Neurale Netwerken

Als je alleen een hamer hebt zie je elk probleem als een spijker.
— Maslow

Inhoudsopgave

Woord vooraf

Deze syllabus biedt een beperkte inleiding in het onderwerp *neurale netwerken*. Het materiaal is bestemd voor studenten alfa-informatica, richting kunstmatige intelligentie en natuurlijke taalverwerking. Dit houdt in dat het niveau van de tekst en de veronderstelde achtergrondkennis zouden moeten passen bij studenten alfa-informatica.

Deze syllabus is **niet** bedoeld als inleiding voor het toepassen van neurale netwerken in de kunstmatige intelligentie en natuurlijke taalverwerking. Het is in die zin geen extensie. Wel wordt op een aantal plaatsen ernstige twijfels uitgesproken over de aanpak in de traditionele kunstmatige intelligente (AI) en de computationele verwerking van natuurlijke taal (CL). Neurale netwerken zijn meer dan een instrument. De studie naar neurale netwerken behelst een visie tegenover onderwerpen als intelligent gedrag, bewustzijn en taal die fundamenteel afwijkt van die in de traditionele AI en CL. Ik hoop de student hier begrip voor bij te brengen.

Opzet van de syllabus

De onderwerpen die aan bod komen vallen uiteen in vijf hoofdgroepen:

- Feedforward netwerken en backpropagation, inclusief enkele extensies zoals Elman's recurrente netwerken en Miikkulainen's FGREP. (Hoewel FGREP is gebaseerd op backpropagation gaat het eigenlijk om het concept *neuraal lexicon*, iets heel anders.)
- Relaxatie in **echte** recurrente netwerken zoals het Hopfield netwerk en de Boltzmann-machine.
- Zelf-ordening in topografische kaarten, *competitive learning*.
- Genetische algoritmen.
- Grounding.

De nadruk ligt op het eerste onderwerp, alhoewel dit zeker niet het meest belangrijke onderwerp is. Maar in een inleidende tekst lijkt deze grote aandacht haast onvermijdelijk. Backpropagation netwerken zijn relatief simpel en worden op dit moment het meest toegepast in commerciële toepassingen. Het grote potentieel dat neurale technieken biedt wordt in dit soort netwerken niet benut. Voor een meer diepgaande vervolgstudie raad ik aan feedforward netwerken links te laten liggen.

Aanvullende literatuur

Deze syllabus dient met tenminste twee teksten te worden aangevuld.

1. Voor de theorie over backpropagation kan gekozen worden uit twee teksten:
   - [RHW86], tot aan het kopje *The XOR Problem*, pagina 318 tot en met 330.
   - Hoofdstuk 5 uit [MR88], tot aan het kopje *Implementation*, pagina 121 tot en met 137.

   De tweede tekst is duidelijker, meer breedvoerig. Vooral *gradient descent* wordt goed uitgelegd. De eerste tekst bevat een formele afleiding die in de tweede tekst ontbreekt. (In de tweede tekst staan verwijzingen zoals PDP:2 en PDP:8. Hiermee worden hoofdstuk 2 en hoofdstuk 8 van [RMtPRG86] bedoeld. PDP:8 is de eerste tekst.)
2. De tekst van Diane Law [Law94] is onmisbaar bij het hoofdstuk over *grounding*. Eventueel kan daarnaast [LM94] gebruikt worden dat een aantal ideeën uit het eerste artikel demonstreert aan de hand van experimenten.

Het gebruik van [MRH86] kan een aardige aanvulling zijn op het eerste hoofdstuk uit deze syllabus. Deze tekst illustreert het karakter van *Parallel Distributed Processing* (PDP). Er worden enkele voorbeelden besproken die demonstreren hoe de PDP-aanpak afwijkt van de traditionele symbolische technieken. Het is een goede inleiding om 'de smaak te pakken' te krijgen.

Verder is het raadzaam naast deze syllabus een of meer studieboeken te raadplegen. Achter in de syllabus worden enkele suggesties gedaan.

Practica

Er is practicum-materiaal beschikbaar (programma's en documentatie) voor[1]:

- Inleiding en feedforward netwerken (één practicum voor twee hoofdstukken).
- Backpropagation.
- Topografische kaarten.
- FGREP (Automatisch lexicon).
- Hopfield netwerken.

Voor genetische algoritmen ontbreekt helaas een practicum.

Bij de eerste twee practica wordt gebruik gemaakt van het programma `gnuplot`. Ook voor dit programma is een (zeer beknopte) handleiding aanwezig.

Alle programma's zijn zowel voor Unix (of Linux) als MS-DOS beschikbaar. Onder Unix dient voor de practica Topografische kaarten en FGREP een PostScript printer of PostScript viewer (bijvoorbeeld `ghostview`) aanwezig te zijn. Voor de MS-DOS versies kan volstaan worden met een VGA monitor.

1. Er is een practicum *Adaptive Resonance Theory*, ookal ontbreekt dit thema in de definitieve versie van de syllabus.

Hoofdstuk 1

Inleiding

Ondanks hun enorme rekenkracht zijn moderne computers niet of nauwelijks in staat tot het uitvoeren van taken die een mens (of een muis, of een mier) met gemak uitvoert. Nog niet zo lang geleden dacht men dat intelligentie los staat van de hardware (hersenen, chips in een computer), en dat elke verwerking van informatie is te definiëren door een formele symbolische procedure. De hedendaagse Von Neumann-computer zou in staat moeten zijn om al die procedures te gebruiken om uiteindelijk gedrag op een hoog niveau te bereiken. Deze opvatting blijkt onjuist te zijn. Vandaar dat men de laatste decennia meer en meer de aandacht richt op bestaande goed werkende intelligente systemen: de biologische hersenen.

1.1 Neuro-Informatica

Neurale netwerken kunnen ontwikkeld worden met het doel de werking van biologische hersenen zo nauwkeurig mogelijk te imiteren. Aan de andere kant bestaan er neurale architecturen die slechts in de verte geïnspireerd lijken op echte hersenen. De studie naar neurale netwerken vormt een zeer breed vakgebied. Onderzoekers die zich er mee bezighouden komen uit zeer uiteenlopende disciplines, zoals neurofysiologie, psychologie, wiskunde, theoretische en technische natuurkunde, kunstmatige intelligentie (A.I.) en kunstmatig leven (*a*-life). Het gevolg is dat de manieren waarop het vak benaderd wordt nogal uiteen lopen. De doelstellingen worden vaker door het thuisvak van de onderzoeker bepaald dan door neuro-informatica als geheel, waardoor vaak voorbij wordt gegaan aan de fundamentele eigenschappen van neurale netwerken. Dit probleem speelt vooral bij onderzoekers afkomstig uit de A.I., die een dubbele handicap hebben:

- De A.I. maakt intensief gebruik van de traditionele computer. De hele manier van denken is gevormd door de formeel-symbolische opvatting. Rumelhart en McClelland [RM86, p. 127] verwoorden het zo:

  > ...certain aspects of cognitive theory have been too strongly influenced by the discrete, sequential algorithms available for expression in most current high-level languages.

- De benadering in de A.I. is *top down*. De hoogste intelligente processen in mensen worden gemodelleerd, zoals redeneren en taalverwerking. Een zelfde aanpak met neurale netwerken leidt tot systemen die even dom zijn als hun symbolische tegenhanger, en minder goed. De kracht van neurale netwerken zit in de *bottom up* benadering. Begin met het ontwikkelen van een systeem dat het verstand van een bloedzuiger bezit, en probeer dit systeem te evolueren tot een hoge vorm van intelligentie. Onderzoekers uit de *a*-life zijn hier in het voordeel.

1.2 Eigenschappen van neurale netwerken

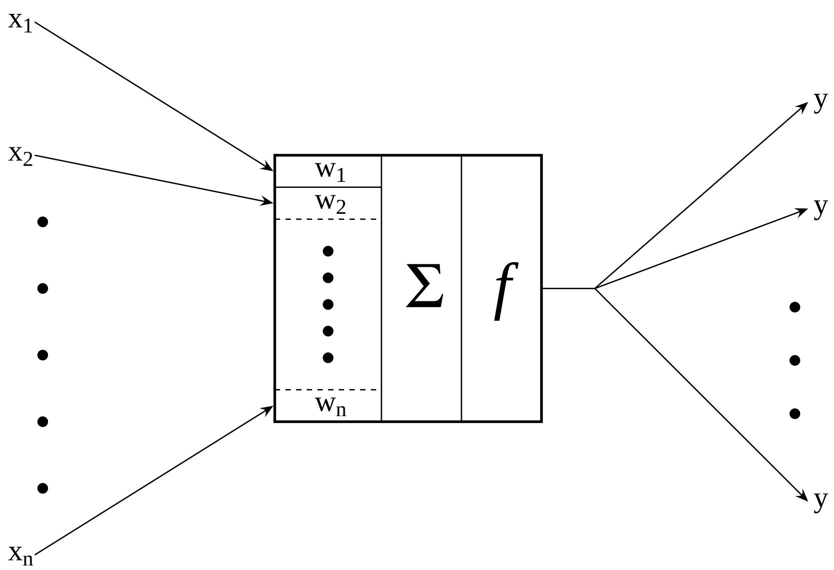
Wat zijn nu de eigenschappen van neurale netwerken waardoor ze zo anders zijn dan de traditionele computer?

- Emergente eigenschappen: het geheel is meer dan de som van de delen. De eigenschappen van een netwerk zijn niet te ontleden in eigenschappen van individuele neuronen. Gedrag dat in symbolische systemen bestuurd wordt door de 'baas' van een hiërarchie blijkt zich in neurale netwerken voor te doen door interactie van 'domme' elementen, zonder dat een *supervisor* of coördinator aanwezig is.
- Een neuraal netwerk werkt parallel. Alle neuronen voeren tegelijk hun bewerkingen uit, waardoor ondanks de traagheid van individuele neuronen complexe bewerking zeer snel uitgevoerd worden. (Dit voordeel gaat verloren wanneer een neuraal net geïmplementeerd wordt als simulatie in een traditionele computer. Zulke simulaties zijn meestal beperkt tot netwerken van enkele duizenden neuronen.)
- Neurale netwerken kunnen aanzienlijke hoeveelheden informatie met complexe onderlinge relaties op een analoge (niet symbolische) wijze integreren. Formele beschrijvingen zijn overbodig, als ze al mogelijk zouden zijn.
- Een neuraal netwerk bevat geen algoritmen en declaratieve informatie voor het oplossen van problemen. Het is zelfs niet correct een neuraal netwerk te zien als een 'probleemoplosser'. Een netwerk verwerft door ervaring bepaalde vaardigheden waarmee het goed kan functioneren in zijn omgeving. Het leert zich aan te passen.
- Neurale netwerken zijn goed bestand tegen ruis. Het presteert nog goed wanneer de invoer kleine verstoringen bevat. Bij grotere afwijkingen verslechteren de prestaties van het netwerk gelijdelijk. Deze eigenschap wordt *graceful degradation* genoemd.

Meer over de karakteristieken van neurale netwerken is te vinden in [MRH86].

1.3 Niveaus in neurale netwerken

De studie van neurale netwerken heeft drie niveaus die hieronder kort worden omschreven:



Figuur 1.1: De unit: een kunstmatig neuron

De neuron Welke eigenschappen heeft een neuron? Hoe wordt de invoer omgezet in een uitvoer? Hoe geeft een neuron zijn uitvoer door aan een volgend neuron?

Het enkelvoudig netwerk Hoe kunnen meerdere neuronen zo worden georganiseerd dat een bruikbaar netwerk ontstaat? Hoe kan zo'n netwerk een bepaalde taak leren?

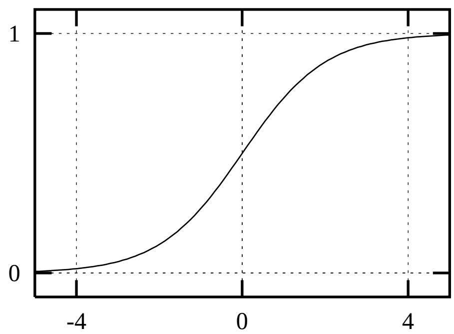
De rest van de syllabus zal voornamelijk vragen op dit niveau behandelen.

Complexe systemen Hoe kunnen enkelvoudige netwerken geïntegreerd worden in een complex systeem dat in staat is tot hoger intelligent gedrag?

1.4 De biologisch neuron en de kunstmatige unit

Biologische neuronen zijn zeer gevarieerd in bouw en complex in hun werking. Zie bijvoorbeeld [CA86] voor een korte inleiding. Een uitgebreide inleiding is te vinden in [Kal92]. Modellen die het gedrag van een biologisch neuron nauwkeurig simuleren zijn eveneens niet eenvoudig. Zie [MK95] voor een voorbeeld.

In kunstmatige neurale netwerken worden vereenvoudigde modellen van neuronen gebruikt. Deze worden vaak units genoemd om aan te geven dat het geen echte neuronen zijn. Figuur 1.1 laat een model zien van zo'n unit. De unit zet een invoervector $\mathbf{x}$, bestaande uit de signalen $x_1 \cdots x_n$ om in een uitvoersignaal y. De signalen x_i worden vermenigvuldigd met de bijbehorende gewichten w_i, die als equivalent van de sterkte van een biologische synaps gezien kunnen worden. De produkten worden bij elkaar opgeteld; de som wordt de *netinput* genoemd. Van de netinput wordt vaak een drempelwaarde afgetrokken, de zogenaamde *bias*, en tot slot zet een functie de waarde om in een uitvoersignaal.



Figuur 1.2: $f(x) = 1/(1+e^{-x})$

In plaats van een bias wordt vaak een extra invoersignaal $x_0 = 1$ en een gewicht w_0 toegevoegd. w_0 functioneert dan als bias (om precies te zijn: de bias keer -1).

De uitvoerfunctie kan verschillende vormen hebben, zoals lineair, een stapfunctie, of een sigmoïde functie, zoals in figuur 1.2. Over het algemeen is het een *squashing*-functie die tot doel heeft de uitvoer terug te brengen tot een waarde tussen een minimum en een maximum.

In het kort komt de werking van een unit op het volgende neer: Als de netinput onder de bias-waarde blijft is de uitvoer van de unit nul (of klein), voor grotere netinput-waarden is de uitvoer maximaal (of groot).

Formeel kan de werking van een unit beschreven worden door vergelijking 1.1.

$$y = f(\sum_i x_i w_i - bias) \tag{1.1}$$

Het hier beschreven model is het meest gangbare, voornamelijk in simulaties van neurale netwerken op seriële computers, maar er worden zeer veel variaties gebruikt.

1.5 Hardware implementatie vs computer-simulatie

Het hierboven beschreven model wordt veel gebruikt bij simulaties van neurale netwerken op seriële computers. Een belangrijk kenmerk van de digitale seriële computer is de stap voor stap verwerking. Een simulatie-programma biedt een unit een invoer aan, de uitvoer van de unit wordt berekend en daarna gebruikt voor verdere verwerking. Daarna begint een nieuwe cyclus met de aanbieding van een nieuw invoerpatroon.

Een echt neuron werkt analoog en continu. Het neuron reageert continu op een aantal variërende invoersignalen. Vergelijking 1.2 is een voorbeeld van de beschrijving van het verloop van de activatie van een unit in de tijd.

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -decay(y - rest) + (max - y) f_E(E) - (y - min) f_I(I) \tag{1.2}$$

Hierin is *decay* de snelheid waarmee de activatie uit zichzelf terugvalt naar de rustpotentiaal, die wordt weergegeven door *rest*. *min* en *max* zijn de minimum en maximum activatiewaarden. f_E en f_I zijn twee functies die hun invoer schalen tussen 0 en 1, zoals de functie in figuur 1.2. E en I zijn de netinput van verbindingen met resp. positieve en negatieve gewichten.

Voor een seriële computer is dit model een veel zwaardere belasting dan het eerder beschreven discrete model. Meestal kan een proces in zo'n vorm gegoten worden dat een discreet model volstaat, maar dit is niet altijd mogelijk.

Voor professionele toepassingen is het streven gebruik te maken van speciale neurale hardware. Hierin worden de netwerken uit analoge componenten opgebouwd, die optimaal gebruik kunnen maken van het continumodel.

Hoofdstuk 2

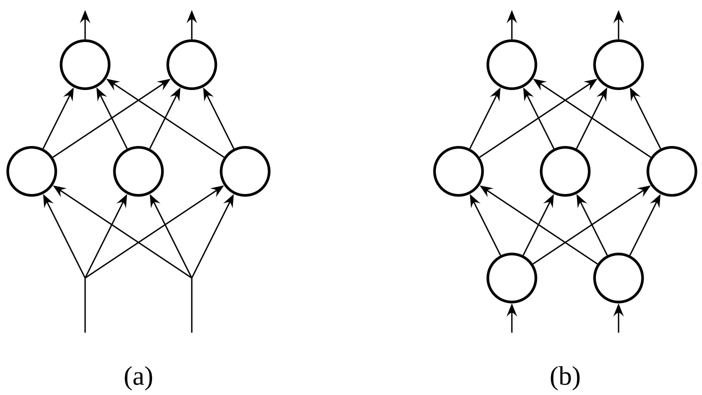
Feedforward netwerken

Het meest eenvoudige netwerk is het feedforward netwerk. Dit netwerk dankt zijn naam aan het gegeven dat de units onderling zo zijn verbonden dat de informatie in één richting stroomt. Het netwerk bevat geen lussen.

Vaak zijn de units in lagen georganiseerd, waarbij de units in één laag alleen verbindingen hebben naar de units in de eerstvolgende laag. Er is een invoerlaag, die invoer ontvangt uit de omgeving, eventueel één of meer verborgen lagen, en een uitvoerlaag, waar de resultaten aan afgelezen kunnen worden.

Figuur 2.1 geeft twee afbeeldingen van zo'n netwerk, in beide gevallen gaat het om hetzelfde netwerk. Er is een invoerlaag, één verborgen laag, en een uitvoerlaag. In figuur 2.1(a) zijn de invoerunits niet getekend. De invoerunits doen eigenlijk niets. Ze dubliceren slechts hun invoer voor elke verbinding naar de verborgen units. Je kunt ze opvatten als de uitvoerunits van één of meerdere andere netwerken, of als sensorische zenuwen. Vaak is het gemakkelijk om de invoerunits wel te tekenen, zoals in figuur 2.1(b), en ze te gebruiken bij de berekeningen in simulaties van netwerken.

De twee afbeelding illustreren tevens een veel voorkomende verwarring over de naamgeving van gelaagde netwerken. Sommigen noemen het netwerk in figuur 2.1 een tweelaags netwerk (alleen de lagen met 'werkzame' units worden



Figuur 2.1: Twee afbeeldingen van één netwerk

geteld), anderen noemen het een drielaags netwerk (de invoerlaag wordt meegeteld).

2.1 Werking

Het feedforward netwerk zet een invoervector $\mathbf{x} = x_1 \cdots x_n$ om in een uitvoervector $\mathbf{y} = y_1 \cdots y_m$. De activaties van de invoerunits a_i wordt bepaald door de invoerwaarden x_i. De activaties van de verborgen units en de uitvoerunits wordt bepaald door:

$$a_i = f(\sum_j a_j w_{ij} - bias_i) \tag{2.1}$$

Hierin zijn a_j de activaties van alle units die een verbinding hebben naar unit i. De activatie van unit i kan dus pas berekend worden als alle activaties van de units j bekend zijn. w_{ij} is het gewicht van unit j naar unit i. (De notatie w_{ji} wordt ook vaak gebruikt.) $f()$ is vaak de sigmoïde functie:

$$y = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Een andere veel gebruikte functie is de binaire threshold functie:

$$y = \begin{cases} 1 & : \quad x > 0 \\ 0 & : \quad x \leq 0 \end{cases} \tag{2.3}$$

De uitvoervector $\mathbf{y}$ van het netwerk wordt afgelezen van de uitvoerunits.

