi de puncte. Din momentul plasării între cele 2 mulțimi de puncte termenul de eroare rămâne pe 0 și convergența este atinsă. Figura 1.22 indică evoluția dreptei separatoare pe parcursul aplicării perechilor de antrenament (**x**, d) precum și evoluția termenului de eroare în funcție de numărul de cicluri de antrenament efectuate (într-un ciclu de învățare se aplică succesiv toate perechile de antrenament).




a) dreapta separatoare b) eroarea

Fig. 1.22. Evoluția procesului de învățare

Cum se poate constata din acest exemplu, din momentul în care termenul de eroare se anulează, el rămâne definitiv nul și nu va mai avea loc nici o corecție. Deoarece convergența algoritmului a fost demonstrată atunci când există o soluție [MIN 88], aceasta este atinsă și ponderile sinaptice rămân într-o configurație stabilă.

### 1.5.4. Concluzii

Algoritmul perceptronului este un algoritm de învățare de bază care se limitează la probleme care admit soluții pe bază de clasificatori liniari. Aceasta reprezintă o limitare severă care nu permite decât tratarea cazurilor simple. Cu toate acestea, metoda a fost larg utilizată în aplicațiile de recunoaștere a formelor [DUD 73]. În majoritatea aplicațiilor trebuie mers dincolo de modelul perceptronului. Vom vedea în cele ce urmează că există algoritmi de ajustare a ponderilor (învățare) care permit separarea spațiului vectorilor de intrare în regiuni cu contururi oarecare și care depășesc deci limitele acestui model simplu.

### 1.6. Modelul retro-propagării gradientului

#### 1.6.1. Introducere

Strategiile de învăţare aplicate reţelelor monostrat adaptive oferă o funcţionalitate redusă la nivelul unui simplu clasificator liniar. Capacitatea de procesare a informaţiei este foarte limitată şi nu permite decât rezolvarea problemelor foarte simple. Pentru depăşirea acestor limite trebuie utilizate RN multistrat care vor fi apte să soluţioneze probleme mult mai complexe. Din nefericire, algoritmul perceptronului nu permite ajustarea ponderilor unui strat care nu-şi cunoaşte obiectivul pe care trebuie să-l atingă şi aceasta este problema care apare în momentul conectării în cascadă a 2 sau mai multe straturi.

Această problemă a limitat dezvoltarea reţelelor de perceptroni multistrat şi o soluţie a fost propusă doar relativ recent de *Yann Le Cun* [LeC 85], *Rumelhart*, *Hinton* şi *Williams* [RUM 86] şi *Paul Werbos* [WER 88], pentru a nu cita decât autorii cei mai cunoscuţi. Vom analiza mai întâi carenţele perceptronilor monostrat, pentru a arăta mai apoi care sunt posibilităţile oferite de perceptronii multistrat.

#### 1.6.2. Problema SAU - Exclusiv

Pentru a sesiza limitele intrinseci ale modelului perceptron, vom studia următoarea problemă: implementarea funcţiei SAU - Exclusiv utilizând un singur perceptron. În figura 1.23 s-a reprezentat funcţia SAU - Exclusiv în spaţiul vectorilor de intrare.

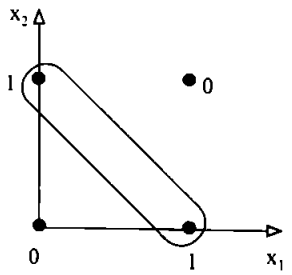



Fig. 1.23. Reprezentarea problemei SAU - Exclusiv

Fiecărui punct din fig. 1.23 îi este asociată ieşirea, care denotă clasa la care el trebuie să aparţină. După cum se observă, punctele din aceiaşi clasă (0 sau 1) nu pot fi separate cu o dreaptă. Această problemă relativ simplă nu poate fi rezolvată cu ajutorul unui singur perceptron. Dacă utilizăm o reţea bazată pe unităţi binare cu prag, cu o unitate ascunsă, de forma celei din fig. 1.24, problema poate fi rezolvată [McC 43].

Acest comportament se explică simplu, constatând că adăugarea unei unităţi ascunse permite rescrierea ecuaţiei funcţiei SAU - Exclusiv adăugând un termen redundant:

$$y = x_1 \oplus x_2 = (x_1 + x_2) \cdot \overline{(x_1 \cdot x_2)}$$

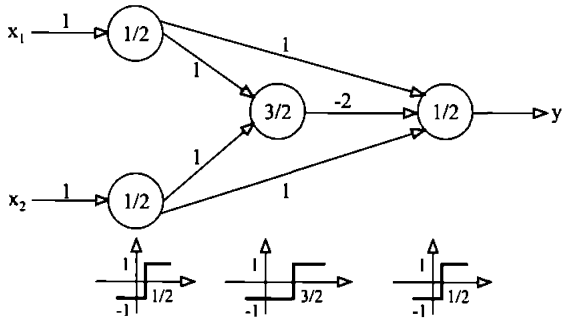



Fig. 1.24. Reţea de unităţi binare pentru rezolvarea problemei SAU - Exclusiv

Reprezentarea geometrică se va face acum utilizând o dimensiune suplimentară; pe axa z se reprezintă termenul $x_1 \cdot x_2$ (fig. 1.25).

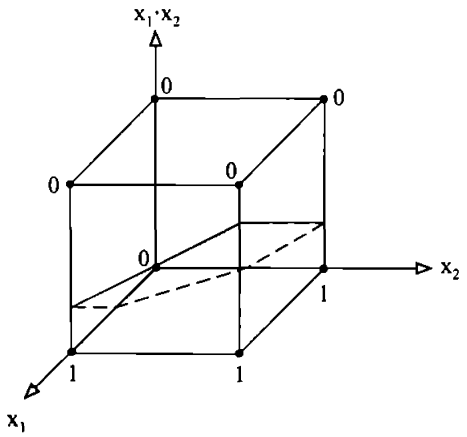



Fig. 1.25. Planul de separare a celor 2 clase

Am găsit astfel o soluţie pentru problema SAU - Exclusiv, bazată pe o reţea de celule liniare cu valori de prag asimetrice şi care generează un plan de separare capabil să claseze convenabil vectorii de intrare. Totuşi, ceea ce am făcut manual relevă o anume complexitate care dă naştere la o întrebare: cum vom alege numărul de unităţi ascunse (pentru o problemă dată) şi cum îşi va ajusta reţeaua ponderile şi valorile de prag pentru a rezolva problema dată?

Pentru RN multistrat, *Y. Le Chun* [LeC 85] a propus minimizarea unui termen de eroare pătratică printr-o metodă similară celei propuse de *Windrow* la modelul *Adaline*. Metoda presupune bineînţeles existenţa unui supervizor capabil să genereze ieşirea dorită pentru fiecare vector aplicat la intrare. Inovaţia soluţiei lui *Le Chun* rezidă în posibilitatea de a comunica recursiv neuronilor din straturile ascunse termenul de eroare care le revine (contribuţia fiecăruia la eroarea globală de la ieşire). Cunoscându-şi starea precum şi obiectivul care trebuie atins (minimizarea propriului termen de eroare), neuronii din straturile ascunse îşi vor putea ajusta ponderile sinaptice în acord cu acest obiectiv.

Totuşi, metoda propusă de *Le Chun* nu oferă răspuns la întrebarea: care este arhitectura optimă a RN chemată să rezolve o anumită problemă? Metoda permite doar calculul ponderilor pentru o reţea cu straturi ascunse (neuroni care nu aparţin stratului de ieşire).

Singurele rezultate exploatabile, referitoare la optimizarea arhitecturii RN, se referă la capacitatea de clasificare în funcţie de numărul de straturi. Lucrări relativ recente au arătat că o reţea cu 2 straturi permite realizarea oricărei funcţii booleene în raport cu variabilele de pe intrare [HAM 87] şi că o reţea cu 3 straturi este suficientă pentru rezolvarea tuturor problemelor de clasificare [LIP 87]. Noi presupunem că există de asemenea şi un număr optim de neuroni pe fiecare strat dar acesta nu putem încă să-l determinăm cu precizie. După cum vom vedea, acest parametru are o importanţă crucială asupra vitezei de convergenţă a algoritmului de învăţare şi chiar, pur şi simplu, asupra posibilităţii de a găsi o soluţie la o problemă.

În general, o RN care conţine în structura sa neuroni ascunşi va putea genera o suprafaţă de decizie complexă şi, prin urmare, va putea rezolva probleme de clasificare complexe. Suplimentar, utilizând o funcţie de transfer neliniară la nivelul neuronilor, capacitatea RN creşte mult putând atinge o capacitate vecină metodelor de interpolare neliniară.

### 1.6.3. Arhitectura RN multistrat

Într-o reţea multistrat, toate celulele stratului de intrare sunt urmate de o succesiune de straturi, referite cu termenul straturi ascunse, pentru ca în final să se ajungă la stratul de ieşire. Când reţeaua este simplu etajată, fiecare neuron ce aparţine unui strat nu va primi pe intrare decât ieşirile neuronilor din stratul anterior. În sens larg, într-o RN multistrat, nimic nu ne împiedică să luăm în considerare şi interacţiuni între straturi îndepărtate. În general se face uz doar de interconexiuni locale între straturi adiacente aşa cum arată figura 1.26.

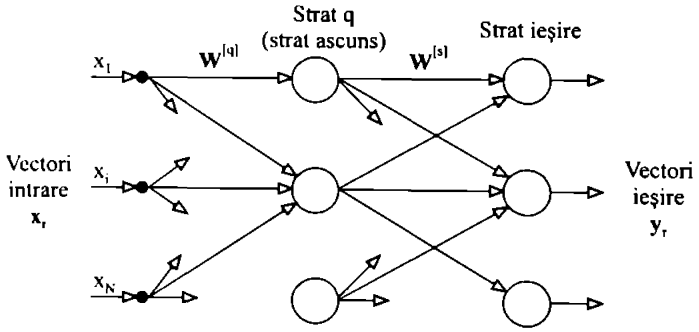



Fig. 1.26. Arhitectura generală a reţelei multistrat

Prezenţa stratului intermediar (straturilor intermediare) aduce un plus de bogăţie parametrică care trebuie să conducă la creşterea capabilităţilor reţelei. Trebuie să menţionăm faptul că neuronii interni nu au nici o conexiune predefinită; ei vor contribui doar la obţinerea rezultatelor dorite la ieşire.

O altă diferenţă, esenţială, se referă la funcţiile de transfer aferente neuronilor. Vor fi funcţii neliniare, numite şi "funcţii de compresie" pentru că ele comprimă potenţialul neuronului în intervalul [-1, +1]. Funcţiile de transfer utilizate trebuie să fie continue şi derivabile (vom vedea de ce). Prin urmare, neuronii emit valori continue şi nu discrete pe ieşire.

Pentru un neuron i din stratul [q], considerând că $x_j^{[q]}$ sunt intrările în acest strat, potenţialul $p_i^{[q]}$ va fi:

$$p_i^{[q]} = \sum_j W_{ij} \cdot x_j^{[q]} \tag{1.25}$$

iar ieşirea $y_i^{[q]}$ se va obţine pornind de la acest potenţial:

$$y_i^{[q]} = \sigma(p_i^{[q]}) \tag{1.26}$$

Funcţia de transfer $\sigma$ trebuie să fie continuă şi derivabilă; se utilizează de regulă funcţia sigmoidală unipolară sau bipolară (relaţiile (1.3) şi (1.4)) sau alte funcţii continue şi derivabile cum ar fi:

$$\sigma(p) = th(p) = \frac{e^p - e^{-p}}{e^p + e^{-p}} \tag{1.27}$$

Arhitectura pe care o descriem comportă mai mulţi parametri precum ar fi: forma funcţiei de transfer, numărul de straturi, numărul de neuroni per strat. Nu există încă o metodă analitică pentru determinarea acestor parametri structurali. Este nevoie de experienţă în domeniu pentru a ajunge la cel mai bun model pentru o anume aplicaţie.

### 1.6.4. Algoritmul retropropagării erorii (Error Back - Propagation)

Reţeaua multistrat (*"feedforward"*) are o structură formată din starturi succesive, traversate de semnalele de intrare care se propagă până la ieşire. Dacă dorim să transpunem algoritmul de învăţare dezvoltat pentru reţelele monostrat, va trebui să definim un termen de eroare. Învăţarea (ajustarea ponderilor sinaptice) va urmări minimizarea acestui termen de eroare. Termenul de eroare poate fi definit dacă există un set de vectori de intrare pentru care se cunosc valorile dorite la ieşirea RN. Învăţarea va fi deci supervizată şi se bazează pe un set de perechi de învăţare sau de antrenament ($\mathbf{x}_r$, $\mathbf{d}_r$). Termenul de eroare va fi definit ca în fig. 1.27.

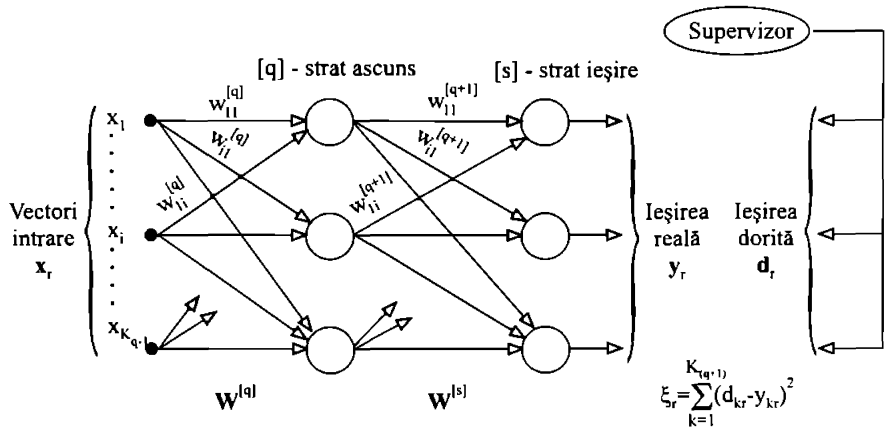



Fig. 1.27. Arhitectura reţelei *feedforward* şi definirea termenului de eroare

Se utilizează strategia de minimizare a erorii $\xi_r$ bazată pe gradientul stohastic. Dar, aşa cum se observă din fig. 1.27, numai stratul de interconexiuni asociat stratului de ieşire cunoaşte direct

valoarea de minimizat. Straturile interne, asociate neuronilor ascunşi, nu dispun de un termen de eroare pentru operaţia de minimizare. Rezultă că, pentru straturile ascunse, nu se vor putea calcula direct valorile de ajustare a ponderilor sinaptice.

Principiul algoritmului *Error Back - Propagation* (EBP) constă în ideea de a impune o funcţie obiectiv neuronilor din straturile ascunse. Pe baza acestei funcţii obiectiv se vor putea calcula şi valorile de ajustare a ponderilor sinaptice din straturile ascunse. Funcţia obiectiv trebuie să rezulte din eroarea $\xi_r$ măsurată la ieşirea reţelei. Un neuron situat într-un strat intern nu poate să vadă eroarea calculată la ieşire decât traversând toate ponderile sinaptice care-l leagă de stratul de ieşire. Aşadar, algoritmul ar putea modifica ponderile aferente conexiunilor care intră în neuronul respectiv cu scopul de a reduce participarea acestuia la eroarea calculată la ieşire. Aceasta înseamnă că eroarea calculată la ieşirea reţelei este propagată în sens invers cu aceleaşi ponderi sinaptice pe conexiuni (identice cu cele utilizate în faza *feedforward* în care se generează ieşirea $\mathbf{y}_r$). În acest mod, pe ieşirea neuronului din stratul ascuns, ajunge un termen de eroare care reprezintă contribuţia sa la eroarea globală calculată la ieşirea reţelei; pe baza acestui termen de eroare vor putea fi ajustate ponderile sinaptice aferente conexiunilor de intrare în neuronul respectiv.
În consecinţă:

- ieşirile neuronilor din faza *feedforward* devin puncte de însumare a erorilor retro - propagate;
- punctele de însumare din faza *feedforward* (intrările) devin ieşiri pentru erori;
- funcţiile de transfer ale neuronilor devin propriile lor derivate.

Procedura de ajustare a ponderilor sinaptice prevăzută în cadrul algoritmului EBP va fi:

Date iniţiale: Setul de antrenament format din R perechi
$(\mathbf{x}_r, \mathbf{d}_r)$; r = 1, 2, ..., R
$\mathbf{x}_r$ - vectorul aplicat la intrare
$\mathbf{d}_r$ - vectorul dorit la ieşire (aferent vectorului de intrare $\mathbf{x}_r$)

1. Se iniţializează toate ponderile sinaptice (matricile $\mathbf{W}^{[q]}$, unde q = 1, 2, ..., s) cu valori mici aleatoare.
2. Se aplică o pereche de antrenament $(\mathbf{x}_r, \mathbf{d}_r)$;
   r = 1, 2, ..., R
   *NOTA:* Pentru o convergenţă cât mai bună, perechile de antrenament $(\mathbf{x}_r, \mathbf{d}_r)$ se selectează în ordine aleatoare din setul de antrenament (minimizarea erorii se face pe baza gradientului stohastic!)
3. Se calculează ieşirea reală prin propagarea directă (*feedforward*) a vectorului de intrare prin toate straturile RN:

$$y_{jr}^{[q]} = \sigma\left(\sum_{i=1}^{K_q} w_{ji}^{[q]} \cdot y_{ir}^{[q-1]}\right) = \sigma\left(\sum_{i=1}^{K_q} w_{ji}^{[q]} \cdot x_{ir}^{[q]}\right) \qquad (1.28)$$

   unde $\sigma$ este funcţia de transfer şi q = 1, 2, ..., s.
   Pentru stratul de intrare q = 1, $\mathbf{y}^{[q-1]} = \mathbf{x}$.

4. Se calculează eroarea la ieşire:

$$\xi_r = \frac{1}{2}\sum_{k=1}^{K_s} (d_{kr} - y_{kr}^{[s]})^2 \qquad (1.29)$$

unde: $\mathbf{d}_r$ - ieşirea dorită asociată vectorului de intrare $\mathbf{x}_r$
$\mathbf{y}_r^{[s]}$ - ieşirea reală din RN obţinută în urma prezentării vectorului de intrare $\mathbf{x}_r$.
$\xi_r$ - eroarea la ieşire calculată pentru cuplul $(\mathbf{x}_r, \mathbf{d}_r)$

5. Retro - propagarea semnalului de eroare ($\delta_{ir}$) dinspre stratul de ieşire spre stratul de intrare.
   a) pentru neuronii din stratul de ieşire:

$$\delta_{ir}^{[s]} = \sigma'(p_{ir}^{[s]}) \cdot (d_{ir} - y_{ir}^{[s]}) \tag{1.30}$$

   b) pentru neuronii din straturile ascunse:

$$\delta_{ir}^{[q]} = \sum_{k=1}^{K_q} \delta_{kr}^{[q+1]} \cdot w_{ki}^{[q+1]} \cdot \sigma'(p_{ir}^{[q]}) \tag{1.31}$$

   unde $\sigma'(p)$ reprezintă derivata funcţiei de activare.

6. Ajustarea ponderilor sinaptice după regula:

$$\Delta w_{ij}^{[q]} = \alpha \cdot \delta_{ir}^{[q]} \cdot x_{jr}^{[q]} \tag{1.32}$$

   unde $\alpha$ este constanta de învăţare cuprinsă între 0 şi 1.

7. Se revine la pasul 2 atâta timp cât mai există perechi $(\mathbf{x}_r, \mathbf{d}_r)$ de aplicat.
8. Un ciclu de învăţare se încheie atunci când sunt epuizate (aplicate) toate perechile de antrenament. Cu ponderile sinaptice $\mathbf{W}^{[q]}$ obţinute în ciclul de învăţare curent se va starta un nou ciclu. Învăţarea se încheie când se constată convergenţa algoritmului (eroarea $\xi_r$ atinge un minim şi rămâne staţionară).

Nu vom prezenta în această lucrare demonstraţia algoritmului EBP. Această demonstraţie poate fi găsită în [LeC 85], [RUM 86], [WER 88] precum şi în alte lucrări care au preluat această problematică.

Schema bloc a algoritmului EBP este prezentată în figura 1.28.

Pe schema bloc din figura 1.28 se disting 2 faze:

1. faza directă (*feedforward*) în care sunt implicate blocurile (1) şi (2) şi în care RN, pe baza vectorului aplicat la intrare ($\mathbf{x}_r$), generează ieşirea reală $\mathbf{y}_r$. Blocurile (1) realizează produsul între o matrice şi un vector:

$$\mathbf{p}^{[q]} = \mathbf{W}^{[q]} \cdot \mathbf{y}^{[q-1]} \tag{1.34}$$

unde: $\mathbf{p}^{[q]}$ - vectorul potenţialelor neuronilor din stratul [q]
$\mathbf{W}^{[q]}$ - matricea ponderilor sinaptice aferentă neuronilor din stratul [q]
$\mathbf{y}^{[q-1]}$ - vectorul de ieşire din stratul anterior [q-1].

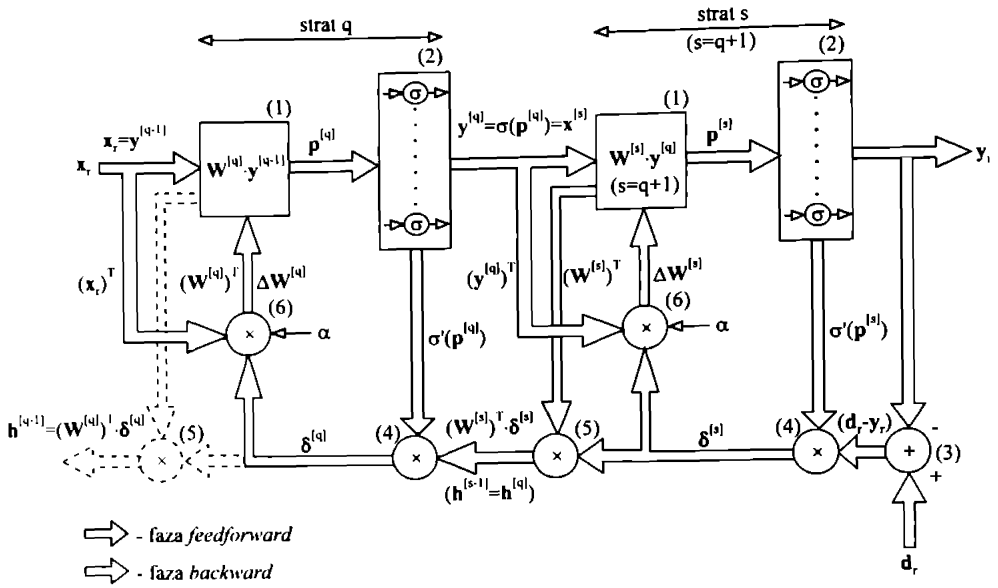



Fig. 1.28. Schema bloc a algoritmului EBP

$$\mathbf{p}^{[q]} = \begin{vmatrix} p_1 \\ p_2 \\ \dots \\ p_{K[q]} \end{vmatrix}^{[q]} ; \quad \mathbf{W}^{[q]} = \begin{Vmatrix} w_{11} & w_{12} & & w_{1K[q-1]} \\ w_{21} & w_{22} & & w_{2K[q-1]} \\ \dots & \dots & & \\ w_{K[q]1} & w_{K[q]2} & \dots & w_{K[q]K[q-1]} \end{Vmatrix}^{[q]} ; \quad \mathbf{y}^{[q-1]} = \begin{vmatrix} y_1 \\ y_2 \\ \dots \\ y_{K[q-1]} \end{vmatrix}^{[q-1]}$$

$w_{ij}$ - reprezintă ponderea sinaptică care leagă ieşirea neuronului j din startul [q-1] cu intrarea neuronului i din stratul [q]
$p_i$ - reprezintă potenţialul neuronului i din stratul [q]
$y_j$ - reprezintă ieşirea neuronului j din stratul [q-1]
K[q] - numărul de neuroni din stratul [q]
K[q-1] - numărul de neuroni din stratul [q-1]

Linia i a matricii $\mathbf{W}^{[q]}$ conţine ponderile conexiunilor de intrare în neuronul i din stratul [q]. Coloana j a matricii $\mathbf{W}^{[q]}$ conţine ponderile conexiunilor de ieşire aferente neuronului j din startul [q-1].
Blocurile (2) implementează funcţia de transfer σ:

$$\mathbf{y}^{[q]} = \sigma(\mathbf{p}^{[q]}) \tag{1.35}$$

Să notăm că în cadrul aplicaţiilor faza 1 este unica fază parcursă. În procesul de învăţare faza 1 este urmată de faza 2 (ajustarea ponderilor).

2. faza de propagare a erorii înapoi (*backward*) în care sunt implicate blocurile (3), (4), (5) şi (6). Blocul (3) calculează diferenţa dintre vectorul dorit la ieşire $\mathbf{d}_r$ şi vectorul de ieşire real $\mathbf{y}_r$.
   Blocurile (5) realizează produsul dintre o matrice şi un vector:

$$\mathbf{h}^{[q]} = (\mathbf{W}^{[q+1]})^T \cdot \delta^{[q+1]} \tag{1.36}$$

unde:
$(\mathbf{W}^{[q+1]})^T$ - transpusa matricii $\mathbf{W}^{[q+1]}$
$\delta^{[q+1]}$ - vectorul de eroare asociat stratului [q+1]

$$\delta^{[q+1]} = \begin{vmatrix} \delta_1 \\ \delta_2 \\ \dots \\ \delta_{K[q+1]} \end{vmatrix}^{[q+1]} ; \quad (\mathbf{W}^{[q+1]})^T = \begin{Vmatrix} w_{11} & w_{21} & & w_{K[q+1]1} \\ & & \dots & \\ w_{1K[q]} & w_{2K[q]} & & w_{K[q+1]K[q]} \end{Vmatrix}^{[q+1]} ; \quad \mathbf{h}^{[q]} = \begin{vmatrix} h_1 \\ h_2 \\ \dots \\ h_{K[q]} \end{vmatrix}^{[q]}$$

Să notăm că, în faza *feedforward*, în calcule se utilizează matricea sinaptică $\mathbf{W}$, iar în faza *backward* (retro - propagarea erorii), se utilizează transpusa acesteia $\mathbf{W}^T$.
Blocurile (4) calculează vectorii de eroare $\delta^{[q]}$. Vectorul de eroare asociat stratului de ieşire ($\delta^{[s]}$) se obţine:

$$\delta^{[s]} = (\mathbf{d}_r - \mathbf{y}_r) \otimes \sigma'(\mathbf{p}^{[s]}) \tag{1.37}$$

unde:
$\mathbf{d}_r$ - vectorul dorit la ieşire (pentru intrarea $\mathbf{x}_r$)
$\mathbf{y}_r$ - vectorul real obţinut la ieşire (în urma aplicării intrării $\mathbf{x}_r$)
$\sigma'(\mathbf{p}^{[s]})$ - vectorul derivatelor funcţiei de transfer în raport cu potenţialele neuronilor din stratul de ieşire.

$$\sigma'(p^{[s]}) = \begin{vmatrix} \sigma'(p_1) \\ \sigma'(p_2) \\ \dots \\ \sigma'(p_{K[s]}) \end{vmatrix}^{[s]} \quad \text{unde} : \sigma'(p_i^{[s]}) = \frac{\partial y_i^{[s]}}{\partial p_i^{[s]}}; \; i = 1, 2, \dots, K[s] \qquad d_r = \begin{vmatrix} d_1 \\ d_2 \\ \dots \\ d_{K[s]} \end{vmatrix}^{[s]} ; \quad y_r = \begin{vmatrix} y_1 \\ y_2 \\ \dots \\ y_{K[s]} \end{vmatrix}^{[s]}$$

K[s] - numărul de neuroni din stratul de ieşire [s]

În continuare, vectorii de eroare se obţin prin retro - propagarea vectorilor prin blocurile (5) şi (4). Fiecare strat are asociat o pereche de blocuri (5) - (4). Vectorii de eroare se obţin printr-un pseudoprodus a 2 vectori.

$$\delta^{[q]} = \mathbf{h}^{[q]} \otimes \sigma'(\mathbf{p}^{[q]}) \tag{1.38}$$

unde $\sigma'(\mathbf{p}^{[q]})$ reprezintă vectorul derivatelor funcţiei de transfer (derivate parţiale) în raport cu potenţialele neuronilor din stratul [q].

$$\sigma'(\mathbf{p}^{[q]}) = \begin{vmatrix} \sigma'(p_1) & & \\ & \sigma'(p_2) & \\ & & \\ & & \sigma'(p_{K[q]}) \end{vmatrix}^{[q]} \qquad \text{unde}: \sigma'(p_i^{[q]}) = \frac{\partial y_i^{[q]}}{\partial p_i^{[q]}};\ i = 1, 2, ..., K[q]$$

Fiecare componentă a vectorului de eroare $\delta^{[q]}$ (şi $\delta^{[s]}$) se obţine din produsul a 2 componente:

$$\delta_i^{[q]} = h_i^{[q]} \cdot \sigma'(p_i^{[q]});\quad i = 1, 2, \ldots, K[q] \text{ (indicele neuronului)}$$

Am notat această operaţie cu $\otimes$ şi am denumit-o pseudoprodus deoarece nu este vorba de produsul (scalar) a 2 vectori (în sens matematic); este o operaţie de înmulţire a 2 vectori în sensul în care aceasta este implementată în limbajele de programare.

Să notăm că operaţia cea mai complexă şi mai costisitoare din punct de vedere al timpului de execuţie, atât pentru faza *feedforward* cât şi pentru faza *backward*, este înmulţirea unei matrici cu un vector. Astfel de operaţii se implementează foarte eficient pe reţele sistolice (RS); acesta este de fapt obiectivul acestei lucrări.

### 1.6.5. Aplicarea algoritmului EBP

Aplicarea algoritmului EBP în aplicaţiile practice reale reclamă stabilirea unui context de lucru. Acest context introduce parametrii suplimentari care trebuie stăpâniţi şi ajustaţi (pe parcursul învăţării) astfel încât să se obţină o funcţionare corectă şi rapid convergentă a algoritmului. Redăm în continuare câţiva dintre cei mai importanţi parametrii de context, rezultaţi din experimentarea algoritmului pe aplicaţii practice concrete.

#### A. Termenul "momentum"

Versiunea teoretică a algoritmului EBP reclamă în mod normal o variaţie infinitezimală a ponderilor pe parcursul învăţării. Această variaţie este controlată de parametrul $\alpha$ numit constantă (parametru) de învăţare. Cu cât $\alpha$ este mai mare cu atât variaţia ponderilor va fi mai mare. În acest caz, procesul de învăţare este rapid dar poate conduce la un comportament oscilant. O variaţie bruscă a ponderilor poate avea o acţiune semnificativă asupra termenului de eroare. Corecţia, care ar trebui să fie semnificativă, va avea tendinţa să genereze o nouă variaţie bruscă. Va apărea în consecinţă un fenomen de oscilaţie care va împiedica stabilizarea procesului de învăţare. Este posibilă limitarea acestui comportament oscilant (fără a micşora $\alpha$) prin introducerea în corecţia din pasul curent a unui procent din corecţia pasului anterior. Formula de ajustare a ponderilor sinaptice devine:

$$w_{ij}^{[q]}(t+1) = w_{ij}^{[q]}(t) + \alpha \cdot \delta_i^{[q]} \cdot x_j^{[q]}(t) + \beta \cdot (\Delta w_{ij}^{[q]}(t)) \qquad (1.39)$$

Valoarea parametrului $\beta$ trebuie determinată experimental. Se ia în general între 0,6 şi 0,9. Acest nou termen impune memorarea corecţiei precedente (pentru fiecare pondere sinaptică) ceea ce înseamnă consum suplimentar de resurse de memorie.

### B. Utilizarea erorii medii sau instantanee

Ca şi în cazul *Adaline*, versiunea exactă a algoritmului EBP revendică cunoaşterea mediei semnalelor de intrare şi a mediei semnalelor dorite la ieşire. Este totuşi posibilă învăţarea pornind de la valori instantanee, dacă se menţine o constantă de învăţare ($\alpha$) mică. Observăm că apare un antagonism între viteza algoritmului de învăţare şi procedura de învăţare instantanee (care este mult mai uşor de implementat în practică).

Se poate totuşi implementa un algoritm care realizează o formă de mediere pe termen scurt, acumulând variaţiile gradientului pentru mai mulţi vectori de intrare aplicaţi succesiv (fără a ajusta ponderile), apoi făcând o ajustare a ponderilor care ţine cont de toate variaţiile de gradient cumulate. Este vorba de metoda ajustării cumulative spre deosebire de metoda ajustării incrementale (care realiza ajustarea ponderilor după fiecare prototip aplicat la intrare). Ambele tipuri de ajustări pot conduce la rezultate bune.

Trebuie subliniat însă că învăţarea decurge bine în condiţii aleatoare. Pare, deci, mai convenabilă ajustarea incrementală (pentru că este mai uşor de implementat), dar cu preluarea prototipurilor din setul de antrenament şi aplicarea acestora pe intrarea RN într-o secvenţă aleatoare. Acest caracter aleator al secvenţei de prototipuri introduce acel "zgomot" necesar rezolvării problemei de mediere.

### C. Iniţializarea ponderilor

Ponderile nu trebuie iniţializate cu valori egale. În acest caz, pentru că termenul de eroare este perceput de către neuronii unui strat după ce acesta a traversat ponderile sinaptice, stratul intern care precede stratul de ieşire va percepe erori identice pentru toţi neuronii săi (fig. 1.29).

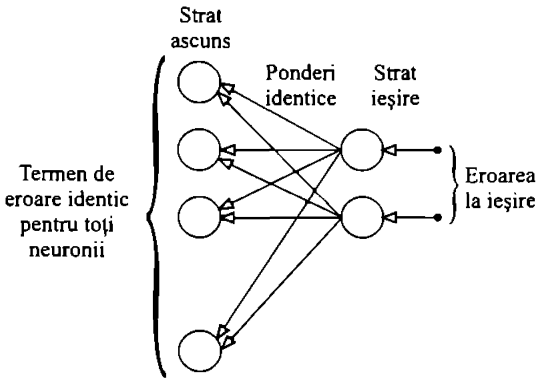



Fig. 1.29. Problema ponderilor iniţiale egale

Ponderile stratului ascuns din fig. 1.29 nu vor varia independent creând un fel de stare stabilă parazită care împiedică un proces de învăţare corect şi care reduce artificial numărul de straturi din componenţa RN. Soluţia la această problemă rezidă în iniţializarea ponderilor sinaptice cu valori aleatoare mici.

Faza de iniţializare a ponderilor este importantă şi din alt punct de vedere; vom ilustra acest lucru printr-un exemplu inspirat din [McC 86].

Considerăm o RN cu 2 straturi, în care fiecare strat conţine un singur neuron (fig. 1.30).

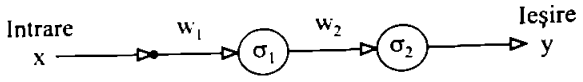



Fig. 1.30. Exemplu relativ la iniţializarea ponderilor sinaptice

Cele 2 funcţii de transfer $\sigma_1$ şi $\sigma_2$ sunt identice (funcţia sigmoidală unipolară exprimată prin relaţia (1.3)). Valoarea de prag a ambilor neuroni este 0. Dorim ca această RN să înveţe funcţia identică pe două perechi de antrenament:

| x | d |
|---|---|
| 0 | 0 |
| +1 | +1 |

Dacă trasăm suprafaţa de eroare $(y - d)^2$ în funcţie de ponderile $w_1$ şi $w_2$ obţinem suprafaţa din fig. 1.31.

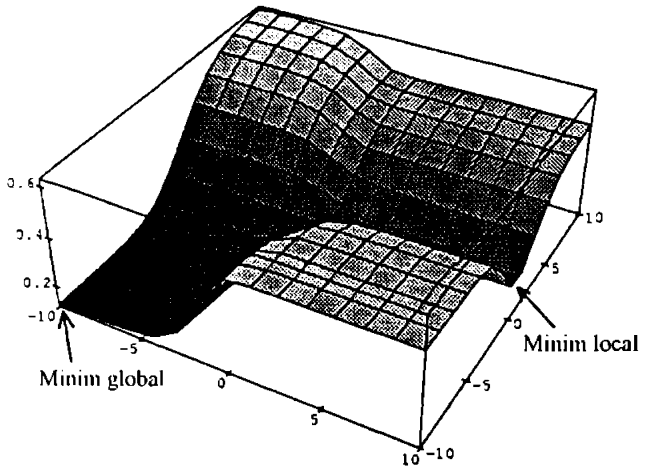



Fig. 1.31. Suprafaţă de eroare cu 2 puncte de minim

Suprafaţa de eroare din fig. 1.31 indică faptul că sistemul admite două puncte de minim: un minim global cu o eroare minimă de aproximativ 0,025 şi un minim local cu o eroare asociată de 0,45. Atingerea unuia sau a altuia dintre aceste 2 minime depinde esenţial de iniţializarea ponderilor sinaptice. În acest caz simplu, vizualizarea suprafeţei de eroare a fost posibilă, dar nu va fi posibilă şi în cazul general. Aşadar, chiar şi iniţializarea cu valori aleatoare poate conduce RN (în timpul învăţării) spre minime locale. Pentru a evita această capcană învăţarea trebuie restartată cu alte ponderi iniţiale (aleatoare).

**D. Panta funcţiei de activare**

Funcţiile de activare (transfer) sigmoidale (relaţiile (1.3) şi (1.4) şi figura 1.3) se caracterizează prin factorul de pantă $\lambda = 1/T$. Derivata funcţiei de activare atinge valoarea maximă $(\lambda/2)$ în punctul $p = 0$. Algoritmul EBP ajustează ponderile sinaptice proporţional cu factorul de scară $\sigma'(p)$. Drept consecinţă, ponderile aferente neuronilor nedecişi vor fi puternic ajustate în timp

ce ponderile aferente neuronilor supraexcitaţi sau suprainhibaţi vor fi ajustate mult mai puţin în procesul de învăţare. Deoarece semnalele locale de eroare $\delta_{ir}^{[q]}$ se obţin utilizând $\sigma'(p)$ ca factor multiplicator, componentele de eroare transmise înapoi spre straturile anterioare, vor fi mari numai pentru neuronii aflaţi în zona de indecizie. Pentru o anumită constantă de învăţare $\alpha$ fixată, ajustarea ponderilor va fi proporţională cu factorul de pantă $\lambda = 1/T$. Deci, utilizând funcţii de transfer cu pantă mare vom obţine rezultate similare cu cele obţinute în cazul utilizării constantelor de învăţare ($\alpha$) mari. Sugestia noastră este de a menţine panta funcţiei de activare la valoarea standard $\lambda = 1/T = 1$ şi de a controla viteza de convergenţă a procesului de învăţare numai din constanta $\alpha$.

### 1.6.6. Aplicaţie demonstrativă

Vom considera 2 mulţimi de puncte în planul cartezian (fig. 1.32), care definesc două regiuni dintre care una este convexă.

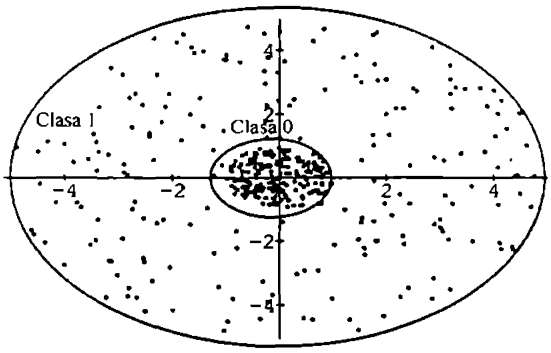



Fig. 1.32. Repartiţia punctelor în planul cartezian

Dacă dorim să clasificăm aceste puncte în două clase (asociate celor 2 regiuni), în funcţie de coordonatele carteziene al acestor puncte, putem utiliza un perceptron cu 2 straturi (fig. 1.33).
Vectorii de intrare **x** vor conţine coordonatele carteziene aferente punctelor de clasificat iar ieşirea y va specifica clasa căreia îi aparţine vectorul de intrare **x**.
Aplicând algoritmul EBP, cu următorii parametrii:

- $\alpha = 0,7$
- fără termen *momentum*
- ajustare incrementală a ponderilor (învăţare instantanee)
- valoarea de prag a fiecărui neuron este realizată printr-o intrare fixată la +1 şi un coeficient sinaptic variabil
- variabile reale
- obiectivul vizat este clasificarea punctelor în cele 2 clase cărora li s-au asociat valorile $y = +1$ şi respectiv $y = 0$
- funcţia de transfer utilizată:

$$\sigma(x) = \text{th}(x)$$
$$\sigma'(x) = 1 - \text{th}^2(x)$$

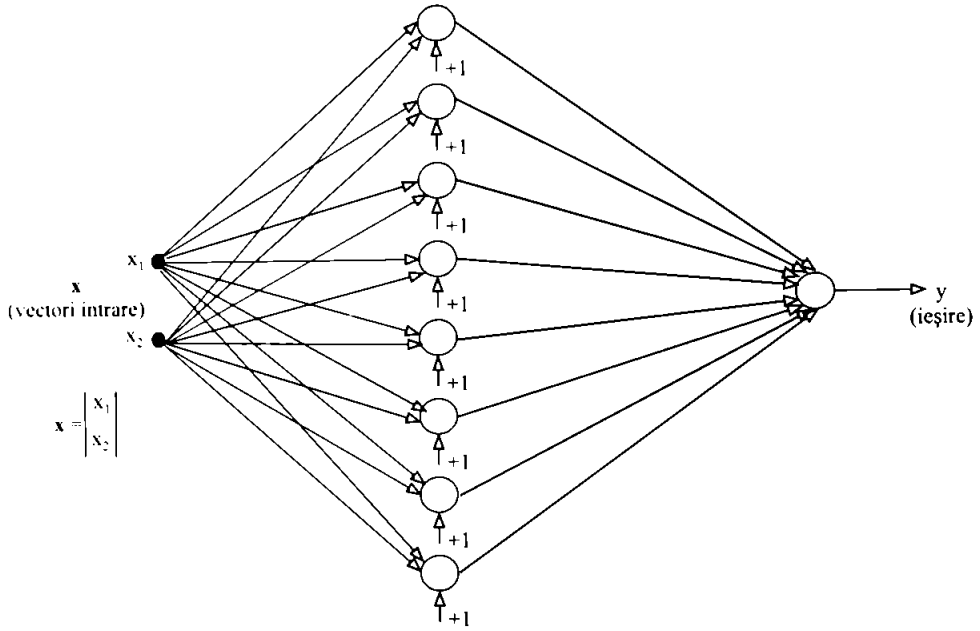
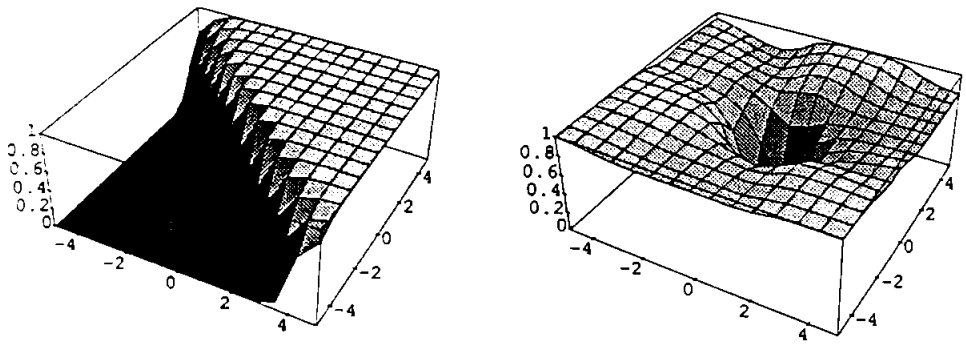


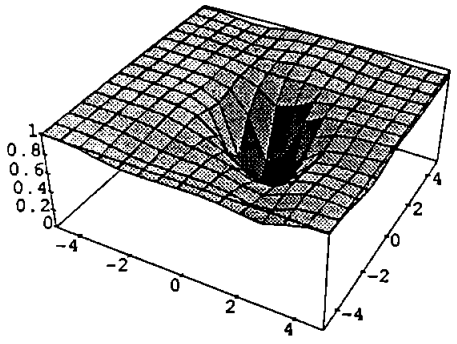

Fig. 1.33. Structura reţelei

S-a obţinut separarea în clase reprezentată în fig. 1.34. Setul de antrenament (prototipuri) a fost format din 500 perechi (**x**, d). În figura 1.34 se redau suprafeţele de separare (y = f(**x**)) obţinute. După cum se poate constata rezultatul nu este perfect dar constituie un separator convenabil. Trebuie să notăm că numărul de iteraţii poate fi mărit şi că parametrii utilizaţi n-au fost aleşi în scopul obţinerii celei mai bune soluţii posibile (parametrii optimali nu sunt cunoscuţi).

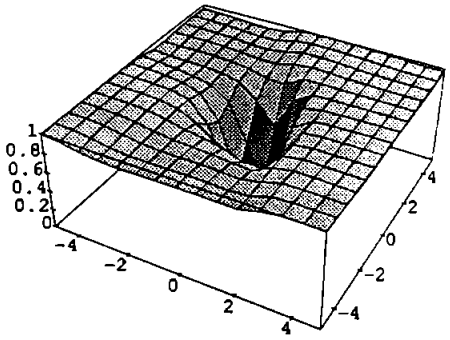


a) suprafaţa de separare iniţială

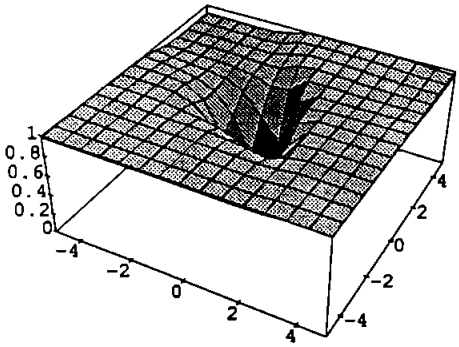
b) după 200 prezentări




c) după 400 prezentări




d) după 1000 prezentări




e) după 2000 prezentări

Fig. 1.34. Evoluția suprafeței de separare în timpul învățării

### 1.6.7. Concluzii

După cum am putut constata, aplicarea algoritmului EBP este relativ complicată şi calculele efectuate sunt destul de departe de ceea ce se întâmplă într-un sistem nervos. În plus, algoritmul suferă de anumite limitări:

- incapacitatea de a determina numărul optim de neuroni din fiecare strat în funcție de aplicație;
- viteza de convergență lentă care depinde de mărimea setului de antrenament şi de parametrii rețelei;
- riscul deraierii spre minime locale.

Trebuie să-i recunoaştem totuşi o certă aplicabilitate industrială, care l-a transformat într-un motor al cercetării în domeniul conexionist. Dar RN nu trebuie limitate la arhitectura *feedforward*; cercetările au evoluat şi trebuie să evolueze şi mai departe pe arhitecturi mai complexe (rețele recurente, rețele probabiliste etc.).

### 1.7. Modelul Hopfield

#### 1.7.1. Introducere

Modelul *Hopfield* este un model recurent sau rebuclat. Într-o reţea rebuclată fiecare neuron este în relaţie directă cu toţi ceilalţi. Nu se disting deci unităţi (neuroni) de intrare şi unităţi de ieşire. Fiecare neuron este în acelaşi timp şi intrare şi ieşire, şi mărimea care-l caracterizează este starea sa (fig. 1.35).

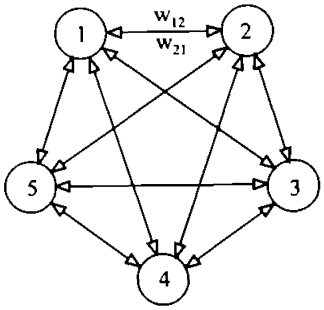



Fig. 1.35. Structura unei reţele *Hopfield*

O reţea recurentă se caracterizează prin dinamica sa, ceea ce înseamnă evoluţia stărilor unităţilor sale în timp. Această evoluţie nu este controlată şi nivelul de activitate al neuronilor va evolua evident datorită interacţiunilor acestora cu ceilalţi neuroni din componenţa reţelei. În practică, pentru a crea sisteme exploatabile, se caută ca aceste reţele să aibă o dinamică care să conducă spre un punct de stabilitate, numit atractor. Această dinamică va interesa atunci când aplicaţia revendică verdicte (răspunsuri) unice. Nu trebuie exclusă utilitatea unor atractori circulari. În acest caz starea de echilibru a reţelei nu este punctuală ci este ciclică (un ciclu limitat de stări); aceste cicluri limitate pot fi utilizate pentru memorarea unor secvenţe.

Într-o reţea *Hopfield* există o dinamică naturală. Obiectivul demersului nostru este de a exploata această dinamică naturală pentru a determina reţeaua să funcţioneze ca o memorie asociativă. Problema trebuie abordată sub următoarele 4 aspecte:

1. În anumite condiţii se poate asocia o energie globală întregului sistem dinamic (reţelei). Reducerea acestei energii reprezintă o condiţie de stabilitate pentru sistemul dinamic. Cum va putea acţiona o variaţie locală (la nivelul unui neuron) pentru reducerea energiei globale?

2. În general, care sunt condiţiile care conduc la un sistem a cărui energie scade? Răspunsul la această întrebare ne va permite să controlăm reţeaua.

3. Cum se poate construi un sistem ale cărui stări stabile reprezintă cu precizie informaţia pe care dorim s-o memorăm? Dacă vom putea găsi o regulă care să răspundă la această întrebare, atunci vom fi capabili să construim memorii asociative a căror dinamică naturală de sistem va conduce spre informaţiile pe care le-am stocat.

4. Există reguli care răspund la întrebarea precedentă: regula lui *Hebb*, regula proiecţiei. Vom face analiza acestora.

### 1.7.2. Dinamica reţelelor Hopfield

Dinamica reţelelor *Hopfield* compuse din N neuroni este descrisă de ecuaţia clasică a neuronului *McCulloch-Pitts*. În formă discretă ecuaţia se scrie:

$$p_i(t) = \sum_{j=1}^{N} w_{ij} \cdot x_j(t-1) \tag{1.40}$$

$$\text{şi} \quad x_i(t) = \text{sign}(p_i(t) - 2\theta_i) \tag{1.41}$$

unde:

$p_i$ - potenţialul neuronului i
$w_{ij}$ - ponderea conexiunii dinspre neuronul j spre neuronul i
$x_i$ - ieşirea (starea) neuronului i
$\theta_i$ - valoarea de prag asociată neuronului i
sign - funcţia sign reprezentată în fig. 1.2. b).

În cazul reţelelor recurente se disting 2 metode de actualizare a stării neuronilor: secvenţială aleatoare şi respectiv paralelă. În cazul metodei secvenţiale, la momentul t, se alege aleator un neuron i pe baza unei reguli de distribuţie uniformă. Starea neuronului selectat se actualizează utilizând regula *McCulloch-Pitts* şi se continuă selecţia aleatoare. În cazul actualizării paralele, la momentul t, se calculează potenţialele tuturor neuronilor şi starea acestora se actualizează simultan.

*Hopfield* a arătat [HOP 82] că dinamica secvenţială poate conduce la o stare stabilă în anumite condiţii:

- matricea ponderilor sinaptice **W** să fie simetrică ($w_{ij} = w_{ji}$);
- elementele diagonalei matricii **W** să fie nenegative.

Demonstraţia se bazează pe faptul că putem asocia o funcţie de energie unui sistem dinamic astfel încât toate schimbările de stare aferente elementelor acestuia să nu poată conduce decât la scăderea funcţiei de energie. Suplimentar, această funcţie de energie admite un minim absolut şi descreşterea energiei se face în timp finit.

Prin definiţie, funcţia de energie se exprimă la momentul t prin relaţia:

$$E(t) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} \cdot x_i(t) \cdot x_j(t) \tag{1.42}$$

unde N reprezintă numărul de componente din vectorul **x**, cu alte cuvinte numărul de neuroni din reţea. Vom pune în evidenţă efectul pe care-l va avea asupra energiei sistemului, schimbarea stării unui singur neuron. Considerând timpul discret, vom avea:

$$E(t+1) - E(t) = -\frac{1}{2}\left[\sum_{i=1}^{N}\sum_{j=1}^{N}\left(w_{ij} \cdot x_j(t+1) - 2\theta_i\right) x_i(t+1) - \sum_{i=1}^{N}\sum_{j=1}^{N}\left(w_{ij} \cdot x_j(t) - 2\theta_i\right) x_i(t)\right] \tag{1.43}$$

Vom considera că actualizarea stării sistemului este de tip secvenţială aleatoare. La fiecare moment t, se selectează aleator un neuron şi i se calculează noua stare. Această metodă de selecţie este denumită metoda *Monte Carlo*.

Pot să apară 2 situaţii:

- neuronul selectat nu-şi schimbă starea şi, în acest caz, dacă i este indicele neuronului extras, vom avea $x_i(t+1) = x_i(t)$;
- neuronul selectat îşi schimbă starea: $x_i(t+1) = - x_i(t)$.

Care va fi variaţia de energie dacă un singur neuron k îşi schimbă starea?
Trebuie dezvoltată expresia $\Delta E$ pentru a face să apară explicit sumele pentru $i \neq k$, $j \neq k$, $i = k$, $j = k$; apoi expresia trebuie prelucrată pentru a o face dependentă numai de variabila t. După câteva calcule simple [ZUR 92] se va obţine:

$$\Delta E = \sum_{j=1}^{N} \left( w_{kj} \cdot x_j(t) - 2\theta_k \right) x_k(t) + \sum_{i=1}^{N} w_{ik} \cdot x_k(t) \cdot x_i(t) - 2w_{kk} \cdot x_k^2(t) \qquad (1.44)$$

Prin ipoteză, matricea **W** este simetrică, ceea ce conduce la:

$$\Delta E = 2x_k(t) \cdot \left( \sum_{j=1}^{N} w_{kj} \cdot x_j(t) - \theta_k \right) - 2w_{kk} \cdot x_k^2(t) \qquad (1.45)$$

Dacă alegem valoarea de prag $\theta_i = 0$, şi substituim valorile lui $x_i$, se obţine:

$$\Delta E = - 2 - 2w_{kk} \qquad (1.46)$$

Dacă se respectă condiţia $w_{kk} \geq - 1$, variaţia de energie provocată de schimbarea stării unui singur neuron este întotdeauna negativă. Funcţia de energie va fi monotonă descrescătoare şi valoarea ei evoluează până la atingerea unui minim (cel puţin un minim local) după care va rămâne constantă.

Rămâne de arătat că funcţia de energie are o valoare minimă finită. Din moment ce numărul de neuroni din reţea este finit şi valorile de pe ieşirile acestora nu pot fi decât +1 sau -1, este evident că minimul funcţiei de energie este finit. Funcţia de energie este o sumă dublă în care intră stările tuturor neuronilor şi, în consecinţă, va fi finită. Rezultă că, după un număr de iteraţii, va fi atins un punct de minim energetic care reprezintă un atractor al sistemului. În figura 1.36 se prezintă imaginea procesului de descreştere a funcţiei de energie în spaţiul stărilor.

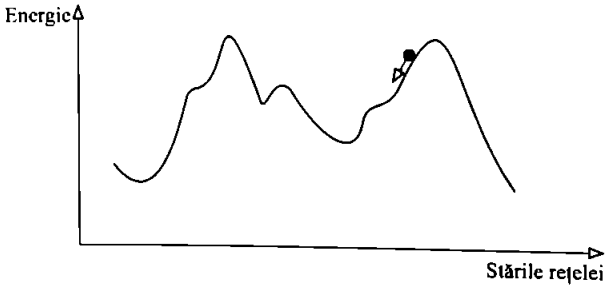



Fig. 1.36. Reprezentarea funcţiei de energie

Este important de subliniat că aceeaşi reţea (sistem) admite mai multe minime care pot avea energii asociate diferite şi care sunt accesibile pornind de la configuraţii iniţiale diferite. De asemenea, minimul absolut poate fi atins plecând din mai multe configuraţii iniţiale.

Va trebui analizat cum dinamica unei reţele recurente va permite acesteia, în anumite condiţii, să evolueze spre aceşti atractori. Dacă dorim să exploatăm această proprietate, pentru a construi o memorie asociativă de exemplu, trebuie să fim capabili să controlăm poziţia acestor atractori. Această problemă se numeşte "problema dinamicii inverse" şi se formulează astfel: "Este posibilă construcţia unei reţele recurente care să admită ca atractori un ansamblu de stări alese arbitrar?"

### 1.7.3. Problema dinamicii inverse

Să considerăm că dispunem de un ansamblu de configuraţii (*patterns*) care dorim să constituie baza de referinţă (setul de stări stabile) pentru reţeaua recurentă. Această operaţie de forţaj este denumită fază de învăţare.

Atunci când aplicăm o configuraţie (stare) deteriorată vom dori ca reţeaua să răspundă prin una dintre configuraţiile (stările) de referinţă. Acest principiu de restaurare a unei configuraţii de referinţă pornind de la o configuraţie deteriorată se numeşte memorie asociativă. Principiul este descris în figura 1.37.

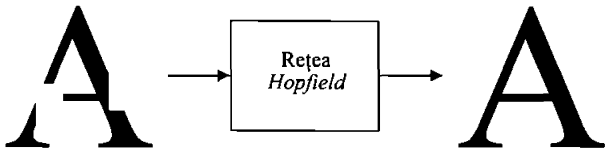



Fig. 1.37. Recunoaşterea caracterelor cu ajutorul reţelei *Hopfield*

În acest exemplu se presupune că imaginea caracterului A este un prototip. La intrarea reţelei se aplică un A deteriorat. După câteva iteraţii, reţeaua trebuie să răspundă cu prototipul A.

Prima rezolvare (parţială) la care putem recurge este regula lui *Hebb*, care a fost descrisă anterior. Matricea sinaptică **W** controlează poziţia atractorilor. Calculând convenabil valorile coeficienţilor sinaptici $w_{ij}$, atractorii vor deveni un codaj de prototipuri şi reţeaua, prin dinamica ei, va juca rolul de decodor.

### 1.7.4. Regula lui Hebb

Regula lui *Hebb* pune la dispoziţie o metodă foarte simplă pentru calculul ponderilor sinaptice. Formularea acesteia se rezumă la:

$$w_{ij} = \frac{1}{N}\left(\sum_{s=1}^{P} x_i(s) \cdot x_j(s)\right) \qquad (1.47)$$

$w_{ij}$ - coeficientul sinaptic ij din matricea **W**.
$x_i(s)$ - componenta i a stării prototip s
P - numărul de stări prototip
N - numărul de neuroni din reţea

Pentru fiecare prezentare a unui prototip **x**(s), se ajustează matricea sinaptică cu valori rezultate din activitatea conjugată a componentelor $x_i$, $x_j$.

### 1.7.4.1. Interpretare

Din punct de vedere geometric, dacă stările prototip sunt ortogonale, matricea **W** poate fi interpretată ca matrice de proiecţie ortogonală în subspaţiul generat de vectorii prototip.

### 1.7.4.2. Demonstraţie

Presupunem că stările prototip sunt reciproc ortogonale. Vom forma matricea A de dimensiune N x P şi ale cărei coloane vor conţine cele P stări prototip.

$$\mathbf{A} = (\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(P))$$

Regula lui *Hebb* se va scrie:

$$\mathbf{W} = \frac{1}{N} \cdot \mathbf{A} \cdot \mathbf{A}^T \tag{1.48}$$

Prin definiţie, proiecţia ortogonală a tuturor vectorilor **x** pe subspaţiul definit de coloanele matricii A va fi:

$$\mathrm{Proj}_{\perp}\mathbf{x} = \mathbf{A}\cdot(\mathbf{A}^T\cdot\mathbf{A})^{-1}\cdot\mathbf{A}^T\cdot\mathbf{x} \tag{1.49}$$

unde $\mathrm{Proj}_{\perp}$ reprezintă operatorul de proiecţie ortogonală.
Matricea $\mathbf{A}^T\cdot\mathbf{A}$ va avea forma:

$$\mathbf{A}^T \cdot \mathbf{A} = \begin{Vmatrix} \mathbf{x}(1)^2 & \mathbf{x}(2)\cdot\mathbf{x}(1) & & \ldots \\ \mathbf{x}(1)\cdot\mathbf{x}(2) & \mathbf{x}(2)^2 & \ldots & \ldots \\ \ldots & \ldots & \mathbf{x}(3)^2 & \ldots \\ & & & \ldots \\ & & & \mathbf{x}(P)^2 \end{Vmatrix}$$

unde **x**(i) reprezintă starea prototip numărul i.

Am presupus prin ipoteză că vectorii stărilor prototip sunt ortogonali. Rezultă că produsele scalare $\mathbf{x}(i)\cdot\mathbf{x}(j)$ vor fi nule pentru $i \neq j$. Matricea $\mathbf{A}^T\cdot\mathbf{A}$ este deci o matrice diagonală, având pe diagonală elementele $\mathbf{x}(i)^2$. Dar componentele oricărui vector sunt egale cu +1 sau -1; rezultă că toate elementele de pe diagonala matricii $\mathbf{A}^T\cdot\mathbf{A}$ vor avea valoarea N. Deci:

$$\mathbf{A}^T\cdot\mathbf{A} = N\cdot\mathbf{I} \qquad \text{iar} \qquad (\mathbf{A}^T\cdot\mathbf{A})^{-1} = \frac{1}{N}\cdot\mathbf{I} \tag{1.50}$$

unde **I** reprezintă matricea unitate de dimensiune P.
Dacă prototipurile sunt ortogonale 2 câte 2, relaţia (1.49) devine:

$$\mathrm{Proj}_{\perp}\mathbf{x} = \frac{1}{N}\cdot\mathbf{A}\cdot\mathbf{A}^T\cdot\mathbf{x} \tag{1.51}$$

Am regăsit astfel formula lui *Hebb*. Din moment ce această operaţie este o proiecţie ortogonală, va fi adevărată şi ecuaţia:

$$\mathbf{W}\cdot\mathbf{x}(s) = \mathbf{x}(s) \text{ pentru orice prototip } \mathbf{x}(s);\ s = 1, 2, \ldots, P \qquad (1.52)$$

Dacă prototipurile sunt neortogonale relaţia (1.52) nu se verifică.

Rămân de valorificat, în final, şi proprietăţile funcţiei de transfer (neliniare) aplicată de către fiecare neuron propriului potenţial. Reamintim că toate componentele vectorilor prototip au valorile ±1. Înseamnă că este suficient să fixăm o valoare de prag între 0 şi +1, şi stările prototip vor fi automat şi stări stabile ale reţelei.

Operaţia de proiecţie efectuată prin produsul dintre un vector de intrare (oarecare) şi matricea ponderilor sinaptice nu conservă norma. Componentele vectorilor de intrare pot fi oarecare şi pot să nu corespundă vectorilor prototip. Dar fiecare componentă care depăşeşte valoarea de prag va fi restabilită la +1, iar cele care nu depăşesc pragul vor fi restabilite la -1. Astfel, configuraţiile prototip pot fi restaurate dacă se porneşte de la o configuraţie vecină.

### 1.7.5. Proprietăţile reţelei Hopfield

Reţeaua *Hopfield* este capabilă să realizeze funcţia de memorie asociativă. Dacă iniţializăm reţeaua într-o stare oarecare şi această stare nu este prea îndepărtată de un prototip codificat (distanţă *Hamming*), reţeaua va evolua spre prototip. În caz contrar, reţeaua va evolua spre o stare nedefinită numită stare parazită.

Atractivitatea se defineşte astfel: o configuraţie **x** este un k-atractor ($1 \le k \le N$) dacă, plecând dintr-o stare alterată care prezintă k erori în raport cu configuraţia prototip **x**, reţeaua va evolua într-un mod iterativ sincron şi determinist stabilizându-se în starea **x**.

Utilizând regula lui *Hebb*, dacă toate prototipurile sunt reciproc ortogonale, fiecare prototip va reprezenta un atractor cu un bazin de atracţie până la distanţa *Hamming*:

$$k = \left[\frac{N}{2P}\right] \qquad (1.53)$$

Deci, se observă că, cu cât stocăm mai multe prototipuri cu atât atracţia fiecărui prototip va scădea. Pentru a implementa o funcţie de memorie asociativă performantă, va fi necesar un număr mare de neuroni corelat cu stocarea a cât mai puţine prototipuri.

Această atractivitate reprezintă un fel de măsură globală a performanţelor reţelei dar principala constrângere provine din necesitatea de a avea vectori prototip ortogonali. Este evident posibilă aplicarea unor proceduri de ortogonalizare de tip *Gramm - Schmidt* (o preprocesare a vectorilor prototip pentru ortogonalizare), dar este de asemenea posibilă utilizarea altor metode de rezolvare mai puţin restrictive.

O altă limitare severă, în cazul utilizării regulii lui *Hebb*, se referă la capacitatea de stocare a configuraţiilor prototip. Această capacitate este de ordinul 14% din numărul de neuroni dacă nu acceptăm o eroare mai mare de 3%, estimată prin distanţa *Hamming* dintre vectorul prototip şi vectorul aplicat. Pe măsură ce se depăşeşte această capacitate, performanţele scad dramatic. Reţeaua începe să uite chiar şi ce a învăţat.

În cele ce urmează, vom prezenta câteva metode algebrice descrise în [COT 88] care permit creşterea capacităţii de stocare şi, în acelaşi timp, eliminarea unor constrângeri foarte limitative. Primul principiu este cel enunţat în [KOH 86] (memoria optimală), regulă reluată mai târziu sub denumirea de regula proiecţiei.

### 1.7.6. Regula proiecţiei

Vom studia posibilitatea construirii unei matrici sinaptice care garantează stabilitatea configuraţiilor prototip şi vom examina capacitatea de asociere aferentă reţelei obţinute. Considerăm de această dată că vectorii prototip nu mai sunt ortogonali între ei, dar sunt încă liniar independenţi. Formal, sistemul de rezolvat se reduce la:

$$\text{sign}(\mathbf{W}\cdot\mathbf{A}) = \text{sign}(\mathbf{A})$$

Într-o primă abordare, vom considera neuronii fără intrare de prag şi fără iteraţii repetate. La nivel fundamental problema rămâne identică [COT 88] şi poate fi rezolvată cu ajutorul sistemului de ecuaţii:

$$\mathbf{A} = \mathbf{W}\cdot\mathbf{A} \tag{1.54}$$

unde $\mathbf{A} = (\mathbf{x}(1), \mathbf{x}(2), \ldots, \mathbf{x}(P))$. **A** este o matrice (N x P), unde N reprezintă numărul de neuroni şi P numărul de prototipuri. Este cazul tipic de memorie asociativă.

Matricea **A** nu este pătrată şi nu admite inversă. Totuşi, există o soluţie non - unică, dacă rangul matricii **A** (numărul de vectori prototip) este inferior lui N (numărul de componente dintr-un vector prototip = numărul de neuroni). Este vorba de generalizarea inversei pentru matrici dreptunghiulare.

$$\mathbf{W} = \mathbf{A}\cdot\mathbf{A}^{+} \tag{1.55}$$

unde:

$\mathbf{A}^{+}$ este o matrice (P x N) numită pseudo - inversa matricii **A**.

$\mathbf{A}^{+}$ va fi:

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\cdot\mathbf{A})^{-1}\cdot\mathbf{A}^{T} \tag{1.56}$$

Condiţia de liniar independenţă impusă coloanelor matricii **A**, ne asigură că $\mathbf{A}^{T}\cdot\mathbf{A}$ are rangul P şi este deci inversabilă, ceea ce asigură existenţa matricii $\mathbf{A}^{+}$. Această metodă a fost tratată exhaustiv de [KOH 86], [PER 86], [PER 87], [GUY 88]. În cazul prototipurilor liniar independente, pentru problema auto - asocierii soluţia va fi:

$$\mathbf{W} = \mathbf{A}\cdot(\mathbf{A}^{T}\cdot\mathbf{A})^{-1}\cdot\mathbf{A}^{T} \tag{1.57}$$

Şi în acest caz, **W** va fi o matrice de proiecţie pe spaţiul vectorial generat de vectorii prototip. Este evident că, în cazul în care configuraţiile prototip sunt ortogonale, matricea pseudo - inversă devine:

$$\mathbf{A}^{+} = \frac{1}{N}\cdot(\mathbf{A}^{T}) \tag{1.58}$$

În acest caz particular, matricea sinaptică devine:

$$\mathbf{W} = \frac{1}{N}\cdot(\mathbf{A}\cdot\mathbf{A}^{T}) \tag{1.59}$$

Am regăsit astfel regula lui *Hebb*.

Există mai multe metode de calcul a pseudo - inversei unei matrici. Prima se referă evident la calculul algebric cunoscut, dar acest principiu este relativ greoi deoarece el obligă la recalcularea completă a pseudo - inversei după aplicarea fiecărui prototip. Suplimentar, calculul pseudo - inversei necesită aplicarea unor algoritmi cum ar fi cel al lui *Choleski*. O altă metodă de calcul a pseudo - inversei constă în utilizarea algoritmului lui *Greville* [KOH 86] care simplifică mult procesul. Redăm acest algoritm în paragraful următor:

### 1.7.7. Algoritm de proiecție iterativ

1. Inițializare:

$$\mathbf{x}(0) = 0$$
$$\mathbf{W}(0) = 0$$

2. Aplicarea unui prototip $\mathbf{x}(t)$ și calculul lui q:

$$q = \frac{1}{\left\|(\mathbf{x}(t) - \mathbf{W}(t-1)) \cdot \mathbf{x}(t)\right\|^2} \qquad (1.60)$$

Dacă q este foarte mare se rejectează prototipul $\mathbf{x}(t)$ și se revine la pasul 2. Dacă nu, se trece la pasul 3

3. Actualizarea matricii $\mathbf{W}$.

$$\mathbf{W}(t+1) = \mathbf{W}(t) + q \cdot [\mathbf{I} - \mathbf{W}(t)] \cdot \mathbf{x}(t) \cdot [(\mathbf{I} - \mathbf{W}(t)) \cdot \mathbf{x}(t)]^T \qquad (1.61)$$

4. Revenire la pasul 2 atâta timp cât mai există prototipuri de învățat.

Este interesant de observat că, dacă q este foarte mare (tinde la infinit), avem de-a face cu un prototip $\mathbf{x}(t)$ liniar dependent în raport cu coloanele (prototipurile) deja introduse. Acest nou prototip nu aduce informații suplimentare și din acest motiv va fi rejectat.

Pentru matricea de proiecție obținută, fiecare prototip stocat va fi un atractor până la o distanță *Hamming* [COT 88]:

$$k = \frac{1}{2 \cdot \max_i(w_{ii})}; \quad 1 \le i \le N \qquad (1.62)$$

Inconvenientul acestei metode rezidă din faptul că, pentru fiecare pondere sinaptică $w_{ij}$, calculul face apel la un factor global reprezentat de termenul q. Există totuși o metodă locală [DIE 87] care, pentru un neuron dat, nu face apel decât la potențialul său și al neuronului la care acesta este conectat. Este metoda gradientului stohastic care se apropie de cea propusă de [WID 60] pentru modelul *Adaline*.

### 1.7.8. Metoda gradientului (Proiecția Delta)

Este posibilă utilizarea unei metode de gradient stohastic care minimizează o funcție de eroare (vezi modelul *Adaline*). În acest caz învățarea se va realiza printr-o prezentare repetată a tuturor vectorilor prototip. Această metodă este de asemenea costisitoare (în ceea ce privește

cantitatea de calcul) dar limitează aria calculelor doar la interacţiuni strict locale şi se înscrie ca o candidată preferabilă pentru implementările VLSI.

Metoda gradientului vizează minimizarea unui termen de eroare. Pentru o reţea recurentă, cum este cazul reţelei *Hopfield*, se propune utilizarea unui termen de eroare de forma [DIE 87]:

$$\varepsilon(t) = \sum_{i=1}^{N}\left(\sum_{j=1}^{N} w_{ij}\cdot x_j(t) - x_i(t)\right)^2 \tag{1.63}$$

Pentru o memorie auto - asociativă, acest termen de eroare măsoară diferenţa dintre starea corespunzătoare prototipului dorit şi cea obţinută prin înmulţirea acestuia cu matricea sinaptică **W**. Dacă vom reuşi să reducem acest termen de eroare la zero, înseamnă că am memorat prototipul **x**(t) prezentat. Va trebui calculat gradientul acestui termen de eroare în funcţie de matricea **W**.

$$\frac{\partial\varepsilon(t)}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{i=1}^{N}\left(\sum_{j=1}^{N} w_{ij}\cdot x_j(t) - x_i(t)\right)^2\right) = 2\sum_{j=1}^{N}\left(w_{ij}\cdot x_j(t) - x_i(t)\right)x_j(t) \tag{1.64}$$

Dacă vom recurge la o metodă de ajustare identică cu cea propusă de *Widrow* pentru modelul *Adaline*, va fi suficient să ajustăm coeficienţii matricii **W** după fiecare prezentare a unui vector prototip:

$$w_{ij}(t) = w_{ij}(t-1) - \alpha\frac{\partial\varepsilon(t)}{\partial w_{ij}} \tag{1.65}$$

Parametrul $\alpha$ trebuie astfel ales încât rezultatele să rămână independente de numărul de elemente (puncte) de calcul, atunci când acesta tinde la infinit. Parametrul $\alpha$ va fi un fel de factor de normalizare. Pentru simplificare, vom alege:

$$\alpha = \frac{1}{N} \tag{1.66}$$

Regula de ajustare a ponderilor sinaptice devine:

$$w_{ij}(t) = w_{ij}(t-1) + \frac{1}{N}\left[x_i(t) - \sum_{j=1}^{N} w_{ij}(t-1)\cdot x_j(t)\right]\cdot x_j(t) \tag{1.67}$$

Această regulă de ajustare este strict locală deoarece nu face apel la cunoaşterea matricii **W** pentru fiecare element (i, j). Algoritmul de ajustare a ponderilor va fi:

1. Iniţializare:

   **x**(0) = 0
   **W**(0) = 0

2. Aplicarea unui prototip **x**(t) şi actualizarea matricii **W**:

$$w_{ij}(t) = w_{ij}(t-1) + \frac{1}{N}\left[x_i(t) - \sum_{j=1}^{N} w_{ij}(t-1) \cdot x_j(t)\right] \cdot x_j(t)$$

3. Revenire la pasul 2.

Trebuie să subliniem că, în cadrul acestui algoritm trebuie aplicat de mai multe ori ansamblul de vectori prototip pentru ca eroarea să scadă pe durata acestor prezentări succesive.

### 1.7.9. Aplicațiile rețelelor recurente

Rețeaua *Hopfield* este larg utilizată pentru implementarea funcției de memorie asociativă. Vom prelua ca exemplu problema recunoașterii caracterelor. Două prototipuri sunt prezentate în figura 1.38. Fiecărui pixel (reprezentat printr-un pătrățel) îi este asociat un neuron. Rețeaua va conține 12 x 16 = 192 neuroni.

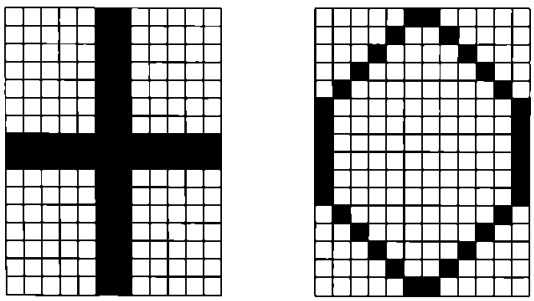


Fig. 1.38. Vectorii prototip

Prin convenție, neuronii activi vor avea ieşirea la valoarea +1 şi cei inactivi la valoarea -1. Pentru fiecare vector prototip, se poate reprezenta starea rețelei printr-un grafic 3D, ca în figura 1.39.

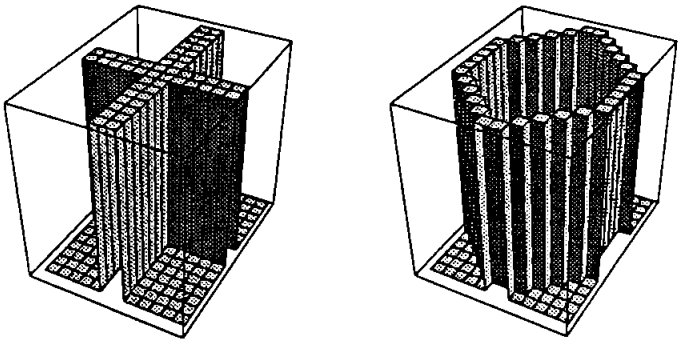


Fig. 1.39. Activitatea rețelei pentru fiecare prototip ce trebuie învățat

Aceste 2 prototipuri pot fi învăţate utilizând regula lui *Hebb*. Apoi, în timpul fazei normale de lucru (recunoaşterea caracterelor), reţeaua va fi exploatată ca o memorie asociativă. Se prezintă reţelei un caracter oarecare (denaturat), ca de exemplu cel reprezentat în figura 1.40. Prezentarea unui caracter constă în forţarea stării neuronilor pe valorile +1 şi respectiv -1 impuse de vectorul de intrare (caracterul) respectiv.

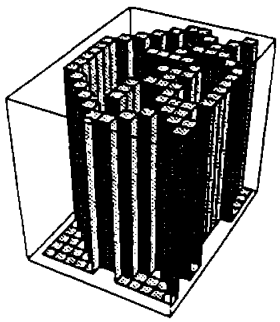


Fig. 1.40. Starea iniţială a reţelei pentru caracterul de recunoscut

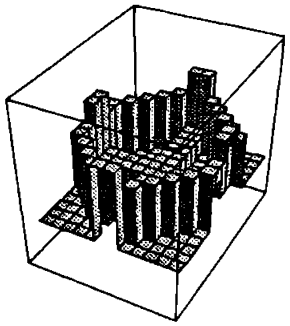


Fig. 1.41. Activitatea reţelei după convergenţă

Se lasă apoi reţeaua să evolueze liber spre o stare stabilă; pe parcursul evoluţiei starea neuronilor a fost actualizată utilizând procedura paralelă de actualizare. La convergenţă se va obţine graficul de activitate prezentat în figura 1.41. Să notăm că în acest grafic sunt reprezentate produsul scalar dintre vectorul de stare al neuronilor şi matricea sinaptică **W** obţinută în urma fazei de învăţare. Activitatea fiecărui neuron va fi reprezentată deci de o valoare reală (analogică).

Se aplică apoi funcţia de transfer (de exemplu funcţia sign(x)). Astfel, dacă activitatea unui neuron este pozitivă, ieşirea lui va fi forţată la +1, în caz contrar la -1. Nivelul valorii de prag este reprezentat în figura 1.42. În final, după aplicarea funcţiei sign(x) asupra potenţialelor tuturor neuronilor se va obţine noua stare a reţelei, prezentată în figura 1.43.