2.2 Mogelijkheden van een feedforward netwerk

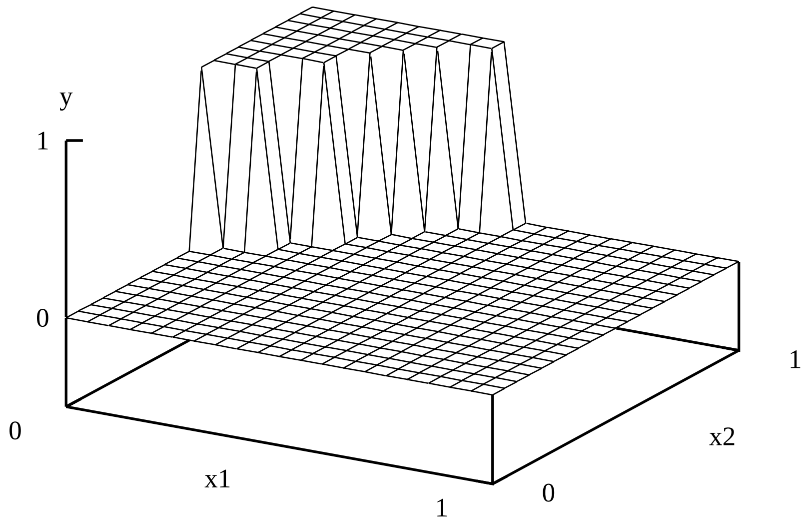
Een feedforward netwerk voert een functie uit:

$$\mathbf{y} = f(\mathbf{x}) \tag{2.4}$$

De waarden van x_i en y_j kunnen zowel binaire als reële waarden zijn. Een veelvoorkomend gebruik is voor classificatie van de invoer: $y_j = 1$ betekent dat $\mathbf{x}$ in klasse j valt, $y_j = 0$ betekent dat $\mathbf{x}$ niet in klasse j valt.

Wat voor classificaties kan een feedforward netwerk uitvoeren?

Een netwerk zonder verborgen units kan alleen vectoren scheiden in lineair verdeelbare klassen. Een set vectoren met dimensie n is lineair verdeelbaar in twee klassen als de klassen gescheiden kunnen worden door een hypervlak met dimensie $n - 1$. Dit hypervlak wordt de *decision surface* genoemd voor de unit die de classificatie uitvoert. Bijvoorbeeld, een set van tweedimensionale vectoren is te scheiden door een rechte lijn, een set driedimensionale vectoren is te scheiden door een plat vlak. Figuur 2.2 toont een voorbeeld voor een lineair gescheiden tweedimensionale set. Als activatie-functie is de threshold functie uit vergelijking 2.3 gebruikt.

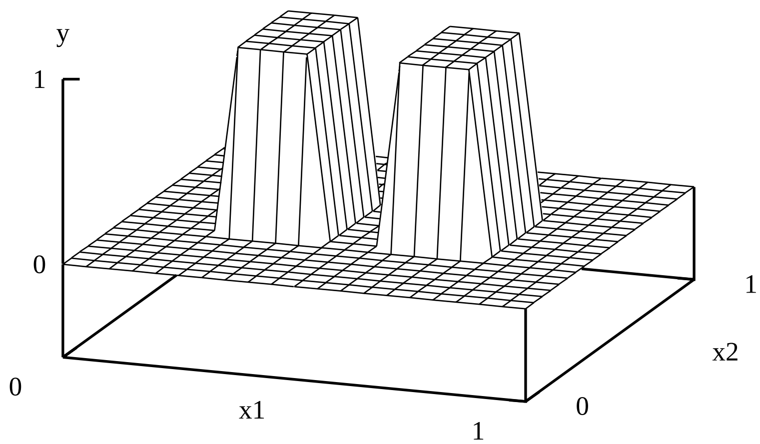


Figuur 2.2: Lineaire classificatie

Een netwerk met verborgen units die een lineaire activatie-functie[1] hebben kan niet meer dan een netwerk zonder verborgen units. Maar wanneer de verborgen units een non-lineaire activatie-functie hebben zoals de sigmoïde of de threshold kunnen door het netwerk decision surfaces van verborgen units gecombineerd worden. Een netwerk met slechts één laag (met voldoende) verborgen units kan elke denkbare indelingen in klassen uitvoeren. (Meer lagen met verborgen units kunnen toegevoegd worden om het netwerk de taak efficiënter te laten uitvoern.) Figuur 2.3 laat een voorbeeld zien. Weer is de threshold functie gebruikt.

Behalve voor classificatie kunnen feedforward netwerken gebruikt worden voor het benaderen van functies. In dit geval wordt vaak een lineaire activatie-functie gebruikt voor de uitvoerunits, en sigmoïde activatie-functies voor verborgen units. Zo'n netwerk met één laag verborgen units kan elke continue functie benaderen. Met twee lagen verborgen units kunnen ook functies die discontinuïteiten bevatten worden benaderd.

1. Een functie f is lineair als hij aan de volgende twee eisen voldoet: $f(cx) = cf(x)$ en $f(x_1 + x_2) = f(x_1) + f(x_2)$



Figuur 2.3: Complexe classificatie

Hoofdstuk 3

Backpropagation

Het werk van Rumelhart, McClelland en de PDP Research Group, zoals beschreven in [RMtPRG86, MRtPRG86], is voor een groot deel verantwoordelijk voor de hernieuwde belangstelling voor neurale netwerken sinds het eind van de jaren tachtig.

Ook de populariteit van backpropagation als leermethode voor (voornamelijk) feedforward netwerken is door hun bekend gemaakt. Het is beschreven in [RHW86]. Toch zijn Rumelhart, Hinton en Williams niet de ontdekkers van backpropagation. Eerdere ontdekkingen van het algoritme zijn beschreven in [Wer74, Par85].

De beschrijving van het algoritme wordt hier achterwege gelaten. Zie hiervoor [MR88, Chapter 5] of [RHW86].

3.1 Beperkingen van backpropagation

Het backpropagation netwerk (BPN) heeft een aantal ernstige nadelen waardoor het onbruikbaar is voor het ontwikkelen van grotere intelligente systemen.

- Een BPN kan moeilijk 'opgeschaald' worden voor grotere problemen. Relatief kleine verhogingen van het aantal units in een verborgen laag kan het aantal trainingsronden dat nodig is explosief doen toenemen. Hetzelfde geldt voor wanneer de verborgen laag gesplitst moet worden in meerdere lagen.
  Reeksverwerking met simpele recurrente netwerken zoals het Elman-netwerk draagt nog extra bij aan de inefficiëntie. Verlenging van een reeks met één symbool kan het aantal trainingsronden verdubbelen. Bovendien zijn voor het leren van langere reeksen meer units nodig in de verborgen laag.
- Backpropagation maakt gebruik van *supervised learning*. Dit houdt in dat het netwerk een taak alleen kan leren als er voorbeelden van invoer en uitvoer beschikbaar zijn. Deze zijn er lang niet altijd, zeker niet voor de meer basale taken zoals (coördinatie van) waarneming en beweging.
- Een BPN heeft een leerfase en een gebruiksfase. In de gebruiksfase kan het netwerk zich niet aanpassen aan veranderingen in de omgeving. Om

te kunnen reageren op nieuwe situaties moet het netwerk opnieuw worden getraind. Hierbij gaat oude kennis verloren, tenzij de oude trainings-paren meegenomen worden in de nieuwe trainingsset.

- De taak die met één netwerk kan worden verricht is nogal beperkt: het omzetten van een set invoerpatronen in een set uitvoerpatronen. Het gedrag van een intelligent systeem is uiteraard veel complexer. Vele subnetwerken moeten op een zeer ingewikkelde manier met elkaar samenwerken. Backpropagation netwerken kunnen niet aan elkaar worden gekoppeld, omdat de patronen die de communicatie tussen de netwerken vormen niet bekend zijn uit voorbeelden, en dus niet geleerd kunnen worden.
- Een BPN is een *feedforward* netwerk, dat statische invoer omzet in een statische uitvoer. Complexe handelingen bestaan uit reeksen instructies die onder invloed van *feedback* worden bijgestuurd. Netwerken die gebruik maken van feedback kunnen moeilijk met backpropagation worden getraind. Bovendien moeten dergelijke netwerken vaak ook de *lay-out* voor de verbindingen leren. De vraag hoeveel units er nodig zijn en hoe ze onderling moeten worden verbonden zijn problemen waarop het systeem zelf een antwoord zal moeten vinden.

Meer over beperkingen van feedforward netwerken is te vinden in [MP88].

Voor iemand die gewend is om voor het oplossen van een probleem een computer-programma te schrijven is het idee dat een netwerk z'n eigen gewichten leert misschien al heel revolutionair. Maar dit gaat lang niet ver genoeg om echte intelligente systemen te kunnen ontwikkelen. Hiervoor zijn netwerken nodig die zich ontwikkelen door middel van *unsupervised learning*, dat wil zeggen dat ze leren reageren op hun omgeving zonder dat hun van buiten af verteld wordt wat er van ze wordt verwacht.

Hoofdstuk 4

Ruimte en tijd in neurale netwerken

Intelligent gedrag vereist verwerking van zowel ruimtelijke als temporele informatie. Gedrag houdt verandering in. De huidige situatie is anders in vergelijking met eerdere en latere situaties. De volgorde waarin de veranderingen plaatsvinden is gestructureerd.

Voor een systeem van neurale netwerken vormen invoer- en uitvoer-vectoren de ruimtelijk informatie. De opeenvolging van vectoren vormt de temporele dimensie.

Een eenvoudig feedforward netwerk maakt geen gebruik van temporele informatie. Het netwerk verwerkt de huidige invoer zonder rekening te houden met oudere invoer.

4.1 Modellering van temporele processen

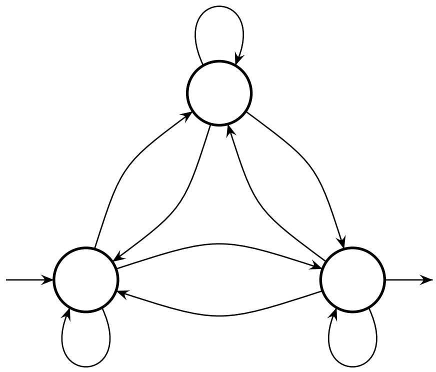
Er zijn twee methodes om de verwerking van ruimtelijk/temporele informatie in een neuraal netwerk te modelleren. De eerste methode gebruikt discrete stappen in de tijd. De toestand $\mathbf{a}$ van het netwerk — dat wil zeggen het geheel van alle activaties in het netwerk — is een functie F van de huidige invoer $\mathbf{x}$ en de toestand van het netwerk in de vorige fase:

$$\mathbf{a}_t = F(\mathbf{x}_t, \mathbf{a}_{t-1}) \tag{4.1}$$

Dit is een effectieve methode voor simulaties van neurale netwerken in seriële computers.

Het nadeel van de discrete methode is dat de invoer en de uitvoer gesegmenteerd moeten zijn. Alleen de tweede methode is in staat continue invoer en uitvoer te verwerken. Hierin wordt de verandering van de toestand van het netwerk in de tijd $\mathrm{d}\mathbf{a}/\mathrm{d}t$ weergegeven:

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = F(\mathbf{x}, \mathbf{a}) \tag{4.2}$$



Figuur 4.1: Recurrent netwerk

4.2 Feedback

Feedback is onmisbaar in temporele processen. Deze feedback kan zowel van buiten het netwerk komen, door het waarnemen van veranderingen in de omgeving die het gevolg zijn van de uitvoer van het netwerk, als van binnen het netwerk.

Een recurrent netwerk maakt gebruik van interne feedback. Figuur 4.1 toont een simpel voorbeeld. Elke unit kan zowel voor invoer als uitvoer gebruikt worden. Het belangrijkste kenmerk van dit netwerk is dat de informatie niet in één richting stroomt, zoals in een feedforward netwerk.

De manier waarop de activaties voor een unit worden berekend hangt af van het gebruik van het netwerk: discreet of continu. In het discrete model is de activatie a van unit i bepaald door (voorbeeld):

$$a_{i,t} = f(x_{i,t} + \sum_j w_{ij} a_{j,t-1}) \tag{4.3}$$

Hierin is x_i de externe invoer naar unit i. w_{ij} is het gewicht van unit j naar unit i. f is een functie, bijvoorbeeld een sigmoid.

In het discrete model spelen oude activatiewaarden alleen indirect een rol door de recurrente verbindingen. In het continumodel worden deze waarden ook direct gebruikt. De activatie van unit i wordt bepaald door (voorbeeld):

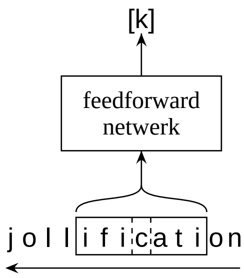
$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = -Pa_i + Qf(x_i + \sum_j w_{ij} a_j) \tag{4.4}$$

P en Q zijn positieve constanten.

4.3 Voorbeelden

In algemen boeken over neurale netwerken, zoals overzichten en inleidingen, krijgen continue modellen van temporele netwerken weinig of geen aandacht. Zie [FS91, Chapter 9] voor een voorbeeld van spraakherkenning.

Hieronder volgen beschrijvingen van discrete modellen.



Figuur 4.2: NETtalk

4.3.1 Ruimtelijke representaties van tijd

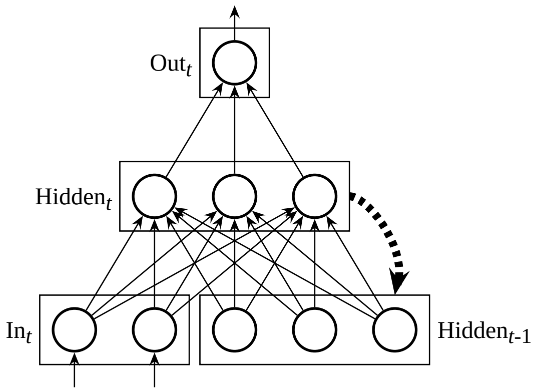
De meest eenvoudige methode die gebruikt kan worden voor het omgaan met temporele patronen is deze vertalen naar een ruimtelijke weergave. Deze methode is gebruikt in het veelbesproken systeem NETtalk [SR87]. Figuur 4.2 laat de architectuur van NETtalk zien. NETtalk wordt gebruikt voor het omzetten van geschreven tekst in spraak. De figuur laat zien hoe een letter samen met enkele voorgaande en volgende letters wordt aangeboden aan een standaard feedforward netwerk. Als uitvoer geeft het netwerk een fonetisch symbool voor de weergave van de letter uit het centrum van het 'invoervenster'. Daarna worden alle letters 1 positie opgeschoven voor het genereren van het volgende foneem. NETtalk is zeer succesvol. Binnen enkele uren leert het tekst om te zetten in spraak van een redelijk hoge kwaliteit. In dit model wordt het temporele aspect buiten het netwerk om gemodelleerd. Voor het netwerk maakt het niet uit in welke volgorde de lettergroepen worden aangeboden.

Deze ruimtelijke methode heeft een zeer belangrijke beperking. Het gaat uit van een (bijna) 1 op 1 relatie tussen invoer en uitvoer: Elke invoer aan het netwerk levert een uitvoer, invoerreeks en uitvoerreeks zijn even lang. In NETtalk levert dit al problemen. Groepen letters representeren samen één foneem. Dit is op te lossen door voor elke letter uit zo'n groep het zelfde foneem te genereren en voor de verdere produktie van spraak dubbele fonemen te negeren. Maar hoe moet het wanneer één invoersymbool een reeks uitvoersymbolen moet genereren? Hoe dit probleem in NETtalk wordt opgelost vertelt het verhaal niet.[1]

Je kunt natuurlijk de hele reeks invoersymbolen aan elkaar plakken en in één keer aanbieden aan het netwerk, waarbij het netwerk de uitvoer op gelijksoortige wijze produceert. Dit heeft enkele grote nadelen:

- Er moeten genoeg invoer- en uitvoer-units zijn om de langste reeks te kunnen representeren. De maximale lengte kan zeer groot of onbekend zijn. Meestal zal het merendeel van de units niet gebruikt worden.
- Invoeging van een symbool aan het begin van de reeks levert een compleet nieuw patroon op.
- Herhalingen in de reeks worden niet als zodanig herkend.

1. Misschien wel het originele artikel, ik heb dit niet gelezen.



Figuur 4.3: Elman netwerk

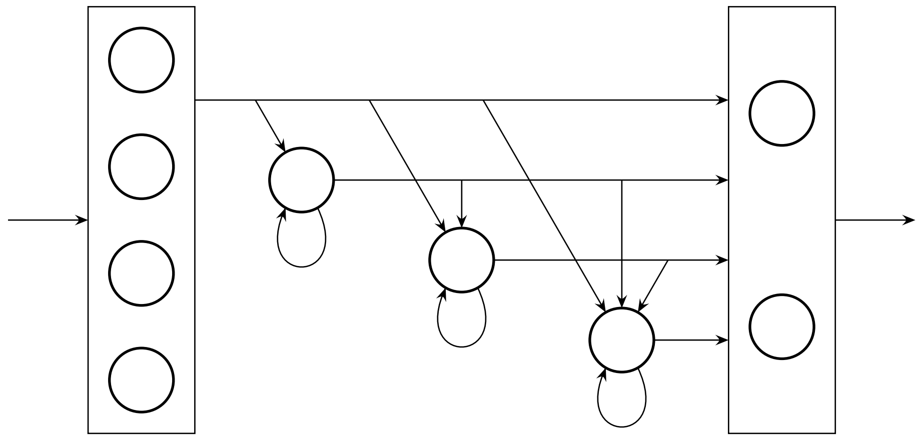
4.3.2 Het netwerk van Elman

Jeffrey Elman heeft een netwerk ontworpen dat instaat tussen een feedforward netwerk en een recurrent netwerk zoals afgebeeld in figuur 4.1. Het Elman-netwerk is te zien in figuur 4.3. Dit netwerk wordt wel een *simple recurrent network* genoemd. In principe is dit een standaard feedforward netwerk dat getraind kan worden met backpropagation. Het verschil zit hem in de manier waarop de invoer aan het netwerk wordt aangeboden. Slechts een gedeelte van de invoerunits wordt gebruikt voor het aanbieden van een patroon. De rest van de invoerunits ontvangt een kopie van de waarden van de verborgen units na de verwerking van het vorige patroon uit de reeks.

Dit netwerk is op verschillende manieren te gebruiken. Het genereert voor elke invoer een uitvoer. Je kunt voor elke stap het zelfde patroon aanbieden als invoer. In dit geval zet het netwerk één symbool om in een reeks symbolen. Je kunt het netwerk een reeks symbolen aanbieden waarna alleen de uitvoer na de laatste invoer van belang is. Op deze wijze zet het netwerk een reeks symbolen om in één symbool. Vooral deze laatste functie is echter zeer moeilijk te leren.

Het Elman-netwerk leert nog veel trager dan een gewoon backpropagation netwerk. Dit is als volgt te verklaren: Stel dat het eerste patroon uit een reeks aan het netwerk wordt aangeboden. Na de feedforward-fase worden op de gebruikelijke manieren de gewichten aangepast door middel van backpropagation. Daarna wordt het tweede patroon uit de reeks aangeboden. De fout die het netwerk nu produceert is niet alleen het gevolg van foutieve gewichten, maar ook van de foutieve activatiewaarden van de verborgen units uit de eerste feedforward-fase uit de reeks. De correcte waarden voor deze units zijn niet bekend. Het netwerk moet dus leren met een foutieve invoer.

Ondanks dit is een Elman-netwerk tot leuke dingen in staat. Jeffrey Elman heeft laten zijn dat zijn netwerk een context-vrije grammatica kan leren waarbij het niveau van center-embedding is beperkt. ([Elm90], herdrukt in [Alt90]) Het netwerk bleek de taak te kunnen leren wanneer de trainingsset eerst alleen korte



Figuur 4.4: Recurrent Cascade-Correlation Architecture

eenvoudige zinnen bevatte en langzaam werd uitgebreid met langere meer complexe zinnen. De stadia van 'taalvaardigheid' die het netwerk doorloopt lijken op de stadia die kinderen doorlopen.

4.3.3 Recurrent Cascade-Correlation Architecture

Een echt recurrent netwerk is de *Recurrent Cascade-Correlation Architecture* [Fah91]. Dit is een recurrente versie van Cascade-Correlation [FL90]. Dit netwerk leert sneller dan het netwerk van Elman, en is in staat taken te leren die voor het Elman-netwerk te moeilijk zijn.

Cascade-Correlation bouwt een netwerk op door verborgen units toe te voegen wanneer dit nodig is. Het leerproces begint met een netwerk zonder verborgen units. Dit netwerk wordt zo goed als mogelijk getraind. Wanneer een optimum is bereikt zonder dat het netwerk de taak correct uitvoert wordt een verborgen units toegevoegd. Een nieuwe verborgen unit krijgt inkomende verbindingen van alle invoerunits en alle eerder toegevoegde verborgen units, en uitgaande verbindingen naar alle uitvoerunits. Wanneer een nieuwe unit wordt toegevoegd worden de gewichten van de oudere units bevroren, alleen de nieuwe unit wordt getraind.

In de recurrente versie hebben de verborgen units verbindingen naar zichzelf, maar niet terug naar de oudere units (die immers niet meer mee doen in het leren). Figuur 4.4 laat dit netwerk zien. (Voor de duidelijkheid zijn slechts verbindingen voor één invoer- en één uitvoer-unit getekend.) In tegenstelling tot Elman's netwerk houdt de leerprocedure voor dit netwerk wel rekening met foutieve activatie-waarden in de verborgen units.

4.4 Timing

In temporele processen is niet alleen de volgorde waarin acties plaatsvinden van belang, maar ook de snelheid.

In het discrete model bepaald het netwerk alleen het aantal stappen dat gebruikt wordt voor een complexe bewerking. Het tempo wordt bepaald door de tijd die voor één discrete bewerking wordt gebruikt — door een externe klok.

In het continu model zijn het de activatiefuncties voor de units die het tempo bepalen.

Een voorbeeld waarin timing een grote rol speelt is het uitvoeren van ballistische bewegingen. Ballistische bewegingen zijn bewegingen die geen gebruik maken van feedback, maar alleen gestuurd worden door een van tevoren vastgestelde reeks commando's aan de spieren. Een voorbeeld is het raken van een doel op grote afstand door er een voorwerp naar toe te gooien. De spieren moeten een reeks bewegingen zeer snel uitvoeren, er is geen tijd voor verwerking van feedback en bijsturen van de commando's tijdens de beweging. Slechts achteraf kan beoordeeld worden of de beweging correct was: trof het voorwerp doel?

Het uitvoeren van reeksen commando's zonder feedback lijkt simpeler dan met feedback, maar is in feite veel moeilijker wanneer dezelfde precisie wordt vereist. Wil je een klein prooidier dodelijk treffen van veertig meter afstand dan moeten alle spierbewegingen nauwkeurig op elkaar zijn afgesteld. Het gaat nu niet alleen om het in juiste volgorde uitvoeren van commando's. Timing is essentieel voor het bepalen van het juiste moment van begin en eind van bewegingen van de individuele spieren.

Alleen mensen kunnen beschikken over de timing die nodig is voor het nauwkeurig gooien van voorwerpen. Hoe hebben we deze nauwkeurigheid bereikt? Is het een gevolg van de hogere intelligentie die ons onderscheidt van dieren? Volgens William Calvin is ons werpvermogen juist de bron van ons grote denkvermogen [Cal86, Cal, Cal90]. Zijn theorie gaat als volgt:

Biologische neuronen zijn zeer onnauwkeurig als het gaat om timing. De tijd die het een neuron van nature kost om een inkomend signaal om te zetten in een uitgaand signaal is te willekeurig om te kunnen gebruiken. Wanneer je zeer veel neuronen kunt gebruiken voor één taak, dan leidt het gemiddelde van de variaties in de individuele neuronen tot een nauwkeurige timing, hoe meer neuronen gebruikt worden, hoe nauwkeuriger de timing. De jacht was voor de eerste mensen de evolutionaire motor achter de grote toename van onze herseninhoud. Elke toename leidde tot een betere precisie in het werpen van wapens naar prooidieren, en zo tot betere overlevingskansen.

Volgens William Calvin is ons vermogen taal te gebruiken een bijprodukt van ons verworven vermogen tot het precies uitvoeren van werpbewegingen. Ons grote vermogen voor coördinatie van reeksen bewegingen zou ons in staat stellen reeksen symbolen te manipuleren.

Hoe goede timing is te bereiken in neurale netwerken is te complex om hier te bespreken. Zie bijvoorbeeld [LMT91] *Part III Oscillation.*

4.5 Natuurlijke taal en neurale netwerken

Hoe kunnen neurale netwerken gebruikt worden voor de verwerking van natuurlijke taal? Een eenvoudig recurrent netwerk zoals dat van Elman kan een finite state grammatica leren. Door een gedeelte van de capaciteit van de verborgen units te gebruiken als stack of queue kan een context-vrije of context-gevoelige grammatica worden gesimuleerd (met beperkte diepte voor center-embedding of crossing dependencies). Echter, wanneer je met de huidige stand van zaken neurale netwerken wilt gebruiken voor hogere cognitieve processen zoals taal, dan begin je aan het verkeerde eind. Je herhaalt de fout van de traditionele AI. In [Cli91, p. 29] wordt dit zo verwoord:

> ...the study of linguistic processes using network models is wildly premature. The study of insects is advocated as the most fruitful path for future research.

en verder (p. 37–38):

> Language will be best understood as a very high layer in a subsumption architecture: how it interacts with lower layers could be of vital importance, and we should study these lower layers first.
>
> If we are still yet to determine the subsymbolic or neural basis underlying the dance-language of bees, how then are we supposed to study such aspects of human language at anything but the most gross level of neuroanatomy (i.e. studies of lesioned patients)? We simply do not know enough.

Hoofdstuk 5

Topografische kaarten

Figuur 5.1 laat een voorbeeld zien van een *topografische neurale kaart*.[1] Een groep units (external input) heeft verbindingen naar elke unit in een *competitive layer* die de topografische kaart vormt.

Topografisch houdt in dat de units op de kaart zijn geordend. Dat wil zeggen dat de unit die het sterkst reageert op een invoer omringd wordt door units die ook reageren op die invoer, maar minder sterk. Units die verder weg liggen reageren niet of nauwelijks.

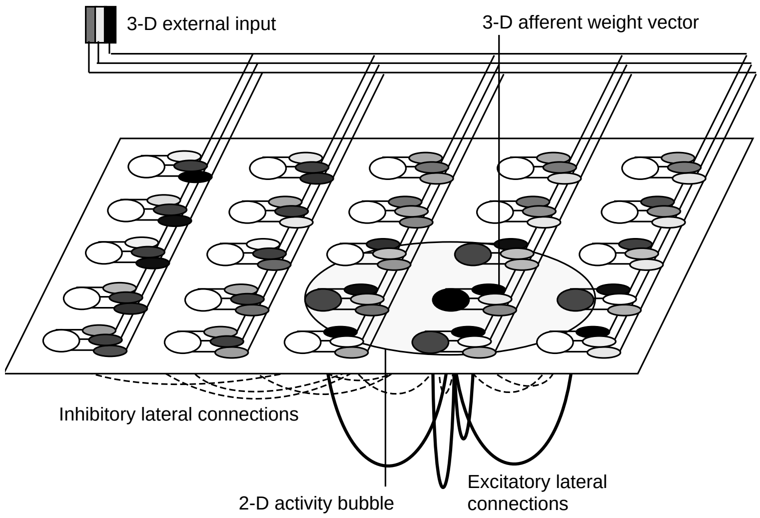
De kaart bevat *laterale connecties*: verbindingen tussen units in dezelfde laag. Elke unit heeft exciterende verbindingen naar nabijgelegen units, en inhiberende verbindingen naar units die verder weg liggen. Deze verbindingen hebben twee functies. Ten eerste vergroten ze het contrast waarmee de units in de kaart reageren op invoer van buiten af. Ten tweede maken deze laterale verbindingen het mogelijk dat de kaart zichzelf ordent wanneer het leert reageren op externe invoer. De kaart past zich aan aan de invoer, zonder dat er van buiten af verteld wordt hoe op de data gereageerd moet worden. Daarom noemt men zo'n topografische kaart ook wel een *Self-Organizing Map*, kortweg SOM.

5.1 Feedback connecties

Backpropagation maakt gebruik van feedforward netwerken. Aanpassing van de activatiewaarde van een unit in zo'n netwerk is eenvoudig. De invoerwaarden worden vermenigvuldigd met de gewichten en daarna bij elkaar opgeteld. Dit is de netinput, die door middel van een sigmoïde functie in één keer omgezet wordt in de activatiewaarde.

De topografische kaart heeft laterale verbindingen die voor feedback zorgen. Unit a wordt beïnvloed door unit b en genereert een uitvoer naar unit b, die daardoor z'n invloed op unit a verandert, waardoor unit a z'n invloed op unit b verandert, die daardoor…enzovoort. Om de activatie van de units te beschrijven wordt gebruik gemaakt van differentiaal-vergelijkingen. Deze vergelijkingen berekenen niet rechtstreeks de activatie van de units (a_i), maar beschrijven hoe

1. Afbeelding uit [SM94]



Figuur 5.1: Topografische neurale kaart

de activatie verandert met de tijd ($\mathrm{d}a_i/\mathrm{d}t$). Een eenvoudig voorbeeld van zo'n vergelijking is:

$$\frac{\mathrm{d}a_i}{\mathrm{d}t} = -decay(a_i - rest) + (max - a_i)\sigma_E(E) - (a_i - min)\sigma_I(I) \qquad (5.1)$$

Hierin zijn min, max en $rest$ resp. de minimale activatiewaarde, de maximale activatiewaarde en de rustactivatie van de unit. $rest$ ligt tussen min en max, gewoonlijk dichter bij min dan max. Voorbeelden zijn: $min = 0$, $rest = 0.2$ en $max = 1$.

De eerste term na de = in vergelijking 5.1 beschrijft het verlies van de activatie van unit i. $decay$ is een waarde tussen 0 en 1. Het effect van deze term is dat als de er geen externe invoer naar de unit is de activatie van de unit verandert richting $rest$.

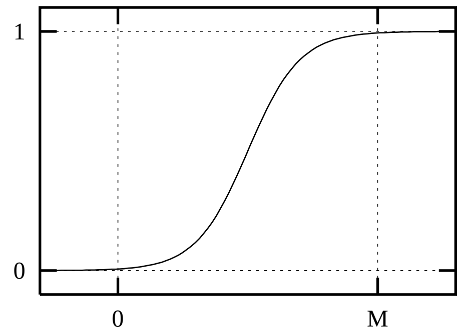
De tweede term beschrijft het effect van exciterende verbindingen naar de unit. E staat voor de som van inkomende verbindingen maal gewichten waarvoor geldt dat de gewichten positief zijn. σ_E is een sigmoïde functie die E schaalt tussen 0 en 1. De tweede term verhoogt a_i in de richting van max.

De derde term is vergelijkbaar met de tweede. Het is het effect van inhiberende verbindingen op de unit, die a_i omlaag drijft richting min.

De sigmoïde functies schalen een waarde tussen 0 en een bepaald maximum M (Dat onder andere afhangt van het aantal verbindingen) naar een waarde tussen 0 en 1. Een voorbeeld is weergegeven in figuur 5.2.

5.2 Leren in een competitive map

Voor het vervolg van de uitleg is het goed om patronen en gewichten op te vatten als vectoren. Een patroon met de waarden $x_1, x_2, \cdots, x_n$ vormt de vector $\mathbf{x}$. De



Figuur 5.2: $\sigma(x) = \frac{1}{1+e^{5-10x/M}}$

vector $\mathbf{x}$ wijst van de oorsprong naar een punt in de n-dimensionale ruimte, en heeft de lengte $\sqrt{\sum_i x_i^2}$ (Stelling van Pythagoras voor $n = 2$). De vector $\mathbf{w}_i$ wordt gevormd door de gewichten naar unit i.

Leren vindt plaats door middel van een variant op *Hebbian Learning*. Het volgende citaat is uit [Heb49, p. 50]:

> 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 that A's efficiency, as one of the cells firing B, is increased.

De volgende vergelijking implementeert dit idee (α is de leerconstante):

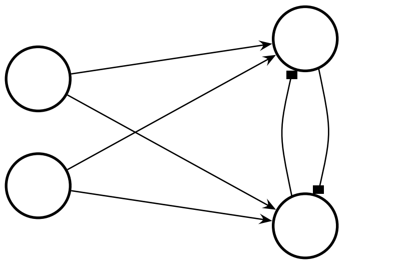
$$\frac{\mathrm{d}\mathbf{w}_i}{\mathrm{d}t} = \alpha \mathbf{x} a_i \tag{5.2}$$

Wil het netwerk goed in staat zijn om te leren dan moeten de inputvectors en de weightvectors van te voren geschaald worden zodat ze allemaal de lengte 1 hebben.[2] Vergelijking 5.2 wordt aangepast zodat de lengtes van de weightvectors constant blijven: alleen de richtingen veranderen. (Je zou dit kunnen opvatten als het herverdelen van beschikbare middelen.)

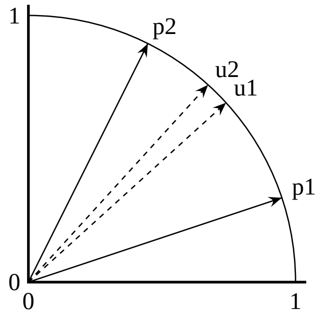
5.3 Een voorbeeld: De minimale competitive kaart

Figuur 5.3 toont een netwerk dat gebruikt wordt voor de demonstratie die volgt. De competitive laag heeft slechts twee units waardoor het de invoer kan scheiden in maximaal twee groepen. De invoerpatronen bevatten slechts twee elementen,

2. Hierbij gaat uit de invoer informatie verloren: Vectoren in N-dimensionale ruimte worden teruggebracht naar vectoren naar een hypersfeer in die N-dimensionale ruimte. Bijvoorbeeld, de vectoren $\mathbf{x}_1 = [1, 2, 2]$ en $\mathbf{x}_2 = [2, 4, 4]$ worden beiden geschaald naar $\mathbf{x} = [\frac{1}{3}, \frac{2}{3}, \frac{2}{3}]$. Om te voorkomen dat essentiële informatie verloren gaat kun je het volgende doen: Verhoog het aantal dimensies van de invoervector met 1, en geef deze de waarde 1. Schaling van $\mathbf{x}_1$ en $\mathbf{x}_2$ levert nu 2 unieke vectoren op: $\mathbf{x}'_1 = [0.316, 0.632, 0.632, 0.316]$ en $\mathbf{x}'_2 = [0.329, 0.658, 0.658, 0.164]$. Deze bewerking is omkeerbaar.



Figuur 5.3: Minimaal competitive netwerk



Figuur 5.4: Invoervectoren en gewichtsvectoren

dus invoervector en weightvectors hebben dimensie 2. De units in de competitive laag zijn onderling alleen verbonden door inhiberende verbindingen. De gewichten van deze verbindingen zijn -1 en veranderen niet. De verbindingen die van de invoer komen zijn exciterend.

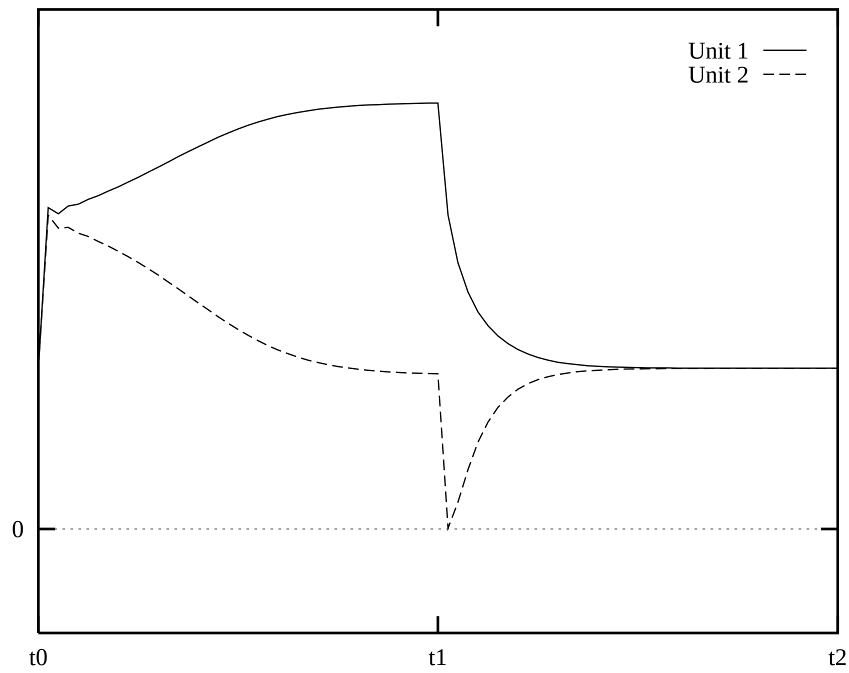
Er worden 2 invoervectoren gebruikt. Figuur 5.4 laat deze vectoren zien ($p1$ en $p2$), en toont de beginwaarden van de gewichten ($u1$ voor unit 1, $u2$ voor unit 2). De bedoeling is dat de vectoren van de gewichten tijdens het leren zich verplaatsen naar de invoervectoren.

Bekijk eerst het effect van vergelijking 5.1. Figuur 5.5 laat het verloop van de activatie van de beide units zien als functie van de tijd. Vergelijking 5.2 die de gewichten verandert wordt hierbij niet gebruikt ($\alpha = 0$).

Op tijdstip $t0$ wordt patroon 1 aangeboden aan het netwerk. De activatie van zowel unit 1 als unit 2 lopen in het begin zeer snel op, maar de activatie van unit 1 stijgt net iets sneller dan die van unit 2. Hierdoor remt unit 1 unit 2 sterker dan andersom. Uiteindelijk wint unit 1 het van unit 2.

Op tijdstip $t1$ wordt de invoer verwijderd. De hoge activatie van unit 1 zorgt er nu voor dat de activatie van unit 2 zeer snel daalt. Uiteindelijk keren de activaties van beide units terug naar een rustwaarde.[3]

3. De rustwaarde is kleiner dan *rest* in vergelijking 5.1 omdat de beide units elkaar blijven remmen.



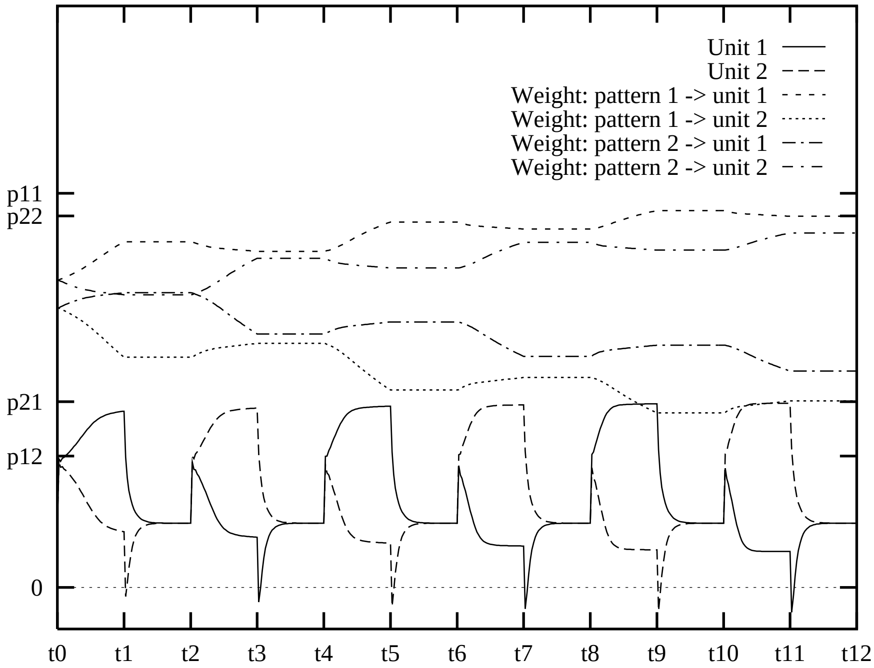
Figuur 5.5: Reactie van ongetrainde units op patroon 1

Figuur 5.6 laat zien hoe de activatie van de beide units verloopt wanneer beide patronen om de beurt worden aangeboden aan het netwerk. Vergelijking 5.1 en vergelijking 5.2 zijn nu gelijktijdig actief. Van tijdstip $t0$ tot tijdstip $t1$ wordt patroon 1 aangeboden. Van $t1$ tot $t2$ is er geen externe invoer. Van $t2$ tot $t3$ wordt patroon 2 aangeboden. Hierna volgt weer een rustperiode tot $t4$. Deze cyclus wordt nog 2 keer herhaald. Het is te zien dat in de derde cyclus de activatiewaarden van de units sneller naar hun minimum of maximum gaan. Bovendien liggen de minima en maxima in de derde cyclus verder uit elkaar dan in de eerste cyclus. De waarden van de gewichten veranderen in de richting van waarden uit de invoervectoren.