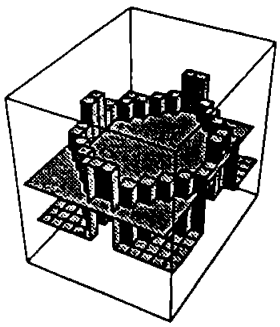


Fig. 1.42. Reprezentarea valorii de prag pe graficul de activitate

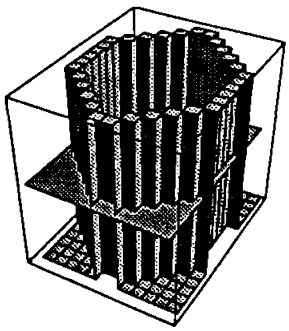


Fig. 1.43. Starea reţelei după aplicarea funcţiei de transfer

Să notăm că reţeaua a recunoscut prototipul 2; ea a asociat modelului denaturat aplicat la intrare prototipul cel mai asemănător (memorie asociativă). Această structură de reţea complet

interconectată poate fi de asemenea utilizată pentru rezolvarea oricărei probleme de optimizare. *Hopfield* şi *Tank* au utilizat-o pentru rezolvarea problemei agentului comercial. Principiul de lucru impune determinarea unei funcţii de energie adecvate care va plasa soluţiile problemei de rezolvat în minimele locale aferente acestei funcţii.

### 1.7.10. Concluzii

Modelul *Hopfield* are la bază o reţea recurentă şi exploatează proprietăţile naturale ale acesteia de convergenţă spre un punct de stabilitate (stare sau ciclu de stări). Acest model suportă mai multe reguli de învăţare cum ar fi: regula lui *Hebb*, regula proiecţiei şi proiecţiei Delta. Lista nu este exhaustivă şi se poate afirma că există atâtea reguli de învăţare câte probleme sunt de soluţionat. Reţeaua *Hopfield* şi dinamica sa are limitări deja cunoscute:

- capacitatea de învăţare după regula lui *Hebb* este redusă. După [AMI 85], presupunând că numărul de neuroni N este foarte mare, numărul maxim de prototipuri corect recunoscute este sub 0,138 N. În aceste condiţii, distorsiunea admisă este de ordinul a 3% (estimată prin produsul scalar dintre vectorul prototip normat şi vectorul de recunoscut). Dacă numărul de prototipuri depăşeşte limita de 0,138 N, altfel spus dacă se încearcă "supraîncărcarea" memoriei în timpul învăţării, toate prototipurile învăţate anterior se pierd (fenomenul de uitare). Totuşi, utilizând regula proiecţiei, capacitatea de învăţare creşte şi poate atinge circa 40% din numărul de neuroni.
- informaţiile nu pot fi stocate sub formă brută în memorie. Trebuie realizată o pre - procesare a datelor în scopul ortogonalizării, dacă se vizează atingerea unei capacităţi de învăţare apropiată de posibilităţile teoretice ale reţelei.
- trebuie evitate erorile, cum ar fi recunoaşterea falsă care constă în clasarea greşită a unei configuraţii de intrare.

## 1.8. Algoritmul de auto-organizare al lui Kohonen

### 1.8.1. Observaţii fiziologice şi funcţionale

Reţelele biologice reale au o structură organizatorică ordonată care depinde adesea de activitatea pe care sunt specializate. Se constată că, pe cortex, zonele receptoare sunt ordonate într-o manieră identică cu cea a unităţilor organului senzorial asociat. În plus, se remarcă o corespondenţă topologică între aceste zone: de exemplu, două zone vecine din cortexul vizual sunt corespondentele a 2 zone vecine de pe retină [HUB 77]. Se poate extinde această observaţie şi asupra cortexului auditiv care prezintă un asemenea aranjament topologic al neuronilor încât, frecvenţe vecine sunt detectate de către neuroni vecini.

Mai multe experienţe au pus în evidenţă că această organizare nu este genetică şi că reprezintă rezultatul unui proces de învăţare de tip hebbian.

Funcţia esenţială a algoritmului lui *Kohonen* este de a realiza o cuantificare a unui spaţiu de stimuli continuu sub forma unui spaţiu de ieşire discret de dimensiune inferioară sau egală cu cea a spaţiului de stimuli. Se presupune mai întâi că, în spaţiul de stimuli, există o relaţie de vecinătate, de exemplu cea definită prin distanţa euclidiană. Se poate spune în acest caz că spaţiul de intrare este structurat. Apoi, în spaţiul de ieşire constituit din N neuroni, se fixează o relaţie de vecinătate de tipul celei descrise prin figura 1.45.

Scopul algoritmului este de a găsi o funcţie $\phi$ care, fiecărui element din spaţiul de stimuli face să-i corespundă un neuron, ales după valoarea activităţii (ieşirii) sale, într-o asemenea manieră încât pentru doi stimuli vecini cele 2 imagini obţinute în spaţiul de ieşire vor fi de asemenea vecine.

Dacă această condiţie este îndeplinită pentru toate cuplurile de stimuli, vom spune că funcţia $\phi$ respectă topologia.

### 1.8.2. Principiul algoritmului

Pe baza simbolismului adoptat, algoritmul de auto-organizare al lui *Kohonen* se reprezintă ca în figura 1.44.

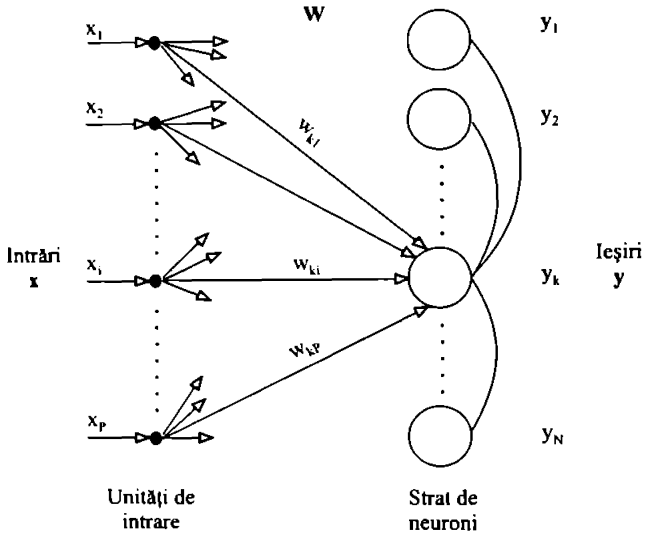



Fig. 1.44. Structura reţelei *Kohonen* (*Self Organizing Feature Map* - SOFM)

Reţeaua este constituită din 2 straturi de interconexiuni şi un singur strat de neuroni. Primul strat de interconexiuni este aşa-numitul strat plastic, deoarece acesta are atribuite ponderile sinaptice $w_{ij}$ şi, drept consecinţă, va suporta ajustarea acestor ponderi. Al doilea strat de interconexiuni este fix şi joacă rolul unei reţele competitive.

### 1.8.3. Funcţia primului strat

Primul strat de interconexiuni este utilizat pentru a determina activitatea fiecărui neuron de ieşire, în funcţie de vectorul stimul aplicat la intrare. Dacă notăm:

N - numărul de neuroni din stratul de ieşire
P - numărul de intrări în fiecare neuron
$\mathbf{x}$ - vectorul de intrare (de stimuli), $\mathbf{x} = (x_1, x_2, \ldots, x_P)^T$
$\mathbf{W}$ - matricea ponderilor sinaptice de dimensiune N x P

$$\mathbf{W} = \begin{Vmatrix} w_{11} & w_{12} & & w_{1P} \\ w_{21} & w_{22} & & w_{2P} \\ \cdots & & & \cdots \\ w_{N1} & w_{N2} & \cdots & w_{NP} \end{Vmatrix}$$

Fiecare linie a matricii **W** formează un vector cu P componente notat $\mathbf{w}_k$; k=1, 2, ..., N. Vectorul $\mathbf{w}_k$ conţine ponderile sinaptice aferente conexiunilor de intrare în neuronul k.

**y** - vectorul de activitate al neuronilor (vectorul de ieşire), $\mathbf{y} = (y_1, y_2, \ldots, y_N)^T$

atunci:

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$$

Am regăsit deci modelul neuronului *McCulloch - Pitts*.
Considerând că ponderile (vectorii $\mathbf{w}_k$) şi intrările (vectorii **x**) sunt normate, produsul scalar $\mathbf{w}_k \cdot \mathbf{x}$ va fi maxim când cei doi vectori sunt coliniari. Trebuie să subliniem importanţa normei în acest calcul, deoarece principalul scop este de a găsi o măsură a gradului de apropiere sau proximitate dintre vectorii de ponderi sinaptice şi vectorul stimul aplicat la intrare (*nearest neighbour classifier*). Dacă vectorii nu sunt corect normaţi, produsul scalar nu va reprezenta o măsură corectă a gradului de proximitate. Trebuie deci să se utilizeze o normalizare adecvată pentru a compara direcţiile vectorilor fără a pierde norma acestora.

### 1.8.4. Funcţia celui de-al doilea strat

Al doilea strat de interconexiuni este o reţea de inhibiţii laterale recurente. Fiecare neuron va fi în relaţie cu neuronii vecini pe baza unei funcţii de interacţiune de tipul celei descrise în figura 1.45.

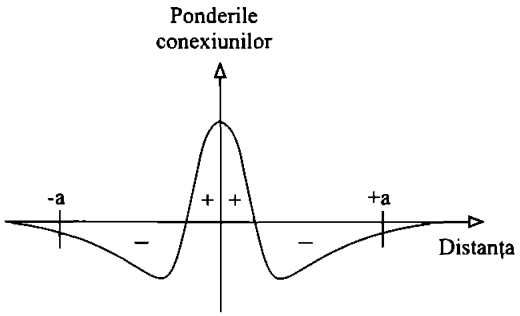



Fig. 1.45. Funcţia de interacţiune denumită "pălărie de mexican"

Vom considera iniţial o vecinătate monodimensională fixă în timp. Fiecare neuron k este în relaţie cu vecinii săi (k - a, ..., k - 1, k + 1, ..., k + a) şi fiecare dintre aceştia participă la elaborarea potenţialului său prin adăugarea propriului potenţial ponderat pe baza funcţiei "pălărie de mexican". Efectul acestor interacţiuni este de a crea, în ansamblul de neuroni de ieşire, o zonă de activitate stabilă în jurul neuronului care răspunde preferenţial la stimulul de intrare aplicat. Această zonă va desemna foarte precis neuronii ale căror ponderi sinaptice vor fi ajustate pe parcursul fazei de învăţare. Funcţia "pălărie de mexican" poate fi extinsă la cazul bidimensional.

După stabilizarea calculelor, se va determina care este componenta maximală dintre cele N componente ale vectorului de ieşire **y**.

$$y_k^* = \max_k y_k; \qquad k = 1, 2, \ldots, N \qquad (1.68)$$

Se obţine astfel vectorul de ponderi sinaptice $\mathbf{w}_k$ (asociat neuronului $y_k^*$) care este cel mai apropiat de vectorul stimul **x**. Pentru a analiza acest comportament vom reprezenta vectorii **x** şi $\mathbf{w}_k$ (k = 1, 2, ..., N) sub formă geometrică. Pentru aceasta vom considera că vectorii de ponderi sunt normalizaţi ($\| \mathbf{w}_k \| = 1$).

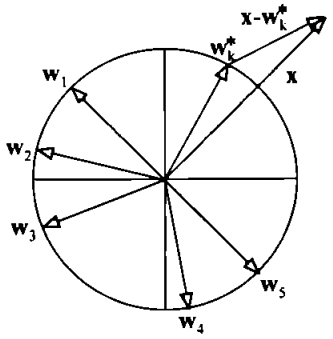



Fig. 1.46. Căutarea proximităţii relative dintre vectorul **x** şi vectorii $\mathbf{w}_k$

După cum se poate constata din reprezentarea geometrică prezentată în figura 1.46, găsirea produsului scalar maxim dintre vectorii **x** şi $\mathbf{w}_k$ se reduce la determinarea distanţei euclidiene minime dintre vectorul **x** şi vectorii $\mathbf{w}_k$ (dacă vectorii $\mathbf{w}_k$ sunt normaţi). După efectuarea acestei operaţii şi determinarea neuronului câştigător $y_k^*$, se vor ajusta ponderile vectorului $\mathbf{w}_k^*$ (asociat neuronului $y_k^*$), precum şi ponderile care aparţin neuronilor din vecinătatea sa (funcţia "pălărie de mexican"), utilizând următoarea regulă de ajustare:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha \cdot y_k(\mathbf{x} - \mathbf{w}_k(t)) \quad \text{cu } 0 < \alpha < 1 \tag{1.69}$$

Pentru neuronii care nu se află în vecinătatea neuronului $y_k^*$, ponderile rămân nemodificate:

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) \tag{1.70}$$

Ecuaţia (1.69) controlează gradul de ajustare a ponderilor sinaptice aferente celui mai activ neuron. Relaţia menţionată arată că se ia o fracţiune ($\alpha$) din distanţa euclidiană dintre **x** şi $\mathbf{w}_k^*$, $(\mathbf{x} - \mathbf{w}_k^*)$, şi se "apropie" vectorul $\mathbf{w}_k^*$ de **x** precum şi toţi ceilalţi neuroni din vecinătatea neuronului $y_k^*$. Această operaţie este ilustrată în figura 1.47.

Să notăm că, în baza ecuaţiei precedente, nivelul ajustării ponderilor $\mathbf{w}_k$ depinde în egală măsură si de valoarea de activare a neuronului $y_k$. Deci, cu cât răspunsul neuronului este mai puternic, cu atât ajustarea ponderilor va fi mai semnificativă. În sfârşit, să notăm că ponderile care au fost ajustate nu mai sunt normalizate; ele trebuie normalizate pentru a fi pregătite pentru următoarea operaţie de comparare.

Rezultatul acestui algoritm este că "densitatea spaţiului de stimuli converge spre o imagine discretizată astfel încât elementele acestei imagini respectă topologia spaţiului de intrare" [COT 89].

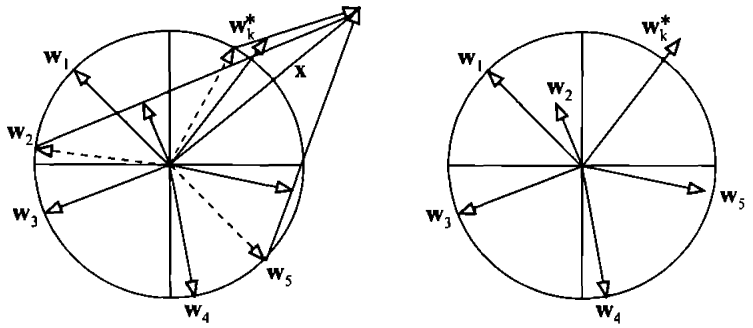



Fig. 1.47. Efectul ajustării ponderilor sinaptice. Vectorii $w_1$ şi $w_5$ sunt vectorii aferenţi neuronilor $y_1$ şi respectiv $y_5$ din vecinătatea învingătorului $y_k^*$.

### 1.8.5. Algoritmul SOFM simplificat

Una dintre primele simplificări aduse algoritmului se referă la metoda de evaluare a ieşirii celei mai active. Această măsură a activităţii se elaborează cu ajutorul produsului scalar dintre vectorul de stimuli şi vectorii de ponderi sinaptice. Cum am putut deja constata, este posibilă înlocuirea acestei operaţii cu una de calcul a distanţelor euclidiene. Deci, determinarea celui mai activ neuron se poate face astfel:

$$\|\mathbf{x} - \mathbf{w}_k^*\| = \min_k \|\mathbf{x} - \mathbf{w}_k\| \qquad (1.71)$$

Etapa de ajustare a ponderilor sinaptice aferente stratului plastic poate fi şi ea modificată. Din moment ce nu se mai calculează produsele scalare pentru determinarea celui mai activ neuron, valoarea $y_k$ poate fi fixată arbitrar. Se poate considera că, pentru neuronul învingător, $y_k^* = 1$ şi pentru toţi ceilalţi neuroni $y_k = 0$. Nu se va selecta astfel decât un singur neuron activ.

În sfârşit, procesul generat de către stratul de inhibiţii laterale are ca efect crearea unei "bule" de activitate în jurul neuronului învingător. Acest fenomen este controlat prin forma funcţiei "pălărie de mexican". Se poate defini o vecinătate în jurul neuronului învingător prin calcul de distanţe euclidiene şi se vor ajusta ponderile sinaptice aferente neuronilor care aparţin acestei vecinătăţi.

Dacă notăm cu V vecinătatea din jurul neuronului învingător, algoritmul de ajustare a ponderilor se rescrie:

- pentru $|k - k^*| \le V$

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t) + \alpha(\mathbf{x} - \mathbf{w}_k(t)) \qquad 0 < \alpha < 1 \qquad (1.72)$$

apoi se efectuează operaţia de normalizare:

$$\mathbf{w}_k = \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|} \qquad (1.73)$$

- pentru $|k - k^*| > V$

$$\mathbf{w}_k(t+1) = \mathbf{w}_k(t)$$

Operaţia de normalizare poate fi eliminată cu condiţia ca **x** să varieze într-un domeniu convex mărginit în care sunt iniţializate ponderile $w_k$. Într-o astfel de ajustare va fi aplicată norma euclidiană, vectorii ponderilor sinaptice fiind repartizaţi pe un cerc de rază egală cu o unitate.

Să notăm că vecinătatea din jurul neuronului învingător poate fi o funcţie de timp. Această funcţie poate lua mai multe forme care depind esenţial de topologia planului (stratului) de ieşire. În figura 1.48 se arată evoluţia unei vecinătăţi, în funcţie de timp, potrivit cu o topologie liniară şi respectiv planară a stratului de ieşire.

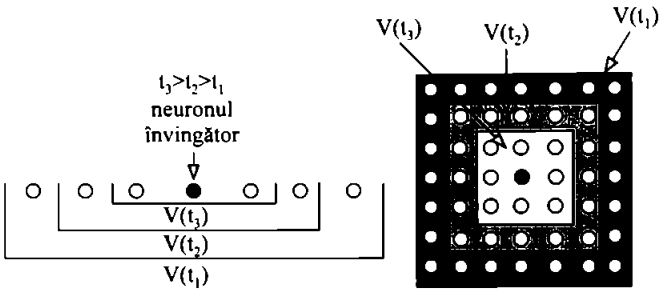



Fig. 1.48. Evoluţia unei vecinătăţi în jurul neuronului învingător (1D şi 2D)

Algoritmul SOFM depinde în faza de învăţare de 2 parametri:

V - vecinătatea neuronului învingător, care poate fi o funcţie de timp

$\alpha$ - constanta de învăţare

Aceşti doi parametri determină comportamentul reţelei. Trebuie să menţionăm că algoritmul prezintă uneori şi aberaţii funcţionale care pot fi generate de o alegere incorectă a celor 2 parametri. Aceasta denotă faptul că algoritmii sunt complecşi şi punerea lor la punct necesită în continuare studii teoretice cu privire la strategiile de optimizare a valorilor parametrilor. Totuşi, anumite principii au fost deja clarificate prin simulări numerice şi studii teoretice. Algoritmul SOFM poate fi divizat în 2 etape: o organizare grosieră a reţelei urmată de o convergenţă fină spre soluţie. Parametrii $\alpha$ şi V vor avea valori mari pe parcursul primei faze ($\alpha$ aproape de 0,5 şi V extins la toată suprafaţa reţelei). Apoi, pe parcursul fazei de convergenţă, $\alpha$ descreşte spre 0 şi V spre o vecinătate egală cu 0. După *Kohonen* [KOH 88a], $\alpha(t)$ este o funcţie descrescătoare în timp care respectă următoarele condiţii:

$$\sum_{s=0}^{\infty}\alpha(s)=\infty \quad \text{şi} \quad \sum_{s=0}^{\infty}\alpha^2(s)<\infty \tag{1.74}$$

Alegerea valorilor parametrilor $\alpha$ şi V va avea consecinţe majore asupra vitezei de auto - organizare a reţelei [COT 89].

### 1.8.6. Exemple de utilizare a algoritmului SOFM

Considerăm un spaţiu de intrare constituit dintr-un plan în care s-au trasat aleator puncte. Coordonatele fiecărui punct ($x_1$, $x_2$) vor fi intrările în reţeaua SOFM. În fig. 1.49 se prezintă punctele trasate în plan (spaţiul de intrare) precum şi dispozitivul de citire a acestor puncte.


Fig. 1.49. a) Distribuţia punctelor în plan; b) Dispozitivul de citire

Braţul robotului care citeşte punctele este constituit dintr-un braţ telescopic care generează coordonatele polare aferente fiecărui punct citit. Vom face astfel un anumit număr de citiri pe care le vom aplica pe intrarea reţelei SOFM.

Reţeaua este constituită din 2 intrări şi 100 neuroni aşa cum indică figura 1.50.

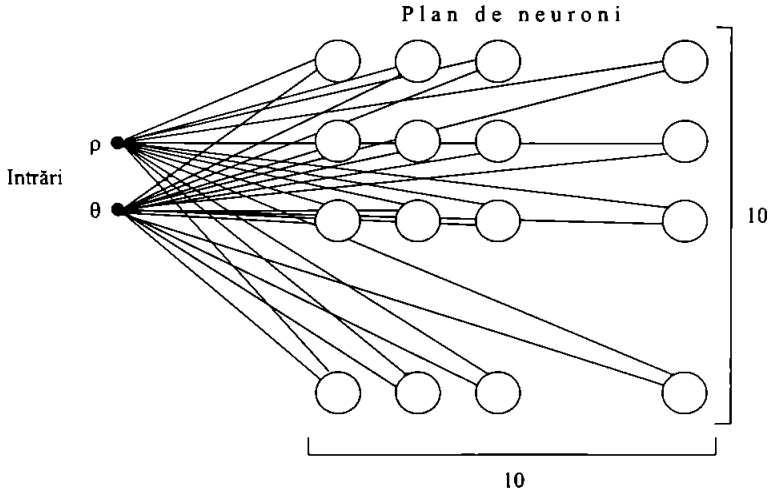



Fig. 1.50. Structura reţelei SOFM

Se iniţializează matricea **W** (ponderile sinaptice aferente stratului de interconexiuni plastice) cu valori aleatoare şi se va urmări evoluţia acestora pe parcursul procesului de ajustare (învăţare).

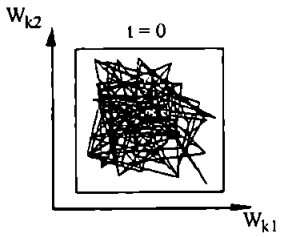



Fig. 1.51. Repartiţia iniţială a neuronilor în spaţiul ponderilor

Trebuie să subliniem că termenul de auto - organizare nu înseamnă o migrare a neuronilor în cadrul reţelei. Este vorba de reprezentarea aleasă care indică un aranjament sub formă de plasă sau grilă regulată. Această reprezentare, aşa cum arată figura 1.52, descrie în plan "poziţia" fiecărui neuron prin cuplul (vectorul) său de ponderi sinaptice. Liniile dintre neuroni rezultă dintr-un etichetaj şi indică cei 4 vecini imediaţi ai fiecărui neuron. Deoarece neuronii sunt aranjaţi într-un plan, am considerat că fiecare se află în relaţie cu cei 4 vecini ai săi.

Fenomenul de auto - organizare care se dezvoltă pe parcursul procesului de învăţare reflectă 2 componente:

- valorile ponderilor sinaptice tind să acopere planul în care sunt reprezentate urmărind forma şi densitatea spaţiului de intrare.
- valorile ponderilor se ordonează în planul de ieşire. Această ordine depinde de relaţia care există între elementele spaţiului de intrare.

În figura 1.52 se reprezintă evoluţia vectorilor sinaptici $\mathbf{w}_k$ în timp (pe parcursul procesului de învăţare) unind cu segmente de dreaptă punctele care în spaţiul de ieşire corespund la neuroni vecini.

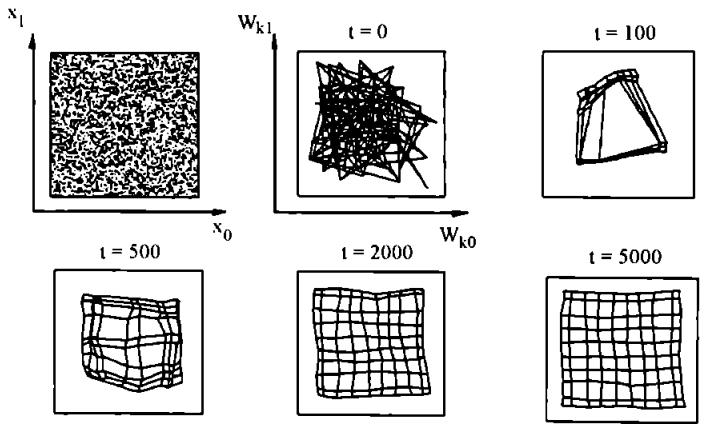



Fig. 1.52. Evoluţia organizării în timp

Se observă convergenţa procesului de auto - organizare care conduce la o imagine discretizată a densităţii spaţiului de stimuli. În exemplul prezentat, densitatea spaţiului de stimuli este uniformă (poziţionare aleatoare a punctelor în spaţiul de intrare). Imaginea discretizată va conţine o grilă (plasă) regulată. Să remarcăm numărul mare de iteraţii necesar pentru derularea procesului de auto - organizare. Este unul dintre principalele inconveniente ale acestui algoritm care are o viteză de convergenţă redusă.

Reprezentarea aleasă în figura 1.52 plasează neuronii în planul ponderilor sinaptice. Trebuie să înţelegem poziţia fiecărui neuron drept o reprezentare a vectorilor $\mathbf{w}_k$, aşa cum se arată în figura 1.53.

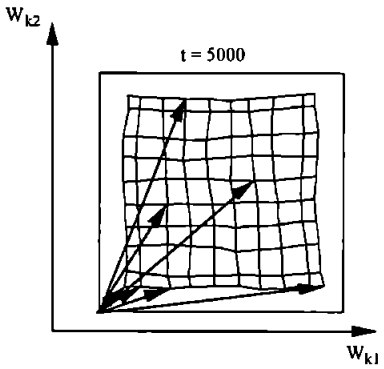



Fig. 1.53 Reprezentarea vectorilor sinaptici în spaţiul bidimensional

Această reprezentare presupune că vectorii $\mathbf{w}_k$ nu sunt normaţi, ceea ce înseamnă că, pentru detectarea gradului de proximitate dintre vectorii stimuli şi vectorii $\mathbf{w}_k$, s-au utilizat distanţele euclidiene. Dacă se doreşte utilizarea metodei produselor scalare, trebuie realizată normalizarea vectorilor $\mathbf{w}_k$ într-un spaţiu de dimensiune superioară, şi apoi calcularea produselor scalare în acest spaţiu, pentru a nu modifica norma vectorilor $\mathbf{w}_k$.

După faza de învăţare, exploatarea reţelei se poate face în conformitate cu procesul descris în figura 1.54.


a) poziţiile succesive ale braţului robotului b) răspunsul reţelei SOFM

Fig. 1.54. Exemplu de exploatare a reţelei organizate

Se constată că mişcările executate de braţul robotului în spaţiul de intrare sunt reproduse pe planul de neuroni, cu o anumită simetrie, marcată pe grila definită de caroiajul virtual stabilit în faza de învăţare. Reproducerea nu este identică ci doar simetrică datorită faptului că, în timpul învăţării, neuronii nu au avut repere absolute, ci doar relative constituite prin relaţia de proximitate dintre stimulii spaţiului de intrare.

Dintre reprezentările utilizate pentru a explica funcţionarea algoritmului SOFM, amintim "mozaicul lui *Voronoi*", care defineşte o zonă de referinţă în jurul fiecărei ponderi; această zonă corespunde punctelor din spaţiul de intrare care excită preferenţial acelaşi neuron. Este zonă de stimuli de intrare care aparţine aceluiaşi agregat. Figura 1.55 prezintă un exemplu:

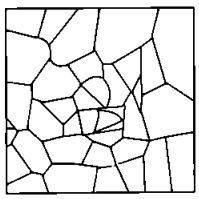


Fig. 1.55. Mozaicul lui *Voronoi*

### 1.8.7. Concluzii

După cum am constatat, algoritmul SOFM reprezintă un instrument foarte puternic pentru pre - tratamentul datelor destinate unei viitoare clasificări. Algoritmul elaborează o reprezentare a datelor agregând vectorii vecini care aparţin aceleiaşi clase şi separând agregatele care corespund unor clase diferite. El este de altfel utilizat pe post de strat de pre - tratare în algoritmul *Counter propagation* [HEC 87].

Aplicaţiile bazate pe reţeaua SOFM sunt foarte numeroase. Putem cita, spre exemplificare, analiza unei mari cantităţi de date tehnologice aferente unei linii de fabricaţie circuite VLSI [TRY 89], crearea unui spectru topografic pentru analiza vorbirii [KOH 88b], controlul unui braţ de robot într-un câmp prevăzut cu obstacole [GRA 88], [RUE 90] sau a unui robot mobil într-un labirint [SOR 89].

## 1.9. Modele avansate

Modelele descrise în capitolele precedente sunt deja modele clasice. O serie de alte modele încep să fie utilizate în diverse aplicaţii. Dintre aceste modele citãm:

- Modelul *Hérault - Jutten* care prezintă un interes deosebit pentru procesarea de semnal (problema separării surselor) [JUT 87], [JUT 88], [COM 90].
- Modelul ART (*Adaptive Resonance Theory*) elaborat de *Grossbeerg* [GRO 87] şi divizat în 2 clase: ART 1 cu valori de intrare binare şi ART 2 cu valori de intrare analogice. Acest model serveşte la formarea de categorii de date cu ajutorul unui parametru de vigilenţă care defineşte fineţea categoriilor.
- Modelul RCE (*Restricted Coulomb Energy*) elaborat de *Cooper* (şi alţii), utilizează învăţarea supervizată şi este destinat problemelor de clasificare.
- Modelul LVQ (*Learning Vector Quantization*) [KOH 86] care este un clasificator şi care se bazează pe un algoritm de învăţare semi-supervizată [GUY 90]. Este unul dintre cei mai performanţi algoritmi de clasificare. În mod particular se adaptează bine problemelor care revendică un set de antrenament foarte amplu [BEN 90].
- Modelul *Cognitron* al lui *Fukushima* [FUK 75] construit pe o structură multistrat cu o tehnică de auto - organizare pentru învăţare. Se aplică în mod esenţial la recunoaşterea caracterelor.
- Modelul reţelelor selective, propus de *Edelman* [EDE 82] care introduce noţiunea de evoluţie a unui sistem (în sensul selecţiei naturale), noţiune absentă la majoritatea celorlalte modele. Acest model stă la baza seriei de maşini DARWIN.

Mai multe variante au fost de asemenea dezvoltate, pornind de la modelele clasice:

- reţele multistrat recurente cu învăţare pe baza algoritmului EBP [ALM 89].
- reţele TDNN (*Time Delay Neural Networks*) [WAI 89] bazate pe perceptroni multistrat şi care învaţă după algoritmul EBP. O caracteristică a acestor reţele este că neuronul recepţionează simultan mai multe eşantioane întârziate aferente semnalelor de intrare.
- reţeaua BAM (*Bidirectional Associative Memory*) [KOS 87] construită pe baza a 2 reţele *Hopfield* rebuclate.
- reţeaua *Counter propagation* [HEC 87] construită dintr-o reţea *Kohonen* urmată de un perceptron.

Această listă incompletă indică faptul că domeniul este în plină ascensiune. Teoriile sunt încă insuficient elaborate pentru a satisface toate problemele reale. O abordare mai "experimentală" survine adesea pentru a ascunde aceste carenţe teoretice, dar reclamă numeroase clarificări pentru aplicaţiile concrete.

Algoritmii de învăţare pot fi grupaţi în 4 categorii:

- algoritmi care generează ponderi fixe
- algoritmi de învăţare supervizată
- algoritmi de învăţare semi - supervizată
- algoritmi de învăţare nesupervizată

Se defineşte învăţarea supervizată acea tehnică de învăţare care minimizează o funcţie de cost, care este în general un criteriu de eroare pătratică. Minimizarea este realizată printr-o metodă de coborâre (pe suprafaţa de eroare) pe direcţia gradientului negativ sau prin metoda gradientului stohastic. Cu această metodă, se cunoaşte exact vectorul de eroare, care apare ca diferenţa între ieşirea dorită şi ieşirea reală.

Învăţarea semi - supervizată nu operează asupra unui vector de eroare ci asupra unui scalar care indică calitatea răspunsului. Este vorba de un coeficient de similaritate între ieşirea dorită şi cea reală care ne oferă o indicaţie referitoare la mărimea erorii comise.

Învăţarea nesupervizată nu face apel la nici o ieşire dorită. Ajustarea ponderilor nu depinde decât de valorile de pe intrarea reţelei şi de ponderile sinaptice calculate şi actualizate până în momentul respectiv.

În tabelul 1.1. redăm câteva exemple de aplicaţii posibile în funcţie de tipul reţelei; acest tabel nu include evident o tratare exhaustivă.

| Aplicaţie | Tip reţea | | | |
|---|---|---|---|---|
| | Fără învăţare | Înv. supervizată | Înv. semi-supervizată | Înv. nesupervizată |
| Procesarea vorbirii sau a imaginilor | Inhibiţii laterale recurente | TDNN<br>MLP + *Back Propagation*<br>MLP rebuclată | LVQ | SOFM<br>*Hérault - Jutten* |
| Conducere de proces | | MLP + *Back Propagation*<br>MLP rebuclată<br>*Adaline* | *Counter propagation*<br>*Reinforcement learning* | ART 3 |
| Optimizare | *Hopfield - Tank* | *Hopfield* | *Reinforcement learning* | SOFM |
| Analiză de date<br>Clasificare<br>Extragere de caracteristici (compresie date) | Reţele ad - hoc | MLP + *Back Propagation* | LVQ<br>*Neocognitron* | RCE<br>SOFM |
| Procesare de semnal | | TDNN<br>*Adaline* | | *Hérault - Jutten* |
| Memorie asociativă | *Interactive Activation and Competition* | *Hopfield*<br>BAM | | |

Tabelul 1.1. Aplicaţii ale RN

După cum se constată din tabelul 1.1, există încă o "latură artistică" în ceea ce priveşte aplicaţiile RN. Care este reţeaua cea mai bine adaptată unei aplicaţii date, care sunt parametrii optimali

(mărime, constantă de învăţare, etc.), care este tipul de învăţare, sunt tot atâtea întrebări la care trebuie să răspundă perspicacitatea inginerului. În general, se poate găsi întotdeauna o RN capabilă să rezolve o problemă dată. Dar, rămâne de văzut dacă soluţia neuronală este mai bună decât o soluţie clasică, dacă soluţia neuronală este implementabilă pe calculatoarele disponibile şi dacă rezultatele obţinute pe cale neuronală au o semnificaţie corectă.

Cu modelele actuale, inginerul trebuie să integreze o parte din ştiinţa sa în reţeaua neuronală. Alegerile sale sunt în mod necesar ghidate de cunoaşterea apriorică a problemei tratate.

RN nu trebuie privită ca o cutie neagră care spionează un proces real pentru a-i extrage un model numeric general. Actualmente, conexionismul nu reprezintă decât un concept relativ nou, care face apel la o reprezentare distribuită, la un paralelism masiv, la noţiunea de învăţare şi nu de programare şi la o tratare fundamental numerică şi nu simbolică a informaţiei.

## 2. IMPLEMENTĂRILE VLSI

Calculatoarele actuale, inclusiv cele bazate pe structuri paralele, sunt construite în jurul unui număr mic de unităţi de calcul rapide, precise şi complexe, fiind perfect adaptate algoritmilor de calcul tradiţionali. Aceste maşini nu reprezintă un optim şi pentru algoritmii conexionişti care revendică alte aptitudini: precizia este o constrângere minoră, fiecare operaţie este simplă dar paralelismul execuţiei este esenţial. Pentru domeniul conexionist trebuie dezvoltate maşini care satisfac aceste cerinţe.

### 2.1. Introducere

În figura 2.1. se prezintă modelul neuronului. Acest model stă la baza tuturor reţelelor conexioniste prezentate în această lucrare.

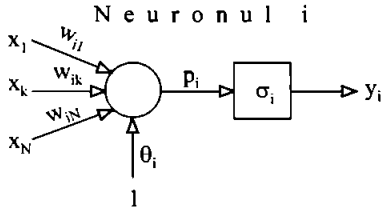



Fig. 2.1. Modelul neuronului

Notaţiile utilizate sunt:

$x_k$ - valoarea de activare a (ieşirea) neuronului k
$w_{ik}$ - valoarea ponderii sinaptice aferentă conexiunii dintre neuronul k (sursă) şi neuronul i (destinaţie)
$p_i$ - potenţialul neuronului i
$\sigma_i$ - funcţia de activare (transfer) aferentă neuronului i
$\theta_i$ - valoarea de prag asociată neuronului i
$y_i$ - valoarea de activare a (ieşirea) neuronului i

Problema construcţiei unei reţele conexioniste constă în definirea structurii acestei reţele şi a parametrilor aferenţi. Structura este definită prin conexiunile (locale, totale, parţiale, plastice sau nu etc.) dintre neuroni, iar parametrii sunt reprezentaţi de funcţia de transfer a neuronului (continuă, discontinuă, valorile de activare între 0 şi 1 sau între -1 şi +1 etc.) şi valorile de prag.

În funcţie de modelul arhitectural ales, se defineşte o regulă de învăţare. Aceasta descrie evoluţia coeficienţilor sinaptici în timp. Ecuaţia următoare reprezintă o formă generală a acestei reguli:

$$\Delta \mathbf{W} = \alpha(t)\cdot\sigma(\mathbf{u}(t))\cdot\tau(\mathbf{v}(t))^T\cdot\xi(t) \qquad (2.1)$$

**W** - matricea de ponderi sinaptice
$\alpha(t)$ - constantă (parametru) de învăţare, de regulă cuprins între 0 şi 1.
$\sigma$, $\tau$ - funcţii de transfer
**u**(t), **v**(t) - doi vectori.

Faza de recunoaştere face apel la un subansamblu de funcţii utilizate în faza de învăţare. În general, recunoaşterea (o fază mai simplă) se derulează astfel:

- elaborarea potenţialelor neuronilor

$$p_i(t) = \sum_{k=1}^{N} w_{ik} \cdot x_k + \theta_i \qquad (2.2)$$

- calculul valorilor de activare (ieşirilor):

$$y_i(t) = \sigma(p_i(t)) \qquad (2.3)$$

Deci, realizarea unei reţele conexioniste se reduce la implementarea eficientă a unor funcţii ca:

- produsul scalar, produsul matrice - vector şi produsul unui vector cu transpusul său;
- aplicarea unor funcţii neliniare (funcţiile de activare sau transfer).

### 2.2. Soluţii posibile pentru implementarea reţelelor neuronale

Există trei soluţii posibile pentru implementarea reţelelor neuronale:

- Implementări *software* care constau în elaborarea unor simulatoare care rulează pe calculatoare universale sau a unor biblioteci de funcţii standard cu ajutorul cărora se pot dezvolta aplicaţii proprii.
- Implementări *hardware* dezvoltate pe calculatoare universale. Acestea presupun adăugarea unor plăci specializate (acceleratoare) gestionate prin intermediul unor simulatoare adecvate.
- Implementări *hardware* speciale care vizează realizarea unor circuite VLSI specializate pe aplicaţii particulare.

Simulatoarele *software* reprezintă o soluţie ieftină dar care realizează performanţe modeste; performanţele sunt limitate de performanţele maşinii suport. În tabelul 2.1 sunt prezentate câteva dintre cele mai cunoscute simulatoare soft pentru RN, existente pe piaţă la începutul anului 1998.

| Nr. crt. | Simulator | Producător | Caracteristici / Comentarii |
|---|---|---|---|
| 1 | *Brain Maker* | *California Scientific Software* | Reţele *feedforward* multistrat (maxim 8 straturi), reţele recurente etc. |
| 2 | *SAS Neural Network Application* | *SAS Institute* | Perceptroni multistrat, counterpropagation, reţele *feedforward* multistrat etc. |
| 3 | *Neural Works* | *Neural Ware* | *Backpropagation*, ART-1, *Kohonen*, reţele probabilistice, reţele SOFM etc. |
| 4 | *MATLAB Neural Network Toolbox* | *Math Works* | Reţele *feedforward*, reţele asociative, reţele SOFM etc. |
| 5 | *Propagator* | *ARD Corporation* | *Backpropagation* cu maxim 5 straturi |
| 6 | *Neuralyst* | *Cheshire Engineering Corporation* | Algoritmul *backpropagation* |
| 7 | *Neu Fuz 4* | *National semiconductor Corporation* | Reţele Neuro - fuzzy |
| 8 | *Neuro Solution v3.0* | *Neuro Dimension* | Reţele de perceptroni multistrat, reţele |

| | | | |
|---|---|---|---|
| | | | *feedforward* generalizate, reţele SOFM, reţele recurente etc. |
| 9 | *Qnet For Windows v2.0* | *Vesta Services* | Sistem de dezvoltare pe 32 biţi pentru *Windows* NT, 95 şi 3.1. |
| 10 | *havBpNet*<br>*havFmNet*<br>*havBpNet* | *hav. Software* | Reţele *feedforward*, recurente, SOFM |
| 11 | *IBM Neural Network Utility* | *IBM* | Modele multiple de RN, sisteme *fuzzy* şi sisteme combinate: *fuzzy* - RN |
| 12 | *Neuro Genetic Optimizer NGO v2.0* | *BioComp Systems* | Utilizează algoritmi genetici pentru optimizarea intrărilor şi structurii RN |
| 13 | *OWL Neural Network Library* | *HyperLogic Corporation* | *Backpropagation*, DBAM (*Discrete Bidirectional Associative Memory*), ABAM (*Adaptive Bidirectional Associative Memory*) etc. |
| 14 | *Pattern Recognition Workbench Expo PRO* | *Unica Technologies* | *Backpropagation* şi alte arhitecturi de RN utilizate în recunoaşterea modelelor |

Tabelul 2.1. Simulatoare soft dedicate reţelelor neuronale

Plăcile acceleratoare specializate se implementează într-un sistem gazdă şi au la bază procesoare specializate (de exemplu procesoare de semnal) sau structuri multiprocesor special proiectate pentru implementarea modelelor conexioniste.

Implementările VLSI (*Very Large Scale Integration*) presupun realizarea unor circuite specializate dedicate modelelor conexioniste. Aceste circuite pot fi utilizate atât în construcţia plăcilor acceleratoare cât şi autonom (în aplicaţii particulare care revendică procesare în timp real).

Vom face în continuare o analiză a implementărilor VLSI, evidenţiind problematica pe care proiectantul trebuie să o rezolve.

### 2.3. Structuri candidate la integrare

Între modelele teoretice distingem RN monostrat şi RN multistrat, stratul făcând referinţă la un grup de neuroni. Această diferenţiere nu are influenţe profunde în termeni structurali. Este suficient să considerăm RN ca un ansamblu de neuroni între care există sau nu legături.

Diferenţele între diferitele arhitecturi de RN apar la nivelul matricii de interconexiuni (matricii de ponderi sinaptice **W**) care va fi mai mult sau mai puţin populată cu elemente nenule. Implementările VLSI vor putea eventual exploata proprietăţile matricii **W**. Câteva exemple de matrici de interconexiuni sunt prezentate în fig. 2.2.

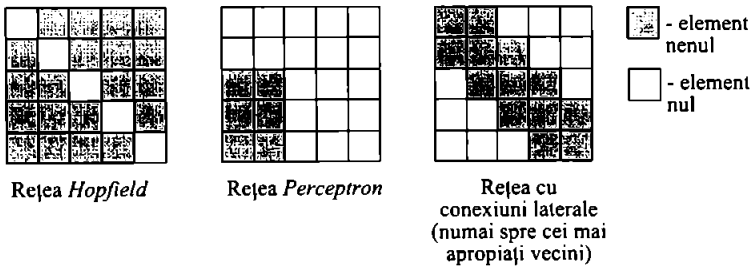



Fig. 2.2. Exemple de matrici de interconexiuni

Matricea **W** descrie structura reţelei de sinapse din cadrul RN. Poziţia neuronilor este total determinată prin matricea **W** şi prezenţa lor nu se justifică decât prin funcţia de transfer $\sigma$. Această constatare conduce la prima problemă (de decizie) pe care trebuie s-o rezolve proiectantul: va integra sinapse sau va integra neuroni? A integra sinapse înseamnă a integra în structura VLSI o matrice de intreconexiuni care va pune în relaţie neuronii exteriori circuitului. A integra neuroni înseamnă că blocurile de calcul a funcţiei de transfer vor fi prezente în circuitul VLSI şi, de asemenea, şi interconexiunile dintre aceste blocuri.

### 2.3.1. Integrarea neuronilor

Dacă interconexiunile sunt locale, limitate la cei mai apropiaţi vecini, soluţia de integrare a neuronilor este realistă. Dacă însă avem de-a face cu o reţea complet interconectată, cum este cazul reţelei *Hopfield*, integrarea neuronilor devine dificilă din două motive:

- interconexiunile dintre neuroni vor ocupa cea mai mare parte a suprafeţei de implantare;
- extensibilitatea reţelei va fi limitată de numărul de intrări / ieşiri.

Dacă dorim o RN complet interconectată, trebuie permis accesul fiecărui neuron din fiecare circuit spre toţi ceilalţi neuroni din toate celelalte circuite. Această constrângere nu va putea fi satisfăcută pentru RN cu un număr mare de neuroni decât în cazul aplicării unei tehnici sistolice (de comunicaţie).

### 2.3.2. Integrarea sinapselor

Integrarea sinapselor permite extensia reţelei şi nu pune probleme de intrare / ieşire (fig. 2.3).

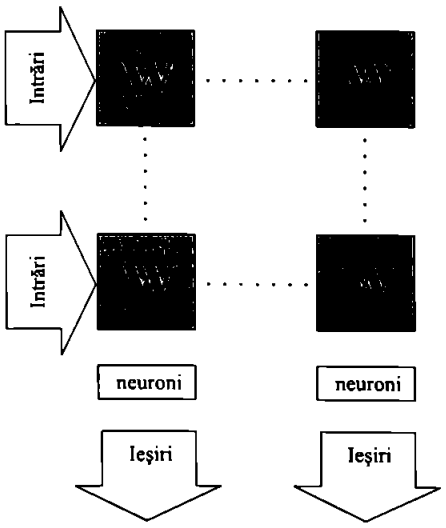



Fig. 2.3. Extensia matricii sinaptice

Dimensiunea RN poate fi mărită printr-un simplu pavaj rectangular de circuite sinaptice. RN multistrat vor putea fi uşor implementate prin conectarea în cascadă a circuitelor sinaptice, aşa cum indică figura 2.4.

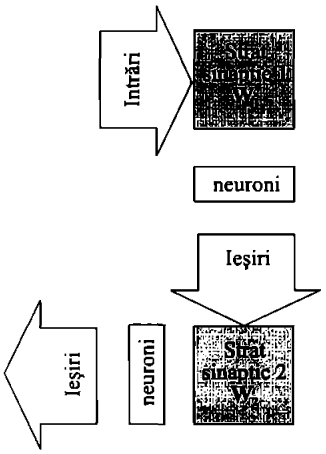



Fig. 2.4. RN multistrat

În capitolele 3, 4 şi 5 vom detalia proiectarea unei reţele sistolice (RS) destinată modelelor conexioniste şi care integrează sinapse.

Referitor la implementarea reţelelor neuronale pe structuri VLSI, pe plan mondial se fac cercetări pe mai multe direcţii. Implementările VLSI pot fi grupate în patru categorii:

- implementări analogice;
- implementări digitale;
- implementări hibride (digital - analogice);
- implementări opto - electronice.

### 2.4. Implementările VLSI analogice

Un exemplu de implementare VLSI analogică ar fi retina lui *Mead* şi *Mahowald* care mimează funcţionarea unei retine biologice [SHE 95]. Implementarea în discuţie reprezintă un model analogic al retinei mamiferelor superioare. Intrările în reţea sunt optice. Prototipul realizat integrează o retină de 48 x 48 pixeli ale cărei caracteristici de funcţionare se apropie de cele aferente celulelor bipolare din retina vertebratelor.

Este vorba de o reţea hexagonală realizată în tehnologie analogică. Circuitul se descompune în trei părţi care corespund unor funcţii specifice pe care le regăsim într-o retină biologică:

- fotoreceptorul care calculează logaritmul intensităţii luminoase pe care o recepţionează; este un dispozitiv cu control automat al amplificării;
- evaluatorul mediei temporale şi spaţiale de la ieşirea fotoreceptorului; acesta corespunde celulelor orizontale din retina biologică;
- evaluatorul diferenţei dintre semnalul fotoreceptorului şi semnalul furnizat de celulele orizontale; acesta corespunde celulelor bipolare din retina biologică.

Rezultatul va fi un circuit de detecţie a conturului prin diferenţiere spaţială, sau de detecţie a mişcării prin diferenţiere temporală. Vom descompune circuitul în elementele de bază pentru a preciza funcţionalitatea şi arhitectura acestora.

### 2.4.1. Fotoreceptorul

Fotoreceptorul transformă semnalul optic recepţionat de un fototranzistor într-un potenţial proporţional cu logaritmul intensităţii luminoase. În figura 2.5 se prezintă o implementare posibilă a fotoreceptorului.

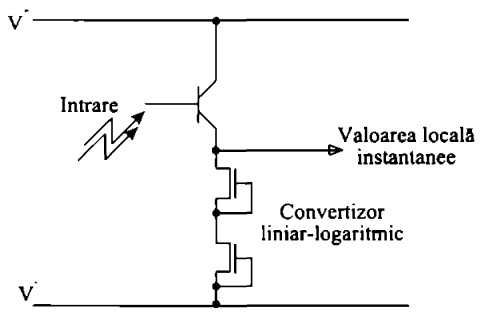



Fig. 2.5. Structura fotoreceptorului cu doi tranzistori

Acest dispozitiv preia curentul generat de elementul fotoreceptor şi îl injectează într-un element a cărui caracteristică curent - tensiune este logaritmică. Convertizorul liniar - logaritmic este realizat din două diode construite din tranzistoare MOS, conectate în serie. Să notăm că tranzistorul bipolar utilizat nu este fabricat; este un element parazit generat implicit prin fabricarea tranzistoarelor MOS.

### 2.4.2. Diferenţiatorul (celulele bipolare)

Pornind de la semnalul furnizat de către fotoreceptor, este posibilă percepţia unei mişcări într-o imagine. Există o mişcare atunci când se detectează o variaţie de intensitate luminoasă pe un pixel, adică atunci când de detectează local o derivată temporală nenulă. Pentru această detecţie se implementează un dispozitiv simplu, descris în figura 2.6, care elaborează rezultatul comparaţiei dintre valoarea instantanee a unui pixel şi valoarea aceluiaşi pixel convenabil întârziată.

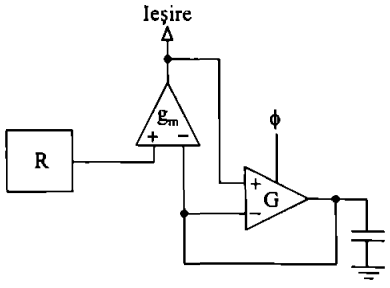



Fig. 2.6. Diferenţiatorul

În figura 2.6 R reprezintă fotoreceptorul. Transconductanţa G se comportă ca un tranzistor de trecere dar care permite eliminarea sarcinilor parazite generate în momentul comutaţiei. Acest dispozitiv furnizează valoarea locală a derivatei care nu ţine cont de valorile celulelor vecine (pixelilor vecini). Cu scopul de a construi un model cât mai apropiat de cel real, autorii au dezvoltat o reţea de rezistenţe care pune la dispoziţie în fiecare punct (pixel) media ponderată a valorilor vecine (pixelilor vecini).

### 2.4.3. Reţeaua rezistivă

Structura reţelei rezistive este descrisă în figura 2.7.

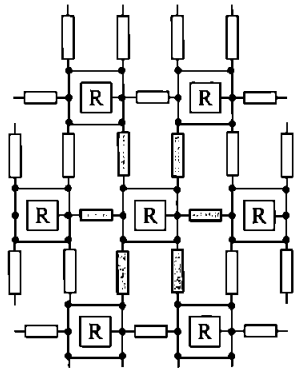



Fig. 2.7. Interconectarea receptorilor

Reţeaua hexagonală de rezistenţe furnizează (în fiecare punct) media valorilor de ieşire aferente celor 6 receptori vecini. Acest dispozitiv calculează derivatele spaţiale aferente imaginii proiectate pe retină şi permite de asemenea detecţia contururilor.

### 2.4.4. Reţeaua completă

Reţeaua completă este constituită din cei trei operatori anterior descrişi. În figura 2.8 se prezintă structura globală a unui element din reţea.

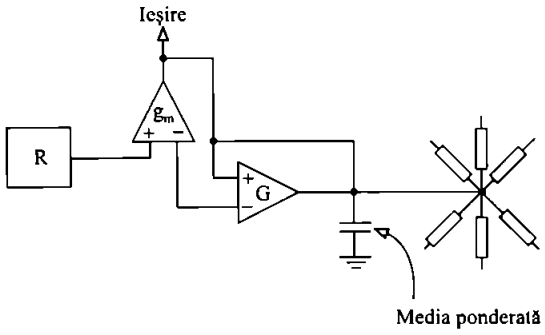



Fig. 2.8. Schema generală a unui receptor

În această reţea, amplificatorul calculează diferenţa dintre curentul generat de către fotoreceptorul R şi cel generat de reţeaua de rezistenţe. Capacitatea C stochează media spaţiu / timp a ieşirilor fotoreceptorilor vecini. Exploatarea reţelei se poate face în două moduri:

- dacă se deconectează reţeaua de rezistenţe, dispozitivul calculează derivata temporală pentru detecţia de mişcare;
- dacă reţeaua de rezistenţe este conectată, dispozitivul calculează diferenţa relativă dintre intensitatea luminoasă recepţionată într-un punct (pixel) şi media intensităţilor luminoase recepţionate în cele 6 puncte vecine. Dispozitivul este foarte sensibil în cazul punctelor izolate, mai puţin sensibil în punctele din colţuri şi insensibil la un gradient uniform. Această proprietate este de asemenea valabilă dacă obiectul din imagine este în mişcare.

Dispozitivul permite deci detecţia unui contur sau a unei mişcări. În scopul integrării acestui dispozitiv într-o structură VLSI, toate elementele sunt realizate cu tranzistoare (nu apar rezistenţe şi nici capacităţi integrate).

### 2.5. Implementările VLSI digitale

Calculatoarele clasice cu procesare serială a instrucţiunilor nu sunt capabile de o emulare eficientă a algoritmilor neuronali, care revendică pentru eficienţă ridicată arhitecturi paralele de tip DAP (*Distributed Array Processor*), MIMD (*Multiple Instruction Multiple Data Computer*) sau reţele sistolice RS (*Systolic Array Processor*).

Un neuroprocesor digital (un neurocomputer) este un microsistem VLSI digital ale cărui caracteristici au fost optimizate pentru execuţia algoritmilor neuronali. Neuroprocesoarele trebuie să exceleze prin programabilitate, flexibilitate, precizie şi *pipeline*-abilitate. Arhitectura VLSI a neuroprocesoarelor digitale trebuie optimizată în acord cu următoarele criterii de performanţă:

- Adaptabilitate: *On - chip learning*;
- Flexibilitate: Programabilitate pentru cele mai variate arhitecturi de RN;
- Cost redus: Tehnologii ieftine (de exemplu CMOS);
- Viteză ridicată: Paralelism masiv;
- Integrare la nivelul sistemului: Co-proiectarea *hardware - software*

Un neuroprocesor digital, bazat pe o arhitectură multiprocesor, este prezentat în figura 2.9. În afara ariei de procesoare EP, în structura neuroprocesorului mai apar şi alte subsisteme ca: *controller*-ul ariei de procesoare, memoria sistem aferentă ariei de procesoare, interfaţa cu sistemul gazdă (*host*) etc. Fiecare EP conţine în structura sa câteva blocuri funcţionale. Un set de registre va fi utilizat pentru memorarea datelor intermediare cum ar fi constanta de învăţare, numărul de identificare al elementului EP etc.

O unitate de înmulţire / adunare va fi utilizată în interiorul fiecărui EP pentru calculul sumelor ponderate care intervin în cadrul algoritmilor conexionişti. Numărul de biţi necesari pentru reprezentarea datelor procesate se determină pe baza preciziei ce se urmăreşte a se realiza. Pentru creşterea productivităţii sistemului se aplică tehnica *pipeline*. Ponderile sinaptice sunt memorate într-o memorie de ponderi a cărei adresare este controlată de un generator de adrese special. Funcţia de transfer neliniară aferentă neuronului poate fi implementată sub formă tabelară (*look - up table*), prin memorarea tabelei în memoria neuronului împreună cu valoarea de prag. Opţional, în structura EP pot fi introduse şi alte blocuri cum ar fi, de exemplu, o unitate ALU adiţională pentru calculul derivatei funcţiei de transfer pentru algoritmul EBP.

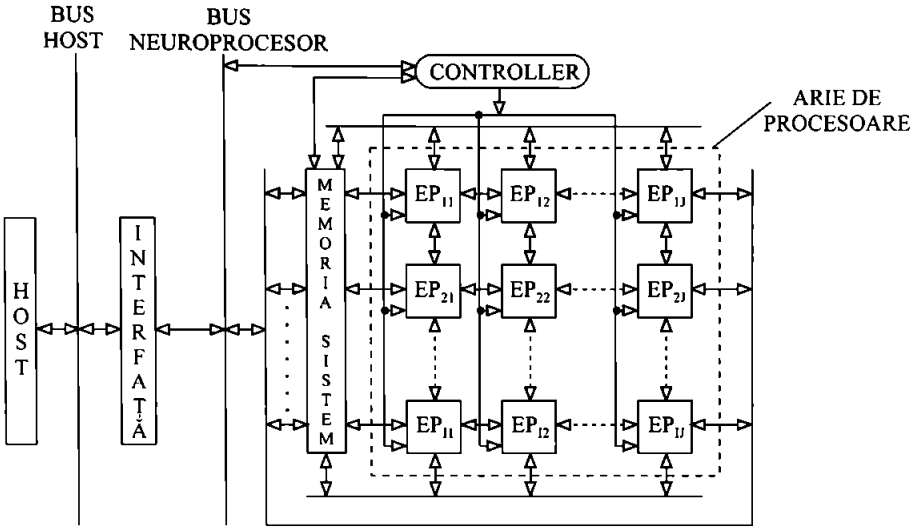



Fig. 2.9. Arhitectura neuroprocesorului digital constituit
dintr-o arie de elemente de procesare (EP)

O metodă de implementare care conduce la o eficiență foarte ridicată constă în utilizarea ariilor (rețelelor) sistolice (RS), care permit un grad înalt de paralelism și o *pipeline*-izare completă a algoritmilor conexioniști. RS rezolvă de asemenea și problema comunicațiilor interprocesor.

O RN cu trei straturi este prezentată în figura 2.10. a) iar în figura 2.10. b) se prezintă modul de implementare al acestei RN pe RS bidimensională. Alocarea corespunzătoare a procesoarelor (maparea corespunzătoare a RN pe elementele EP din structura RS) va evita conflictele în transferurile de date dintre procesoarele EP. În cazul schemei de alocare din figura 2.10, sumele ponderate se pot obține de-a lungul fiecărui rând și a fiecărei coloane.

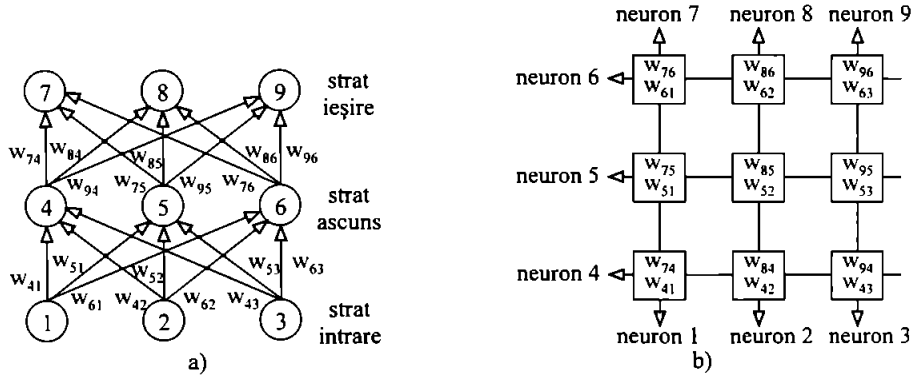



Fig. 2.10. a) Arhitectura RN cu 3 straturi; b) Maparea RN în aria de procesoare

În figura 2.11 este reprezentat fluxul de date în RS în cazul implementării algoritmului EBP (faza *feedforward*).

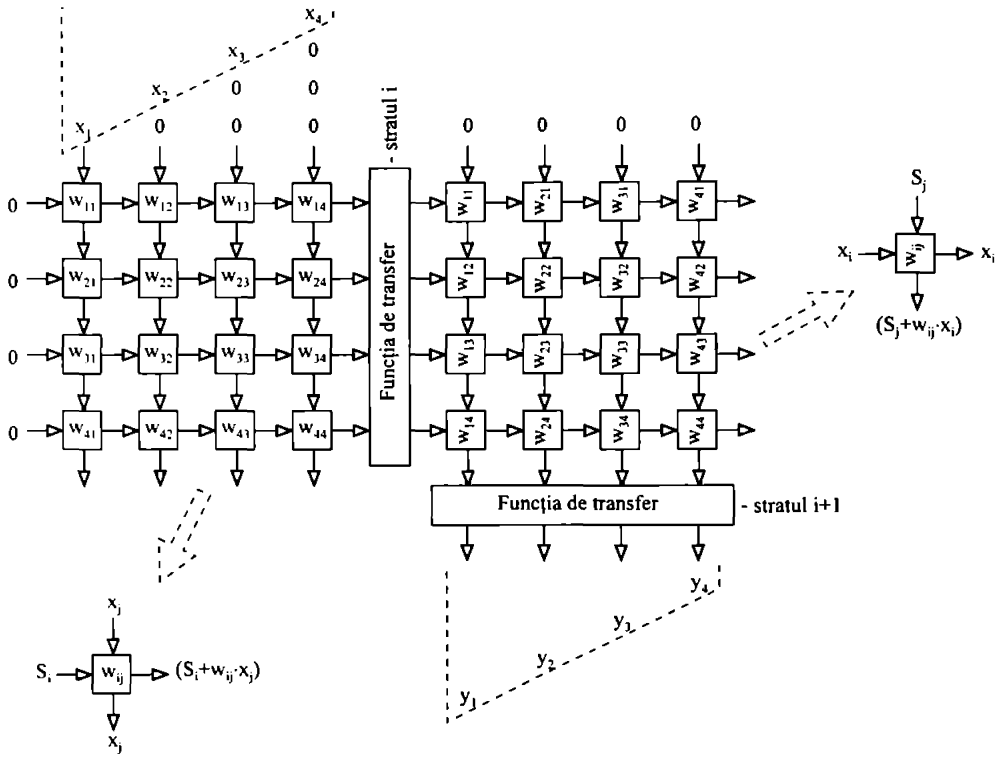



Fig. 2.11. Fluxul de date în RS pentru faza de recunoaştere aferentă algoritmului EBP

Pe parcursul fazei de recunoaştere (*feedforward*) datele se propagă dinspre stratul de intrare spre stratul de ieşire. RS funcţionează cu un tact global care asigură sincronizarea operaţiilor executate în interiorul celulelor sistolice (CS); fiecare CS execută operaţiile aritmetice şi de transfer indicate în figura 2.11. În cadrul fazei de învăţare fluxul de date îşi schimbă direcţia şi va evolua dinspre stratul de ieşire spre stratul de intrare (propagarea erorii înapoi); punctul de start al fluxului de date va fi constituit de semnalele de eroare asociate stratului de ieşire.

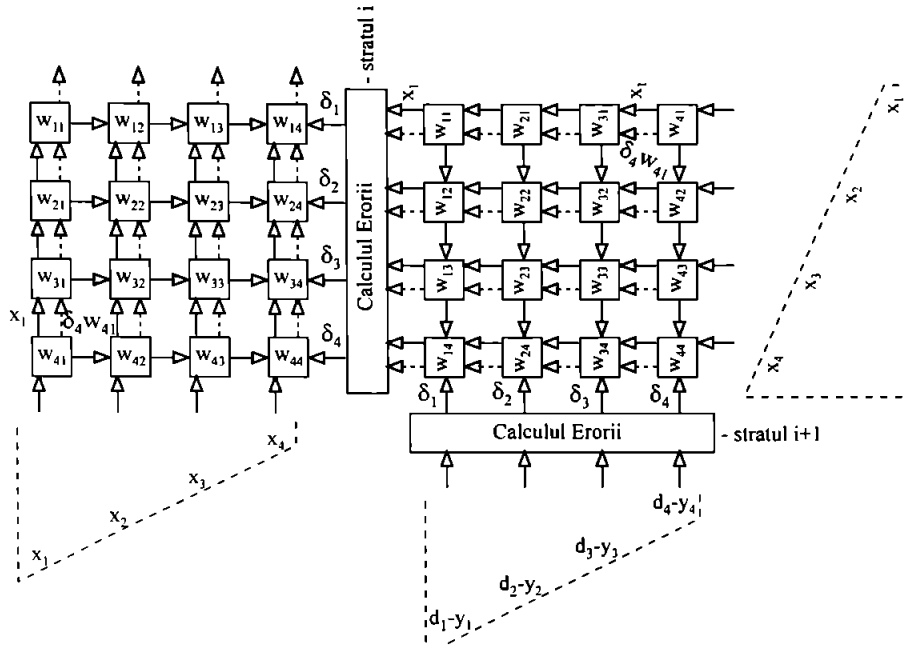
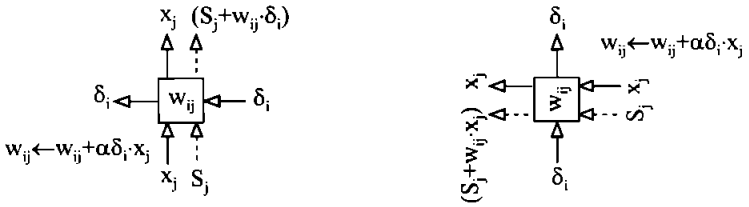



Fig. 2.12. Fluxul de date în RS pentru faza de învățare (*backward*) aferentă algoritmului EBP

Pe baza descrierii algoritmului EBP, prezentată la 1.6.4 putem deduce şi operaţiile efectuate de CS în faza *backward*. Aceste operaţii sunt prezentate în figura 2.13.



a) Stratul i b) Stratul i+1

Fig. 2.13. Operaţiile efectuate de CS în faza *backward*

Se observă că, indiferent de stratul neuronal din care fac parte, celulele sistolice au structura identică. Să mai notăm că în faza *backward*, în afară de operaţiile de transfer evidenţiate în figura 2.13, celulele sistolice efectuează şi o ajustare a mărimii de stare: $w_{ij} \leftarrow w_{ij} + \alpha \cdot \delta_i \cdot x_j$. Această operaţie realizează ajustarea ponderilor sinaptice (învăţarea).

### 2.5.1. Sistemul CNAPS

*Adaptive Solutions Inc.* a dezvoltat sistemul CNAPS (*Connected Network of Adaptive Processors*) [MUE 93], [GRI 91]. Comparativ cu maşinile tradiţionale bazate pe microprocesoare

sistemul CNAPS este capabil de performanţe mult superioare (câteva ordine de mărime). Aceste performanţe se obţin printr-un paralelism masiv şi prin capabilităţile de adaptare a RN pentru aplicaţii de recunoaştere a formelor, a vorbirii, a caracterelor scrise şi pentru probleme de control automat. Beneficiind de o coproiectare *hardware - software* eficientă sistemul CNAPS se pretează pentru aplicaţii flexibile care revendică înaltă programabilitate.

### A. Arhitectura CNAPS

În figura 2.14 este prezentată arhitectura sistemului CNAPS care este compus din 2 subsisteme principale: Aria de procesoare CNAPS şi Procesorul de Control (CP) [MUE 93]. Aria de procesoare execută algoritmii neuronali pe baza unui program CNAPS. Procesorul de control este un *controller* care controlează operaţiile de bază cum ar fi încărcarea programului şi interfaţa de intrare / ieşire cu *hardware*-ul extern. Aceste două subsisteme sunt interconectate prin busul sistemului.

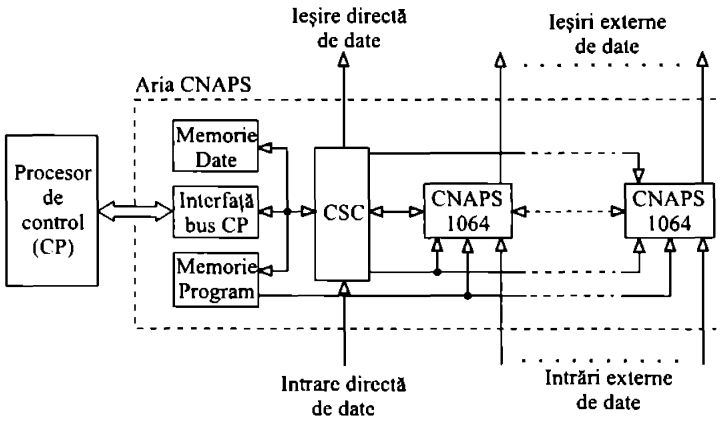



Fig. 2.14. Arhitectura sistemului CNAPS

Aria de procesoare CNAPS conţine câteva subsisteme; ea conţine mai multe circuite CNAPS - 1064 care procesează concurent datele numerice. CSC (*CNAPS Sequencer Chip*) controlează operaţiile interne sistemului multiprocesor (operaţiile dintre *chip*-urile CNAPS - 1064) precum şi operaţiile aferente celor două module de memorie şi interfaţa cu busul sistemului. Instrucţiunile de 64 biţi lungime sunt stocate în memoria program; 32 biţi din codul instrucţiunii sunt destinaţi *chip*-ului CSC. Modulul de interfaţare cu busul CP asigură conexiunea între busul sistem şi busul intern ariei CNAPS. Memoria de date (memoria DRAM) este utilizată pentru memorarea locală a datelor de intrare şi de ieşire. Memoria program (SRAM) este utilizată pentru stocarea programelor care sunt încărcate înainte de startarea execuţiei procedurilor RN. Datorită modularităţii şi unui control eficient al interfeţei aria CNAPS poate fi uşor extinsă dimensional în scopul implementării RN de dimensiuni mari. În figura 2.15 se prezintă configuraţia unui sistem care conţine 4 arii CNAPS. Fiecare arie CNAPS conţine 8 *chip*-uri CNAPS - 1064 astfel încât cele 2048 procesoare integrate în acest sistem conduc la o performanţă de 40 GCPS (*giga - conections per second*).

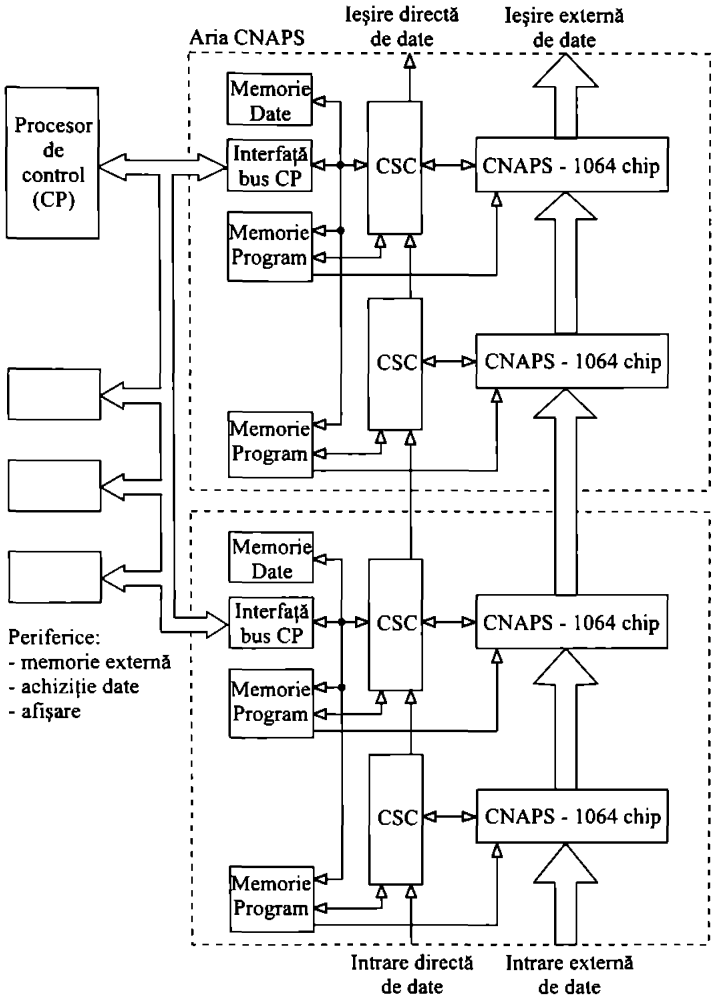



Fig. 2.15. Extensia arhitecturii CNAPS

### B. Circuitul CNAPS - 1064

Circuitul CNAPS - 1064 conţine o arie de 64 EP. Schema bloc a circuitului precum şi structura unui nod EP sunt prezentate în figura 2.16. Procesoarele EP sunt interconectate într-o configuraţie SIMD. Fiecare EP recepţionează date prin intermediul unui bus de 8 biţi (IN bus), emite date prin intermediul unui alt bus de 8 biţi (OUT bus) şi recepţionează instrucţiunile prin intermediul unui bus de 32 biţi (EPCMD bus). Adiţional, procesoarele EP vecine pot comunica direct (date) prin intermediul busului intern de 16 biţi (Inter - EP bus).

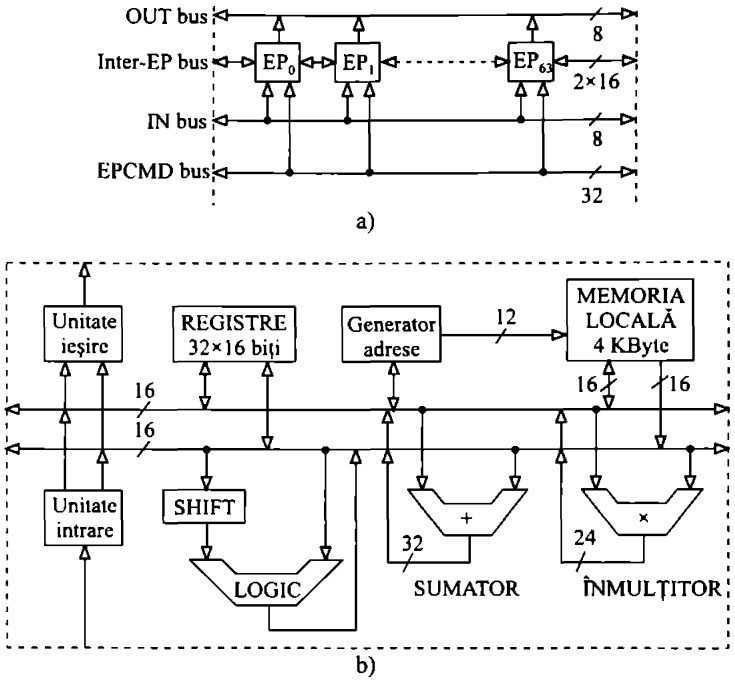



Fig. 2.16. Circuitul CNAPS - 1064. a) Schema bloc; b) Structura EP

Un procesor (EP) conţine câteva blocuri. Unitatea de intrare decodifică toate semnalele de control recepţionate (pentru elaborarea comenzilor interne) şi recepţionează date de pe busul IN - bus pe care le memorează temporar. Setul de registre este utilizat pentru memorarea datelor intermediare cum ar fi constanta de învăţare şi numărul (indexul) procesorului (EP) respectiv. Combinaţia ÎNMULŢITOR - SUMATOR este utilizată pentru calculul sumelor ponderate proprii algoritmilor conexionişti ($\Sigma\ w_{ij} \cdot x_i$). Pentru eficienţă maximă se realizează o procesare în tehnică *pipeline*. Ponderile sinaptice sunt memorate în memoria locală. Funcţia de transfer neliniară şi valoarea de prag pot fi memorate sub formă tabelară (*look - up table*) în memorie. Unitatea de ieşire memorează rezultatele într-un *buffer* de ieşire şi conţine şi circuitul de arbitrare a comunicaţiilor globale şi / sau inter - EP. Microrutine dedicate controlează atât procesarea la nivelul fiecărui EP cât şi configuraţia întregii reţele de procesoare (EP).

### C. Circuitul de secvenţiere

Circuitul de secvenţiere (CSC) este o unitate de control care controlează operaţiile neuronale la nivel *hardware*. CSC controlează 4 stări: IDLE, RUN, SPIN şi FLUSH. Diverse semnale de comandă de tip "*wait*" şi "*execute*" sunt utilizate pentru accesarea memoriilor locale şi pentru procesarea condiţiilor de întrerupere.

CSC conţine 5 module aşa cum indică figura 2.17. CPIF (*control processor interface*) controlează interfaţarea cu memoria locală.

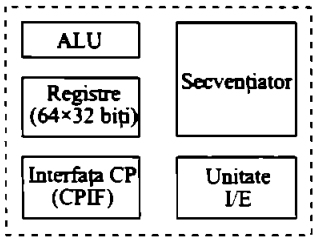



Fig. 2.17. Structura CSC

Semnalele de control emise de CP traversează interfața CPIF cu scopul de a schimba starea CSC. CP poate starta, opri sau înlănțui stările CSC prin intermediul registrelor de stare din CPIF; CPIF are logică proprie de întrerupere. Într-o execuție normală a unui program CNAPS, CPIF supervizează accesele busului local la memoria de date. Secvențiatorul controlează activitatea ariei CNAPS prin execuția instrucțiunilor care specifică secvențierea operațiilor și care controlează operațiile de intrare ieșire. Conține un "*program counter*" care poate adresa 64 K instrucțiuni de 64 biți fiecare. Unitatea I E poate configura diverse fluxuri de date care sunt sincronizate și interacționează cu busurile IN bus și OUT bus.

### 2.5.2. Sistemul multiprocesor MA 16

Circuitul MA 16 fabricat de *Siemens Corporation* operează la o frecvență de tact de 50 MHz și conține 4 lanțuri elementare conectate într-o rețea sistolică (RS) [RAM 91a]. MA 16 realizează 800 MCPS, unde o conexiune echivalează cu o operație pe 16 biți. Pentru o RS bidimensională compusă din 16 x 16 circuite MA 16 se estimează un vârf de performanță de 205 GCPS [RAM 91b]. În [RAM 91c] se propun 3 ecuații unificate care acoperă dinamica rețelelor neuronale. Aceste ecuații corespund regulii învățării supervizate:

Căutare: $$x_i = f_i\left(\left[\sum_{j=1}^{N} w_{ij} \cdot x_j\right] \cdot \lambda_i\right) \qquad (2.4)$$

Învățare: $$\Delta n = f_i' \cdot \left(\frac{\partial E}{\partial x_n} \cdot o_n + \sum_{i=1}^{N} w_{in} \cdot \Delta_i\right) \qquad (2.5)$$

Actualizare: $$w_{ij}(t+1) = w_{ij}(t) - \alpha \cdot \left(\sum_{\text{pattern}} \Delta_i \cdot x_j + \frac{\partial E}{\partial w_{ij\ explicit}}\right) \qquad (2.6)$$

unde: $o_n$ - funcția caracteristică a neuronului de ieșire
$w_{n,0} = 1$
$w_{n,0}$ = valoarea de prag
$x_i$ = intrarea neuronului
$x_0 = -1$
$\Delta_n$ = termen de corecție

În figura 2.18 se prezintă structura unui lanț elementar iar în figura 2.19 se prezintă interconectarea celor 4 lanțuri elementare în cadrul circuitului MA 16. Lanțul elementar din figura 2.18 este capabil să proceseze ecuațiile (2.4), (2.5) și (2.6).