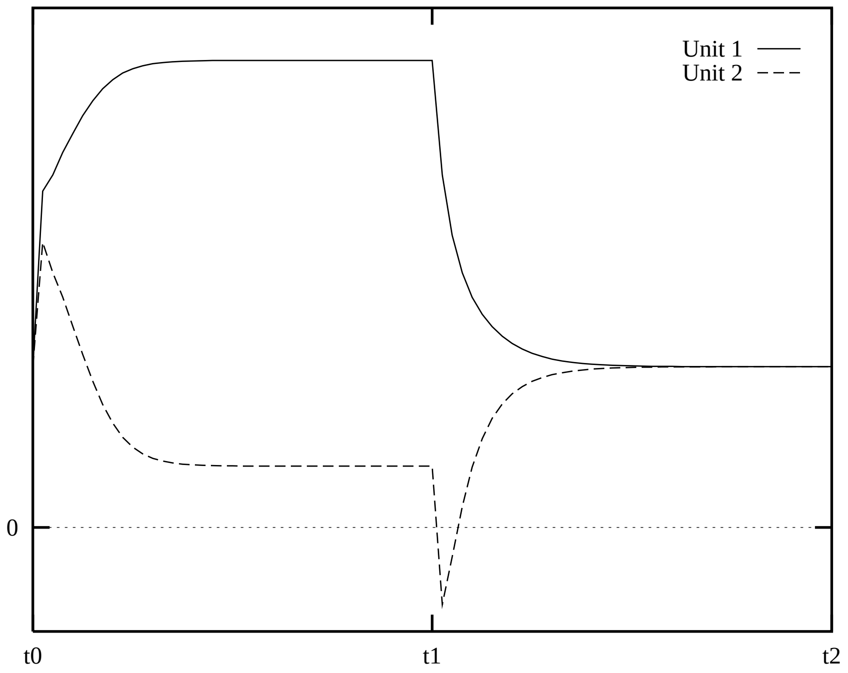
Figuur 5.7 laat zien hoe de activatie van de beide units verloopt wanneer unit 1 maximaal reageert op patroon 1, en unit 2 maximaal reageert op patroon 2. (Nu weer met $\alpha = 0$.) Vergelijk deze grafiek met die in figuur 5.5.

5.4 Ordening

Hoe komt het dat de kaart zich ordent tijdens het leren? De unit die sterk reageert op de invoer exciteert de units in de directe omgeving. Door hun verhoogde activatie leren die buurunits mee in dezelfde richting als de exciterende unit. Daardoor zullen ze later sterker reageren op invoerpatronen die lijken op het patroon waarmee ze eerder hebben geleerd. Inhiberende verbindingen naar units



Figuur 5.6: Training van minimaal competitive netwerk



Figuur 5.7: Reactie van optimaal getrainde units op patroon 1

op grotere afstand zorgen er voor dat er altijd maar één winnend gebied is op de kaart dat kan leren. Dit is een nogal intuïtieve beschrijving voor ordening. Zie [Koh89] voor een wiskundige verklaring. De verklaring is vrij pittig.

5.5 De Kohonen-kaart

Voor een hardware-implementatie van een topografische kaart vormen de vergelijkingen die tot nu toe zijn gebruikt een bruikbaar model. Een simulatie in een seriële computer is echter zeer inefficiënt. Gelukkig zijn er modellen denkbaar waarmee het effect van een topografische kaart bereikt kan worden zonder de computationele belasting van het neurale model te gebruiken. Iemand die veel onderzoek op dit terrein heeft gedaan is Teuvo Kohonen. Zijn boek [Koh89] geeft een zeer goede inleiding in het onderwerp. Dankzij zijn werk worden de vereenvoudigde computermodellen vaak Kohonen-kaarten genoemd. Hieronder volgt een beschrijving van zo'n model.

In dit model is het niet nodig om de invoervectoren te schalen naar gelijke lengte. Het model maakt onderscheid tussen $\mathbf{x}_1 = [1, 2, 2]$ en $\mathbf{x}_2 = [2, 4, 4]$. Voor het leren krijgen alle gewichten van alle units een random startwaarde. Wanneer een patroon wordt aangeboden wordt dit vergeleken met elke unit. De *Euclidean Distance* wordt berekent:

$$D_i = \sqrt{\sum_j (x_j - w_{ij})^2} \tag{5.3}$$

D_i is de Euclidean Distance tussen unit i en invoervector $\mathbf{x}$. x_j is het $j$de element uit $\mathbf{x}$. w_{ij} is het gewicht van x_j naar unit i.

De units hebben geen laterale verbindingen. Het effect van die verbindingen wordt als volgt geïmiteerd: De unit met de kleinste D_i is de winnaar.[4] (In het neurale model is er een groep winnende units met een hoge activatie. De verder weg gelegen units hebben een sterk afnemende activatie.) De groep units die mag leren bestaat behalve uit de winnende unit uit de units die maximaal N posities op de kaart van de winnende unit zijn verwijderd. Deze units leren aan de hand van de volgende vergelijking:

$$\Delta w_{ij} = \alpha(x_i - w_{ij}) \tag{5.4}$$

De omgeving N is groot in de beginfase van het leren, bijvoorbeeld de helft van de diameter van de kaart, en neemt gelijdelijk af naar de eindfase tot bijvoorbeeld 1. De waarde van de leerconstante α neemt tijdens het leren af van maximaal 1 in het begin naar minimaal 0 op het einde.

4. In plaats van de Euclidean Distance kan het kwadraat van de Euclidean Distance worden gebruikt. Dit levert dezelfde winnaar, maar gebruikt een rekenkundige bewerking minder.

5.6 Functies en toepassingen van topografische kaarten

Tot zover is alleen beschreven hoe topografische kaarten reageren op — en leren aan de hand van — invoer. Er is niets over uitvoer gezegd, maar zonder uitvoer hebben de kaarten geen nut. Onze hersenen maken zeer veel gebruik van topografische kaarten. Wat de rol van deze kaarten is is moeilijk te doorzien. Verschillende functies zijn denkbaar. Het voorbeeld in figuur 5.1 op pagina 24 toont een invoervector van dimensie 3. De kaart bevat 25 units. Wanneer de activaties van deze units samen een invoervector voor een volgende fase vormen dan is er dus sprake van een sterke vergroting van het aantal dimensies. Het kan ook andersom: invoervectoren met meer elementen dan het aantal units op de kaart. Ook is het mogelijk om de positie van de meest actieve unit in de kaart te bepalen. Dan is de invoer teruggebracht naar 2 dimensies: de kaartcoördinaten.

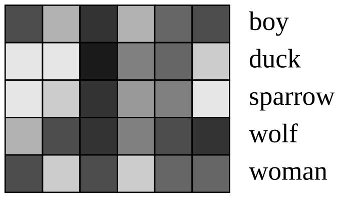
De topografische kaart kan tijdens het gebruik blijven leren. Als een verandering in de omgeving er toe leidt dat de reacties in een bepaald gebied toenemen kan die gebied zich 'uitbreiden' en daarmee de omliggende gebieden 'samenpersen'. Hierdoor leert de kaart beter onderscheid te maken tussen de vectoren in de nieuwe invoer, terwijl het onderscheid tussen meer zeldzame invoervectoren langzaam vervaagt.

Een topografische kaart kan omgekeerd worden gebruikt, een positie van hoogste activatie omzetten in een uitvoer, of een kaart kan de invoer- en uitvoerfunctie kombineren. Twee kaarten, één voor invoer, één voor uitvoer kunnen tegenover elkaar geplaatst en met elkaar verbonden zorgen voor complexe coördinatie, bijvoorbeeld tussen waarnemen en beweging. Lagen van kaarten kunnen complexe invoer omzetten van verzamelingen eigenschappen naar steeds hogere concepten.

Ritter en Schulten [RS86, RS87, RMS89] hebben een topografische kaart gebruikt voor het uitvoeren van eenvoudige ballistische bewegingen door een robotarm.

Teuvo Kohonen heeft een topografische kaart gebruikt voor spraakherkenning [Koh88]. De kaart vormt daarbij representaties voor fonemen, waarop woorden trajecten achterlaten.

Een variant van het *Travelling Salesman Problem* kan opgelost worden met een topografische kaart. In deze variant van het probleem is er een aantal plaatsen waarvan de positie bekend is als een $x-$ en een y-coördinaat. De onderlinge afstand tussen twee plaatsen is evenredig met de rechte lijn die deze twee coördinaten-paren verbindt. De opgave is de kortste route (met eindpunt gelijk aan beginpunt) te vinden die alle plaatsen met elkaar verbindt. Voor dit probleem wordt een 1-dimensionale Kohonen-kaart gebruikt: een lijn. Begin en eindpunt van deze kaart werden met elkaar verbonden zodat een lus ontstaat. Het aantal units in deze lus is een ruim stuk groter dan het aantal plaatsen in de route. Het beschreven Kohonen-algoritme ordent de plaatsen op de lus in een volgorde die overeen komt met een korte route.



Figuur 5.8: Voorbeeld data set: vectoren als grijswaarden

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	1	2	3	4	5	6
1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	1	2	3	4	5	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	8	3	3	3	3	6	6	6	6	6	6	6	6	6	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	1	2	3	4	2	2	2	2	2	2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6

Figuur 5.9: Kohonen input vectors

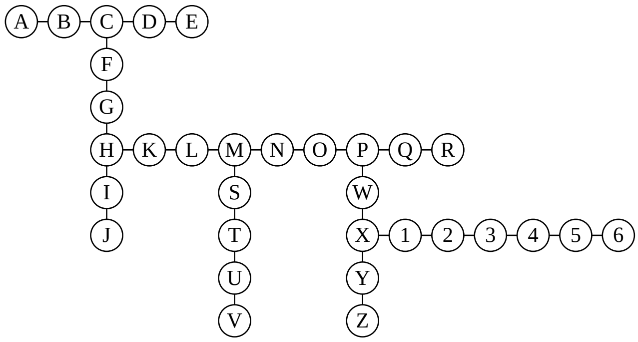
5.7 Analyse van complexe data

Stel: je hebt een grote hoeveelheid gegevens bestaande uit items met een label en een reeks getallen die elk een of andere statistische eigenschap van het item representeren. (De items vormen vectoren met een naam.) Hoe kun je deze data in beeld brengen zodat er verbanden gelegd kunnen worden tussen items in de set gegevens?

Het meest simpele wat je kunt doen is de vectoren af te beelden met reeksen grijswaarden zoals in figuur 5.8. Hierin staat wit voor een lage waarde en zwart voor een hoge waarde. Deze figuur verduidelijkt al meer dan een reeks getallen. Er is duidelijk te zien dat *duck* en *sparrow* sterk op elkaar lijken, en dat *boy* sterk van deze twee afwijkt.

Een Kohonen-kaart kan gebruikt worden voor het zichtbaar maken van verbanden in grote hoeveelheden statistische data. Een hoogdimensionale data-set wordt teruggebracht tot een 2-dimensionale set. Kohonen [Koh89] demonstreert dit met een voorbeeld. Hij gebruikt de data uit figuur 5.9. De onderlinge relaties tussen de vectoren kunnen in dit geval goed worden weergegeven door middel van de *Minimal Spanning Tree* in figuur 5.10. Figuur 5.11 toont een Kohonen-kaart die hiervoor gegenereerd wordt. Voor elke invoervector is aangegeven welke unit in de map het sterkst reageert op die invoer, nadat training van de kaart is voltooid. Er zijn in dit geval meer units dan invoervectoren, zodat de grootste activatie voor elke invoer door een aparte unit wordt bereikt. Wanneer er minder units dan invoerpatronen zijn dan voegt de Kohonen-kaart patronen samen in klassen.

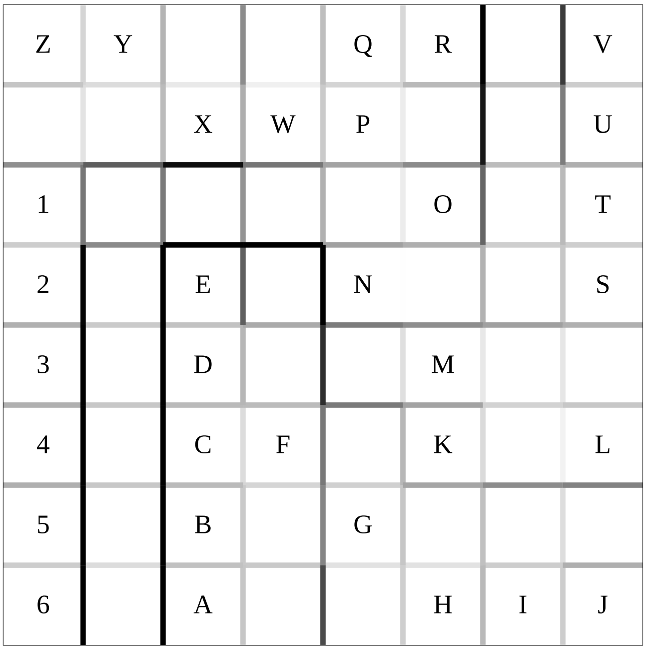
Op deze kaart is te zien dat vectoren die veel overeenkomst vertonen dicht bij elkaar komen te liggen. Clustergrenzen zijn niet goed te zien. Zo is de afstand op de kaart tussen *1* en *X* gelijk aan de afstand tussen *1* en *E*, de



Figuur 5.10: Minimal Spanning Tree voor Kohonen input vectors

Z	Y			Q	R		V
		X	W	P			U
1					O		T
2		E		N			S
3		D			M		
4		C	F		K		L
5		B		G			
6		A			H	I	J

Figuur 5.11: Kohonen-kaart



Figuur 5.12: Kohonen-kaart met clustergrenzen

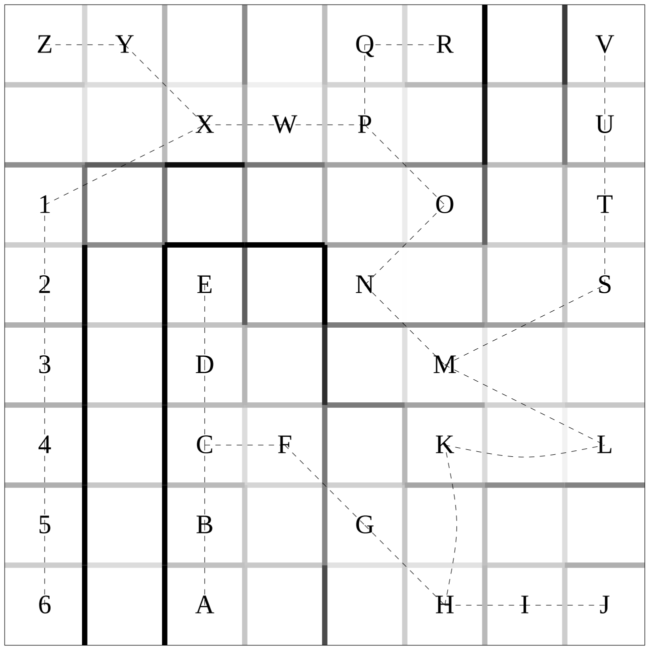
korte afstand tussen *1* en X is terecht, die tussen *1* en E niet (zie figuur 5.10). De clustergrenzen kunnen zichtbaar gemaakt worden door de lijnen tussen de units donkerder te kleuren naarmate het verschil tussen de beide units groter is (Euclidean Distance). Dit is te zien in figuur 5.12.

Je kunt uit de kaart meer informatie aflezen door een minimal spanning tree over de kaart heen te leggen. Het creëren van zo'n figuur voor een willekeurige set gegevens gaat als volgt:

1. Plaats elke vector in een apart cluster. Er zijn in het begin evenveel clusters als vectoren.
2. Herhaal de volgende stappen zolang er nog meer dan 1 cluster over is:
   (a) Selecteer de twee clusters waarvoor de kortst mogelijke verbinding gemaakt kan worden tussen twee vectoren behorende bij elk van de twee clusters. Onthoud deze afstand.
   (b) Voor alle paren vectoren, elk uit een van de twee gevonden clusters, waarvoor geldt dat de afstand gelijk is aan de afstand die gevonden is in de vorige stap: Maak een verbinding tussen beide vectoren.
   (c) Voeg de twee clusters samen tot één cluster.

Wanneer je dit algoritme toepast op de set vectoren uit tabel 5.9 krijg je de structuur die in figuur 5.10 is weergegeven. Als je deze structuur aanbrengt op de Kohonen-kaart krijg je het resultaat in figuur 5.13.

Zoals de kleur van de unit-grenzen iets zegt over het verschil tussen naburige units, zo kan de kleur van de lijnen in de spanning tree gebruikt worden



Figuur 5.13: Kohonen-kaart met clustergrenzen en minimal spanning tree

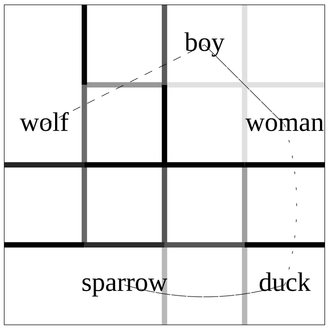
om iets te zeggen over de relatieve afstand van vectoren die door de spanning tree zijn verbonden. In figuur 5.13 zijn de lijnen van de spanning tree identiek, omdat de onderlinge afstand van twee aangrenzende vectoren gelijk is voor alle paren. Figuur 5.15 laat een voorbeeld met verschillende verbindingslijnen zien: de dichte verbinding tussen *sparrow* en *duck* duidt op grote overeenkomst tussen beide vectoren. De zeer open verbindingen tussen *woman* en *duck* duidt op weinig overeenkomst.

Tot slot kun je de set vectoren bewerken door middel van het *merge clustering algorithm.* De werkwijze is vrijwel gelijk aan die voor de minimal spanning tree:

1. Plaats elke vector in een apart cluster. Er zijn in het begin evenveel clusters als vectoren.
2. Herhaal de volgende stappen zolang er nog meer dan 1 cluster over is:
   (a) Selecteer de twee clusters waarvoor de kortst mogelijke verbinding gemaakt kan worden tussen twee vectoren behorende bij elk van de twee clusters. Maak een verbinding tussen beide clusters waarbij de lengte van de verbinding het verschil representeert.
   (b) Voeg de twee clusters samen tot één cluster.

Figuur 5.14 laat het resultaat zien van merge clustering voor de vectoren uit figuur 5.8. De afstand tussen de labels en de verticale lijnen geven de afstand

Figuur 5.14: Voorbeeld data set: merge clustering



Figuur 5.15: Voorbeeld data set: Kohonen kaart met minimal spanning tree

tussen de clusters aan. De afstand tussen de vectoren $\mathbf{x}$ en $\mathbf{y}$ is gedefinieerd als:

$$\sum_i (x_i - y_i)^2$$

Dit is gelijk aan de Euclidean Distance in het kwadraat.

Merge clustering en minimal spanning tree zijn twee elkaar aanvullende manieren voor het zichtbaar maken van verbanden: Vergelijk figuur 5.14 met figuur 5.15, die een minimal spanning tree voor dezelfde dataset laat zien. In figuur 5.14 is te zien dat *boy* en *woman* sterk op elkaar lijken, maar of *wolf* meer lijkt op *boy* of op *woman* is niet te zien. In figuur 5.15 is dit wel te zien, maar echt kleine verschillen vallen niet op. Tot slot is merge clustering veel sneller dan het generen van een Kohonen-kaart, vooral als de vectoren erg lang zijn.

5.8 Verwante architecturen

De Kohonen-kaart combineert ruimtelijke relaties tussen units en competitive learning.

Ruimtelijke informatie wordt ook gebruikt in netwerken die tot taak hebben bepaalde features te ontdekken in een invoer die bestaat uit een tweedimensionaal beeld. Hierbij wordt unsupervised Hebbian learning gebruikt. In een meerlaags netwerk ontvangt elke unit invoer uit een beperkt gebied van de onderliggende laag. Dit wordt het *receptive field* van de unit genoemd. De receptive fields van de units in een laag overlappen elkaar. In simulaties is gebleken dat lagere lagen units het contrast in het invoerbeeld vergroten, terwijl units in hogere lagen leren te reageren op ingewikkelder beeldelementen zoals lijnelementen met een bepaalde oriëntatie. Dit zijn eigenschappen die ook in biologische verwerking van visuele informatie voorkomen. Zie [Lin86], en [HKP91, §8.4] voor een samenvatting.

Competitive learning ([RZ86]) wordt onder andere gebruikt voor het categoriseren van de invoer in clusters, zonder dat er een bepaalde ordening van de clusters plaatsvindt zoals in de Kohonen-kaart. In dit geval zijn er veel meer invoerpatronen dan er units in het netwerk zijn.

In het counterpropagation netwerk ([HN87, HN88, HL88]) wordt competitive learning gecombineerd met supervised learning. Zo'n netwerk bevat een verborgen laag die getraind wordt met competitive learning en een uitvoerlaag die getraind wordt met de delta-regel. Het counterpropagation netwerk kan in principe dezelfde taken leren als een backpropagation netwerk, en in minder tijd. Voorwaarde is dat er voldoende units in de verborgen laag zijn om de invoer in de juiste categorieën te scheiden. In het meest extreme geval is er voor elk invoerpatroon één verborgen unit. In dit geval functioneert het netwerk als een soort opzoek-tabel voor invoer/uitvoer-paren.

Hoofdstuk 6

FGREP: Een automatisch lexicon

Een homogeen neuraal netwerk is niet in staat tot het verrichten van hogere cognitieve taken, zoals taalverwerking en logisch redeneren. Een architectuur bestaande uit onderling samenwerkende neurale modules zou deze taken wel aankunnen. Bij het ontwerpen van zo'n architectuur[1] spelen de volgende vragen een rol ([MD91]):

1. How the overall task should be broken into modules and how the modules should be organized;
2. How PDP subnetworks should be designed so that they can serve as modular building blocks;
3. How communication among such blocks can be established; and
4. How to make use of the modular structure in the training system.

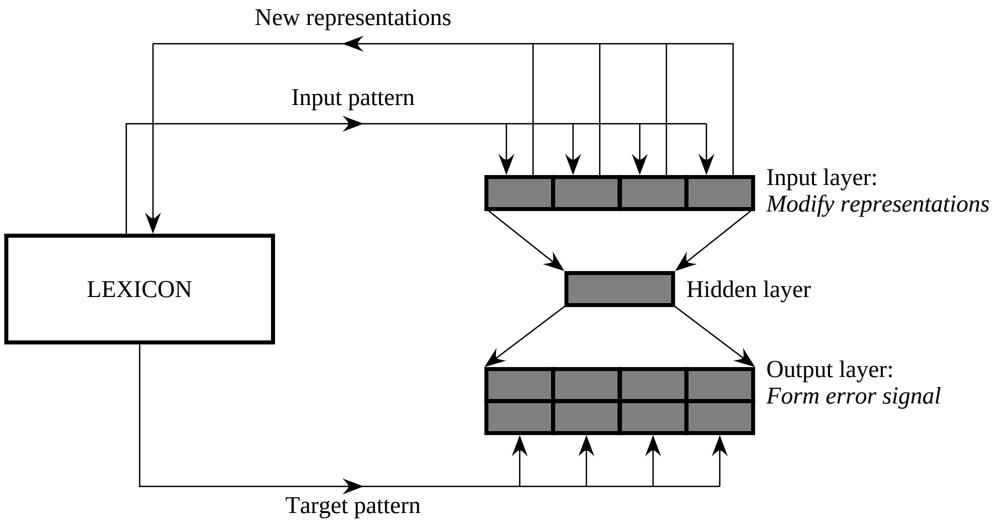
FGREP, dat staat voor *<u>F</u>orming <u>G</u>lobal <u>R</u>epresentations with <u>E</u>xtended Back<u>P</u>ropagation*, hoopt op enkele van deze vragen een antwoord te bieden.

De afbeeldingen in dit hoofdstuk en de bijbehorende teksten zijn uit [MD91].

6.1 Een lexicon voor natuurlijke taal

Een taalverwerkend systeem zal gebruik moeten maken van een lexicon. Stel je voor wat er bij komt kijken wanneer je een handgecodeerd lexicon zou moeten ontwerpen voor het gebruik door neurale netwerken. De meest simpele oplossing is elk woord een unieke codering te geven. Een lokale representatie is ondoenlijk: $(1, 0, 0, \ldots, 0) = aai$, $(0, 1, 0, \ldots, 0) = aak$, $\ldots$, $(0, 0, 0, \ldots, 1) = zymose$. Dit zou gigantische invoer- en uitvoer-vectoren vergen. Met een binaire codering lijkt het wel haalbaar. Met vectoren van lengte 24 zijn al 16 777 216 combinaties mogelijk. Het nadeel van deze werkwijze is dat dit lexicon geen enkele generalisaties bevat die het neurale netwerk zou kunnen benutten, waardoor de complexiteit voor het netwerk explosief toeneemt lang voordat het lexicon een realistische omvang bereikt.

1. De vraag is of het ontwerpen van een autonoom hoog-intelligent systeem ooit mogelijk is. Het kan zijn dat evolutie hierin een onmisbare rol speelt. Zie het hoofdstuk over genetische algoritmen.



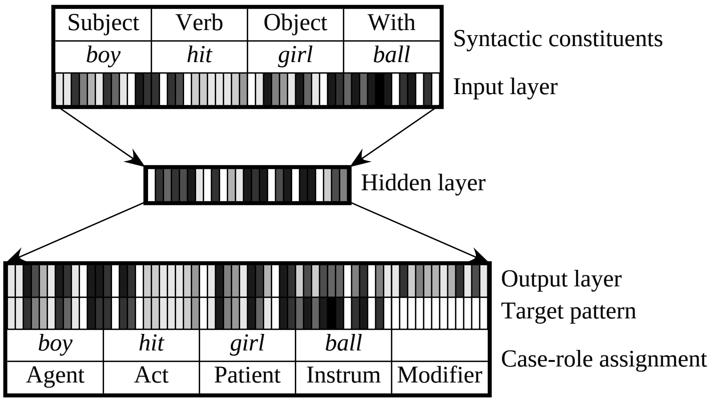
Figuur 6.1: **Basic FGREP architecture.** The system consists of a three-layer backpropagation network and an external, global lexicon which contains the I/O representations. Each input and output assembly (grey rectangle in the figure) holds a single word representation, taken from the lexicon. In the figure, four words are concatenated in the input and output layers. At the end of each backpropagation cycle, the current input representations are modified at the input layer according to the error signal. The new representations are loaded back to the lexicon, replacing the old ones.

Een bruikbaar lexicon zal een codering voor woorden moeten bevatten die rijk is aan informatie. Iedere waarde in de vector zou voor een bepaalde eigenschap kunnen coderen. Denk bijvoorbeeld aan subcategorisaties in symbolische linguïstiek, semantische informatie, idiomatische informatie, maar ook statistische informatie zoals: Hoe waarschijnlijk is het dat dit woord voorkomt in combinatie met een bepaald soort ander woord; hoe dierlijk is dit woord; hoe breekbaar is dit ding. In de traditionele linguïstiek is gebleken dat alle relevante informatie nooit op een symbolische wijze gecodeerd kan worden.

Er blijft maar één oplossing over. Het neurale systeem zal zelf een afdoende codering in het lexicon moeten genereren. Dit is precies wat FGREP doet.

6.2 De werking van FGREP

Figuur 6.1 is een schematische weergave van een FGREP-module. De werking van FGREP is als volgt: Een feedforward netwerk zet een invoer-patroon om in een uitvoer-patroon. Backpropagation wordt niet alleen gebruikt voor het aanpassen van de gewichten, maar ook voor het aanpassen van de woorden in het lexicon. De invoer voor het netwerk wordt gevormd door de concatenatie van vector-representaties van woorden uit het lexicon. De gewenste uitvoer wordt op dezelfde wijze gevormd. Het foutsignaal wordt teruggevoerd door het netwerk voor de aanpassing van de gewichten, maar dit proces stopt niet bij de gewichten.



Figuur 6.2: **Snapshot of Basic FGREP simulation.** The input and output layers of the network are divided into assemblies, each holding one word representation at a time. Each unit in an input assembly is set to the activity value of the corresponding component in the lexicon entry. The input layer is fully connected to the hidden layer and the hidden layer to the output layer. Connection weights are omitted from the figure. If the network has succesfully learned the task, each output assembly forms an activity pattern identical to the lexicon representation of the word filling that role. The correct role assignment is shown at the bottom of the display. This pattern forms the output target for the network. Grey-scale values from white to black are used in the figure to code the unit activities, which vary within the range [0, 1].

Er volgt nog een stap waarin het foutsignaal gebruikt wordt voor de aanpassing van de invoer-patronen zelf. Daarna worden de aangepaste representaties van de woorden teruggeplaatst in het lexicon.

Het lexicon wordt dus gevormd op hetzelfde moment dat het netwerk zijn taak leert. In een complex systeem met meerdere FGREP-modules die allen hetzelfde lexicon gebruiken leren alle modules gelijktijdig, waarbij het lexicon gevormd wordt door de gezamenlijke inspanningen van de afzonderlijke modules samen.

Merk op dat zowel het invoer- als gewenste uitvoer-patroon samengesteld wordt uit het lexicon, maar dat aanpassing van het lexicon alleen gebeurt via het invoer-patroon. Het uitvoer-patroon zal dus gebruik moeten maken van woorden die ook in de invoer gebruikt worden (van deze of een andere FGREP-module), anders worden de woorden voor de uitvoer niet ontwikkeld.

Het is niet ondenkbaar een architectuur te ontwerpen die het lexicon aanpast via de uitvoer-patronen.

6.3 Toepassingen van FGREP

Figuur 6.2 toont een afbeelding van een FGREP-module die gebruikt wordt in een taalverwerkende taak. De module krijgt als invoer een zin (in telegramstijl) en moet als uitvoer de juiste *case-role assignment* genereren. Merk op dat in dit voorbeeld de zin ambigu is. De bal kan gebruikt zijn door de jongen om het meisje te raken, of het meisje was in het bezit van de bal. De ambiguïteit is terug te zien in de uitvoer van het netwerk.

Een meer realistisch model maakt gebruik van sequentiële invoer, waarin de woorden één voor één aan het netwerk worden aangeboden. Het netwerk moet dan de juiste uitvoer hebben gevormd als de invoer is voltooid. Deze taak is uitgevoerd met een recurrente FGREP-module, waarin in plaats van een standaard feedforward netwerk een Elman-netwerk is gebruikt. De invoer is niet langer in telegramstijl, maar bestaat uit complete zinnen, met lidwoorden, voorzetsels en een punt aan het einde. (De punt wordt ook opgenomen in het lexicon.)

DISPAR [MD91], dat staat voor *Distributed Paraphraser*, bestaat uit vier recurrente FGREP-modules. Het systeem leest korte op scripts gebaseerde verhaaltjes, en vertelt die na met aanvulling van informatie die ontbrak in de invoer. De eerste module zet zinnen om in case-role assignments, zoals hierboven is beschreven. De tweede module zet reeksen van deze case-role assignments om in een representatie van een ingevuld script. De derde en vierde module voeren de taken van de tweede en de eerste module uit in omgekeerde volgorde.

DISCERN [Mii93] is een extensie van DISPAR. Het systeem is uitgebreid met extra FGREP-modules en met hiërarchische topografische kaarten. Het systeem kan vragen beantwoorden over eerder gelezen (op scripts gebaseerde) verhaaltjes. De hiërarchische topografische kaarten worden gebruikt als geheugen voor de gelezen verhaaltjes. Het lexicon met daarin de representaties van de woorden wordt verondersteld te worden geïmplementeerd in een gewone topografische kaart.

Hoofdstuk 7

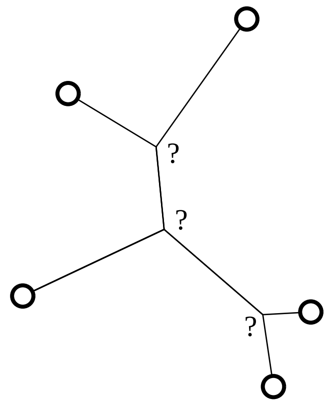
Relax...

Het feedforward netwerk is een veelgebruikt model vanwege zijn eenvoud en efficiëntie. Toch mist dit netwerk een belangrijke eigenschap die neurale netwerken tot een krachtig instrument maken: *relaxatie*.

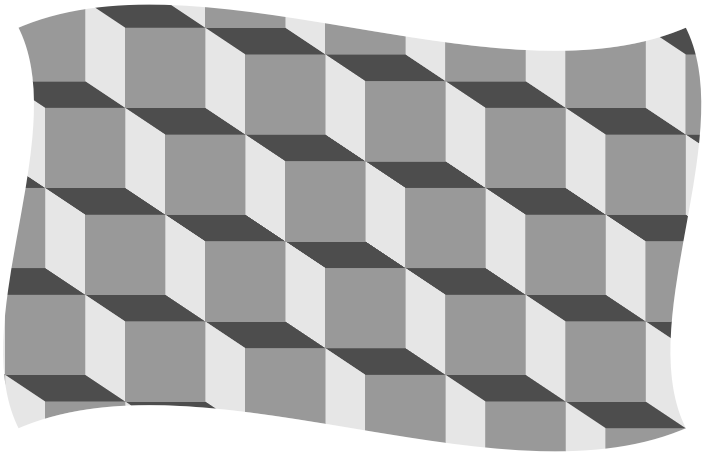
Relaxatie speelt een centrale rol in hoge cognitieve processen, en op kleinere schaal in de interactie tussen netwerken. Maar er zijn ook netwerken die intern van relaxatie gebruik maken. Dit zijn de recurrente netwerken.[1]

Wat is relaxatie? Bekijk figuur 7.1. Hier zijn vijf posities aangegeven door een ring. Deze posities moeten onderling met lijnen worden verbonden. De vraag is: waar moeten de knooppunten geplaatst worden (vraagtekens in de figuur) zodat de totale lengte van de lijnen zo kort mogelijk is? Ook al zijn de exacte coördinaten van de ringen bekend, dan nog is de oplossing voor dit probleem niet eenvoudig te berekenen. Maar met de volgende middelen zou de optimale oplossing te construeren zijn: leidt een elastiek door de ringen, en gebruik vrij beweegbare ringen om het elastiek op de knooppunten bij elkaar te houden. Het elastiek moet zonder wrijving door de ringen kunnen bewegen. De spanning in

1. Hier worden niet de semi-recurrente netwerken zoals het Elman-netwerk bedoeld, maar bijvoorbeeld een netwerk zoals in figuur 4.1 op pagina 17.



Figuur 7.1: Is dit de kortste verbinding?



Figuur 7.2: Blokken van onder af of van boven af gezien

het elastiek zal er voor zorgen dat de kortste verbinding wordt gevormd.[2] De spanning in het elastiek is dan op alle plaatsen gelijk. Het elastiek heeft een toestand bereikt die zo ontspannen mogelijk is, voor zover dit wordt toegelaten door de *constraints* die de ringen vormen.

Een tweede voorbeeld is te zien in figuur 7.2, dat een fragment van een blokken-landschap toont. Voordat een waarnemer de prikkels die in het netvlies ontstaan heeft omgezet in een intern driedimensionaal beeld moet veel gebeuren. De informatie over licht en donker moet gecombineerd worden met kennis over lijnen en vlakken. Vele constraints werken op elkaar in om uiteindelijk een consistente interpretatie te vormen. De afbeelding is ambigu: worden de kubussen van onder af of van boven af bekeken? Als eenmaal de interpretatie voor één kubus is bepaald ligt de mogelijke interpretatie van de andere kubussen vast.

Relaxatie is een proces dat het mogelijk maakt vele bronnen van informatie te combineren om tot een bepaalde conclusie te komen. Dit proces lijkt op unificatie in de computationele linguïstiek ([Shi86]), maar is veel krachtiger. Alle constraints zijn tegelijk werkzaam zodat een zoekproces met backtracking overbodig is. Er wordt gebruik gemaakt van statistische en andere numerieke informatie, terwijl unificatie alleen symbolen kan verwerken. Fuzzy constraints worden gebruikt voor snelle inductie van informatie, terwijl symbolische deductie al gauw een computationeel te complex proces is.

Het *Jets and Sharks* experiment ([MRH86, p. 27], [McC81]) is een aardige demonstratie van relaxatie.

Harmony Theory [Smo86] is een mathematisch model dat ontwikkeld is voor de beschrijving van dynamische systemen zoals die betrokken zijn bij cognitieve processen. Deze tekst wordt sterk aanbevolen.

2. In [FF87] wordt voorgesteld een zeep-oplossing tussen twee glasplaten te gebruiken.

7.1 Het Hopfield netwerk

In recurrente netwerken wordt relaxatie vaak beschreven als het zoeken naar een minimum in een energie-landschap: een patroon van activaties over alle units dat een zo voordelig mogelijke configuratie vormt. De energie van een Hopfield netwerk is gedefinieerd in vergelijking 7.1.

$$H = -\frac{1}{2}\sum_{i,j} w_{ij}a_i a_j \tag{7.1}$$

Hierin is a_i de activatie van unit i, en w_{ij} het gewicht van de verbinding tussen unit i en unit j.

Het Hopfield netwerk is niet gemodelleerd naar biologische netwerken, maar naar eigenschappen van magnetische atomen in een kristalrooster. De verbindingen tussen de units zijn symmetrisch: is er een verbinding van unit i naar unit j, dan is er een verbinding met dezelfde sterkte van unit j naar unit i.

De ontwikkeling van de activatiewaarden in een Hopfield netwerk wordt beschreven door vergelijking 7.2.

$$a_i = sgn(\sum_j w_{ij}a_j) \tag{7.2}$$

De sgn-functie is gedefinieerd in vergelijking 7.3.

$$sgn(x) = \begin{cases} 1 & : \quad x > 0 \\ -1 & : \quad x \leq 0 \end{cases} \tag{7.3}$$

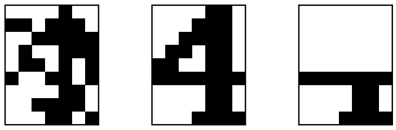
De ontwikkeling in een Hopfield netwerk verloopt als volgt: In willekeurige volgorde worden units geselecteerd. Een geselecteerde unit berekent een nieuwe activatie door middel van vergelijking 7.2. Een verandering van de activatie-waarde leidt tot een lagere totale energie, zoals beschreven in vergelijking 7.1. Het proces eindigt als geen van de activaties van de units nog verandert. Er is dan een lokaal minimum in het energie-landschap bereikt.

7.1.1 Opslag en reconstructie van patronen

Het Hopfield netwerk kan gebruikt worden als geheugen. Er kunnen patronen worden opgeslagen die uit evenveel bipolaire waarden bestaan als er units in het netwerk zijn. Wanneer daarna een verminkt of onvolledig patroon aan het netwerk wordt aangeboden, en het netwerk zich vervolgens mag ontwikkelen tot een energieminimum, is — als het goed is — het originele patroon gevormd. Een voorbeeld is te zien in figuur 7.3. In het netwerk zijn afbeeldingen van cijfers opgeslagen bestaande uit $7 \times 9 = 63$ bits. Het plaatje links is een afbeelding van het cijfer 4 waarvan 16 bits fout zijn. Het plaatje rechts is een onvolledige afbeelding van het cijfer 4. In beide gevallen wordt het origineel — te zien in het middelste plaatje — correct gereconstrueerd.[3]

---

3. Het gebruik van bitmaps van cijfers is slechts bedoeld als demonstratie. Voor een praktische toepassing van neurale netwerken zijn dergelijke representaties van cijfers (en letters) niet bruikbaar.