În structura lanţului elementar distingem:

- 4 multiplicatoare şi 3 sumatoare pentru procesarea unei matrici 4 x 4;
- 4 *buffer*-e (trei de 2 x 16 cuvinte şi unul de 2 x 4 cuvinte) pentru transferurile de date;
- un multiplicator pentru înmulţirea cu o constantă;
- un sumator pentru acumularea sumelor parţiale.

Deoarece ponderile sinaptice sunt stocate în memorie (în exteriorul circuitului), numărul de pini necesari pentru comunicaţiile de date este relativ mare. Prin utilizarea celor 4 *buffer*-e I/O, numărul de pini poate fi redus; organizând memoria în m module cu adresare întreţesută, numărul de pini necesari transferului ponderilor sinaptice şi a valorilor (vectorilor) de intrare poate fi redus de m ori. *Buffer*-ul B1 este utilizat pentru valorile de intrare, B2 pentru ponderile sinaptice şi valorile $\Delta_i$ şi B3 pentru acumularea rezultatelor parţiale.

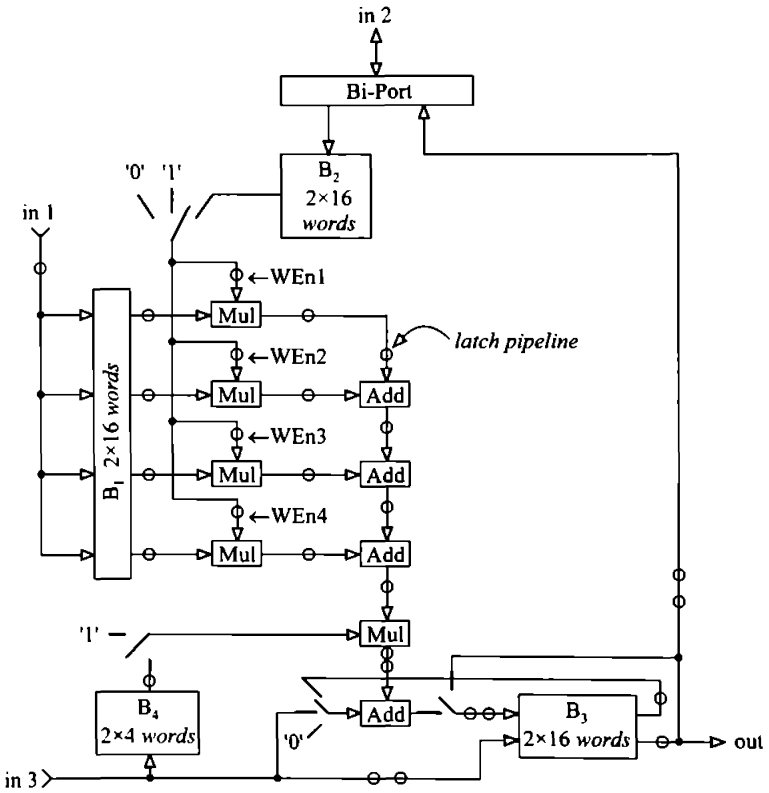



Fig. 2.18. Schema bloc a lanţului elementar

În figura 2.19 se prezintă schema bloc a circuitului MA 16 care include 4 lanţuri elementare, fiecare având structura indicată în figura 2.18 [RAM 91b]. Portul de intrare IN1 alimentează cu date toate lanţurile prin propagare sistolică. Portul IN2 este un bi-port (2 căi) care aplică ponderea sinaptică $w_{ij}$ sau valoarea de corecţie $\Delta_i$ pe intrarea fiecărui lanţ (în funcţie de operaţia efectuată). Porturile IN3 (de intrare) şi OUT (de ieşire) sunt utilizate pentru interconectarea lanţurilor elementare vecine

şi în consecinţă vor fi utilizate pentru transferul rezultatelor parţiale. Cele trei faze (*search*, *learning* şi *update*) descrise prin ecuaţiile (2.4) la (2.6) sunt mapate optimal la nivel *hardware* în operaţii sistolice aşa cum se arată în figura 2.19. Toate operaţiile sunt executate în manieră sistolică afectând un bloc de 4 x 4 operanzi.

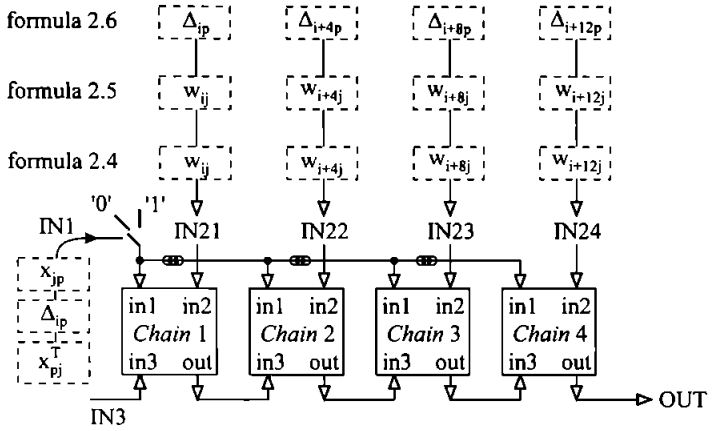



Fig. 2.19. Cele 4 lanţuri elementare utilizate pentru execuţia algoritmilor neuronali

Faza de căutare: *Buffer*-ul de intrare (B1) şi *buffer*-ul de ponderi sinaptice (B2) recepţionează ieşirea neuronului anterior $x_j$ şi respectiv ponderile $w_{ij}$. Deoarece fiecare lanţ procesează 4 neuroni (i = 1, ..., 4), vor rezulta 16 sume parţiale care se vor acumula în acumulatorul final. Toate valorile calculate sunt transferate prin lanţul de *buffer*-e B3 aşa cum se indică în figura 2.19.

Faza de învăţare: *Buffer*-ul de intrare recepţionează termenii de eroare $\Delta_i$ iar *buffer*-ul B2 recepţionează ponderile sinaptice în formă transpusă astfel încât se va calcula suma $\sum_i w_{in} \cdot \Delta_i$ aferentă relaţiei (2.5). Sumarea se poate efectua în formă distribuită conectând acumulatorul final (fig. 2.18) în buclă cu *buffer*-ul B3. Termenul de eroare ($d_n - x_n$) este preîncărcat în B3 iar termenul $f'_n$ este de asemenea încărcat în B4.

Faza de actualizare: *Buffer*-ul de intrare (B1) recepţionează intrarea în neuron calculată în faza de căutare iar *buffer*-ul B2 recepţionează termenii $\Delta_i$. Constanta de învăţare $\alpha$ este preîncărcată în *buffer*-ul B4.

Mai multe circuite MA 16 pot fi configurate fie într-o reţea sistolică liniară fie într-o reţea sistolică bidimensională. MA 16 dispune de circuite de memorie dedicate pentru stocarea ponderilor sinaptice şi a valorilor $\Delta_i$.

### 2.5.3. Circuitul NI 1000

Circuitul digital NI 1000 fabricat de firma Intel este destinat implementării RN de tip RBF (*Radial Basis Function*). Reţelele RBF prezintă anumite avantaje cum ar fi învăţarea rapidă şi o mai bună estimare, comparativ cu RN multistrat convenţionale. O reţea RBF conţine 3 straturi neuronale. Primul strat este stratul de intrare iar al doilea strat formează o arie a funcţiilor radiale de bază. Interconexiunile sinaptice dintre primul strat şi al doilea strat calculează distanţa dintre semnalul (vectorul) aplicat la intrare şi semnalul (vectorul) de ponderi memorat. Măsura acestei distanţe $D_j$ poate fi dată de distanţa euclidiană dintre cei doi vectori:

$$D_j = \sum_i |x_i - w_{ji}| \tag{2.7}$$

unde:

$x_i$ - semnalul de intrare
$w_{ji}$ - ponderea sinaptică memorată

Dacă funcţia radială de bază este o funcţie simetrică exponenţial descrescătoare, ieşirea celui de-al doilea strat va fi:

$$z_j = A_j \cdot e^{-B_j \cdot D_j} \tag{2.8}$$

unde: $A_j$ - factorul de ponderare al funcţiei exponenţiale
$B_j$ - distanţa caracteristică a funcţiei exponenţiale

Interconexiunile sinaptice dintre al doilea strat şi stratul de ieşire conduc la calculul unei sume ponderate pentru determinarea ieşirii finale:

$$y_k = \sum_k v_{kj} \cdot z_j \tag{2.9}$$

unde $v_{kj}$ reprezintă ponderea sinaptică aferentă conexiunii dintre neuronul j din cel de-al doilea strat şi neuronul k din stratul de ieşire.

Circuitul NI 1000 produs de firma INTEL calculează relaţiile (2.7) la (2.9) în manieră digitală. Schema bloc a circuitului este prezentată în figura 2.20.

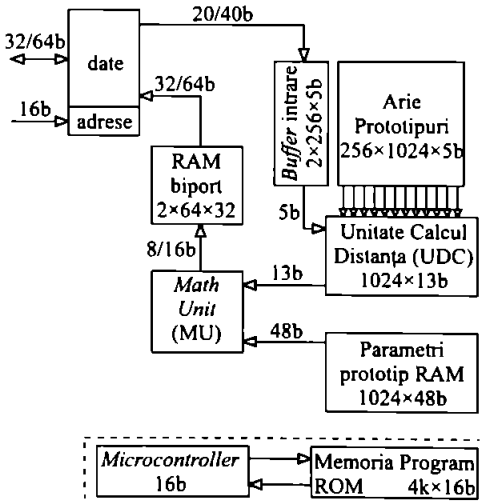



Fig. 2.20. Schema bloc a circuitului NI 1000

Dimensiunile stratului de intrare, stratului RBF şi a stratului de ieşire sunt 256, 1024 şi respectiv 64 neuroni. Reprezentarea internă a ponderilor sinaptice este realizată pe 5 biţi. În unitatea UDC se

calculează distanţa dintre semnalul de intrare (memorat în *buffer*-ul de intrare) şi semnalul sinaptic (memorat în aria de prototipuri). În unitatea MU se calculează funcţia radială de bază şi suma ponderată. Pentru performanţă procesarea se realizează în tehnică *pipeline* (unităţile UDC şi MU formează o cascadă *pipeline* pe 2 nivele). Pentru a mări capabilităţile computaţionale mai multe circuite NI 1000 pot fi conectate pe un bus comun. Se poate deci construi un procesor neuronal a cărui structură este prezentată în figura 2.21.

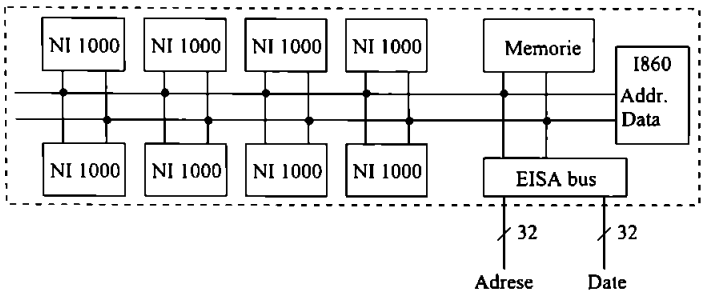



Fig. 2.21. Schema bloc a procesorului neuronal bazat pe circuite NI 1000

### 2.5.4. Alte neuroprocesoare digitale

*Orrey* şi alţii prezintă un neuroprocesor digital performant în [ORR 91]. Circuitul este fabricat în tehnologie CMOS şi conţine 8 procesoare care operează la 20 MHz. Performanţele estimate sunt de 20 GCPS pentru o RN cu 1000 neuroni. *Yasunaga* şi alţii au dezvoltat un neuroprocesor care conţine 576 neuroni şi 36864 sinapse [YAS 90]. Reprezentarea internă a datelor era realizată pe 8 - 9 biţi. Un *chip* neuronal digital de mare performanţă a fost implementat de *Uchimura* şi alţii [UCH 92]. Circuitul conţine 832 sinapse şi 13 neuroni având performanţele evaluate la 8 GCPS; datele sunt reprezentate intern pe 8 biţi. *Watanabe* şi alţii au proiectat un *chip* digital care conţine o RN cu $10^6$ sinapse [WAT 93]. Celule DRAM realizate cu un singur tranzistor au fost utilizate pentru a memora $10^6$ ponderi sinaptice într-o reprezentare întreagă pe 8 biţi. Un *chip* VLSI digital numit *TinMANN* a fost descris de *Melton* şi alţii în [MEL 92]. Circuitul se bazează pe o arhitectură stohastică şi un paralelism intern masiv. Un SAU global cu 2048 intrări (neuroni) poate fi evaluat într-un singur ciclu de tact de 67 nsec. Procesorul SPERT care lucrează la 50 MHz a fost proiectat de *Wawrzynek* şi alţii [WAW 93]. Problematica învăţării "*on-chip*" în neuroprocesoarele digitale a fost analizată în [LEH 93]. Alte implementări VLSI digitale (neuroprocesoare) sunt descrise în [SHE 95] şi [SAN 92].

## 2.6. Principii de proiectare a neuroprocesoarelor sistolice

Pentru a exploata paralelismul intrinsec caracteristic RN, acestea trebuie implementate pe arhitecturi multiprocesor. Arhitectura care realizează comunicaţii eficiente între procesoare va furniza şi o mare putere de calcul pe unitatea de suprafaţă. Implementările sistolice digitale dedicate reţelelor neuronale pot opera cu eficienţă deosebită datorită procesării în tehnică *pipeline* a datelor. Într-o reţea sistolică (RS) pot coexista mai multe fluxuri de date *pipeline*-izate. Arhitectura multi-EP în inel şi respectiv multi-EP matricială [PRZ 91] sunt 2 arhitecturi sistolice reprezentative. Comparativ cu circuitele neuronale analogice, arhitecturile sistolice oferă o mai bună flexibilitate, o mai mare precizie şi o completă *pipeline*-izare a fluxurilor de date.

### 2.6.1. Arhitectura sistemului

Schema bloc a unui sistem bazat pe o arhitectură multi-EP în inel şi respectiv multi-EP matricială este prezentată în figura 2.22.

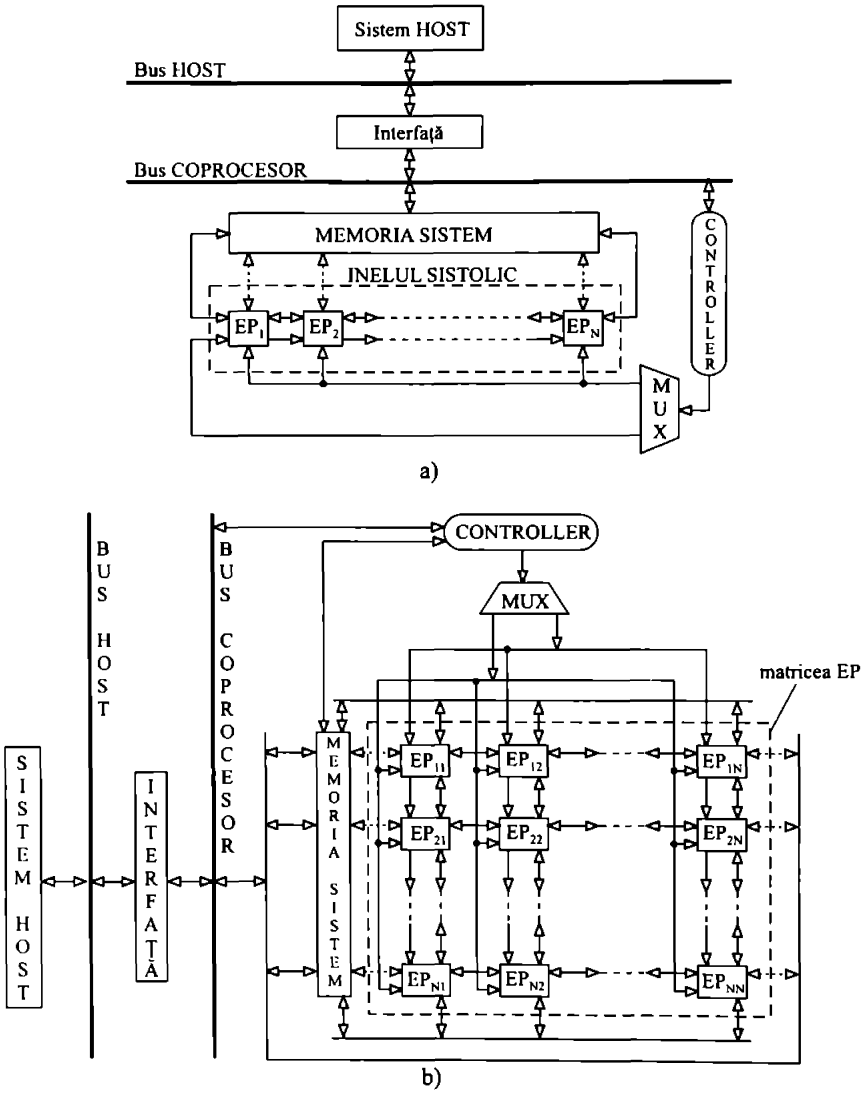



Fig. 2.22. Arhitectura sistemului bazat pe reţele sistolice
a) RS monodimensională conectată în inel; b) RS bidimensională pătratică

*Controller*-ul RS poate fi cablat sau microprogramat. În varianta microprogramată acesta va emite o microinstrucţiune pe fiecare ciclu de tact. Microcodul şi adresele de acces la memoria sistem pot fi

transmise procesoarelor EP din cadrul RS fie în sistem *broadcasting* fie în sistem *pipeline*; tipul transmisiei este selectat cu ajutorul multiplexorului din figura 2.22. Pentru RS conectată în inel, operaţiile I/E dintre RS şi memoria sistem vor fi suportate doar de EP-urile extreme ($EP_1$ şi $EP_N$) şi în acest fel memoria sistem biport va fi partajată de procesoarele inelului sistolic. În ciclul de iniţializare fiecare EP citeşte datele locale aferente din memoria sistem prin intermediul busurilor locale indicate în figura 2.22 prin linii punctate. În cazul RS bidimensionale, fiecare $EP_{ij}$ comunică cu cei patru vecini ai săi iar operaţiile I / E cu memoria sistem apar pe cele patru laturi ale matricii sistolice. În consecinţă, memoria sistem va fi o memorie cu 4 porturi de acces partajată de matricea EP. Similar, în ciclul de iniţializare, datele locale aferente EP-urilor din matricea sistolică se obţin prin intermediul busurilor locale marcate cu linii punctate şi situate pe laturile de est şi respectiv de vest aferente matricii sistolice. Un EP din zona de mijloc a unei linii recepţionează datele locale proprii printr-o comunicaţie *pipeline*.

Memoria sistem recepţionează date de la sistemul gazdă (*host*) în urma activării unei astfel de cereri; cererea este activată de sistemul gazdă şi recepţionată de *controller*. Sistemul gazdă serveşte ca interfaţă între utilizator şi RS. Acesta va furniza informaţiile specifice aplicaţiei, cum ar fi vectorii de intrare, ponderile sinaptice iniţiale, parametrii de control convergenţă etc. *Controller*-ul specifică şi monitorizează procesarea în cadrul fiecărui EP şi guvernează, de asemenea, şi comunicaţiile de date dintre EP. *Controller*-ul va avea memoria sa dedicată unde va stoca microcodul recepţionat de la sistemul gazdă.

### 2.6.2. Proiectarea elementului de procesare (EP)

Implementările digitale dedicate RN au la bază 4 principii:

- implementări bazate pe microprocesoare de uz general
- implementări bazate pe procesoare de semnal (DSP - *digital signal processor*)
- implementări bazate pe neuroprocesoare de uz general
- implementări bazate pe neuroprocesoare dedicate

În primele 2 cazuri sistemul va avea la bază procesoare existente pe piaţă. În al treilea caz se utilizează un *hardware* de uz general în ideea realizării unei neuroprocesări flexibile. Acest *hardware* va putea fi utilizat în diverse aplicaţii de procesare de semnal sau de imagine. In al patrulea caz se construieşte un *hardware* dedicat câtorva algoritmi neuronali. În tabelul 2.2 se prezintă o comparaţie între cele 4 principii de implementare.

| Proprietăţi | Implementări bazate pe microprocesoare de uz general | Implementări bazate pe DSP | Neuroprocesoare de uz general | Procesoare dedicate |
|---|---|---|---|---|
| Paralelism | Cel mai slab | Slab | Înalt | Cel mai înalt |
| Flexibilitate | Cea mai înaltă | Slabă | Înaltă | Cea mai slabă |
| Precizie | Cea mai înaltă | Înaltă | Slabă | Cea mai slabă |
| Programabilitate | Cea mai înaltă | Slabă | Înaltă | Cea mai slabă |
| Integrare / Scalabilitate | Slabă | Înaltă | Cea mai înaltă | Cea mai slabă |
| Viteză de procesare | Cea mai joasă | Joasă | Înaltă | Cea mai înaltă |
| Durată ciclu proiectare | Cel mai scurt | Scurt | Lung | Cel mai lung |
| Costul de producţie | Cel mai înalt | Înalt | Scăzut | Cel mai scăzut |
| Aplicabilitate | De uz general | Procesare de semnal | Diverse | Foarte limitate |

Tabelul 2.2. Comparaţia celor 4 tipuri de implementări digitale dedicate RN

Comparând datele prezentate în tabelul 2.2 putem concluziona că neuroprocesoarele de uz general realizează cel mai bun raport performanţe / cost. O propunere de structură internă pentru nodul sistolic EP este prezentată în figura 2.23 [CHA 94]. Pentru memorarea locală a datelor fiecare EP conţine un *cache* de date. Multiplicatorul şi sumatorul reprezintă principalele unităţi de procesare din structura EP.

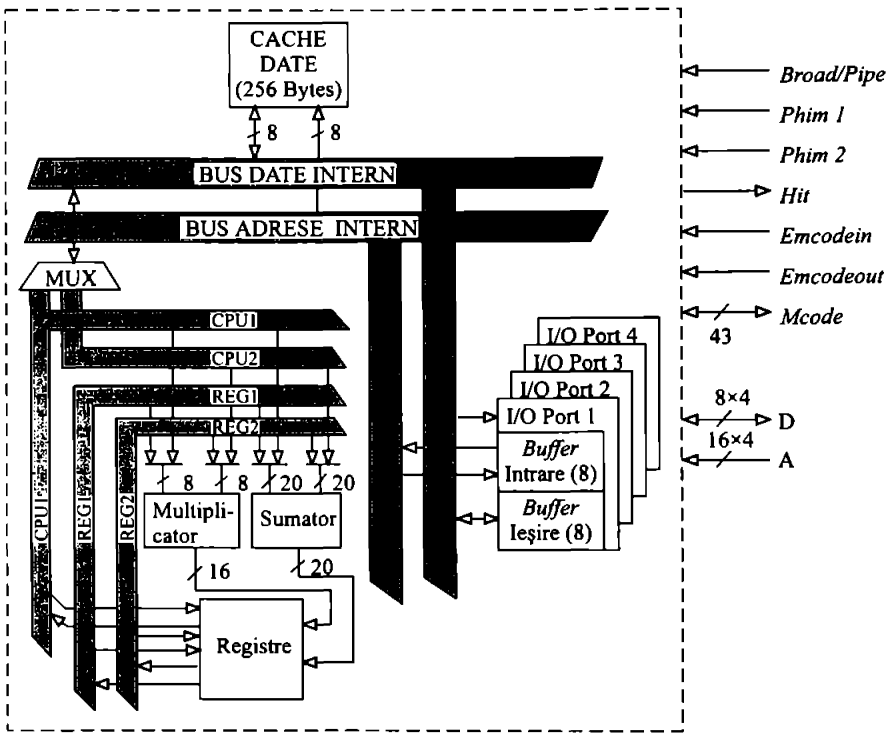



Fig. 2.23. Schema bloc aferentă modulului EP

Structura EP este adaptată RS bidimensionale din figura 2.22. b). Fiecare EP conţine 4 porturi I/O pentru a comunica date cu cele 4 EP vecine. O unitate MMU (*memory management unit*) stabileşte strategiile de alocare a memoriei precum şi strategiile de *fetch* şi actualizare a memoriei principale (în raport cu *cache*-urile locale). MMU este locat în *controller* şi este partajat de toate unităţile EP din cadrul RS. RS operează sincron, cu un tact global.

O tabelă *look - up* va fi utilizată pentru implementarea funcţiei de activare aferentă neuronului. Rutinele de nivel înalt executate de sistemul gazdă sunt compilate în macroinstrucţiuni care sunt transmise apoi *controller*-ului. *Controller*-ul decodifică macroinstrucţiunile şi le translatează în microinstrucţiuni pe care le emite spre EP-urile din structura RS (*broadcast*) prin intermediul celor 43 linii ale busului *Mcode*. Pentru o *pipeline*-izare cât mai eficientă a instrucţiunilor este recomandată utilizarea unui set de instrucţiuni specific RISC (*reduced - instruction set computer*).

Tot în scopul creşterii vitezei de procesare, registrele EP sunt accesibile prin intermediul a 4 busuri de date: REG1, REG2, CPU1 şi CPU2. Drept consecinţă multe operaţii interne care implică

registrele EP vor putea fi suprapuse deoarece nu implică ocuparea aceloraşi busuri. Un multiplexor este utilizat pentru multiplexarea celor 4 busuri pe busul de date intern. Pentru transferul datelor între *cache*-ul de date şi setul de registre este utilizat busul CPU1.

În funcţie de modul de adresare utilizat, atât pentru multiplicator cât şi pentru sumator, unul dintre operanzi este preluat fie de pe busul CPU1 fie de pe busul REG1, iar celălalt operand este preluat fie de pe busul CPU2 fie de pe REG2. Circuitul a fost proiectat cu ajutorul editorului *Magic* elaborat la Universitatea Berkeley. Performanţele circuitului pot fi îmbunătăţite dacă în procesul de proiectare s-ar utiliza pachete CAD comerciale.

# 3. ANALIZA IMPLEMENTĂRILOR SISTOLICE

În acest capitol vom face o prezentare a metodelor şi descrierilor sistolice şi vom arăta că aceste metode permit localizarea spaţială a algoritmilor conexionişti. Această abordare va permite obţinerea unor algoritmi sistolici capabili să execute eficient operaţiile aferente modelelor conexioniste actuale. Vom propune o arhitectură sistolică generală din care vor deriva 3 arhitecturi exploatabile. În sfârşit, vom alege arhitectura optimă şi vom motiva această alegere.

## 3.1. Principiul procesării sistolice

Principiul procesării sistolice a fost introdus de *H. T. Kung* şi *C. E. Leiserson* în anul 1978 [KUN 78]. Termenul "sistolic" provine dintr-o analogie între circulaţia fluxului de date în reţeaua sistolică (RS) şi cea a sângelui în aparatul circulator aferent organismelor vii. Inima este asimilată elementelor de procesare (EP) din componenţa RS, care recepţionează, procesează şi transmit datele. RS reprezintă instrumentul ideal pentru realizarea unei anumite clase de procesoare specializate. Dificil de implementat cu componente discrete din cauza unui cost ridicat, algoritmii sistolici intră în actualitate graţie tehnologiilor VLSI (*Very Large Scale Integration*).

Structura în reţea reprezintă suportul tipic pentru algoritmii sistolici. Celulele sistolice (CS) sunt procesoare elementare identice, conectate exclusiv cu celulele vecine. Fluxurile de date traversează reţeaua prin comunicaţii locale şi într-o manieră sincronă. Organizarea în reţea (fig. 3.1) şi funcţionarea sincronă permit procesarea concurentă a informaţiilor în celulele sistolice conducând la o putere de calcul foarte ridicată. În sfârşit, vom vedea că în anumite condiţii, procesarea în tehnică *pipeline* a datelor devine posibilă, conducând la o exploatare maximală a RS.

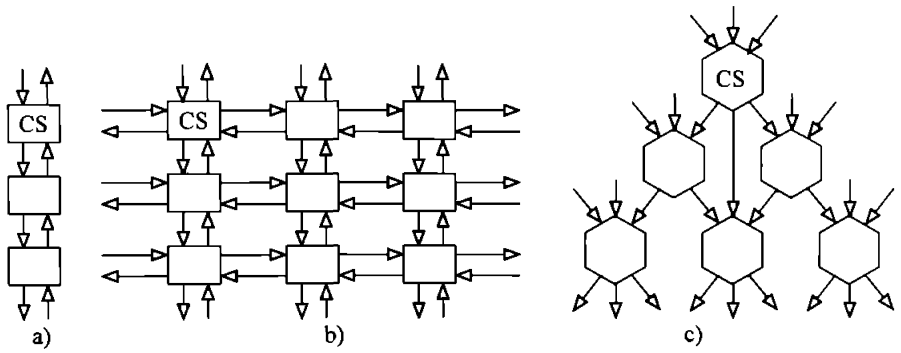



Fig. 3.1. Topologii de RS. a) Liniară; b) Pătrată; c) Hexagonală
(CS - celula sistolică referită adesea şi cu termenul EP - element de procesare)

Formal, o reţea sistolică este o structură de calcul care posedă următoarele caracteristici:

- Funcţionare sincronă;
- Modularitate şi regularitate. RS este un ansamblu de unităţi de calcul (CS) asemănătoare sau identice, conectate prin scheme de interconexiuni omogene, conferind o extensibilitate teoretic infinită;
- Localizare spaţială şi temporală;
- Posibilitatea procesării datelor în tehnică *pipeline*.

Clasa de aplicaţii în care RS sunt extrem de eficiente cuprinde problemele în care volumul de calcul de efectuat este mult mai mare decât cel al transferurilor de date de realizat [COS 89].

Fiecărei topologii de RS i se asociază o clasă de probleme [KUN 88], [QUI 89]. Reţelele liniare sunt utilizate pentru operaţii de convoluţie, înmulţiri matrici - vectori, produse scalare etc. Reţelele rectangulare sunt performante în cazul produselor matriciale şi, în general, pentru aplicaţii de filtrare pe imagini (operaţii cu pixeli). Reţelele triunghiulare sau hexagonale sunt performante în operaţiile de decompoziţie LU a matricilor (descompunerea unei matrici în 2 matrici triunghiulare, triunghiulară inferioară L şi triunghiulară superioară U) şi în operaţiile de înmulţire a matricilor cu structuri particulare (triunghiulare, bandă diagonală).

Toate operaţiile care trebuie implementate pe o RS depind mai întâi de topologia reţelei. Datele tranzitează reţeaua traversând legăturile definite între celulele vecine. Prin urmare, implementarea sistolică a unui algoritm nu este unică ceea ce înseamnă o sarcină relativ complexă pentru proiectant. RS optimă (pentru o anume aplicaţie) va fi aleasă în funcţie de performanţe (viteza de procesare), complexitate cât mai redusă, posibilităţi de optimizare (*pipeline*-izare, suplеţe în execuţia operaţiilor de intrare / ieşire etc.). După cum vom vedea, această etapă de definire a topologiei RS este crucială deoarece ea stabileşte definitiv capabilităţile viitoarei RS.

Există actualmente mai multe instrumente de concepţie a RS care au fost dezvoltate şi puse la dispoziţia proiectanţilor. Metodologia lui *Leiserson* şi *Saxe* se bazează pe conversia unui sistem sincron reprezentat printr-un graf în algoritm sistolic. Metoda lui *Quinton* permite o sinteză automată a RS, pornind de la o formulare recurentă a calculelor de executat [QUI 84]. Metodologia lui *Li* şi *Wah* reprezintă un principiu general pentru optimizarea concepţiei unei RS planare [LI 85]. *Cosnard* şi *Tchuente* au propus, de asemenea, o metodologie bazată pe o analiză descendentă care divizează problema proiecţiei unui graf de dependenţe în 2 etape: construcţia unei reţele liniare ale cărei intrări sunt vectori, apoi sistolizarea acestei transformări [COS 89].

În cele ce urmează, vom utiliza metoda lui *Kung* [KUN 88] pentru descrierea RS şi vom valida formal funcţionarea acesteia pe baza metodei lui *Quinton* [QUI 84]. Metoda lui *Kung* se bazează pe manipulări sistematice de grafuri şi ale proiecţiilor acestora; este perfect adaptată pentru calcule simple, cum ar fi produsul matrice - vector, care stă la baza algoritmilor conexionişti.

Această metodologie de descriere a operaţiilor permite extragerea soluţiei de maxim paralelism; este o descriere sub formă de graf de dependenţă (DG - *Dependece Graph*) [KUN 88] care nu face apel la nici o noţiune arhitecturală, de rebuclaj sau de optimizare temporală. În cadrul DG se iau în considerare numai dependenţele dintre nodurile acestuia. Mai aproape de arhitectură, graful fluxului de date (SFG - *Signal Flow Graph*) este o formulare mai completă care pune în evidenţă relaţiile dintre date, întârzierile (*delays*) şi eventualele rebuclaje. Acesta defineşte de asemenea structura suport aferentă algoritmului.

Trecerea de la un DG la un SFG pune următoarele probleme:

- cum vor fi atribuite (repartizate) operaţiile pe procesoare;
- în ce ordine vor trebui executate aceste operaţii.

Răspunsurile la aceste 2 probleme vor defini asignarea procesoarelor şi secvenţierea operaţiilor. Metodologia pe care o vom descrie este cunoscută sub denumirea de mapare canonică [KUN 88].

### 3.1.1. Graful de dependenţă (DG)

DG este un graf care descrie dependenţele dintre calculele ce compun un algoritm; în nodurile grafului nu vor fi prezentate calculele efectuate.

Pentru a ilustra această definiţie, vom construi DG aferent operaţiei de înmulţire a unei matrici cu un vector.

Vom considera operaţia:

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$$

unde:

**x** - vector de dimensiune [N]
**y** - vector de dimensiune [P]
**W** - matrice [P, N]

Sub formă indicială, ecuaţia matricială devine:

$$y_i = \sum_{j=1}^{N} w_{ij} \cdot x_j \quad \text{cu } i = 1, 2, \ldots, P \tag{3.1}$$

Indecşii de această formă definesc doar localizarea spaţială a elementelor matricii şi a celor 2 vectori implicaţi în operaţie. Introducând un index temporal k, vom putea rescrie algoritmul (3.1) sub o formă în care fiecare variabilă nu este modificată decât o singură dată (*simple assignement form*).

$$y_i^k = y_i^{k-1} + w_i^k \cdot x^k \tag{3.2}$$

cu: $y_i^0 = 0$, $y_i^N = y_i$ şi $w_i^k = w_{ik}$

Această formulare este o ecuaţie recurentă cu un index spaţial (i) şi un index temporal (k). Pe baza ecuaţiei (3.2) vom putea construi un graf de dependenţă, de forma celui prezentat în figura 3.2, unde am considerat vectorul **x** de dimensiune 4, vectorul **y** de dimensiune 3 şi matricea **W** de dimensiune [3, 4].

În general, datele de intrare sunt difuzate spre toate nodurile grafului. Această operaţie revendică un bus de comunicaţie global care va avea repercusiuni negative asupra realizării fizice.

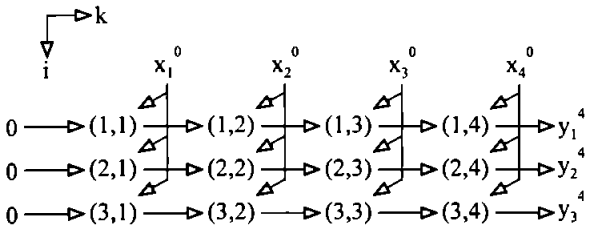



Fig. 3.2. Graful de dependenţă aferent produsului matrice - vector. În acest graf, datele de intrare sunt distribuite tuturor nodurilor. Arcele nu indică decât o dependenţă funcţională între noduri şi nu induc nici o întârziere (*delay*)

Pentru implementările fizice este interesant de a transforma această difuzie de date spre toate nodurile cu o transmisie de tipul din aproape în aproape. În acest sens, algoritmii trebuie localizaţi în scopul obţinerii unui graf de dependenţă local (figura 3.3). Există mai multe studii teoretice dezvoltate pe problema localizării algoritmilor pentru a ajunge sub forma unui Sistem de Ecuaţii Recurente Uniforme (SERU) [QUI 84], dar o metodă generală încă n-a fost adoptată.

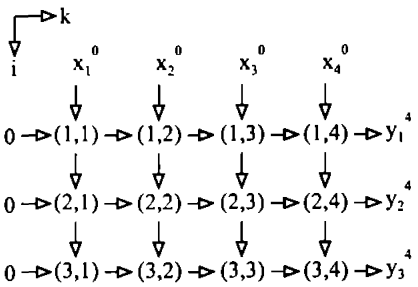



Fig. 3.3. Graful de dependenţă localizat aferent produsului matrice - vector. Un algoritm este localizat dacă toate variabilele nu depind decât de variabile din noduri vecine

Datele proprii unui nod din graf care nu sunt niciodată deplasate sunt denumite date rezidente. O dată poate fi difuzată (spre mai multe noduri simultan), emisă (de la un nod la altul) şi rezidentă.

În DG, operaţiile efectuate asupra datelor nu sunt descrise. Sunt descrise doar relaţiile dintre noduri care definesc dependenţe funcţionale. Ele sunt materializate prin arcele de dependenţă care unesc nodurile grafului.

Totuşi, aşa cum am subliniat deja, pentru a defini o arhitectură de calcul pornind de la DG, trebuie repartizate (alocate) operaţiile pe procesoarele reţelei şi trebuie definită o secvenţiere. Această operaţie poate fi realizată pornind de la graful de dependenţe localizat. Ea va conduce la un graf de mişcare a datelor (SFG - *Signal Flow Graph*) prin efectuarea unor operaţii de proiecţie spaţială şi temporală.

### 3.1.2. Graful de mişcare a datelor (SFG)

Un SFG conţine informaţii funcţionale şi structurale. Informaţiile funcţionale descriu operaţiile efectuate în noduri. Se consideră că aceste operaţii nu generează întârzieri. Informaţiile structurale descriu legăturile în termenii conexiunilor şi întârzierilor. Se separă deci în mod explicit calculul şi timpul. Un DG se transformă în SFG prin 2 operaţii:

- o proiecţie spaţială a cărei direcţie este indicată de un vector de proiecţie spaţială **d**. Aceasta corespunde etapei de asignare a nodurilor la procesoarele reţelei (*processors assignement*);
- o schemă de secvenţiere a operaţiilor (*scheduling*) reprezentată printr-un vector de secvenţiere **s** care pointează pe o direcţie normală liniilor izocrone. Aceste linii trec prin toate nodurile grafului care efectuează acelaşi calcul simultan (dar pe date diferite).

Vectorul **s** trebuie să satisfacă 2 condiţii:

- toate arcele de dependenţă trebuie să pointeze în aceeaşi direcţie şi anume în lungul liniilor izocrone (cauzalitate);
- liniile izocrone nu sunt paralele cu direcţia de proiecţie spaţială **d** (identitate de asignare temporală).

Pentru un DG dat căruia îi asociem o direcţie de proiecţie temporală, există 4 clase de scheme de secvenţiere posibile [KUN 88]:

- secvenţierea normală: vectorii de proiecţie spaţială şi de secvenţiere sunt coliniari;

- secvenţierea recursivă: vectorul de secvenţiere este coliniar cu vectorul de index spaţial al DG;
- secvenţierea sistolică: trebuie să existe un *delay* pe fiecare arc al SFG;
- secvenţierea optimizată: se obţine în urma unei optimizări care ţine cont de dependenţele specifice, de performanţele procesoarelor şi, în general, de tot ceea ce ar putea îmbunătăţi concepţia.

În cele ce urmează vom insista asupra secvenţierii sistolice. Aceasta ne va permite derivarea unor arhitecturi de reţea adaptate implementărilor în circuite VLSI digitale a operaţiilor specifice algoritmilor conexionişti. Vom studia mai multe tipuri de proiecţie spaţială $\mathbf{d} = [i, k]^T$ şi temporală $\mathbf{s} = [i, k]^T$ cu scopul de a ilustra definiţiile precedente şi vom deduce patru SFG cu secvenţiere sistolică. Reluând graful DG aferent produsului matrice - vector, putem deriva trei SFG (fig. 3.4) alegând direcţii de proiecţie coliniare, pointând în cele 3 direcţii permise de o secvenţiere liniară. Toate grafurile SFG rezultate sunt monodimensionale.

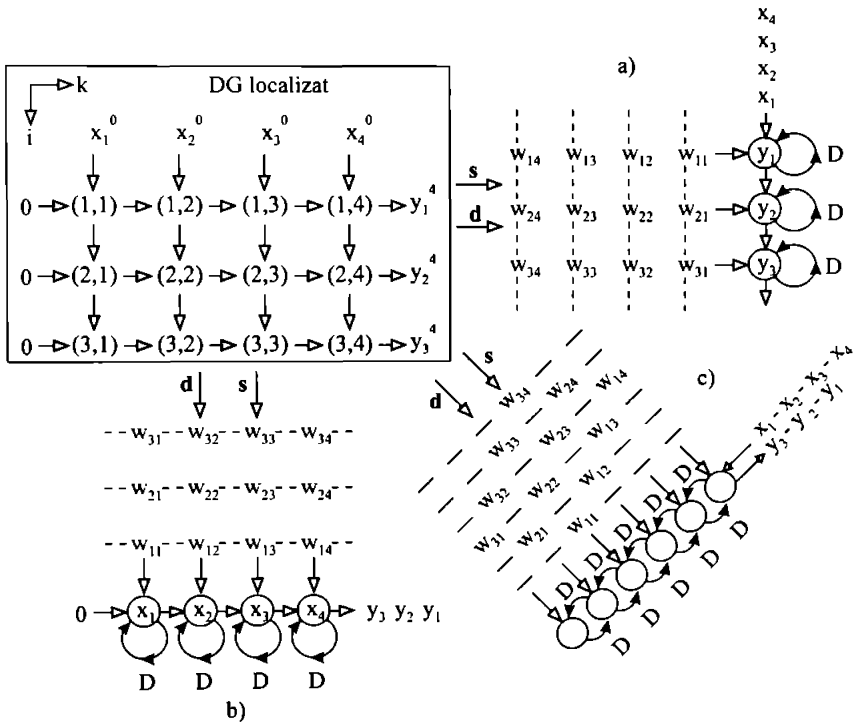



Fig. 3.4. Trei SFG derivate din graful DG localizat aferent produsului matrice - vector

Cele 3 soluţii corespund următoarelor proiecţii:

- soluţia 1 (fig. 3.4. a)):

$$\mathbf{d} = \mathbf{s} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Matricea **W** este prezentată linie cu linie grafului SFG. Am obţinut secvenţierea normală care nu este sistolică deoarece nu există *delay*-uri între noduri. Componentele vectorului **x** sunt introduse succesiv dar la introducere fiecare se distribuie tuturor nodurilor reţelei. Elementele matricii **W** situate pe cele 4 linii izocrone evidenţiate sunt introduse succesiv şi sincron cu cele 4 elemente ale vectorului **x**. Fiecare nod calculează produsul parţial între componenta vectorului **x** şi elementul matricii **W** corespondent. De asemenea fiecare nod însumează produsele parţiale succesiv elaborate rezultând astfel cele 3 componente aferente vectorului **y**. Produsul parţial obţinut în pasul curent este retrimis în acelaşi nod prin arcul de rebuclaj căruia îi este asociat *delay*-ul D. În fiecare pas nodul realizează produsul componentelor de pe intrări şi adună la acest produs produsul parţial anterior. După 4 paşi succesivi componentele vectorului **y** vor fi acumulate în cele 3 noduri, urmând a fi extrase.

- soluţia 2 (fig. 3.4. c))

$$\mathbf{d} = \mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Să remarcăm secvenţierea sistolică realizată de acest SFG datorată faptului că pe fiecare arc există un *delay*. Componentele vectorului **y** circulă în sens opus componentelor vectorului **x**. Componentele vectorului **x** circulă fără a fi modificate. În fiecare nod al grafului, elementele matricii **W** vor intra la intervale constante de 2 impulsuri, aşa cum indică cele 6 linii izocrone. Dacă vectorul înmulţitor (**x**) are N componente, vor fi necesare N - 1 impulsuri pentru introducerea primei sale componente ($x_1$) înainte ca procesul de calcul să înceapă efectiv (în fig. 3.4. c) sunt necesare 3 impulsuri pentru ca $x_1$ să ajungă în nodul care va efectua primul produs parţial: $w_{11} \cdot x_1$). Pentru o matrice [P, N], vor fi necesare alte P - 1 impulsuri din momentul execuţiei ultimului produs ($w_{34} \cdot x_4$) până la extragerea ultimei componente aferentă vectorului **y**. În medie, fiecare nod nu va fi activ decât pe durata unui impuls din două.

- soluţia 3 (fig. 3.4. b))

$$\mathbf{d} = \mathbf{s} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Este tot o secvenţiere standard, nesistolică datorită inexistenţei *delay*-urilor pe arcele dintre nodurile grafului. Matricea **W** este aplicată grafului SFG pe coloane. Componentele vectorului **x** sunt introduse vertical şi simultan în nodurile SFG, înainte de aplicarea elementelor matricii **W**. Liniile izocrone sunt identice cu liniile matricii **W**; elementele unei linii sunt aplicate simultan nodurilor SFG, se elaborează produsele şi se obţine componenta corespondentă a vectorului rezultat **y** (nu există *delay*-uri pe arcele dintre noduri). Să notăm că elementele vectorului **x** sunt rebuclate în nodurile SFG (*delay*) pentru a fi "prezente la întâlnirea" cu următoarea linie a matricii **W**.

Dintre cele 3 SFG prezentate, unul singur are o secvenţiere sistolică (soluţia 2). Alegând convenabil vectorii **d** şi s, şi soluţiile 1 şi 3 pot fi transformate în SFG cu secvenţiere sistolică.

Vom atribui vectorului **s** o valoare care implică un *delay* pe fiecare arc din SFG:

$$\mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

şi vom trece din nou în revistă cele trei direcţii de proiecţie spaţială prezentată anterior. Vom obţine 3 SFG cu secvenţiere sistolică descrise în figura 3.5, cu o perioadă *pipeline* $\alpha = 1$ pentru soluţiile 1 şi 2 (fig. 3.5. a) şi 3.5. b)), şi $\alpha = 2$ pentru soluţia 3 (fig. 3.5. c)).

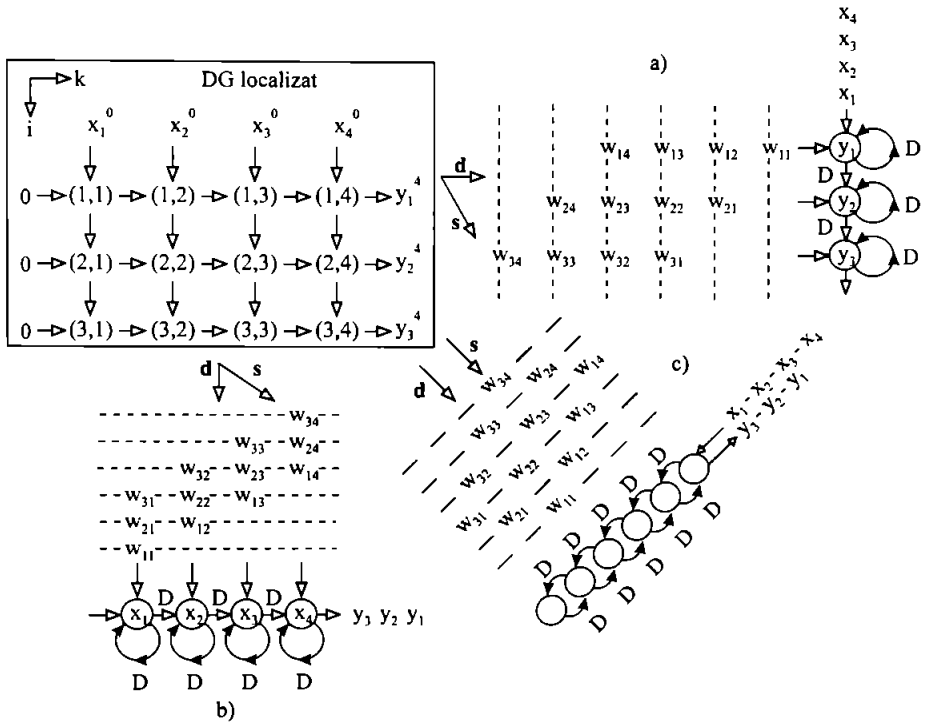



Fig. 3.5. Reţele sistolice (RS) pentru produsul matrice - vector. Liniile izocrone definesc nodurile care procesează simultan şi vectorul **d** este perpendicular pe aceste linii.
Această implementare impune un decalaj în aplicarea elementelor matricii **W**

### 3.1.3. Prima soluţie sistolică

Această soluţie (fig. 3.5. a)) este rezultatul unei proiecţii după vectorii:

$$\mathbf{d} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \text{şi} \qquad \mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

SFG-ul rezultat este compus din P noduri. În termenii transferurilor de date, această soluţie presupune emisia vectorului **x** şi a matricii **W** în timp ce vectorul **y** este rezident. Un produs parţial

$y_i^k$ ($k < N$) se calculează în fiecare nod. Produsul parţial anterior este acumulat (arcul de rebuclaj) pe fiecare impuls produsului curent, pentru a deveni după N impulsuri, componenta corespondentă a vectorului rezultat **y**. Calculul se descompune în 2 faze:

1. Traversarea grafului SFG de către matricea sinaptică **W** şi vectorul **x** cu acumularea sumelor parţiale $y_i^k$ cu $k < N$.
2. Extragerea vectorului **y** în P impulsuri.

Timpul total de calcul va fi deci de (N + P) impulsuri. Această soluţie se pretează la o implementare fizică şi, după cum vom vedea, a şi stat la baza realizării unei reţele conexioniste.

### 3.1.4. A doua soluţie sistolică

Al doilea SFG monodimensional cu secvenţiere sistolică este dualul celui precedent, cu proiecţiile spaţială şi temporală realizate după vectorii:

$$\mathbf{d} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \text{şi} \qquad \mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Se obţine SFG-ul din figura 3.5. b), compus din N noduri. În cadrul acestei variante, componentele vectorului **y** şi ale matricii **W** sunt emise; componentele vectorului **x** sunt rezidente în fiecare nod şi rămân localizate graţie arcelor de rebuclaj. Produsele parţiale $y_i^k$ sunt transmise între noduri şi acumulate în modul destinaţie. Componentele vectorului **x** sunt rezidente (graţie rebuclajului) şi reutilizate pe fiecare impuls pentru elaborarea produsului local (al doilea termen al produsului va fi componenta matricii **W** care intră în nod pe impulsul respectiv). Calculul se descompune în 2 faze:

1. Introducerea vectorului **x** în N impulsuri;
2. Traversarea grafului SFG de către matricea sinaptică **W** (suprapusă cu tranzitul sumelor parţiale $y_i^k$, cu $k < P$) în P impulsuri.

Timpul total de calcul va fi deci (N + P) impulsuri.

### 3.1.5. A treia soluţie sistolică

Al treilea SFG (fig. 3.5. c)) corespunde unei proiecţii după următorii vectori:

$$\mathbf{d} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad \text{şi} \qquad \mathbf{s} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Particularitatea acestei soluţii constă în faptul că nodurile SFG nu conţin date rezidente. Graful SFG este compus din (N + P - 1) noduri. Componentele vectorilor **x** şi **y** circulă în direcţii opuse. Componentele matricii **W** traversează reţeaua pe o direcţie ortogonală acestei circulaţii. Calculul se derulează în 3 faze:

1. Introducerea vectorului **x** pe plasamentul corespunzător iniţierii calculelor (N impulsuri);
2. Calculul propriu-zis care durează N + P - 1 impulsuri pentru fiecare componentă;
3. Extragerea ultimei componente din vectorul **y** (P - 1 impulsuri).

Timpul total de calcul va fi deci 2(N + P - 1) impulsuri. Acest SFG nu este avantajos, nici în spațiu și nici în timp, pentru calcule care implică matrici **W** complet populate (cu elemente nenule). Soluția devine interesantă pentru matricile cu o structură de tip bandă diagonală, datorită proiecției pe diagonală.

### 3.1.6. A patra soluție sistolică

Respectând ipoteza unei secvențieri liniare mai rămâne o soluție pe care o vom detalia pe larg în cele ce urmează. Aceasta corespunde unei asocieri directe între DG și SFG, ceea ce înseamnă că vectorii de proiecție vor fi:

$$\mathbf{d} = \mathbf{s} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Vectorul **x** este introdus din direcția Nord în rețea și va traversa celulele sistolice dispuse într-o grilă de formă rectangulară (fig. 3.6).

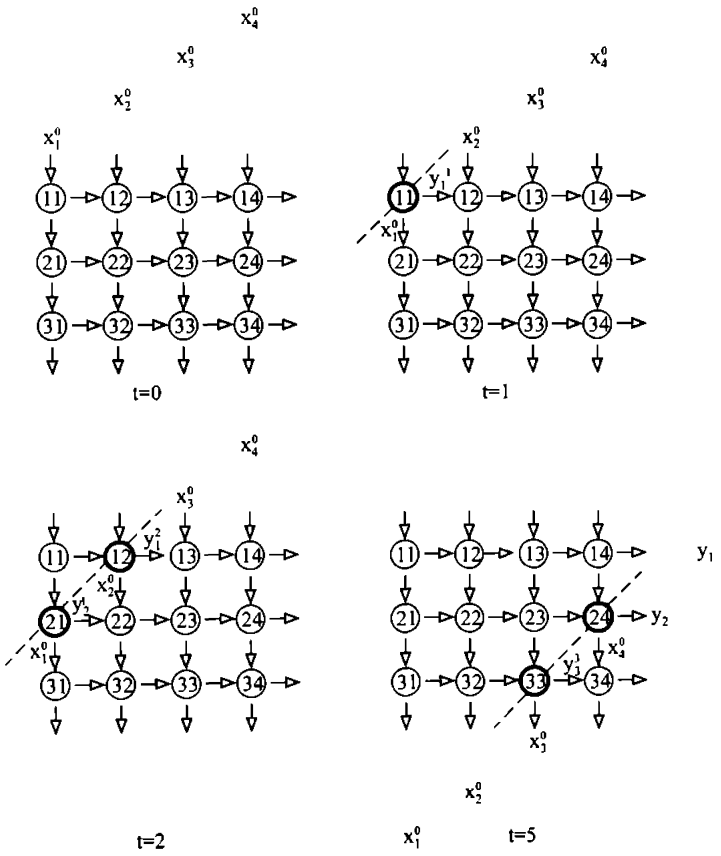



Fig. 3.6. Rețea sistolică (RS) bidimensională

Sumele parţiale tranzitează arcele dintre celulele dispuse pe aceeaşi linie şi ies din reţea (sub formă de sume finale care reprezintă componentele vectorului **y**) pe direcţia Est. Considerăm acelaşi caz general în care matricea **W** are dimensiunea [P, N], vectorul **x** are dimensiunea N şi vectorul **y** dimensiunea P. Timpul de calcul pentru produsul $\mathbf{W} \cdot \mathbf{x}$ se descompune astfel:

- N unităţi de timp pentru încărcarea matricii **W**.
- N unităţi de timp pentru încărcarea vectorului **x**.
- P unităţi de timp pentru ieşirea vectorului **y**.

Ieşirea vectorului **y** se face în paralel cu ieşirea vectorului **x**.
În total, vor fi necesare 2N + P unităţi de timp pentru obţinerea vectorului rezultat **y**.

Timpul total de calcul este deci superior comparativ cu soluţiile anterioare. În ceea ce priveşte lărgimea busurilor de date, regăsim inconvenientul asociat primei soluţii sistolice, şi anume acela că lărgimea busurilor pe care circulă sumele parţiale depinde de numărul de celule sistolice de pe traseu. La o primă privire, această soluţie pare costisitoare în ceea ce priveşte suprafaţa ocupată deoarece ridică numărul de CS la o valoare egală cu numărul de ponderi sinaptice din matricea **W**. Totuşi, după cum vom vedea în cele ce urmează, performanţele cresc spectaculos în cazul execuţiei unor produse succesive între matricea sinaptică **W** şi un set de vectori de intrare succesivi $\mathbf{x}^{[k]}$, datorită *pipeline*-izării acestor operaţii. Acest aspect este extrem de interesant pentru algoritmii conexionişti în care se execută produse succesive între matricea sinaptică **W** şi vectorii de intrare $\mathbf{x}^{[k]}$. Aceasta va conduce şi la distribuirea costului încărcării matricii **W** pe un număr mare de produse efectuate; la nivel global, costul încărcării matricii **W** devine neglijabil.

Pentru toate tipurile de SFG cu secvenţiere sistolică (anterior descrise), se pot deriva direct reţele sistolice (RS) ale căror performanţe şi constrângeri legate de integrare variază. Aceasta presupune că proiectantul, care vizează realizarea unui circuit VLSI, va selecta arhitectura pe care o va implementa în funcţie de limitările tehnologice pe care le cunoaşte. Vom trece în revistă câteva dintre acestea şi vom analiza incidenţa lor asupra fazei de alegere a RS adecvate implementării VLSI.

### 3.1.7. Implantabilitatea reţelelor sistolice

O reţea sistolică este formată dintr-o structură regulată de celule sistolice identice. Realizarea RS pretinde deci un efort care se concentrează asupra definirii şi optimizării unei singure celule. Reţeaua va fi apoi formată prin multiplicarea liniară sau bidimensională a acestui element de bază. Schema de interconexiuni, dictată de poziţia punctelor de intrare / ieşire în CS, este locală exceptând interconexiunile aferente semnalelor de sincronizare. Aceasta conduce la a doua proprietate a RS: conexiunile locale facilitează implantabilitatea. Până în prezent, comunicaţiile au fost considerate ca un element de cost mic în concepţia procesoarelor clasice. Realizarea unor structuri masiv paralele infirmă acest punct de vedere deoarece costul interconexiunilor din structura VLSI devine superior celui aferent elementelor active. Trebuie deci localizate schimburile de date dintre elementele active, ceea ce este în mod natural realizat în RS.

Granularitatea, adică numărul de elemente de procesare (EP) utilizate pentru calcul reprezintă de asemenea un factor esenţial pentru implementarea reţelelor sistolice. Acesta rezultă în general dintr-un compromis între constrângerile de intrare / ieşire, contextul procesării în interiorul RS şi tehnologia utilizată.

Robusteţea RS reprezintă de asemenea un factor care trebuie luat în considerare. Pentru o reţea compusă din N elemente EP de robusteţe r, robusteţea reţelei va fi $r^N$. Cum r este întotdeauna mai mic decât 1, robusteţea de ansamblu aferentă unui sistem sistolic este foarte slabă. Totuşi,

utilizată în contextul algoritmilor conexioniști, RS poate fi perfect adaptată pentru a beneficia de robustețea intrinsecă a modelelor conexioniste.

Puterea de calcul nu este singurul factor care definește eficiența unei arhitecturi sistolice. Această putere trebuie considerată în contextul unui sistem (sistolic) care schimbă date cu un sistem gazdă. Lărgimea de bandă aferentă canalului de comunicație cu sistemul gazdă reprezintă un parametru esențial care stabilește o limită superioară pentru puterea de calcul a sistemului luat în ansamblu. Trebuie deci aleasă o arhitectură care să permită echilibrarea timpului de calcul cu cel de comunicație (intrare / ieșire din RS).

În sfârșit, partiționabilitatea unei aplicații mari pe RS trebuie luată în considerare pentru a realiza RS de mărime rezonabilă (adaptată necesităților).

În cele ce urmează vom descrie 4 RS capabile să calculeze produsul matrice - vector. Vom alege o structură care nu impune partiționarea acestui produs, adică numărul de EP va fi egal cu pătratul numărului de componente aferente vectorului înmulțitor. Această constrângere va sta la baza tuturor dezvoltărilor pe care le vom lua în considerare în cele ce urmează.

### 3.2. Arhitecturi de RS destinate algoritmilor conexioniști

După cum am menționat deja, produsul matrice - vector este operația de bază, comună tuturor modelelor conexioniste. În faza de recunoaștere acest produs apare fie individual, fie ca operație intermediară conjugată cu alte operații matriciale sau vectoriale. În ipoteza unei secvențieri sistolice, pornind de la SFG-urile descrise anterior, vom deduce 4 tipuri de RS. Pentru fiecare tip vom analiza implementabilitatea, extensibilitatea și raportul performanțe / complexitate.

Reluând notațiile utilizate pentru descrierea modelelor conexioniste, vom defini câțiva termeni adecvați descrierii datelor care intră într-un calcul sistolic:

**x** - vectorul de intrare în rețea, de dimensiune [N]
**y** - vectorul de ieșire, de dimensiune [P]
**W** - matricea sinaptică, de dimensiune [P, N]
**PS** - vectorul de sume parțiale care tranzitează conexiunile dintre celulele sistolice

Să notăm că, în cazul rețelei *Hopfield*, nu vom diferenția între intrare și ieșire și ne vom referi doar la starea celulelor. Aplicarea unui vector de intrare **x** revine la a forța starea celulelor la valoarea **x**, iar ieșirea vectorului **y** revine la citirea acestei stări.

Arhitecturile sistolice dedicate produsului matrice - vector pe care urmează să le prezentăm nu vor fi toate foarte bine adaptate pentru implementarea algoritmilor conexioniști. Ele nu iau în considerare secvențierea specifică a calculelor executate în cadrul algoritmilor conexioniști. Cascadarea (înlănțuirea) produselor matrice - vector este specifică, de exemplu, fazei de recunoaștere aferentă rețelei *Hopfield*. Accesibilitatea la elementele matricii **W** (ponderi sinaptice) trebuie asigurată dacă dorim să implementăm și faza de învățare. Vectorul rezultat din produsul **W**· **x** trebuie transformat apoi printr-o funcție neliniară (funcția de activare). Operațiile de intrare / ieșire în / din RS trebuie studiate pentru a obține o interfață a cărei lărgime de bandă să reprezinte optimul dintre viteza de procesare a RS și cea a procesorului gazdă. Toate aceste probleme reprezintă constrângeri conjugate, proprii fazei de concepție a unui circuit specializat dedicat unei clase de aplicații, sau a unui coprocesor studiat în mediul său de lucru.

Vom relua sistematic soluțiile sistolice prezentate anterior și vom studia arhitecturile care vor putea fi exploatate pentru cele două operații esențiale ale calculului conexionist: faza de recunoaștere și faza de învățare. Recunoașterea, după cum am subliniat și în capitolele precedente, este identică pentru toate modelele conexioniste. Este un produs matrice - vector unic (*Adaline*, Perceptroni mono și multistrat etc.) sau iterativ (*Hopfield*, *Boltzmann* etc.). Învățarea va fi evaluată

din punctul de vedere al accesibilităţii la coeficienţii sinaptici şi la operaţiile necesare pentru calculul valorilor de ajustare aferente acestor coeficienţi.

### 3.2.1. Arhitectura generală

Arhitectura generală pe care o vom studia realizează faza de recunoaştere aferentă reţelei *Hopfield*. Reamintim că acest model reclamă un calcul iterativ al produselor dintre un vector de stare şi matricea de ponderi sinaptice. După fiecare înmulţire, vectorul rezultat suferă o transformare neliniară (funcţia de transfer) generând astfel noul vector de stare al reţelei *Hopfield*. Acesta este din nou înmulţit cu matricea sinaptică şi operaţiile se repetă până la convergenţă (egalitate între 2 vectori de stare succesivi).

Formal, arhitectura generală trebuie să permită execuţia celor 3 operaţii expuse:

- produse matrice - vector iterative

$$\mathbf{p}(t+1) = \mathbf{W} \cdot \mathbf{x}(t) \tag{3.3}$$

- aplicarea unei funcţii neliniare (funcţia de activare)

$$\mathbf{x}(t+1) = \sigma(\mathbf{p}(t+1)) \tag{3.4}$$

- detecţia convergenţei

$$\mathbf{x}(t+1) = \mathbf{x}(t) \tag{3.5}$$

O schemă generală a acestei arhitecturi este prezentată în figura 3.7.

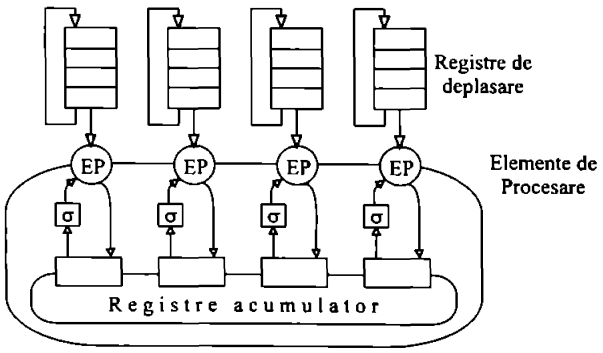



Fig. 3.7. Arhitectura generală în inel

Am definit această arhitectură generală pornind de la analiza RS liniare descrise anterior. Am completat legăturile de date aferente reţelelor elaborate cu rebuclaje care vor permite cascadarea produselor $\mathbf{W} \cdot \mathbf{x}$ succesive. În figura 3.7 sunt reprezentate toate căile de date posibile: comunicaţiile între EP, între registrele acumulator şi între registrele de deplasare care memorează ponderile sinaptice.

Arhitecturile care vor deriva din această formă generală se vor caracteriza prin amplasări specifice ale acestor căi de comunicaţie. Le vom referi prin menţionarea operanzilor ficşi (rezidenţi).

Trei arhitecturi implementabile vor fi propuse: cu sume parţiale fixe (fără căi de comunicaţie între registrele acumulator), cu componentele vectorului fixe (fără căi de comunicaţie între procesoarele EP) şi cu coeficienţi sinaptici ficşi (fără căi de date între coeficienţii matricii **W**). Să notăm, cu titlu de generalitate, că este posibilă de asemenea o soluţie fără operanzi ficşi. Aceasta n-o vom descrie datorită faptului că presupune un volum de transferuri de date comparabil cu cel al calculelor efectuate. În plus, soluţia în discuţie nu prezintă eficienţă în cazul algoritmilor conexionişti.

Operatorul funcţiei neliniare $\sigma$ este plasat pe calea de comunicaţie dintre fiecare EP şi acumulatorul său. Acest amplasament rezultă dintr-o transcriere directă a ecuaţiei (3.2) care specifică faptul că funcţia neliniară se aplică vectorului rezultat din produsul $\mathbf{W} \cdot \mathbf{x}$; fiecare registru acumulator va conţine o componentă a acestui vector rezultat.

Resursele nodurilor reţelei sistolice (celulelor sistolice) sunt specificate în figura 3.8. Această structură respectă de asemenea secvenţierea sistolică definită. Singurul element suplimentar care specializează această structură pentru algoritmii conexionişti este funcţia neliniară $\sigma$.

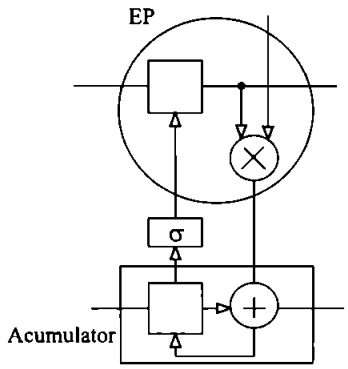



Fig. 3.8. Structura celulei sistolice

În sfârşit, pentru transferurile de date sunt posibile 2 tehnici: transmisia serială sau paralelă. Soluţia serială reduce numărul de linii de comunicaţie dintre celulele sistolice dar conduce la creşterea timpilor de încărcare şi vidare a inelului sistolic. Soluţia paralelă reduce timpii de comunicaţie dar creşte numărul de conexiuni. Vom detalia în capitolele următoare avantajele şi inconvenientele fiecărei soluţii.

Mai rămâne de stabilit repartiţia coeficienţilor sinaptici în registrele de deplasare. Această repartiţie depinde exclusiv de datele pe care vrem să le tranzităm între nodurile RS. În cele ce urmează, vom definii diferitele repartiţii posibile şi vom deriva arhitecturile rezultate.

### 3.2.2. Inel sistolic cu componentele vectorului fixate

Datele mobile vor fi sumele parţiale şi componentele matricii **W**. Componentele matricii **W** trebuie repartizate în registrele de deplasare, astfel încât să obţinem secvenţierea sistolică definită. Componentele vectorului **x** sunt rezidente în fiecare nod.

Dispunerea coeficienţilor sinaptici depinde de dispunerea componentelor vectorului (în timpul procesului de calcul). În faza de iniţializare, componentele vectorului **x** vor fi introduse secvenţial în inelul sistolic iar coeficienţii matricii **W** în registrele de deplasare. În faza de calcul celulele sistolice vor recepţiona componentele matricii simultan. Decalajul de prezentare a

coeficienţilor matricii **W** impus de algoritmul sistolic trebuie implementat printr-o repartiţie corespunzătoare a acestor coeficienţi în registrele de deplasare, aşa cum indică figura 3.9.

Pe fiecare impuls, cele N produse parţiale sunt calculate şi acumulate în memoriile locale (registrele acumulator). Sumele parţiale PS sunt apoi transmise între nodurile reţelei pentru o nouă acumulare.

Dacă luăm în considerare numai faza de calcul, vor fi necesare N operaţii de înmulţire şi de transmisie a sumelor parţiale, urmate de conversia rezultatelor obţinute cu ajutorul operatorilor care implementează funcţia neliniară $\sigma$.

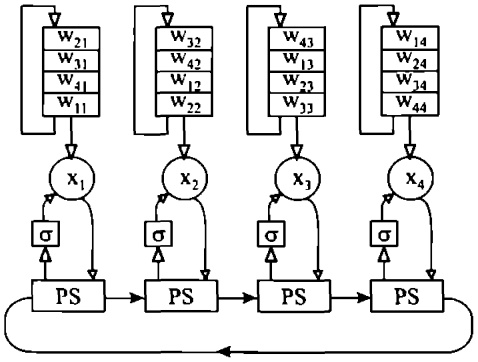



Fig. 3.9. Inel sistolic cu componentele vectorului fixe

Coeficienţii matricii **W** sunt deplasaţi în registrele de deplasare şi, după N impulsuri, se vor regăsi în poziţia iniţială. De asemenea, componentele vectorului rezultat se regăsesc în poziţia corectă, necesară declanşării unui nou produs.

În cadrul acestei soluţii, componentele vectorului de multiplicat sunt rezidente în fiecare CS. Ele nu vor fi afectate în timpul înmulţirilor şi deci reprezentarea lor (numărul de biţi necesari pentru reprezentare) nu va varia. Sumele parţiale PS care tranzitează între celulele sistolice vor avea o reprezentare variabilă, care creşte în funcţie de numărul de celule traversate. Pentru o reţea cu N celule, fiecare ieşire $y_i$ se va obţine prin însumarea a N produse $w_{ij} \cdot x_j$. Presupunem că există o valoare maximă stabilită pentru componentele $x_j$ şi respectiv $w_{ij}$. Numărul de biţi necesari pentru reprezentarea acestora va fi:

$$Tmax(x_j) = Log_2(Max[x_j])$$
$$Tmax(w_{ij}) = Log_2(Max[w_{ij}])$$

Numărul maxim de biţi necesari reprezentării componentelor vectorului rezultat va fi:

$$Tmax(y_i) = Log_2(N) + Tmax(x_j) + Tmax(w_{ij}) \qquad (3.6)$$

Transferul sumelor parţiale între celule trebuie realizată printr-o comunicaţie care trebuie să ţină cont de relaţia de mai sus. Avem la dispoziţie mai multe soluţii:

- calculul lăţimii căii de comunicaţie dintre celule (numărul de biţi necesari) pe baza relaţiei de mai sus şi construcţia unei arhitecturi fixe

- fixarea unei căi de comunicaţie a cărei lăţime (număr de biţi) care depinde de valoarea maximă admisibilă pentru PS. Această dimensiune fixă limitează, de asemenea, numărul maxim de celule din RS, dacă dorim evitarea depăşirilor de capacitate de reprezentare

- implementarea unei transmisii seriale (bit după bit) a sumelor PS, tehnică ce introduce noţiunea de *bit level systolic array* [KNO 88]. Vom distinge în acest caz două nivele de *pipelining*, primul situat la nivelul celulelor sistolice şi al doilea la nivelul comunicaţiei dintre celule. Să notăm că dimensiunea registrului destinat memorării componentei PS în celula sistolică va reprezenta de asemenea o limitare dar această dimensiune poate fi fixată la o valoare acoperitoare.

Inelul sistolic cu componentele vectorului fixate este teoretic extensibil. Lungimea reţelei poate fi mărită prin simpla adăugare a unor noduri suplimentare. Este, de asemenea, posibilă implementarea unor produse ample, în care numărul de CS este inferior numărului de componente vectoriale; este suficientă completarea reţelei cu un dispozitiv destinat acumulării produselor parţiale.

Timpul de încărcare a matricii sinaptice se evaluează în funcţie de tipul comunicaţiei implementate între RS şi sistemul gazdă. Dacă accesul la RS este secvenţial, vor fi necesare $N^2$ cicluri de încărcare şi un singur port de acces (matricea **W** este o matrice pătrată de dimensiune [N x N] în cazul reţelei *Hopfield*). Dacă accesul este paralel vor fi necesare N cicluri de încărcare şi N porturi de acces (această remarcă va fi valabilă şi pentru inelul sistolic cu sume parţiale fixe care va fi prezentat în secţiunea 2.3.2).

Componentele vectorului înmulţitor sunt localizate în fiecare celulă sistolică. Încărcarea sau citirea acestui vector se poate face fie secvenţial, fie paralel. În cazul paralel sunt necesare N porturi de intrare / ieşire, dar timpul de acces se reduce la un ciclu.

Dacă luăm în considerare comunicaţia paralelă, timpul total de procesare va fi:

N cicluri pentru încărcarea matricii **W**
1 ciclu pentru încărcarea vectorului **x**
k · N cicluri de calcul, k reprezentând numărul de iteraţii necesare atingerii unei stări stabile (convergenţei)
k cicluri pentru aplicarea funcţiei neliniare $\sigma$
1 ciclu pentru citirea vectorului rezultat

Rezultă un total de N + k(N + 1) + 2 cicluri de procesare, dacă presupunem că toate operaţiile revendică cicluri de durate identice.

Acest tip de arhitecturi sistolice este destinat implementării reţelelor de dimensiuni mari datorită proprietăţii de extensibilitate. Performanţele canalului de comunicaţie dintre RS şi calculatorul gazdă sunt esenţiale din punctul de vedere al performanţelor globale deoarece N componente trebuie transferate simultan.

În cazul comunicaţiei seriale, timpii de încărcare cresc considerabil, dar teoretic sunt suficiente 2 porturi de intrare / ieşire pentru orice dimensiune de RS. Timpul total de procesare se evaluează astfel:

$N^2$ cicluri pentru încărcarea matricii **W**
N cicluri pentru încărcarea vectorului **x** în celulele sistolice
k · N cicluri de calcul, k reprezentând numărul de iteraţii necesare atingerii unei stări stabile
k cicluri pentru aplicarea funcţiei neliniare $\sigma$
N cicluri pentru citirea vectorului rezultat

Rezultă un total de $N^2 + (k + 2)N + k$ cicluri de procesare, presupunând că toate operaţiile necesită cicluri identice (ca durată).

Soluţia secvenţială este avantajoasă deoarece decongestionează comunicaţia cu sistemul gazdă. Ea revendică însă o cale de comunicaţie suplimentară între elementele EP (pentru încărcarea vectorului **x**) care vine să dubleze pe cea existentă între acumulatoare; aceasta conduce la un cost suplimentar în ceea ce priveşte suprafaţa de integrare ocupată. Este preferabilă alegerea unei arhitecturi de RS care conţine deja o cale de comunicaţie între elementele EP (necesară pentru calcule), cale care va fi utilizată şi pentru operaţiile de intrare / ieşire.

### 3.2.3. Inel sistolic cu sume parţiale fixe

Această arhitectură presupune circulaţia componentelor vectorului **x** între elementele EP; aşa cum am subliniat mai sus, va fi o arhitectură bine adaptată operaţiilor de comunicaţie cu sistemul gazdă. Din aceleaşi motive prezentate la 3.2.2, repartiţia elementelor matricii **W** în registrele de deplasare se va realiza ca în figura 3.10.

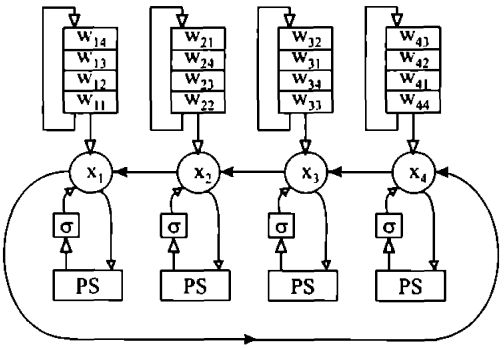



Fig. 3.10. Inel sistolic cu vector mobil şi sume parţiale fixe

În această arhitectură, componentele vectorului **x** se deplasează între elementele EP şi, în interiorul fiecărui EP, sunt înmulţite cu coeficienţii matricii **W** prezenţi pe intrări. Sumele parţiale sunt acumulate în interiorul EP. După N cicluri de calcul / transmisie, sumele parţiale acumulate sunt transformate prin funcţia neliniară σ (implementată în fiecare EP). Vectorul rezultat este astfel gata pentru execuţia următorului produs $\mathbf{W} \cdot \mathbf{x}$. La finele operaţiilor (convergenţă) componentele vectorului de stare sunt extrase secvenţial din RS şi încărcate în sistemul gazdă.

Această arhitectură, în termeni de implantabilitate, este mai bună decât precedenta. Datele mobile sunt aici elementele matricii **W** şi componentele vectorului **x**. Aceste componente nu sunt modificate pe parcursul calculelor şi prin urmare dimensiunea (numărul de biţi alocaţi) nu variază. Să notăm că elementele vectorului **x** se modifică periodic dar în urma aplicării funcţiei σ; această funcţie neliniară restrânge valoarea componentelor la domeniul de reprezentabilitate, care este domeniul în care ia valori σ ([-1, +1]). În consecinţă, căile de comunicaţie între EP au o lăţime fixă indiferent de lungimea inelului sistolic. Registrele de acumulare pot fi dimensionate pornind de la dimensiunea operanzilor, aşa cum s-a arătat în paragraful precedent.