Figuur 7.3: Reconstructie van patronen

Hoe succesvol het netwerk is in de reconstructie hangt van een aantal zaken af, zoals hoe sterk het aangeboden patroon afwijkt van het te reconstrueren patroon. Het netwerk kan terecht komen in een *spurious state*, een minimum in het energie-landschap dat niet correspondeert met een opgeslagen patroon. Het kan bijvoorbeeld een combinatie zijn van een (oneven) aantal opgeslagen patronen, of het negatief van een opgeslagen patroon.

Het aantal patronen dat in een Hopfield netwerk kan worden opgeslagen is beperkt, en hangt af van de wijze die gebruikt wordt voor de opslag. Een Hopfield netwerk kent geen leerproces, zoals een feedforward netwerk. Een patroon wordt als het ware in één keer vastgelegd in de gewichten. De methoden die hieronder worden besproken zijn incrementeel, dat wil zeggen: de patronen worden één voor één opgeslagen, en het netwerk kan al gebruikt worden voor het reconstrueren van patronen voordat alle patronen zijn opgeslagen.

De meest eenvoudige methode voor het opslaan van patronen wordt *correlation recording* genoemd, en wordt weergegeven door vergelijking 7.4.

$$w_{ij} = \begin{cases} \frac{1}{N} \sum_{\mu} x_i^{\mu} x_j^{\mu} & : \quad i \neq j \\ 0 & : \quad i = j \end{cases} \tag{7.4}$$

Hierin is $\mathbf{x}^{\mu}$ het patroon μ, en x_i^{μ} unit i voor patroon μ. Met correlation recording is de capaciteit van het netwerk sterk beperkt. Wanneer de patronen willekeurig zijn gevormd kunnen maximaal 0,15n patronen worden opgeslagen, waarbij n het aantal units in het netwerk is. Wanneer de patronen niet onafhankelijk zijn loopt de capaciteit sterk terug.

De werking van bovenstaande methode is vrij doorzichtig: als twee units in een patroon dezelfde waarde moeten hebben, verhoog dan het gewicht op de onderlinge verbinding, als de units een tegengestelde waarde moeten hebben, verlaag dan het gewicht van de verbinding. Het lijkt op Hebbian learning (zie pagina 25): als twee units gelijktijdig actief zijn wordt de verbinding tussen beide units versterkt.

Een tweede methode voor het opslaan van patronen in een Hopfield netwerk heet *projection recording*. Deze methode is alleen te begrijpen voor iemand die een goed inzicht heeft in het rekenen met vectoren en matrices. Het komt globaal op het volgende neer: Voor dat een patroon aan het netwerk wordt toegevoegd wordt eerst een 'hypervlak' geconstrueerd, 'gedragen' door de eindpunten van de eerder opgeslagen vectoren. Dit hypervlak heeft een dimensie gelijk aan het aantal draagvectoren, en kleiner dan de dimensie van de vectorruimte (dit is de dimensie van de patronen zelf). Het toe te voegen patroon wordt ontleed in twee vectoren: (1) de projectie van het patroon op het hypervlak en (2) een vector loodrecht op dit vlak. De eerste component is een lineaire combinatie van de eerder opgeslagen patronen, terwijl de tweede component op geen enkele manier is samen te stellen uit combinaties van oudere patronen. Door deze informatie te gebruiken is het mogelijk om even veel vectoren in het netwerk op te slaan als er units in het netwerk zijn, mits de patronen lineair onafhankelijk zijn. (Voor correlation recording geldt de veel strengere voorwaarde dat de patronen een set orthogonale vectoren moeten vormen.) Het nadeel van deze methode is dat alle patronen afzonderlijk bewaard moeten zijn om een nieuw patroon te kunnen toevoegen. In tegenstelling tot correlation recording is projection recording niet biologisch plausibel.

Hier volgt een algoritme voor het opslaan van patronen door middel van projection recording. De werkwijze is uit [Koh89, §6.7.2]. De matrix van gewichten $\mathbf{W}$ bestaat uit nullen wanneer er nog geen patroon in het netwerk is opgeslagen. De verandering van de gewichten voor het toevoegen van een patroon $\mathbf{x}$ is gedefinieerd in vergelijking 7.5.

$$\Delta \mathbf{W} = (\mathbf{x} - \mathbf{W}\mathbf{x}) \frac{\tilde{\mathbf{x}}^{\mathrm{T}}}{||\tilde{\mathbf{x}}||^2} \tag{7.5}$$

Deze vergelijking wordt alleen toegepast voor $\tilde{\mathbf{x}} \neq 0$. Vergelijking 7.6 definieert de berekening van $\tilde{\mathbf{x}}$.

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \sum_{j<i} \frac{(\mathbf{x}_i, \tilde{\mathbf{x}}_j)\tilde{\mathbf{x}}_j}{||\tilde{\mathbf{x}}_j||^2} \tag{7.6}$$

Het subscript geeft het volgnummer van de vector weer, lagere nummers zijn oudere — eerder toegevoegde – vectoren. De summatie geldt alleen voor $\tilde{\mathbf{x}}_j \neq 0$.

Correlation recording en projection recording zijn niet de enige methoden voor het opslaan van vectoren in een Hopfield netwerk. Voor andere methoden zie [Has93] waarin meer over dynamische neurale geheugens is te vinden dan je ooit zou willen weten.

7.1.2 Constraint satisfaction

Het Hopfield netwerk kan gebruikt worden om bepaalde *constraint satisfaction* problemen of optimalisatie-problemen op te lossen. In dit geval worden er geen patronen in het netwerk opgeslagen, maar de constraints worden vertaald naar waarden voor de gewichten in de hoop dat het netwerk daarna een patroon kan construeren dat de optimale, of op zijn minst relatief goede oplossing voor het probleem vormt.

Wanneer het netwerk een verstoord patroon moet reconstrueren moet het netwerk zoeken naar een lokaal minimum in het energie-landschap, het minimum dat het dichtst bij het verstoorde patroon ligt. Voor constraint satisfaction moet het netwerk het globale minimum zien te vinden. De methode gedefinieerd in vergelijking 7.2 en vergelijking 7.3 kan dus niet worden gebruikt.

Er zijn twee methoden voor het vinden van het globale minimum in een Hopfield netwerk.

Stochastische units De units in het netwerk berekenen hun activatie door middel van een stochastische functie:

$$P(a_i = 1) = \frac{1}{1 + e^{-\sum_j w_{ij} a_j / T}} \tag{7.7}$$

Deze functie beschrijft de kans dat de activatie-waarde a van unit i gelijk wordt aan 1. Deze functie wordt vele malen toegepast voor elke unit. De waarde T is een temperatuur, die hoog is in het begin en gelijdelijk afneemt naar 0. De willekeur in het gedrag van de units neemt daardoor gelijdelijk af. Dit gebeurt het eerst voor die units waarop de constraints de grootste invloed uitoefenen.

Continue activatie De activatie van een unit is niet -1 of 1, maar een reële waarde tussen -1 en 1, gegeven door de volgende functie:

$$a_i = \tanh(\beta \sum_j w_{ij} a_j) \tag{7.8}$$

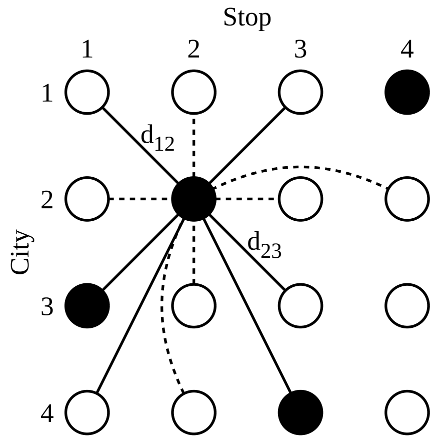
β is klein (vlak boven 0) in het begin, waardoor de activatiewaarden van de units dicht bij 0 blijven. Na verloop van tijd neemt de waarde β toe. Als deze waarde zeer groot wordt begint de functie steeds meer de bipolaire stap-functie te benaderen.

Geen van beide methoden garandeert dat het netwerk het globale minimum vindt.

Een veel geciteerd voorbeeld is het *Travelling Salesman Problem* of TSP. (Het vinden van de kortste route die een aantal steden met elkaar verbindt.) Om het TSP door een Hopfield netwerk te laten oplossen is het nodig het probleem uit te drukken in energie-functie die geminimaliseerd moet worden, zodanig dat deze functie vertaald kan worden naar de gewichten in het netwerk. De details blijven hier achterwege, zie bijvoorbeeld [HKP91]. Hier volgt een meer intuïtieve beschrijving. Figuur 7.4 (uit [HKP91]) toont het netwerk voor een TSP van vier steden. De verticale positie van de units correspondeert met de verschillende steden, de horizontale positie correspondeert met de volgorde in de route. De figuur toont als oplossing de route 3–2–4–1(–3). Alleen de verbindingen voor de tweede stad in de tweede positie in de route zijn getekend. De gestippelde verbindingen zijn negatief. Horizontaal geven deze aan dat elke stad maar één keer in de route mag voorkomen, verticaal geven ze aan dat de reiziger maar één stad tegelijk kan aandoen. Al deze verbindingen zijn even sterk. De doorgetrokken lijnen zijn verbindingen met verschillende gewichten. De waarde van elk gewicht hangt samen met de afstand tussen de beide steden. Zulke verbindingen zijn er alleen naar de voorgaande en de eerstvolgende positie in de route.

7.2 De Boltzmann-machine

Het Hopfield netwerk is zeer krachtig, en patronen kunnen er in worden opgeslagen zonder langdurig leerproces. Het nadeel is dat het aantal patronen dat



Figuur 7.4: Hopfield netwerk voor Travelling Salesman Problem

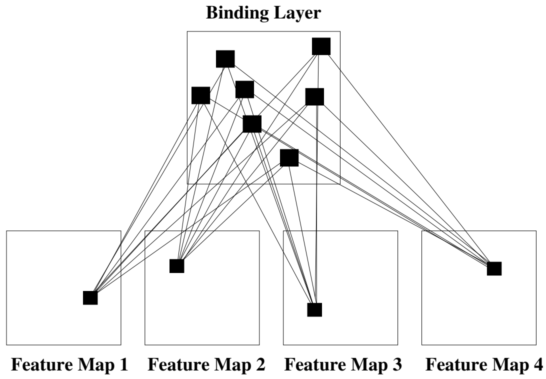
opgeslagen kan worden sterk beperkt is. De capaciteit van het netwerk kan in principe onbeperkt worden uitgebreid door het toevoegen van voldoende units aan het netwerk. Maar wat voor waarden moeten deze extra units hebben voor elk van de te bewaren patronen? Je zit met een probleem dat doet denken aan het vinden van de gewichten voor verborgen units in een meerlaags feedforward netwerk.

Er zijn algoritmen voor het trainen van een Hopfield netwerk met verborgen units. Eén daarvan is Boltzmann-learning, beschreven in [HS86], zie ook [HKP91, §7.1]. Dit algoritme werkt voor netwerken met stochastische units, zoals gedefinieerd in vergelijking 7.7. Het netwerk wordt dan een Boltzmann-machine genoemd. (Verwarrend is dat deze benaming soms ook gebruikt wordt voor stochastische netwerken in het algemeen.)

Een Boltzmann-machine is een zeer krachtig netwerk. Daar staat tegenover dat Boltzmann-learning in een simulatie op een seriële computer extreem veel rekentijd vraagt, zelfs in vergelijking met standaard backpropagation. Dit komt doordat het leerproces vier geneste lussen bevat:

1. In de buitenste lus moeten de waarden van alle gewichten vele malen worden aangepast.
2. Binnen deze lus moet het gemiddelde worden berekend van $S_i S_j$ in een vrijlopend netwerk, en met vaste waarden op de zichtbare units voor elk te leren patroon.
3. Voor elk van deze gemiddelden moet het netwerk een equilibrium bereiken door gebruik te maken van *annealing schedule*, waarbij de 'temperatuur' in het netwerk gelijdelijk wordt verlaagd.
4. Voor elke temperatuur moeten vele units elk vele malen hun activatiewaarden opnieuw bepalen.

Een algoritme genaamd *mean field annealing* [PA87] gebruikt in plaats van stochastische units, units die voldoen aan vergelijking 7.8. Leren door middel van



Figuur 7.5: **Storage.** The weights on the connections between the appropriate feature units and the binding representation of the pattern are set to 1.

deze methode is een stuk sneller dan Boltzmann-learning. Zie [HKP91, p. 171] voor details.

7.3 Convergence-Zone Episodic Memory

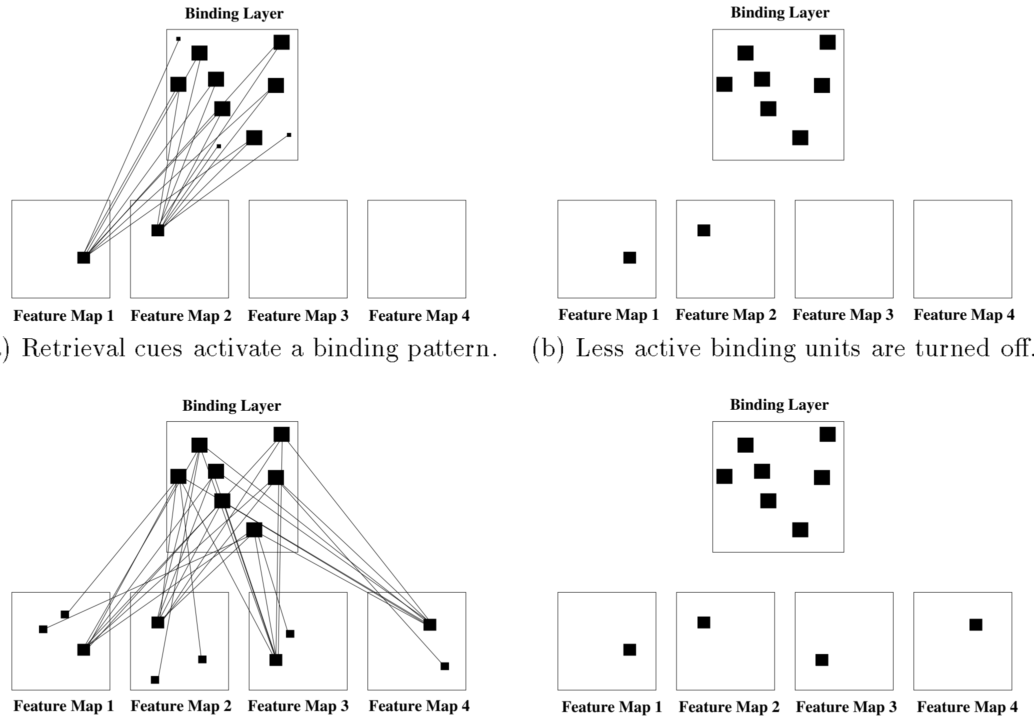
Het Hopfield netwerk en de Boltzmann-machine zijn twee 'basis'-netwerken die zeer uitgebreid zijn bestudeerd. Vele andere modellen zijn ontwikkeld die min of meer zijn geïnspireerd op deze netwerken. De bespreking die volgt is slechts een voorbeeld van één zo'n model.

Het geheugen van de mens is opmerkelijk. In ons leven slaan we zeer veel herinneringen op. Een gebeurtenis hoeft zich maar één keer voor te doen om te worden onthouden, voor het onthouden van wat je is overkomen is geen leerproces nodig of mogelijk. Herinneringen worden tot op zeer hoge leeftijd toegevoegd, terwijl herinneringen aan de vroegste kinderjaren bewaard blijven. Het lijkt alsof ons geheugen nooit vol raakt.

Wat voor kunstmatig netwerk heeft vergelijkbare eigenschappen? Het Hopfield netwerk kan zeer snel patronen toevoegen aan wat eerder is opgeslagen, maar het netwerk is al snel vol. De Boltzmann-machine kan veel gegevens opslaan maar vraagt een traag leerproces. (De auteurs [HS86] veronderstellen wel dat het model biologisch gemotiveerd is, maar voor andere taken.)

Er zijn verschillende netwerk-architecturen ontwikkeld die een model moeten vormen voor een krachtig geheugen. Voorbeelden zijn het *Sparse Distributed Memory* [Kan88], en *Correlation Matrix Memory* [Koh72]. Hier volgt een beschrijving van *Converge-Zone Episodic Memory* [MM95].

Figuur 7.5 toont het model. Bovenaan is een *binding layer* of *convergence-zone*. Deze heeft symmetrische verbindingen met een rij *feature maps* daaronder. De gewichten van deze verbindingen zijn 0 of 1. De feature maps zijn vergelijkbaar met de topografische kaart van Kohonen. Er is een aparte kaart voor elke feature, zoals kleur, vorm, afmeting. Een 'gebeurtenis' wordt aangeboden als een patroon



(a) Retrieval cues activate a binding pattern. (b) Less active binding units are turned off.

(c) Binding pattern activates feature units. (d) Less active feature units are turned off.

Figuur 7.6: **Retrieval.** A stored pattern is retrieved by presenting a partial representation as a cue. The size of the square indicates activation level of the unit.

van waarden voor features, zoals kleur=rood; vorm=rond. Op elke feature map is één unit actief. Wanneer een patroon moet worden opgeslagen in het geheugen wordt een zeer klein aantal units in de binding layer actief. De gewichten tussen de actieve units in de binding layer en de actieve features worden op 1 gezet.

In figuur 7.6 is te zien hoe een herinnering wordt gereconstrueerd. Een gedeeltelijke herinnering wordt aangeboden aan de feature maps, die op hun beurt zorgen voor activaties in de binding layer. Dit is te zien in figuur (a). Vervolgens worden de units in de binding layer met een lage activatie uitgeschakeld, zoals in figuur (b). Vervolgens activeren de actieve units in binding layer de ontbrekende features in de feature maps, zie figuur (c). Tenslotte worden de minder actieve units in de feature maps uitgeschakeld zodat alleen de meest actieve units over blijven. Dit is te zien in figuur (d).

Het aantal patronen (herinneringen) dat kan worden opgeslagen is groot in vergelijking met het aantal units in de binding layer. Experimenten leverden de volgende resultaten: Voor vier feature maps met 5 000 units, een binding layer met 60 000 units en 125 actieve units in de binding layer per patroon konden 70 000 patronen opgeslagen worden, die voor 99% correct werden gereconstrueerd, wanneer de patronen met 3 van de 4 features werden opgeroepen.

Hoofdstuk 8

Genetische algoritmen, en meer...

Genetische algoritmen zijn evenals neurale netwerken geïnspireerd op de natuur. Evolutie van soorten door middel van ontwikkeling van erfelijk materiaal vormt de basis. In genetische algoritmen draait alles om drie aspecten:

Mutaties in het erfelijk materiaal zorgen er voor dat er soorten met nieuwe eigenschappen ontstaan.

Recombinatie van erfelijk materiaal laat nakomelingen ontstaan die combinaties van eigenschappen van verschillende ouders bezitten.

Selectie zorgt er voor dat individuen met gunstige (combinaties van) eigenschappen, zoals die zijn ontstaan door mutatie en recombinatie, een grotere kans hebben deze eigenschappen door te geven aan hun nakomelingen.

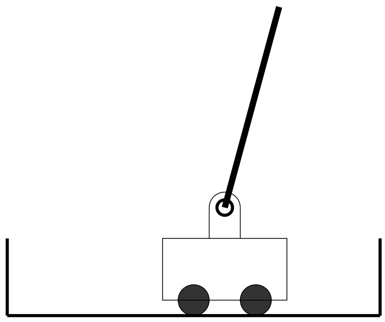
8.1 Genetisch algoritmen

Genetische algoritmen (GA's) werden ontwikkeld door John Holland als een zoektechniek gebaseerd op evolutie volgens Darwin. ([Hol92, Gol89])

Een GA kan gebruikt worden voor het ontwikkelen van een gunstig ontwerp. Hierbij gaat het er om de juiste combinatie van eigenschappen voor het ontwerp te vinden. Hiervoor is het noodzakelijk dat deze eigenschappen te coderen zijn door middel van een reeks symbolen. Elk kenmerk dient gecodeerd te worden in een patroon van meerdere symbolen. De specifieke combinatie van symbolen bepaald het 'uiterlijk' of de 'waarde' van het kenmerk. Al deze patronen samen vormt het erfelijk materiaal dat gebruikt wordt in het GA.

Het GA bestaat uit de volgende stappen:

1. Ontwerp een string-codering voor de eigenschappen van een individu: de interpretatie van de *genetische code*.
2. Stel een populatie samen van individuen met willekeurige waarden in het erfelijk materiaal. Het erfelijk materiaal van één individu wordt een *genotype* genoemd.
3. Genereer elk individu aan de hand het genotype. Het individu met zijn uiterlijke eigenschappen wordt een fenotype genoemd. Test dit fenotype en geeft het een numerieke score.



Figuur 8.1: Balanceren van een paal op een rijdend wagentje

4. Creëer een nieuwe populatie, door genotypen uit de oude populatie te kopiëren, waarbij de kans dat een genotype gekozen wordt het grootst is voor corresponderende fenotypen met de hoogste score. Ga door tot de nieuwe populatie even groot is als de oude, en vervang de oude populatie door de nieuwe.
5. Selecteer een aantal keer twee willekeurige genotypen in de populatie en pas recombinatie toe. Doe dit door middel van *crossover*: Verwissel een deel van de string met erfelijk materiaal van één genotype met het corresponderende deel van de string van het andere genotype.
6. Selecteer een aantal keer een willekeurig genotype uit de populatie en pas *mutatie* toe. Doe dit door op een willekeurige plaats in de string met erfelijk materiaal de waarde te veranderen. Voor een gunstige ontwikkeling is het aantal mutaties vaak erg klein, veel kleiner dan het aantal crossovers.
7. Herhaal stap 3 tot en met 6 net zolang tot een fenotype wordt aangetroffen die goed genoeg is, of totdat de gemiddelde score van de populatie niet meer toeneemt.

8.2 Neuro-Evolutie

Wanneer GA's gebruikt worden voor het ontwikkelen of trainen van neurale netwerken wordt wel van neuro-evolutie gesproken. Een GA kan gebruikt worden om een feedforward netwerk te trainen wanneer geen directe feedback uit de omgeving beschikbaar is om een leerproces zoals backpropagation te sturen, maar er slechts een globale evaluatie-functie voor het netwerk is. Niet elke uitvoer van het netwerk produceert een foutsignaal. In plaats daarvan wordt de prestatie van het netwerk bepaald nadat het netwerk een lange reeks van invoersignalen in uitvoersignalen heeft omgezet. In dit geval kan het netwerk getraind worden met een methode die *reinforcement learning* wordt genoemd. Een GA kan als reinforcement learning worden gebruikt.

Een voorbeeld wordt geïllustreerd in figuur 8.1 (afbeelding uit [MMss]). Op een wagentje staat een paal die kan omvallen naar links of rechts, in de rijrichting van het wagentje. Het wagentje kan slechts een klein stuk heen en weer rijden. Het probleem is het wagentje zo te besturen dat voorkomen wordt dat de paal omvalt. Met een GA kan een netwerk geleerd worden deze taak uit te voeren. Het netwerk krijgt vier gegevens als invoer: de huidige positie en snelheid van het wagentje, en de huidige hoek en hoeksnelheid van de paal. Het netwerk moet als uitvoer aangeven of het wagentje naar links of naar rechts moet bewegen. Dit wordt ongeveer vijftig keer per seconde herhaalt.

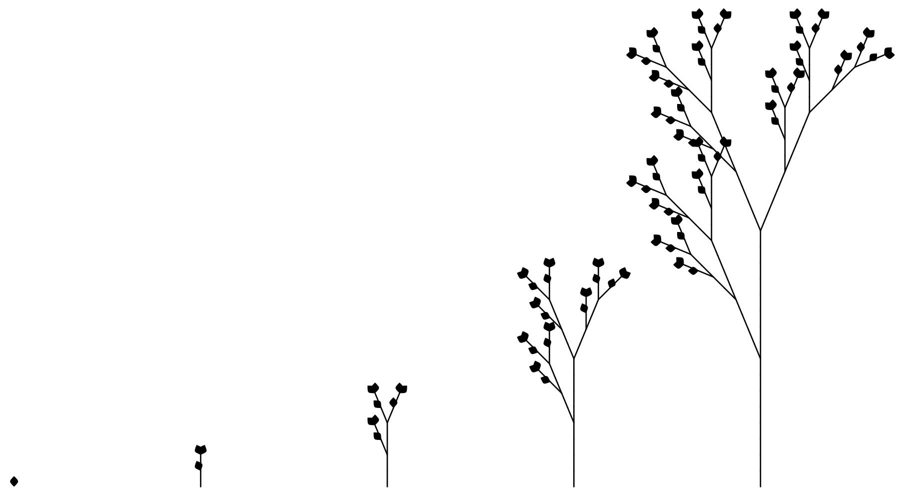
Het GA maakt gebruik van een populatie van genotypen die elk voor een netwerk coderen. Elk genotype is een verzameling van waarden voor de gewichten van een netwerk: het fenotype. Een fenotype wordt hoger gewaardeerd naarmate het netwerk langer in staat is met het wagentje de paal te balanceren. Het GA reproduceert steeds nieuwe, betere netwerken. (Er is dus geen individueel lerend netwerk.)

Een GA kan evenals andere zoektechnieken vast komen te zitten in een lokaal optimum. In dit geval wordt de populatie overheerst door één sub-optimale oplossing, zonder dat het optimum bereikt kan worden door recombinatie en mutatie. Moriarty [MMss] heeft een variant GA ontwikkeld dat dit probleem omzeilt. Deze methode wordt *Symbiotic, Adaptive Neuro-Evolution* genoemd, kortweg SANE. In plaats van een populatie van netwerken wordt er een populatie van verborgen units geëvolueerd. Voor elke generatie worden willekeurig units (fenotypen) samengevoegd tot een netwerk. Elke unit wordt een aantal keer gebruikt in verschillende samenstellingen van netwerken. Een unit wordt hoger gewaardeerd naarmate de netwerken waarin de unit deelneemt beter presteren. Met deze methode wordt voorkomen dat één variant de populatie van units gaat overheersen omdat niet één unit in staat is de taak alleen uit te voeren. Units zijn afhankelijk van andere units, met andere gewichten. Er is een symbiotische relatie tussen de individuen in de populatie. Moriarty heeft deze methode uitgetest in het boven beschreven balanceer-probleem en vergeleken met een aantal andere methodes. Symbiotische evolutie bleek relatief snel en effectief.

8.3 Ontwikkeling

Alhoewel in het voorgaande verschillende keren de term evolutie werd gebruikt is er van echte ontwikkeling geen sprake. Het waren slechts leermethodes. De interpretatie van het genotype ligt vast, het GA wordt gebruikt om de juiste waarden van een aantal variabelen te vinden. Het genotype codeert niet voor eigenschappen, maar alleen voor waarden van vaststaande eigenschappen.

Ontwikkeling gebaseerd op evolutie moet het mogelijk maken nieuwe eigenschappen en structuren te laten ontstaan. Dit kan niet met het standaard GA van Holland. Het is onder andere nodig dat bij de reproduktie van individuen genotypen ontstaan met meer of minder genetisch materiaal, bijvoorbeeld door invoeging of verwijdering van materiaal, of door crossover op ongelijke punten van twee ouders. De interpretatie van de genetische code kan nu niet langer van



Figuur 8.2: Vijf groeistadia van plant A

buitenaf komen. Het genetisch materiaal moet geschreven zijn in een taal die op eigen kracht het individu kan creëren.

L-Systemen [PL90] vormen een eenvoudig voorbeeld van het gebruik van een code voor het genereren van figuren. Figuur 8.2 toont vijf fasen uit de groei van een plant. De code die de ontwikkeling van deze plant genereert bestaat uit twee herschrijfregels:

knop → steel [+ knop] steel [+ knop] − knop
steel → steel steel

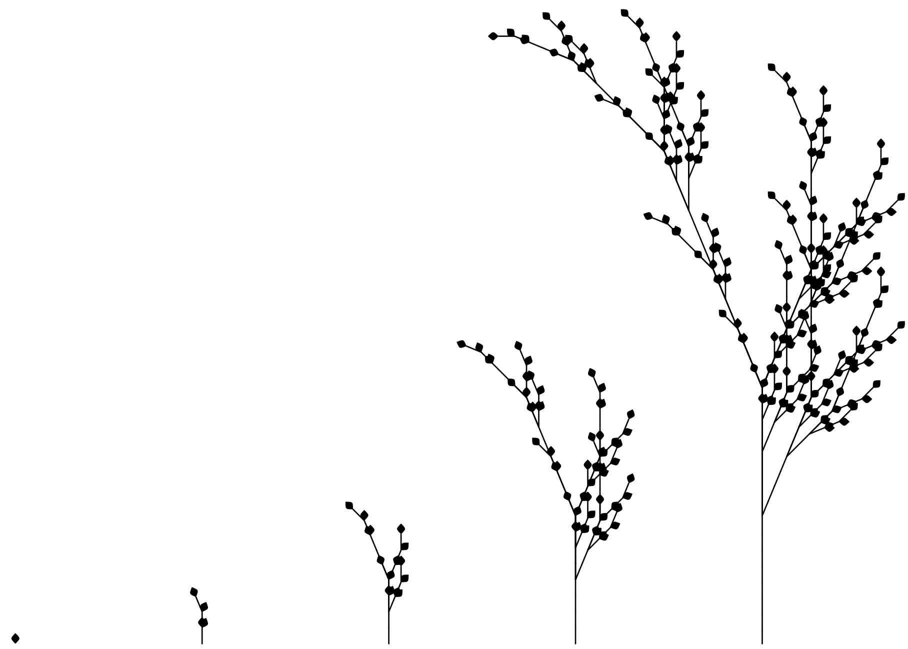
De eerste regel geeft aan wat er uit een knop groeit. De tweede regel geeft aan hoe een steel zich verder ontwikkeld. '[' en ']' geven begin en eind van een aftakking aan, '+' en '−' geven een hoek naar links en naar rechts aan. De eerste fase bestaat uit alleen een knop. Elke volgende fase ontstaat uit de vorige fase door eenmaal de herschrijfregels toe te passen.

Figuur 8.3 laat vijf groeistadia van een andere plant zien. De codering voor deze plant is als volgt:

knop → steel [knop] [− knop] steel [+ steel knop] − knop
steel → steel steel

Hoewel de gebruikte soorten symbolen dezelfde zijn — alleen het aantal en de volgorde is anders — verschilt het uiterlijk van de tweede plant nogal van dat van de eerste.

Het ultieme voorbeeld van een generatieve genetische code is uiteraard het biologische DNA. In DNA worden codes weergegeven door reeksen van slechts vier basen, aangegeven met de letters A, C, T, en G. Groepen van drie basen coderen voor een aminozuur. Er zijn dus in principe codes voor 64 aminozuren, maar niet alle combinaties worden gebruikt. Sommige combinaties markeren het begin of het eind van een reeks, sommige aminozuren worden door meerdere combinaties gecodeerd. Binas ([VBK+86, p. 157]) meldt 20 aminozuren. De aminozuren zijn



Figuur 8.3: Vijf groeistadia van plant B

de bouwstenen van de eiwitten. Het aantal mogelijke combinaties in reeksen aminozuren, en dus het aantal eiwitten is gigantisch, en elk eiwit speelt op een of andere wijze een rol in het ontstaan van een eigenschap.

Wil je genetische algoritmen gebruiken voor de ontwikkeling van complexe netwerk-structuren bestaande uit sub-netwerken, dan zul je een genetische codering moeten gebruiken die vergelijkbaar expressief is als bijvoorbeeld een L-Systeem of biologisch DNA. Alleen dan kun je echte neuro-evolutie toepassen.

8.4 Synthetisch leven

De meest radicale toepassing van kunstmatige evolutie vindt plaats in TIERRA, een virtuele computer ontwikkeld door Thomas Ray.

De bestudering van het ontstaan en de ontwikkeling van het leven op aarde is problematisch. Ten eerste is de geschiedenis van dit proces slechts zeer beperkt te reconstrueren aan de hand van fossielen. Het verloop van de evolutie is niet direct te volgen. Ten tweede is al het biologisch leven op aarde het gevolg van slechts één evolutionair experiment. Hierdoor valt zeer weinig te zeggen over welke ontwikkelingen toevalligheden waren, en welke ontwikkelingen wetmatig zijn in evolutie. Met TIERRA heeft Ray een systeem ontworpen dat gebruikt kan worden om evolutie-processen met elkaar te vergelijken.

TIERRA is een kunstmatige leefwereld in de computer, bevolkt door zelf-replicerende programma's. Deze bewoners van de computer zijn onderworpen aan de wetten van de logica, opgelegd door de instructie-set van de CPU. De leefwereld is dus radicaal anders dan de 3-dimensionale leefwereld van biologische wezens, waarin alles onderworpen is aan natuurkundige en chemische wetten. Dit is met opzet.

Thomas Ray probeert niet het ontstaan van het eerste leven te bestuderen, maar de ontwikkeling van primitieve wezens naar complexe wezens. Hij hoopt vooral omstandigheden te creëren waarin een explosie van diversificatie kan plaats vinden, vergelijkbaar met de Cambrische explosie op aarde, 600 miljoen jaar geleden. Het eerste wezen dat TIERRA bevolkte was een programma van 80 machine-instructies, geschreven door Ray, dat niets anders deed dan kopieën van zichzelf maken.

In TIERRA zijn er geen externe mechanismen die zorgen voor evaluatie en reproduktie van de individuen, zoals in GA's. Deze mechanismen zijn ingebouwd in de organismen zelf. Bovendien zijn de organismen niet geïsoleerd van elkaar, maar leven ze samen in één milieu, het RAM van TIERRA, waarin ze elkaar kunnen waarnemen en op elkaar kunnen reageren. Hierdoor is de evolutie een zichzelf stimulerend proces doordat het leefmilieu als gevolg van de evolutie van de bewoners steeds complexer wordt.

TIERRA is een virtuele computer die in veel opzichten lijkt op onze traditionele computer zoals de PC, maar die op enkele belangrijke punten afwijkt:

- Een gewone computer maakt gebruik van operatoren met operanden die samen 16 of 32 bits gebruiken, waarbij dus 65 536 of 4 294 967 296 bewerkingen mogelijk zijn. De kans dat een mutatie in zo'n instructie een nuttige variant oplevert is vrijwel uitgesloten. In TIERRA bestaat de instructie-set uit slecht 32 bewerkingen, zonder operanden.
- Een standaard machinetaal maakt gebruik van een numerieke adressering van onder andere subroutines. In TIERRA wordt gebruik gemaakt van *templates*. Zo wordt een jump-instructie in een programma vergezeld van een code die het complement vormt van een code die voorafgaat aan de reeks instructies waarnaar gesprongen moet worden. Het is de taak van de CPU de passende template in het geheugen te zoeken. Het programma hoeft dus niet de exacte plaats van een subroutine te weten.
- In een gewone computer worden vrijwel geen fouten gemaakt. In TIERRA wel. Ten eerste zijn er willekeurige veranderingen in het geheugen van de computer. Wanneer deze mutaties plaats vinden in de code van een programma dan zijn deze veranderingen overerfbaar. Ten tweede is de CPU onnauwkeurig in het uitvoeren van instructies. Zo kan bijvoorbeeld de instructie om de waarde van een register met 1 te verhogen het resultaat hebben dat het register met 0 of met 2 wordt verhoogd. Dit leidt onder andere tot fouten in het reproduktie-proces dat door de individuele programma's wordt uitgevoerd.
- In TIERRA wordt de tijd die de CPU beschikbaar heeft verdeeld over alle in het geheugen aanwezige programma's, of zelfs over subprogramma's van 'meercellige' organismen. (In een parallelle machine kan voor elk

programma of programma-cel één processor worden gebruikt.) Geheugen wordt toegewezen aan programma's die daar om vragen om een kopie van zichzelf te kunnen maken. Geactiveerde kopieën worden geregistreerd als nieuwe programma's. Wanneer het geheugen vol begint te raken zorgt TIERRA ervoor dat oude programma's worden opgeruimd.

Thomas Ray werd verrast door de grote diversiteit aan programma's die TIERRA bevolkte gedurende het eerste experiment. Uit het eerste programma ontstonden, behalve kleinere programma's die de taak van het zichzelf kopiëren efficiënter uitvoerden onder andere:

- Parasieten die gebruik maakten van subroutines in andere organismen. Dit is niet direct schadelijk voor de non-parasitaire programma's, maar wel indirect doordat de parasieten (door hun kleinere omvang) sneller reproduceren en zo veel van het schaarse geheugen voor zich opeisen.
- Non-parasitaire programma's met immuniteit tegen de parasieten.
- Nieuwe parasieten die de immuniteit van de vorige programma's doorbraken.
- Programma's die de parasieten misleidden waardoor de parasieten niet zichzelf reproduceerden, maar de nieuwe programma's.
- Programma's die konden samenwerken met soortgenoten.
- Programma's die vals speelden bij de samenwerking, waardoor ze zichzelf bevoordeelden ten koste van de samenwerkende programma's.

Al deze manieren van overleven zijn bekend in de biologische wereld.

Thomas Ray uit bezwaren tegen het gebruik van kunstmatige neurale netwerken die zeker het overwegen waard zijn. Neurale netwerken zijn het biologische antwoord op eisen die gesteld werden in een 3-dimensionale wereld beheersd door natuurkundige en chemische wetten. Volgens Ray zijn niet de neurale netten essentiëel voor intelligentie, maar de evolutie. In een logische leefwereld met een radicaal andere ruimtelijke structuur zoals in de computer worden heel andere eisen gesteld. De optimale antwoorden zullen heel anders zijn, en alleen door evolutie kunnen deze antwoorden worden gevonden. Intelligentie die op deze wijze ontstaat in de computer zal gevormd zijn door het leven in de computer, en sterk afwijken van onze intelligentie. Ray acht het niet ondenkbaar dat er complexe sterk parallelle programma's ontstaan, in intelligentie vergelijkbaar met ons niveau, zonder dat we elkaar zouden kunnen begrijpen, door het radicaal andere karakter van de intelligentie.

We beschikken over massief parallelle computers, waarin elke processor een eigen sub-programma kan uitvoeren. Deze computers hebben een potentiële verwerkings-capaciteit die vergelijkbaar is met die van complexe neurale netwerken, maar we zijn niet in staat om programma's voor deze computers te schrijven. Met behulp van systemen zoals TIERRA zouden dergelijke programma's ontwikkeld kunnen worden.

De eerste beschrijving van TIERRA is te vinden in [Ray91]. In [Ray92] doet Ray een avontuurlijk verslag van zijn ontwikkeling van TIERRA. Het artikel [Ray94a] gaat uitvoerig in op de achtergronden van TIERRA. Het gebruik van meercellige organismen in TIERRA wordt beschreven in [TR]. In 1994 deed Ray in [Ray94b] het voorstel een TIERRA netwerk op te zetten op het internet. De eerste vorderingen van dit project worden beschreven in [Cha95].

Veel van deze artikelen en de source codes voor de TIERRA simulator zijn verkrijgbaar door middel van anonymous ftp van `tierra.slhs.udel.edu`. Zie ook `http://vrml.arc.org/tierra/index.html`.

Hoofdstuk 9

Grondverf

Wat is de plaats die neuro-informatica inneemt — of zou moeten innemen — binnen *artificial intelligence*? Dit hangt af van de doelstellingen die je definieert voor het vakgebied van de AI.

De *strong* AI heeft als einddoel het creëren van een intelligente machine. Niet alleen een machine die bepaalde moeilijke taken in een beperkt domein verricht, zoals een schaak-computer of een expert-systeem, maar een machine die echt denkt, een zelfbewustzijn heeft, en even veelzijdig is als de mens. De *traditionele* of *standaard* AI denkt dit doel te verwezenlijken door er van uit te gaan dat denken een proces is van verwerking van formele symbolen. In onderzoek in deze richting is steeds meer de nadruk komen te liggen op representatie en verwerking van kennis.

John Searle [Sea80] bestrijdt met behulp van zijn bekende parabel van de Chinese kamer dat computers ooit kunnen denken. Zijn argumenten zijn gericht tegen de werkwijze van de standaard AI. Diane Law stelt dat systemen gebaseerd op neurale netwerken tegemoet komen aan de tekortkomingen van symbolische AI. Haar artikel [Law94] dient naast de tekst in dit hoofdstuk gelezen te worden. (Het artikel [LM94] vormt een aardige illustratie.)