Dacă reţeaua este constituită din N celule, fiecare registru PS va conţine componenta finală a vectorului rezultat după traversarea a N componente $x_j$ prin EP-ul respectiv. Dimensiunea registrului PS va fi calculată astfel încât acesta să poată memora cel mai mare rezultat posibil:

$$\text{Dimensiune(PS)} = \text{Log}_2(y_{imax}) = \text{Log}_2(N) + \text{Tmax}(x_j) + \text{Tmax}(w_{ij}) \qquad (3.7)$$

Pentru această arhitectură, vom considera că încărcarea şi citirea datelor se face secvenţial. Timpul total de procesare va conţine:

$N^2$ cicluri pentru încărcarea matricii **W**
N cicluri pentru încărcarea vectorului **x**
k · N cicluri de calcul, k reprezentând numărul de iteraţii până la convergenţă
k cicluri de aplicare a funcţiei neliniare σ
N cicluri pentru citirea vectorului rezultat

Rezultă un total de $N^2 + N(k + 2) + k$ cicluri, dacă presupunem că toate operaţiile revendică cicluri identice (ca durată) pentru execuţie. Timpul de procesare este identic cu cel obţinut pentru arhitectura anterioară (cazul comunicaţiei seriale). Această arhitectură este însă mai bună din 2 motive:

- schimburile de date cu sistemul gazdă sunt favorizate de comunicaţia dintre EP-uri
- căile de comunicaţie dintre EP-uri au lăţimea fixă deoarece datele vehiculate nu depăşesc domeniul de reprezentare stabilit. Funcţia neliniară σ este o funcţie de compresie care comprimă valorile într-un interval fixat. Se poate deci determina în avans lăţimea căilor de comunicaţie; aceasta rămâne fixă indiferent de lungimea inelului sistolic.

Punctele slabe care revin acestei arhitecturi sunt:

- extensibilitate dificilă datorată registrelor de deplasare. Elementele matricii **W** sunt stocate în aceste registre. Lungimea acestor registre (numărul de locaţii disponibile pentru elementele $w_{ij}$) trebuie să fie egală cu numărul de noduri din RS. Ori, dacă dorim o reţea extensibilă, numărul de noduri nu va fi cunoscut a priori.
- dimensiunea acumulatoarelor (numărul de biţi) depinde de dimensiunea RS (numărul de noduri). Dacă dorim o precizie ridicată de reprezentare a sumelor parţiale, trebuie prevăzută o dimensiune corespunzătoare pentru acumulator. Această dimensiune trebuie definită la construcţia RS şi va limita ulterior extinderea inelului sistolic.

Această soluţie a fost propusă şi de *Weinfield* [WEI 89] pentru integrarea unei reţele sistolice liniare dedicată fazei de recunoaştere aferentă modelului *Hopfield*.

Cea mai importantă limitare a sistemelor sistolice liniare prezentate constă în extensibilitatea limitată datorită atribuirii unui număr relativ mare de coeficienţi sinaptici fiecărui EP. O reţea sistolică bidimensională poate depăşi această limitare deoarece coeficienţii matricii **W** devin locali (câte unul alocat pe fiecare EP). Suprafaţa totală va creşte desigur deoarece elementele EP vor fi multiplicate. Acest cost suplimentar va fi fără îndoială compensat de posibilităţile de extindere precum şi de puterea de calcul mult crescută graţie unui *pipelining* vectorial. Această soluţie urmează a fi prezentată.

### 3.2.4. Arhitectura cu coeficienţi sinaptici ficşi

În această reţea bidimensională, numărul de elemente EP va fi egal cu numărul de coeficienţi sinaptici din matricea **W**. Sumele parţiale şi componentele vectorului **x** vor fi mobile pentru a realiza două funcţii:

- punerea la dispoziţie a componentelor $x_i$ necesare în fiecare linie a reţelei sistolice

- transmiterea rezultatelor parţiale de-a lungul liniilor RS.

Transmisia datelor este descrisă în figura 3.11.

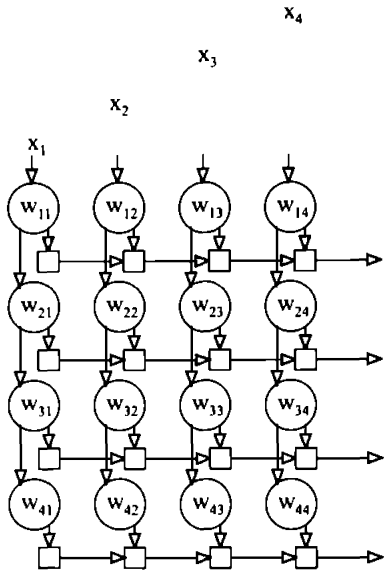



Fig. 3.11. Arhitectura reţelei sistolice bidimensionale

Componentele vectorului **x** sunt introduse secvenţial în reţea. Produsul local este calculat imediat ce o componentă $x_i$ este introdusă într-un EP. Produsele locale sunt apoi transmise celulelor vecine pe direcţia est şi acumulate la nivelul fiecărei celule. În momentul în care ultima componentă a vectorului **x** părăseşte reţeaua pe direcţia sud, ultima componentă a vectorului rezultat ($\mathbf{W} \cdot \mathbf{x}$) va părăsi reţeaua pe direcţia est. Apare deci o dublă mişcare a datelor:

- componentele vectorului **x** tranzitează reţeaua dinspre nord spre sud între EP-urile aceleiaşi coloane
- sumele parţiale tranzitează reţeaua de la vest spre est între EP-urile aceleiaşi linii.

Pentru a avea celule identice indiferent de amplasamentul lor în cadrul RS, dimensiunea registrelor PS trebuie să permită memorarea celui mai mare rezultat posibil:

$$\text{Dimensiune PS} = \text{Log}_2(N) + \text{Tmax}(x_j) + \text{Tmax}(w_{ij}) \qquad (3.8)$$

Pentru această arhitectură vom considera că încărcarea şi citirea datelor în / din RS se face secvenţial, dar încărcarea matricii **W** se face în paralel. Timpul de calcul va include:

N cicluri pentru încărcarea matricii **W**
N cicluri pentru încărcarea vectorului **x**
k · N cicluri de calcul, k reprezentând numărul de iteraţii până la convergenţă
k cicluri de aplicare a funcţiei neliniare
N cicluri pentru citirea vectorului rezultat

Rezultă un timp de procesare total de k + N(k + 3) cicluri, dacă presupunem că toate operaţiile revendică cicluri identice (ca durată) pentru execuţie.

Această arhitectură prezintă 2 avantaje:

- viteză ridicată în raport cu reţelele sistolice liniare deoarece permite procesarea vectorilor de intrare în tehnică *pipeline*
- extensibilitate. Fiecare EP conţine un singur coeficient sinaptic. Este suficientă extinderea pavajului bidimensional pentru extinderea reţelei.

Constrângerile referitoare la lăţimea căilor de date şi cele referitoare la dimensiunea acumulatoarelor care servesc la tranzitarea sumelor parţiale rămân în vigoare şi sunt identice cu cele din cadrul arhitecturilor precedente.

Inconvenientul major constă în suprafaţa ocupată de reţeaua bidimensională. În plus, gradul de utilizare a celulelor sistolice care este 100 % în cazul liniar, în cazul bidimensional şi pentru această soluţie elementară, depinde de momentul evaluării şi este maxim 50 %. Această limitare este compensată de posibilitatea *pipeline*-izării fluxului de vectori de intrare, ceea ce conduce la un grad de utilizare de 100 % în toate momentele de timp. Vom detalia această *pipeline*-izare în capitolele următoare. În sfârşit, vectorul **x** este introdus de pe direcţia nord şi noua sa valoare se obţine pe latura de est a RS. Dacă dorim înlănţuirea mai multor produse matrice - vector pe RS diferite (succesive), va trebui studiată topologia de implementare în VLSI pentru a liniariza transferurile de date.

### 3.3. Concluzii

Dintre cele 3 soluţii prezentate vom reţine reţeaua bidimensională rectangulară din mai multe motive:

În primul rând, datorită posibilităţilor de extensie care permit construirea unor reţele mari. Mărirea reţelei se face printr-o extensie a pavajului bidimensional (de celule) şi această extensie nu afectează resursele celulei sistolice. Fiecare celulă memorează o singură pondere sinaptică. Creşterea numărului de sinapse va fi egală cu pătratul numărului de neuroni adăugaţi. Creşterea suprafeţei ocupate de o arhitectură bidimensională este de asemenea o funcţie pătratică. Raportul (număr sinapse) / (suprafaţa ocupată) rămâne constant ceea ce indică faptul că extensibilitatea nu va fi limitată de un factor de suprafaţă disponibilă.

În al doilea rând, principiul distribuţiei calculului pe un număr mare de procesoare corespunde mult mai bine principiului conexionist. Robusteţea furnizată de modelele conexioniste poate fi mai bine exploatată printr-o amenajare adecvată a unor linii de reconfigurare care vor permite izolarea şi înlocuirea celulelor defecte.

În al treilea rând, soluţia sistolică permite efectuarea calculelor cu o precizie suficientă pentru majoritatea algoritmilor conexionişti. Tehnologia actuală permite integrarea unui număr suficient de celule sistolice cu un randament ridicat.

În sfârşit, după cum vom constata în cele ce urmează, această arhitectură bidimensională permite implementarea unor mecanisme mai complexe, proprii algoritmilor conexionişti: detecţie de convergenţă, gestiune eficientă a operaţiilor de intrare / ieşire, învăţare după reguli multiple, interconectarea subreţelelor etc.

## 4. PROIECTAREA RS DEDICATE ALGORITMILOR CONEXIONIŞTI

În acest capitol vom detalia funcţionarea reţelei sistolice bidimensionale obţinută în capitolul anterior pentru produsul matrice - vector. Vom arăta că există un flux de date şi o secvenţiere care permit înlănţuirea unor astfel de produse matrice - vector; funcţia neliniară σ se va aplica tuturor componentelor vectorului rezultat. Aceste operaţii reprezintă esenţa majorităţii algoritmilor conexionişti.

### 4.1. Soluţia cu valori de activare şi sume parţiale mobile

Reţeaua bidimensională obţinută în cap. 3 alocă un EP per sinapsă. Pentru un vector cu N componente, reţeaua va conţine $N^2$ sinapse cărora li se vor aloca $N^2$ procesoare EP. Fiecare EP va conţine un singur coeficient sinaptic şi va acumula rezultatele parţiale pe care le-am denumit sume parţiale (PS). Pentru RS liniare, am arătat că este posibilă înlănţuirea produselor matrice - vector şi aplicarea funcţiei neliniare fiecărei componente din vectorul rezultat, dacă se repartizează corect coeficienţii matricii **W** în interiorul fiecărei celule şi dacă se realizează un rebuclaj al reţelei sistolice liniare.

Circulaţia datelor în interiorul RS bidimensionale este descrisă în figura 4.1. Se poate constata că, iniţial, vectorul înmulţitor este introdus de pe direcţia Nord în reţea şi vectorul rezultat este emis pe direcţia Est din reţea.

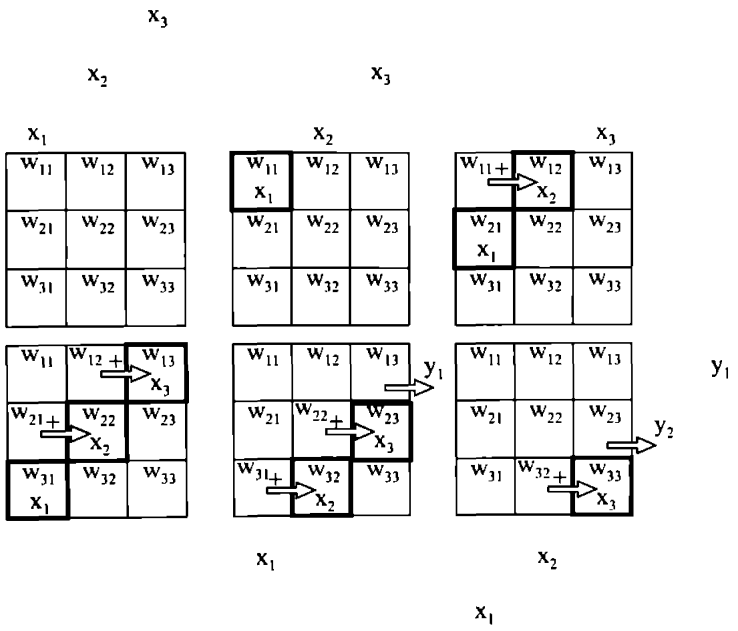



Fig. 4.1. Circulaţia datelor în RS bidimensională

Datele mobile sunt:

- Componentele vectorului **x** care se deplasează dinspre Nord spre Sud;
- Sumele Parţiale (PS) care se deplasează dinspre Vest spre Est.

Fiecare celulă sistolică (i, j) recepţionează componenta $x_j$ a vectorului **x** în momentul (t), calculează produsul local $w_{ij} \cdot x_j$ şi îl adaugă sumei parţiale $PS_i$ recepţionată din direcţia vest. Rezultatul acestei acumulări este transmis celulei vecine de pe direcţia Est. Dacă vectorul **x** are N componente, pentru a obţine fiecare componentă a vectorului **y** sunt necesare N operaţii de tipul celei descrise anterior, şi fiecare componentă se calculează cu un decalaj de o unitate de timp faţă de cea precedentă. Vectorul **y** va fi obţinut deci după N unităţi de timp. Vom denumi pas de recurenţă etapa care constă în calculul unui produs $\mathbf{W} \cdot \mathbf{x}$.

În momentul în care o componentă a vectorului rezultat ($y_i$) apare pe latura de Est a RS, se poate aplica funcţia neliniară $\sigma$. Funcţia de activare $\sigma$ nu va fi integrată în interiorul RS; ea se va calcula cu ajutorul unor operatori specializaţi plasaţi pe latura de est a RS. Apoi, vectorul rezultat **y** va trebui reintrodus în RS ca vector de intrare **x**, pentru calculul următorului produs $\mathbf{W} \cdot \mathbf{x}$. Acest rebuclaj se poate realiza cu ajutorul unor căi de date care leagă latura de Est cu latura de Nord, aşa cum se arată în fig. 4.2.

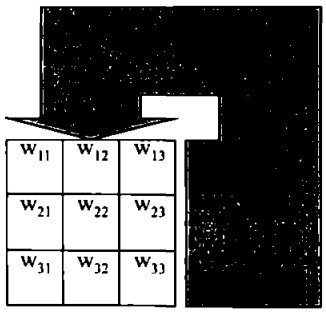



Fig. 4.2. Rebuclajul datelor pe intrarea RS

Nemaifiind o transmisie locală, această soluţie distruge omogenitatea transmisiilor de date şi poate genera timpi suplimentari de propagare care vor afecta performanţele globale ale RS. Este preferabil să se găsească o soluţie care respectă regula de transmisie la cel mai apropiat vecin, fără legături la distanţe mari.

### 4.2. O soluţie care implică transmisii locale de date

Vom discuta această soluţie asociată cu faza de recunoaştere pe o RN de tip *Hopfield*. Soluţia se bazează pe o circulaţie locală a datelor care respectă 3 reguli:

- (r1): Secvenţa operaţiilor de calcul este identică cu cea prezentată în paragraful 4.1.
- (r2): Fiecare celulă sistolică (CS) nu cunoaşte decât propria stare şi stările celor 4 celule vecine.
- (r3): La sfârşitul fiecărui pas de recurenţă, vectorul înmulţitor trebuie să se afle în aceeaşi poziţie iniţială în care se afla şi vectorul înmulţitor precedent (cel care l-a generat).

Ideea de bază se referă la poziţia iniţială a vectorului înmulţitor: acesta trebuie plasat pe diagonala principală a RS, aşa cum indică figura 4.3.

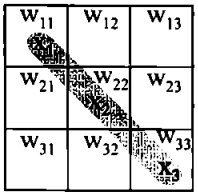



Fig. 4.3. Poziţia iniţială a vectorului **x**

Toate celulele sistolice sunt identice şi nu vor schimba informaţii decât cu celulele vecine (cei 4 vecini cei mai apropiaţi). Structura CS este prezentată în figura 4.4.

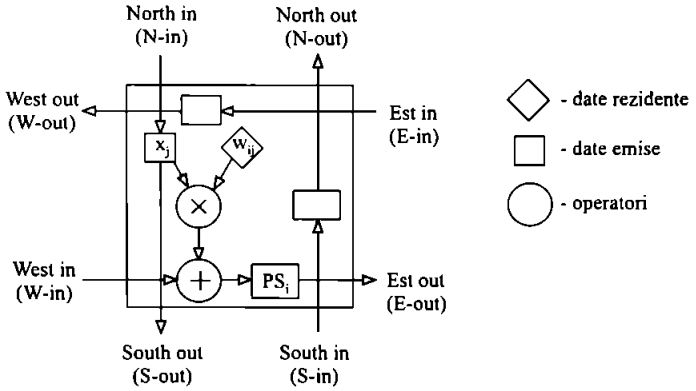



Fig. 4.4. Schema logică a unei CS

### 4.2.1. Derularea algoritmului

Algoritmul de calcul al unui vector rezultat (un pas de recurenţă) poate fi descompus în 3 faze succesive:

- elaborarea progresivă a sumelor parţiale
- aplicarea funcţiei neliniare
- repoziţionarea vectorului rezultat în poziţia iniţială pentru un nou pas de recurenţă

Vom prezenta derularea completă a unui pas de recurenţă pentru o matrice **W** de dimensiune 3 x 3.

**A. Prima etapă**

După cum am precizat deja, vectorul înmulţitor **x** va fi plasat iniţial pe diagonala RS. Respectând regula r1 nu vom putea efectua decât un singur produs, aşa cum se indică în figura 4.5.

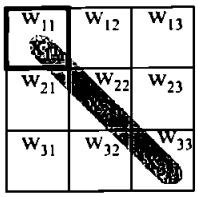



Fig. 4.5. Prima etapă (r - pasul de recurenţă)

Dacă notăm:

$PS_{i,j}$ - suma parţială prezentă în celula (i, j)

$x_{i,r}$ - componenta i a vectorului **x** în pasul de recurenţă r

vom calcula prima sumă parţială:

$$PS_{1,1} = w_{11} \cdot x_{1,1} \tag{4.1}$$

**B. A doua etapă**

În a 2-a etapă vor intra în joc celulele $w_{21}$ şi $w_{12}$. Vom propaga datele în RS astfel încât să le punem la dispoziţia acestor celule:

- coeficienţii matricii **W** sunt date rezidente; nu vor fi deplasaţi
- componentele vectorului **x** vor fi date emise şi se vor propaga astfel:
  - componenta $x_{1,1}$ va fi propagată spre sud; ea va fi astfel prezentă pentru calculul produsului şi sumei parţiale în celula $w_{12}$
  - componenta $x_{2,1}$ va fi propagată spre nord pentru a fi pusă la dispoziţia celulei $w_{21}$
  - componenta $x_{3,1}$ va fi propagată de asemenea spre nord pentru a o apropia de celula $w_{13}$ (anticipare)
- suma parţială $PS_{1,1}$ va fi transmisă spre est.

După aceste transmisii locale de date vor fi calculate produsele locale, care vor fi adăugate sumelor parţiale care intră în celule din direcţia vest:

$$PS_{1,2} = PS_{1,1} + w_{12} \cdot x_{2,1} \tag{4.2}$$

$$PS_{2,1} = w_{21} \cdot x_{1,1} \tag{4.3}$$

Figura 4.6. explicitează deplasarea datelor şi calculele efectuate.

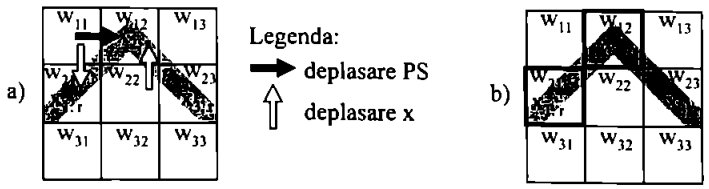



Fig. 4.6. A doua etapă. a) comunicaţii; b) calcul

La sfârşitul acestei etape, sumele parţiale $PS_{1,2}$ şi $PS_{2,1}$ vor fi prezente în celulele $w_{12}$ şi respectiv $w_{21}$.

**C. A treia etapă**

Datele mobile vor fi transmise ca în etapa precedentă. Calculele se vor efectua în celulele $w_{31}$, $w_{22}$ şi $w_{13}$.

- $x_{1,1}$ şi $x_{2,1}$ sunt propagate spre sud
- $x_{3,1}$ este propagat spre nord
- sumele parţiale $PS_{1,2}$ şi $PS_{2,1}$ sunt transmise spre est.

Vor fi calculate produsele locale şi vor fi acumulate sumelor parţiale care intră dinspre vest:

$$PS_{1,3} = PS_{1,2} + w_{13} \cdot x_{3,1} \tag{4.4}$$

$$PS_{2,2} = PS_{2,1} + w_{22} \cdot x_{2,1} \tag{4.5}$$

$$PS_{3,1} = w_{31} \cdot x_{1,1} \tag{4.6}$$

La sfârşitul acestei etape, prima componentă a vectorului rezultat (prima sumă finală) va fi prezentă în celula $w_{13}$ (de pe marginea de est a RS) aşa cum indică şi figura 4.7.

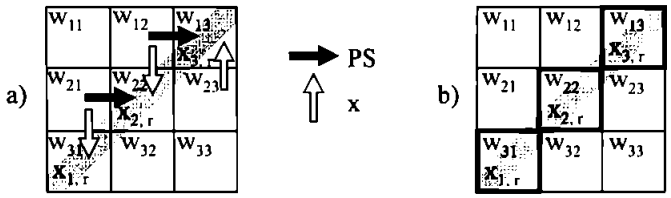



Fig. 4.7. A treia etapă. a) comunicaţii; b) calcul

Următoarele sume parţiale sunt încă în curs de calcul şi, pentru a înlănţui recurenţele, va trebui să reamplasăm vectorul rezultat în diagonala reţelei (vectorul rezultat devine vectorul înmulţitor pentru următorul pas de recurenţă).

**D. A patra etapă**

Cu scopul de a respecta regula r3, vom încerca să exploatăm decalajul temporal care apare pe latura de est a RS (componentele vectorului rezultat se obţin în etape succesive în celulele de pe latura de est a RS). Vom profita de acest fapt şi vom retransmite componentele deja calculate pe o cale de întoarcere spre diagonala principală. Prima componentă a vectorului rezultat este deja disponibilă pe latura de est a RS. Ea va fi transformată prin aplicarea funcţiei de transfer $\sigma$ în componenta corespondentă a noului vector **x**, care va fi vectorul înmulţitor pentru următorul pas de recurenţă:

$$x_{1,2} = \sigma(PS_{1,3}) \tag{4.7}$$

Datele mobile sunt transmise:

- $x_{1,2}$ se va propaga dinspre est spre vest (Să remarcăm trecerea la următorul pas de recurenţă!);
- $x_{2,1}$ şi $x_{3,1}$ sunt propagate spre sud;
- componenta $x_{1,1}$, prezentă pe latura de sud, va fi retrimisă spre nord. Această retransmisie spre nord va servi mai târziu la detecţia convergenţei pentru faza de recunoaştere aferentă reţelei *Hopfield* (deocamdată nu detaliem);
- sumele parţiale $PS_{3,1}$ şi $PS_{2,2}$ sunt transmise spre est.

Calculele efectuate sunt:

$$PS_{2,3} = PS_{2,2} + w_{23} \cdot x_{3,1} \tag{4.8}$$

$$PS_{3,2} = PS_{3,1} + w_{32} \cdot x_{2,1} \tag{4.9}$$

Noutatea adusă de această etapă constă în aplicarea funcţiei neliniare $\sigma$ componentei $PS_{1,3}$ prezente deja pe latura de est a RS (ec. (4.7)) şi retransmisia rezultatului $x_{1,2}$ spre diagonală.

Circulaţia datelor şi calculele efectuate sunt evidenţiate în figura 4.8. Lungimea căii de revenire pe diagonală (de pe latura de est) este evident egală cu cea parcursă dinspre diagonală spre latura de est. Această lungime se diminuează cu o unitate pe măsură ce avansăm de la o linie la alta dinspre nord spre sud.

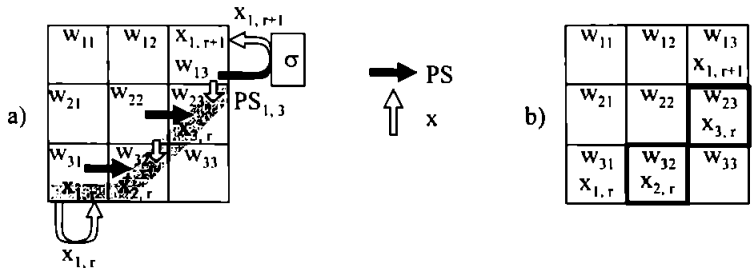



Fig. 4.8. Etapa a patra: aplicarea funcţiei neliniare $\sigma$ şi retransmiterea spre diagonală
a) comunicaţii; b) calcul

Decalajul temporal între sosirile succesive pe latura de est ale componentelor vectorului rezultat este de asemenea de o unitate de timp. Rezultă că toate componentele vectorului rezultat se vor găsi în acelaşi moment pe diagonală şi vor fi gata pentru declanşarea următorului pas de recurenţă.

**E. A cincea etapă**

Etapa a cincea este similară celei anterioare; se va retransmite spre diagonală a 2-a componentă a vectorului rezultat (după aplicarea prealabilă a funcţiei neliniare $\sigma$).

$$x_{2,2} = \sigma(PS_{2,3}) \tag{4.10}$$

Vom avea de asemenea următoarele transferuri de date:

- $x_{1,2}$ şi $x_{2,2}$ sunt propagate dinspre est spre vest;

- $x_{3,1}$ este propagat spre sud;
- $x_{1,1}$ şi $x_{2,1}$ (prezentă pe latura de sud) sunt retrimise spre nord;
- suma parţială $PS_{3,2}$ este transmisă spre est.

$$PS_{3,3} = PS_{3,2} + w_{33} \cdot x_{3,1} \tag{4.11}$$

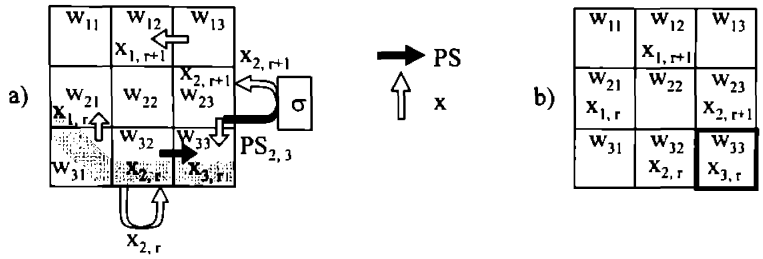



Fig. 4.9. Etapa a cincea. a) comunicaţii; b) calcul

În etapa a şasea va fi obţinut rezultatul aşteptat: produsul $\mathbf{W} \cdot \mathbf{x}$ va fi finalizat, funcţia neliniară σ va fi aplicată pentru ultima componentă şi vectorul rezultat va fi repoziţionat pe diagonala principală a RS.

**F. A şasea etapă**

Suma parţială $PS_{3,3}$ este prezentă pe latura de est a RS. Se va aplica funcţia de transfer σ şi se va obţine:

$$x_{3,2} = \sigma(PS_{3,3}) \tag{4.12}$$

Se vor executa de asemenea următoarele transferuri de date:

- $x_{1,2}$, $x_{2,2}$ şi $x_{3,2}$ sunt propagate dinspre est spre vest;
- $x_{1,1}$, $x_{2,1}$ şi $x_{3,1}$ sunt retrimise spre nord.

În această etapă nu se execută calcule. Figura 4.10. prezintă situaţia RS la finele etapei a şasea (care coincide cu sfârşitul primului pas de recurenţă).

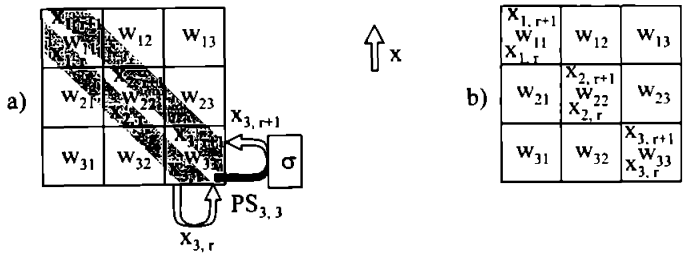



Fig. 4.10. Etapa a şasea. a) comunicaţii; b) calcul

La finele celei de-a şasea etape, atât componentele noului vector $\mathbf{x}_{r+1}$ cât şi cele ale vechiului vector $\mathbf{x}_r$ vor fi prezente în celulele sistolice de pe diagonala RS. Regulile r1, r2 şi r3 au fost respectate şi reţeaua este pregătită pentru lansarea următorului pas de recurenţă.

Prezenţa simultană a vechiului vector şi a noului vector în celulele de pe diagonală ne permite să comparăm cei doi vectori termen cu termen. Vom dispune astfel de N semnale de comparare locală în diagonala RS care vor trebui compuse pentru a genera semnalul de comparare globală (rezultatul comparării). Această funcţie este esenţială pentru faza de recunoaştere aferentă algoritmului *Hopfield*, pe parcursul căreia trebuie detectată convergenţa spre o stare stabilă. Algoritmul presupune calculul:

$$\mathbf{x}_r = \sigma(\mathbf{W} \cdot \mathbf{x}_{r-1}) \tag{4.13}$$

până la atingerea convergenţei (unei stări stabile):

$$\mathbf{x}_r = \mathbf{x}_{r-1} \tag{4.14}$$

### 4.2.2. Detecţia convergenţei

După un pas de recurenţă fiecare celulă diagonală va conţine $x_{i,r}$ şi $x_{i,r+1}$. Compararea acestor 2 valori ne va furniza un rezultat local în legătură cu convergenţa vectorului **x**. Semnalul de convergenţă global, pentru o RS de dimensiune N x N, se va calcula:

$$\text{Conv}(N) = \prod_{i=1}^{N} (x_{i,r} = x_{i,r+1}) = \prod_{i=1}^{N} CV_i \tag{4.15}$$

Ecuaţia (4.15) se poate scrie şi sub formă recursivă:

$$\begin{aligned} &\text{Conv}(0) = 1 \\ &\text{Conv}(N) = \text{Conv}(N-1) \cdot \text{Conv}(N) \end{aligned} \tag{4.16}$$

Fiecare celulă diagonală va conţine un dispozitiv de comparare care va genera valoarea booleană $CV_i$. Acest semnal va fi transmis local (spre vest) şi va fi sintetizat pe latura de vest a RS printr-un dispozitiv care calculează sistolic conjuncţia prezentă în ecuaţia (4.16). Vom abandona pentru moment propagarea componentelor vectorilor **x** şi **PS**. Poziţia iniţială a vectorului **CV** este prezentată în figura 4.11.

| | | |
|---|---|---|
| $CV_1$ $w_{11}$ | $w_{12}$ | $w_{13}$ |
| $w_{21}$ | $CV_2$ $w_{22}$ | $w_{23}$ |
| $w_{31}$ | $w_{32}$ | $CV_3$ $w_{33}$ |

Fig. 4.11. Poziţia iniţială a semnalelor de convergenţă locale

Toate aceste semnale vor fi transmise spre latura de vest. Dispozitivul sistolic care implementează ecuaţia 4.16 combină apoi aceste semnale locale pentru a genera semnalul de convergenţă global (fig. 4.12).

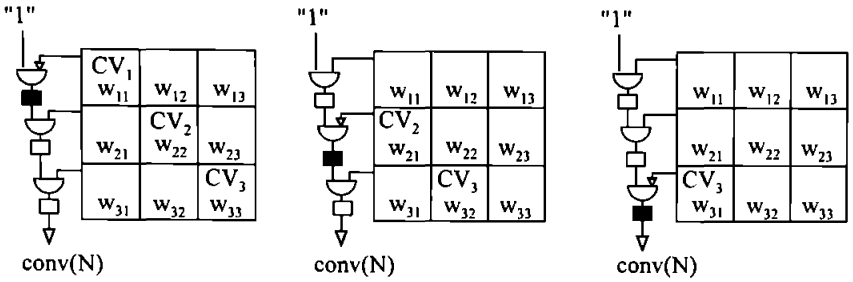



Fig. 4.12. Propagarea sistolică a semnalului de convergenţă

Decalajul temporal implicat de transmisia diagonalei spre latura de vest a RS este exploatat pentru sincronizarea calculului semnalului de convergenţă. Această subreţea poate fi descrisă de asemenea sub forma unui sistem de ecuaţii recurente uniforme şi validitatea sa poate fi demonstrată.

### 4.2.3. Definirea unei arhitecturi logice

Vom propune o arhitectură logică pentru RS bidimensională care implementează funcţiile anterior descrise (fig. 4.13). Rebuclajele prevăzute pe laturile de Nord, Sud şi Est vor permite repoziţionarea noului vector **x** pe diagonala RS în scopul înlănţuirii paşilor de recurenţă succesivi. În ceea ce priveşte calculele efectuate, toate celulele sistolice sunt identice. În ceea ce priveşte comunicaţiile, celulele diagonale vor trebui specializate. După cum a fost deja descris, în timpul primei faze de calcul, componentele vectorului **x**, prezente iniţial în diagonală, sunt propagate mai întâi spre nord, pentru ca apoi să fie rebuclate spre sud. Componentele noului vector **x** sunt propagate de pe latura de est şi sunt poziţionate pe diagonală pentru declanşarea următorului pas de recurenţă.

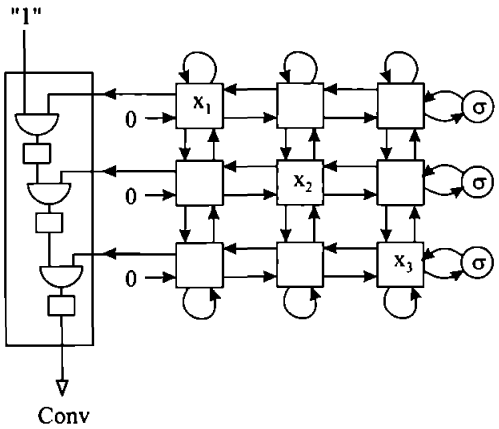



Fig. 4.13. Arhitectura logică a RS

În diagonală componentele noului vector vor fi reflectate spre nord. Acest moment al schimbării direcţiei spre nord coincide cu startul următorului ciclu de recurenţă. Figura 4.14 ilustrează reflexia în diagonală a componentelor noului vector.

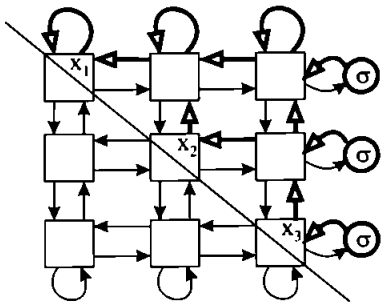
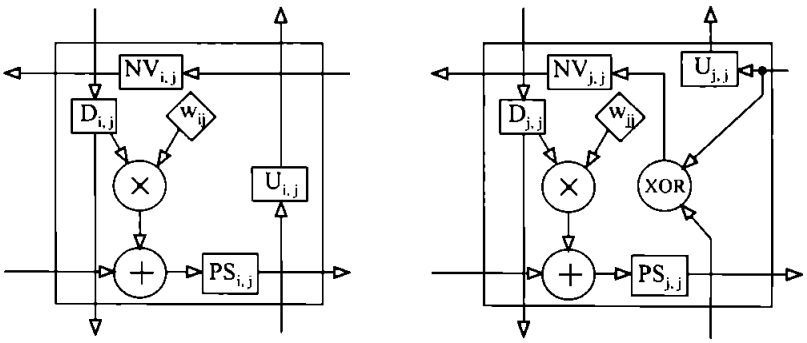



Fig. 4.14. Propagarea vectorului x cu schimbarea direcţiei pe diagonală

Vom defini în continuare arhitectura logică a CS diferenţiind celulele diagonale de cele non - diagonale. Pe baza acestor definiţii arhitecturale, vom detalia toate calculele realizate într-o celulă.

### 4.2.4. Specificaţiile celulei sistolice

Arhitectura generală a RS a fost prezentată în fig. 4.14. Vom preciza în detaliu operaţiile efectuate în fiecare celulă, precum şi căile de comunicaţie asociate fiecărei celule. Figura 4.15 pune în evidenţă aceste aspecte.




Transmisii date:

$D_{i,j} = D_{i-1,j}$
$U_{i,j} = U_{i+1,j}$
$NV_{i,j} = NV_{i,j+1}$

Calcule:

$PS_{i,j} = PS_{i,j-1} + w_{ij} \cdot D_{i,j}$

a) non - diagonale

Transmisii date:

$D_{j,j} = D_{j-1,j}$
$U_{j,j} = N_{j,j+1}$

Calcule:

$NV_{j,j} = U_{j+1,j} \oplus NV_{j,j+1}$
$PS_{j,j} = PS_{j,j-1} + w_{jj} \cdot D_{jj}$

b) diagonale

Fig. 4.15. Specificaţiile celulelor sistolice

Celula diagonală realizează virtual trecerea de la un pas de recurenţă la pasul următor, prin încărcarea noii valori NV (noul vector **x**) în registrul U (*Up*). Din acest moment, noua valoare devine valoarea curentă (actuală) a vectorului **x** deoarece s-a declanşat pasul de calcul al unui nou vector **x**.

Pentru completarea specificaţiilor din figura 4.15 mai trebuie precizate două condiţii de limită. Condiţia aferentă laturii de nord care realizează rebuclajul se exprimă prin:

$$D_{1,i} = U_{1,i}$$

iar condiţia de iniţializare aferentă laturii de vest va fi:

$$PS_{i,1} = 0$$

ceea ce înseamnă că se demarează pasul recurenţial cu o sumă parţială nulă. Vom vedea că această condiţie de iniţializare cu zero nu va mai fi respectată în faza de învăţare.

Complexitatea circulaţiei datelor în cadrul RS reclamă o etapă de sinteză, care ne va permite introducerea unui simbolism de care vom face uz în cele ce urmează. Acesta ne va permite să evidenţiem mişcarea conjugată a datelor care circulă în RS.

Prima etapă constă în introducerea vectorului înmulţitor ($x_r$) în reţea. Această etapă va consuma N unităţi de timp.

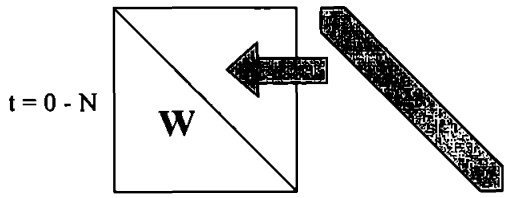



Fig. 4.16. Încărcarea vectorului $x_r$

În momentul în care vectorul $x_r$ atinge diagonala, va continua să se propage spre nord (reflectat spre nord), până pe latura de nord a reţelei. Aici este rebuclat (reflectat) pe direcţia sud. Din momentul rebuclării, sumele parţiale (**PS**) vor fi elaborate progresiv şi propagate dinspre vest spre est (fig. 4.17).

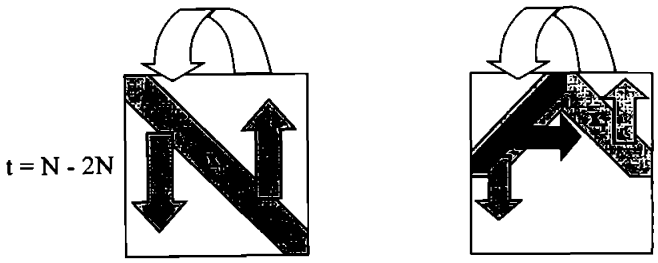



Fig. 4.17. Rebuclarea vectorului $x_r$ pe latura de nord şi generarea sumelor parţiale **PS**

În momentul în care prima sumă parţială (devenită finală) atinge latura de est, prima componentă a vectorului $\mathbf{x}_r$ participantă la elaborarea **PS** atinge latura de sud (figura 4.18).

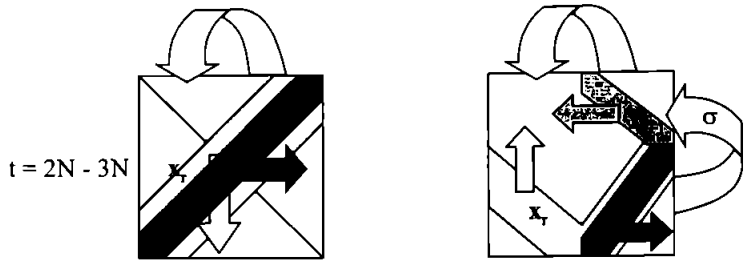



Fig. 4.18. Elaborarea sumelor parţiale **PS** şi aplicarea funcţiei de activare σ

Din momentul în care prima suma parţială (finală) atinge latura de est, începe generarea componentelor noului vector $\mathbf{x}_{r+1}$ prin aplicarea funcţiei neliniare σ. Componentele noului vector vor fi repropagate spre vest pentru a se întâlni pe diagonală cu componentele vechiului vector $\mathbf{x}_r$ (care revin din direcţia sud). Comparaţia celor 2 vectori (pentru detecţia convergenţei) se va face deci în celulele diagonale (fig. 4.19).

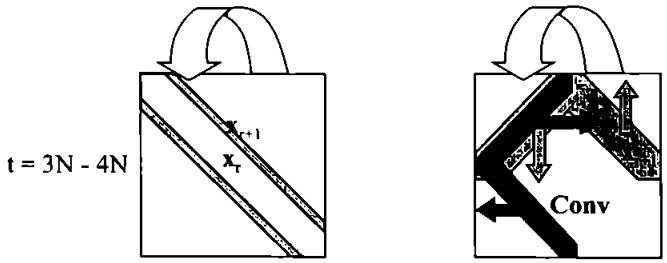



Fig. 4.19. Revenirea pe diagonală şi calculul convergenţei

Elaborarea semnalului de convergenţă global se realizează în paralel cu evoluţia calculelor pentru al doilea pas de recurenţă. Acest semnal de convergenţă este procesat sistolic dinspre nord spre sud (lanţul de porţi şi elemente de memorare din figura 4.12) şi procesarea sa va dura N cicluri (timpul în care prima sumă parţială parcurge distanţa de la diagonală până la marginea de est a RS). În cazul algoritmului *Hopfield*, dacă se atinge convergenţa, starea nu mai variază (sumele parţiale nu se mai modifică de la un pas de convergenţă la altul). Produsele suplimentar calculate nu vor modifica rezultatele. Vectorul care soseşte pe latura de est poate deci fi extras, componentă cu componentă imediat după detecţia convergenţei. În general, această proprietate va fi exploatată pentru gestionarea operaţiilor de intrare / ieşire ale RS, pentru toţi algoritmii care fac apel la o recursivitate a cărei profunzime este limitată de detecţia unei stări stabile.

Această prezentare geometrică a circulaţiei datelor în diferitele faze ale algoritmului nu constituie o demonstraţie a validităţii acestuia. Pentru a demonstra validitatea algoritmului vom relua ecuaţiile elaborate în fig. 4.15 şi le vom exprima sub forma unui Sistem de Ecuaţii Recurente Uniforme (SERU). Vom demonstra apoi că operaţiile asociate transferurilor de date realizează corect calculul dorit. Acest principiu va fi reluat pentru a valida, în secţiunile următoare, algoritmii destinaţi fazei de învăţare.

### 4.2.5. Descrierea Sistemului de Ecuaţii Recurente Uniforme

Pentru reţeaua *Hopfield*, funcţia de recunoaştere se exprimă prin ecuaţia recurentă:

$$x_{i,r} = \sigma\left(\sum_{j=1}^{N} w_{ij} \cdot x_{j,r-1}\right)$$

unde:

i şi j sunt indicii spaţiali
r este pasul de recurenţă
t este indice temporal (va fi utilizat în continuare)
N ordinul matricii pătrate **W** (matricea de coeficienţi sinaptici)
σ este funcţia de transfer neliniară

Vom scrie ecuaţiile SERU care vor exprima funcţia fiecărui element component al celulei sistolice precum şi comunicaţiile acestuia. Această descriere nu este posibilă decât pentru o RS localizată.

Sumele parţiale evoluează în reţea dinspre latura de vest spre latura de est (ec. 4.16); aceste sume parţiale sunt iniţializate pe latura de vest prin injectarea unor valori nule (ec. 4.17).

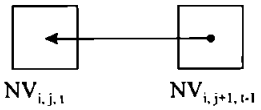
$$PS_{i,j,t} = PS_{i,j-1,t-1} + w_{i,j} \cdot D_{i,j,t-1} \quad (4.16)$$

$$PS_{i,1,t} = 0 \quad (4.17)$$

Noile valori (componentele noului vector **x**) sunt introduse din direcţia est şi tranzitează RS spre vest. Aceste valori se obţin pe latura de est (în exteriorul RS) prin aplicarea funcţiei neliniare σ sumelor parţiale obţinute în celulele de pe latura de est a RS.

$$NV_{i,N,t} = \sigma(PS_{i,N,t-1}) \quad (4.18)$$

În interiorul reţelei noile valori tranzitează spre vest (fără a fi modificate) până la diagonală:



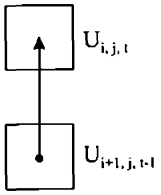

$$NV_{i,j,t} = NV_{i,j+1,t-1} \quad (4.19)$$

Începând de la diagonală şi până pe latura de vest, noile valori vor reprezenta de fapt rezultatul comparaţiilor locale efectuate în celulele diagonale (convergenţa):

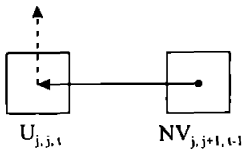
$$NV_{j,j,t} = U_{j+1,j,t-1} \oplus NV_{j,j+1,t-1} \quad (4.20)$$

Valorile care urcă traversând reţeaua dinspre sud spre nord sunt descrise de ecuaţia (4.21). Acestea sunt noile valori recepţionate în diagonală dinspre est şi reflectate în diagonală spre nord (ec. 4.22).

$$U_{i,j,t} = U_{i+1,j,t-1} \quad (4.21)$$

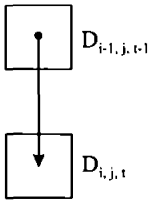



$$U_{j,j,t} = NV_{j,j+1,t-1} \tag{4.22}$$

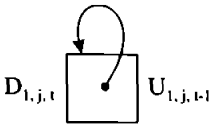



Valorile descendente care traversează reţeaua dinspre nord spre sud (ec. 4.23) provin din valorile care au urcat spre nord şi care au fost rebuclate pe latura de nord a RS (ec. 4.24):

$$D_{i,j,t} = D_{i-1,j,t-1} \tag{4.23}$$




$$D_{1,j,t} = U_{1,j,t-1} \tag{4.24}$$




Vom urmări căile de acces la o celulă sistolică cu scopul de a ajunge la marginile RS; cunoaştem de asemenea valorile de intrare şi momentele de timp la care acestea sunt aplicate la marginile RS.
Vom analiza mai întâi traseul notat cu 2 în figura 4.20. Vom porni de la celula (i, j) şi vom urca până la latura de nord a RS aplicând succesiv ecuaţia (4.23):

$$D_{i,j,t} = D_{1,j,t-i+1} \tag{4.25}$$

Pe latura de nord, valorile care urcă dinspre sud (U), sunt rebuclate din nou spre sud, pe calea descendentă (D). Acest rebuclaj descris de ecuaţia (4.24) ne permite să scriem:

$$D_{1,j,t-i+1} = U_{1,j,t-i} \tag{4.26}$$

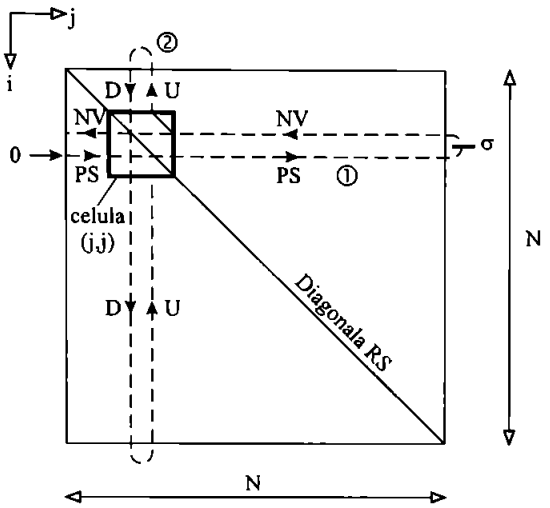



Fig. 4.20. Căile de acces la o celulă sistolică

Calea ascendentă (U) porneşte din diagonală având ca sursă calea NV de pe traseul notat cu 1. Vom aplica succesiv ecuaţia (4.21) până la diagonală:

$$U_{1,j,t-i} = U_{1+j-1,j,t-i-(j-1)}$$

$$U_{1,j,t-i} = U_{j,j,t-i-j+1} \qquad (4.27)$$

Valorile ascendente (U) se obţin din valorile NV calculate la marginea de est a RS (prin aplicarea funcţiei neliniare $\sigma$) şi care sunt emise spre vest până la diagonală (ec. 4.22).

$$U_{j,j,t-i-j+1} = NV_{j,j+1,t-i-j} \qquad (4.28)$$

În sfârşit, dacă aplicăm succesiv ecuaţia (4.19) pe calea NV (traseul 1) până la marginea de est a RS, obţinem:

$$NV_{j,j+1,t-i-j} = NV_{j,j+1+(N-j-1),t-i-j-(N-j-1)}$$

$$NV_{j,j+1,t-i-j} = NV_{j,N,t-i-N+1} \qquad (4.29)$$

Combinând ecuaţiile (4.25) şi (4.29) obţinem:

$$D_{i,j,t} = NV_{j,N,t-i-N+1} \qquad (4.30)$$

Vom substitui în ecuaţia (4.16) expresia lui $D_{i,j,t}$ obţinută în (4.30):

$$PS_{i,j,t} = PS_{i,j-1,t-1} + w_{i,j} \cdot NV_{j,N,t-i-N} \qquad (4.31)$$

Pe latura de est a RS, conform (4.16), vom avea valorile:

$$PS_{i,N,t} = \sum_{j=1}^{N} w_{i,j} \cdot D_{i,j,t-N+j-1}$$

care în baza relaţiei (4.30) devin:

$$PS_{i,N,t} = \sum_{j=1}^{N} w_{i,j} \cdot NV_{j,N,t-2N-i+j} \qquad (4.32)$$

Aplicând (4.18) şi (4.19) obţinem:

$$NV_{i,N,t} = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot NV_{j,N,t-2N-i+j-1}\right) = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot NV_{j,N-1,t-2N-i+j}\right) \qquad (4.33)$$

Considerând originea timpului momentul introducerii primei componente din vectorul **x** în RS (starea iniţială prezentată în fig. 4.16), şi observând că un pas de recurenţă are durata de 2N unităţi de timp, atunci a *i* - a componentă a vectorului **x** va atinge latura de est a reţelei (venind dinspre vest) la momentele 2rN + i. Deci:

$$x_{i,r} = NV_{i,N,2rN+i} \qquad (4.34)$$

Utilizând relaţia (4.34) vom putea rescrie ecuaţia (4.33) astfel:

$$NV_{i,N,2rN+i} = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot NV_{j,N-1,2N(r-1)+j}\right)$$

Rezultă astfel:

$$x_{i,r} = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot x_{j,(r-1)}\right) \qquad (4.35)$$

Am arătat deci că SERU definit realizează corect produsele succesive $\mathbf{W} \cdot \mathbf{x}$ (paşii de recurenţă sunt corect executaţi). Operaţiile descrise sunt implementate în celulele sistolice şi, prin urmare RS va efectua corect calculul iterativ. Demonstrarea validităţii mecanismului de detecţie a convergenţei se face după acelaşi principiu.

Demonstraţia prezentată nu se referă decât la faza de recunoaştere aferentă unui singur vector de intrare. Gradul de utilizare a celulelor sistolice este dat de raportul dintre numărul de celule active şi numărul total de celule disponibile (pe durata unui pas de recurenţă). Gradul mediu de utilizare se poate calcula la nivelul unui pas de recurenţă, contorizând celulele utilizate pe parcursul celor 6 etape conţinute într-un ciclu de recurenţă; gradul mediu va fi foarte redus: 1/2 N. Totuşi, performanţele reţelei pot fi mult îmbunătăţite şi se poate ajunge chiar la un grad de utilizare de 100 %, dacă se ia în considerare o procesare în tehnică *pipeline* a vectorilor de intrare. În acest scop, vom considera un şir de vectori aplicaţi succesiv la intrarea reţelei. Fiecare vector este notat cu un indice p care indică numărul său de ordine în cadrul şirului. SERU definit anterior nu se va modifica. Ştiind că un pas de recurenţă are durata 2N unităţi de timp, a *i* - a componentă a vectorului de indice p va ieşi din RS la momentul 2rN + i + p (cu $p \leq 2N$).

Ecuaţia (4.34) va deveni:

$$x_{i,r,p} = NV_{i,N,2rN+i+p} \qquad p \le 2N \qquad (4.36)$$

Dacă substituim (4.36) în (4.33) obţinem:

$$x_{i,r,p} = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot NV_{j,N\cdot1,2N(r\cdot1)+j+p}\right) \qquad (4.37)$$

şi, în final:

$$x_{i,r,p} = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot x_{j,r-1,p}\right) \qquad p \le 2N \qquad (4.38)$$

După cum se poate constata din (4.38), indicele de aplicare a vectorilor p rămâne neschimbat pentru fiecare pas de recurenţă. Aceasta înseamnă că nu poate să apără confuzie între vectori şi procesarea în tehnică *pipeline* se va efectua corect. În aceeaşi manieră se poate arăta că *pipelining*-ul nu va afecta funcţionarea corectă a mecanismului de detecţie de convergenţă.

### 4.3. Extensia arhitecturii pentru învăţare

Principalii algoritmi conexionişti sunt constituiţi din operaţii matriceale şi vectoriale simple şi din transformări neliniare (funcţia de activare). În cele ce urmează, vom arăta că regula Perceptronului, regula lui *Hebb* şi cea a lui *Kohonen* pot fi implementate sub forma unui algoritm sistolic. Pentru toate modelele, faza de recunoaştere este identică şi se rezumă la un simplu produs între matricea sinaptică **W** şi vectorul de intrare **x**, urmat de aplicarea funcţiei neliniare $\sigma$. Diferenţele apar la faza de învăţare.

#### 4.3.1. Resursele comune diferitelor modele

Regulile de învăţare se bazează pe câteva operaţii matriceale şi vectoriale rezumate în tabelul 4.1.

| Denumire Operaţie | Expresie matematică | Identificator |
|---|---|---|
| Produsul unui vector cu transpusul său = matrice | $\mathbf{v} \cdot \mathbf{v}^T$ | (1) |
| Sumă / Diferenţă matriceală | $\mathbf{W_1} \pm \mathbf{W_2}$ | (2) |
| Funcţia neliniară | $\sigma(\mathbf{v})$ | (3) |
| Produsul matrice - vector | $\mathbf{W} \cdot \mathbf{v}$ | (4) |

Tabelul 4.1. Operaţiile de bază utilizate în faza de învăţare

unde:

**v** - vector coloană
**W** - matrice
$\sigma$ - funcţie neliniară

Toate regulile de învăţare vor putea fi construite cu ajutorul acestor operaţii de bază (set de primitive). Tabelul 4.2 prezintă câteva reguli de învăţare precum şi operaţiile de bază utilizate pentru implementarea acestora.

| Regula de învăţare | Faza de recunoaştere | Faza de învăţare | Operaţii de bază |
|---|---|---|---|
| *Hebb* | $\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x})$ | $\mathbf{W} := \mathbf{W} + \mathbf{x} \cdot \mathbf{x}^T$ | (1) (2) (3) (4) |
| *Perceptron, Adaline* | $\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x})$ | $\mathbf{W} := \mathbf{W} + \alpha \cdot (\mathbf{d} - \mathbf{W} \cdot \mathbf{x}) \cdot \mathbf{x}^T$ | (1) (2) (3) (4) |
| *Kohonen* | $\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x})$ | $\mathbf{W}_i := \mathbf{W}_i + \alpha \cdot \lambda(\Delta_{topo}(i,I))(\mathbf{x}^T - \mathbf{W}_i)$ | (2) (3) (4) |
| *Back Propagation* | $\mathbf{y} = \sigma(\mathbf{W} \cdot \mathbf{x})$ | $\delta^{[s]} = \sigma'(\mathbf{p}^{[s]}) \otimes (\mathbf{d}-\mathbf{y})$ - strat de ieşire<br>$\delta^{[q]} = \sigma'(\mathbf{p}^{[q]}) \otimes (\mathbf{W}^{[q+1]})^T \cdot \delta^{[q+1]}$ - straturi ascunse<br>$\mathbf{W} := \mathbf{W} + \alpha \cdot \delta^{[q]} \cdot (\mathbf{x}^{[q]})^T$ | (1) (2) (3) (4) |

Tabelul 4.2. Câteva reguli de învăţare

unde:

- $\mathbf{x}$ - vectorul aplicat la intrare
- $\mathbf{d}$ / $\mathbf{y}$ - vectorul dorit / real de ieşire
- $\mathbf{W}$ - matricea de coeficienţi sinaptici
- $\mathbf{W}_i$ - linia *i* a matricii $\mathbf{W}$
- $\alpha$ - constanta de învăţare
- $\lambda$ - funcţia de vecinătate (restricţionează actualizarea ponderilor doar pentru neuronii i aflaţi în vecinătatea învingătorului I)
- $\Delta_{topo}(i,I)$ - distanţa între neuronul i şi învingătorul I în spaţiul topologic definit
- $\sigma$ - funcţia de transfer neliniară (funcţia de activare)
- $\delta^{[s]}$ - vectorul de eroare asociat stratului de ieşire
- $\delta^{[q]}$ - vectorii de eroare asociaţi straturilor interne (ascunse)
- $\sigma'(\mathbf{p}^{[q]})$ - vectorul derivatelor funcţiei de transfer în raport cu potenţialele neuronilor din stratul [q]

Am arătat deja în paragraful alocat funcţiei de recunoaştere că operaţia (4) poate fi implementată pe reţeaua sistolică definită. Operaţia (3) a fost rejectată în exteriorul RS dar poate fi aplicată facil datorită accesibilităţii la datele care circulă în interiorul RS (în momentul rebuclării acestora pe latura de est). În cele ce urmează vom arăta că şi învăţarea (după diferitele reguli) este implementabilă pe RS definită.

### 4.3.2. Regula lui Hebb

Vom considera deocamdată regula lui *Hebb* ca un mecanism de calcul şi nu ca unul de învăţare (pentru simplificare). Performanţele acestei reguli sunt insuficiente pentru a permite aplicaţii concrete. Totuşi, mecanismul definit de regula lui *Hebb* este comun şi altor reguli mai evoluate (de exemplu regula "*Delta projection*") capabile de performanţe superioare.

Pentru actualizarea ponderilor $w_{i,j}$ trebuie să dispunem, în fiecare celulă (i, j), de valorile $x_i$, $x_j$. Aceasta se va realiza prin introducerea vectorului $\mathbf{x}$ în diagonala RS şi obligându-l să circule simultan spre Nord şi spre Vest. Baleiajul realizat de cele două direcţii perpendiculare va pune la dispoziţia fiecărei celule sistolice (i, j), dubletele $(x_i, x_j)$. Va fi suficient să se calculeze produsul $x_i \cdot x_j$ şi să se adauge valoarea obţinută la valoarea coeficientului sinaptic local. Circulaţia datelor este prezentată în figura 4.21.

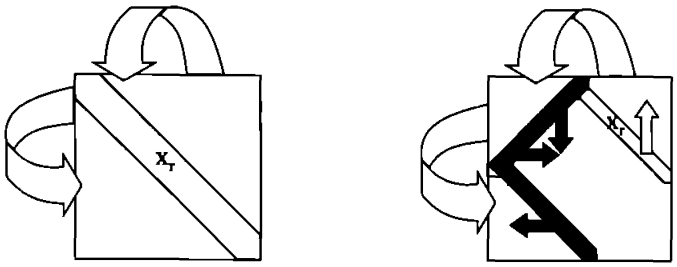



Fig. 4.21. Circulaţia datelor în RS pentru regula lui Hebb

Să notăm că, pe durata fazei de învăţare, ponderile sinaptice **W** nu trebuie să interacţioneze cu datele care traversează RS de la Vest spre Est. Dispozitivul de înmulţire din CS este deci liber şi poate fi utilizat pentru calculul produselor locale $x_i \cdot x_j$. Fiecare CS trebuie să memoreze activitatea neuronilor i şi j, să calculeze produsul $x_i \cdot x_j$ şi să adauge rezultatul la valoarea ponderii sinaptice locale. Aceste operaţii sunt realizabile cu o CS a cărei structură internă este descrisă în figura 4.22. a).

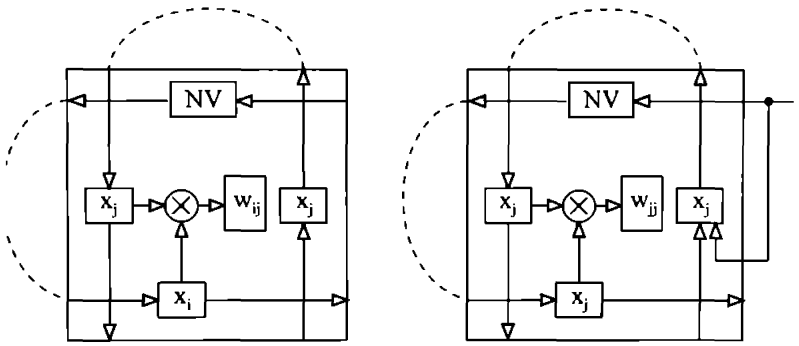


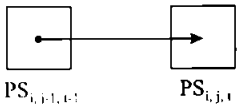

Fig. 4.22. Resursele interne ale CS

Circuitele de înmulţire şi împărţire au fost prevăzute şi pentru faza de recunoaştere. Principalele diferenţe se vor referi la secvenţiere şi la liniile de date prin care se accesează resursele interne ale CS. Ca şi în cazul fazei de recunoaştere, celulele diagonale au o structură de comunicaţie uşor diferită de celelalte celule. Celulele diagonale trebuie să asigure retransmiterea componentelor vectorului de stare spre latura de nord a RS. Dar, pentru faza de învăţare, ea trebuie să rămână de asemenea transparentă pentru a transmite vectorul de stare spre latura de vest. Aceasta permite o retransmitere care sincronizează baleiajul planului de ponderi sinaptice de către cei doi vectori de stare identici (dar pe direcţii perpendiculare). Aceste modificări sunt redate în figura 4.22. b), unde se poate observa legătura dintre registrul NV şi registrul care emite componenta $x_j$ pe direcţia nord. Această legătură se va putea realiza cu ajutorul unui multiplexor suplimentar în structura celulei.

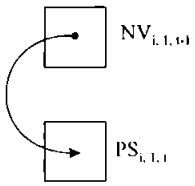
În cele ce urmează vom arăta că această structură definită poate calcula produsul unui vector cu transpusul său şi că matricea rezultat poate fi adăugată matricii curente de ponderi sinaptice.

### 4.3.3. Funcţionarea RS pentru faza de învăţare Hebb

Vom defini ecuaţiile SERU aferente acestei faze. Sumele parţiale vor evolua în RS dinspre marginea de vest spre cea de est (ec. 4.39). Sumele parţiale injectate pe latura de vest a reţelei se obţin din valorile NV care tranzitează RS dinspre est spre vest (ec. 4.40).

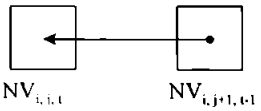



$$PS_{i,j,t} = PS_{i,j-1,t-1} \tag{4.39}$$



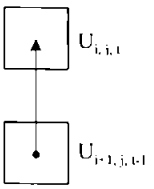

$$PS_{i,1,t} = NV_{i,1,t-1} \tag{4.40}$$

Noile valori (NV) sunt introduse prin latura de est a reţelei şi tranzitează RS până la marginea de vest.

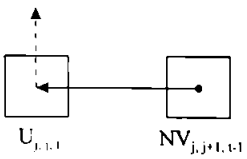



$$NV_{i,j,t} = NV_{i,j+1,t-1} \tag{4.41}$$

Valorile ascendente traversează reţeaua dinspre sud spre nord (ec. 4.42) şi sunt obţinute din noile valori (NV) reflectate pe direcţia nord de către celulele diagonale (ec. 4.43).

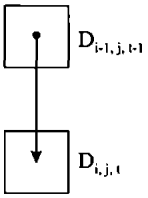



$$U_{i,j,t} = U_{i+1,j,t-1} \tag{4.42}$$

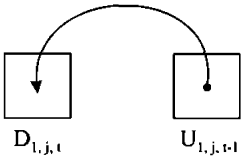



$$U_{j,j,t} = NV_{j,j+1,t-1} \tag{4.43}$$

Valorile descendente tranzitează din nord spre sud (ec. 4.44), şi provin din valorile ascendente care sunt rebuclate pe latura de nord (ec. 4.45).

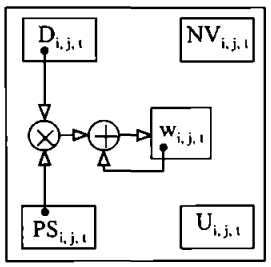



$$D_{i,j,t} = D_{i-1,j,t-1} \tag{4.44}$$




$$D_{1,j,t} = U_{1,j,t-1} \tag{4.45}$$

Fiecare pondere sinaptică este actualizată local cu produsul dintre valoarea descendentă şi cea tranzitată prin registrele PS.




$$W_{i,j,t} = W_{i,j,t-1} + PS_{i,j,t-1} \cdot D_{i,j,t-1} \tag{4.46}$$

Pentru o celulă (i, j), vom relua traseul în spaţiu şi timp până la atingerea unei laturi a RS. Vom prezenta mai întâi traseul notat cu 1 în figura 4.23. Pornind dintr-o celulă oarecare (i, j), vom urmări traseul spre direcţia vest până la rebuclaj. Această operaţie va fi efectuată prin aplicarea ecuaţiei (4.39) pentru $j < 1$.

$$PS_{i,j,t} = PS_{i,j-(j-1),t-(j-1)} = PS_{i,1,t-j+1}$$

Am ajuns pe latura de vest a reţelei şi aici vom aplica ecuaţia (4.40):

$$PS_{i,j,t} = NV_{i,1,t-j} \tag{4.47}$$

Refacem apoi traseul până pe latura de vest a RS, pe traseul NV, prin aplicarea succesivă a ecuaţiei (4.41).

$$NV_{i,\,1,\,t-j} = NV_{i,\,1+(N-1),\,t-j-(N-1)} = NV_{i,\,N,\,t-j-N+1} \qquad (4.48)$$

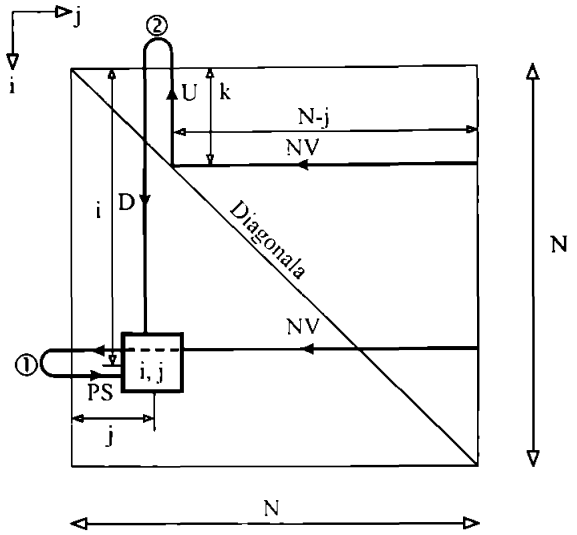



Fig. 4.23. Căile de acces la celula (i, j)

Deci, pentru traseul 1 am obţinut:

$$PS_{i,\,j,\,1} = NV_{i,\,N,\,t-j-N+1} \qquad (4.49)$$

Vom urmări acum traseul 2, aşa cum este indicat în figura 4.23. Aplicând ecuaţia (4.44) până la latura de nord a RS (i < 1), obţinem:

$$D_{i,\,j,\,t} = D_{1+(i-1),\,j,\,t-(i-1)} = D_{1,\,j,\,t-i+1}$$

Rebuclajul pe latura de nord se va face conform ecuaţiei (4.45):

$$D_{1,\,j,\,t-i+1} = U_{1,\,j,\,t-i}$$

Coborâm apoi până la diagonala RS, aplicând succesiv ecuaţia (4.44):

$$U_{1,\,j,\,t-i} = U_{1+(j-1),\,j,\,t-i-(j-1)} = U_{j,\,j,\,t-i-j+1}$$

Reflexia direcţiei pe diagonală se face după ecuaţia (4.43):

$$U_{j,\,j,\,t-i-j+1} = NV_{j,\,j+1,\,t-i-j}$$

Pe baza ecuaţiei (4.41), pe care o aplicăm succesiv până la j = N, vom reface traseul până la latura de est a RS:

$$NV_{j,\,j+1,\,t-i-j} = NV_{j,\,j+1+(N-j-1),\,t-i-j-(N-j-1)} = NV_{j,\,N,\,t-i-N+1}$$

Rezumând, pentru traseul 2 obţinem:

$$D_{i,j,t} = NV_{j,N,t-i-N+1} \tag{4.50}$$

Aplicând ecuaţia (4.46), obţinem ecuaţia de actualizare a ponderilor sinaptice:

$$w_{i,j,t} = w_{i,j,t-1} + NV_{i,N,t-j-N} \cdot NV_{j,N,t-i-N} \tag{4.51}$$

Din expresia (4.51) deducem că indicii temporali aferenţi variabilelor NV depind de localizarea spaţială a celulei. Dacă vom considera că toate componentele vectorului sunt aplicate simultan pe latura de est a reţelei, celula nu va recepţiona corect componentele ($x_i$, $x_j$) ale vectorului din cauza diferenţei de lungime între cele 2 trasee (fig. 4.24).

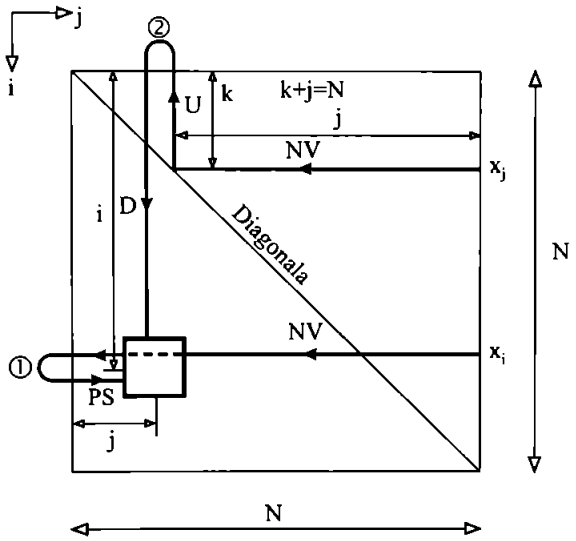



Fig. 4.24. Lungimea celor două căi de acces la celula i, j

Traseul 1 are lungimea N + j iar traseul 2 are lungimea N + i. Va trebui să introducem un decalaj temporal în aplicarea componentelor vectorului **x** care va compensa diferenţa de lungime. Aplicarea vectorului **x** se va face astfel:

$$NV_{k,N,t} = x_{k,t-k} \tag{4.52}$$

În concluzie, vom respecta constrângerea fixată deja de a aplica componentele vectorului în paralel şi simultan celulelor de pe diagonala RS, aşa cum indică figura 4.25.

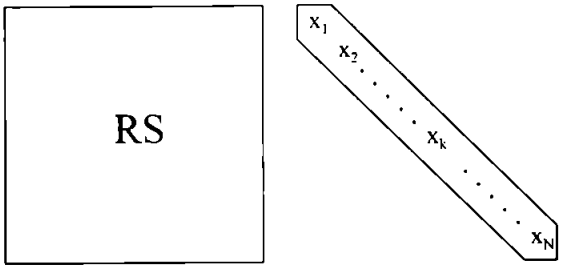



Fig. 4.25. Aplicarea vectorului **x**

Relaţia (4.51) poate fi rescrisă utilizând componentele vectorului **x**:

$$w_{i,j,t} = w_{i,j,t-1} + x_{t,N,t-j-N-i} \cdot NV_{j,N,t-i-N-j}$$
$$w_{i,j,t} = w_{i,j,t-1} + x_{i,\ t-i-j-N} \cdot x_{j,\ t-i-j-N}$$

Dacă eliminăm indicele temporal care este identic pentru cele două componente ale vectorului **x**, obţinem:

$$w_{i,j,t} = w_{i,j,t-1} + x_i \cdot x_j \qquad (4.53)$$

Circulaţia datelor şi operaţiile locale descrise, asociate cu decalajul temporal la introducerea componentelor vectorului **x**, permit calculul matricii **W** conform cu regula lui *Hebb*.

Ansamblul de ecuaţii (4.39) până la (4.48) constituie un SERU deoarece nu conţine decât translatări independente care operează asupra indicilor. Pentru a finaliza demonstraţia, mai trebuie definită o funcţie de timp T compatibilă cu dependenţele definite prin ecuaţiile elaborate, şi aleasă o funcţie de alocare A care va atribui operaţii concurente diverselor procesoare. Pentru a obţine reţeaua descrisă, funcţiile T şi A vor fi:

$$T(i, j, t) = t$$
$$A(i, j, t) = (i, j)$$

Să notăm că sunt posibile şi alte funcţii T şi A dar vor conduce la implementări diferite. Aceste funcţii corespund vectorilor de proiecţie **d** şi **s** definiţi în capitolul 3.

Am arătat că operaţiile (1) şi (2) din tabelul 4.1 pot fi implementate pe RS bidimensională definită, asigurând funcţionarea RS pentru regula lui *Hebb*. Vom arăta că aceste operaţii vor permite deopotrivă implementarea regulii Perceptronului.

### 4.3.4. Regula Perceptronului

Algoritmul Perceptronului, prezentat sintetic la 1.5, poate fi simplificat pentru a pune în evidenţă produsul scalar $\mathbf{W} \cdot \mathbf{x}$, ca indicator în procesul de ajustare a ponderilor. Regula poate fi rescrisă:

pas 1: Dacă ieşirea dorită d asociată vectorului de intrare **x** este negativă, schimbă semnul lui **x**:

- dacă d = 1 atunci,
  $\mathbf{z} = \mathbf{x}$
- dacă d ≠ 1 atunci,
  $\mathbf{z} = -\mathbf{x}$

pas 2: Dacă $\mathbf{W} \cdot \mathbf{z} < 0$ atunci,

$$\mathbf{W} = \mathbf{W} + \alpha \cdot \mathbf{z}$$

unde: $\alpha$ - constanta de învăţare

$\mathbf{W}$ - vectorul de ponderi sinaptice de dimensiune P

Primul pas al algoritmului va fi implementat în exteriorul RS. El constă în pregătirea datelor de intrare; semnul vectorului de intrare $\mathbf{x}$ va fi setat la valoarea celui al ieşirii dorite d (se obţine vectorul $\mathbf{z}$). Pasul 2 va fi executat pe RS care va fi configurată ca pentru faza de recunoaştere (paragraful 4.2.1). Reţeaua sistolică va fi identică cu cea prezentată la 4.2.1 dar matricea $\mathbf{W}$ se reduce la un simplu vector de dimensiune P (numărul de intrări în RN). Va exista apoi o singură ieşire (cea care corespunde singurului neuron din structura Perceptronului).

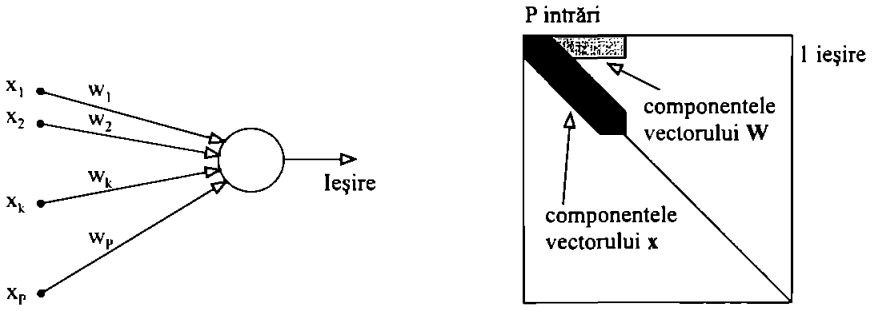



Fig. 4.26. Structura RS pentru regula Perceptronului

Pentru execuţia pasului 2 trebuie examinat semnul produsului scalar $\mathbf{W} \cdot \mathbf{z}$ care ne va indica dacă ponderile sinaptice trebuie ajustate sau nu. Această operaţie va fi realizată simplu printr-o funcţie de tip treaptă (fig. 4.27).

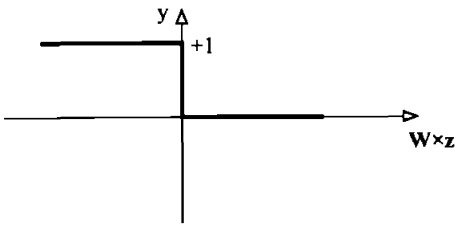



Fig. 4.27. Funcţia $\sigma$ de tip treaptă

Regula de ajustare a ponderilor sinaptice va fi:

$$\mathbf{W} := \mathbf{W} + \alpha \cdot \mathbf{z} \cdot \sigma(\mathbf{W}^T \cdot \mathbf{z})$$

unde:

$$\mathbf{W} = \begin{vmatrix} w_1 \\ w_2 \\ \dots \\ w_p \end{vmatrix}$$ - vectorul ponderilor sinaptice

$$\mathbf{z} = \begin{vmatrix} z_1 \\ z_2 \\ \dots \\ z_p \end{vmatrix}$$ - vectorul de intrare (cu semnul ajustat conform pasului 1)

Regula de ajustare a ponderilor revendică operaţia (1) din tabelul 4.1; corectitudinea execuţiei acestei operaţii pe RS definită a fost demonstrată la 4.3.3.

Pe parcursul calculului valorii de ieşire (y), vectorul de intrare (z) va continua să circule în reţea pe direcţia verticală. El va fi disponibil, pentru fiecare CS, exact în momentul în care şi indicatorul de modificare $\sigma(\mathbf{W}^T \cdot \mathbf{z})$ va fi prezent în celula respectivă. Pentru ajustarea ponderilor sinaptice va fi suficientă adăugarea valorii $\alpha \cdot z_i$ la valoarea curentă a ponderii $w_i$. Circulaţia datelor în RS este prezentată în figura 4.28.

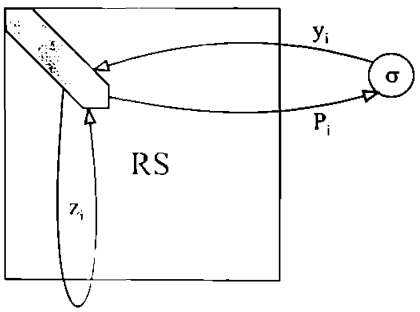



Fig. 4.28. Circulaţia datelor în RS pentru regula Perceptronului

Arhitectura fiecărei CS va fi identică cu cea dezvoltată pentru regula lui *Hebb*. Sumatorul rămâne nemodificat, iar multiplicarea cu constanta de învăţare $\alpha$ (subunitară) se poate realiza printr-o simplă deplasare la dreapta (împărţire cu o constantă supraunitară).

În cazul implementării unui singur Perceptron, RS va fi subutilizată. Am putea exploata paralelismul şi independenţa operaţiilor din RS pentru a implementa atâţia Perceptroni câte linii de celule există în RS, realizând astfel un sistem asociativ de tip memorie asociativă sau clasificator liniar. Numărul de intrări, pentru fiecare Perceptron, va fi identic şi este limitat superior de numărul de coloane din RS (figura 4.29).

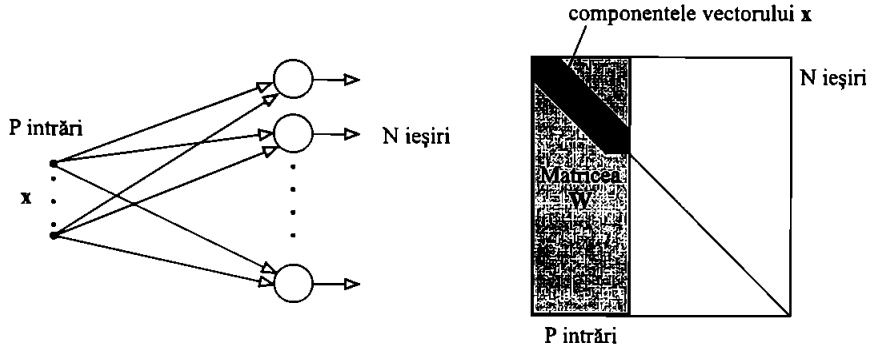



Fig. 4.29. Rețea formată din mai mulți Perceptroni

Această formă de rețea o regăsim în cadrul RN *feedforward* multistrat utilizate frecvent în aplicații. Analiza efectuată până în acest moment ne permite să extindem posibilitățile RS și la o altă regulă de învățare: auto-organizarea elaborată de *Kohonen*.

### 4.3.5. Regula lui Kohonen

Regula lui *Kohonen* (în varianta inițială), se aplică pe o structură cu două straturi de interconexiuni. Primul strat este startul de interconexiuni plastice asupra căruia va acționa algoritmul de învățare (paragraful 1.8). Al doilea strat este un strat de interconexiuni laterale, care cablează explicit un operator Laplacian destinat ortogonalizării vectorilor de ieșire. Vom implementa fizic ambele straturi pe aceeași RS bidimensională. Figura 4.30 indică dispunerea datelor în cadrul RS.

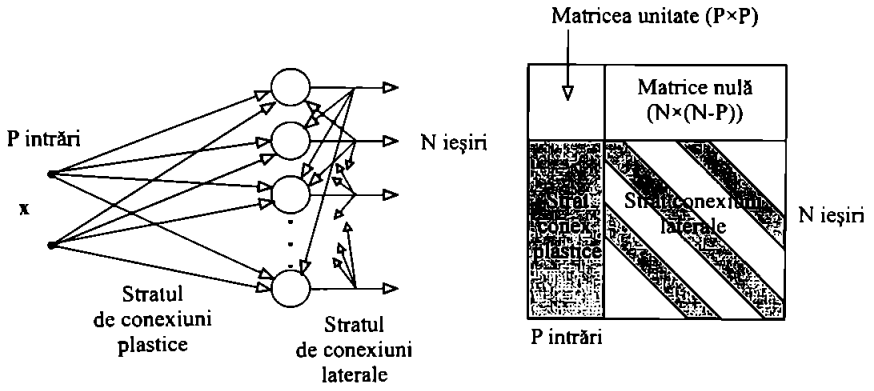



Fig. 4.30. Dispunerea matricilor de interconexiuni în cadrul RS

RS va conține patru matrici de coeficienți. Matricea unitate de dimensiune (P x P) va servi la recopierea valorilor vectorului de intrare cu scopul de a facilita circulația acestuia în cadrul RS. Matricea nulă de dimensiune (N x P) va interzice toate interacțiunile dintre valorile de intrare și potențialele neuronilor în timpul fazei de formare a zonei din jurul neuronului învingător (zona de activitate maximă). Matricea stratului plastic de dimensiune (N x P) va conține toate ponderile

sinaptice aferente stratului de conexiuni plastice. Matricea de conexiuni laterale este o matrice formată din benzi diagonale, de dimensiune (N x N) şi care defineşte vecinătatea de interacţiune dintre neuroni (funcţia de vecinătate). Ea va fi utilizată în faza de formare a zonei de activitate maximă. După definirea structurii RS, vom descrie traseele de comunicaţie prin care se vor pune la dispoziţia fiecărei CS informaţiile necesare ajustării ponderilor sinaptice.

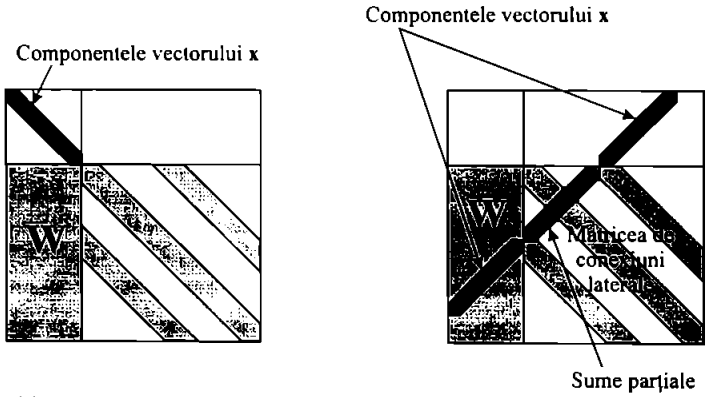
Regula de învăţare originală formulată de Kohonen se descompune în două faze de calcul:

- Evaluarea activităţii neuronilor în urma aplicării unui vector de intrare
- Actualizarea ponderilor sinaptice în funcţie de activitatea calculată şi vectorul de intrare aplicat.

Prima fază se descompune, la rândul ei, în 2 sub-faze:

- Calculul activităţii iniţiale (potenţialului) fiecărui neuron în urma aplicării unui vector de intrare. Aceasta înseamnă efectuarea produsului dintre matricea sinaptică **W** (N x P) şi vectorul de intrare **x**(P); rezultă un vector de lungime N ale cărui componente reprezintă activitatea iniţială a fiecărui neuron;
- Evoluţia şi stabilirea activităţii neuronilor prin interconexiunile laterale (N x N). Aceasta înseamnă calculul vectorului de activitate (de potenţiale) final; această etapă necesită un anumit număr de iteraţii pentru a asigura instalarea unei activităţi (potenţiale) stabile. Kohonen denumea această fază: formarea "bulei" de activitate.

Cele două sub-faze sunt descrise în figura 4.31.




a) poziţia iniţială a vectorului de intrare b) activitatea RS în timpul calculelor

Fig. 4.31. Calculul vectorului de activitate

Procesele ilustrate în fig. 4.31 sunt identice cu cele descrise anterior. Vectorul **x** este plasat în diagonala RS, tranzitează spre nord unde va fi rebuclat spre sud. Când atinge zona de coeficienţi sinaptici **W**, va fi generat vectorul de sume parţiale **PS** care va evolua până în momentul în care ultima componentă a vectorului **x** atinge latura de sud a RS.

Procesul de evoluţie şi stabilizare a "bulei" de activitate are o dinamică identică cu cea a fazei de recunoaştere aferentă reţelei *Hopfield*. Acesta constă în calculul iterativ al produselor dintre matricea de interconexiuni laterale şi vectorul de activitate obţinut anterior, până la stabilizarea

vectorului de activitate. Am arătat că această operaţie poate fi implementată pe RS definită (paragraful 4.2). În acest caz funcţia neliniară σ nu se va aplica după calculul fiecărui produs.

Pe durata acestor iteraţii, vectorul de intrare **x** trebuie conservat şi sincronizat cu mişcarea vectorilor de activitate. Structura pătrată a RS, cu căi de comunicaţie de lungimi egale, asigură această sincronizare, după cum rezultă din figura 4.32.

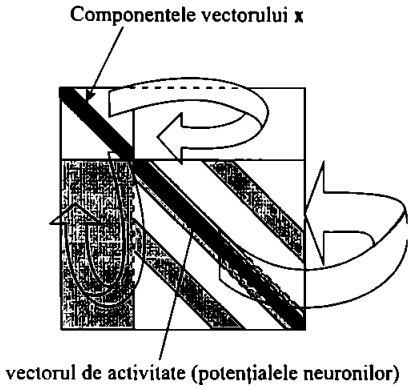



Fig. 4.32. Sincronizarea circulaţiei vectorilor de intrare şi de activitate în interiorul RS

După faza de stabilizare a "bulei" de activitate, vom observa că există neuroni cu activitate intensă (potenţial ridicat) şi neuroni cu activitate slabă. Potenţialul neuronilor este comparat cu o valoare de prag, convertit apoi de o funcţie treaptă în valorile +1 şi 0. Funcţia prag transformă potenţialul într-un indicator de stare care va indica neuroni ale căror ponderi sinaptice trebuie ajustate. Ajustarea va fi calculată pe baza componentelor vectorului de intrare **x**, prezent în primele P CS de pe diagonală (fig. 4.33).

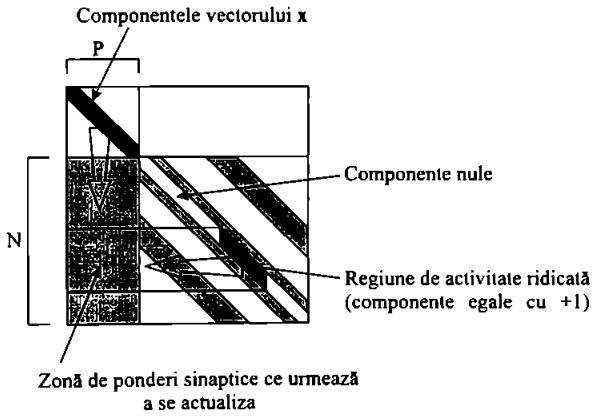



Fig. 4.33. Circulaţia datelor în RS pentru actualizarea ponderilor sinaptice

Se aplică sistemul de circulaţie a datelor prezentat anterior; vectorul de activitate (transformat în vector indicator de actualizare în momentul aplicării funcţiei treaptă) va traversa planul de conexiuni plastice. Zona de intersecţie dintre coloanele vectorului de intrare şi liniile vectorului

indicator de actualizare determină ponderile sinaptice ce vor fi actualizate. Ajustarea se va face după următoarea regulă:

- pentru neuronii i a căror potenţial este +1:

$$w_{ij} := w_{ij} + \alpha \cdot (x_i - w_{ij}) \tag{4.54}$$

- pentru ceilalţi neuroni (care au potenţialul 0), ponderile rămân nemodificate.

Considerând vectorul de activitate $y \in [0,1]^N$, formula de actualizare a ponderilor sinaptice devine:

$$w_{ij} := w_{ij} + \alpha \cdot (x_i - w_{ij}) \cdot y_j \tag{4.55}$$

Aceasta înseamnă că, pentru fiecare coeficient $w_{ij}$, se evaluează diferenţa dintre componenta $x_i$ a vectorului de intrare şi ponderea $w_{ij}$ aferentă. Se calculează apoi produsul dintre diferenţa obţinută şi componenta $y_j$ a vectorului de activitate. Dacă $y_j = 1$ atunci ponderea $w_{ij}$ va fi ajustată, dacă $y_j = 0$ atunci ponderea $w_{ij}$ rămâne nemodificată.

Învăţarea este deci integrată în sistemul de circulaţie a datelor. Avantajul metodei rezidă în localizarea fizică a operaţiilor care, printre altele, pot fi realizate în paralel. Ca şi în cazul funcţiei de recunoaştere, reţeaua sistolică va admite un flux de vectori **x** aplicaţi în sistem *pipeline* pe intrare. RS, compusă din $(N + P)^2$ celule sistolice poate conţine simultan $2(N + P)$ vectori de intrare procesaţi în tehnică *pipeline*. Suplimentar, structura care se bazează numai pe conexiuni locale permite o extensie simplă a RS prin simpla interconectare a mai multor subreţele sistolice.

Vom prezenta în cele ce urmează forma matricii de interconexiuni laterale şi vom discuta câteva probleme legate de normalizarea datelor.

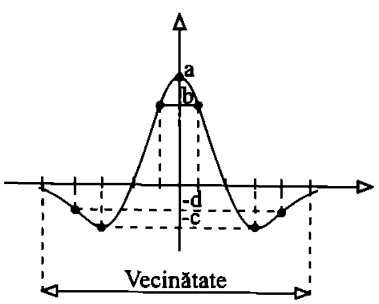
### A. Construcţia matricii de conexiuni laterale

Faza de formare a bulei se bazează pe un mecanism de competiţie între neuroni. Competiţia se realizează printr-o interacţiune între neuroni. Interacţiunea se defineşte prin valoare şi prin rază (în general o funcţie de tipul "Pălărie de mexican"). În cazul implementării noastre această interacţiune poate fi modelată printr-o matrice de interconexiuni care realizează o formă de convoluţie între celule. Pentru o reţea de N neuroni, această matrice are $N^2$ elemente. Ea este definită de către utilizator în funcţie de aplicaţia aleasă. Prin această matrice, se pot defini vecinătăţi de forme şi dimensiuni diferite.

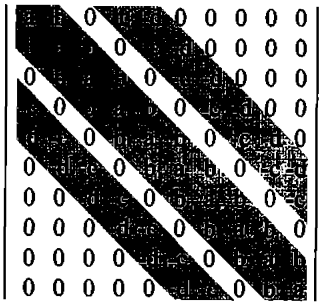
Să considerăm o reţea SOFM formată din 9 neuroni; această reţea poate fi organizată în linie (reţea monodimensională), în plan (reţea bidimensională) sau sub formă de cub (reţea tridimensională). Reţeaua SOFM realizează o proiecţie a unui spaţiu de dimensiune mare (spaţiu de intrare) pe un spaţiu de dimensiune redusă (spaţiu de ieşire). În spaţiul de reprezentare ales, mărimea zonei de interacţiune poate varia, adică fiecare neuron poate interacţiona cu vecinii săi până la o anumită distanţă.

Matricea de interconexiuni va fi deci o matrice formată din benzi, diagonale în cazul unui spaţiu monodimensional şi a unei vecinătăţi directe. Va fi o matrice pătrată construită astfel:

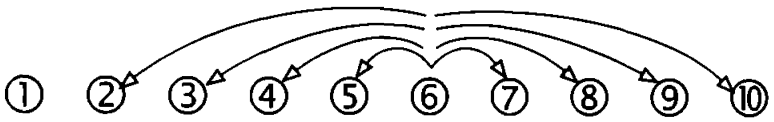
- se etichetează (numerotează în ordine) fiecare neuron din reţea;
- se completează elementele matricii de interconexiuni cu valorile de interacţiune dintre fiecare cuplu de neuroni.




a) funcția de interacțiune

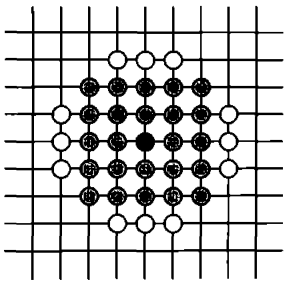


b) matricea de interconexiuni

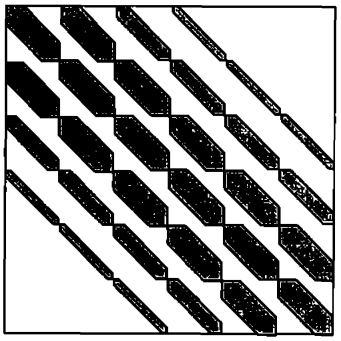



c) rețeaua SOFM liniară considerată

Fig. 4.34. Rețele SOFM monodimensionale; exemplu de vecinătate și matricea de interconexiuni asociată



a) Vecinătatea



b) Matricea de interconexiuni

Fig. 4.35. Rețele SOFM bidimensionale; exemplu de vecinătate și matricea de interconexiuni asociată

Soluția utilizării unei matrici de interconexiuni prezintă mai multe avantaje:

- mărimea vecinătății nu este fixă; ea depinde de elementele matricii;
- utilizatorul poate optimiza funcția de interacțiune în funcție de rezultatele obținute;
- RN poate fi extinsă fără nici o dificultate.

Dezavantajul major al acestei soluții rezidă în dimensiunea matricii de interconexiuni; ea are dimensiunea N x N pentru o rețea SOFM compusă din N neuroni.

Validitatea acestui principiu a fost experimentată prin simulări pe o reţea SOFM de 15 x 15 neuroni dispuşi în plan. După prezentarea vectorului de intrare toţi neuronii au prezentat un nivel de activitate (potenţial) dependent de gradul de apropiere (proximitate) dintre vectorul de intrare aplicat şi vectorul ponderilor sinaptice aferent neuronului în cauză. Pe durata formării "bulei de activitate" (procesul iterativ care implică matricea de interconexiuni), potenţialele au evoluat spre o stare stabilă de tipul celei prezentate în figura 4.36.

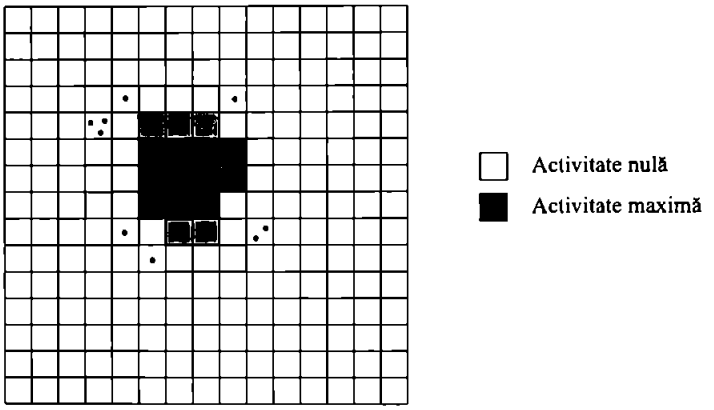



Fig. 4.36. Formarea unei zone de activitate stabilă

Toţi neuronii din zona (zonele) de activitate intensă vor suporta procesul de ajustare a ponderilor sinaptice asociate; restul neuronilor nu vor beneficia de ajustare.

**B. Normalizarea datelor**

Algoritmul lui *Kohonen* se bazează pe măsurarea distanţelor dintre vectorul de intrare şi vectorii de ponderi sinaptice. Această măsură se poate realiza prin produs scalar, calcul de distanţe euclidiene, Manhattan etc. Reţeaua sistolică propusă favorizează măsurarea distanţelor cu ajutorul produselor scalare. Vectorii de intrare şi de ponderi vor trebui să aibă norme comparabile astfel încât potenţialele neuronilor să reprezinte cât mai fidel diferenţele unghiulare dintre vectori.

Trebuie deci, teoretic, să se parcurgă un proces de normalizare a vectorilor de ponderi, după fiecare proces de ajustare. Normalizarea impune un calcul global care este incompatibil cu arhitectura distribuită a RS care lucrează exclusiv local. Această constrângere poate fi eliminată uşor dacă se operează cu vectori de intrare normalizaţi. În acest caz, a fost demonstrat [OJA 82] că norma vectorilor de ponderi sinaptice tinde statistic spre norma vectorilor de intrare. În baza acestui fapt, produsul scalar va reprezenta o măsură corectă a unghiului dintre vectori şi deci măsoară corect proximitatea acestora. Această proprietate a fost demonstrată prin simulări.

### 4.4. Evaluarea Reţelei Sistolice propuse

Calitatea unei implementări depinde de criteriile de evaluare utilizate. Vom utiliza un set de criterii cu scopul de a pune în evidenţă principalele calităţi precum şi deficienţele soluţiei sistolice propuse.

RS propusă reprezintă o implementare care integrează sinapse. În contextul unei implementări VLSI, paralelismul masiv reprezintă un factor esenţial. Paralelismul trebuie să fie cât mai fin posibil pentru a reproduce structura distribuită aferentă RN. Orice multiplexare a calculelor va conduce la reducerea calităţilor intrinseci aferente reţelelor conexioniste: reprezentare distribuită şi toleranţă la defect. RS propusă satisface aceste condiţii deoarece fiecare sinapsă este integrată într-o celulă sistolică proprie (cel mai fin paralelism posibil).

Viteza de procesare reprezintă de asemenea un factor determinant care trebuie să justifice soluţia propusă. Vom vedea că RS propusă este capabilă de performanţe deosebite (comparativ cu alte implementări) deoarece permite procesarea în tehnică *pipeline* a vectorilor de intrare.

Observabilitatea funcţionării este facilitată de circulaţia datelor în interiorul RS. Potenţialele neuronilor pot fi accesate (observate) pe latura de est a RS, înainte şi după aplicarea funcţiei de transfer. Controlul algoritmilor şi vizualizarea stării RN sunt de asemenea asigurate. Evoluţia ponderilor sinaptice nu poate fi observată "*on - line*" datorită căilor de date comune, care permit atât încărcarea matricii **W** cât şi tranzitarea componentelor vectorilor de intrare. Această situaţie conduce la ideea implementării unor căi de date separate pentru încărcarea matricii **W**. Aceasta va permite implementarea eficientă a RN multistrat deoarece în timpul calculelor aferente stratului curent, în paralel se poate încărca matricea de ponderi aferentă stratului următor. Vom analiza această soluţie în capitolul dedicat perfecţionării RS propuse.

Integrarea fazei de învăţare este posibilă prin compunerea operaţiilor de bază executate de RS. RS poate fi configurată după necesităţi, în ideea implementării celor mai diverse reguli de învăţare.

Extensibilitatea reţelei este posibilă datorită interconexiunilor exclusiv locale. Comunicaţiile se reduc la legături cu cei mai apropiaţi vecini ceea ce permite o creştere virtual nelimitată a dimensiunii RS. Vom vedea în cele ce urmează că formatul de reprezentare a datelor reprezintă un factor de limitare fizică, care poate fi în parte compensat prin utilizarea unei aritmetici care ţine cont de depăşirile de capacitate de reprezentare. Dimensiunea RS propuse (care integrează sinapse) reprezintă un factor limitativ care pare prohibitiv dacă îl comparăm cu soluţiile care integrează neuroni. Numărul de sinapse creşte proporţional cu pătratul numărului de neuroni dar această creştere este indispensabilă dacă dorim să asigurăm caracterul de generalitate al RS. În cazul circuitelor neuronale dedicate, va fi posibilă optimizarea numărului de CS în funcţie de forma matricii **W** (diagonală, simetrică etc.).

Simplitatea şi identitatea structurii celulelor sistolice reprezintă un mare avantaj în faza de proiectare. Odată construită o CS, ea va fi multiplicată pentru realizarea RS în ansamblu. După amplasament (pe diagonală sau în afara diagonalei RS), comunicaţiile CS vor fi specializate şi se va putea construi orice formă de reţea.

### 4.5. Arhitectura RS

#### 4.5.1. Specificaţiile celulei sistolice

Fiecare CS execută următoarele operaţii:

- emisie / recepţie date la / de la celulele vecine: Nord, Sud, Est, Vest;
- memorarea coeficientului sinaptic $w_{ij}$, a rezultatului parţial $PS_{ij}$, a elementelor vectorilor care se transferă în sens ascendent şi descendent în interiorul RS ($Up_{ij}$, $Down_{ij}$), şi a unui element din vectorul rezultat $NV_{ij}$;
- calculul unei sume ponderate ($PS_{ij}$) cu detecţia depăşirii capacităţii de reprezentare;
- compararea a două valori pentru detecţia convergenţei;
- modificarea ponderii sinaptice ($w_{ij}$) în procesul de învăţare.

### 4.5.2. Formatul operanzilor

Oricare ar fi faza executată în RS (recunoaştere sau învăţare), toate CS execută aceleaşi operaţii de calcul:

- în cadrul fazei de recunoaştere:

$$PS_{i,j} = PS_{i,j-1} + D_{i-1,j} \cdot w_{i,j} \tag{4.56}$$

- în cadrul fazei de învăţare:

$$w_{i,j} = w_{i,j} + \alpha \cdot D_{i-1,j} \cdot PS_{i,j-1} \tag{4.57}$$
$$w_{i,j} = w_{i,j} + \alpha \cdot (D_{i-1,j} - w_{i,j}) \tag{4.58}$$

Variabilele care intervin în cadrul algoritmilor neuronali sunt booleene (*Hopfield*, *Perceptron*) şi respectiv reale (*Adaline*, *Kohonen* etc.). Reprezentarea datelor se poate face în virgulă fixă sau în virgulă mobilă. Reprezentarea în virgulă fixă este caracterizată de un domeniu de reprezentare şi o precizie a reprezentării mai reduse; ea revendică însă dispozitive de adunare / înmulţire mai simple din punct de vedere structural. Reprezentarea în virgulă flotantă extinde domeniul de reprezentare şi precizia dar revendică dispozitive de adunare / înmulţire mai complexe. Evaluările făcute pe cale experimentală au relevat că o reprezentare în virgulă fixă pe 8 biţi oferă suficientă precizie (pentru algoritmii conexionişti studiaţi). Deci, registrele NV, Down şi Up vor avea lungimea de 8 biţi (într-o primă variantă care conduce la o structură mai simplă a CS).

Registrul coeficientului sinaptic (W) va conţine valori care rezultă din două surse posibile:

- învăţare externă;
- învăţare internă.

În cazul învăţării externe, calculul coeficienţilor sinaptici va fi realizat de calculatorul gazdă, în virgulă flotantă. În acest caz, reprezentarea în virgulă flotantă trebuie redusă la o reprezentare în virgulă fixă pe p biţi, după ce în prealabil coeficienţii fiecărei linii au fost normalizaţi prin procedura următoare:

- se va utiliza o funcţie MaxAbs(.) care generează cea mai mare valoare (absolută) dintre toate valorile coeficienţilor ce aparţin unei linii a matricii **W**;
- pentru i = 1, ..., N
  pentru j = 1, ..., N

$$w_{ij} = \frac{w_{ij}}{\text{MaxAbs}(\mathbf{W}_i)}$$

unde $\mathbf{W}_i$ - linia i a matricii **W**

Această operaţie de normalizare are 2 consecinţe asupra matricii **W**:

- pierderea simetriei;
- scăderea preciziei.

În cazul reţelei *Hopfield*, pierderea de simetrie este fără consecinţe pentru faza de recunoaştere. Dacă funcţia de transfer este sign(.), avem următoarea proprietate:

$$\sigma(a \cdot \mathbf{x}) = \sigma(\mathbf{x}) \qquad \text{dacă } a > 0 \tag{4.59}$$

dar: $$\frac{1}{\text{MaxAbs}(\mathbf{W}_i)} > 0 \tag{4.60}$$

ceea ce înseamnă că:

$$\sigma\left(\frac{1}{\text{MaxAbs}(\mathbf{W}_i)} \cdot \mathbf{x}\right) = \sigma(\mathbf{x}) \tag{4.61}$$

În ceea ce priveşte precizia, mai multe simulări au arătat că pierderea de precizie este fără consecinţe asupra comportamentului RN în faza de recunoaştere, atâta timp cât:

$$p \geq \log_2(N) + 1 \tag{4.62}$$

Optând pentru p = 8, rezultă că vom putea construi o RN de 128 neuroni fără a suporta o degradare a performanţelor din cauza preciziei de reprezentare.

Reprezentarea întreagă pe 8 biţi va conduce la o structură foarte simplă a celulei sistolice. În cele ce urmează vom prezenta etapele de proiectare a CS pentru p = 8. Pentru modele conexioniste mai complexe, valoarea p = 8 va fi revizuită şi CS reproiectată.

În cazul învăţării interne, calculele se vor executa în interiorul RS în virgulă fixă iar rezultatele vor fi aproximate prin trunchiere.

Registrul PS va avea o lungime care depinde de lungimea registrului W şi de numărul de neuroni simulaţi de reţeaua sistolică. Dacă notăm:

k - numărul de biţi alocaţi pentru reprezentarea componentelor vectorului **x** (lăţimea registrelor Down şi respectiv Up).
p - lăţimea registrului W (numărul de biţi)
m - lăţimea registrului PS (numărul de biţi)
N - dimensiunea reţelei sistolice (N x N celule)

în baza relaţiei (4.55) putem scrie:

$$m = k + p + \log_2 N \tag{4.63}$$

Pentru prima variantă de CS, vom avea m = 8 + 8 + 7 = 23 biţi. Vom aloca registrului PS 24 biţi (multiplu de 8).

Cei 24 biţi ai registrului PS trebuie transmişi de la o celulă la alta din direcţia vest spre est în timpul elaborării potenţialelor neuronilor. Transmisia se poate realiza în serie sau în paralel. În cazul paralel, transmisia se face într-o singură perioadă de tact dar revendică:

- un dispozitiv de înmulţire paralel;
- un bus de comunicaţie cu lăţimea de 24 biţi;
- 24 pini de intrare / ieşire pentru fiecare linie a matricii **W**.

În cazul serial, transmisia este de 24 ori mai lentă dar comunicaţiile între celule se vor realiza pe un singur fir. Sumatorul şi circuitul de înmulţire din interiorul CS vor lucra secvenţial şi vor avea o structură mult simplificată.

Vom reţine (pentru început) transmisia serială, pe care o justificăm pe baza a 2 considerente:

- structura internă a celulei sistolice trebuie să fie cât mai simplă pentru a putea implementa un număr cât mai mare de celule (sinapse);
- viteza de calcul a RS trebuie corelată cu lărgimea de bandă a canalului de comunicaţie dintre RS şi calculatorul gazdă. Dacă lărgimea de bandă a canalului de comunicaţie este inferioară vitezei de calcul a RS, RS va recurge la stări "*wait*".

Patru comenzi determină funcţionarea CS în 4 moduri de lucru:

RESET: şterge toate registrele interne
CLEARX: şterge toate registrele cu excepţia registrului W (coeficienţii sinaptici).
SHIFTW: conţinutul registrului W se deplasează cu o poziţie pe fiecare impuls de tact. Comanda va fi utilizată pentru încărcarea matricii sinaptice **W** în RS, din direcţia nord spre sud.
COMPUTE: conţinutul tuturor registrelor se deplasează cu un bit pe fiecare impuls de tact şi se elaborează suma parţială PS care se transmite, bit cu bit, spre est (serial).

Celulele diagonale se deosebesc de cele nondiagonale prin modul de comunicaţie. Pentru diagonale, intrarea din est este "reflectată" spre ieşirea de nord pentru a permite transmisia noii valori a vectorului **x** în RS, pentru iteraţia următoare. Pe ieşirea spre vest celula diagonală transmite rezultatul comparaţiei locale dintre vechea valoare $x_{ij}$ şi noua valoare $x_{ij}$ (evaluarea convergenţei algoritmului în faza de recunoaştere - reţeaua *Hopfield*). În faza de învăţare, noua valoare tranzitează de la vest spre est atât celulele diagonale cât şi cele nondiagonale.

Interfaţarea CS se face prin 4 intrări şi 4 ieşiri seriale, şi câteva semnale de comandă:

CLK - semnalul de tact principal
RESET - iniţializează la zero registrele UP, DOWN, NV, PS şi W
CLEARX - iniţializează la zero registrele UP, DOWN, NV şi PS
SHIFTX - deplasarea registrelor UP, DOWN şi NV
SHIFT PS - deplasarea registrului PS
SHIFT W - deplasarea registrului W (încărcarea matricii sinaptice)
DIAG - celulă diagonală (DIAG = 1) şi respectiv nondiagonală (DIAG = 0).

Structura celulei sistolice este redată în figura 4.37. Structura celulei este simplă şi poate fi multiplicată într-un număr mare de exemplare. Comunicaţiile se realizează numai cu celulele vecine. Celula poate fi programată diagonală (DIAG = 1) sau nondiagonală (DIAG = 0); acest lucru va permite realizarea RS reconfigurabile.

Vom expune în continuare funcţiile fiecărei unităţi din componenţa CS.

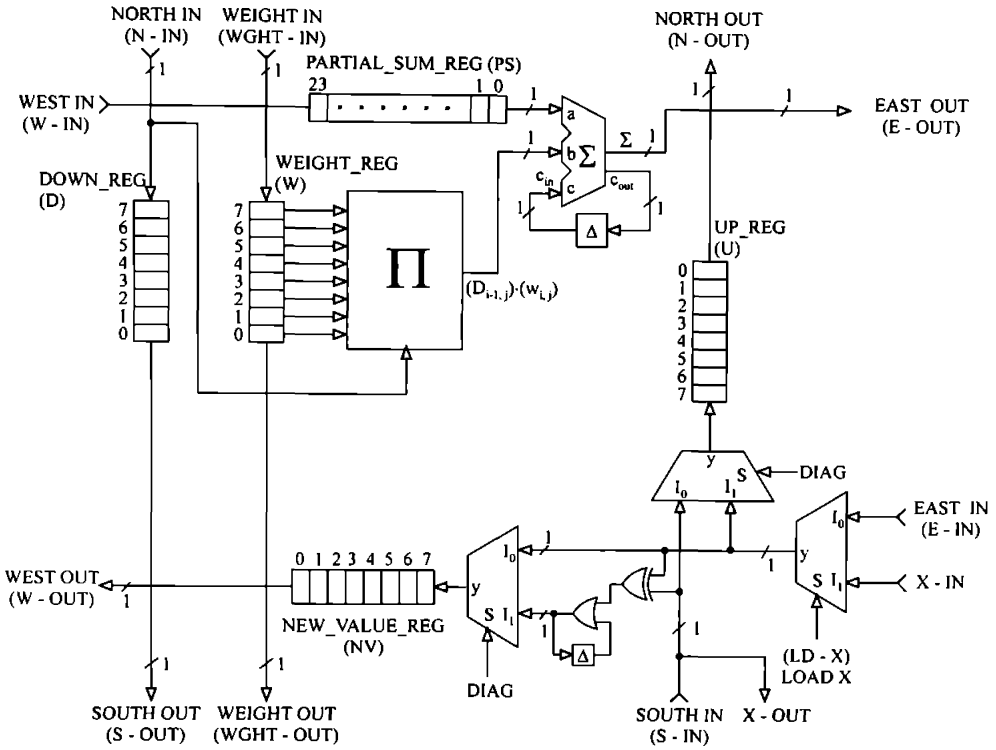



Fig. 4.37. Celula sistolică - schema bloc

### 4.5.3. Unitatea de calcul

Unitatea de calcul elaborează sumele parțiale locale în conformitate cu ecuația (4.56). Suma parțială de ieșire (transmisă spre est) se obține adăugând la suma parțială de intrare (transmisă dinspre vest) produsul dintre componenta vectorului **x** (transmisă dinspre nord) și coeficientul sinaptic local (memorat în registrul WEIGHT REG). Principiul este redat în figura 4.38.

Unitatea de calcul revendică un sumator și un înmulțitor. Pentru adunare putem utiliza un sumator serie, paralel sau cu generare anticipată a transportului. De asemenea, înmulțitorul poate fi serie, paralel, serie - paralel etc. După cum am precizat anterior, am optat pentru o transmisie serială a datelor între CS vecine. Din acest motiv, datele mobile vor revendica operatori seriali iar datele rezidente vor revendica operatori paraleli.

Sumatorul va însuma suma parțială cu rezultatul produsului dintre coeficientul sinaptic local și componenta vectorului **x** care intră în CS din direcția nord. Dacă dorim să simplificăm structura celulei vom renunța la memorarea locală a rezultatului (sumei) și îl vom transmite direct (serial) spre CS următoare. Drept consecință sumatorul va fi un simplu sumator serie.

Înmulțitorul lucrează cu o dată rezidentă (coeficientul sinaptic) și o dată mobilă (componenta vectorului **x**). Vom recurge deci la un înmulțitor serie - paralel (fig. 4.38) cu o structură simplă și optimală pentru CS cu transmisii seriale. Deînmulțitul (**W**) este aplicat în paralel unui sumator pe 16 biți (extensie semn), prin intermediul unui set de porți AND pilotate de bitul

curent al înmulţitorului (x). Cei 16 biţi ai produsului parţial obţinut pe tactul curent vor fi decalaţi cu o poziţie şi apoi adăugaţi produsului parţial obţinut pe tactul următor.

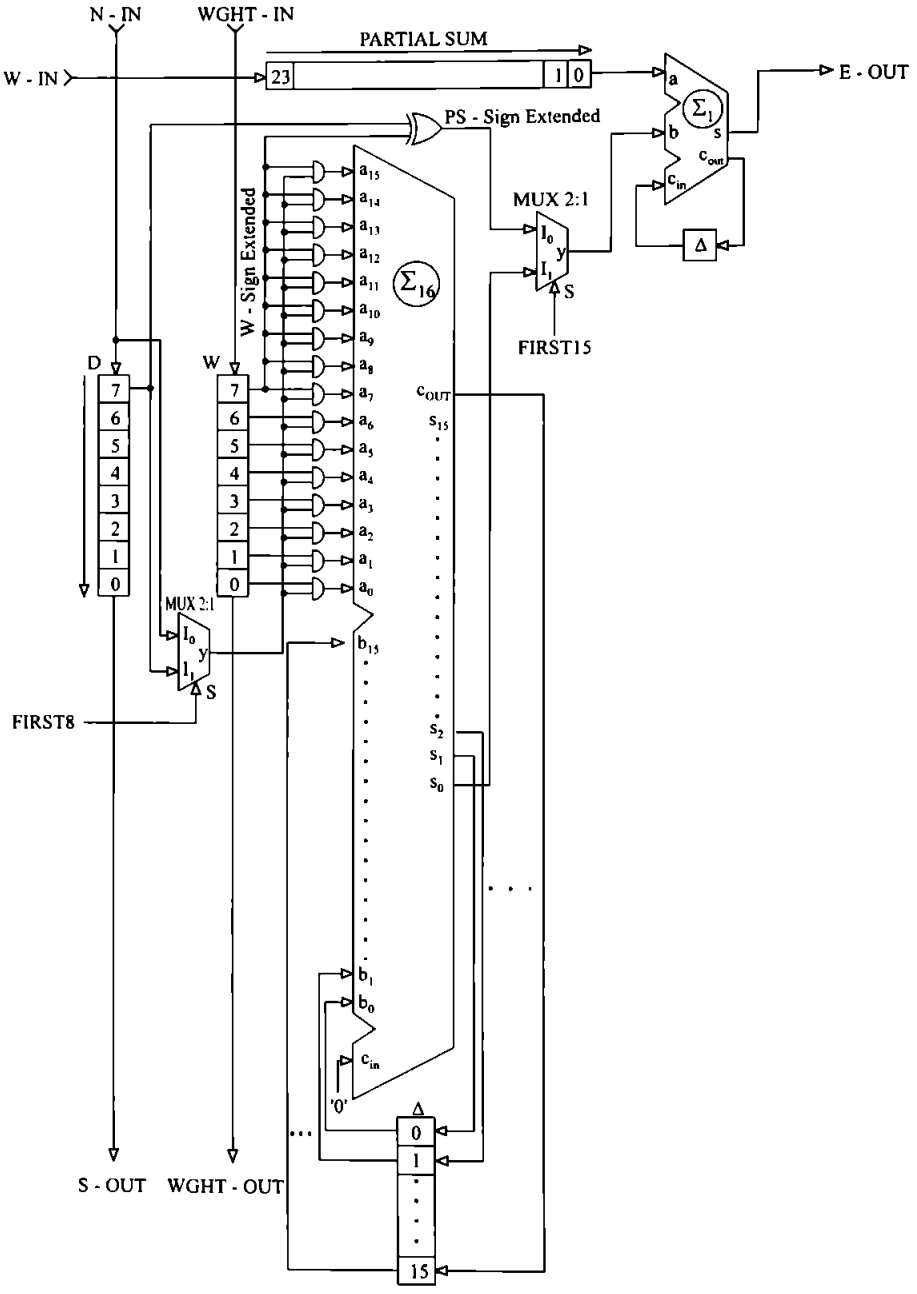



Fig. 4.38. Unitatea de calcul

Unitatea de calcul a CS va fi deci compusă din două sumatoare şi registrele de memorare / deplasare aferente.

### 4.5.4. Unitatea de comunicaţie

Unitatea de comunicaţie se ocupă de transmisiile de date între celule. Am stabilit două tipuri de comunicaţii asociate celor 2 tipuri de celule: diagonale şi respectiv nondiagonale (fig. 4.39).

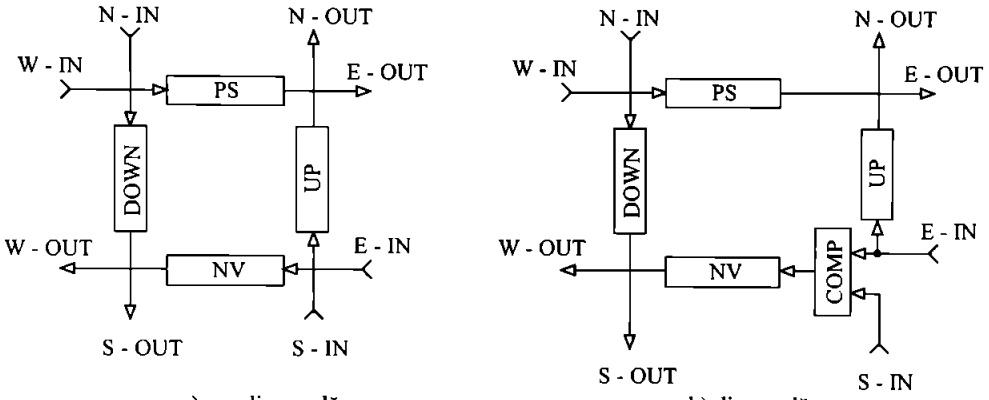



Fig. 4.39. Comunicaţiile celulei sistolice

Gestiunea comunicaţiilor reprezintă un mecanism esenţial în RS. Toate celulele sistolice au resurse de comunicaţie identice; fiecare celulă poate fi programată diagonală sau nondiagonală. Aceasta permite extensia RS printr-un simplu pavaj de circuite VLSI aşa cum se arată în figura 4.40.

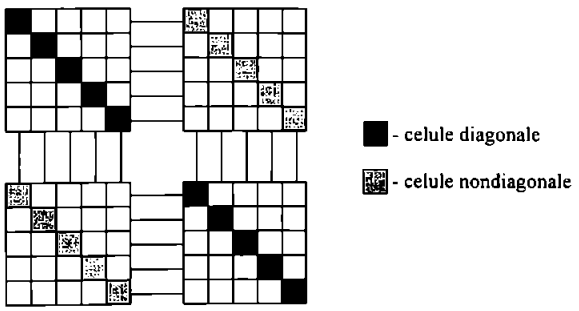



Fig. 4.40. Extensia RS prin conectarea mai multor circuite VLSI şi programarea corespunzătoare a celulelor diagonale şi nondiagonale

Mecanismul de comparare, activ în celulele diagonale, serveşte în faza de recunoaştere (reţeaua *Hopfield*) la detecţia identităţii dintre vectorii de stare care corespund la două iteraţii succesive (detecţia convergenţei). Procesul de comparare se va realiza printr-un comparator serie.

Concepţia părţii de comunicaţie se reduce la concepţia unor registre de deplasare şi a unui comparator.

### 4.5.5. Adaptarea arhitecturii pentru învăţare

Pentru regulile de învăţare derivate din regula lui *Hebb* (*Delta Projection*, *Adaline*, *Perceptron*) ajustarea arhitecturală necesară este minoră. Trebuie adăugate anumite căi de date care să faciliteze implementarea ecuaţiilor impuse de funcţia de învăţare. Arhitectura prezentată în fig. 4.41 şi care conţine 4 multiplexoare suplimentare satisface aceste cerinţe.

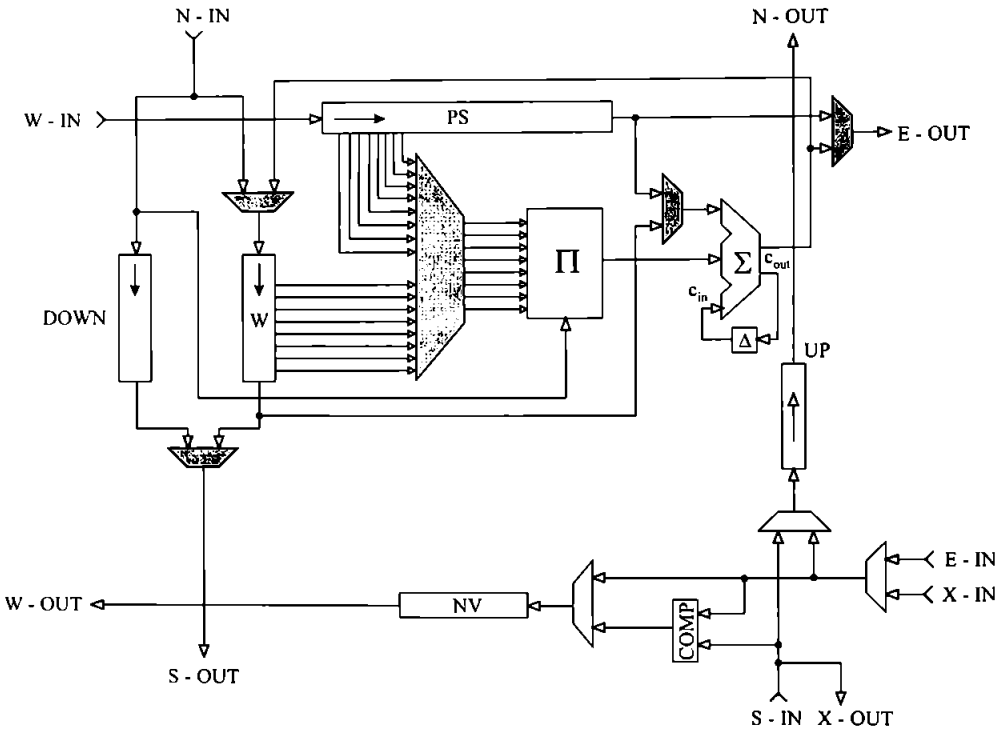



Fig. 4.41. Structura CS dotată cu funcţia de învăţare

Multiplexoarele evidenţiate nu provoacă o creştere spectaculoasă a complexităţii CS dar vor permite ajustarea ponderilor sinaptice pe baza regulii de învăţare dorite.

### 4.6. Evaluarea performanţelor

Pentru evaluarea performanţelor diferitelor implementări (dedicate algoritmilor conexionişti) se utilizează două măsuri:

- Numărul de conexiuni pe secundă (CPS - *Connections Per Second*). Trebuie să menţionăm că CPS are o semnificaţie dependentă de model (arhitectură, precizie etc.). De exemplu, pe o reţea de tip *Adaline* (directă şi fără rebuclaj) care operează în faza de recunoaştere, CPS măsoară numărul de operaţii de înmulţire + acumulare. În cazul unei reţele recurente care operează în faza de învăţare CPS nu mai reprezintă acelaşi lucru; va

reprezenta numărul de conexiuni ajustate pe secundă. De asemenea, CPS nu ţine cont de timpii de intrare / ieşire;

- Numărul de actualizări pe secundă (UPS - *Updates Per Second*) reprezintă o măsură mai globală care ţine cont de timpii de intrare / ieşire.

În RS bidimensională propusă, durata unei iteraţii se defineşte ca timpul scurs între momentul plecării şi momentul revenirii unui vector din / în diagonală. Dacă N reprezintă numărul de neuroni din reţea (RS va avea dimensiunea N × N), o iteraţie va fi executată în 2 · 24 · N perioade de tact (fiecare sumă parţială PS este codificată pe 24 biţi). Pentru RS sistolică bidimensională cele două măsuri se vor calcula astfel:

- Numărul de UPS (notat cu $\delta$) va fi:

$$\delta = 2NL / T_u \tag{4.64}$$

unde: $T_u = 2Nm / f$ (4.65)

$f$ - frecvenţa semnalului de tact
$T_u$ - timp de *update* (timp de actualizare a matricii **W**). Pentru actualizarea tuturor elementelor $w_{ij}$ se va parcurge un ciclu complet; pentru un ciclu complet vectorul **x** porneşte din diagonală şi revine în diagonală, după timpul $T_u$.
$m$ - dimensiunea PS (număr de biţi).
$L$ - rata *pipeline*-ului.

Dacă L = 1 (100 %), în timpul $T_u$, funcţia neliniară $\sigma$ (vezi cei 6 paşi din cadrul ciclului $T_u$ prezentaţi la 4.2) se va aplica la 2NL sume parţiale PS, conducând la 2NL actualizări.
Din (4.64) şi (4.65) rezultă că:

$$\delta = L \cdot f / m \tag{4.66}$$

ceea ce înseamnă că numărul de actualizări pe secundă este independent de numărul de neuroni (N).

- Numărul de CPS (notat cu $\gamma$) va fi:

$$\gamma = LN^2 / T_s \tag{4.67}$$

unde: $T_s = m / f$ (4.68)

$f$ - frecvenţa tactului (tactul de bază care va realiza şi deplasarea biţilor în PS)
$m$ - dimensiunea PS (număr de biţi).
$T_s$ - timp de calcul sinapsă (timpul de calcul al unui set de sume parţiale).

Dacă rata *pipeline*-ului ar fi de 100 % (L = 1) atunci se vor calcula $N^2$ sume PS în timpul $T_s$, ceea ce confirmă relaţia (4.67).

Pentru o frecvenţă de tact f = 100 MHz (valoare permisă de tehnologiile actuale), m = 24 biţi, L = 1 (100 %) - permis de RS propusă, şi pentru un număr de neuroni rezonabil N = 32, obţinem:

$$\delta = 1 \cdot 100 \cdot 10^6 / 24 = 4{,}1(6) \text{ MUPS}$$
$$\gamma = 1 \cdot 32^2 \cdot 100 \cdot 10^6 / 24 = 4{,}2(6) \text{ GCPS}$$

Aceste performanţe estimate ne îndreptăţesc să considerăm că RS analizată conduce la implementări VLSI utilizabile în aplicaţii reale. Conform [JUT 90], identificarea surselor radar reclamă aproximativ 625 MCPS, controlul unui robot aproximativ 350 KCPS, recunoaşterea cuvintelor aproximativ 24 MCPS. Se observă că măsurile de performanţă estimate depăşesc confortabil aceste valori. Cu titlu de comparaţie, în tabelul 4.3 prezentăm performanţele raportate pentru câteva implementări comerciale:

| | | |
|---|---|---|
| INTEL 80170NW - ETANN (*) | Circuit VLSI | 2 GCPS |
| Fujitsu Neurocomputer | Placă DSP | 500 MCPS |
| Cray X - MP | Parallel computer | 50 MCPS |

(*) ETANN - Electronically Trainable Analog Neural Network

Tabelul 4.3. Performanţele câtorva implementări comerciale

Datele prezentate în tabelul 4.3 trebuie interpretate cu anumite precauţii. Parametrul CPS dă o indicaţie aproximativă asupra performanţelor maşinilor citate. El nu reprezintă o măsură precisă deoarece el nu înglobează timpii de intrare / ieşire, timpii de calcul ai unor funcţii cum ar fi funcţia de transfer etc. În ceea ce priveşte RS propusă se impun trei precizări:

- Pentru RS, parametrul CPS este o funcţie care depinde de pătratul numărului de neuroni. Pentru N mare se poate ajunge teoretic la valori mult mai mari (de exemplu 68,2 GCPS pentru N = 128 neuroni);

- RS este bine adaptată execuţiei algoritmilor conexionişti dar funcţionarea sa depinde de lărgimea de bandă a canalului de intrare / ieşire (viteza de transfer a interfeţei cu sistemul gazdă);

- Numărul de circuite VLSI necesar pentru implementarea RS aferentă, de exemplu, unei RN compusă din 128 neuroni reprezintă un alt element de care trebuie să se ţină cont. Numărul de circuite depinde de numărul de sinapse integrate (creşte cu pătratul lui N). Totuşi, CS propusă implică comunicaţii seriale locale, are o structură foarte simplă şi RS (care se bazează pe comunicaţii strict locale) se pretează excelent pentru implementările VLSI.

### 4.7. Implementarea RS în structuri FPGA

Proiectul ce urmează a fi descris în acest paragraf a fost realizat în laboratorul de cercetare "*Système logiques et numériques*" din cadrul *Universităţii Libere Bruxelles*. Utilizând Sistemul de Dezvoltare al firmei XILINX, RS descrisă a fost proiectată, implementată în structuri FPGA, testată funcţional şi evaluată. Sistemul de dezvoltare al firmei XILINX este constituit dintr-un pachet de programe care permit proiectarea structurilor logice, testarea funcţională, evaluarea performanţelor structurilor proiectate şi implementarea acestora în circuite FPGA (*Field Programmable Gate Array*). FPGA este un circuit VLSI programabil; fiecare FPGA conţine un anumit număr de blocuri

logice (CLB - *Configurable Logic Block*). De exemplu, circuitul XC 4025 E produs de firma XILINX conține $2^{10}$ CLB; structura unui CLB este prezentată în figura 4.42.

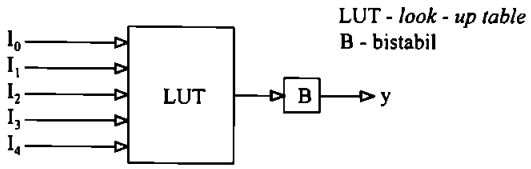



Fig. 4.42. Structura CLB din cadrul circuitului FPGA XC 4025 E

Fiecare CLB conține o tabelă (LUT - *look - up table*) cu 5 intrări și un element de memorare (B - circuit bistabil). Programarea circuitelor FPGA constă în:

- stabilirea conținutului tabelelor LUT din CLB-urile circuitului FPGA;
- stabilirea interconexiunilor dintre intrările și ieșirile diverselor CLB din structura FPGA.

RS proiectată în această lucrare este constituită dintr-un anumit număr de CS identice. Cu ajutorul sistemului de dezvoltare XILINX a fost proiectată și testată funcțional CS din fig. 4.37. Schema bloc (aferentă CS) rezultată în urma acestei etape de proiectare este prezentată în fig. 4.43.

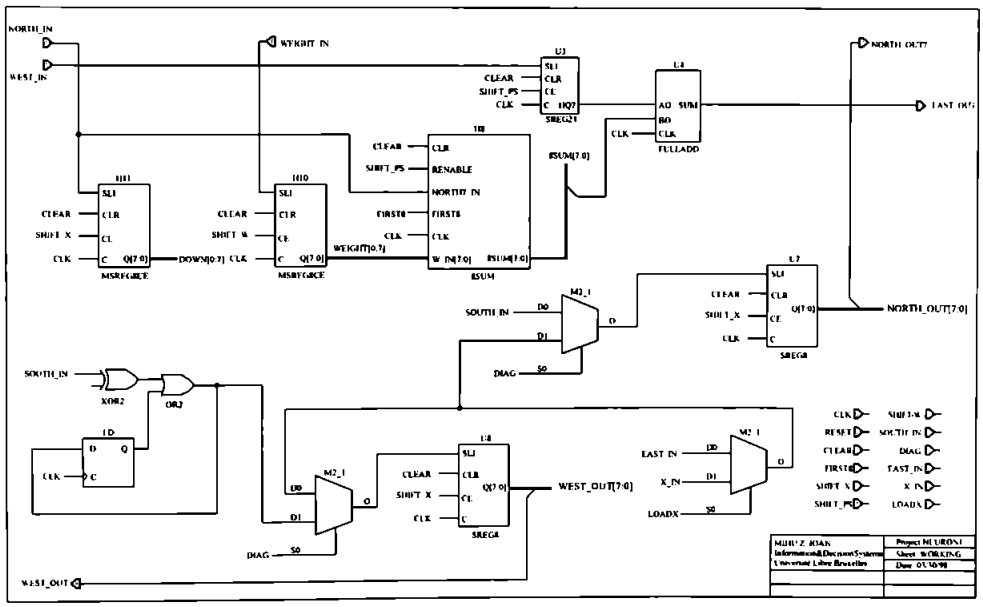



Fig. 4.43. Schema CS proiectată în sistemul de dezvoltare XILINX

### 4.7.1. Testarea funcțională a CS

Celula sistolică a fost în întregime simulată și testată funcțional în sistemul de dezvoltare XILINX. Pentru testarea funcțională s-au utilizat seturi adecvate de vectori de test. Cu titlu de

exemplu, vom prezenta procedura de testare funcţională a CS pentru un set de vectori de test. În cadrul fazei de recunoaştere funcţionarea CS este descrisă de ecuaţia (4.56). În figura 4.44 se prezintă programul de testare funcţională iar în figura 4.45 se prezintă setul de vectori de test utilizat. Fişierul *data.dat* din figura 4.45 defineşte următorii vectori de test:

$PS_{i,j-1} = 128$
$D_{i-1,j} = 005$
$w_{i,j} = 006$

În figurile 4.46, 4.47 şi 4.48 se prezintă rezultatele obţinute în cadrul fazei de testare funcţională. În fig. 4.46 se prezintă starea CS după iniţializare (RESET):

$PS_{i,j-1} = 0$
$D_{i-1,j} = 0$
$w_{i,j} = 0$

În figura 4.47 se prezintă starea CS în momentul iniţierii etapei de calcul (după încărcarea serială a registrelor):

$PS_{i,j-1} = 128$
$D_{i-1,j} = 005$
$w_{i,j} = 006$

În figura 4.48 se prezintă starea CS la finele etapei de calcul:

$PS_{i,j} = 158$
$D_{i,j} = 005$
$w_{i,j} = 006$

Rezultatele obţinute demonstrează corecta funcţionare atât a unităţii de calcul cât şi a unităţii de comunicaţie din componenţa CS. Aplicarea acestei proceduri pe un număr mare de seturi de vectori de test a permis testarea completă a CS. Toate testele efectuate au demonstrat funcţionarea corectă a CS atât în faza de calcul cât şi în faza de comunicaţie cu CS vecine.

```
|**************************************
|Script file for the neuro cell test
|10/April/1998
|Mihu Z. Ioan, Univesité Libre Bruxelles
|**************************************

clpr test.out
|delete_signals
restart
set_mode functional


vector R R[0:23]
vector DOWN DOWN[0:7]
vector WEIGHT WEIGHT[0:7]

radix dec R
radix dec WEIGHT
radix dec DOWN
```

```
watch CLK SHIFT_PS SHIFT_W EAST_OUT

|*****************
|CLK
|*****************
wfm CLK @0=1\d (10ns=inc by 1)*130

low SHIFT_W
low SHIFT_PS
low FIRST8
sim 20ns

|*************************
|LOAD D with 0 PS & Weight
|*************************

high SHIFT_X
(assign NORTH_IN < data.dat; sim 20ns)* 8

high SHIFT_PS
high SHIFT_W
(assign WEST_IN < data.dat; asssign WEIGHT_IN < data.dat; sim 20ns) * 8
low SHIFT_W
(assign WEST_IN < data.dat; sim 20ns) * 16
low SHIFT_PS

sim 20ns

|****************
|Compute
|****************

high SHIFT_PS
high SHIFT_X
high FIRST8
(assign NORTH_IN < data.dat; sim 20ns) * 8
low FIRST8
low SHIFT_X
sim 320ns
low SHIFT_PS
sim 20ns
```

Figura 4.44 Programul de test pentru CS

```
|***************************************
|Data file for the neuro cell test
|10/April/1998
|Mihu Z. Ioan, Univesité Libre Bruxelles
|***************************************

|INIT NORTH_IN d REG
0
0
```

```
0
0
0
0
0
0
|LOAD PS & LOAD D
0
0
|
0
1
|
0
1
|
0
0
|
0
0
|
0
0
|
0
0
|
1
0
|
0
0
0
0
0
0
0
0
0
0
0
0
0
0
|NORTH_IN
1
0
1
0
0
0
0
0
```

Figura 4.45 Un set de vectori de test

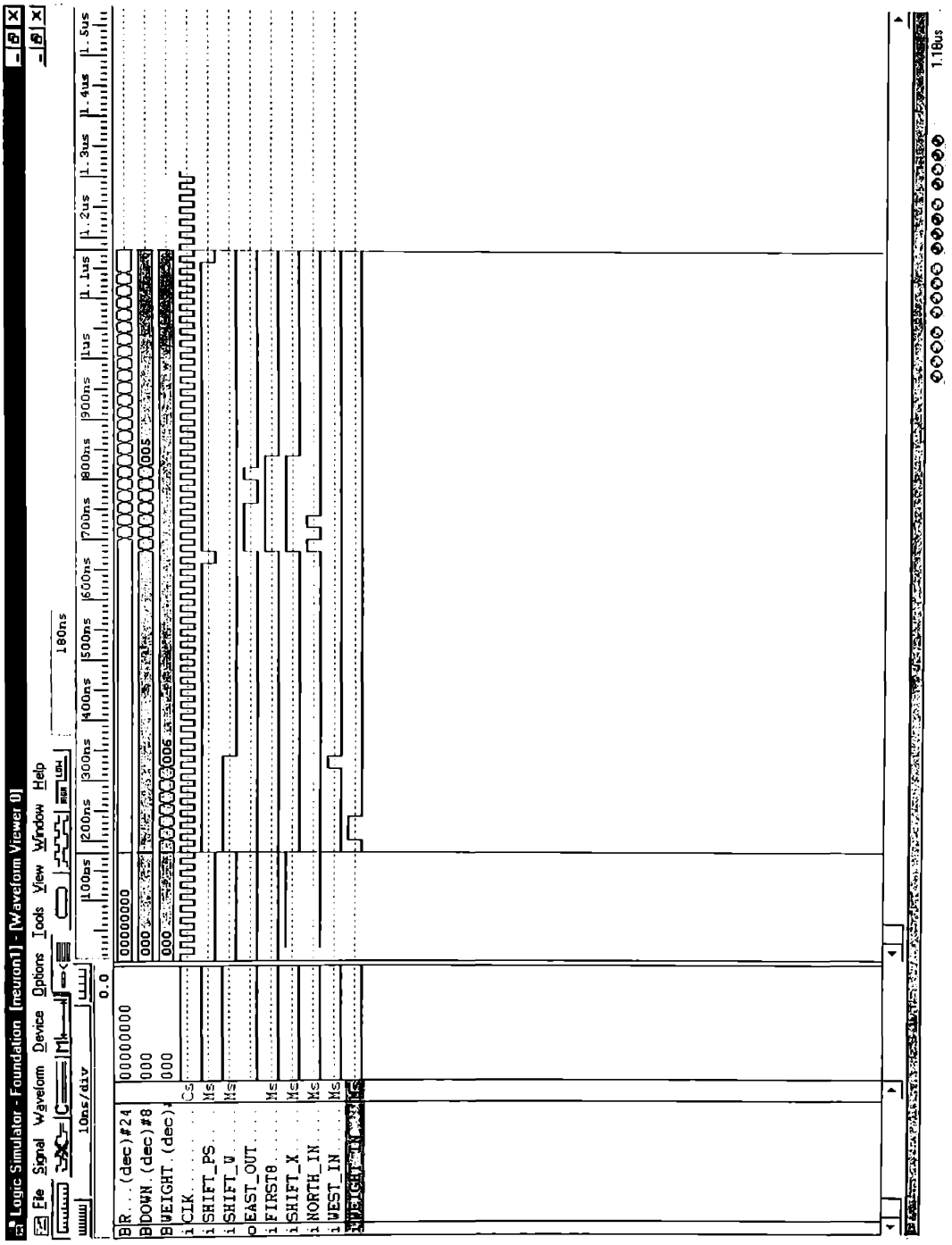



Fig. 4.46. Testarea funcțională a CS. Faza de inițializare: $PS_{i,j-1} = 0$; $D_{i-1,j} = 0$; $w_{i,j} = 0$

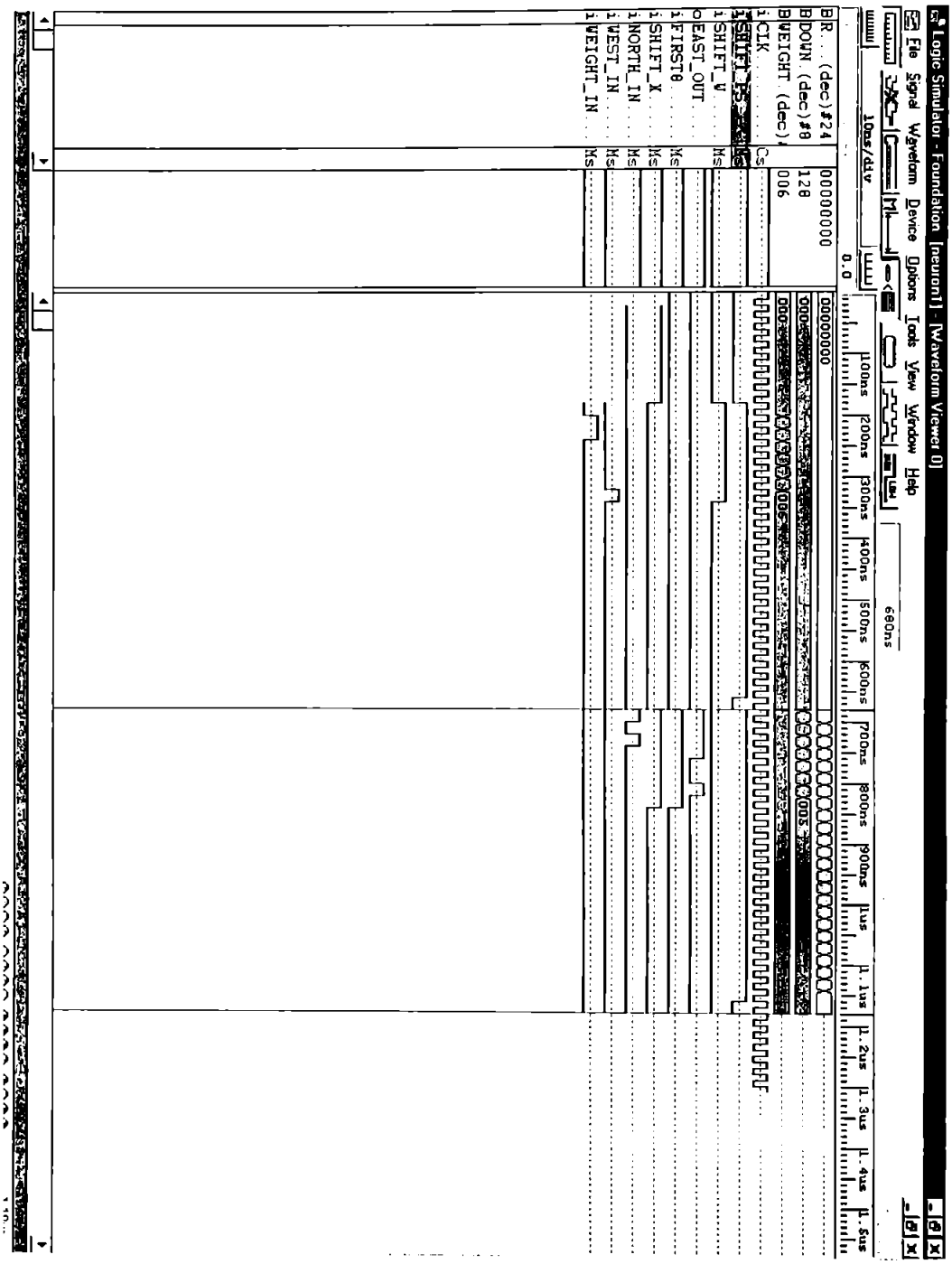



Fig. 4.47. Testarea funcțională a CS. Starea CS în momentul inițierii etapei CALCUL: $PS_{i,j-1} = 128$; $D_{i-1,j} = 005$; $w_{i,j} = 006$

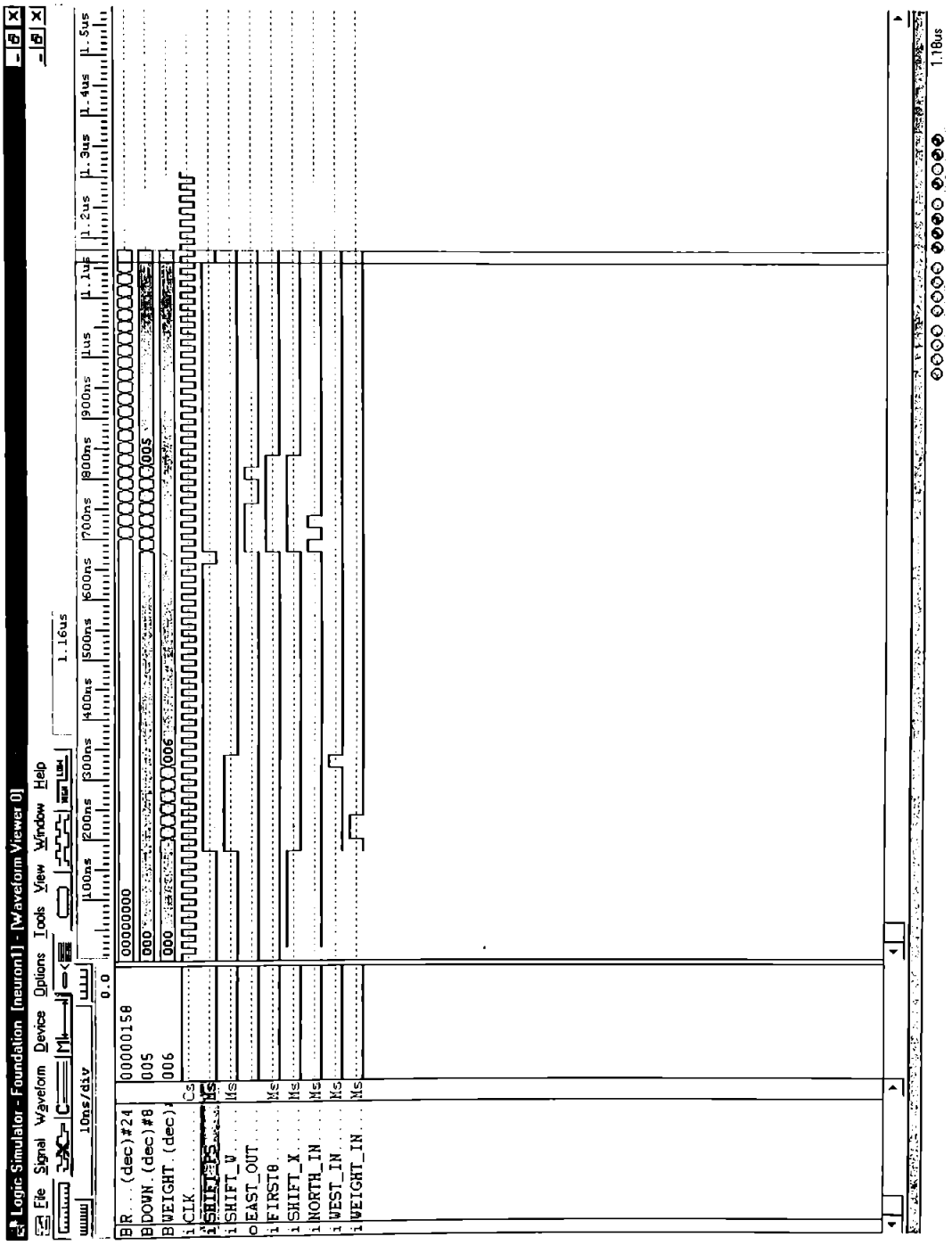



Fig. 4.48. Testarea funcțională a CS. Starea CS la finele etapei CALCUL: $PS_{i,j-1} = 158$; $D_{i-1,j} = 005$; $w_{i,j} = 006$

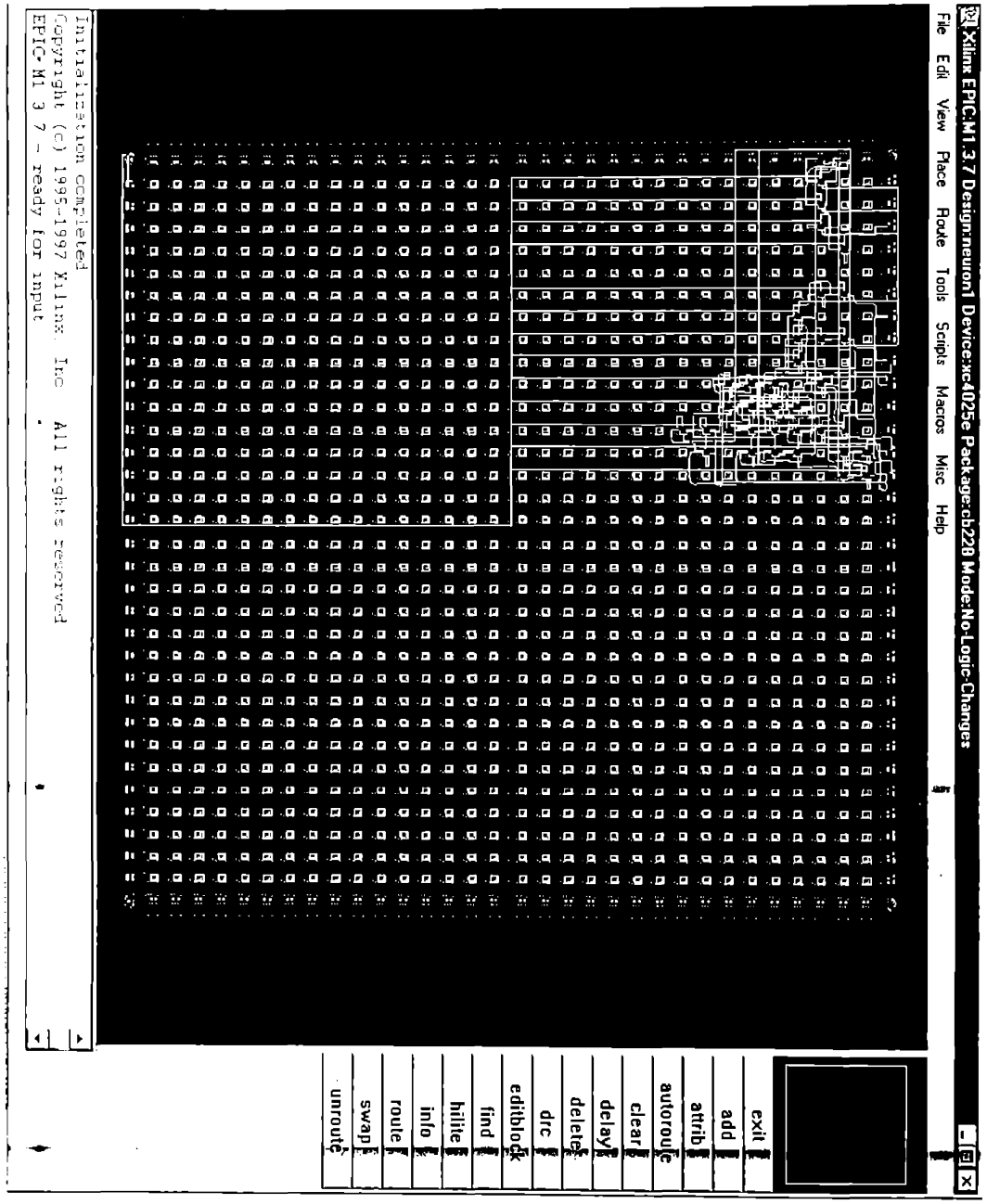



Fig. 4.49. Implementarea CS în circuitul FPGA XC4025E

### 4.7.2. Implementarea CS

CS proiectată și testată funcțional pe baza procedurilor expuse la 4.7.1. a fost implementată în circuitul FPGA XC 4025 E produs de firma XILINX. Circuitul XC 4025 E conține 1024 CLB organizate într-o matrice de dimensiune 32 x 32. Implementarea fizică a CS în acest circuit FPGA este prezentată în figura 4.49. Din cele 1024 CLB existente în circuitul XC 4025 E celula sistolică a ocupat 35 CLB ceea ce înseamnă că un singur circuit XC 4025 E va putea găzdui 29 celule sistolice. Aceasta înseamnă că într-un singur circuit XC 4025 E se poate implementa ușor o RS de 5 x 5 celule sistolice; extensia RS se poate realiza prin conectarea mai multor circuite XC 4025 E ca în fig. 4.40.

Implementarea în variantă FPGA prezintă un avantaj major: reconfigurabilitate dinamică. Implementarea în FPGA poate constitui soluția practică pentru construcția unei plăci neuronale acceleratoare care urmează a fi integrată într-un sistem gazdă. În aceste condiții RS (placa acceleratoare) nu trebuie să acopere, în orice moment de timp, toți algoritmii conexioniști cunoscuți deoarece o astfel de abordare ar complica peste măsură structura CS din componența RS și ar induce limitări severe în ceea ce privește dimensiunea RS implementabile pe o placă acceleratoare. În funcție de gradele de asemănare dintre diverși algoritmi, se pot forma mai multe clase de algoritmi și pentru fiecare clasă se poate proiecta o RS (CS) adecvată. În funcție de numărul de clase de algoritmi definite se vor obține mai multe arhitecturi de RS, dar toate aceste arhitecturi vor avea o trăsătură comună: simplitatea structurii CS (structura internă a CS diferă de la o RS la alta dar toate CS vor avea structuri interne simple). *Software*-ul de gestiune aferent plăcii acceleratoare va putea reconfigura dinamic RS implementată pe această placă (în funcție de necesități). Pentru o RN de tip SOFM, de exemplu, RS ar putea avea o arhitectură, iar pentru o RN de tip *Hopfield* RS ar putea avea o altă arhitectură. Reconfigurarea RS se va face dinamic în funcție de algoritmul neuronal ce urmează a se executa. Reconfigurabilitatea va conduce la o creștere a eficienței RS (plăcii acceleratoare). De vreme ce structura CS rămâne în permanență simplă (deși se modifică în timp), se vor putea obține (pentru fiecare algoritm neuronal în parte) viteze maxime de procesare în interiorul RS.

Circuitul XC 4025 E nu este un FPGA de densitate mare. În momentul actual firma XILINX produce o diversitate de circuite FPGA de dimensiuni (densități) diverse. De exemplu, circuitul XC 4062 XL conține 48 x 48 CLB și ar putea găzdui 65 celule sistolice. Aceasta înseamnă că XC 4062 XL ar putea găzdui o RS pătrată de 8 x 8 celule sistolice. În consecință, o placă acceleratoare cu 36 circuite XC 4062 XL ar putea constitui suportul fizic pentru o RS de 48 x 48 celule sistolice.

Circuitul XC 40250 conține 20.000 CLB și ar putea găzdui fără probleme o RS de 22 x 22 celule sistolice. Placa acceleratoare cu 36 circuite XC 40250 ar putea constitui suportul fizic pentru o RS de 132 x 132 celule sistolice. Toate aceste elemente demonstrează viabilitatea implementării RS cu structuri de tip FPGA.

## 4.8. Concluzii

Am arătat că RS bidimensională permite implementarea operațiilor vectoriale și matriciale revendicate de algoritmii conexioniști. Implementarea sistolică prezintă mai multe avantaje:

- permite implementarea fazei de recunoaștere pentru toți algoritmii clasici;
- permite implementarea celor mai diverse reguli de învățare. Această capacitate rezultă din faptul că RS realizează o înlănțuire de calcule matricial - vectoriale fără a specializa mașina suport pe funcții particulare. Structura RS este flexibilă și se bazează pe

comunicaţii strict locale. Forma pătrată a RS facilitează sincronizarea acestor comunicaţii locale;

- procesarea în tehnică *pipeline* a vectorilor de intrare este perfect posibilă deoarece toate operaţiile se efectuează cu date strict locale. Eficienţa RS va creşte spectaculos, putând atinge valori de 100 % în fazele de procesare staţionară (recurenţe);
- extensibilitatea RS se poate realiza printr-un simplu pavaj de circuite VLSI. În cazul comunicaţiilor seriale, numărul de pini necesari interconectării circuitelor VLSI este foarte mic.

# 5. CONSIDERAȚII PRIVIND PERFECȚIONAREA RS DEDICATE ALGORITMILOR CONEXIONIȘTI

## 5.1. Introducere

Modelul neuronului artificial este reprezentat de un dispozitiv cu N intrări și o ieșire. Ieșirea $y_i$ a neuronului i se calculează:

$$y_i = \sigma(p_i) = \sigma\left(\sum_{j=1}^{N} w_{i,j} \cdot x_j\right) ; \qquad i = 1, 2, \ldots, M \tag{5.1}$$

unde:

$w_{ij}$ reprezintă ponderea sinaptică asociată intrării j a neuronului i
$p_i$ - potențialul neuronului i
$x_j$ - componentele vectorului de intrare
$\sigma$ - funcția de transfer

Sub formă matricială ecuația (5.1) devine:

$$\mathbf{y} = \sigma(\mathbf{p}) = \sigma(\mathbf{W} \cdot \mathbf{x}) \tag{5.2}$$

Într-o RN, neuronii sunt organizați în straturi; neuronii din componența unui strat vor recepționa același vector pe intrări și vor avea ieșirile conectate la intrările neuronilor din următorul strat. Matricile sinaptice vor fi $\mathbf{W}^{[q]}$, unde q reprezintă indicele sau numărul stratului.

În faza de antrenament (învățare), matricile $\mathbf{W}^{[q]}$ sunt ajustate iterativ pe baza unei reguli de învățare. Una dintre cele mai simple este regula lui *Hebb*:

$$\mathbf{W} := \mathbf{W} + \alpha \cdot (\mathbf{y} \cdot \mathbf{x}^T) \tag{5.3}$$

unde $\alpha$ reprezintă constanta de învățare. Cele mai multe reguli practice de învățare se bazează pe reguli derivate din (5.3).

RN multistrat fac posibilă implementarea oricărei funcții $\mathbf{y} = \phi(\mathbf{x})$, $\mathbf{x}$ reprezentând vectorul de intrare în primul strat iar $\mathbf{y} = \mathbf{y}^{[L]}$ vectorul de ieșire din ultimul strat (L). Cea mai uzuală funcție de transfer $\sigma$ este tangenta hiperbolică sau funcția sigmoidală ((1.3) și (1.4)). RN învață funcția $\phi$ printr-un proces iterativ de prezentare a perechilor $\{\mathbf{x}_k, \mathbf{d}_k\}$, numite prototipuri sau perechi de antrenament. Vectorul $\mathbf{d}_k$ reprezintă vectorul dorit la ieșire atunci când pe intrarea RN se aplică vectorul $\mathbf{x}_k$.

Algoritmul *Error Back Propagation* (EBP) realizează o ajustare a ponderilor sinaptice cu scopul minimizării unei funcții de eroare (eroarea medie pătratică); se asigură astfel o coborâre pe suprafața de eroare spre un minim (local sau global) pe direcția și în sens invers gradientului funcției de eroare.

În acest scop se calculează un semnal de eroare asociat fiecărui strat:

$$\delta_i^{[L]} = (d_i - y_i^{[L]}) \cdot \sigma'(p_i^{[L]}) \tag{5.4}$$

$$\delta_i^{[q]} = \left( \sum_{k=1}^{M_{q+1}} w_{k,i}^{[q+1]} \cdot \delta_k^{[q+1]} \right) \cdot \sigma'(p_i^{[q]}) \; ; \qquad q = 1, 2, \ldots, L - 1 \qquad (5.5)$$

unde: $\sigma'(v) = \dfrac{d\sigma(v)}{dv}$

Ecuaţiile (5.4) şi (5.5) sunt valide pentru toţi neuronii i = 1, 2, ..., $M_q$ aferenţi stratului q. După ce eroarea este propagată dinspre ieşire spre intrarea RN prin toate straturile, ponderile sinaptice vor fi actualizate astfel:

$$\mathbf{W}^{[q]} := \mathbf{W}^{[q]} + \alpha \cdot \delta^{[q]} \cdot (\mathbf{y}^{[q-1]})^T \qquad q = 1, 2, \ldots, L \qquad (5.6)$$

unde: $\mathbf{y}^{[0]} = \mathbf{x}$.

Reţelele recurente complet conectate, cum este cazul modelului *Hopfield*, pot memora şi apoi detecta modelele de intrare. După aplicarea unui vector de intrare, ieşirea va fi calculată şi rebuclată pe intrare până când reţeaua va atinge o stare stabilă. Starea stabilă va reprezenta ieşirea **y** asociată vectorului de intrare **x**. O regulă de învăţare posibilă pentru acest model ar fi regula lui *Hebb* (ecuaţia (5.3)), forţând **y** = **x**.

Un alt model de RN este reţeaua SOFM (*Self Organizing Feature Map*) a lui *Kohonen*. Există mai multe variante pentru algoritmul SOFM. Neuronii sunt organizaţi sub forma unei grile regulate pe care se definesc anumite relaţii topologice (de exemplu, distanţe). Toţi neuronii vor recepţiona acelaşi vector de intrare. Pentru fiecare vector de intrare **x**, în spaţiul n - dimensional de intrare, se vor calcula distanţele de la **x** până la fiecare neuron i (vectorul ponderilor sinaptice aferent neuronului i). Cele mai uzuale distanţe sunt produsul scalar $\left( \sum_{j=1}^{N} w_{i,j} \cdot x_j \right)$ şi distanţa euclidiană:

$$y_i = \Delta_{in}(\mathbf{x}, \mathbf{W}_i) = \sqrt{\sum_{j=1}^{N} (x_j - w_{i,j})^2} \quad ; \qquad i = 1, 2, \ldots, M \qquad (5.7)$$

unde $\mathbf{W}_i$ reprezintă rândul i al matricii **W** (vectorul de ponderi sinaptice aferent neuronului i). Adesea, spaţiul de intrare are o dimensiune mult mai mare decât spaţiul topologic al reţelei SOFM. Neuronul învingător $I \in \{1, 2, \ldots, M\}$ va fi cel care deţine vectorul de ponderi sinaptice $\mathbf{W}_I$ cel mai apropiat de vectorul de intrare **x**, şi va fi identificat prin:

$$\Delta_{in}(\mathbf{x}, \mathbf{W}_I) \le \Delta_{in}(\mathbf{x}, \mathbf{W}_i) \qquad \forall\, i \in \{1, 2, \ldots, M\} \qquad (5.8)$$

ponderile sinaptice vor fi apoi ajustate:

$$\mathbf{W}_i := \mathbf{W}_i + \alpha \cdot \lambda(\Delta_{topo}(i, I)) \cdot (\mathbf{x}^T - \mathbf{W}_i) \; ; \qquad i = 1, 2, \ldots, M \qquad (5.9)$$

unde $\Delta_{topo}(i, I)$ reprezintă distanţa de la neuronul i la neuronul învingător I în spaţiul topologic al reţelei SOFM. Funcţia de vecinătate $\lambda$ va permite ajustarea ponderilor sinaptice numai pentru neuronii aflaţi în vecinătatea învingătorului I. Rezultatul obţinut, după prezentarea repetată a vectorilor de intrare, va fi o auto-organizare a reţelei SOFM în populaţii de neuroni a căror selecţie prin ecuaţia (5.8) este echiprobabilă. Într-o altă versiune a algoritmului SOFM, operaţiile de detecţie

a neuronului învingător I și de actualizare a ponderilor sinaptice pe baza funcției de vecinătate $\lambda(\Delta_{topo}(i, I))$, sunt înlocuite cu vectorul de ieșire aferent unei rețele recurente fixe care a fost lăsată să evolueze până la convergență. Această ultimă variantă a algoritmului SOFM este descrisă de ecuația (5.1) în care vectorul **x** aferent iterației curente va fi reprezentat de vectorul **y** aferent iterației anterioare.

### 5.2. Arhitectura RS

În capitolul 4 am proiectat o RS bidimensională dedicată algoritmilor conexioniști; funcționalitatea rețelei a fost demonstrată pentru faza de recunoaștere a rețelei *Hopfield* (RN recurentă care în faza de recunoaștere revendică un anumit număr de iterații până la convergență). Am arătat de asemenea că o mare parte a algoritmilor conexioniști pot fi descompuși în operații matricial - vectoriale executate cu mare eficiență pe RS bidimensională. În cele ce urmează vom extinde schema de bază a RS proiectate în capitolul 4, în ideea integrării mai multor algoritmi conexioniști (inclusiv cu faza de învățare). Figura 5.1 prezintă arhitectura de bază a RS, în care se evidențiază următorii operanzi:

- matricea ponderilor sinaptice **W**, rezidentă în cele N x N celule sistolice
- secvența de vectori de intrare $\mathbf{I}^v$ (în partea superioară)
- secvența de vectori de intrare $\mathbf{I}^h$ (pe latura de vest)

Rezultatele vor consta în actualizarea ponderilor sinaptice (faza de învățare) sau în vectorii de ieșire $\mathbf{O}^v$ sau $\mathbf{O}^h$ (faza de recunoaștere).

Evaluarea funcției de transfer $\sigma$ din ecuația (5.1) se va face cu ajutorul unei tabele (*look - up table*) în exteriorul RS. Similar, scăderea și înmulțirea din ecuația (5.4) se vor executa în exteriorul RS, cu ajutorul unei rețele liniare formată din unități aritmetice auxiliare.

#### 5.2.1. Fluxul feedforward de date

Prima clasă de operații executate în RS va produce ca rezultat un vector de ieșire $\mathbf{O}^h$ sau $\mathbf{O}^v$; aceste operații nu vor modifica matricea de ponderi sinaptice **W**.

Operațiile executate de celula sistolică $CS_{i,j}$ vor fi:

$$h_{i,j} := \psi(w_{i,j}, v_{i-1,j}, h_{i,j-1}) \quad (5.10)$$

$$v_{i,j} := v_{i-1,j} \qquad \text{pentru } i, j = 1, 2, \ldots, N \quad (5.11)$$

Condițiile de margine stabilite pe cele 4 laturi ale RS vor fi:

$$h_{i,0} := I_i^h \quad (5.12)$$

$$v_{0,j} := I_j^v \quad (5.13)$$

$$O_i^h := h_{i,N} \quad (5.14)$$

$$O_j^v := v_{N,j} \qquad \text{pentru } i, j = 1, 2, \ldots, N \quad (5.15)$$

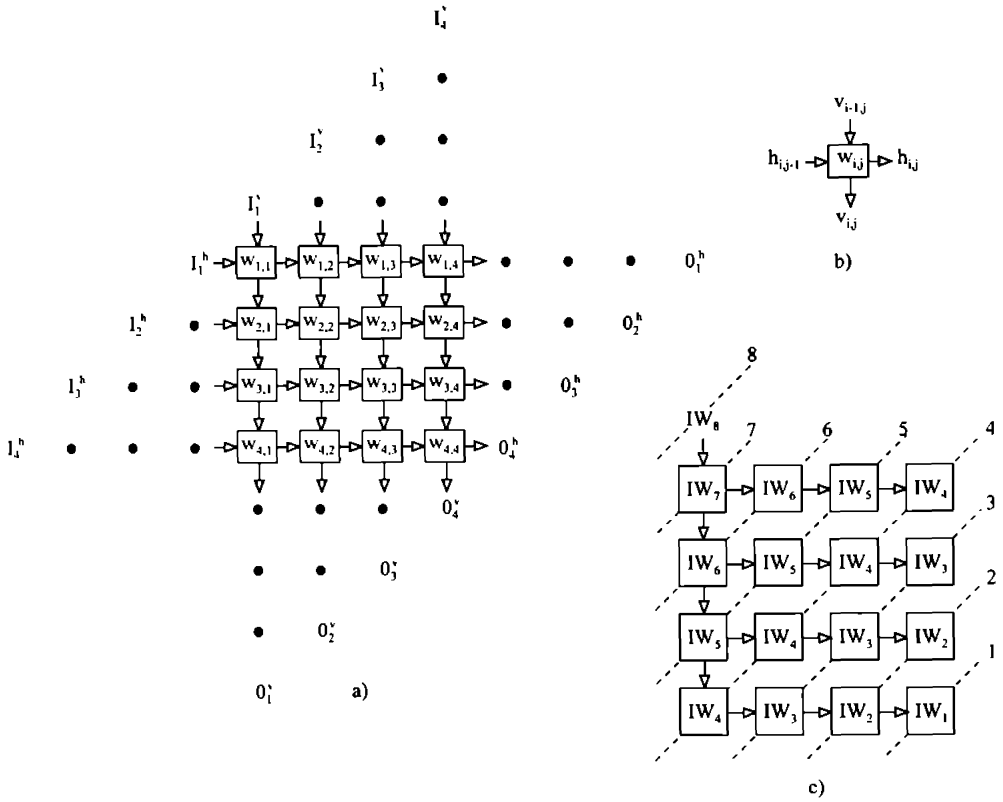



Fig. 5.1. a) Arhitectura RS. b) Operaţiile de intrare / ieşire aferente unei CS.
c) Fluxul sistolic de instrucţiuni în RS (IW - *instruction word*)

În tabelul 5.1 se prezintă setul de funcţii $\psi$ necesare implementării principalilor algoritmi neuronali; toate aceste funcţii utilizează fluxul sistolic de date descris matematic prin relaţiile (5.10) ÷ (5.15) şi grafic în figura 5.1. Simbolurile + ∞ şi - ∞ specifică valoarea maximă şi respectiv minimă reprezentabilă (limita maximă şi respectiv minimă a domeniului de reprezentare); aceste limite depind de formatul de reprezentare (virgulă fixă, virgulă flotantă) şi de numărul de biţi alocaţi pentru reprezentare. Operaţia *mprod* reprezintă o operaţie clasică de înmulţire a unei matrici cu un vector. Operaţia *euclidean* este similară unei operaţii *mprod* în care produsul este înlocuit cu pătratul unei diferenţe; această operaţie va permite calculul distanţelor euclidiene dintre un vector şi fiecare rând aferent matricii sinaptice **W**. Deoarece vectorul de distanţe rezultat este utilizat numai în ecuaţia (5.8) pentru identificarea unei valori minime, se poate renunţa la calculul rădăcinii pătrate. Includerea termenului adiţional $\mathbf{I}^h$ în cele două operaţii, *mprod* şi *euclidean*, este absolut necesară pentru cumularea rezultatelor obţinute pe matrici **W** diferite. Această cumulare de rezultate devine necesară atunci când RS fizică este mai mică decât RN implementată pe RS. În acest caz, matricea **W** va fi subdivizată într-un set de matrici care se vor încărca succesiv în RS iar rezultatele obţinute se vor cumula pentru a genera un rezultat global.

| Denumire operație | ψ(a,b,c) | Rezultatul obținut în modul normal de operare | Utilizare |
|---|---|---|---|
| *mprod* | $a \cdot b + c$ | $\mathbf{O}^h = \mathbf{W} \cdot \mathbf{I}^v + \mathbf{I}^h$ | ec. (5.1) |
| *euclidean* | $(b - a)^2 + c$ | $O_i^h = \sum_{j=1}^{N} \left(I_j^v - w_{i,j}\right)^2 + I_i^h$ | ec. (5.7) |
| *min* | $c$ dacă $c \le b$<br>$+\infty$ dacă $c > b$ | Dacă $I_I^h = \min_i (I_i^h)$ atunci $O_I^h = I_I^h$, toate celelalte elemente din vectorul $\mathbf{O}^h$ fiind setate la $+\infty$ | ec. (5.8) |
| *max* | $c$ dacă $c \ge b$<br>$-\infty$ dacă $c < b$ | Dacă $I_I^h = \max_i (I_i^h)$ atunci $O_I^h = I_I^h$, toate celelalte elemente din vectorul $\mathbf{O}^h$ fiind setate la $-\infty$ | ec. (5.8) |

Tabelul 5.1. Operațiile impuse de faza *feedforward* (recunoaștere) în modul de lucru normal

Operațiile *min* și *max* sunt operații de căutare al celui mai mic, respectiv mai mare, element dintr-un vector. Algoritmul utilizat impune ca vectorul investigat să fie introdus în RS simultan atât din direcția vest ($\mathbf{I}^h$) cât și din direcția nord ($\mathbf{I}^v$). Aceasta înseamnă că toate componentele introduse orizontal vor întâlni succesiv toate componentele introduse vertical. La nivelul fiecărei CS, pentru operația *min* de exemplu, componenta care intră în CS dinspre vest, va putea tranzita CS orizontal (nemodificată) numai dacă este mai mică decât componenta care intră în CS din direcția nord; în caz contrar componenta care tranzitează orizontal CS va fi saturată la $+\infty$. Cum toate elementele care tranzitează RS orizontal vor întâlni (succesiv) toate elementele care tranzitează RS vertical, pe latura de est a RS vom obține nesaturată numai cea mai mică componentă din vectorul investigat. Operația *min* va fi utilizată pentru calculul distanțelor euclidiene în timp ce operația *max* va fi utilizată la normalizarea ponderilor sinaptice și la calculul produsului scalar (produsul scalar maxim va reveni neuronului al cărui vector de ponderi sinaptice normalizat formează unghiul minim cu vectorul de intrare normalizat). Acest neuron va răspunde preferențial la vectorul de intrare respectiv.

Să notăm că retro-propagarea erorii (faza *backward*) revendică tot înmulțirea unei matrici cu un vector (ec. 5.5), dar matricea sinaptică utilizată în (5.5) este transpusa matricii utilizate în (5.1) (faza *feedforward*). Efectuarea calculelor care implică transpusa matricii $\mathbf{W}$ reprezintă o sarcină simplă care se poate realiza prin simpla interschimbare a rolurilor celor două fluxuri de date din RS (fluxul orizontal și respectiv vertical). Pentru operațiile care implică $\mathbf{W}^T$, relațiile (5.10) și (5.11) devin:

$$h_{i,j} := h_{i,j-1} \tag{5.16}$$

$$v_{i,j} := \psi(w_{i,j}, h_{i,j-1}, v_{i-1,j}) \qquad \text{pentru } i, j = 1, 2, \ldots, N \tag{5.17}$$

Rolul intrărilor va fi de asemenea interschimbat, iar ieșirea va fi:

$$\mathbf{O}^v = \mathbf{W}^T \cdot \mathbf{I}^h + \mathbf{I}^v \tag{5.18}$$

Deși toate celelalte operații din tabelul 5.1 (*euclidean*, *min*, *max*) pot fi executate în modul transpus (arhitectura RS permite), aceste noi operații care implică $\mathbf{W}^T$ nu se vor dovedi necesare.

### 5.2.2. Fluxul de date impus de ajustarea ponderilor sinaptice

A doua clasă de operații executate pe RS va realiza ajustarea ponderilor sinaptice ($\mathbf{W}$) fără a produce nici un alt rezultat pe ieșirile $\mathbf{O}^h$ sau $\mathbf{O}^v$. Operațiile executate de celula sistolică $CS_{i,j}$ vor fi:

$$h_{i,j} := h_{i,j-1} \tag{5.19}$$

$$v_{i,j} := v_{i-1,j} \tag{5.20}$$

$$w_{i,j} := \psi(w_{i,j}, v_{i-1,j}, h_{i,j-1}) \qquad \text{pentru } i, j = 1, 2, \ldots, N \tag{5.21}$$

Condiţiile de margine stabilite pe laturile RS sunt cele fixate prin ecuaţiile (5.12) şi (5.13). Tabelul 5.2 prezintă setul de funcţii $\psi$ necesare pentru implementarea fazei de învăţare (ajustarea ponderilor sinaptice).

| Denumire operaţie | $\psi(a,b,c)$ | Rezultatul obţinut în modul normal de operare | Utilizare |
|---|---|---|---|
| *hebbian* | $a + c \cdot b$ | $\mathbf{W} = \mathbf{W} + \mathbf{I}^h \cdot (\mathbf{I}^v)^T$ | ec. (5.3) şi (5.5) |
| *Kohonen* | $a + c \cdot (b - a)$ | $\mathbf{W}_i = \mathbf{W}_i + \mathbf{I}_i^h \cdot ((\mathbf{I}^v)^T - \mathbf{W}_i)$ | ec. (5.9) |

Tabelul 5.2. Operaţiile impuse de faza de învăţare (*backward*) în modul de lucru normal

Pentru ambele reguli de învăţare prezentate în tabelul 5.2, vectorul $\mathbf{I}^h$ este deja înmulţit cu constanta de învăţare $\alpha$ (în unităţi aritmetice auxiliare situate în exteriorul RS). Termenul $\lambda(\Delta_{topo}(i, I))$ din ecuaţia (5.9) va fi calculat astfel:

1. Un vector care va conţine cele m distanţe (m fiind numărul de neuroni din RN) va fi obţinut printr-o operaţie *min* (distanţe euclidiene) sau o operaţie *max* (produs scalar). O tabelă externă RS (*look - up table*) va fi utilizată pentru conversia tuturor valorilor $+\infty$ (în cazul *min*) şi respectiv $-\infty$ (în cazul *max*) la 0; doar valoarea minimă detectată (în cazul *min*) şi respectiv maximă (în cazul *max*) va fi convertită la valoarea 1.

2. Acest vector obţinut (cu m-1 componente 0 şi o componentă 1) va fi apoi multiplicat cu o matrice $\Lambda$ care exprimă în acelaşi timp şi distanţele topologice dintre neuroni şi funcţia de vecinătate utilizată. Elementele acestei matrici simetrice sunt $\Lambda_{i,j} = \lambda(\Delta_{topo}(i, j))$. Această formulare matricială a distanţei topologice are avantajul că permite efectuarea calculelor pentru orice tip de vecinătate (paragraful 4.5.3. A).

### 5.3. Fluxul sistolic al instrucţiunilor

RS este capabilă să efectueze o procesare în tehnică *pipeline*. Dacă numărul de nivele *pipeline* (marcate cu linii punctate în fig. 5.1. c)) este T (incluzând nivelele impuse de tabelele *look - up* şi de unităţile aritmetice auxiliare externe RS), atunci T vectori de intrare pot fi procesaţi concurent ajungându-se astfel la o eficienţă maximă a RS. Dacă un grup de T vectori se aplică succesiv pe intrarea RS, primul rezultat va fi disponibil pe ieşirea RS în ciclul care urmează ciclului de injectare în RS a ultimului vector de intrare. RS va genera în continuare următoarele T-1 rezultate în timp ce la intrarea RS va trebui setat un nou mod de operare. Procesarea SIMD (*Single instruction flow, multiple data flow*) clasică presupune golirea RS înaintea startării unui nou mod de operare (a unei noi instrucţiuni). Pentru a îmbunătăţi performanţele RS, fiecărui vector de intrare i se poate ataşa un cod de instrucţiune (IW - *instruction word*) care va fi propagat sistolic (tehnică *pipeline*) în interiorul RS. În figura 5.1. c) se indică modul de propagare a IW în RS împreună cu nivelele *pipeline* rezultate. Aceasta permite schimbarea progresivă a modului de operare în

interiorul RS, fără a fi necesară golirea RS la trecerea de la un mod de operare (instrucțiune) la alt mod (instrucțiune).

### 5.4. Structura celulei sistolice

Arhitectura RS şi structura CS sunt prezentate figura 5.2.

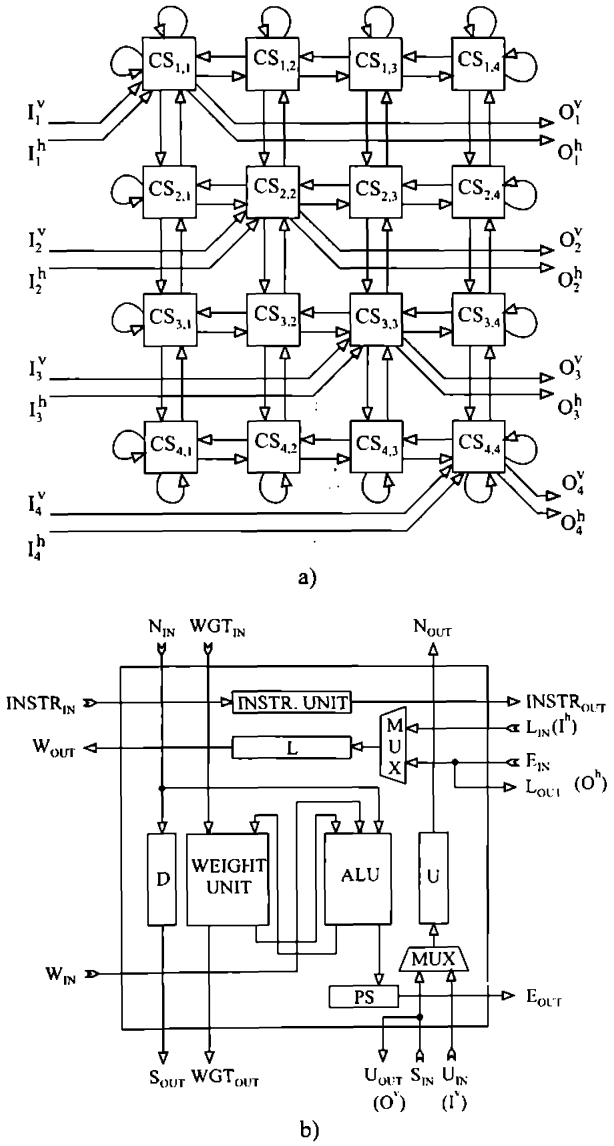



Fig. 5.2. a) Arhitectura RS 4 x 4. b) Structura CS

Spre deosebire de arhitectura de principiu prezentată în figura 5.1 a), operaţiile de intrare / ieşire sunt implementate cu ajutorul unor porturi locate în CS diagonale. Intrările $U_{IN}$ şi $L_{IN}$ şi ieşirile $U_{OUT}$ şi $L_{OUT}$ vor fi prezente în toate celulele sistolice (structură uniformă) dar vor fi utilizate numai în cazul celulelor diagonale; în celulele non - diagonale vor rămâne neconectate. Avantajul major al acestui sistem de intrare / ieşire constă în faptul că nu mai este nevoie de un *hardware* adiţional pentru diagonalizarea vectorilor de intrare şi de-diagonalizare a vectorilor de ieşire (capitolul 4). Toate componentele unui vector sunt introduse sau extrase simultan.

Mecanismul de transpunere, descris de ecuaţiile (5.16) şi (5.17), poate fi implementat cu ajutorul a două semnale de control furnizate de INSTRUCTION UNIT. Primul semnal va interschimba intrările $N_{IN}$ şi $W_{IN}$, iar al doilea va interschimba ieşirile $S_{OUT}$ şi $E_{OUT}$ (figura 5.3).

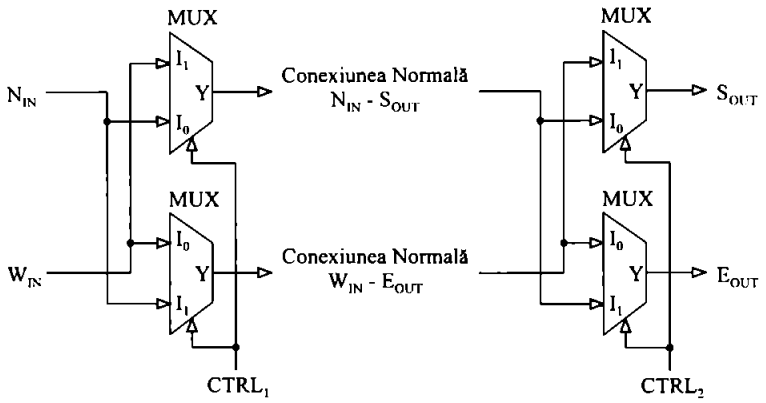



Fig. 5.3. Mecanismul care permite operarea cu transpusa matricii **W**

Când startează o operaţie care implică $\mathbf{W}^T$, intrările vor fi interschimbate (se activează $CTRL_1$). Registrele interne CS care conţin valori aferente operaţiei anterioare (care implică **W**) sunt emise pe căile normale de ieşire din CS. În următorul ciclu vor fi interschimbate şi ieşirile (se activează $CTRL_2$). Aceeaşi succesiune de operaţii va avea loc când RS revine la modul direct de operare (matricea **W**).

Comunicaţiile din interiorul RS sunt locale (între CS vecine) şi vor fi seriale (primul bit transmis va fi bitul cel mai puţin semnificativ). Protocolul serial se impune din două considerente:

- un înmulţitor paralel ar avea o logică foarte complexă şi va reduce numărul de CS implementabile într-un circuit (paragraful 4.5.4)
- comunicaţia paralelă ar revendica un număr mare de pini pentru circuitul VLSI.

### 5.4.1. Dublarea registrului W

În figura 5.2. b) sunt prezentate două căi de comunicaţie între celulele sistolice care nu sunt reprezentate şi figura 5.2. a):

- legătura dedicată ponderilor sinaptice (nord → sud)
- legătura dedicată instrucţiunilor (vest → est)

Legătura dedicată instrucțiunilor a fost deja discutată în corelație cu figura 5.1. c). Registrele dedicate ponderilor sinaptice (*weight unit*) sunt încărcate prin intermediul unor linii de comunicație dedicate (verticale) și independente de liniile de intrare standard. Cheia succesului pentru un *hardware* accelerator dedicat RN constă în abilitatea acestuia de a emula RN mai mari decât dimensiunea fizică pe care RS o pune la dispoziție. Unitatea *weight unit* conține două registre dedicate ponderilor sinaptice, unul conectat pe linia $WGT_{IN}$ - $WGT_{OUT}$ și celălalt utilizat în calcule (furnizează ponderea sinaptică pentru calculele efectuate în CS respectivă). Multiplexoarele din fig. 5.4 vor realiza interschimbarea celor două registre între două instrucțiuni consecutive.

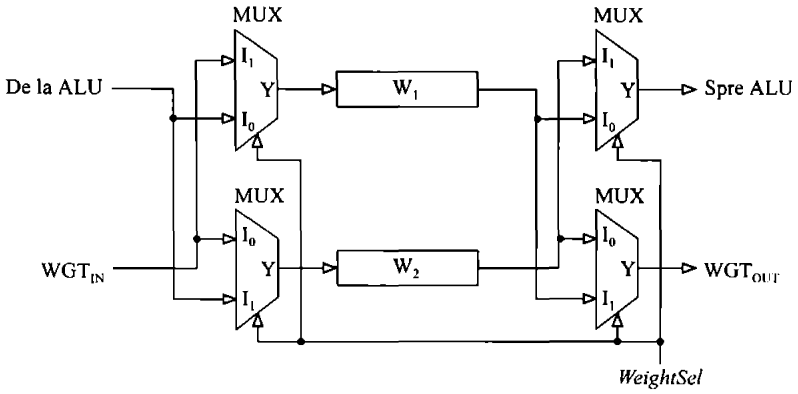



Fig. 5.4. Conectarea celor două registre din unitatea *Weight Unit*

În acest fel, o nouă matrice sinaptică poate fi încărcată în RS (de exemplu în registrele $W_2$) în timp ce RS procesează matricea anterioară (preluată din registrele $W_1$). În cazul RN mari, care depășesc capacitatea RS, cele 2 matrici sinaptice vor proveni din aceeași matrice care a trebuit să fie partiționată. În figura 5.5 se prezintă timpii de interschimbare pentru situația în care fiecare nouă epocă (care procesează T vectori de intrare) necesită încărcarea unei noi matrici sinaptice.

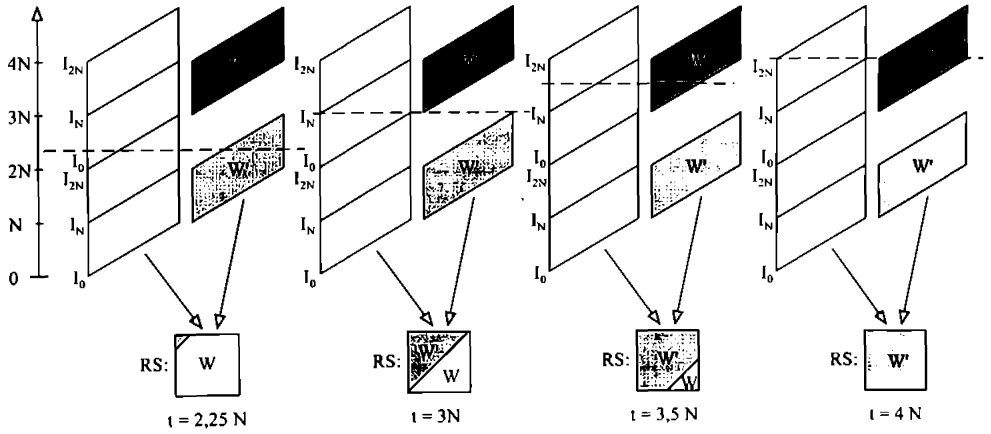



Fig. 5.5. Mecanismul de interschimbare a matricilor sinaptice

Pătratele din subsolul figurii reprezintă RS (N x N) iar zonele delimitate în interior indică matricea sinaptică utilizată în calcule de către celulele sistolice din zona respectivă. Fiecare epocă va conţine 2N vectori de intrare. 2N reprezintă şi latenţa *pipeline*-ului la nivelul instrucţiunii propagate în RS (fig. 5.1. c)). Grupurile de 2N vectori de intrare, fiecare compus din N elemente, sunt reprezentate prin intermediul unor paralelograme în figura 5.5. Vectorii de intrare sunt figuraţi ca şi cum ar fi injectaţi în RS din direcţia nord; în realitate aceştia sunt injectaţi în celulele sistolice de pe diagonala RS. Toate componentele unui vector intră simultan în RS (capitolul 4). Până în momentul t = 2N RS utilizează în calcul matricea sinaptică **W** (injectată anterior). De la t = 0 până la t = 2N se încarcă în RS (în registrele W libere) matricea sinaptică **W'**. În momentul t = 2,25 , 3/4 din celulele sistolice aferente primei linii din RS utilizează încă în calcul matricea **W**; matricea **W'** va intra în calcul progresiv începând cu celula sistolică situată în colţul de nord - vest al RS. În momentul t = 3N (la jumătatea celei de a doua epoci reprezentate în figură), jumătate din celulele sistolice utilizează în calcul matricea **W'**. Linia oblică împarte RS în două părţi: partea nehaşurată reprezintă celulele sistolice care utilizează în calcul matricea **W** şi cea haşurată reprezintă celulele care utilizează în calcul matricea **W'**. Linia oblică va intersecta în permanenţă setul de celule în care se realizează interschimbarea registrelor W (semnalul *Weight Sel* din figura 5.4). La momentul t = 4N toate celulele sistolice utilizează în calcul matricea **W'**. Încărcarea matricii **W''** demarează în momentul t = 3N iar în calcul aceasta va fi utilizată începând din momentul t = 4N. Acest proces revendică o epocă cu lungimea 2N vectori de intrare (egală cu latenţa *pipeline*-ului).

Matricile sinaptice diagonalizate sunt introduse din direcţia nord în RS. Trei registre **W** implementate în fiecare celulă sistolică ar fi permis încărcarea normală a matricilor sinaptice (nediagonalizate); un registru suplimentar creşte însă complexitatea CS şi reduce numărul de celule integrabile într-un singur circuit VLSI.

### 5.4.2. Precizia calculelor

Stabilirea preciziei care se impune în calcul este o problemă delicată. O primă variantă, motivată de simplitatea *hardware*-ului, constă în reprezentarea tuturor variabilelor în virgulă fixă (complement de 2). Cât despre numărul de biţi necesari în reprezentare, analizele teoretice conduc de regulă spre valori minime care însă nu garantează convergenţa pentru orice aplicaţie. În absenţa unor studii analitice clarificatoare, proiectanţii recurg adesea la simulări experimentale. În [ASA 90] şi [HOL 91] sunt prezentate simulări pentru determinarea preciziei revendicate de anumite aplicaţii tipice ale algoritmului EBP. Astfel de simulări au fost realizate şi pentru reţeaua SOFM a lui *Kohonen*. Concluziile acestor investigaţii pot fi sintetizate astfel:

- Ponderile sinaptice pentru operaţiile *mprod* şi *euclidean* (faza *feedforward*) oferă suficientă precizie dacă sunt reprezentate pe 16 biţi. Simulările au arătat că pentru aplicaţiile reţelei SOFM sunt necesari mai mult de 9 - 10 biţi în reprezentare. Este deci oportună alegerea primului număr multiplu de 8 superior acestei valori. În concluzie $N_W$ = 16 biţi. În mod similar, pentru vectorii de intrare se poate conveni $N_D$ = 16 biţi. $N_W = N_D$ deoarece, în cazul reţelei SOFM, ponderile sinaptice şi componentele vectorilor de intrare aparţin aceluiaşi spaţiu.

- Notăm cu n numărul maxim de intrări în RN. Parametrul n este independent de dimensiunea RS. În aplicaţiile reale în general n nu depăşeşte valoarea $256 = 2^8$. Numărul de biţi necesari reprezentării sumelor parţiale PS (în cadrul operaţiei *mprod*) va fi:

$$N_{PS} = N_W + N_D + \lfloor \log_2(n) \rfloor \qquad (5.22)$$

Obținem valoarea $N_{PS} = 16 + 16 + 8 = 40$ biți. Această valoare permite implementarea RN cu până la 256 intrări.

- Practic, datorită mecanismului de transpunere a matricii **W** și datorită operațiilor *min* și *max*, este de dorit ca toate registrele de date din structura CS să conțină 40 biți. Pentru a evita depășirile de capacitate în timpul calculelor este de dorit ca inițializarea RS să se facă cu $N_W = N_D = 16$ biți.

### 5.4.3. Unitatea ALU

ALU execută operațiile listate în tabelele 5.1 și 5.2 și va conține 3 blocuri: un înmulțitor, un sumator și un scăzător. Stabilirea structurii circuitului dedicat înmulțirilor reprezintă o chestiune delicată. Operația *mprod* nu pune probleme deosebite. Din acest punct de vedere se poate implementa un înmulțitor serie - paralel deoarece coeficientul sinaptic este rezident în celula sistolică și poate fi preluat în paralel, în timp ce celălalt factor este preluat de pe linia serială de comunicație nord - sud (sub formă serială). Din păcate această schemă nu poate fi utilizată și în operațiile de ajustare a ponderilor sinaptice (*hebbian* și *Kohonen*) și nici pentru operația *euclidean*, deoarece ambii factori sunt recepționați sub formă serială de pe liniile de comunicație intercelulare. O primă soluție ar consta în implementarea următoarei secvențe de operații:

- recepția serială a unui factor și memorarea sa într-un *buffer* în interiorul CS
- recepția serială a celui de-al doilea factor și execuția concomitentă a operației de înmulțire cu ajutorul înmulțitorului serie - paralel.

Această soluție permite utilizarea unui înmulțitor cu structură simplă (serie - paralel) dar implică o pierdere de performanță (un ciclu de transfer suplimentar pentru transferul și memorarea primului operand).

O a doua soluție ar consta în utilizarea unui înmulțitor serie - serie. Înmulțitoarele serie - serie tipice prezintă de regulă un număr de cicluri de latență între momentul aplicării primilor biți pe intrare și momentul obținerii primului bit pe ieșire [DAD 89]. Această latență conduce de asemenea la degradarea performanțelor RS. Problematica înmulțitoarelor serie - serie performante (zero cicluri latență) a fost tratată și în [IEN 94]. Fiecare CS revendică un înmulțitor serie - serie pe 17 biți; în cele ce urmează vom propune un astfel de înmulțitor care prezintă 2 avantaje majore:

- zero cicluri latență;
- extensia automată a semnului produsului atâta timp cât pe intrare cei doi factori își mențin extensia de semn.

Să notăm că ALU recepționează operanzii direct de pe liniile de comunicație externe (fig. 5.2) și încarcă rezultatele serial în registre (PS, $W_1$, $W_2$) plasate pe ieșirile ALU. Acest fapt simplifică implementarea operațiilor *min* și respectiv *max*, deoarece permite, pe ultimul ciclu de tact (la încheierea operației de comparare), saturarea la $+\infty$ sau $-\infty$ a valorii propagate în registrul PS.

#### A. Notații și simboluri utilizate

Vom considera că cei 2 factori aplicați pe intrarea înmulțitorului, X și Y, au N biți și sunt reprezentați în cod complementar (virgulă fixă). Produsul P obținut la ieșire va avea K biți, unde $K \geq 2N$. În figura 5.6 sunt reprezentate simbolurile logice utilizate. Simbolurile a) până la e) reprezintă elemente de întârziere (bistabile D care întârzie semnalul de la intrare cu un ciclu de tact). Pentru simplificare nu s-a mai reprezentat și intrarea de *clock*.

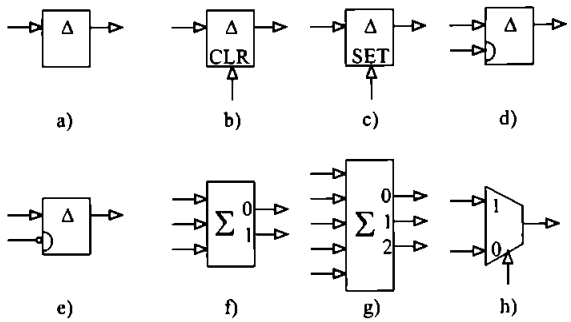



Fig. 5.6. Simbolurile aferente elementelor utilizate. a) Element de întârziere (bistabil D).
b) Element de întârziere cu intrare de ştergere sincronă activă *high*.
c) Element de întârziere cu intrare de set sincronă activă *high*.
d) Element de întârziere cu intrare de validare (*enable*) activă *high*.
e) Element de întârziere cu intrare de validare (*enable*) activă *low*.
f) *(3,2) counter* (*full - adder*). g) *(5,3) counter*. h) multiplexor 2:1

Simbolurile d) şi e) au un semnal de validare activ *high* şi respectiv *low*. Pentru a încărca valoarea de pe intrare în element, semnalul de validare trebuie să fie activ în momentul tranziţiei semnalului de *clock*.

Termenul *(n, k) counter* desemnează un dispozitiv cu n intrări şi k ieşiri; codul binar furnizat pe cele k ieşiri reprezintă numărul de intrări active. O definiţie alternativă a acestui dispozitiv ar fi aceea de sumator care însumează n cuvinte de un bit aplicate la intrare şi generează suma reprezentată pe k biţi la ieşire. Numărul de ieşiri va fi: $k = \lceil \log_2(n+1) \rceil$

### B. Algoritmi fundamentali

În figura 5.7 este redat algoritmul de înmulţire fundamental rezultat din definiţia codului complementar (complementul faţă de 2).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
| | | | | | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| + | | | | | | $x_2y_0$ | $x_1y_0$ | $x_0y_0$ |
| − | | | | | $x_3y_0$ | | | |
| + | | | | | $x_2y_1$ | $x_1y_1$ | $x_0y_1$ | |
| − | | | | $x_3y_1$ | | | | |
| + | | | | $x_2y_2$ | $x_1y_2$ | $x_0y_2$ | | |
| − | | | $x_3y_2$ | | | | | |
| − | | | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | |
| + | | $x_3y_3$ | | | | | | |
| | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

Fig. 5.7. Algoritmul de înmulţire dedicat numerelor reprezentate în cod complementar

Algoritmul calculează produsul final pe baza unor produse parțiale. Primii N - 1 biți ai produsului parțial se adună la suma de produse anterior obținută, iar bitul N (cel mai semnificativ) va fi scăzut din suma de produse anterior obținută. Excepție de la această regulă face ultimul produs parțial unde operațiile de adunare și respectiv scădere sunt inversate. Acest algoritm revendică un control complex (alternarea operațiilor de adunare și respectiv scădere) și din acest motiv nu reprezintă o bună soluție pentru implementarea în *hardware*.

O posibilă simplificare a algoritmului constă în extensia de semn la nivelul produselor parțiale până la lungimea K (lungimea produsului final), și execuția "normală" a operațiilor de adunare și respectiv scădere a produselor parțiale, păstrând de fiecare dată doar cei mai puțin semnificativi K biți. Algoritmul rezultat este reprezentat în figura 5.8 unde s-a considerat K = 9. Acest algoritm poate fi privit și ca pornind de la un operand X cu semn extins la lungimea de K biți.

| | | | | | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| + | | | | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_2y_0$ | $x_1y_0$ | $x_0y_0$ | |
| + | | | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_2y_1$ | $x_1y_1$ | $x_0y_1$ | | |
| + | | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_2y_2$ | $x_1y_2$ | $x_0y_2$ | | | |
| − | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | |
| | | | | $p_7$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ | |

Fig. 5.8. Algoritmul de înmulțire dedicat numerelor reprezentate în cod complementar cu X reprezentat cu extensie de semn

Totuși algoritmul din figura 5.8 prezintă dezavantajul că ultimul produs parțial trebuie scăzut în timp ce toate celelalte produse parțiale trebuie adunate. Acest dezavantaj poate fi eliminat dacă se realizează o extensie de semn prealabilă și la operandul Y. Se va intra în operație cu ambii operanzi extinși la K biți și după obținerea produsului final se vor reține doar cei mai puțini semnificativi K biți ai acestuia. Acest algoritm, prezentat în figura 5.9, este unul dintre cei mai utilizați algoritmi. Atât algoritmul din figura 5.8 cât și cel din figura 5.9 generează rezultatul cu extensie de semn.

| | | | | | | | | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_3$ | $x_2$ | $x_1$ | $x_0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | $y_3$ | $y_3$ | $y_3$ | $y_3$ | $y_3$ | $y_3$ | $y_2$ | $y_1$ | $y_0$ |
| + | | | | | | | | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_3y_0$ | $x_2y_0$ | $x_1y_0$ | $x_0y_0$ |
| + | | | | | | | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_3y_1$ | $x_2y_1$ | $x_1y_1$ | $x_0y_1$ | |
| + | | | | | | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_3y_2$ | $x_2y_2$ | $x_1y_2$ | $x_0y_2$ | | |
| + | | | | | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | |
| + | | | | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | |
| + | | | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | | |
| + | | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | | | |
| + | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_3y_3$ | $x_2y_3$ | $x_1y_3$ | $x_0y_3$ | | | | | | | |
| | | | | | | | | $p_7$ | $p_7$ | $p_6$ | $p_5$ | $p_4$ | $p_3$ | $p_2$ | $p_1$ | $p_0$ |

Fig. 5.9. Algoritmul de înmulțire a 2 numere reprezentate în cod complementar cu extensie de semn

### C. Înmulţitorul serie - serie

Utilizând algoritmul din figura 5.9, numărul de termeni care trebuie adunaţi pentru a genera un bit $p_i$ al produsului (numărul de termeni de pe o coloană) creşte liniar cu ponderea i. Pentru rangurile $p_i$ de pondere mai mare sau egală cu $2^{2N-1}$ cei mai mulţi termeni care trebuie adunaţi se datorează extensiei de semn şi au forma $x_{N-1} \cdot y_{N-1}$. Se poate arăta că regularitatea acestor termeni face posibilă ignorarea lor fără a afecta corectitudinea rezultatului.

Procedând într-o manieră similară celei expuse în [STR 82], produsul parţial obţinut după aplicarea bitului i poate fi exprimat sub o formă recursivă:

$$P_i = X_i \cdot Y_i = P_{i-1} + x_i \cdot Y_{i-1} \cdot 2^i + y_i \cdot X_{i-1} \cdot 2^i + x_i \cdot y_i \cdot 2^{2i} \qquad (5.23)$$

unde $X_i$ şi $Y_i$ reprezintă valorile celor 2 operanzi considerând doar biţii de la 0 la i (deci, $X_i = X \bmod 2^{i+1}$). Valorile iniţiale sunt: $P_{-1} = X_{-1} = Y_{-1} = 0$. Dacă extensia semnului este menţinută până în momentul declanşării următoarei înmulţiri, numai primii N-1 biţi aferenţi celor 2 operanzi vor trebui memoraţi (biţii de semn vor rămâne în continuare stabili pe intrări până la declanşarea următoarei operaţii de înmulţire). Această idee poate sugera un algoritm în care, în ciclurile $i \geq N$, operanzii parţiali se consideră extinşi numai până la bitul N-2 şi astfel termenii simetrici nu vor mai intra în sumă:

$$\bar{P}_i = \bar{P}_{i-1} + x_{N-1} \cdot Y_{N-2} \cdot 2^i + y_{N-1} \cdot X_{N-2} \cdot 2^i \quad \text{pentru } i \geq N \qquad (5.24)$$

$$\bar{P}_i = P_i \qquad \text{pentru } i < N \qquad (5.25)$$

Eroarea indusă prin utilizarea relaţiei (5.24) în locul relaţiei (5.23), în ciclul $j \geq N$ va fi:

$$\begin{aligned} &\left(P_j - P_{j-1}\right) - \left(\bar{P}_j - \bar{P}_{j-1}\right) = x_j \cdot Y_{j-1} \cdot 2^j - x_{N-1} \cdot Y_{N-2} \cdot 2^j + y_j \cdot X_{j-1} \cdot 2^j - \\ &- y_{N-1} \cdot X_{N-2} \cdot 2^j + x_j \cdot y_j \cdot 2^{2j} = x_{N-1} \cdot y_{N-1} \cdot \left(2^{2j} + 2 \cdot 2^j \cdot \sum_{k=N-1}^{j-1} 2^k\right) \end{aligned} \qquad (5.26)$$

Ultima expresie se obţine considerând că $x_j = x_{N-1}$, pentru $j \geq N$, care implică de asemenea $Y_{j-1} = Y_{N-2} + y_{N-1} \cdot \sum_{k=N-1}^{j-1} 2^k$. Relaţii similare pot fi deduse pentru $y_j$ şi $X_{j-1}$. Prin urmare, după ciclul i, eroarea totală va fi:

$$\begin{aligned} E_i &= x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + 2 \cdot 2^j \cdot \sum_{k=N-1}^{j-1} 2^k\right) = \\ &= x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + \sum_{k=N}^{j} 2^{j+k}\right) = \\ &= x_{N-1} \cdot y_{N-1} \cdot \sum_{j=N}^{i} \left(2^{2j} + \sum_{k=j}^{i} 2^{j+k}\right) = \\ &= x_{N-1} \cdot y_{N-1} \cdot 2^{i+1} \cdot \sum_{j=N}^{i} 2^j \end{aligned} \qquad (5.27)$$

Din relația (5.27) rezultă că: $E_i \bmod 2^{i+1} = 0$, pentru orice i. Aceasta indică faptul că partea ignorată din suma parțială nu va afecta ieșirea serială a înmulțitorului (produsul) deoarece, până la ciclul i = K - 1 (la sfârșitul operației de înmulțire), vor fi generați doar cei mai puțini semnificativi K biți ai produsului.

Mai mult, dacă se înmulțesc numere fără semn (extinse cu zero), $E_i$ este identică cu zero. Prin urmare expresia $\overline{P}_i$ poate fi utilizată la proiectarea unui înmulțitor care va opera corect atât pentru operanzi reprezentați în cod complementar (cu extensie de semn) cât și pentru operanzi fără semn (extinși cu zero).

Tehnica clasică *add & shift* poate fi utilizată la implementarea algoritmului dacă se introduce:

$$Q_i = \frac{\overline{P}_i}{2^{i+1}} = \frac{1}{2}\left(Q_{i-1} + x_i \cdot Y_{i-1} + y_i \cdot X_{i-1} + 2^i \cdot x_i \cdot y_i\right) \qquad \text{pentru } i < N \qquad (5.28)$$

$$Q_i = \frac{\overline{P}_i}{2^{i+1}} = \frac{1}{2}\left(Q_{i-1} + x_{N-1} \cdot Y_{N-2} + y_{N-1} \cdot X_{N-2}\right) \qquad \text{pentru } i \geq N \qquad (5.29)$$

Ecuația (5.28) arată că fiecare produs parțial poate fi generat cu o întârziere de tip combinațional (de propagare printr-o logică combinațională) utilizând valorile memorate ale celor 2 operanzi dar numai până la nivelul bitului precedent ($X_{i-1}$ și $Y_{i-1}$) și biții de intrare curenți ($x_i$ și $y_i$). Termenul simetric $x_i \cdot y_i$ trebuie adunat *on - line* la bitul de pe rangul i. Termenul 1/2 echivalează cu o deplasare aritmetică la dreapta și partea fracțională (bitul) care rezultă din această deplasare reprezintă noul bit aferent produsului final. Înainte de deplasare, valoarea care se adună produsului parțial obținut în ciclul anterior va fi o valoare întreagă. Deci, dacă restul împărțirii (deplasării) este extras din înmulțitor la fiecare iterație, vom putea construi un sistem echivalent care va memora numai partea întreagă a rezultatului:

$$\overline{Q}_i = \left\lfloor \frac{1}{2} \cdot \left(Q_{i-1} + x_i \cdot Y_{i-1} + y_i \cdot X_{i-1} + 2^i \cdot x_i \cdot y_i\right) \right\rfloor \qquad \text{pentru } i < N \qquad (5.30)$$

O expresie similară poate fi derivată din (5.29) și pentru $i \geq N$.

Algoritmul poate fi înțeles mai ușor dacă se prezintă prin comparație cu algoritmul de bază din figura 5.9. În figura 5.10 se prezintă un exemplu de înmulțire a 2 operanzi pe 4 biți, care generează un produs pe 9 biți.

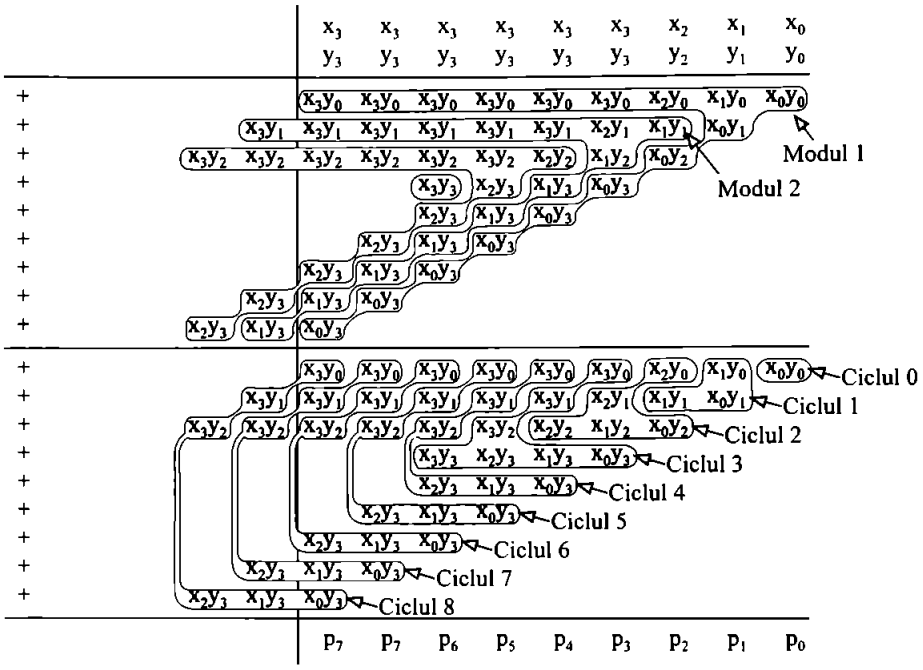



Fig. 5.10. Algoritm de înmulţire serie - serie pentru numere reprezentate în cod complementar

Termenii prezenţi în figura 5.9 şi absenţi în figura 5.10 corespund termenilor generaţi de ultima sumă din relaţia (5.27).

Un *bit - slice* aferent înmulţitorului bazat pe algoritmul (5.30) este redat în figura 5.11.

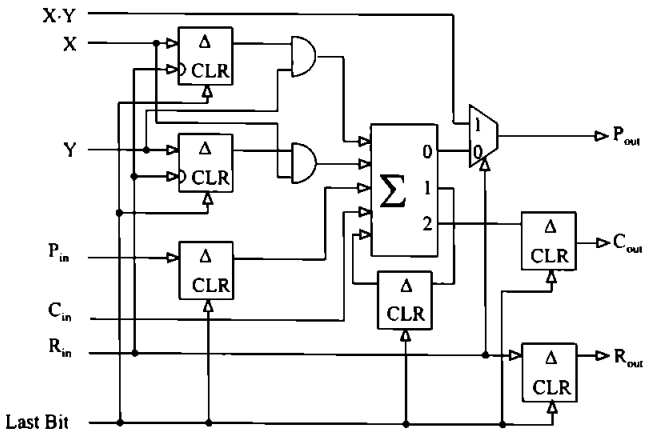



Fig. 5.11. Un *bit - slice* aferent înmulţitorului serie - serie

În acord cu structura sa, acest înmulţitor revendică operanzi de aceeaşi lungime. Dacă operanzii sunt de lungimi diferite, atunci operandul mai scurt va suporta o extensie de semn până va egala

lungimea celuilalt operand. N-1 astfel de *bit - slice* -uri vor fi necesare pentru înmulțirea a 2 numere reprezentate pe N biți. Cele N-1 *bit - slice* -uri trebuie interconectate ca în figura 5.12.

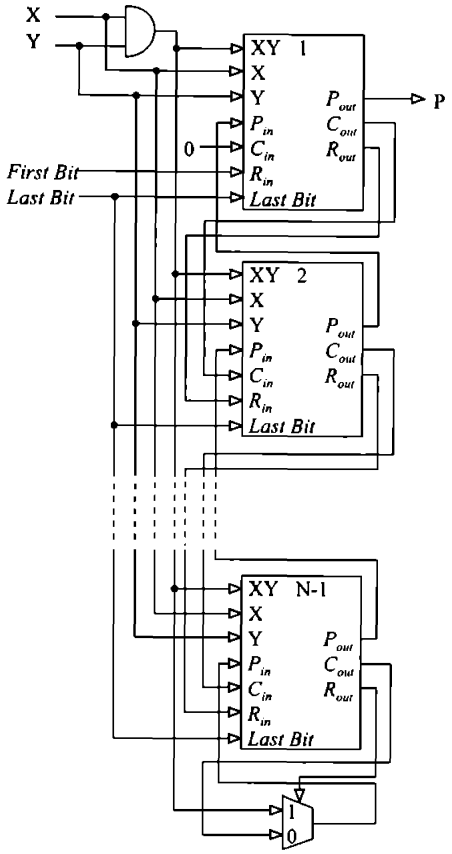



Fig. 5.12. Înmulțitor serie - serie pentru numere reprezentate în cod complementar

Prin multiplicarea porții AND din figura 5.12 și includerea ei în interiorul fiecărui modul și dacă se înlocuiește multiplexorul cu un al N-lea modul care are intrarea $P_{in}$ conectată la zero, se va obține o modularitate perfectă (cu costul unui *hardware* redundant). Ambii operanzi vor intra serial în circuit, începând cu cel mai puțin semnificativ bit; preluarea operanzilor se va realiza în N cicluri de tact. Așa cum am menționat deja, dacă se operează cu numere fără semn, acestea trebuie extinse cu zerouri pe durata ultimelor K - N cicluri (cu $K \geq 2N$ pentru a asigura un rezultat corect); dacă se operează cu numere reprezentate în complement față de 2 acestea vor suporta extensia semnului. Comanda *FirstBit* va fi setată *high* pe durata primului ciclu după care va fi menținută la zero. Similar, comanda *LastBit* va fi activată pe durata ultimului ciclu din cadrul operației de înmulțire sau, mai precis, pe durata ciclului ce precede o nouă înmulțire. Ieșirea (produsul) va fi disponibilă sub formă serială începând cu primul ciclu. Termenii încadrați în seturile din jumătatea superioară a figurii 5.10 corespund termenilor procesați (însumați) de modulul specificat; termenii încadrați în seturile din jumătatea inferioară corespund celor care vor fi adăugați sumei parțiale în același ciclu de tact.

În general doi noi termeni sunt adunaţi la suma parţială în fiecare *bit - slice*, pe fiecare ciclu de tact. Aceasta sugerează utilizarea unui element *(5, 3) counter* care va genera un *far carry* (pondere x 4) spre *bit - slice*-ul următor. Apare deci un element *delay* pe ieşirea $C_{out}$. Un alt element *delay* este prevăzut pentru memorarea semnalului *carry* local. Multiplexorul de pe ieşirea bit - slice - ului (i + 1) este utilizat pentru adunarea termenului simetric $x_i \cdot y_i$ la bitul corespunzător din cadrul sumei parţiale. Elementele *delay* cu intrări de validare (*enable*) memorează progresiv operanzii ($X_i$, $Y_i$) şi sunt utilizate pentru a genera cei 2 termeni noi care se vor aduna sumei parţiale ($x_i \cdot Y_{i-1}$ şi $y_i \cdot X_{i-1}$). În sfârşit, elementele *delay* conectate pe traseul $P_{in}$ - $P_{out}$ compun registrul care memorează şi deplasează suma parţială $\overline{Q}_i$.

Registrul de deplasare obţinut prin înlănţuirea semnalelor $R_{in}$ - $R_{out}$ aferente diverselor *slice*-uri activează succesiv modulele înmulţitorului; aceasta permite adunarea termenului simetric $x_i \cdot y_i$ şi validează memorarea biţilor aplicaţi pe intrarea *slice*-urilor. Înmulţitorul serie - serie prezentat are câteva avantaje majore (foarte importante pentru RS proiectată în această lucrare):

- grad înalt de modularitate (extrem de important pentru implementările VLSI);
- latenţă redusă între biţii aplicaţi pe intrare şi bitul corespondent obţinut la ieşire. Latenţa este cauzată doar de un circuit combinaţional care conţine pe calea critică numai un *(5, 3) counter* şi câteva porţi logice;
- permite extensia produsului obţinut pe ieşire la orice lungime; acesta rămâne corect (extensie semn) atâta timp cât se menţine extensia de semn la operanzii aplicaţi la intrare.

### 5.5. Concluzii

Simplitatea celulei sistolice, comunicaţiile strict locale (limitate la nivelul vecinilor cei mai apropiaţi), numărul extrem de redus de conexiuni şi regularitatea fac ca reţeaua sistolică prezentată să reprezinte o candidată cu mari şanse la integrarea în structuri VLSI. Plăcile acceleratoare integrate într-un calculator de tip universal (*workstation*) constituie o soluţie alternativă în munca de cercetare în domeniul conexionist. Comparativ cu un supercomputer de mare capacitate, calculatorul de tip *workstation* dotat cu un *hardware* accelerator reprezintă un bun compromis între cost şi performanţe. RS prezentată reprezintă propunerea noastră pentru acest *hardware* accelerator.

## 6. O PROPUNERE ALTERNATIVĂ: REȚEAUA SISTOLICĂ LINIARĂ

RS bidimensională se bazează pe schema de principiu prezentată în figura 6.1.

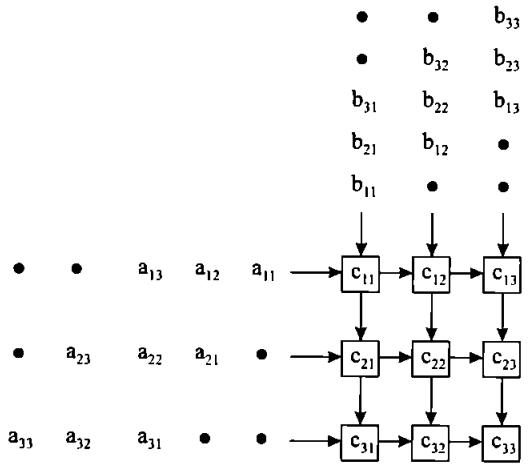



Fig. 6.1. Implementarea produsului matricial pe RS bidimensională

Pe fiecare impuls de tact, celulele sistolice din rețeaua rectangulară vor efectua operațiile prezentate în figura 6.2.

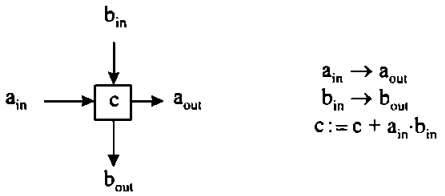



Fig. 6.2. Funcționarea celulei sistolice

Elementele matricii produs $c_{ij}$ se vor acumula ca mărimi de stare în celulele rețelei. În cazul în care matricile A şi B vor fi N x N va fi necesară o RS cu $N^2$ celule iar matricea rezultat C(N x N) se va obține în 3N - 2 perioade tact. În cazul general (A(N x M), B(M x K)) va fi necesară o RS dreptunghiulară de dimensiuni N x K, iar matricea produs C(N x K) se va obține în N + M + K - 2 perioade tact.

Când se aplică în domenii care revendică procesarea unor cantități mari de date în timp real, cum ar fi de exemplu procesarea imaginilor, rețelele neuronale revendică implementări *hardware* speciale. În capitolele precedente am investigat implementarea RN pe RS bidimensionale rectangulare. În acest capitol vom investiga implementarea RN pe rețele sistolice liniare (RSL). Vom investiga posibilitatea implementării RN *feedforward* multistrat pe RSL, ca soluție alternativă la RS bidimensională.

După cum s-a arătat în capitolul introductiv, implementările sistolice se pot realiza în două variante:

- cu grad înalt şi fin de paralelism (paralelism la nivelul sinapselor). În acest caz celula sistolică este alocată unei sinapse. RS bidimensională studiată în capitolele precedente face parte din această categorie;
- cu grad grosier de paralelism (paralelism la nivelul neuronilor). În acest caz celula sistolică va fi alocată unui neuron; va avea o structură mai complexă şi va avea o memorie locală în care va stoca toţi coeficienţii sinaptici aferenţi neuronului respectiv. Pentru a evidenţia această diferenţă de complexitate, vom denumi celula sistolică aferentă RSL element de procesare (EP).

Au fost propuse diverse variante de RSL dedicate algoritmilor conexionişti; s-au efectuat simulări şi s-au dezvoltat modele analitice pentru identificarea principiilor care trebuie să stea la baza implementărilor algoritmilor conexionişti pe RSL [NAY 95]. Pentru analiza ce urmează vom utiliza drept suport RSL HANNIBAL care implementează fazele *feedforward* (recunoaştere) şi respectiv *backward* (învăţare) aferente RN *feedforward* multistrat bazate pe algoritmul EBP. Proiectul HANNIBAL a fost dezvoltat de British Telecom în colaborare cu Loughborough University of Technology [MYE 91].

În figura 6.3 se prezintă o RSL care execută un produs matricial.

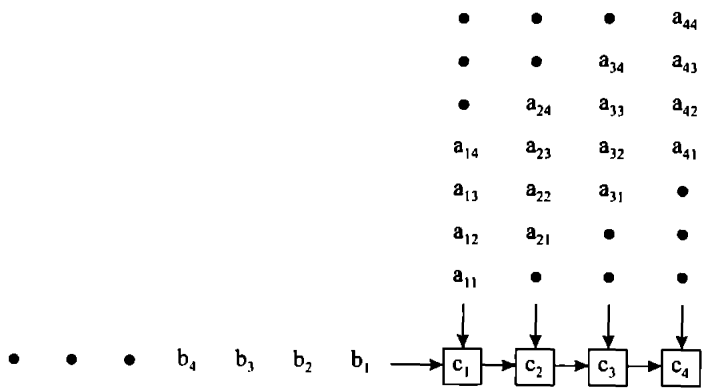



Fig. 6.3. Implementarea produsului matricial pe RSL

Fiecare CS va calcula un element al matricii (vectorului) produs pe baza operaţiilor prezentate în figura 6.4.

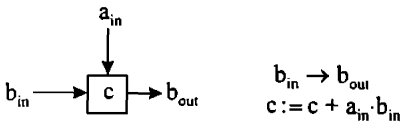



Fig. 6.4. Funcţionarea celulei sistolice aferentă RSL

În cazul general în care matricea A este de dimensiuni N x M, va fi necesară o RSL cu N celule iar matricea produs C de dimensiuni N x 1 se va obţine în N + M + 1 perioade de tact.

### 6.1. Arhitectura RSL dedicată RN feedforward multistrat

#### 6.1.1. Corespondența CS - neuron

HANNIBAL este un circuit VLSI care poate fi cascadat în scopul construiri unei RSL dedicate algoritmului EBP. Semnalele de control și de tact sunt generate centralizat de către un *controller* RSL, iar celulele sistolice adiacente pot transfera date prin intermediul unui bus de 16 biți. Multe studii care vizează estimarea performanțelor RSL dedicate algoritmilor conexioniști iau ca referință arhitectura HANNIBAL pe care o simulează pe calculatoare universale.

Modul de implementare a unei RN *feedforward* multistrat pe o astfel de RSL este redat în figura 6.5.

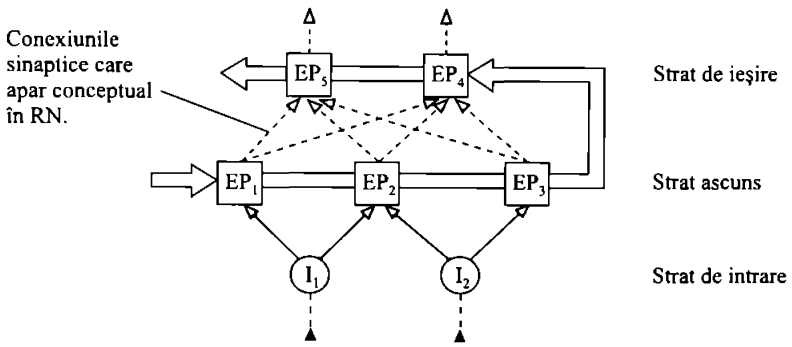
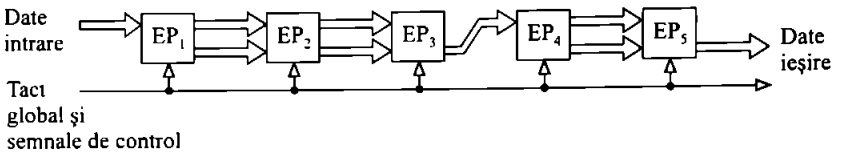



Fig. 6.5. O RN *feedforward* implementată pe RSL. RSL formează o cascadă *pipeline* în care datele de intrare se aplică direct primului element procesor din stratul ascuns

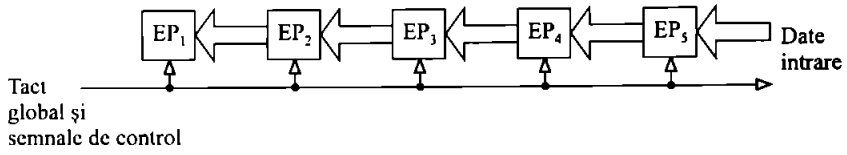
Figura 6.5 prezintă principial modul de mapare în RSL a unei RN *feedforward* cu 3 straturi (2 - 3 - 2); 2 intrări, 3 neuroni în stratul ascuns și 2 în stratul de ieșire. Conexiunile sinaptice conceptuale indică o RN complet conectată; o rețea incomplet conectată se poate obține prin simpla setare a unor *flag*-uri asociate conexiunilor sinaptice inexistente. Datele de intrare sunt aplicate direct startului ascuns. Neuronii din stratul ascuns și stratul de ieșire sunt mapați în elementele de procesare (EP) conținute în aria sistolică HANNIBAL. Un circuit HANNIBAL conține 4 EP fiecare dotat cu o memorie RAM locală care permite memorarea a 256 ponderi sinaptice reprezentate pe 16 biți. Fiecare EP conține o singură unitate de înmulțire - acumulare (MAC) pentru execuția calculelor matricial - vectoriale, comune celor mai mulți algoritmi neuronali. Unitatea MAC lucrează cu operanzi în virgulă fixă pe 32 biți. O operație MAC (înmulțirea a doi factori și acumularea produsului la rezultatul obținut în operația MAC anterioară) se execută într-o singură perioadă de tact. Funcția de activare neliniară din cadrul fiecărui EP se obține prin aproximarea funcției sigmoidale. Busul care transferă date între două EP adiacente are 16 biți. El poate fi configurat în structură *feedforward* și respectiv în structură *feedback* (fig. 6.6).

Structura *feedforward* divizează cascada *pipeline* de elemente EP în două busuri de date de 8 biți. Busul superior (cascada *pipeline* superioară) este utilizat (pentru fiecare grup de procesoare EP aferente unui anumit strat neuronal) pentru transmiterea datelor de intrare, în timp ce busul inferior extrage rezultatele obținute de acest strat neuronal și le transmite primului EP din următorul strat. Granița dintre două straturi neuronale (între EP3 și EP4 în exemplul nostru) este configurată la inițializarea rețelei. Avantajul acestei structuri *pipeline* liniare constă în faptul că datele de ieșire aferente fiecărui strat neuronal pot fi transmise următorului strat de îndată ce devin disponibile. Un

nou vector de intrare poate fi transmis unui start neuronal fără a aştepta ca datele de ieşire să fie emise din cascada *pipeline* aferentă acestui strat.




a)




b)

Fig. 6.6. Interconectarea *pipeline* a procesoarelor EP
a) Structura *feedforward* dispune de 2 busuri de 8 biţi pentru transmiterea simultană a datelor de intrare şi respectiv ieşire
b) Structura *feedback* utilizează un singur bus de 16 biţi pentru propagarea înapoi a erorilor în timpul procesului de învăţare

Structura *feedback* prezentată în fig. 6.6. b) este utilizată numai în timpul antrenării RN (învăţării) şi reconfigurează RSL sub forma unui bus *pipeline* de 16 biţi inversând direcţia datelor. Această structură este utilizată pentru propagarea erorii înapoi prin lanţul de EP aferente fiecărui strat neuronal spre EP-urile corespunzătoare din stratul neuronal anterior. Există studii care arată că reprezentarea erorilor pe 16 biţi asigură acurateţea necesară procesului de ajustare a ponderilor sinaptice şi este suficientă pentru a asigura convergenţa algoritmului de învăţare [VIN 92].

### 6.1.2. Implementarea algoritmului EBP pe RSL

Implementarea algoritmului EBP pe RSL HANNIBAL se realizează cu doar câteva instrucţiuni sau moduri. *Controller*-ul fiecărui EP generează informaţia de iniţializare cum ar fi poziţia în cadrul stratului neuronal, numărul stratului neuronal etc., ceea ce permite execuţia aproape autonomă a algoritmului. *Controller*-ul centralizat al RSL trebuie să controleze operaţiile mai puţin frecvente şi să sincronizeze aplicarea datelor de intrare şi memorarea rezultatelor.

#### A. Faza feedforward

Faza *feedforward* utilizează *pipeline*-ul de date din fig. 6.6. a) pentru transferul datelor de intrare şi a valorilor de activare aferente EP-urilor din RSL. Valoarea de activare, $A_j^l$, aferentă unui EP (neuron) din stratul ascuns sau de ieşire este dată de ecuaţia fazei *feedforward* caracteristică RN multistrat:

$$A_j^l = \sigma\left(\theta_k^l + \sum_{k=0}^{K-1}\left(w_{jk}^l \cdot A_k^{l-1}\right)\right) \qquad (6.1)$$

unde:

l - indexul stratului neuronal (l = 0 strat intrare, l = 1 strat ascuns etc.)
j - indexul EP din stratul curent
k - indexul EP din stratul anterior stratului curent
K - numărul de EP-uri (neuroni) conținute în stratul anterior celui curent
w - ponderea sinaptică pentru EP-ul indicat de indecși
θ - valoarea de prag aferentă EP-ului indicat de indecși
σ(·) - funcția sigmoidală

Structura *feedforward pipeline* dual (fig. 6.6. a)) permite transferul spre stratul de ieșire a valorilor de activare generate de către stratul ascuns în paralel cu aplicarea unui nou vector de intrare în stratul ascuns. În consecință, mai mulți vectori de intrare parțial procesați pot fi prezenți simultan în RSL, deși fiecare strat poate procesa doar elementele unui singur vector la un moment dat.

*Controller*-ul ariei RSL selectează modul ACTIVARE pentru a iniția secvența de stări RSL caracteristică fazei *feedforward*. Diagrama spațiu - timp pentru rețeaua 2 - 3 - 2 din fig. 6.5 care operează în modul ACTIVARE este redată în fig. 6.7.

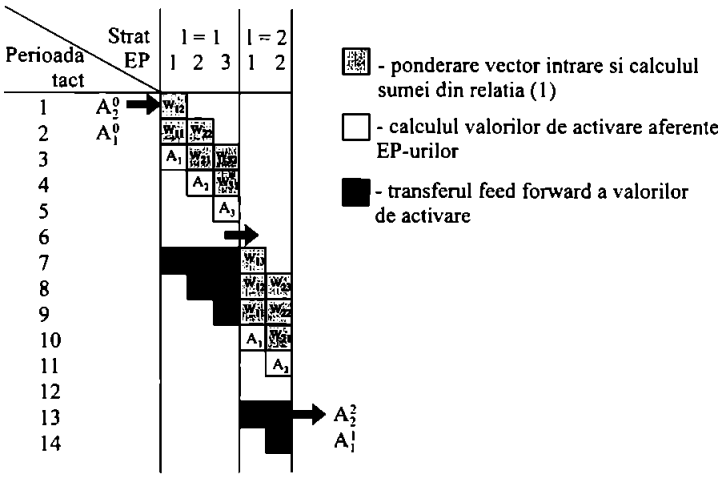



Fig. 6.7. Diagrama spațiu - timp corespunzătoare modului ACTIVARE aferentă RN 2 - 3 - 2. $A_1$ - $A_3$ sunt calculate în acord cu relația (6.1), înainte de a fi transferate stratului de ieșire (perioada de tact 7) pe busul *pipeline* inferior din fig. 6.6. a). Același proces se repetă și pentru stratul de ieșire și rezultatul (vectorul de ieșire) se generează în perioada de tact nr. 13.

Fig. 6.7 indică procesele care au loc în fiecare EP din RSL, în fiecare ciclu de tact, în cazul aplicării unui vector la intrarea RN (vectorul $\mathbf{A}^0 = A_1^0, A_2^0$). Spațiile libere indică cicluri WAIT. $EP_1$ recepționează primul element al vectorului de intrare ($A_2^0$) în primul ciclu, execută prima operație MAC și transferă data de la intrare prin cascada *pipeline* spre $EP_2$. Aceste operații se repetă pentru toate EP-urile din stratul neuronal respectiv și pentru toate elementele vectorului de intrare. Așa cum indică fig. 6.6. a), doar busul de date superior este utilizat pentru transferul datelor de intrare prin cascada *pipeline* ce formează stratul neuronal respectiv. În momentul în care $EP_1$ a recepționat

toate elementele vectorului de intrare, va calcula valoarea de activare $A_i^1$ şi apoi va intra în WAIT. Când toate EP-urile stratului respectiv au definitivat aceste operaţii, într-un ciclu suplimentar, toate valorile de activare sunt transferate busului inferior aferent cascadei *pipe*. Acelaşi proces repetă în stratul de ieşire care va genera vectorul de ieşire din RN, $\mathbf{A}^2$, asociat vectorului de intrare, $\mathbf{A}^0$.

Fiecare EP din RSL va repeta operaţiile aferente fazei *feedforward* procesând noi vectori de intrare atâta timp cât *controller*-ul centralizat menţine modul ACTIVARE activ. Frecvenţa cu care noii vectori de intrare pot fi aplicaţi la intrarea RSL este determinată de structura RN. Aceasta va determina în consecinţă caracteristicile de productivitate, latenţă şi eficienţă aferente RSL. Deci, structura RN este foarte importantă pentru performanţele RSL în faza *feedforward*.

**B. Învăţarea**

Procesul de învăţare sau antrenare a RN constă în ajustarea ponderilor sinaptice în ideea reducerii erori globale calculată la ieşirea RN. În cazul algoritmului EBP aceasta se realizează prin regula de învăţare delta generalizată. Diferenţa dintre valoarea dorită la ieşire şi respectiv valoarea reală obţinută la ieşire generează eroarea aferentă stratului de ieşire care este propagată înapoi prin reţea spre straturile ascunse (stratul ascuns). Ponderile sinaptice sunt ajustate pentru a reduce gradientul erorii spre un minim global. Acest proces trebuie aplicat pentru fiecare pereche de vectori intrare - ieşire pe care RN trebuie s-o înveţe. Există 3 stadii în procesul învăţării:

1. propagarea înainte (*forward*) a valorilor de activare
2. propagarea înapoi (*backpropagation*) a erorilor
3. ajustarea ponderilor sinaptice şi a valorilor de prag.

Primul stadiu reprezintă tocmai faza *feedforward* care se obţine în cadrul RSL prin aplicarea modului ACTIVARE. Va conţine aceeaşi secvenţă de operaţii prezentată deja în fig. 6.7, până la nivelul ciclului de tact nr. 11 moment în care acest prim stadiu se încheie. Valorile vectorului de ieşire nu sunt transferate la ieşirile RSL în timpul învăţării (aşa cum se întâmplă în cazul fazei *feedforward*), deoarece aceste valori sunt utilizate doar intern la calculul erorilor.

Următorul stadiu al procesului de învăţare poate să înceapă imediat după ciclul nr. 11 din fig. 6.7. Acest stadiu implică calculul erorii dintre vectorul de ieşire aşteptat şi cel real. EP-urile stratului de ieşire calculează erorile locale $\delta_j^2$ pe baza ecuaţiei:

$$\delta_j^l = A_j^l(1 - A_j^l)(D_j - A_j^l) \qquad (6.2)$$

unde $D_j$ reprezintă valoarea dorită la ieşirea neuronului j din stratul de ieşire, iar $A_j$ reprezintă valoarea reală obţinută pe ieşirea aceluiaşi neuron. Ecuaţia (6.2) este valabilă în cazul funcţiei de activare unipolară (1.3).

EP-urile din stratul ascuns generează erorile locale pe baza ecuaţiei:

$$\delta_j^l = A_j^l(1 - A_j^l)\sum_{i=0}^{I-1}(\delta_j^{l+1} \cdot w_{ij}^{l+1}) \qquad (6.3)$$

unde: i - indexul EP-urilor din stratul următor celui curent
I - numărul de EP-uri din stratul următor celui curent

Relaţia (6.3) reprezintă în fapt propagarea erorii înapoi într-o reţea *feedforward* de forma celei din fig. 6.8:

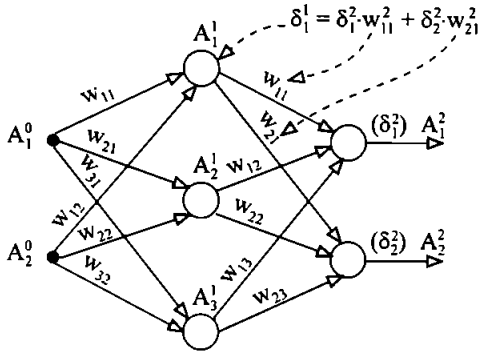



Fig. 6.8. Propagarea erorii înapoi în RN *feedforward* 2 - 3 - 2

Valorile erorilor de la nivelul stratului de ieşire (valori pe 16 biţi) sunt propagate înapoi prin cascada RSL utilizând structura *pipeline* prezentată în fig. 6.6. b). Ecuaţiile (6.2) şi (6.3) sunt implementate în arhitectura HANNIBAL în modul CALCUL. Diagrama spaţiu - timp aferentă acestui mod pentru o RN *feedforward* 2 - 3 - 2 este prezentată în fig. 6.9:

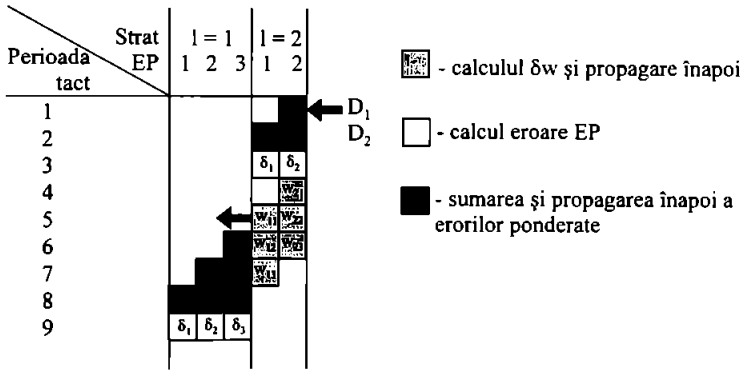



Fig. 6.9. Diagrama spaţiu - timp aferentă modului CALCUL pentru RN 2 - 3 - 2. Vectorul de ieşire aşteptat D (8 biţi) este încărcat în RSL în sens invers (fig. 6.6. b)) şi se calculează erorile aferente stratului de ieşire în ciclul 3 (relaţia (6.2)). Termenii obţinuţi prin ponderarea erorilor de ieşire (16 biţi) sunt transferaţi înapoi spre stratul ascuns unde se calculează erorile aferente acestui strat (relaţia (6.3))

Vectorul dorit la ieşire, $\mathbf{D} = (D_1, D_2)$ conţine 2 componente reprezentate pe 8 biţi. Acestea sunt încărcate succesiv în EP-urile corespunzătoare din stratul de ieşire. Vectorii de eroare, $\delta_1^2$ şi $\delta_2^2$, sunt apoi calculaţi pe baza relaţiei (6.2). Derivata întâi a funcţiei de activare sigmoidale unipolare, $A_j^1(1-A_j^1)$, este precalculată iniţial de către fiecare EP în momentul în care se determină $A_j^1$. Pentru a genera eroarea aferentă procesorului $EP_1$ din stratul ascuns, suma erorilor ponderate, $\delta_1^2 \cdot w_{11}^2 + \delta_2^2 \cdot w_{21}^2$, trebuie propagată înapoi prin EP-urile 2 şi 3 aferente acestui strat

(fig. 6.8 şi fig. 6.6. b)). După această propagare ecuaţia (6.3) se poate aplica pentru calculul erorii $\delta_1^1$. Acelaşi proces se repetă întocmai pentru erorile EP-urilor 2 şi 3 din stratul ascuns.

Exemplul dat prezintă o RN cu un singur strat ascuns. Pentru RN cu mai multe straturi ascunse ciclurile pentru generarea erorilor pe baza ecuaţiei (6.3) se vor repeta pentru fiecare strat ascuns.

Stadiul final aferent procesului de învăţare implică ajustarea ponderilor sinaptice şi a valorilor de prag în acord cu eroarea memorată acum în fiecare EP. Actualizarea ponderilor sinaptice pentru fiecare EP se face pe baza relaţiei:

$$w_{jk}^l \leftarrow w_{jk}^l + \alpha \cdot \delta_j^l \cdot A_k^{l-1} \tag{6.4}$$

Valorile de prag vor fi actualizate pe baza relaţiei:

$$\theta_j^l \leftarrow \theta_j^l + \alpha \cdot \delta_j^l \tag{6.5}$$

unde $\alpha$ reprezintă constanta de învăţare.

În arhitectura HANNIBAL, ajustarea ponderilor sinaptice şi a valorilor de prag se realizează în modul ACTUALIZARE, utilizând structura *pipeline* din fig. 6.6. a). Fig. 6.10 reprezintă diagrama spaţiu - timp aferentă acestui mod pentru o RN 2 - 3 - 2.

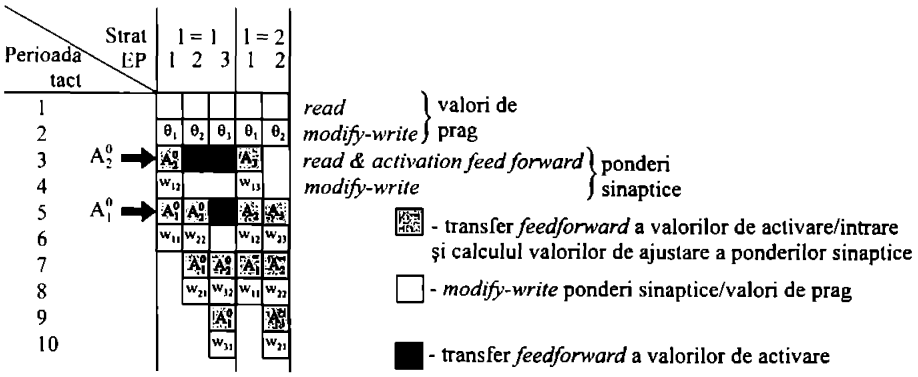



Fig. 6.10. Diagrama spaţiu - timp aferentă modului ACTUALIZARE pentru RN 2 - 3 - 2. Fiecare strat execută în paralel două cicluri *read - modify - write* asupra ponderilor sinaptice. Datele de intrare pentru fiecare strat (vectorul de intrare în RN sau valorile de activare aferente stratului anterior) sunt transferate de-a lungul structurii *pipeline* din două în două cicluri. Ecuaţiile (6.4) şi (6.5) descriu procesul de actualizare a ponderilor sinaptice şi respectiv a valorilor de prag

Procesul de actualizare este în fapt o operaţie *read - modify - write* cu durata a două cicluri care se aplică mai întâi valorilor de prag. Relaţia (6.4) arată că ajustarea ponderilor sinaptice respectă următoarea regulă: pentru $EP_j$, valoarea cu care se modifică ponderea sinaptică aferentă sinapsei ce conectează $EP_j$ la $EP_k$ din stratul anterior, este o funcţie care depinde de valoarea de activare aferentă procesorului $EP_k$ şi de eroarea din $EP_j$. În consecinţă, vectorul de intrare trebuie aplicat din

nou pe intrarea RSL şi valorile de activare aferente stratului ascuns transferate stratului de ieşire. Structura *pipeline* a RSL din fig. 6.6 a) permite execuţia în paralel a acestor transferuri. Frecvenţa cu care se aplică datele pe intrarea EP din stratul ascuns şi respectiv de ieşire trebuie înjumătăţită datorită duratei extinse (2 cicluri) a operaţiilor *read - modify - write*. În primul ciclu se citeşte valoarea ponderii sinaptice din memorie şi se calculează valoarea cu care trebuie ajustată. În ciclul 2 ponderea sinaptică este ajustată şi rescrisă în RAM.

Învăţarea este, prin urmare, un proces secvenţial care permite numai unei perechi de antrenament (vectori intrare - ieşire) să fie procesată la un moment dat. Consecinţa acestui fapt va fi că latenţa şi productivitatea procesului de învăţare vor fi mai puţin afectate de structura RN decât faza *feedforward*. Totuşi, stratul care va necesita cele mai multe cicluri pentru actualizarea ponderilor sinaptice aferente, va determina durata globală a procesului de învăţare. Prin urmare, optimizând structura RN se poate obţine o creştere a performanţelor procesului de învăţare bazat pe algoritmul EBP.

### 6.2. Analiza performanţelor

În cele ce urmează ne vom concentra asupra performanţelor pe care arhitectura HANNIBAL le obţine în procesul emulării RN *feedforward* multistrat cu algoritmul de învăţare EBP. Pentru caracterizarea performanţelor *hardware*-ului vom utiliza 3 metrici:

1. productivitatea (modele procesate per 1000 cicluri de tact)
2. latenţa (numărul de cicluri necesare pentru procesarea unui model)
3. eficienţa ariei RSL

Prin model înţelegem un vector de intrare în cazul fazei *feedforward* şi respectiv o pereche de antrenament (o pereche de vectori intrare - ieşire) în cazul procesului de învăţare. Eficienţa ariei reprezintă o măsură a gradului de utilizare a procesoarelor EP din cascada *pipeline* pe durata de timp necesară procesării unui model. Pentru a cuantifica eficienţa, procesele care apar pe durata unui ciclu de tact în fiecare EP au fost evaluate şi asignate uneia dintre următoarele 4 categorii:

a) calcul (*calc*) - procesări în ALU, citire / scriere memorie, transfer între registre
b) comunicaţie (*com*) - transferurile de date între diversele EP care vor afecta ieşirea reţelei
c) calcul şi comunicaţie (*calc & com*) - procesele a) şi b) executate simultan. Un procentaj ridicat de astfel de operaţii caracterizează o arie sistolică eficientă.
d) wait (*wait*) - un procesor (EP) în stare de aşteptare sau o operaţie care nu are efect asupra ieşirilor reţelei.

Evaluările de performanţă au fost făcute luând în considerare un singur parametru: dimensiunea stratului ascuns în cazul RN cu un singur strat ascuns (fig. 6.5). Dimensiunea stratului ascuns este considerată mult mai flexibilă comparativ cu cea a stratului de intrare şi respectiv ieşire. Dimensiunea acestor straturi este adesea fixată şi restricţionată de formatul datelor (vectorilor de intrare - ieşire) utilizate în aplicaţia RN. A fost de asemenea arătat [KUN 88] că schimbarea dimensiunii stratului ascuns nu va conduce în mod obligatoriu la deteriorarea performanţelor algoritmice ale aplicaţiei. În consecinţă, un număr de EP redundante pot fi adăugate în stratul ascuns dacă acest lucru se dovedeşte benefic pentru performanţele RSL. O dimensiune tipică de RN utilizată în recunoaşterea imaginilor a fost utilizată pentru evaluarea performanţelor. Numărul de EP din stratul ascuns (h) a fost variat între 50 şi 1000, iar dimensiunile stratului de intrare şi respectiv ieşire au fost fixate (I = 512, O = 10).

### 6.2.1. Faza feedforward

O RN cu vectori de intrare I-dimensionali, cu h neuroni în stratul ascuns şi cu O neuroni în stratul de ieşire va executa un ciclu complet în modul ACTIVARE, ilustrat în fig. 6.7, în X cicluri de tact. Pentru procesarea a N vectori de intrare secvenţa din fig. 6.7 trebuie repetată de N ori. Totuşi, datorită structurii *pipeline* duale a RSL caracteristică fazei *feedforward* (fig. 6.6. a)), cei N vectori vor fi procesaţi în mai puţin de X · N cicluri de tact. Fig. 6.11 prezintă o formă simplificată a diagramei spaţiu - timp aferentă modului ACTIVARE. Această formă de prezentare a diagramei spaţiu - timp simplifică analiza performanţelor RSL, reducând-o la calculul unor arii geometrice.

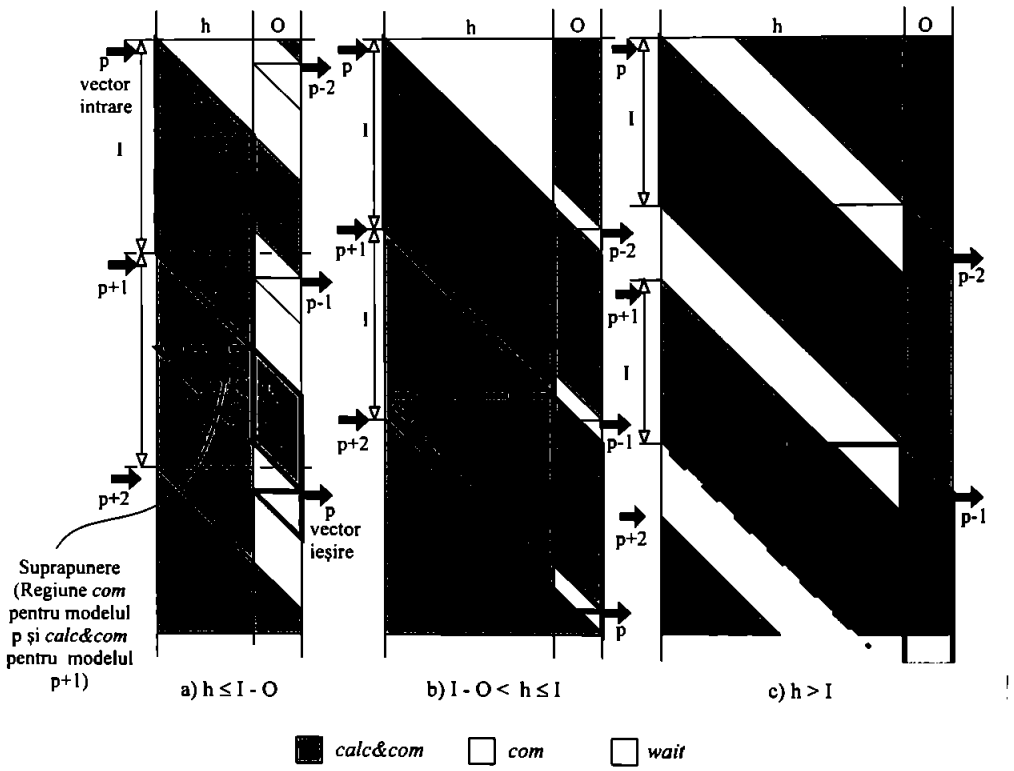



Fig. 6.11. Diagrama spaţiu - timp simplificată aferentă fazei *feedforward*. Regiunile marcate cu linie îngroşată indică operaţii din RSL asociate vectorului de intrare p. În cazul a) stratul ascuns operează cu eficienţă maximă şi I determină productivitatea RN (modelele de intrare sunt aplicate la intervale de I cicluri). Pe măsură ce h se apropie de valoarea I, aşa cum indică cazul b), zonele *wait* şi *com* tind spre zero. În cazul c) stratul de ieşire operează cu eficienţă maximă şi productivitatea devine invers proporţională cu h

Anumite procese cum ar fi calculul valorilor de activare prezintă doar o creştere liniară în raport cu h şi au fost considerate neglijabile. Fig. 6.11 arată cum mai mulţi vectori de intrare pot fi procesaţi de către RSL simultan (structură *pipeline*). Regiunile marcate cu linie îngroşată evidenţiază operaţiile RSL dedicate calculului vectorului de ieşire asociat vectorului de intrare p. Se observă că

tipul, frecvenţa şi comutarea operaţiilor depind nu numai de h ci şi de mărimea sa în raport cu I şi O. Pentru RN cu I > O, apar trei cazuri în funcţie de valoarea parametrului h:

a) $0 < h \leq I - O$ (fig. 6.11. a)). Stratul ascuns procesează vectorii de intrare cu eficienţă maximă. Prin urmare productivitatea fiecărui strat şi productivitatea globală a RSL este determinată de acest strat. Stratul de ieşire poate executa toate operaţiile MAC ce-i revin şi poate descărca vectorul de ieşire aferent modelului de intrare p-1 înainte ca valorile de activare aferente modelului p (următor) să fie generate de stratul ascuns şi evident transmise. În consecinţă, un număr de cicluri *wait* vor apărea în stratul de ieşire.
b) $I - O < h \leq I$ (fig. 6.11. b)). Dimensiunea stratului ascuns determină suprapunerea descărcării vectorului de ieşire aferent modelului p-1 cu calculul vectorului de ieşire aferent modelului p. Pentru că cele două operaţii utilizează părţi diferite din *pipeline*-ul RSL, eficienţa procesoarelor EP din startul ascuns nu suferă şi productivitatea globală a RSL rămâne nemodificată.
c) $h > I$ (fig. 6.11. c)). Va exista cel puţin un EP în stratul ascuns care va revendica mai multe cicluri de tact pentru transferul spre stratul de ieşire a valorilor de activare aferente modelului p-1 decât ar fi obligat să revendice pentru execuţia operaţiilor MAC aferente modelului p. Prin urmare, stratul de ieşire poate recepţiona în mod continuu valorile de activare transmise de stratul ascuns în timp ce simultan descarcă vectorul de ieşire curent. Deci, stratul de ieşire operează cu eficienţă maximă, în timp ce stratul ascuns va trebui să introducă periodic un număr de stări *wait*.

Cele două straturi ale RN funcţionează în cadrul RSL ca două module *pipeline* cascadate. În consecinţă, latenţa se obţine simplu cumulând întârzierile de procesare cauzate de cele două straturi.

$$\text{Latenţa} = \text{Întârziere strat ascuns} + \text{Întârziere strat ieşire} = (I + h) + (h + O) = I + 2h + O \qquad (6.6)$$

Productivitatea este determinată de dimensiunea individuală a straturilor neuronale şi nu de dimensiunea RN în ansamblu. Productivitatea este reprezentată de numărul de modele procesate în unitatea de timp (perioadă de tact, 1000 perioade de tact etc.). Figurile 6.11. a) şi b) indică:

$$\text{Productivitatea} = \frac{1}{I} \qquad \text{pentru } h \leq I \qquad (6.7.\text{ a})$$

iar fig. 6.11. c) indică:

$$\text{Productivitatea} = \frac{1}{h} \qquad \text{pentru } h > I \qquad (6.7.\text{ b})$$

Relaţiile (6.7) indică faptul că productivitatea este determinată de cel mai mare strat neuronal din componenţa RN, fiind invers proporţională cu dimensiunea acestuia. Dacă dimensiunea acestui strat este fixă (impusă de către aplicaţie) atunci un grad de flexibilitate va exista pentru celelalte straturi fără a afecta productivitatea. În analiza noastră de exemplu (I = 512 = fix), productivitatea maximă se va obţine atâta timp cât este îndeplinită condiţia $h \leq I$, indiferent de dimensiunea O. Pentru $h > I$, productivitatea descreşte invers proporţional cu h.

Fig. 6.12 prezintă rezultatele obţinute în baza acestor simulări pentru productivitatea RSL şi latenţă.

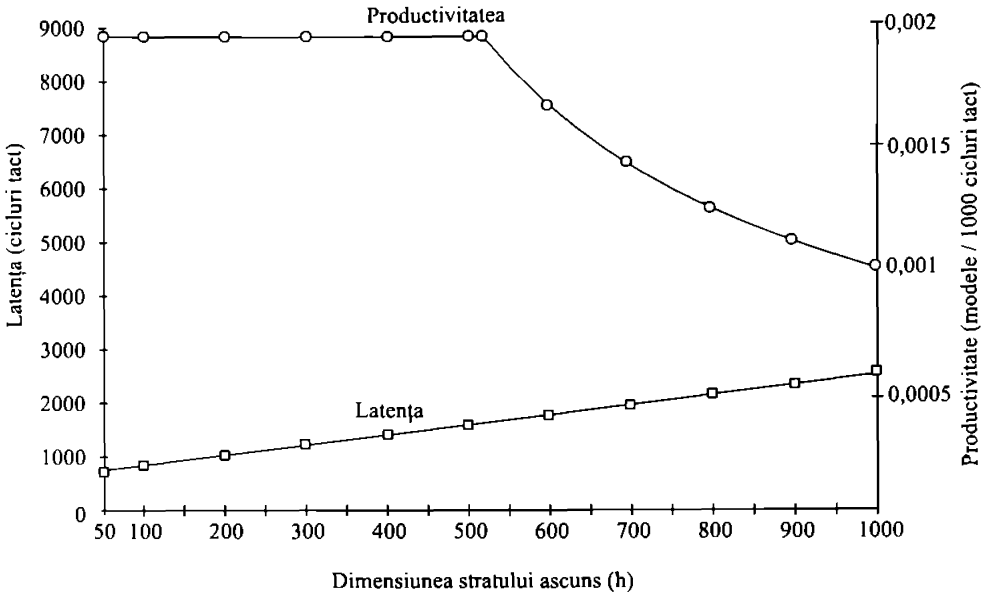



Fig. 6.12. Latenţa şi productivitatea aferente fazei *feedforward* pentru o RN 512 - h - 10. Productivitatea este invers proporţională cu dimensiunea celui mai mare strat din structura RN. Latenţa este dată de suma numărului de intrări şi ieşiri aferente tuturor straturilor din structura RN (straturile de intrare şi respectiv ieşire participă la sumă cu numărul de EP ce le conţin iar straturile ascunse cu dublul numărului de EP ce apar în componenţa lor)

În regiunea h < I productivitatea este constantă. Latenţa este independentă de productivitate pentru orice valoare a lui h.

Pentru a evalua eficienţa RSL trebuie examinate operaţiile executate de fiecare EP pe durata unui ciclu *feedforward*. Un astfel de ciclu începe cu aplicarea primei componente a unui vector de intrare şi se încheie după obţinerea ultimei componente din vectorul de ieşire asociat. Pe baza ariilor delimitate în fig. 6.11 corelate cu diagrama spaţiu - timp din fig. 6.7 se pot deduce următoarele relaţii cantitative pentru cele 4 categorii de procese executate de către procesoarele EP pe durata unui ciclu *feedforward*:

$$calc = h + O \qquad \text{pentru } \forall\, h \qquad (6.8)$$

Cea mai mare pondere în cadrul unui ciclu *feedforward* o au procesele *calc & com*. Cantitativ aceste procese prezintă o dependenţă liniară în raport cu h şi relaţia poate fi dedusă analitic pe baza ariilor delimitate în fig. 6.11:

$$calc\ \&\ com = h\cdot(I + O) \qquad \text{pentru } \forall\, h \qquad (6.9)$$

Conform figurii 6.11 ultimele două procese prezintă evaluări cantitative dependente de h. Stratul mic de ieşire (O) utilizat în această analiză a limitat influenţa regiunii $I - O < h \leq I$ în raport cu răspunsul global al RSL.

Ecuațiile aferente fiecărei regiuni sunt:

$$com = \frac{O^2}{2} \qquad \text{pentru } 0 < h \le I - O \qquad (6.10)$$

$$com = \frac{(I-h)^2}{2} \qquad \text{pentru } I - O < h \le I \qquad (6.11)$$

$$com = \frac{(h-I)^2}{2} \qquad \text{pentru } h > I \qquad (6.12)$$

$$wait = O(I-h) - \frac{O^2}{2} \qquad \text{pentru } 0 < h \le I - O \qquad (6.13)$$

$$wait = O(I-h) - \frac{(I-h)^2}{2} = \frac{I-h}{2}(2O - I + h) \qquad \text{pentru } I - O < h \le I \qquad (6.14)$$

$$wait = \frac{h^2}{2} - \frac{I^2}{2} \qquad \text{pentru } h > I \qquad (6.15)$$

Fig. 6.13 prezintă rezultatele obținute în baza acestei simulări.

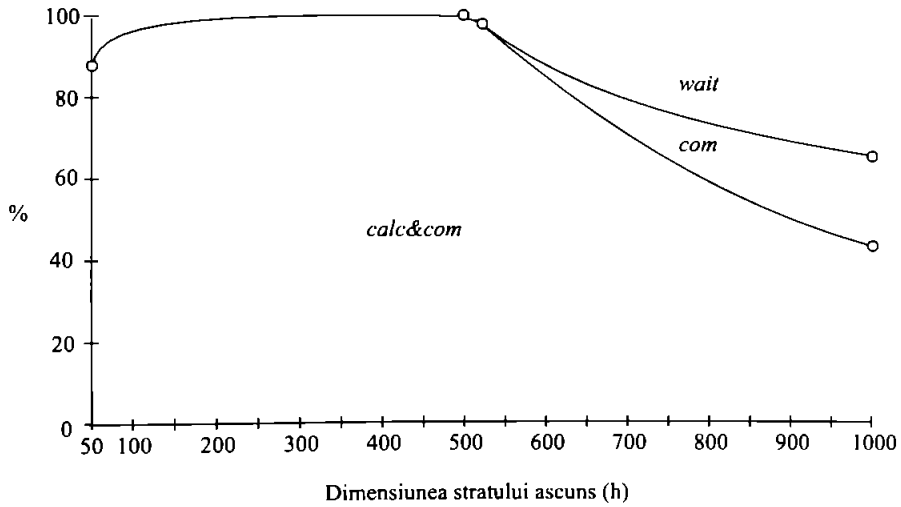



Fig. 6.13. Structura proceselor din RSL pentru RN 512 - h - 10 pe durata fazei *feedforward*. Eficiența RSL se apropie de 100 % când h = I = 512. Peste această valoare performanțele se degradează. Un alt punct de discontinuitate apare la valoarea h = I - O = 502.

Zona *com* este neglijabilă şi constantă pentru h ≤ I - O şi tinde la zero pe măsură ce h se apropie de valoarea I. Un strat de ieşire de dimensiune mai mare ar provoca creşterea influenţei operaţiilor *com* în aceste zone. Pentru h > I o ecuaţie pătratică descrie creşterea ponderii proceselor *com*. Procesele *wait* sunt dependente de h în toate regiunile. Regresia liniară din zona h ≤ I - O, (6.13), şi dependenţa pătratică în raport cu h în zona I - O < h ≤ I, (6.14), sugerează că RSL lucrează cu eficienţă maximă când h = I. Aceasta se poate observa şi din fig. 6.13. Pentru h > I, procesele *wait* cresc cu $h^2$ şi atât procesele *wait* cât şi procesele *com* sunt independente de O. În consecinţă, dimensiunea stratului de ieşire este flexibilă în această regiune şi ar putea constitui un parametru ajustabil pentru creşterea eficienţei RSL.

### 6.2.2. Învăţarea

Diagrama spaţiu - timp din fig. 6.14 reprezintă o formă simplificată a celor prezentate deja în fig. 6.7, 6.9 şi 6.10 pentru modurile ACTIVARE, CALCUL şi respectiv ACTUALIZARE.

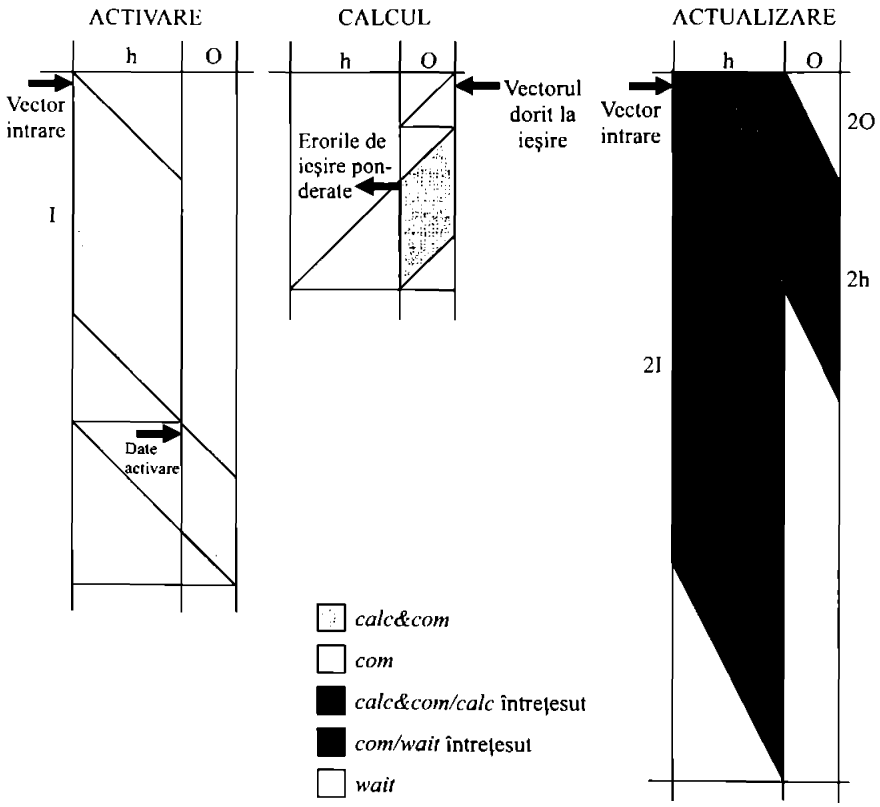



Fig. 6.14. Diagrama spaţiu - timp simplificată aferentă celor 3 stadii (moduri) ce compun învăţarea RN. Modul ACTIVARE nu mai conţine zona *com* aferentă stratului de ieşire ca în cazul fazei *feedforward* (la învăţare vectorul de ieşire nu mai trebuie comunicat în exterior; el va fi utilizat în modul calcul la evaluarea erorilor). Modul ACTUALIZARE conţine operaţii întreţesute datorită ciclurilor *read - modify - write* care se execută pe două perioade de tact succesive.

Se poate observa din fig. 6.14 că există un procentaj semnificativ de cicluri *wait* şi respectiv *com* în cadrul fazei de învăţare. Acestea vor afecta serios eficienţa RSL şi latenţa.

$$\text{Latenţa} = (I + 2h + O) + (2O + h) + (2 \max_{l=0}^{2}(L_l + L_{l+1}) + 2h + 2O) \qquad (6.16)$$

unde $L_l$ reprezintă dimensiunea stratului l.

Relaţia (6.16) indică faptul că dimensiunile relative ale straturilor RN joacă un rol pentru latenţă în cazul învăţării aşa cum s-a întâmplat şi în faza *feedforward*. Din (6.16) se poate deduce expresia latenţei pentru RN cu un singur strat ascuns:

$$\text{Latenţa} = 3I + 7h + 5O \qquad \text{pentru } O \le I \qquad (6.17)$$

$$\text{Latenţa} = I + 7h + 7O \qquad \text{pentru } O > I \qquad (6.18)$$

În consecinţă, dacă două RN conţin acelaşi număr de neuroni, cea care îndeplineşte condiţia $O \le I$ va prezenta cea mai mică latenţă. Pentru reţeaua analizată de noi (512 - h - 10) latenţa este dată de relaţia (6.17). Pentru că perechile de antrenament sunt procesate secvenţial putem afirma că în acest caz este valabilă relaţia care exprimă legătura dintre productivitate şi latenţă în cazul proceselor secvenţiale:

$$\text{Productivitate} = \frac{1}{\text{Latenta}} \qquad (6.19)$$

Fig. 6.15 prezintă rezultatele obţinute pentru latenţă şi productivitate în urma acestor simulări ale procesului de învăţare.

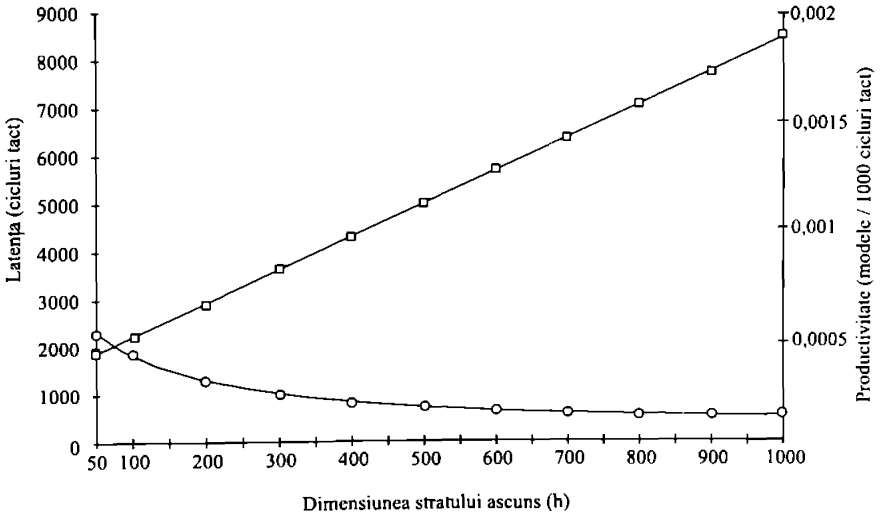



Fig. 6.15. Latenţa şi productivitatea aferente fazei de învăţare pentru o RN 512 - h - 10. Modelele sunt procesate secvenţial şi ciclul de învăţare aferent modelului curent trebuie încheiat înainte de startarea ciclului aferent modelului următor. În această situaţie productivitatea este invers proporţională cu latenţa.

Pentru a evalua eficienţa RSL trebuie examinate operaţiile executate de fiecare EP pe durata unui ciclu de învăţare. Pe baza ariilor delimitate în fig. 6.14 şi a diagramelor spaţiu - timp din figurile 6.7, 6.9 şi 6.10 se deduc următoarele relaţii:

$$\begin{aligned} calc &= \text{ACTIVARE}_{calc} + \text{CALCUL}_{calc} + \text{ACTUALIZARE}_{calc} = \\ &= (h + O) + (h + O) + (Ih + hO + 2h + 2O) = \\ &= h(4 + I) + O(4 + h) \end{aligned} \tag{6.20}$$

Procentul de procese *calc* care apar în modurile ACTIVARE şi CALCUL este neglijabil şi nici nu s-au mai evidenţiat aceste procese în fig. 6.14, dar au fost incluse în relaţia (6.20) pentru o evaluare cât mai riguroasă.

$$calc\ \&\ com = (Ih + hO) + (hO) + (Ih + hO) = h(2I + 3O) \tag{6.21}$$

Procentul de procese *com* este mult mai mare în cazul învăţării decât în cazul fazei *feedforward*. Aceasta se datorează propagării înainte a valorilor de activare urmată de propagarea înapoi a erorilor. Propagarea înapoi are un efect mult mai amplu.

$$com = \left(\frac{h^2}{2}\right) + \left(\frac{O^2}{2} + \frac{h^2}{2}\right) + \left(\frac{h^2}{2}\right) = \frac{3h^2}{2} + \frac{O^2}{2} \tag{6.22}$$

După cum indică ecuaţia (6.22) procentul de procese *com* este independent de I.
Ca şi în cazul latenţei, procentul de cicluri *wait* este determinat de dimensiunile relative ale straturilor adiacente. Stratul care va avea cele mai multe ponderi sinaptice de ajustat va determina cât de mult vor rămâne în *wait* toate celelalte straturi. De asemenea, stări *wait* adiţionale sunt introduse în modul ACTUALIZARE datorită faptului că valorile de activare sunt transferate spre EP următor prin cascada *pipeline* RSL doar din două în două cicluri de tact. Aceste stări *wait* pot fi observate în ciclurile de tact 4 şi 6 din fig. 6.10.

$$\begin{aligned} wait &= \left(\frac{3h^2}{2} + O^2 + O(I + 2h)\right) + \left(\frac{3O^2}{2} + 2Oh + \frac{h^2}{2}\right) + \left(\frac{3h^2}{2} + 2IO\right) = \\ &= \frac{7h^2}{2} + \frac{5O^2}{2} + O(3I + 4h) \qquad \text{pentru } O \le I \end{aligned} \tag{6.23}$$

$$\begin{aligned} wait &= \left(\frac{3h^2}{2} + O^2 + O(I + 2h)\right) + \left(\frac{3O^2}{2} + 2Oh + \frac{h^2}{2}\right) + \left(\frac{3h^2}{2} + 2O^2 + 2h(O - I)\right) = \\ &= \frac{7h^2}{2} + \frac{9O^2}{2} + 6Oh + I(O - 2h) \qquad \text{pentru } O > I \end{aligned} \tag{6.24}$$

Pentru RN 512 - h - 10 pe care o analizăm este valabilă relaţia (6.23) care cuantifică cel mai mic procent de stări *wait*. Fig. 6.16 ilustrează eficienţa RSL pe durata învăţării. Răspunsul RSL este foarte diferit pe durata învăţării în raport cu faza *feedforward* şi nu numai prin faptul că cantitatea de procese *calc* devine mult mai semnificativă în faza de învăţare. Deşi toate cele patru categorii de procese cresc cantitativ cu h, fig. 6.16 arată că termenii de gradul 2 care apar în expresiile

proceselor *com* şi *wait* au un efect devastator asupra eficienței RSL. În faza de învățare nu va fi niciodată posibil să se atingă nivelul înalt de eficiență obținut în faza *feedforward*.

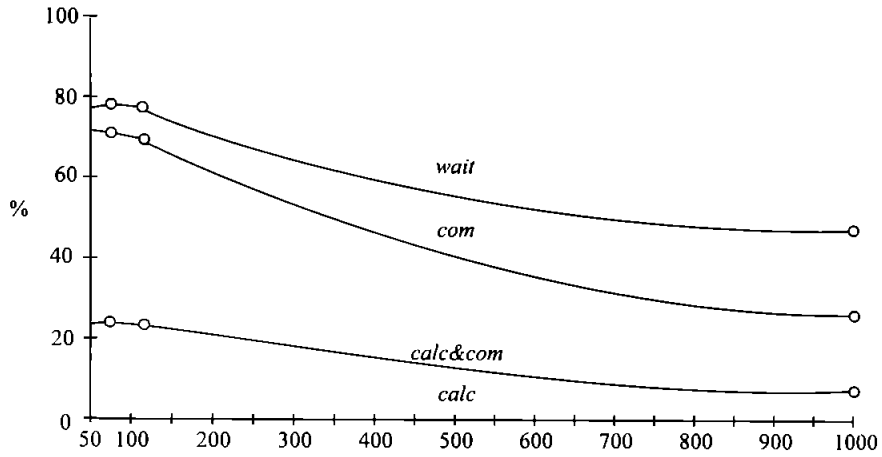



Fig. 6.16. Structura proceselor din RSL pentru RN 512 - h - 10 pe durata învățării. Eficiența RSL este mult mai mică la învățare decât în cadrul fazei *feedforward*; eficiența scade pe măsură de h creşte, indiferent de structura rețelei

### 6.3. Considerații privind optimizarea implementării

Analiza pe care am efectuat-o poate fi utilizată pentru a determina opțiunile posibile atunci când se pune problema optimizării structurii RN mapate în RSL. Performanțele algoritmice ale aplicației trebuie însă menținute permanent în atenție, deoarece atunci când rețeaua este restructurată cu scopul de a optimiza performanțele *hardware*-ului, se poate ajunge la variante de RN incapabile să învețe pentru respectiva aplicație.

Adesea, cel mai important parametru utilizat în evaluarea performanțelor RN este productivitatea aplicației neuronale. În consecință, strategiile care optimizează înainte de toate acest parametru sunt larg aplicate. Aşa cum am arătat prin analiza pe care am efectuat-o, eficiența RSL, latența şi productivitatea nu sunt întotdeauna dependente între ele şi ca atare o creştere a eficienței RSL nu este automat urmată de o creştere similară a productivității sau de o descreştere a latenței. Aceasta este în special adevărat în faza *feedforward*. Prin urmare, optimizarea nu trebuie să însemne întotdeauna simpla minimizare a dimensiunii globale a RN.

#### 6.3.1. Faza feedforward

Analiza fazei *feedforward* a demonstrat că productivitatea este determinată de dimensiunea celui mai mare strat din structura RN. Dacă dimensiunea acestui strat nu se măreşte pe parcursul restructurării, atunci productivitatea poate fi menținută constantă, în timp ce celelalte straturi sunt modificate pentru a optimiza latența şi eficiența RSL. Pe baza analizei făcute, aceasta înseamnă a se opera în zona $h \leq I$. Obținem astfel:

**Regula 1:** Abilitatea de a optimiza productivitatea unei RN implementată într-o RSL crește dacă stratul cel mai puțin flexibil din punct de vedere dimensional (constrâns de către aplicație la o anume dimensiune) este și cel mai mare.

Această regulă poate fi aplicată cel mai adesea stratului de intrare a cărui dimensiune este foarte puțin flexibilă (impusă de către aplicație).

Latența minimă se atinge cu cea mai mică RSL posibilă. Dacă latența este foarte importantă pentru aplicație, atunci reducând dimensiunile straturilor ascunse vom obține performanțe maxime în ceea ce privește performanțele *hardware*-ului.

**Regula 2:** Reducerea latenței se obține prin reducerea dimensiunii globale a RSL, și în particular prin reducerea numărului de EP din stratul ascuns (straturile ascunse). Structura optimă care generează latența minimă nu produce în mod necesar și productivitate maximă. Dacă productivitatea și / sau eficiența sunt considerate mai importante, atunci latența trebuie reevaluată după optimizarea rețelei din punct de vedere al productivității și / sau eficienței.

Eficiența RSL trebuie îmbunătățită fără degradarea productivității și fără a cauza o creștere semnificativă a latenței. Setând $h \leq I$ pentru a obține productivitatea optimă, structura RSL devine parțial determinată. Fig. 6.13 și ecuațiile care definesc eficiența indică faptul că un strat mic de ieșire asigură menținerea unei eficiențe mari în regiunea $h \leq I$. Dacă se operează în afara acestei regiuni, sau cu un strat de ieșire mare, se vor degrada productivitatea, latența și eficiența.

**Regula 3:** Menținerea unei productivități maxime și creșterea eficienței se obțin prin descreșterea monotonă a dimensiunilor straturilor RN, $I > h > O$. Dacă $h = I$ și $O < I$, se poate obține o eficiență de 100 % a RSL fără a degrada productivitatea.

În cazul $h = I$; $O < I$ trebuie acceptată totuși o concesie: creșterea latenței. Este un caz tipic de acceptare a creșterii latenței în favoarea creșterii eficienței RSL.

Ultima regulă se va referi la necesitatea unui strat de ieșire mic. Dacă cel mai mare strat este l, atunci eficiența maximă se va obține în procesoarele EP din stratul $l + 1$. Fig. 6.11 demonstrează această afirmație pentru $l = 0$ și $l = 1$. Cazul $l = 2$ (stratul de ieșire) nu a fost tratat în analiza pe care am efectuat-o datorită valorii fixe alocată parametrilor O și I. Ecuația care exprimă eficiența RSL este validă pentru $I > O$, dar efectul vizibil în fig. 6.11. c) unde stările *wait* se cantonează în stratul ascuns, ar fi vizibil și în stratul de ieșire dacă condiția $I > O$ ar fi inversată. În concluzie, rețelele în care stratul de ieșire este cel mai extins dimensional trebuie evitate.

**Regula 4:** Stratul de ieșire nu trebuie să fie cel mai mare din cadrul rețelei. Dacă această condiție nu poate fi satisfăcută, eficiența RSL nu poate atinge niciodată limita 100 %, indiferent de valorile dimensiunilor I și h.

### 6.3.2. Învățarea

Considerațiile privind optimizarea implementării pentru învățare revendică alte principii comparativ cu faza *feedforward*. În acest caz productivitatea este invers proporțională cu latența. Drept consecință, cel mai important criteriu de optimizare vizează reducerea latenței. Există de asemenea o mai strânsă relație între eficiența RSL și productivitate (latență) comparativ cu faza *feedforward*. Această dependență este reliefată și de rezultatele simulării prezentate în fig. 6.15 și 6.16, unde se poate observa că atât eficiența cât și productivitatea scad pe măsură ce latența crește. Totuși, simpla reducere a dimensiunii rețelei nu este suficientă, deoarece, pentru orice dimensiune

globală particulară a rețelei va exista o structură optimă (o repartizare optimă a procesoarelor EP pe straturi).

Cea mai importantă condiție pentru învățare este asigurarea că O ≤ I. Aceasta va garanta că latența se va evalua pe baza relației (6.17) și ca atare va fi minimizată.

**Regula 5:** Pentru orice rețea și independent de valoarea lui h, pentru asigurarea latenței minime (productivitate maximă) trebuie îndeplinită condiția O ≤ I.

O altă observație importantă care rezultă din relațiile (6.17) și (6.18) este influența puternică pe care stratul ascuns o are asupra latenței. Această dependență se manifestă independent de relația dintre dimensiunile I și O, și este cauzată de strânsa interdependență dintre eficiență și latență. Eficiența scade odată cu creșterea valorii termenilor în $h^2$ din procesele *com* și *wait* (relațiile (6.22), (6.23) și (6.24)). Dimensiunea stratului de ieșire are de asemenea o puternică influență asupra volumului de operații *com* și *wait*. Prin urmare, asigurând condiția ca stratul de intrare să fie cel mai mare (dimensiunea I este impusă de regulă de aplicație) va rezulta o creștere a eficienței și o reducere a latenței.

**Regula 6:** Este indicat ca O și h să fie mai mici decât I. Dacă stratul de intrare este și cel mai mare, eficiența RSL va crește și latența va scădea datorită scăderii timpului alocat operațiilor de comunicație (între procesoarele EP) și de *wait*.

Aceste reguli pot fi utilizate pentru a adopta structura RN atunci când aceasta este implementată în RSL pentru a ne asigura că posibilitățile RSL sunt exploatate pentru performanțe maxime.

### 6.4. Optimizarea performanțelor aplicațiilor

O aplicație neuronală tipică dezvoltată și demonstrată de către mulți cercetători este recunoașterea caracterelor. RN utilizată preia imaginea unui caracter de regulă printr-o matrice de 8 x 8 pixeli. O RN *feedforward* tipică utilizată în clasificarea celor 26 caractere din setul Roman ar avea structura 64 - 20 - 26. Această structură și altele similare au fost considerate tipice pentru problema recunoașterii caracterelor utilizând RN *feedforward* cu algoritmul EBP sau alți algoritmi [VIN 92], [SAM 91], [REY 91]. Pentru implementarea pe RSL această structură de RN necesită o optimizare pentru obținerea unor performanțe deosebite deoarece câteva dintre regulile enunțate la 6.3. sunt violate de către structura inițială. Stratul de intrare este și cel mai extins (regula 6 îndeplinită), dar O > h (regula 3 neîndeplinită). Satisfacerea regulii 3 reprezintă cea mai stringentă necesitate pentru orice RN implementată în RSL. Una dintre principalele probleme care afectează această structură este că atât productivitatea cât și eficiența vor fi reduse datorită lungimii deosebite a vectorului de intrare și a neîndeplinirii condiției h = I (regula 3). De asemenea, deși stratul de ieșire nu este cel mai mare (regula 4 îndeplinită), el este totuși destul de mare pentru a cauza o creștere a procentului de operații *com* în cadrul fazei *feedforward* (regulile 3 și 4) și a celei de învățare (regula 6). Ambele dimensiuni, I și O, trebuie reduse pentru a îmbunătăți performanțele acestei RN mapată în RSL.

O metodă de reducere a dimensiunii vectorului de intrare ar fi compresia. Aceasta impune introducerea unui *hardware* (sau *software*) adițional ceea ce ar putea afecta performanțele aplicației. O altă soluție ar consta în utilizarea unei matrici de pixeli de rezoluție mai redusă; o matrice de 6 x 8 pixeli ar revendica un vector de intrare cu doar 48 componente. O metodă simplă de reducere a dimensiunii vectorului de ieșire constă în utilizarea unui vector de ieșire codificat. Cu numai cinci EP în stratul de ieșire și un decodificator extern care nu va cauza o creștere semnificativă a latenței vor putea fi decodificate cele 26 caractere. Simulările funcționale realizate pentru rețeaua originală

(64 - 20 - 26) și pentru rețeaua modificată (48 - 20 - 5) au arătat că nu sunt necesare mai multe procesoare EP în stratul ascuns al rețelei modificate pentru a produce caracteristici de eroare similare; rețeaua modificată realizează suplimentar o convergență mai rapidă.

O creștere semnificativă a performanțelor s-a constatat odată cu trecerea de la rețeaua originală la cea modificată. Productivitatea RSL a crescut cu 32 % în cadrul fazei *feedforward* și cu 42 % în cadrul fazei de învățare. Latența a scăzut cu 27 % și respectiv 29 % iar eficiența a crescut semnificativ (procentul de operații *calc & com* a crescut de la 58,4 % la 83,1 % în cadrul fazei *feedforward* și de la 23,9 % la 33,3 % în cazul fazei de învățare).

### 6.5. Concluzii

Regulile care au fost definite la 6.3. sunt relativ generale dar nu sunt întotdeauna aplicabile pentru toate RN *feedforward* multistrat mapate în RSL. Există câțiva factori legați de condițiile de investigare fixate care pot influența eficiența acestor reguli.

În primul rând modelul analitic elaborat s-a limitat la RN cu un singur strat ascuns cu $I > O$. Această restricție nu afectează, totuși, atât de mult generalitatea analizei elaborate. Foarte importante sunt dimensiunile straturilor neuronale și pozițiile acestora în cadrul RN. Prin urmare, dacă apar mai multe straturi ascunse acestea pot fi luate în considerare și, de exemplu, regula 3 poate fi extinsă în cazul RN cu *n* straturi ascunse la:

$$I > h_1 > h_2 > \ldots > h_n > O.$$

Cazul $I < O$ nu a fost acoperit și prin simulări, deși analiza teoretică a contribuit la obținerea regulilor elaborate. Concluzia generală care rezultă din aceste reguli privitor la stratul de ieșire este aceea că acest strat ar fi preferabil să fie cel mai mic din cadrul RN.

Un alt factor care trebuie luat în considerare este utilizarea arhitecturii HANNIBAL ca suport de investigare. În modul ACTIVARE se execută operații matricial - vectoriale care sunt comune tuturor algoritmilor neuronali de tip *feedforward* implementați pe RSL. În consecință, regulile obținute pentru faza *feedforward* au un caracter de generalitate și sunt valabile pentru cele mai multe RN *feedforward* implementate pe RSL.

Faza de învățare aferentă algoritmului EBP are o mult mai mare specificitate pentru arhitectura HANNIBAL. Calculul erorilor se poate realiza mult mai eficient în arhitecturile de tip RSL decât în arhitecturile multiprocesor unibus. Degradarea performanțelor este cauzată totuși de inversarea sensului de transfer al informațiilor (propagarea erorilor înapoi). Ar fi necesar totuși un studiu mai larg care să investigheze mai multe tipuri de arhitecturi RSL dedicate RN pentru a asigura o certă generalitate regulilor stabilite pentru faza de învățare.

# 7. ALGORITMI NEURONALI PERFORMANŢI ADAPTAŢI IMPLEMENTĂRII PE REŢELE SISTOLICE

În capitolele 3, 4 şi 5 am proiectat o RS bidimensională dedicată algoritmilor conexionişti. RS obţinută este foarte performantă în execuţia operaţiilor specifice algoritmilor conexionişti şi reprezintă o candidată extrem de bine cotată pentru implementările *hardware* (plăci acceleratoare, circuite VLSI pentru aplicaţii particulare etc.).

Pentru maximizarea performanţelor globale (la nivelul aplicaţiei), trebuie acţionat pe două direcţii:

1. Implementarea RN dedicată aplicaţiei respective pe o structură suport cât mai performantă (maximizarea performanţelor maşinii suport).
2. Maximizarea performanţelor algoritmului neuronal utilizat în aplicaţia respectivă.

Prima direcţie a constituit obiectul capitolelor 3, 4 şi 5 din această lucrare. A doua direcţie vizează performanţele algoritmice ale aplicaţiei şi va constitui obiectul acestui capitol. Vom prezenta în cele ce urmează un exemplu de optimizare algoritmică; algoritmul vizat va fi algoritmul SOFM elaborat de *Kohonen*.

Algoritmul *Self - Organizing Feature Map* (SOFM) a fost pentru prima dată introdus de *Kohonen* [KOH 97] şi de atunci a fost unul dintre cei mai populari algoritmi utilizaţi atât în studiile teoretice cât şi în aplicaţii cum ar fi procesarea vorbirii. RN cu auto-organizare sunt motivate biologic prin abilitatea creierului de a mapa lumea externă pe cortex, unde stimuli externi apropiaţi sau asemănători sunt codaţi pe arii corticale adiacente. Algoritmul SOFM elaborat de *Kohonen* este un algoritm simplu care realizează o astfel de mapare. O secvenţă aleatoare de vectori de intrare este aplicată reţelei, iar ponderile sinaptice sunt astfel ajustate încât să reproducă cât mai fidel posibil distribuţia de probabilitate de la intrare. Ponderile sinaptice se auto-organizează în sensul că neuroni vecini răspund la vectori de intrare asemănători şi tind către valori asimptotice care cuantizează spaţiul de intrare cât mai fidel posibil.

Sistemele *hardware* dedicate implementării RN încearcă să exploateze paralelismul implicit al algoritmilor neuronali. Implementările digitale cele mai indicate sunt arhitecturile sistolice (RS) formate dintr-un număr mare de elemente de procesare (EP) care operează concurent. Maximizarea utilizării acestor resurse distribuite sugerează anumite modificări a algoritmilor neuronali clasici. Astfel de modificări (care vizează creşterea performanţelor RN de tip SOFM implementate în RS) vizează şi algoritmul clasic elaborat de *Kohonen*. În cele ce urmează vom prezenta algoritmul *Kohonen* clasic precum şi două versiuni alternative care devin mai eficiente în cazul implementării SOFM pe RS.

## 7.1. Algoritmul SOFM clasic şi algoritmi SOFM modificaţi

Reţelele SOFM sunt formate fie din M neuroni dispuşi într-o structură liniară (unidimensională) fie, mai general, din M x M neuroni dispuşi într-o structură bidimensională de tip grilă. Vom discuta reţeaua sub forma bidimensională. Răspunsul obţinut pe ieşirea unui neuron $i = (i_1, i_2)$, cu $1 \le i_1, i_2 \le M$, la un vector n - dimensional aplicat la intrare, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, este dat de norma euclidiană $\|\mathbf{x} - \mathbf{w}_i\|_2$, unde $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in}) = (w_{(i1, i2)1}, \ldots, w_{(i1, i2)n})$ este vectorul ponderilor sinaptice aferent neuronului $i = (i_1, i_2)$. Vom nota cu $\mathbf{W}$ setul complet de ponderi sinaptice (vectorul ponderilor sinaptice):

$$\mathbf{W} = (w_{(1,1)}, w_{(2,1)}, \ldots, w_{(M,1)}, w_{(1,2)}, \ldots, w_{(M,M)})$$

### 7.1.1. Algoritmul SOFM clasic

Fie $\mathbf{x}(t)$ vectorul aplicat pe intrarea RN la momentele $t = 0, 1, 2, \ldots$ . Răspunsul $o_i(t)$ obținut pe ieșirea neuronului i va fi o măsură a distanței dintre vectorul său de ponderi sinaptice și vectorul $\mathbf{x}(t)$ aplicat la intrare.

$$o_i(t) = \|\mathbf{x}(t) - \mathbf{w}_i(t)\|_2 \tag{7.1}$$

Regula de selecție a neuronului învingător (*winner*) selectează neuronul $z(t) = (z_1(t), z_2(t))$ care prezintă răspuns minim la ieșire, ca învingător:

$$z(t) = z(\mathbf{x}(t), \mathbf{w}(t)) = \arg \min_i o_i(t) \tag{7.2}$$

Ponderile sinaptice sunt apoi ajustate în acord cu următoarea ecuație:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha(t) \cdot V(z(t) - i, t) \tag{7.3}$$

unde $\alpha(t)$ reprezintă factorul de adaptare sau constanta de învățare ($0 < \alpha(t) < 1$) și $V(z - i, t)$ este o funcție de vecinătate care ia valoarea 1 pentru $i = z$. O formă frecvent aleasă pentru funcția $V(z - i, t)$ este forma rectangulară:

$$V(z-i,t) = \begin{cases} 1 \text{ pentru } \|z-i\|_p \le R(t) \\ 0 \text{ pentru } \|z-i\|_p > R(t) \end{cases} \tag{7.4}$$

unde $R(t)$ reprezintă mărimea vecinătății (în general dependentă de timp). Adesea norma $\|z - i\|_p$ este $\|z - i\|_\infty = \max\{|z_1 - i_1|, |z_2 - i_2|\}$.

Când un vector de intrare $\mathbf{x}$ este aplicat rețelei, regula de ajustare (7.3) "apropie" vectorii ponderilor sinaptice aferenți neuronului învingător și celor din vecinătatea (7.4) de vectorul de intrare $\mathbf{x}$. Vectorii ponderilor sinaptice vor tinde, prin urmare, să reproducă cât mai fidel posibil distribuția de probabilitate de la intrare. Rețeaua SOFM realizează o corespondență între topologia spațiului vectorilor de intrare și topologia funcțiilor de ieșire din matricea SOFM, în sensul că neuroni vecini vor răspunde la stimuli (vectori de intrare) apropiați.

Ecuația (7.2) definește selecția *on-line* a neuronului câștigător, deoarece neuronul câștigător la momentul t, $z(t)$, este determinat comparând vectorul de intrare $\mathbf{x}(t)$ cu vectorul ponderilor sinaptice $\mathbf{W}(t)$. Relația (7.3) realizează actualizarea *on-line* a ponderilor sinaptice în sensul că, după aplicarea vectorului $\mathbf{x}(t)$ pe intrarea rețelei, vectorul ponderilor sinaptice este ajustat în acord cu (7.3). Ponderile sinaptice la momentul $t + 1$, $\mathbf{W}(t + 1)$, depind numai de intrarea $\mathbf{x}(t)$ și de ultima actualizare a vectorului $\mathbf{W}(t)$. Algoritmul definit de ecuațiile (7.2) și (7.3) este algoritmul SOFM clasic. Acesta actualizează ponderile sinaptice după fiecare aplicare a unui vector pe intrarea rețelei.

### 7.1.2. Algoritmul SOFM Batch

În algoritmul *batch*, ponderile sinaptice nu sunt actualizate după aplicarea fiecărui vector de intrare ci numai după aplicarea unui grup de T vectori. Acest nou parametru, T, reprezintă lungimea epocii. Timpul va fi exprimat în acest caz prin relația $t = kT + t'$ unde $k = 0, 1, 2, \ldots$ reprezintă numărul epocii iar $t'$ reprezintă numărul iterației din cadrul unei anumite epoci ($0 \le t' \le T - 1$).

Algoritmul *batch* se defineşte prin selecția unui învingător de grup, deoarece neuronul învingător z(kT + t') va fi selectat prin compararea vectorilor de intrare **x**(kT + t') cu vectorul ponderilor sinaptice **W**(kT) existent la debutul epocii k.

$$z(kT + t') = z(\mathbf{x}(kT + t'), \mathbf{W}(kT)) = \arg\min_{i} \left\| \mathbf{x}(kT + t') - \mathbf{w}_i(kT) \right\|_2 \quad (7.5)$$

Actualizarea ponderilor sinaptice este o actualizare de grup deoarece ajustarea acestora se calculează pornind de la vectorul **W**(kT) existent la începutul epocii k.

$$\mathbf{w}_i(kT + t' + 1) = \mathbf{w}_i(kT + t') + \alpha(kT) \cdot V(z(kT + t') - i, kT) \cdot (\mathbf{x}(kT + t') - \mathbf{w}_i(kT)) \quad (7.6)$$

Dacă facem T = 1 se obține algoritmul SOFM clasic.

### 7.1.3. Algoritmul Batch - M

Algoritmul *batch - M* reprezintă o combinație intermediară între algoritmul SOFM clasic şi algoritmul *batch*. Algoritmul *batch - M* va defini un neuron învingător de grup, aşa cum o face şi algoritmul *batch* prin relația (7.5), dar va realiza o actualizare *on-line* a ponderilor sinaptice:

$$\mathbf{w}_i(kT + t' + 1) = \mathbf{w}_i(kT + t') + \alpha(kT) \cdot V(z(kT + t') - i, kT) \cdot (\mathbf{x}(kT + t') - \mathbf{w}_i(kT + t')) \quad (7.7)$$

Dacă facem T = 1 se obține din nou algoritmul SOFM clasic deoarece lungimea epocii se reduce la o unitate.

### 7.1.4. Implementarea algoritmilor clasic, batch şi batch - M

În funcție de strategia aplicată pentru detecția neuronului învingător şi respectiv pentru ajustarea ponderilor sinaptice, caracteristicile celor 3 algoritmi sunt sintetizate în tabelul 7.1.

| Algoritm | Selecție învingător | Actualizare ponderi sinaptice |
|---|---|---|
| SOFM clasic | *on-line* | *on-line* |
| SOFM *batch* | *batch* | *batch* |
| SOFM *batch - M* | *batch* | *on-line* |

Tabelul 7.1. Caracteristicile fundamentale ale algoritmilor SOFM clasic, *batch* şi *batch - M*

Vom justifica în continuare atenția acordată versiunilor *batch* şi *batch - M* şi vom arăta că acestea prezintă avantaje importante în raport cu versiunea clasică atunci când se pune problema implementării algoritmului SOFM în *hardware*.

Bucla principală a algoritmului SOFM clasic conține următorii paşi:

1. Extrage aleator un vector din setul de antrenament
2. Calculează distanțele dintre vectorul de intrare pe de o parte şi toți neuronii ce compun rețeaua SOFM pe de altă parte (ecuația (7.1))
3. Caută neuronul învingător (ecuația (7.2))
4. Actualizează ponderile sinaptice aferente neuronilor din vecinătatea învingătorului (ecuația (7.3))

Este evident că pasul 2 este cel mai mare consumator de timp şi deci primul candidat la paralelizare. Simplist, neuronii pot fi partiţionaţi în grupe şi distanţele aferente diverselor grupe de neuroni pot fi calculate concurent pe elemente de procesare diferite. Apoi se determină neuronul câştigător şi, în final, unităţile de procesare responsabile cu neuronii din vecinătatea învingătorului vor actualiza ponderile sinaptice tot concurent. Acest prim pas de paralelizare este posibil fără a aduce modificări algoritmului clasic. Creşterea de performanţă se va limita în momentul în care se va atinge pragul de un unic neuron repartizat pe o unitate de procesare. Arhitecturile obişnuite, care dispun de un număr relativ mic de procesoare, nu pot depăşi această limită de paralelizare.

Anumite structuri *hardware* speciale urmăresc creşterea productivităţii prin utilizarea unui număr foarte mare de elemente de procesare (EP) şi deci printr-o paralelizare şi mai accentuată a proceselor. Tipic, această creştere de productivitate se poate obţine aplicând 2 strategii:

1. Procesarea vectorilor de intrare în tehnică *pipeline*. Într-o arhitectură cu număr mare de EP, mai multe unităţi pot partaja calculul unei singure distanţe. Această partajare se poate implementa în tehnică *pipeline*; fiecare EP calculează o distanţă parţială (pe o anumită coordonată) şi o va aduna la suma parţială transmisă din unitate în unitate. Pentru ca o maşină *pipeline* să fie utilizată eficient, procesoarele care şi-au încheiat activitatea pe un set de date (pe un vector de intrare) trebuie să-şi starteze activitatea pe următorul set; toate procesoarele EP vor procesa concurent pe seturi de date diferite. Principala caracteristică a structurii *pipeline* este aceea că are un ciclu (timpul minim între 2 vectori aplicaţi succesiv pe intrări) mai mic decât latenţa (timpul minim dintre aplicarea vectorului de intrare şi obţinerea vectorului rezultat corespunzător). Pentru a menţine structura *pipeline* plină, mai mulţi vectori de intrare vor trebui să intre în pasul 2 (calculul distanţelor) înainte ca primul rezultat să devină disponibil pe ieşire şi deci pasul 4 (actualizarea ponderilor sinaptice) să poată fi iniţiat. Acest tip de procesare *batch* este utilizat în sistemul SYNAPSE - 1 la calculul produsului scalar în cadrul reţelelor de perceptroni multistrat [RAM 92]. Caracterul strict *on-line* al algoritmului SOFM clasic împiedică proiectanţii să aplice această tehnică în cazul reţelelor SOFM, irosind astfel o parte din potenţialul de paralelism al maşinii disponibile. Să notăm că RS proiectată în capitolele 3 şi 4 şi perfecţionată în capitolul 5 asigură procesarea în tehnică *pipeline* a vectorilor de intrare. Fiecărui vector de intrare i se asociază un cod de instrucţiune care permite schimbarea progresivă a modului de operare în interiorul RS (paragraful 5.3.3). Procesarea în tehnică *pipeline* a vectorilor de intrare înseamnă de fapt implementarea pe RS a unui algoritm SOFM de tip *batch* / *batch* - *M*.

2. Procesarea vectorilor de intrare în paralel. O altă posibilitate de a exploata eficient numărul mare de procesoare EP constă în multiplicarea RN pe *hardware*-ul existent; toate copiile RN vor partaja aceleaşi ponderi sinaptice şi deci vor procesa simultan vectori de intrare diferiţi. Această variantă implică de asemenea o procesare de grup (*batch*), deoarece aceeaşi matrice este utilizată pentru a calcula efectele produse de toţi vectorii de intrare procesaţi concurent iar actualizările ponderilor sinaptice vor fi efectuate la sfârşit, cu contribuţia tuturor copiilor RN implementate în matricea EP. Această tehnică este utilizată de exemplu tot în cazul MLP pentru exploatarea arhitecturii de tip dublu inel a sistemului SYNAPSE - 1. Desigur, spre deosebire de costurile introduse de structura *pipeline* din cazul 1., această metodă revendică un pas adiţional pentru actualizarea ponderilor sinaptice partajate (cu contribuţia tuturor copiilor RN multiplicate în aria EP). În anumite cazuri ar putea fi ineficientă partajarea RN pe reţeaua EP şi ar putea fi mai benefică partajarea datelor având în fiecare procesor EP o copie completă a RN [MAN 90].

După implementarea RN în una din cele 2 variante descrise, se poate afirma că paralelismul RN a fost complet exploatat; paralelizarea în continuare ar putea viza numai nivelul intim al operațiilor aritmetico - logice (funcții de activare sau unități aritmetice *pipeline*-izate).

Implementarea SOFM pe sisteme de mare productivitate nu reprezintă singura motivație pentru algoritmii *batch* și *batch - M*. În majoritatea sistemelor există anumite costuri fixe (în ceea ce privește timpul de procesare) asociate cu anumite operații care sunt independente de numărul de vectori procesați. Cel mai la îndemână exemplu ar fi costul încărcării matricii de ponderi sinaptice în sistem. Această operație poate fi foarte costisitoare în anumite sisteme atunci când se implementează RN a căror matrice sinaptică depășește capacitatea memoriei locale la nivelul fiecărui EP și când matricea este descompusă în submatrici. În loc să se reîncarce fiecare submatrice pentru fiecare vector de intrare, acest cost poate fi distribuit pe un grup de vectori de intrare. Costuri similare apar atunci când se transferă anumite tabele în sistem (de exemplu tabela care implementează funcția de activare) sau la golirea / încărcarea structurii *pipeline* atunci când se trece de la pasul curent la pasul următor în cadrul algoritmului.

Atât algoritmul *batch* cât și algoritmul *batch - M* acceptă procesarea de grup (grupuri de vectori de intrare) și deci vor fi foarte indicați pentru implementare pe arhitecturile care vizează un grad înalt de paralelism. Versiunea *batch*, în comparație cu versiunea *batch - M*, diferă doar prin faptul că permite *pipeline*-izarea relației (7.6). Dacă această *pipeline*-izare nu este exploatată, rezultatul $\mathbf{w}(kT + t' + 1)$ va deveni disponibil după procesarea fiecărui vector de intrare și ajustarea ponderilor sinaptice poate fi realizată ca în (7.7). Diferențele între versiunile *batch* și *batch - M* sunt minore în comparație cu distanța de la algoritmul clasic la cei doi algoritmi modificați. Totuși, diferența poate deveni mai evidentă dacă se implementează strategia 2.

Studiile teoretice realizate privind auto - organizarea ponderilor sinaptice și convergența algoritmilor *batch* și *batch - M*, au arătat că aceste 2 versiuni respectă proprietățile fundamentale ale algoritmului clasic [IEN 97]. Primele abordări au avut ca obiect algoritmul clasic. Studiile privind auto - organizarea ponderilor sinaptice (ordonarea ponderilor) consideră algoritmul clasic descris de ecuațiile (7.2) și (7.3) un lanț Markov cu număr infinit de stări (configurațiile ponderilor sinaptice $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M)$). Nu este dificil de arătat că cele 2 configurații strict monotone ale ponderilor sinaptice, $F^+$ (configurația strict crescătoare) și $F^-$ (configurația strict descrescătoare), reprezintă 2 clase de absorbție ale acestui lanț Markov (o clasă de absorbție a unui lanț Markov este reprezentată de un set de stări a căror probabilitate de a rămâne în aceeași clasă este 1). Pentru a demonstra că orice configurație de ponderi inițial dezordonată va deveni ordonată într-un timp finit cu o probabilitate egală cu 1, se utilizează o proprietate a lanțului Markov, care stipulează că, dacă există cel puțin un set de secvențe de vectori de intrare $\mathbf{x}(t)$ care prezintă o probabilitate diferită de zero de a ordona orice configurație de ponderi sinaptice, atunci o secvență aleatoare de vectori de intrare va ordona orice configurație de ponderi într-un timp finit cu probabilitatea 1. Cu alte cuvinte, dacă $P_{\mathbf{W}(0)}$ reprezintă legea de probabilitate a lanțului Markov $\mathbf{W}(t)$, care pornește din starea inițială $\mathbf{W}(0)$ de coordonate distincte ($\mathbf{w}_1(0) \neq \mathbf{w}_2(0) \neq \ldots \neq \mathbf{w}_M(0)$), atunci $P_{\mathbf{W}(0)}(t_{F^+ \cup F^-} < +\infty) = 1$. O astfel de demonstrație a fost conturată de *Kohonen* [KOH 88], a fost stabilită în detaliu de *Cottrell* și *Fort* [COT 87] pentru o distribuție de probabilitate uniformă pe intrare ($P(x) = 1$ pe intervalul $(0,1)$), și de către *Bouton* și *Pages* [BOU 93] pentru o distribuție $P(x)$ neuniformă dar continuă pe intervalul $(0,1)$. Alți autori [ERW 92], [FOR 93] au extins aceste rezultate la alte clase de funcții de vecinătate. Algoritmul *batch - M* respectă în totalitate proprietățile fundamentale de auto - organizare și de convergență ale algoritmului clasic. În condițiile în care factorul de adaptare $\alpha(t)$ are o valoare fixă pozitivă care satisface relația $0 < \alpha(kT) < \frac{1}{T}$ pentru $\forall\, k \geq 0$, atunci și algoritmul *batch* respectă aceste proprietăți de auto - organizare și convergență. Totuși, studiile teoretice lasă și anumite probleme deschise și acestea se referă la viteza de convergență și la robustețea algoritmului în raport cu valorile alese pentru parametrii de învățare. În acest scop, au fost realizate simulări comparative pe probleme reale pentru a demonstra și completa rezultatele teoretice.

## 7.2. Simulări experimentale

### 7.2.1. Iniţializare

Pentru a separa efectul procesării de grup (*batch*) de efectul celorlalte particularităţi algoritmice, acestea din urmă au fost menţinute la minim. Toate valorile (vectori de intrare şi de ieşire, ponderi sinaptice) au fost procesate în virgulă flotantă. Funcţia de vecinătate utilizată a fost cea specificată prin relaţia (7.4). Distanţele în reţeaua SOFM au fost exprimate:

$$\|z - i\|_p = \|z - i\|_1 = |z_1 - i_1| + |z_2 - i_2|$$

Toate simulările au fost realizate cu T = 50 (lungimea epocii). Această valoare poate fi considerată, pe de o parte, suficient de mare pentru a scoate în evidenţă potenţialele deficienţe ale algoritmilor *batch* şi *batch - M* şi, pe de altă parte realistă pentru cazul unor implementări *hardware*. RS de dimensiune N x N perfecţionată în capitolul 5 revendică o epocă T = 2N (egală cu latenţa *pipeline*-ului). T = 50 înseamnă o RS de dimensiune 25 x 25; este o dimensiune rezonabilă pentru tehnologiile actuale dar care poate fi extinsă fără mari eforturi în viitor.

#### A. Setul de antrenament

Datele de test utilizate provin dintr-o aplicaţie de recunoaştere a vorbirii (clasificator de cod de carte prin recunoaşterea vorbirii). Reţeaua SOFM utilizată a fost o reţea bidimensională 10 x 10 iar vectorii de intrare utilizaţi au fost 12 - dimensionali. Setul de antrenament a fost compus din 10000 de prototipuri (vectori de intrare distincţi) pentru fiecare experiment. Fişierul complet de date a fost compus din 112274 vectori. Fiecare rezultat raportat reprezintă media a 10 experimente, prin urmare utilizând 100000 de vectori. Fişierul de test, aplicat în cadrul fiecărui pas al algoritmului de învăţare (epocă), a conţinut 1000 de vectori din cei 12274 neutilizaţi în procesul de învăţare. Vectorii de intrare au fost selectaţi aleator (un vector o singură dată) şi aceleaşi secvenţe de antrenament şi de test au fost aplicate în cazul fiecăruia dintre cei 3 algoritmi (aceleaşi condiţii de antrenament). Fiecare dintre cele 10 experimente au startat cu ponderi sinaptice iniţiale aleatoare dar, ca şi în cazul vectorilor de intrare, ponderile iniţiale au fost aceleaşi în cazul tuturor celor 3 algoritmi.

#### B. Criteriul de convergenţă

Criteriul de eroare utilizat pentru a evalua convergenţa algoritmilor a fost eroarea cuantificată prin relaţia:

$$e(t) = \frac{1}{E \cdot P_{test}} \cdot \sum_{e=0}^{E-1} \sum_{i=0}^{P_{test}-1} \left( \left\| x_i^e - w_{z(i)}(t) \right\|_2 \right)^2 \qquad (7.8)$$

unde:

z - indică indexul neuronului învingător

i - parcurge toate prototipurile (vectorii) din setul de test $P_{test}$.

e - realizează o mediere a erorilor înregistrate în cele E experimente.

Eroarea a fost calculată la finele fiecărei epoci (t = kT). Acest criteriu de eroare nu este singurul criteriu de evaluare a performanțelor rețelelor SOFM în aplicațiile de recunoaștere a vorbirii; alte criterii cum ar fi rata de recunoaștere pot fi luate în considerare. Totuși, criteriul de eroare specificat aici va fi cel mai indicat pentru evaluarea diferențelor dintre algoritmul SOFM clasic și cele 2 versiuni modificate. La compararea rezultatelor experimentale nu a fost utilizată valoarea finală a erorii ci valoarea medie a acesteia pe ultimele 10 epoci (valoarea e). Această strategie elimină hazardul pe care l-ar putea introduce anumite experimente particulare care înregistrează o foarte slabă convergență dar care, printr-un joc al hazardului, înregistrează o foarte mică eroare finală.

### C. Parametrii de învățare

Rata de învățare $\alpha(t)$ din ecuațiile (7.3), (7.6) și (7.7) a fost variată pe parcursul simulărilor conform cu relația:

$$\alpha(t) = \frac{\alpha_0}{1 + K_\alpha \cdot t}$$

În toate experimentele mărimea vecinătății R(t) din relația (7.4) a fost stabilită prin relația:

$$R(t) = 1 + \left\lfloor \frac{R_0}{1 + K_R \cdot t} \right\rfloor$$

În algoritmii *batch* și *batch - M* ambii parametrii ($\alpha(t)$ și R(t)) s-au recalculat la începutul fiecărei epoci (t = kT) și apoi au fost menținuți constanți în interiorul epocii, așa cum indică relațiile (7.6) și (7.7).

### D. Limitarea ponderilor sinaptice

În secțiunea (7.1.4) s-a arătat că, în cazul algoritmului *batch*, pentru a asigura ordonarea ponderilor sinaptice este necesară îndeplinirea condiției:

$$0 < \alpha(kT) < \frac{1}{T} \text{ pentru } \forall\, k \geq 0 \qquad (7.9)$$

În caz contrar, nu numai ordonarea ponderilor nu poate fi garantată ci chiar și menținerea acestor ponderi în domeniul valorilor de intrare. Cu anumite secvențe particulare de vectori de intrare, procesul de învățare poate conduce la divergența anumitor ponderi, cu rezultate catastrofice în ceea ce privește convergența algoritmului *batch*.

Pentru a putea starta algoritmul cu parametrul $\alpha(t)$ plecând de la valori inițiale mari și fără a exista riscul divergenței unor ponderi sinaptice, ponderile pot fi limitate la anumite valori limită minime și respectiv maxime. Deci, la sfârșitul fiecărei epoci ponderile care depășesc limitele prescrise vor fi setate pe aceste limite. Limitarea ponderilor a fost aplicată în cazul algoritmului *batch* în toate experimentele în care valoarea inițială $\alpha_0$ a fost mai mare decât $1/T$ ($\alpha_0 > 0{,}02$). Această tehnică nu a fost aplicată în cazul algoritmilor clasic și *batch - M* deoarece constrângerea cuprinsă în relația (7.9) nu se aplică.

### 7.2.2. Rezultate obţinute

Tabelul 7.2 sintetizează valorile parametrilor utilizaţi în toate experimentele.

| Parametru | Valoare |
|---|---|
| Dimensiunea reţelei SOFM | 10 x 10 |
| Număr intrări | 12 |
| Domeniul de intrare | [-2,01; 1,89] |
| Limitarea ponderilor | -2,5; +2,5 |
| Lungimea epocii (T) | 50 |
| Număr epoci | 200 |
| Număr experimente considerate la mediere | 10 |
| Vectori de test | 1000 |

Tabelul 7.2 Configuraţia parametrilor comună tuturor experimentelor

Mărimea vecinătăţii şi evoluţia acesteia a fost identică în cadrul celor 3 algoritmi; valoarea iniţială a fost setată la $R_0 = 5$ în toate simulările. Această valoare a fost menţinută constantă deoarece ea este determinată de regulă de mărimea matricii SOFM (menţinută constantă în toate experimentele) şi nu depinde de diferenţele existente între diversele versiuni ale algoritmului de învăţare. Efectul parametrului $K_R$ a fost investigat utilizând valorile 0,2 ; 0,02 ; 0,002 ; 0,0002.

Pentru toţi algoritmii rata iniţială de învăţare $\alpha_0$ a fost setată pe 5 valori distribuite relativ uniform pe intervalul (0, 1). Aceste valori au fost: 0,1 ; 0,3 ; 0,5 ; 0,7 şi 0,9. Rata de descreştere $K_\alpha$ a fost setată pe 6 valori: 0,25 ; 0,025 ; 0,0025 ; 0,00025 ; 0,000025 şi 0,0000025.

Toate combinaţiile acestor valori au fost aplicate celor 3 algoritmi; adiţional pentru versiunea *batch* au fost aplicate câteva valori speciale. În acest caz $\alpha_0$ trebuie să rămână în intervalul (0, 1/T), în acord cu condiţia (7.9). Valorile utilizate au fost 0,002 ; 0,006 ; 0,010 ; 0,014 şi 0.018. Valori foarte mici au fost utilizate şi pentru coeficientul $K_\alpha$ pentru a facilita o bună convergenţă a algoritmului *batch*.

#### A. Evoluţia erorii

Fig. 7.1 indică evoluţia erorii pe durata primelor 60 epoci pentru cele mai bune 10 rulări ale fiecăruia dintre cei 3 algoritmi. Axa orizontală pentru versiunile *batch* şi *batch - M* indică numărul epocii; pentru a obţine aceeaşi scară pe abscisă la versiunea normală eroarea a fost marcată după fiecare set de T vectori aplicaţi la intrare. Erorile finale, în cazul celor mai bune rulări pentru cele 3 versiuni clasic, *batch* şi *batch - M*, au fost aproape identice: 0,176227 ; 0,178781 şi respectiv 0.174834.

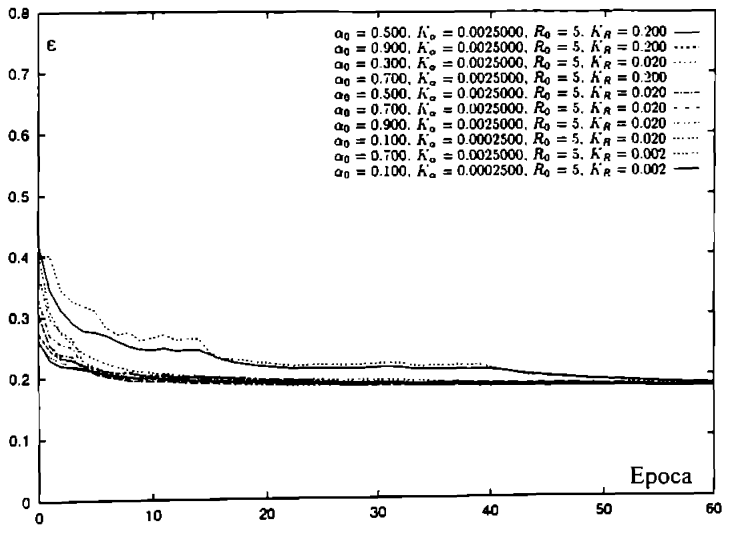
#### B. Viteza de convergenţă

Scopul esenţial al noilor versiuni introduse (*batch* şi *batch - M*) este acela de a creşte viteza procesului de învăţare. Acest deziderat s-ar putea atinge printr-o implementare *hardware* cu un grad mai înalt de paralelism, grad pe care cele 2 versiuni *batch* îl pot permite. Pentru a menţine acest avantaj este important ca cele 2 versiuni *batch* să nu fie semnificativ mai lent convergente decât versiunea clasică.

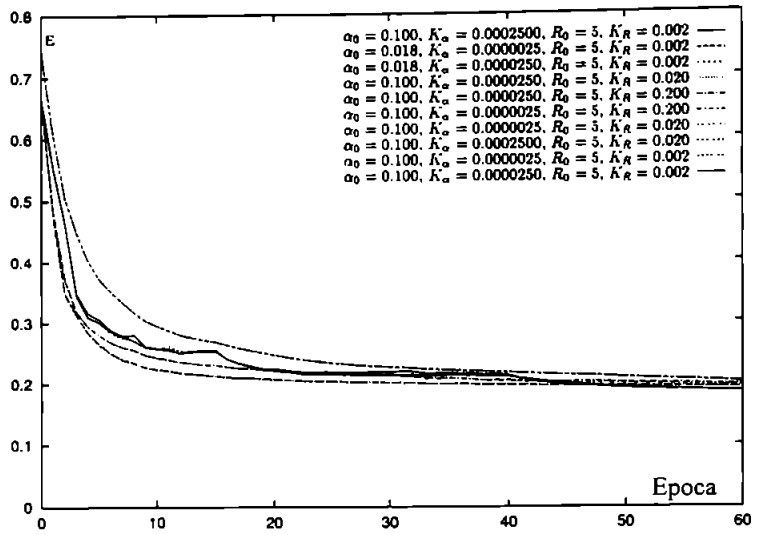
Pentru o comparaţie cât mai exactă, o rulare bună şi cu aceleaşi valori pentru $K_R$ şi $K_\alpha$ trebuie selectată pentru fiecare versiune a algoritmului SOFM. Acest lucru este foarte important deoarece cei 2 parametrii (primul în special) afectează profund viteza de organizare a reţelei SOFM.

Amploarea acestui efect poate fi observată în figura 7.1. a) unde 2 rulări ale algoritmului original converg sensibil mai lent datorită valorilor mici ale coeficientului $K_R$. În afara acestui efect, putem observa că diferențierea între cele mai bune rulări este mică, ceea ce justifică compararea celor mai bune rulări (câte una pentru fiecare versiune).

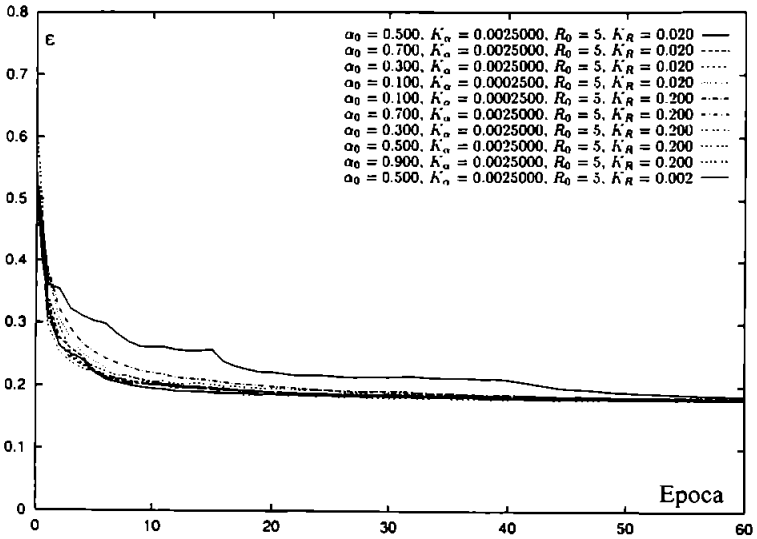
Au fost aleși coeficienții celei mai bune rulări a algoritmului clasic ($K_\alpha$ = 0,0025 și $K_R$ = 0,2); utilizând aceste 2 valori au fost alese și cele mai bune rulări aferente versiunilor *batch*. Prin contrast cu valorile comune alese pentru $K_\alpha$ și $K_R$, pentru coeficientul $\alpha_0$ a fost aleasă acea valoare (diferită de la caz la caz) care corespunde comportamentului optimal al fiecărei versiuni. Rata de descreștere, $K_\alpha$, aleasă pentru algoritmul clasic s-a dovedit a fi optimă și pentru algoritmul *batch - M*. În cazul algoritmului *batch*, luând în considerare cel mai bun experiment nondivergent, convergența este rapidă la început dar valoarea excesivă a ratei de descreștere asociată cu valoarea mică a constantei inițiale $\alpha_0$, încetinește ulterior convergența (viteza de scădere a erorii) comparativ cu celelalte 2 versiuni. Acest lucru este vizibil în figura 7.2 unde sunt ilustrate primele 60 epoci din cadrul algoritmului de învățare pentru fiecare dintre cele 3 versiuni (cu $K_\alpha$ = 0,0025 și $K_R$ = 0,2); tot în fig. 7.2 se asociază și cea mai bună rulare pentru algoritmul *batch* ($K_\alpha$ = 0,000025 și $K_R$ = 0,002). Această cea mai bună rulare care este relativ mai lent convergentă în prima parte a intervalului, este mai apropiată ca performanțe de celelalte rulări și de aceea a fost utilizată în comparațiile care urmează. Algoritmul clasic este cel mai rapid convergent: după cea de-a șaptea epocă atinge deja pragul de $(1 + 20\ \%)\cdot e_f$. Am notat cu $e_f$ - eroarea finală. Algoritmul *batch - M* este cu foarte puțin mai lent convergent decât cel clasic: după a 20-a epocă rezultatele acestuia sunt aproape identice cu cele ale algoritmului clasic. Versiunea *batch*, deși avantajată de acea alegere liberă a parametrilor de învățare, este net mai lent convergentă pe întreg domeniul și necesită 42 epoci pentru a atinge pragul de $(1 + 20\ \%)\cdot e_f$. Rezultatele sunt sintetizate în tabelul 7.3 care evidențiază la ce număr de epocă eroarea devine mai mică sau egală cu valorile de prag considerate. Valorile de prag sunt considerate ca procente peste cea mai mică eroare absolută obținută (versiunea *batch - M*: 0,174834).




a) algoritmul clasic




b) algoritmul *batch*




c) algoritmul *batch - M*

Fig. 7.1. Cele mai bune 10 rulări

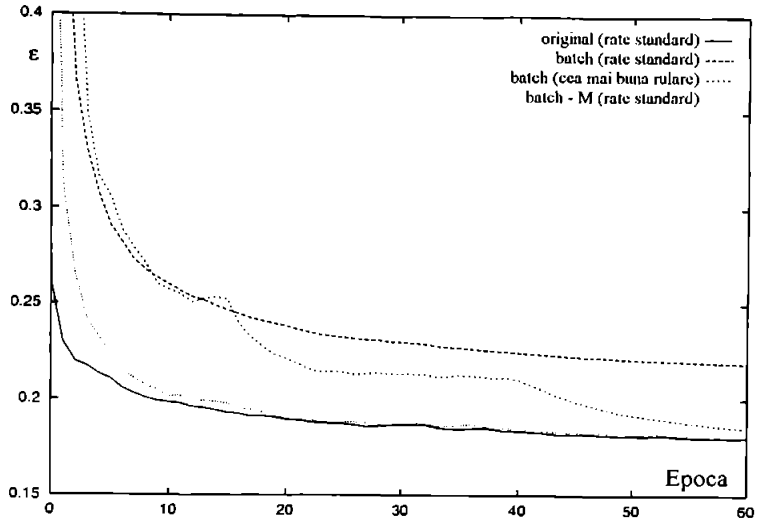



Fig. 7.2. Viteza de convergenţă comparativă a celor mai bune rulări cu rate standard ($K_\alpha$ = 0,0025 ; $K_R$ = 0,2)

| Prag | Eroare | clasic | *batch* | *batch - M* |
|---|---|---|---|---|
| 50 % | 0,262 | 1 | 10 | 4 |
| 20 % | 0,210 | 7 | 42 | 9 |
| 10 % | 0,192 | 18 | 52 | 21 |
| 5 % | 0,184 | 40 | 68 | 43 |
| 1 % | 0,177 | 136 | – | 142 |

Tabelul 7.3. Viteza de convergenţă a celor 3 algoritmi

**C. Robusteţea**

Rezultatele prezentate în paragraful anterior arată că, pe o problemă reală, viteza de convergenţă a algoritmului *batch - M* şi, cu o oarecare aproximaţie, şi a algoritmului *batch* sunt comparabile cu cea a algoritmului SOFM clasic. Datorită imposibilităţii practice de a rula algoritmii cu foarte multe combinaţii ale parametrilor de învăţare, este important să se evalueze cât de critică este problema alegerii acestor parametri.

O primă posibilitate de evaluare constă în raportarea numărului de experimente (pentru fiecare versiune în parte) care conduc la o eroare finală cu peste 20 % mai mare decât cea corespunzătoare celei mai bune rulări, la numărul total de experimente efectuate. În cazul algoritmului original, 58 din cele 120 experimente au condus la o eroare cu 0,210 mai mare decât eroarea finală minimă (au prezentat o distorsiune finală mai mare de 0,210). Similar, în cazul algoritmului *batch - M* acest lucru s-a întâmplat cu 54 experimente. Aşa cum era de aşteptat situaţia este sensibil mai proastă în cazul algoritmului *batch* cu parametri normali, deoarece multe experimente au fost divergente: 89 experimente n-au atins nivelul de distorsiune finală de 0,210. Mai surprinzător, rezultatele sunt chiar mai slabe şi în cazul experimentelor efectuate cu parametri adaptaţi (micşoraţi) caz în care numărul de experimente care nu ating nivelul final de distorsiune

impus este 95. Deoarece rulările cu parametri reduşi sunt mult mai puţin catastrofice decât cele cu parametri normali (toate experimentele converg în cazul rulărilor cu parametri reduşi), este clar că viteza de convergenţă a algoritmului *batch* este "stânjenită" de necesitatea satisfacerii condiţiei (7.9).

În final, trebuie menţionat că parametrii corespunzători la şapte dintre cele mai bune zece experimente aferente algoritmului clasic se regăsesc şi în cele mai bune zece experimente aferente algoritmului *batch - M*: aceasta demonstrează că orice metodă euristică de optimizare a parametrilor aferenţi algoritmului clasic poate fi aplicată fără nici o modificare şi în cazul algoritmului *batch - M*. Rezultatele sunt sintetizate în tabelul 7.4.

| Algoritm | Experimente cu Distorsiune $\mathbf{e} \geq 1{,}2 \cdot \mathbf{e}_{optim}$ | | Distorsiunea **e** în experimentul cel mai slab |
|---|---|---|---|
| clasic | 58 | 48 % | 0,53 |
| *batch* (parametri normali) | 89 | 74 % | 21,03 |
| *batch* (parametri adaptaţi) | 95 | 79 % | 1,22 |
| *batch - M* | 54 | 45 % | 0,43 |

Tabelul 7.4. Robusteţea algoritmilor clasic, *batch* şi *batch - M*

**D. Concluzii**

Simulările experimentale prezentate în paragraful anterior confirmă rezultatele studiilor teoretice. Într-adevăr rezultatele experimentale confirmă că, pe o problemă reală, algoritmul *batch - M* dă rezultate sensibil identice (accidental mai bune) decât algoritmul original. Aceste rezultate au fost evaluate cu referire la:

1. calitatea rezultatului final
2. viteza de convergenţă
3. sensibilitatea redusă la parametrii de învăţare (robusteţe).

Numai în ceea ce priveşte viteza de convergenţă rezultatele experimentale reliefează o convergenţă ceva mai lentă a algoritmului *batch - M*; să menţionăm totuşi că diferenţa este mică şi va putea fi supracompensată şi mult depăşită de gradul mult mai înalt de paralelism pe care acest algoritm îl permite în implementările *hardware* (procesarea în tehnică *pipeline* a vectorilor de intrare, *pipeline*-izare asigurată şi de RS proiectată în capitolele 3, 4 şi 5). Situaţia este mai puţin pozitivă în cazul algoritmului *batch*. În acest caz, restricţia de utilizare a parametrilor adaptaţi, impusă de condiţia (7.9) poate afecta (uneori sever) viteza de convergenţă.

# 8. CONCLUZII

## 8.1. Obiectivele lucrării

Conducerea adaptivă în timp real a proceselor rapide şi complexe reclamă sisteme de conducere capabile să reacţioneze prompt la evenimentele care au loc în proces. *Neurocontroller*-ele implementate în *hardware* propuse în această lucrare satisfac pe deplin aceste cerinţe.

Studiile prezentate în lucrare au avut drept scop concepţia unor arhitecturi specializate pentru implementarea modelelor conexioniste. Problematica vizată a fost descompusă în mai multe părţi:

- În domeniul reţelelor neuronale există o multitudine de modele care diferă prin structură şi prin funcţiile pe care le realizează. Pentru o problemă dată, alegerea reţelei optime nu se poate baza încă pe nişte criterii bine definite; alegerea se bazează în mare măsură pe un grad de pricepere câştigat prin experienţă. În această lucrare s-au studiat principalele modele conexioniste actuale evidenţiindu-se elementele lor constitutive: funcţia de activare neuronală, structura, regulile de învăţare etc. Studiul a permis evidenţierea unui ansamblu de operaţii matricial - vectoriale comune tuturor modelelor studiate. În faza de implementare *hardware* a algoritmilor conexionişti aceste operaţii comune (set de primitive) ne-au permis o detaşare de constrângerile arhitecturale impuse de diversele modele. Astfel, în locul unei simple integrări de modele, studiul s-a concentrat asupra integrării unor lanţuri de primitive astfel încât să se obţină o implementare de înaltă performanţă care să emuleze toate modelele conexioniste studiate.

- Pentru a înţelege şi a inventaria constrângerile pe care le creează implementarea *hardware* (VLSI) a reţelelor neuronale, în capitolul 2 am trecut în revistă cele mai reprezentative realizări ale etapei actuale (digitale şi analogice); aceasta a permis inventarierea principiilor esenţiale care trebuie să stea la baza concepţiei unei arhitecturi conexioniste. Optând pentru integrarea în variantă digitală a matricii sinaptice, am estimat că implementarea sistolică ne va permite să obţinem performanţe deosebite; aceste aşteptări au fost confirmate de RS proiectată, implementată, perfecţionată şi evaluată în capitolele 3, 4 şi 5.

- Reţeaua sistolică bidimensională propusă realizează într-o manieră originală funcţiile esenţiale aferente algoritmilor conexionişti, atât pentru faza de recunoaştere cât şi pentru faza de învăţare. Această arhitectură permite implementarea operaţiilor conexioniste de bază (setul de primitive): produsul matrice - vector, adunarea şi scăderea matricilor, produsul unui vector cu transpusul său, detecţia identităţii dintre 2 vectori etc. Ea permite de asemenea ajustarea ponderilor sinaptice care reprezintă funcţia de bază a tuturor sistemelor evolutive. Arhitectura RS este modulară, se bazează numai pe conexiuni locale şi permite extensia dimensională printr-un simplu pavaj rectangular de circuite VLSI.

- Un prim aport algoritmic se referă la dezvoltarea mai multor algoritmi care se bazează pe setul de primitive stabilit, atât pentru faza de recunoaştere cât şi pentru cea de învăţare. Formularea matematică a operaţiilor impuse de algoritmii vizaţi a fost transformată în ecuaţii de dependenţă între diferitele puncte de calcul (celule) din structura RS. Atât transferurile de date din interiorul RS cât şi cascadarea diferitelor

operaţii impuse de algoritmii vizaţi au fost optimizate, exploatând toate avantajele pe care le oferă reţeaua sistolică bidimensională: sincronizarea naturală a transferurilor de date, reducerea numărului de legături fizice dintre celulele din structura RS (comunicaţii strict locale), *pipeline*-izarea operaţiilor, a transferurilor de date şi a instrucţiunilor în cadrul RS. Calculele efectuate şi comunicaţiile de date sunt strict locale, celula sistolică este programabilă (diagonală sau non-diagonală) ceea ce va permite o extensibilitate foarte comodă a RS.

- Avantajele arhitecturii sistolice prezentate pot fi rezumate astfel:

**A. Modularitate**

Modularitatea este prezentă la mai multe nivele:

- celula sistolică este construită utilizând 3 tipuri de blocuri funcţionale: registru, sumator, multiplexor. Înmulţitorul paralel - serie are la bază un sumator şi un registru. Înmulţitorul serie - serie introduce 2 elemente noi: *(3, 2) counter* (*full - adder*) şi *(5,3) counter*.
- circuitul VLSI este constituit din celule identice care pot fi programate diagonale sau non-diagonale.
- reţeaua sistolică este realizată printr-un pavaj regulat de circuite identice, cu semnalele de la pini interconectate direct (fără interfeţe specializate).

Este posibilă construcţia unor reţele de dimensiuni mari prin simpla adăugare de circuite. Limita practică de extensibilitate este fixată doar de lungimea registrului PS care memorează potenţialul neuronului.

**B. Simplitate**

Celula sistolică este simplă, constituită doar din câţiva operatori şi câteva registre. Proiectarea, testarea şi evaluarea ei se poate realiza rapid şi cu un grad înalt de siguranţă. Validarea celulei se poate face examinând şi evaluând fiecare bloc independent din structura acesteia. Proiectarea şi realizarea celulei sistolice înseamnă de fapt realizarea întregii reţele.

**C. Reconfigurabilitate**

Implementarea în variantă FPGA prezintă avantajul reconfigurabilităţii. Reţeaua sistolică nu trebuie să acopere în orice moment de timp, toţi algoritmii conexionişti cunoscuţi deoarece o astfel de abordare ar complica inutil structura celulei sistolice. Pe baza gradului de asemănare se pot constitui clase de algoritmi conexionişti şi pentru fiecare clasă se poate proiecta celula adecvată. *Software*-ul de gestiune aferent plăcii acceleratoare va putea reconfigura dinamic RS în funcţie de algoritmul neuronal ce urmează a se executa (aplicaţia lansată în execuţie pe sistemul gazdă). De vreme ce structura CS rămâne în permanenţă simplă (deşi se reconfigurează dinamic în timp), se vor putea obţine (pentru fiecare algoritm neuronal în parte) viteze maxime de procesare în interiorul RS. Reconfigurarea RS se reduce la o simplă programare a circuitelor FPGA (modificarea conţinutului tabelelor LUT).

**D. Grad înalt de paralelism**

RS propusă reprezintă o implementare care integrează sinapse şi nu neuroni. În contextul unei implementări VLSI, paralelismul masiv constituie un factor esenţial. Paralelismul trebuie să fie cât mai fin posibil pentru a reproduce

structura distribuită aferentă RN; orice multiplexare a calculelor va conduce la reducerea calităţilor intrinseci aferente reţelelor conexioniste: reprezentare distribuită şi toleranţă la defect. RS propusă satisface aceste deziderate deoarece fiecare sinapsă este integrată într-o celulă sistolică proprie (cel mai fin paralelism posibil). Trebuie să precizăm că decizia de a integra sinapse şi nu neuroni atrage şi un dezavantaj: dimensiunea relativ redusă a RN implementabile într-un VLSI. Aici progresul tehnologic îşi va spune cuvântul. În momentul actual un FPGA XC 40250, de exemplu, poate găzdui o RN recurentă complet interconectată compusă din 22 neuroni. În condiţiile dezvoltărilor arhitecturale propuse la 5.4 o matrice sinaptică de dimensiuni mari ar putea fi partajată în submatrici de dimensiuni egale cu dimensiunea RS. În timp ce RS procesează submatricea curentă, următoarea submatrice poate fi încărcată în RS (fără consum suplimentar de timp). În condiţiile în care interfaţa RS - sistem gazdă asigură transferuri de date în ritmul solicitat de RS, dimensiunea RN emulate pe RS propusă nu mai constituie o problemă.

**E. Performanţe**

RS propusă este perfect adaptată algoritmilor conexionişti. Performanţele (CPS, UPS) evaluate la 4.6 sunt net superioare în raport cu alte realizări cunoscute. Aceste performanţe estimate ne îndreptăţesc să considerăm că RS analizată conduce la implementări VLSI de mare eficienţă utilizabile în aplicaţii reale complexe (timp real).

- Al doilea aport algoritmic se referă la elaborarea unor versiuni alternative ale algoritmilor neuronali clasici. RS reprezintă un *hardware* distribuit care implică un grad masiv şi fin de paralelism în procesare. Maximizarea utilizării acestor resurse *hardware* distribuite sugerează anumite modificări ale algoritmilor neuronali clasici. O astfel de problematică este abordată în capitolul 7 al lucrării unde se prezintă algoritmul SOFM clasic precum şi două versiuni alternative (*batch* şi *batch - M*) care devin mult mai eficiente în cazul implementării reţelei SOFM pe reţeaua sistolică propusă. Spre deosebire de algoritmul clasic, algoritmii *batch* şi *batch - M* permit un grad mult mai înalt de paralelism la nivelul implementării *hardware*. Aceşti algoritmi permit procesarea în tehnică *pipeline* a vectorilor de intrare, *pipeline*-izare asigurată şi chiar revendicată de către maşina suport (RS).

### 8.2. Contribuţii

În continuare vom prezenta, pe capitole, contribuţiile originale aduse în cadrul prezentei lucrări:

Capitolul 1

- o sinteză a principalelor modele conexioniste cu evidenţierea elementelor lor constitutive: funcţia de transfer, structură, reguli de învăţare etc.
- evidenţierea unui set de primitive (operaţii matricial - vectoriale comune tuturor modelelor studiate). Acest set de primitive a permis o detaşare de constrângerile arhitecturale impuse de diversele modele. Implementarea *hardware* vizată nu va fi o simplă integrare de modele; vor fi integrate lanţuri de primitive care vor conduce la o implementare de înaltă performanţă capabilă să emuleze toate modelele conexioniste studiate.

Capitolul 2

- un studiu comparativ al structurilor candidate la integrare (integrarea neuronilor şi respectiv integrarea sinapselor) cu evidenţierea avantajelor / dezavantajelor fiecărei structuri
- o sinteză a celor mai importante realizări în domeniul neuroprocesoarelor digitale şi analogice
- stabilirea principiilor de proiectare a neuroprocesoarelor sistolice.

Capitolul 3

- o analiză a metodelor şi descrierilor sistolice de tip SFG şi DG care permit localizarea spaţială a algoritmilor conexionişti
- elaborarea unei arhitecturi sistolice generale din care vor deriva 3 arhitecturi exploatabile:
  - inel sistolic cu componentele vectorului fixe
  - inel sistolic cu sume parţiale fixe
  - arhitectura cu coeficienţi sinaptici ficşi

Capitolul 4

- definirea arhitecturii sistolice optime şi proiectarea acestei arhitecturi
- definirea algoritmului sistolic şi demonstrarea validităţii acestuia (atât pentru faza de recunoaştere cât şi pentru faza de învăţare) pe baza Sistemului de Ecuaţii Recurente Uniforme
- proiectarea celulei sistolice şi a reţelei sistolice caracterizată de următoarele avantaje:
  - permite implementarea fazei de recunoaştere pentru toţi algoritmii neuronali clasici şi a celor mai diverse reguli de învăţare
  - permite procesarea în tehnică *pipeline* a vectorilor de intrare ceea ce conduce la o creştere spectaculoasă a eficienţei RS; aceasta poate atinge valori de 100 % în fazele de procesare staţionară (recurenţe)
  - permite extensibilitatea RS printr-un simplu pavaj rectangular de circuite VLSI
- proiectarea şi implementarea RS în structuri FPGA; testarea funcţională şi evaluarea performanţelor reţelei proiectate.

Capitolul 5

- elaborarea unei strategii de perfecţionare a reţelei sistolice dedicate algoritmilor conexionişti prin *pipeline*-izarea fluxurilor de date şi de instrucţiuni în interiorul RS, prin perfecţionarea structurii celulei sistolice şi prin implementarea unui mecanism performant de interschimbare a matricilor sinaptice
- proiectarea unui înmulţitor serie - serie de înaltă performanţă dedicat celulei sistolice perfecţionate. Avantajele majore ale înmulţitorului proiectat sunt:
  - grad înalt de modularitate (extrem de important pentru implementările VLSI)
  - latenţă extrem de redusă între biţii aplicaţi la intrare şi bitul corespondent obţinut la ieşire
  - permite extensia produsului obţinut la ieşire la orice lungime (extensie de semn)

Capitolul 6

- analiza unei propuneri alternative: Reţeaua sistolică liniară (RSL) cu integrare de neuroni şi nu de sinapse
- definirea unei strategii şi stabilirea unui set de reguli privind optimizarea şi creşterea eficienţei RSL.

Capitolul 7

- elaborarea unor versiuni alternative pentru algoritmii neuronali clasici, versiuni care permit un grad mult mai înalt de paralelism la nivelul implementării *hardware* (procesarea în tehnică *pipeline* a vectorilor de intrare)
- analiza performanţelor şi a vitezei de convergenţă aferente algoritmilor propuşi prin metode teoretice şi prin simulări experimentale.

### 8.3. Dezvoltări viitoare

Pornind de la studiul prezentat în această lucrare, pot fi angajate mai multe dezvoltări. O primă axă abordată deja parţial în lucrare ar viza realizarea unui neurosistem bazat pe un computer gazdă (de uz general) şi o placă acceleratoare bazată pe RS propusă. Prin puterea de calcul pusă la dispoziţie, un astfel de sistem ar constitui un suport esenţial pentru cercetările desfăşurate în domeniul algoritmilor conexionişti.

Realizarea neurosistemului presupune abordarea următoarei problematici:

- Proiectarea şi construcţia plăcii acceleratoare. Este o sarcină complexă care implică rezolvarea următoarele probleme:
  - definitivarea structurii celulei sistolice
  - alegerea tehnologiei VLSI care se va utiliza pentru implementarea RS
  - definirea dimensiunii RS şi matricii de circuite VLSI aferentă RS
  - proiectarea *hardware*-ului care implementează funcţiile sigmoidale. Funcţia sigmoidală nu a făcut obiectul cercetărilor prezentată în această lucrare. Ea a fost lăsată în sarcina sistemului gazdă dar ar putea fi de asemenea implementată într-un circuit specializat. Circuitul VLSI dedicat s-ar putea baza pe tabele *look - up* sau pe implementări *hardware* speciale [ZHA 96]
  - proiectarea memoriei locale aferentă plăcii acceleratoare. Memoria locală are sarcina de a reţine şi a pune la dispoziţia RS datele de lucru (matricea sinaptică, vectorii de intrare etc.). Interfaţa dintre memoria locală şi RS reclamă o atenţie deosebită; transferurile de date între cele două subsisteme trebuie să asigure un ritm normal de procesare în interiorul RS (fără stări *wait*)
  - proiectarea interfeţei dintre placa acceleratoare şi sistemul gazdă. Lărgimea canalului de comunicaţie dintre sistemul gazdă şi placa acceleratoare trebuie să fie suficient de mare pentru a asigura procesarea în ritm normal în interiorul RS (fără stări *wait* cauzate de indisponibilitatea anumitor date).

- Proiectarea *software*-ului de gestiune aferent plăcii acceleratoare. Performanţele globale depind esenţial de eficienţa acestui *software*.

O a 2-a axă de dezvoltare vizează perfecţionarea algoritmilor conexionişti. Această direcţie a fost abordată în capitolul 7 dar a vizat doar algoritmii SOFM. Cercetarea trebuie continuată şi extinsă şi asupra celorlalte modele conexioniste.

O a 3-a axă de dezvoltare ar viza concepția unor circuite *Wafer Scale Integration* reconfigurabile care ar putea găzdui rețele complexe compuse dintr-un număr foarte mare de celule sistolice. Aceste implementări ar urma să fie utilizate în aplicațiile reale cele mai complexe (timp real).

**Prescurtări utlizate**

ART - *Adaptive Resonance Theory* (arhitectură specializată de RN)
BAM - *Bidirectional Associative Memory* (memorie asociativă bidirecţională)
CLB - *Configurable Logic Block* (bloc logic configurabil)
CNAPS - *Connected Network of Adaptive Processors* (reţea interconectată de procesoare adaptive)
CP - *Control Processor* (procesor de control)
CPIF - *Control Processor Interface* (interfaţa procesorului de control)
CPS - *Connections Per Second* (conexiuni pe secundă)
CS - Celulă Sistolică
CSC - *CNAPS Sequencer Chip* (circuitul de secvenţiere aferent ariei CNAPS )
DAP - *Distributed Array Processor* (arie de procesoare distribuită)
DG - *Dependence Graph* (graf de dependenţă)
DRAM - *Dynamic Random Access Memory* (memorie RAM dinamică)
DSP - *Digital Signal Processor* (procesor de semnal digital)
EBP - *Error Back Propagation* (propagarea erorii înapoi); algoritm de învăţare utilizat în cazul RN *feedforward* multistrat
EP - Element de Procesare
FPGA - *Field Programmable Gate Array* (circuite VLSI programabile)
GCPS - *Giga-Connections Per Second* (giga conexiuni pe secundă)
ICANN - *Proceedings of International Conference on Artificial Neural Networks*
ICNN - *Proceedings of the IEEE International Conference on Neural Networks*
IW - *Instruction Word* (instrucţiune)
LUT - *Look - Up Table* (tabelă *look - up*)
LVQ - *Learning Vector Quantization* (arhitectură specializată de RN)
MAC - *Multiplication & ACumulation unit* (unitate de înmulţire şi acumulare); realizează înmulţirea a 2 factori urmată de acumularea produsului la rezultatul obţinut în operaţia MAC anterioară.
MCPS - *Mega-Connections Per Second* (mega conexiuni pe secundă)
MIMD - *Multiple Instruction - Multiple Data streams* (fluxuri multiple de instrucţiuni şi date)
MLP - *MultiLayer Perceptrons* (reţele de perceptroni multistrat)
MMU - *Memory Management Unit* (management de memorie)
PS - *Partial Sum* (sumă parţială)
RBF - *Radial Basis Function* (reţele neuronale cu funcţii radiale de bază)
RCE - *Restricted Coulomb Energy* (arhitectură specializată de RN)
RISC - *Reduced Instruction Set Computer* (calculator cu set redus de instrucţiuni)
RN - Reţea Neuronală
RS - Reţea Sistolică
RSL - Reţea Sistolică Liniară
SERU - Sistem de Ecuaţii Recurente Uniforme
SIMD - *Single Instruction flow, Multiple Data flow* (flux unic de instrucţiuni şi multiple fluxuri de date)
SFG - *Signal Flow Graph* (graf de mişcare a datelor)
SOFM - *Self Organizing Feature Map* (reţeaua cu auto-organizare a lui *Kohonen*)
SRAM - *Static Random Access Memory* (memorie RAM statică)
TDNN - *Time Delay Neural Networks* (arhitectură specializată de RN)
UPS - *Updates Per Second* (actualizări pe secundă)
VLSI - *Very Large Scale Integration* (circuite integrate pe scară largă)
WCNN - *World Congress on Neural Networks*

**BIBLIOGRAFIE**


[ACI 93] Antonio d'Acierno, Roberto Vaccaro. "A Parallel Implementation of the Back - Propagation of Errors Learning Algorithm on a SIMD Parallel Computer", ICANN 1993, pag. 1074 - 1077.

[ALI 91] C. Alippi, G. Storti - Gajani. "Simple Approximation of Sigmoid Functions: Realistic Design of Digital VLSI Neural Networks", Proc. IEEE Int. Symp. Circuits and Systems, 1991, pag. 1505 - 1508.

[ALM 89] L. Almeida. "Back Propagation in perceptrons with feedback", Neural Computers, Springer Verlag, R. Eckmiller, C.v.d. Malsburg Eds., 1989.

[AMI 85] D. Amit, H. Gutfreund, H. Sompolinski. "Storing infinite number of patterns in a spin - glass model of neural networks", Physical Review Letters, vol. 55, nr. 14, 1985, pag. 1530 - 1533.

[AND 77] J. A. Anderson, J. W. Silverstein, S. A. Ritz, R. S. Jones. "Distinctive features, categorical perception on probability learning: some applications of a neural model", Psychological Review, Nr. 84, 1977, pag. 413 - 451.

[ASA 90] K. Asanovic, N. Morgan. "Experimental determination of precision requirements for back - propagation training of artificial neural networks", 1st International Conference on Microelectronics for Neural Networks, Dortmund,1990, pag. 9 - 15.

[BAN 94] S. H. Bang, B. J. Sheu. "A Quick Search of optimal solutions for Cellular Neural Networks", WCNN, vol. II, 1994, pag. 549 - 554.

[BAR 97] D. Baratta, G. Marco Bo, D. D. Caviglia, M. Valle, G. Canepa, R. Parenti and C. Penno, "A Hardware Implementation of Hierarchical Neural Networks for Real - Time Quality Control Systems in Industrial Applications", ICANN 1997, pag. 1229 - 1234.

[BEN 90] Y. Bennani, F. Fogelman - Soulie, P. Gallinari. "Text - dependent speacker identification using LVQ", Proc of the Int. Neural Networks Conf., vol. 2, Paris, July 1990, pag. 1087 - 1090.

[BER 96] A. Bermak, D. Martinez. "A Variable - Precision Systolic Architecture for ANN Computation", IEEE Proc. of MicroNeuro, 1996, pag. 347 - 354.

[BLA 92] F. Blayo, M. Verleysen. "Setting initial conditions for the RCE model", Proc. of the First IFIP Working Group - 10.6 Workshop, Grenoble, 1992.

[BLA 95] F. Blayo, A. G. Dugué, N. Maria. "Implementing Radial Basis Functions Neural Networks on the Systolic MANTRA Machine", Proceedings of International Workshop on Artificial Neural Networks, Malaga - Torremolinos, Spain, June 1995, pag. 781 - 788.

[BOA 93] K. A. Boahen, A. G. Andeon. "Design of a bidirectional associative memory chip", Associative Neural Memories: Theory and Implementation, M. Hasson, Ed. New York: Oxford Univ. Press, 1993.

[BOS 92] B. Boser, E. Sackinger, J. Bromley, Y Le Cun, L. D. Jackel. "Hardware requirements for neural network pattern classifiers", IEEE Micro, Feb. 1992, pag. 32 - 40.

[BOU 93] C. Bouton, G. Pages. "Self - organization and a.s. convergence of the one - dimensional Kohonen algorithm with nonuniformly distributed stimuli", Stochastic Processes Applicat., vol. 47, 1993, pag. 249 - 274.

[CHA 94] J. C. F. Chang. "An Efficient Digital VLSI Neural Processor Design for Image Processing", Signal and Image Proc. Institute Report, University of Southern California, 1994.

[CHO 93a] W. Choi, H. Burleson, D. Phatak. "Fixed -point round off error analysis of large feed forward neural networks", Proc. IJCNN '93, 1993, pag. 1947 - 1950.

[CHO 93b] A. N. Choudhary, J. H. Patel, N. Ahuja. "NERTA: A hierarchical and Partionable Architecture for Computer Vision Systems", IEEE Trans. on Parallel and Distributed Systems, vol. 4, nr. 10, Oct. 1993, pag. 1092 - 1104.

[CHU 91] Chung Yoon. "A systolic array exploiting the inherent parallelism of artificial neural networks", International Conference on Parallel Processing, 1991.
[CIM 96] L. Ciminiera, P. Montuschi. "Carry - Save Multiplication Schemes Without Final Addition", IEEE Trans. on Computers, vol. 45, nr. 9, Sept. 1996, pag. 1050 - 1055.
[COM 90] P. Comon, C. Jutten, J. Hérault. "Separation of sources: statistical interpretation", Annexe 2 du rapport DR1 - E3, ESPRIT - BRA NERVES Project, June 1990.
[COM 94] P. Comon, J. L. Voz, M. Verleysen, "Estimation of performance bounds in supervised classification", In M. Verleysen, editor, ESANN: European Symposium on Artificial Neural Networks, Bruxelles, April 1994, pag. 37 - 42.
[COR 96] T. Cornu, P. Ienne, D. Niebur, P. Thiran, M. A. Viredaz. "Design, implementation, and test of a multimodel systolic neural - network accelerator", Scientific Programming, vol. 5, nr.1, Spring 1996, pag. 47 - 61.
[COS 89] M. Cosnard, Y. Robert. "Une introduction a l'algorithmique systolique", Actes de l'Université d'été CIRILLE, Lyon, Juillet, 1989.
[COT 87] M. Cottrel, J. C. Fort. "Etude d'un algorithme d'auto - organisation", Annales de l'Institut Henri Poincare, vol. 23, nr. 1, 1987, pag. 1 - 20.
[COT 88] M. Cottrel. "Stability and attractivity in Associative Memory Networks", Biol. Cyb. 58, 1988, pag. 129 - 139.
[COT 89] M. Cottrel, J. C. Fort. "Aspects théoriques de l'algorithme d'auto - organisation de Kohonen", Annales du groupe CARNAC nr. 2, Lausane, 1989.
[DAD 89] L. Dadda. "On serial - input multipliers for two's complement numbers", IEEE Trans. On Computers, vol. 38, nr. 9, Sept. 1989, pag. 1341 - 1345.
[DAD 96] L. Dadda, V. Piuri. "Pipelined Adders", IEEE Trans. on Computers, vol. 45, nr. 3, March 1996, pag. 348 - 356.
[DIE 87] S. Diederich, M. Opper. "Learning of Correlated patterns in spin - glass network by local learning rules", Physical Review Letters, vol. 58, nr. 9, 1987, pag. 949 - 952.
[DOW 93] E. M. Dowling, Z. Fu, R. S. Drafz. "HARP: An Open Architecture for Parallel Matrix and Signal Processing", IEEE Trans. on Parallel and Distributed Systems, vol. 4, nr. 10, Oct. 1993, pag. 1081 - 1091.
[DUD 73] R. O. Duda, P. E. Hart. "Pattern classification and scene analysis", Wiley and Sons, 1973.
[EDE 82] G. M. Edelman, G. N. Reeke. "Selective networks capable of representative transformation, limited generalizations and associative memory", Proc. Natl. Acad. Sci., vol. 79, March 1982, pag. 2091 - 2095.
[ERW 92] E. Erwin, K. Obermayer, K. Schulten. "Self - organizing maps: Ordering, convergence properties, and energy functions", Biol. Cybern., vol. 67, 1992, pag. 47 - 55.
[FAN 92] W. C. Fang et al. "A VLSI neural processor for image data compression using self - organizing networks", IEEE Trans. on Neural Networks, vol. 3, May 1992, pag. 506 - 518.
[FAT 96] Sherif Kassem Fathy, Mostafa Mohamed Syiam. "A Parallel design and Implementation for Backpropagation Neural Network Using MIMD Architecture", ICNN 1996, pag. 1361 - 1366.
[FEN 96] S. T. J. Fenn, M. Benaissa, D. Taylor. "GF ($2^m$) Multiplication and Division Over the Dual Basis", IEEE Trans. on Computers, vol. 45, nr. 3, March 1996, pag. 319 - 327.
[FIS 97] T. Fischer, W. Eppler, H. Gemmeke, G. Kock and T. Becker. "The SAND Neurochip and Its Embedding in the MIMD System", ICANN 1997, pag. 1234 - 1240.
[FOR 87] J. A. B. Fortes, B. W. Wah. "Systolic arrays: from concepts to implementation", IEEE Computer, July 1987.

[FOR 93] J. C. Fort, G. Pages. "Sur la convergence p.s. sure de l'algorithme de Kohonen Généralisé", Comptes Rendus de l'Académie des Sciences de Paris, vol. 317, 1993, pag. 389 - 394.

[FUK 75] K. Fukushima. "Cognitron: a self organizing multilayered neural network", Biol. Cyb. Nr. 20, 1975, pag. 121 - 136.

[GRA 88] D. H. Graf, W. R. Lalonde. "A neural controller for collision free movement of general robot manipulator", Int. Conf. on Neural Networks, San Diego, July 24 - 27, vol. 1, 1988, pag. 77 - 84.

[GRI 91] M. Griffin, G. Tahara, K. Knorpp, R. Pinkham, B. Riley, "An 11-million transistor neural networks execution engine", Tech. Digest. IEEE Inter. Solid - State Circuits Conference, San Francisco, CA, Feb. 1991, pag. 180 - 181.

[GRO 86] S. Grossberg. "The adaptive Brain", Elsevier, North Holland, Amsterdam, 1986.

[GRO 87] S. Grossberg. "Competitive learning: from Interactive Activation to Adaptive Resonance Theory", Cognitive Science, vol. 11, 1987, pag. 23 - 63.

[GUI 89] M. Guivarch. "Réseaux: techniques de base et apprentissage", 3-eme université d'été CIRILLE, Lyon, 3 - 7 Juillet, 1989.

[GUY 88] I. Guyon. "Réseaux de neurones pour la reconnaissance de formes: architectures et apprentissage", These de Doctorat de l'Université Paris VI, Décembre, 1988.

[GUY 90] I. Guyon. "Neural networks and applications", Internal Report AT & T Bell Labs. September 1990.

[HAM 87] S. E. Hampson, D. J. Volper. "Disjunctive models of boolean category learning", Biol. Cyb. Nr 56, 1987, pag. 121 - 137.

[HAM 91] D. Hammerstrom. "A highly parallel digital architecture for neural network emulation", VLSI for Artificial Intelligence and Neural Networks, J. G. Delgado, Frias, W. Moore, Eds., New York: Plenum, 1991, pag. 357 - 366.

[HEB 49] D. Hebb. "The organisation of behavior", John Wiley and sons, New York, 1949.

[HEC 87] R. Hecht - Nielsen. "Counterpropagation networks", Applied Optics, nr. 26, Dec. 1987, pag. 4976 - 4984.

[HOL 91] J. L. Holt, T. E. Baker. "Back propagation simulation using limited precision calculation", International Joint Conference on Neural Networks, Seattle, Wash., 1991.

[HOL 93] J. Holt, J. N. Hwang. "Finite Precision Error Analysis of Neural Network Hardware Implementations", IEEE Trans. on Computers, vol. 42, 1993, pag. 281 - 290.

[HOL 94] P. W. Hollis, J. J. Paulos. "A Neural Network Learning Algorithm Tailored for VLSI Implementation", IEEE Trans. on Neural Networks, vol. 5, nr. 5, Sept 1994, pag. 748 - 791.

[HOP 82] J. J. Hopfield. "Neural Networks and physical systems with emergent collective computational abilities", Proc. of the National Academy of Science, USA, vol. 79, 1982 pag. 2554 - 2558.

[HUB 77] D. Hubbel, T. Wiesel. "Functional architecture of the macaque monkey visual cortex", Ferrier lecture Proc. Royal Society, London. B. 198, 1977, pag. 1 - 59.

[HUG 92] F. M. Hugen, Z. Houkes. "Systolic arrays for real - time recursive linear least squares parameter estimation", Proceedings of the Workshop on Circuits, Systems and Signal Processing, Houthalen, 1992, pag. 283 - 289.

[IEN 94] P. Ienne, M. A. Viredaz. "Bit - serial multipliers and squarers", IEEE Trans. on Computers, vol. 43, nr. 12, Dec. 1994, pag. 1445 - 1450.

[IEN 95] P. Ienne, M. A. Viredaz. "GENEZ IV: A bit - serial processing element for multimodel neural - network accelerator", Journal VLSI Signal Processing, vol. 9, nr. 3, Apr. 1995, pag. 257 - 273.

[IEN 97] P. Ienne, P. Thiran, N. Vassilas. "Modified Self - Organizing Feature Map Algorithms for Efficient Digital Hardware Implementation", IEEE Trans. on Neural Networks, vol. 8, nr. 2, March 1997, pag. 315 - 330.

[INT 93] Intel Neural Network Group. "Design and Implementation of a recognition accelerator", Proc. of the Canadian Conference on Very Large Scale Integration, 1993.

[JAH 97] A. Jahnke, T. Schönauer, U. Roth, K. Mohraz and H. Klar. "Simulation of Spiking Neural Networks on Different Hardware Platforms", ICANN 1997, pag. 1187 - 1192.

[JEA 94] J. S. N. Jean, J. Wang. "Weight Smoothing to Improve Network Generalization", IEEE Trans. on Neural Networks, vol. 5, nr. 5, Sept. 1994, pag. 752 - 763.

[JOH 92] A. Johannet, L. Personnaz, G. Dreyfus, J. D. Gascuel, M. Weinfield. "Specification and implementation of a digital Hopfield - type associative memory with on - chip training", IEEE Transaction on Neural Networks, vol. 3 (4), July 1992, pag. 529 - 539.

[JOH 93] K. T. Johnson, A. R. Hurson. "General - purpose systolic arrays", IEEE Comp., November 1993, pag. 20 - 31.

[JOH 94] S. R. Johnes, K. Sammut. "Learning in linear systolic neural network engines: Analysis and implementation", IEEE Trans on Neural Networks, vol. 5, nr. 4, 1994, pag. 584 - 593.

[JUT 87] C. Jutten. "Calcul neuro - mimetique et traitement du signal. Analise en composantes indépendantes", These de Doctorat d'etat, Grenoble, Juin 1987.

[JUT 88] C. Jutten, J. Hérault. "Une solution neuromimetique au probleme de séparation de sources", Tratement du Signal, vol. 5, nr. 6, 1988, pag. 389 - 403.

[JUT 90] C. Jutten, A. Guérin, J. Hérault. "Simulation machine and integrated implementation of neural networks", Lecture Notes in Computer Science: Neural Networks, Almeida and Wellekens Eds. Springer Verlag, 1990.

[KAN 93a] V. Kantabutra. "Accelerated Two - level carry - Skip Adders - A Type of Very Fast Adders", IEEE Trans. on Computers, vol. 42, nr. 11, Nov. 1993, pag. 1389 - 1393.

[KAN 93b] V. Kantabutra. "A Recursive Carry - Lookahead / Carry - Select Hybrid Adder", IEEE Trans. on Computers, vol. 42, nr. 12, Dec. 1993, pag. 1495 - 1499.

[KIN 94] F. S. King, P. Saratchandran, N. Sundararajan. "A Theoretical Study of Training Set Parallelism for Back propagation Networks on a Trasputer Array", Proceed. of WCNN, 1994, vol. II, pag. 519 - 524.

[KNO 88] S. C. Knowles, R. F. Wood, J. G. McWriter, J. V. McCanny. "Bit level systolic arrays for the IIR filtering", Int. Conf. on Systolic Arrays, San Diego, May 25 - 27, 1988, pag. 653 - 663.

[KOH 86] T. Kohonen. "Self organisation and associative memory", Springer, Berlin, 1986, 2nd Edition.

[KOH 88a] T. Kohonen. "The "neural" phonetic typewriter", Computer, March 1988, pag. 11 - 22.

[KOH 88b] T. Kohonen. "Self - Organization and Associative Memory", 2nd edition, Berlin, Springer - Verlag, 1988.

[KOH 97] T. Kohonen. "Self - Organizing Maps", Springer - Verlag, Berlin Heidelberg, 1997.

[KOL 97] Pasi Kolinummi, Timo Hämäläinen and Jukka Saarinen. "Mapping of Radial Basis Function Networks to Partial Tree Shape Parallel Neurocomputer", ICANN 1997, pag. 1259 - 1264.

[KON 93] A. König, X. Geng, M. Glesner. "Hardware Implementation of Kohonen's Feature Map by Scalar and SIMD - Array Processors", ICANN 1993, pag. 1046 - 1049.

[KOR 90] I. Koren, O. Zinaty. "Evaluating Elementary Functions in a Numerical Co-Processor Based on Rational Approximations", IEEE Trans. on Computers, vol. 39, nr. 8, Aug. 1990, pag. 1030 - 1037.

[KOR 93] I. Koren. "Computer Arithmetic Algorithms", Englewood Cliffs, New Jersey, Prentice Hall, 1993.
[KOS 87] B. Kosko, "Bidirectional associative memories", BYTE Magazine, Sep. 1987.
[KUN 78] H. T. Kung, C. E. Leiserson. "Algorithm for VLSI processor arrays", Symposium on Sparse Matrix Computations and Their Applications, Knoxville, Nov. 2 - 3, 1978.
[KUN 88a] S. Y. Kung. "VLSI array processors", T. Kailath Ed., Prentice Hall, Englewood Cliffs, New Jersey, 1988.
[KUN 88b] S. Y. Kung, J. N. Hwang. "An algebraic projection for optimal hidden units size and learning rates in backpropagation learning", Proc. IEEE International Conference on Neural Networks, San Diego, vol. 1, 1988, pag. 363 - 370.
[LAW 93] J. C. Lawson, N. Maria, H. J. Smart. "A neuro - computer using sparse matrices", In Euro Micro PDP Workshop, Canari (Spain), January 1993.
[LeC 85] Y. Le Cun. "Une procédure d'apprentissage pour réseau a seuil asymetrique", Actes de COGNITIVA 85, Paris, Juin 1985, pag. 599 - 604.
[LEH 93] C. Lehmann, M. Viredaz, F. Blayo. "A generic systolic array building block for neural networks with on - chip learning", IEEE Trans. on Neural Networks, vol. 4, nr. 3, May 1993, pag. 400 - 407.
[LEN 97] P. Lenders, S. Rajopadhye. "Multirate VLSI Arrays and Their Synthesis", IEEE Trans. on Computers, vol. 46, nr. 5, May 1997, pag. 515 - 529.
[LI 85] G. H. Li, B. W. Wah. "The design of optimal systolic arrays", IEEE Transaction on Computers, vol. 34, nr. 1, 1985, pag. 66 - 67.
[LIP 87] R. Lippmann. "An introduction to computing with neural nets", IEEE ASSP Magazine, April 1987, pag. 4 - 22.
[MAL 94] Q. M. Malluhi, M. A. Bayounmi, T. R. N. Rao. "A parallel Algorithm for Neural Computing", WCNN, vol. II, 1994, pag. 563 - 569.
[MAN 90] R. Mann, S. Haykin. "A parallel implementation of Kohonen feature maps on the Warp systolic computer", Proc. Int. Joint Conf. on Neural Networks, vol. II, Washington, Jan. 1990, pag. 84 - 87.
[MAR 90] P. W. Markstein. "Computation of Elementary Functions on the IBM RISC System/6000 Processor", IBM J. Research and Development, vol. 34, nr. 1, Jan. 1990, pag. 111 - 119.
[MAR 94] N. Maria, A. G. Dugué, F. Blayo. "1D and 2D Systolic Implementations for Radial Basis Function Networks", IEEE Proc. of MicroNeuro, 1994, pag. 34 - 45.
[MAS 92] R. Mason et al. "An hierarchical VLSI neural - network architecture", IEEE J. Solid - State Circuits, vol. 27, Jan. 1992, pag. 106 - 108.
[MAS 97] E. I. El - Masry, H. K. Yang, M. A. Yakout. "Implementations of Artificial Neural Networks Using Current - Mode Pulse Width Modulation Technique", IEEE Trans. on Neural Networks, vol. 8, nr. 3, May 1997, pag. 532 - 548.
[MAU 92] N. Mauduit. "Lneuro 1.0: a piece of hardware LEGO for building neural network systems", IEEE Trans. on Neural Networks, March 1992.
[McC 43] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous system", Bulletin of Mathematical Biophysicis, nr. 5, 1943, pag. 115 - 133.
[McG 87] R. J. Mac Gregor. "Neural and Brain modeling", Neuroscience Series Academic Press Inc. London, 1987.
[McL 97] S. McLoone, G. W. Irwin. "Fast Parallel Off - Line Training of Multilayer Perceptrons", IEEE Transactions on Neural Networks, vol. 8, nr. 3, May 1997, pag. 646 - 652.
[MEL 92] M. S. Melton, T. Phan, D. S. Reeves, D. E. Van den Bout, "The TInMANN VLSI chip", IEEE Trans. Neural Networks, vol. 3, nr. 3, May 1992, pag. 375 - 384.
[MIH 93] I. Z. Mihu. "Some aspects concerning the markovian analysis of overlap systems performance", Acta Universitatis Cininiensis Technical series, vol. X(1), Sibiu, 1993, pag. 52 - 59.

[MIH 95] I. Z. Mihu, V. Tulai. "O implementare hard în tehnică pipeline a reţelelor neuronale feed forward", Acta Universitatis Cininiensis, Seria Tehnică, vol. XXII, Sibiu 1995, pag. 72 - 77.
[MIH 96a] I. Z. Mihu. "Tendinţe în ingineria sistemelor asistată de calculator", Referat doctorat, Universitatea Tehnică Timişoara, 1996.
[MIH 96b] I. Z. Mihu. "Sisteme bazate pe ingineria cunoaşterii în conducerea proceselor", Referat doctorat, Universitatea Tehnică Timişoara, 1996.
[MIH 97] I. Z. Mihu. "Arhitectura sistemelor de calcul. Paralelismul în sistemele microprocesor", JEP 08012/94, Univ. Tehnică Cluj - Napoca, 1997.
[MIN 88] M. Minsky, S. Pappert. "Perceptrons", MIT Press, Expanded Edition, 1988.
[MOR 92] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, J. Beer, "The Ring Array Processor: A multiprocessing peripheral for connexionist applications", Journal of Parallel and Distributed Comput., vol. 14, nr. 3, March 1992, pag. 248 - 259.
[MOR 95] J. M. Moreno, J. Madrenas, S. S. Anselmo, F. Castillo, J. Cabestany. "Digital Hardware Implementation of ROI Incremental Algorithms", Proceedings of International Workshop on Artificial Neural Networks, Malaga - Torremolinos, Spain, June 1995, pag. 761 - 770.
[MUE 93] D. Mueller, D. Hammerstrom. "A neural network systems component", Proc. IEEE Inter. Conf. Neural Networks, vol. 3, San Francisco, CA, Mar. 1993, pag. 1258 - 1264.
[MUL 92] U. A. Müler, B. Bäumle, P. Kohler, A. Gunzinger, W. Guggenbühl. "Achieving supercomputer performance for neural - net simulation with an array of digital signal processors", IEEE Micro, Oct. 1992, pag. 55 - 65.
[MUR 92] P. Murtagh, A. C. Tsoi. "Implementation Issues of Sigmoid Function and its Derivative for VLSI Digital Neural Networks", IEE Proc. - E, vol. 139, nr.3, May 1992.
[MYE 91] D. J. Myers, J. M. Vincent, D. A. Orrey, "HANNIBAL: A VLSI building block for neural networks with on - chip back propagation learning", Proc. of 2nd Int. Conf. Microelectronics for Neural Networks, Munich, 1991, pag. 171 - 181.
[MYO 96] Jean Fréderic Myoupo, David Semé. "A single - layer Systolic Architecture for Back propagation Learning", ICNN 1996, pag. 1329 - 1333.
[NAS 92] N. M. Nasrabadi, C. Y. Choo, "Hopfield Network for Stereo Vision Correspondence", IEEE Trans. on Neural Networks, January 1992.
[NAY 95] D. Naylor, S. Jones, D. Myers. "Backpropagation in Linear Arrays - A performance Analysis and Optimization" IEEE Trans. on Neural Networks, vol. 6, nr. 3, May 1995, pag. 583 - 594.
[OJA 82] E. Oja. "A simplified neuron model as a principal component analyzer", J. Math. Biol. Nr. 15, 1982, pag. 267 - 273.
[ORR 91] D. A. Orrey, D. J. Myers, J. M. Vincent, "A high performance digital processor for implementing large artificial neural networks", Proc. IEEE Custom Integrated Circuits Conference, San Diego, May 1991, pag. 16.3.1. - 16.3.4.
[OKL 96] V. G. Oklobdzija, D. Villeger, S. S. Liu. "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", IEEE Trans. on Computers, vol. 45, nr. 3, March 1996, pag. 294 - 306.
[PAR 96] R. Parisi, E. D. Di Claudio, A. Rapagnetta, G. Orlandi. "Recursive Least Squares Approach to Learning in Recurrent Neural Networks", ICNN 1996, pag. 1350 - 1354.
[PAT 96] Minesh I. Patel, N. Ranganathan. "PANTHER: A Parallel Neuro Systolic Architecture for Real - Time Processing", ICNN 1996, pag. 1006 - 1011.
[PER 86] L. Personnaz, I. Guyon, G. Dreyfus. "Collective computational proprietes of neural networks", Physical Review A, vol. 34, nr.5, Nov. 1986, pag. 4217 - 4228.
[PER 87] L. Personnaz, I. Guyon, G. Dreyfus. "High order neural networks: information storage without errors", Europhysics letters, nr. 4(8), Oct. 1987, pag. 863 - 867.

[PER 97] A. Pérez. - Uribe and E. Sanchez, "FPGA Implementation of a Network of Neurolike Adaptive Elements", ICANN 1997, pag. 1246 - 1252.

[PIU 94] V. Piuri, S. Bettola. "High - performance digital Neural Networks: the use of redundant binary representation for concurrent error detection", WCNN, vol. II, 1994, pag. 531 - 535.

[POR 91] W. Porter. "Neuronic Arrays: Design and Performance", Proceedings of the Second Workshop on Neural Networks: Academic / Industrial / NASA / Defense, WNN - AIND 91, pag. 119 - 128.

[PRZ 91] K. W. Przytula, W. M. Lin, V. K. P. Kumar, "Partitioned implementation of neural networks on mesh connected array processors", VLSI Signal Processing IV, IEEE Pres.: New York, Aug. 1991, pag. 106 - 115.

[PUL 96] Ville Pulkki, Taneli Harju. "An implementation of the Self - Organizing Map on the CNAPS Neurocomputer", ICNN 1996, pag. 1345 - 1349.

[QUI 84] P. Quinton. "Automatic synthesis of systolic arrays from Uniform Recurrent Equations", Proc. of 11th Annual Symposium on Computer Architecture, Ann Arbor, Michigan, 1984, pag. 208 - 214.

[QUI 89] P. Quinton, Y. Robert. "Algorithmes et architectures systoliques", Masson, Paris, 1989.

[RAM 91] U. Ramacher. "Guidelines to VLSI Design of Neural Networks", Silicon Archs. For Neural Nets, Eds. M. Sami, J. Calzadilla - Daguerre, Elsevier Science Publisher B. V. (North - Holland), 1991.

[RAM 91a] U. Ramacher, J. Beichter, N. Brűls. "Architecture of a general - purpose neural signal processor", Proc. IEEE / INNS Inter. Joint Conf. Neural Networks, vol. 1, Seattle, WA, July 1991, pag. 443 - 446.

[RAM 91b] U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Brűls, U. Hachmann, M. Wesseling. "Design of a 1st generation neurocomputer", in VLSI Design of Neural Networks (U. Ramacher, U. Rűcket, eds.), Kluwer Academic Publisher, Boston, 1991, pag. 271 - 310.

[RAM 91c] U. Ramacher, B. Schűrmann. "Unified description of neural algorithms for time - independent pattern recognition", in VLSI Design of Neural Networks (U. Ramacher, U. Rűcket, eds.), Kluwer Academic Publisher, Boston, 1991, pag. 255 - 270.

[RAM 92] U. Ramacher. "SYNAPSE - A neurocomputer that synthesizes neural algorithms on a parallel systolic engine". Journal of Parallel and Distributed Comput., vol. 14, nr. 3, Mar. 1992, pag. 306 - 318.

[RAM 93] U. Ramacher, W. Raab, J. Anlauf, J. Beichter, U. Hachmann, N. Brűls, M. Weßeling, E. Sicheneder. "Multiprocessor and Memory Architecture of the Neurocomputer SYNAPSE - 1", ICANN 1993, pag. 1034 - 1038.

[REZ 97] S. M. Rezaul Hasan, N. K. Siong. "A Parallel Processing VLSI BAM Engine", IEEE Trans. on Neural Networks, vol. 8, nr. 2, Mar. 1997, pag. 424 - 436.

[REY 91] L. M. Reyneri, E. Philippi, "An analysis on the performance of silicon implementations of back propagation algorithms for artificial neural networks", IEEE Trans. Comput., vol. 40, nr. 12, Dec. 1991, pag. 1380 - 1389.

[ROB 92] M. E. Robinson, H. Yoneda, E. Sanchez - Sinencio, "A Modular CMOS Design of a Hamming Network", IEEE Trans. on Neural Network, vol. 3, No. 3, May 1992.

[ROS 58] F. Rosenblatt, "The Perceptron; a probabilistic model for information storage and organization in the brain", Psychological Review, Nr. 65, 1958, pag. 386 - 408.

[ROS 97] M. Rossmann, A. Bűhlmeier, G. Manteuffel and K. Goser. "Short - and Long - Term Dynamics in a Stochastic Pulse Stream Neuron Implemented in FPGA", ICANN 1997, pag. 1241 - 1246.

[ROT 97] Ulrich Roth, Axel Jahnke and Heinrich Klar. "On - line Hebbian Learning for Spiking Neurons: Architecture of the Wecht - Unit of NESPINN", ICANN 1997, pag. 1217 - 1222.

[RUE 90] P. F. Ruedi, E. Sorouchyari. "Commande d'un bras de robot redundant par reseaux de neurones", Centre Suisse d'Electronique et de Microtechnique, Rapport technique Nr. 354, Mai 1990.

[RUE 93] S. Rueping, U. Rueckert, K. Goser. "Hardware Design for Self - Organzing Feature Maps with Binary Input Vectors", Proceedings of the IWANN '93, Sigtes, Spain, 1993, pag. 488 - 493.

[RUE 94] S. Rueping, U. Rueckert, K. Goser. " A Chip for Selforganizing Feature Maps", Proc. of the 4th Int. Conf. of microelectronics for Neural Networks and Fuzzy Systems, IEEE Computer Society Press, 1994, pag. 26 - 33.

[RUE 95] S. Rueping, U. Rueckert, K. Goser. "A Chip for Selforganizing Feature Maps", IEEE MICRO, vol. 15, Nr. 3, June 1995, pag. 57 - 59.

[RUI 95] V. Ruiz de Angulo, C. Torras. "On-Line Learning with Minimal Degradation in Feedforward Networks", IEEE Trans. on Neural Netwoks, vol. 6, Nr. 3, May 1995, pag. 657 - 668.

[RUM 86] D. E. Rumelhart, J. McClelland. "Parallel distributed processing, vol. 1: Foundations, vol. 2: Psychological and Biological models", Institute for Cognitive Science, Bradford Book, The MIT press, J. A. Feldman, P. J. Hayes, D. E. Rumelhart Eds., 1986.

[RUM 90] D. E. Rumelhart. "Brain Style Computation: Learning and Generalization", Introduction to Neural and Electronic Networks, New York, Academic Press, 1990.

[RUP 96] S. Rűping, U. Rűckert. "A Scalable Processor Array for Self - Organizing Feature Maps", IEEE Proc. of MicroNeuro, 1996, pag. 285 - 291.

[SAC 96] E. Sackinger, H. P. Graf. "A board System for High - Speed Image Analysis and Neural Networks", IEEE Trans. on Neural Networks, vol. 7, nr. 1, Jan. 1996, pag. 214 - 221.

[SAE 94] R. Saeks, K. Priddy, K. Schneider, S. Stowell. "On the Design of an MIMD Neural Network Processor", WCNN, vol. II, 1994, pag. 590 - 595.

[SAM 91] K. Sammut, S. R. Jones. "Implementing the adaptive resource theory in linear array", Proc. of 2nd Int. Conference Microelectronics for Neural Networks, Munich, 1991, pag. 69 - 76.

[SAN 92] E. Sanchez - Sinencio, C. Lau (Eds.). "Artificial Neural Networks: Paradigms, Applications and Hardware Implementations", IEEE Press: Piscataway, 1992.

[SAT 92] S. Satyanarayan et al. "A reconfigurable VLSI neural network", IEEE J. Solid - State Circuits, vol. 27, Jan. 1992, pag. 67 - 81.

[SCH 92] I. Scherson, D. Kramer, B. Alleyne, "Bit - parallel arithmetic in massively - parallel associative processor", IEEE Trans. on Computers, vol. 41 (10), 1992, pag. 1201 - 1210.

[SER 93] M. J. Serrano, B. Parhani. "Optimal Architectures and Algorithms for Mesh - Connected Parallel Computers with Separable Row / Column Buses", IEEE Trans. on Parallel and Distributed Systems, vol. 4, nr. 10, Oct. 1993, pag. 1073 - 1080.

[SHE 95] Bing J. Shen, J. Choi. "Neural Information Processing and VLSI", Kluwer Academic Publishers, Boston, 1995.

[SIG 91] A. Siggelkow, J. Nijhuis, at. al. "Influence of Hardware Characteristics on the Performance of a Neural System", Artificial Neural Networks, vol. 1, North - Holland, Eds. Kohonen, K. Mäkisaro, O. Simula, J. Kangas, Amsterdam, 1991.

[SPE 93] H. Speckmann, P. Thole, W. Rosenstiel. "COKOS: A COprocessors for Kohonen's Selforganizing Map, ICANN, 1993, pag. 1040 - 1044.

[SPE 94] H. Speckmann, P. Thole, M. Bogdan, W. Rosenstiel. "Coprocessors for special neural networks KOKOS and KOBOLD", WCNN, vol. II, 1994, pag. 612 - 617.

[STE 97] M. Stevenson. "The Effects of Limited - Precision Weights on the Threshold Adaline", IEEE Trans. on Neural Networks, vol. 8, nr. 3, May 1997, pag. 549 - 552.

[STR 82] N. R. Strader, V. T. Ryhne. "A canonical bit - sequential multiplier", IEEE Trans. on Computers, vol. C - 31, nr. 8, Aug. 1982, pag. 791 - 795.

[STR 95] A. Strey, N. Avellana, R. Holgado, J. A. Fernandez, R. Capillas, E. Valderrama. "A Massively Parallel Neurocomputer with a Reconfigurable Arithmetical Unit", Proceedings of International Workshop on Artificial Neural Networks, Malaga - Torremolinos, Spain, June 1995, pag. 800 - 806.

[SUN 93] M. H. Sunwoo, J. K. Aggarwal. "A Sliding Memory Plane Array Processor", IEEE Trans. on Parallel and Distributed Systems, vol. 4, nr. 6, Jun. 1993, pag. 601 - 612.

[TAM 97] S. Tamura, M. Tateishi. "Capabilities of a Four - Layered Feedforward Neural Network: Four Layers Versus Three", IEEE Trans. on Neural Networks, vol. 8, nr. 2, Mar. 1997, pag. 251 - 255.

[THI 94] P. Thiran, M. Hasler. "Self - organization of a one - dimensional Kohonen network with quantized weights and inputs", Neural Networks, vol. 7, nr. 9, 1994, pag. 1427 - 1439.

[THI 95] P. Thissen, M. Verleysen, J. D. Legat. "An associative processor architecture for pattern recognition", Proc. of the ProRISC / IEEE Benelux Workshop an Circuits, Systems and Signal Processing, Martie 1995, pag. 309 - 315.

[TRY 89] V. Tryba, S. Metzen, C. Goser. "Designing basic integrated circuits by self - organizing feature maps", Proc. of Neuro Nimes 89, Int. Workshop, Nimes, Nov. 13 - 16, 1989, pag. 225 - 234.

[TSA 95] J. C. Tsay, P. Y. Chang. "Some New Designs of 2-D Array for Matrix Multiplication and Transitive Closure", IEEE Transactions on Parallel and Distributed Systems, vol. 6, nr. 4, April 1995, pag. 351 - 362.

[UCH 92] K. Uchimura, O. Saito, Y. Amemiya. "An 8G connections - per - second 54mW digital neural network chip with low - power chain - reaction architecture", Tech. Digest IEEE Inter. Solid - State Circuits Conference, San Francisco, Feb. 1992, pag 134 - 135.

[VAS 96a] Nikolaos Vassilas, Patrick Thiran, Paolo Ienne. "On Modifications of Kohonen's Feature map Algorithm for an Efficient Parallel Implementation", ICNN 1996, pag. 932 - 937.

[VAS 96b] S. Vassiliadis, S. Cotofana, K. Bertels. "2 - 1 Addition and Related Arithmetic Operations with Threshold Logic", IEEE Trans. on Computers, vol. 45, nr. 9, Sept. 1996, pag. 1062 - 1067.

[VIN 92] J. M. Vincent. "Finite wordlength, integer arithmetic multi - layer perceptron modeling for hardware realization", Neural Networks for Vision, Speech and Natural Language, R. Linggard, D. J. Myers and C. Nightingale - editors, London, Chapman & Hall, 1992, pag. 293 - 311.

[VIR 92] M. A. Viredaz, C. Lehmann, F. Blayo, P. Ienne. "MANTRA: A multi - model neural - network computer", In Proc. of the 3rd International Workshop on VLSI for Neural Networks and Artificial Intelligence, Oxford, UK, September 1992.

[VIR 93] M. A. Viredaz. "MANTRA I: An SIMD processor array for neural computation", In Proceedings of Euro - ARCH '93, Munich, Germany, October 1993.

[WAI 89] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K. Lang. "Phoneme recognition using time - delay neural networks", IEEE Trans on ASSp, nr. 37, mar. 1989, pag. 328 - 339.

[WAT 93] T. Watanabe, K. Kimura, M. Aoki, T. Sakata, K. Ito. "A single 1,5 V digital chip for $10^6$ synapse neural network", IEEE Trans. Neural Networks, vol. 4, nr. 3, May 1993, pag. 387 - 393.

[WAW 93] J. Wawrzynek, K. Asanovic, N. Morgan. "The design of a neuromicroprocessor", IEEE Trans. Neural Networks, vol. 4, nr. 3, May 1993, pag. 394 - 399.

[WEI 89] M. Weinfield. "A fully digital integrated CMOS Hopfiled network including the learning algorithm", VLSI for Artificial Inteligence, Delgado - Frias & Moore Eds., Kluwer Academic Publishers, 1989.

[WER 88] P. J. Werbos. "Generalization of Back - Propagation with application to Recurrent Gaz Market Model", Neural Networks, vol. 1, 1988, pag. 339 - 356.

[WID 60] B. Widrow, M. E. Hoff. "Adaptive switching circuits", 1960 IRE WESCON Convention Record, New York: IRE, 1960, pag. 96 - 104.

[WID 85] B. Widrow, S. D. Stearns. "Adaptive Signal Processing", Prentice Hall Signal Processing Series, A. V. Oppenheim Ed., Englewood Cliff, New Jersey, 1985.

[WRZ 96] A. Wrzyszcz, D. Milford, E. L. Dagless. "A New Approach to Fixed - Coefficient Inner Product Computation Over Finite Rings", IEEE Trans. on Computers, vol. 45, nr. 12, Dec. 1996, pag. 1345 - 1355.

[YAS 90] M. Yasunaga, N. Masuda, M. Yagyu, M. Asai, M. Yamada, A. Masaki. "Design, fabrication and evaluation of a 5-inch wafer scale neural network VLSI composed of 576 digital neurons", Proc. IEEE / INNS Inter. Joint Conf. Neural Networks, vol. 2, June 1990, pag. 527 - 535.

[ZHA 94] D. Zhang, M. I. Elmasry. "High Performance Compressor Building Blocks for Digital N. N. Implementation", WCNN, vol. II, 1994, pag. 607 - 611.

[ZHA 96] M Zhang, S. Vassiliadis, J. G. Delgado - Frias. "Sigmoid Generators for Neural Computing Using Piecewise Approximations", IEEE Trans. on Computers, vol. 45, nr. 9, Sep. 1996, pag. 1045 - 1049.

[ZHA 97] C. X. Zhang, D. A. Mlynski. "Mapping and Hierarchical Self - Organizing Neural Networks for VLSI Placement", IEEE Trans. on Neural Networks, vol. 8, nr. 2, Mar. 1997, pag. 299 - 314.

[ZUR 92] J. M. Zurada. "Introduction to Artificial Neural Systems", West Publishing Company, St. Paul, 1992.

## CURRICULUM VITAE


| | |
|---|---|
| Nume | : MIHU |
| Prenume | : IOAN |
| Data naşterii | : 14 Decembrie 1954 |
| Locul naşterii | : sat DEAL, jud. ALBA, ROMÂNIA |
| Starea civilă | : căsătorit |
| Copii | : 2 |
| Adresa | : str. LUDOŞ, Nr. 3, Bl. 51, Ap. 45, 2400, SIBIU, ROMÂNIA, Tel. 481006, E-mail: mihuz@cs.sibiu.ro |


**Studii:**

| | |
|---|---|
| 1961 – 1969, | Şcoala Generală, sat DEAL, jud. ALBA |
| 1969 – 1974, | Liceul Industrial Energetic, Sibiu |
| 1976 – 1981, | Facultatea de Electrotehnică, specializarea Automatizări şi Calculatoare, Institutul Politehnic "Traian Vuia" Timişoara. |
| 1992 | Admis la doctorat în domeniul "Sisteme Automate" |
| 1998 | Absolvirea doctoraturii în "Sisteme Automate" (V. T. Timişoara) |

**Experienţa profesională:**

| | |
|---|---|
| 1981 – 1990, | inginer, Întreprinderea pentru Întreţinerea şi Repararea Utilajelor de Calcul Bucureşti, Filiala Sibiu. |
| 1990 – prezent, | şef lucrări, Universitatea "Lucian Blaga" din Sibiu, Facultatea de Inginerie, Catedra de Calculatoare şi Automatică (Discipline: Calculatoare Numerice, Structura Sistemelor de Calcul, Arhitecturi Paralele). |

**Activităţi manageriale:**

| | |
|---|---|
| 1992 – 1993, | secretar ştiinţific al Consiliului Facultăţii de Inginerie |
| 1992 – prezent, | membru al Consiliului Facultăţii de Inginerie |
| 1996 – prezent, | coordonator local Program TEMPUS S_JEP 11005/96 |

**Activitate ştiinţifică şi didactică:**

- 14 lucrări ştiinţifice publicate
- 2 cărţi publicate în domeniul arhitecturii sistemelor de calcul

**Limbi de circulaţie cunoscute:**

- engleza;
- franceza.

**Domenii de interes:**

- arhitecturi / procesări paralele
- reţele neuronale