Neurale netwerken zijn op zich leuke dingen met aardige (kleine) toepassingen waar je heel je leven aan kunt studeren. Dan wordt een netwerk vaak gezien als iets wat een invoer omzet in een uitvoer, zonder dat er echt gekeken wordt naar de omgeving van het netwerk. Hoe de invoer voor het netwerk tot stand komt, of hoe de uitvoer van het netwerk de omgeving beïnvloed speelt geen enkele rol. Bij deze werkwijze heeft neuro-informatica even weinig met intelligentie te maken als informatica.

Je kunt neurale netwerken zien als een instrument voor het oplossen van problemen binnen de traditionele AI waar de AI met symbolische technieken tekort schiet.

Wil je hogere wetenschap bedrijven dan zul je fundamentele vragen moeten stellen over de werkwijze binnen de AI. Uwe Schnepf verwoordt ideeën van Rodney Brooks [Sch91, p. 468]:

> He claimed that, although the representation and the models of the world and the reasoning processes based on them have been a

> central issue in AI over the last 20 years, there is no evidence that our mental capabilities are actually based on these processes.

En verder:

> He argued that, looking at the evolution of intelligence during the billions of years of biological development, it seems that the (in evolutionary terms) recent emergence of cognitive abilities such as problem solving, language, expert knowledge etc. are relatively simple once the substrate of being and reacting in the real world is provided.[1]

De oorzaak van het falen van de standaard AI wordt gelegd bij de top-down benadering in de AI. Door intelligentie los te zien van zaken als gedrag, ontwikkeling en interactie met de omgeving worden foutieve veronderstellingen gebruikt. Het vakgebied dat wordt aangeduidt met de term *artificial life* begint van de andere kant. De zelfredzaamheid van eenvoudige organismen in een complexe dynamische omgeving wordt gezien als de basis van intelligentie.

Dave Cliff [Cli91] stelt dat het bestuderen van geïsoleerde neurale netwerken even onzinnig is als de traditionele AI, als het gaat om de modellering van intelligent gedrag. Een neuraal netwerk dat niet geïntegreerd is in een sensorimotorsysteem is als cognitief model zonder waarde. Cliff pleit voor een benadering die hij *computational neuroethology* heeft gedoopt.

In de AI worden vaak complexe hiërarchische regelsystemen verondersteld, compleet met een soort centrale uitvoerende macht voor het bereiken van ingewikkelde vormen van gedrag. Het wekt vaak verbazing dat een simpel neuraal netwerk een proces dat aan bepaalde regels lijkt te zijn onderworpen kan uitvoeren zonder gebruik te maken van die regels. De neiging van de mens te geloven in centrale machten is zeer sterk. Mitchel Resnick [Res91, p. 535] omschrijft het als volgt:

> ...when people see self-organizing phenomena in the world, they tend to assume centralized control where none exists. When people see a flock of birds or a school of fish, many of them assume (incorrectly) that there must be a leader of the group. And when people see the diversity of complex creatures in the world, many of them assume (incorrectly) that there must be a single Designer of Life. This "centralized mindset" is quite resistant to change. People need better models and metaphors for thinking about self-organization.

P. Hogeweg [HH91] heeft gedemonstreerd dat de oorzaken van complex gedrag veel eenvoudiger kunnen zijn dan vaak wordt verondersteld. Complex (op het oog planmatig) groepsgedrag blijkt een emergente eigenschap van een verzameling individuen met simpele gedragsregels.

1. Waarom is er van die vele diersoorten die *the substrate of being and reacting in the real world* hebben verworven slechts één soort die de *relatively simple* stap naar taal en problem solving heeft gezet? Of zijn wij toch niet zo uniek?

Nawoord

Vanaf het begin van mijn studie alfa-informatica — kunstmatige intelligente en natuurlijke taalverwerking — ben ik nieuwsgierig geweest naar neurale netwerken. Deze nieuwsgierigheid is niet bevredigd. Mijn belangstelling voor neurale netwerken is tijdens m'n studie gegroeid, niet door de aandacht die het onderwerp kreeg (mijn kennis over neurale netwerken heb ik voornamelijk verworven door zelfstudie), maar door m'n teleurstelling over de traditionele aanpak van kunstmatige intelligentie en computationele linguïstiek.

De ontwikkeling van de seriële programmeerbare computer — de Von Neumann computer — en de opkomst van de kunstmatige intelligentie hebben zich niet toevallig min of meer gelijktijdig afgespeeld. In vroegere tijden werden de menselijke hersenen 'begrepen' als een complex uurwerk. Later, met de ontwikkeling van de telegrafie en telefonie, ontstond het beeld van de hersenen als een geavanceerd telefonisch schakelsysteem. Gedurende de laatste halve eeuw stond wederom de techniek model voor ons begrip van de intelligentie. De hersenen worden gezien als een machine die programma's van logische bewerkingen gebruikt om kennis die opgeslagen ligt in een regelsysteem te verwerken. Wetenschappers doen hun best modellen van deze regels en programma's op te stellen.

Kunstmatige intelligentie

In de kunstmatige intelligentie wordt sterk geleund op de scheiding tussen kennis en kennisverwerking. Kennis is declaratief, vaak hiërarchisch. Een logisch redeneer-systeem wordt gebruikt om nieuwe gegevens te combineren met bestaande kennis om beslissingen te nemen. Dit leidt tot gigantische combinatorische problemen, wat bekend is als het *frame* probleem [MH69]: hoe wordt uit alle beschikbare kennis de voor een bepaalde situatie relevante informatie gefilterd?

Wat niet alleen voor mensen, maar ook voor 'lagere' diersoorten als vanzelfsprekend geldt, zoals: *van water wordt je nat*, of: *als je in een gat stapt val je* moet voordat een computer er iets mee kan worden opgeslagen als een complex systeem van predicaten waarin zeer elementaire kennis-atomen of primitieven worden gecombineerd. En dan nog hebben deze primitieven geen enkele betekenis voor de computer. Effectief redeneren met deze zeer complexe kennis over de wereld is voor de snelste supercomputers onbegonnen werk. Een mens of een muis zou over dit ongekende logische vermogen beschikken? En zelfs over het vermogen om dit complex van logisch informatie af te leiden en op te slaan? Dit acht ik uitgesloten.

Natuurlijke taalverwerking

In de taalkunde strijden diverse theorieën om de voorrang: Government Binding, Lexical Functional Grammar, Head-Driven Phrase Structure Grammar, om er maar een paar te noemen. De nieuwere theorieën zijn vaak wat minder absurd dan de oudere transformationele theorieën, maar het detail waarin elke nieuwe theorie wordt uitgewerkt is lachwekkend. Het is alsof natuurkundigen het achtste cijfer achter de komma van een of andere fysische constante proberen te bepalen terwijl er nog geen enkele overeenstemming is over het eerste cijfer achter de komma.

Wat alle theorieën gemeen hebben is dat zowel de verwerving als de verwerking van taal ongekend complex is. Parsers voor grammatica's van natuurlijke taal die een complexiteit in tijd en ruimte hebben die evenredig is met de derde of zelfs vierde macht van de lengte van de zin worden gewoon gevonden — terwijl voor de mens de complexiteit van taal recht evenredig is met de lengte van de zin. Alle theorieën hebben zeer te lijden onder ambiguïteiten in taal – ook dit is geen probleem voor menselijke taalgebruikers. Alle gevestigde taalkundige theorieën gaan nog uit van het bestaan van lege categorieën — terwijl Pickering en Barry [PB91] overtuigend hebben aangetoond dat mensen geen lege categorieën gebruiken.

Taalkundige theorieën zijn meer en meer gebaseerd op het gebruik van herschrijfregels en declaratieve taalkennis. De problemen die zich daarbij voordoen lijken sterk op die van de op regels gebaseerde modellen in de kunstmatige intelligentie: hoeveel kennis moet het systeem bezitten, hoe moet deze gestructureerd worden, en hoe wordt efficiënt de juiste informatie geïntegreerd.

Misschien is het grootste manco van alle taalkundige theorieën dat daarin taal los wordt gezien van de taalgebruiker. Er is geen enkele link tussen het symbolische systeem en de rol die de taal speelt in de interactie tussen de mens en zijn omgeving. De theorieën hebben geen oog voor de mechanismen die de taal gemaakt hebben tot wat zij is.

Symbolisme versus sub-symbolische interactie

De genoemde problemen zijn niet zomaar problemen die om een oplossing vragen. Ze wekken de suggestie dat de fundamenten waarop de theorieën zijn gebouwd niet deugen.

Volgens mij is het grootste obstakel voor het begrijpen van wat taal en intelligentie is het idee van symbolisme — het idee dat denken en begrijpen een proces is van symboolverwerking.

Intelligentie is geen vermogen tot redeneren, maar een proces van reactie van mens of dier op de omgeving waarbij een interne representatie van de omgeving wordt gevormd. De manier waarop we onze omgeving zien en hoe we er over denken is bepaald door de wijze waarop we hebben geleerd te overleven in een wereld die in principe te complex is om in z'n geheel te kunnen worden bevat. En over het menselijk denkvermogen zegt Arnheim ([Arn69] geciteerd in [FF87]):

> My contention is that the cognitive operations called thinking are not the privilege of mental processes above and beyond perception but the essential ingredients of perception itself. I am referring to such operations as active exploration, selection, grasping of essentials, simplification, abstraction, analysis and synthesis, completion, correction, comparison, problem solving, as well as combining, separating, putting into context. These operations are not the prerogative of any one mental function; they are the manner in which the minds of both man and animal treat cognitive material at any level. There is no basic difference in this respect between what happens when a person looks at the world directly, and when he sits with his eyes closed and "thinks".

Het is mijn overtuiging dat de structuur van de hersenen een wezenlijke voorwaarde is voor deze vorm van denken en bewustzijn. De vraag of er andere vormen van intelligentie zijn kan ik niet beantwoorden. Maar alleen deze vorm van intelligentie heeft betekenis in de wereld waarin wij leven.

Determinisme versus holisme

Tot slot: De moderne wetenschap wordt vaak verweten dat zij te deterministisch is. Religies zijn onttroond door de wetenschap die alles tot op het bot ontleedt. Maar veel mensen, vooral niet-wetenschappers, menen nog steeds dat er meer is tussen hemel en aarde, simpel omdat zij de behoefte hebben aan dit meer. De behoefte aan mystiek is een menselijke — en wellicht ook dierlijke — eigenschap die niet met het verstand valt uit te doven.

De kijk op de wereld die ontstaat door het bestuderen van neurale netwerken in een complexe omgeving is anders dan de deterministische visie. Het is een holistische kijk. En holisme is meer dan een kreet uit de New Age supermarkt. Neurale netwerken, kunstmatige populaties van units met simpele eigenschap leiden tot complexe processen. De bestudering hiervan kan leiden tot een beter begrip van de complexiteit die leven definieert. Het biedt — naast religie en deterministische wetenschap — een derde weg.

Peter Kleiweg
Groningen, april 1996

Bijlage A

Vectoren en matrices

Vectoren

Een *vector* wordt aangeduid met een vette kleine letter, bijvoorbeeld: **a**.

Elementen van een vector worden aangeduid met cursieve kleine letters met een subscript dat de positie binnen de vector aangeeft, bijvoorbeeld:

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} \tag{A.1}$$

De *dimensie* van een vector is het aantal elementen in de vector. De vector **a** in vergelijking A.1 heeft dimensie 4. Deze vector kan meetkundig opgevat worden als een pijl van de oorsprong in een 4-dimensionale ruimte naar het punt dat wordt aangeduid met de coördinaten (a_1, a_2, a_3, a_4).

Figuur A.1 laat grafisch enkele bewerkingen op vectoren zien. Twee vectoren met gelijke dimensie kunnen bij elkaar worden opgeteld door hun componenten bij elkaar op te tellen. Vermenigvuldigen van een vector met een constante houdt in dat elke component van de vector wordt vermenigvuldigd met die constante.

De *lengte* van de vector uit vergelijking A.1 is:

$$||\mathbf{a}|| = \sqrt{a_1^2 + a_2^2 + a_3^2 + a_4^2} \tag{A.2}$$

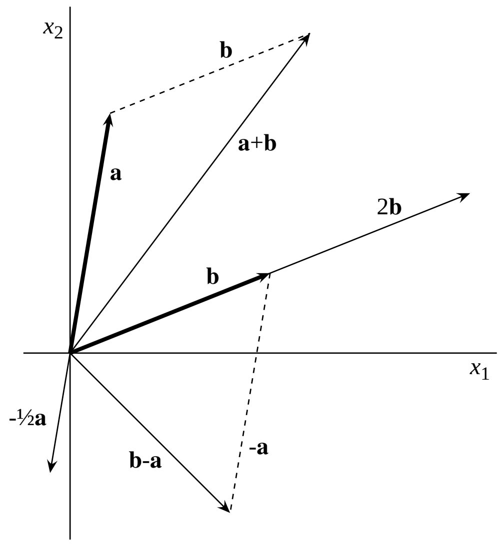
Het *inwendig produkt* van twee vectoren is alleen gedefinieerd voor twee vectoren met gelijke dimensie, bijvoorbeeld voor **a** en **b** met dimensie 4:

$$(\mathbf{a}, \mathbf{b}) = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4 \tag{A.3}$$

De *hoek* α tussen twee vectoren **a** en **b** wordt gedefinieerd door:

$$\cos \alpha = \frac{(\mathbf{a}, \mathbf{b})}{||\mathbf{a}||\,||\mathbf{b}||} \tag{A.4}$$

Een vector is *orthogonaal* t.o.v. een andere vector als het inwendig produkt van beide vectoren 0 is. De hoek is dan 90 graden, en wordt voor de vectoren



Figuur A.1: Bewerkingen op vectoren **a** en **b**

a en **b** symbolisch weergegeven door: $\mathbf{a} \perp \mathbf{b}$. Een set vectoren is orthogonaal als elke vector orthogonaal is t.o.v. elke andere vector in de set. Voor vectoren met dimensie n geldt dat een orthogonale set maximaal n vectoren kan hebben.

Een vector **a** is *lineair afhankelijk* van een groep vectoren $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k$ als geldt:

$$\mathbf{a} = \beta_1 \mathbf{b}_1 + \beta_2 \mathbf{b}_2 + \cdots + \beta_k \mathbf{b}_k \tag{A.5}$$

Hierin zijn $\beta_1, \beta_2, \ldots, \beta_k$ constanten. Een set vectoren is lineair onafhankelijk als elke vector lineair onafhankelijk is van elke subset van de overige vectoren. Voor vectoren met dimensie n geldt dat een lineair onafhankelijke set maximaal n vectoren kan hebben.

Lineaire onafhankelijkheid is een zwakkere beperking dan orthogonaliteit. Twee vectoren zijn orthogonaal als de hoek tussen de vectoren 90 graden is. Twee vectoren zijn lineair onafhankelijk als de hoek ongelijk is aan 0 graden.

Voorbeelden van vectoren in neurale netwerken zijn een invoer- of uitvoer-patroon of de huidige activaties van alle units.

Matrices

Een *matrix* wordt aangeduid met een vette hoofdletter, bijvoorbeeld: **W**. Een matrix kan opgevat worden als een reeks vectoren, bijvoorbeeld: $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3]$. De vectoren vormen de kolommen van de matrix:

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix} \tag{A.6}$$

Dit wordt een 4×3 matrix genoemd (*hoog* $\times$ *breed*).

De *getransponeerde* van een matrix is de spiegeling van een matrix t.o.v. de diagonaal, en wordt aangeduid met een superscript T:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \end{bmatrix} \tag{A.7}$$

Twee matrices kunnen met elkaar vermenigvuldigd worden als de breedte van de eerste matrix gelijk is aan de hoogte van de tweede, bijvoorbeeld:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \\ c_{41} & c_{42} \end{bmatrix} \tag{A.8}$$

De resulterende matrix is even hoog als de eerste matrix, en even breed als de tweede. Kort gezegd: vermenigvuldigen van een $L \times M$ matrix met een $M \times N$ matrix levert een $L \times N$ matrix. Elke waarde in de nieuwe matrix is het inwendig produkt van een rij uit de eerste matrix en een kolom uit de tweede matrix, bijvoorbeeld: $c_{32} = a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32}$. Schematisch weergegeven ziet het er zo uit:

$$\begin{bmatrix} . & . & . \\ . & . & . \\ a_{31} & a_{32} & a_{33} \\ . & . & . \end{bmatrix} \times \begin{bmatrix} . & b_{12} \\ . & b_{22} \\ . & b_{32} \end{bmatrix} = \begin{bmatrix} . & . \\ . & . \\ . & c_{32} \\ . & . \end{bmatrix} \tag{A.9}$$

Een vector kan met een matrix of met een andere vector worden vermenigvuldigd door vectoren op te vatten als matrices met maar één kolom.

Twee vectoren, $\mathbf{a}$ en $\mathbf{b}$, met dimensie n kunnen op twee manier met elkaar worden vermenigvuldigd:

- $\mathbf{a}^{\mathrm{T}}\mathbf{b}$ levert een 1×1 matrix op. (het inwendig produkt)
- $\mathbf{a}\mathbf{b}^{\mathrm{T}}$ levert een $n \times n$ matrix.

Voorbeelden van matrices in neurale netwerken zijn de gewichten op de verbindingen tussen twee lagen units.

Lineaire afbeeldingen

Een matrix $\mathbf{W}$ kan opgevat worden als een functie die een invoer-vector $\mathbf{x}$ omzet in een uitvoer-vector $\mathbf{y}$:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \tag{A.10}$$

Gegeven de matrix $\mathbf{W}$ in vergelijking A.10 is de *inverse* matrix $\mathbf{W}^{-1}$ gedefinieerd als:

$$\mathbf{x} = \mathbf{W}^{-1}\mathbf{y} \tag{A.11}$$

Hoe de inverse $\mathbf{W}^{-1}$ berekend kan worden uit $\mathbf{W}$ — als de inverse bestaat — wordt onder andere uitgelegd in [Koh89].

Vector/matrix-notatie wordt in literatuur over neurale netwerken vaak gebruikt om bewerkingen kompakt weer te geven:

$$\mathbf{y} = f(\mathbf{W}\mathbf{x}) \tag{A.12}$$

is gelijk aan:

$$y_i = f(\sum_j w_{ij} x_j) \tag{A.13}$$

Hierin is w_{ij} het gewicht van unit x_j naar unit y_i.

Literatuur

Meer over bewerkingen met vectoren en matrices, en de toepassing daarvan in de analyse van neurale netwerken is te vinden in [Koh89, Jor86].

Bijlage B

Overzicht van literatuur

[HKP91] Zeer goed, maar pittig studieboek. Minder geschikt voor zelfstudie. Begeleiding van docent met wiskundige kennis gewenst, maar niet echt noodzakelijk om enkele in het boek besproken modellen te kunnen implementeren. De auteurs zijn wetenschappers in de statistische mechanica. De behandeling van de theorie is exact-analytisch, met vrijwel geen aandacht voor neuro-biologische of psychologische aspecten.

[RMtPRG86, MRtPRG86] Deze 'bijbel' van *Parallel Distributed Processing* is voor een groot deel verantwoordelijk voor de hernieuwde belangstelling voor neurale netwerken, na de impasse in de jaren 70. Deze boeken vormen een zeer goede inleiding in het vak, vanuit een cognitieve/psychologische invalshoek. Door deze invalshoek is het werk goed te volgen voor studenten die niet uit de β-wetenschappen komen. Er wordt weinig wiskundige kennis vereist, maar hoofdstuk 9 bevat een introductie in de lineaire algebra, die zeer van pas komt in de studie naar neurale netwerken. De ontwikkelingen sinds de verschijning van deze boeken zijn zeer hard gegaan, zodat aanvullende literatuur nodig is voor het verkrijgen van een goed overzicht van de veelgebruikte hedendaagse technieken. Te vinden in bibliotheek sociale wetenschappen.

[MR88] Het practicum-handboek met software behorend bij [RMtPRG86, MRtPRG86], dat ook los te gebruiken is. Niveau is goed te volgen. Vooral backpropagation wordt helder en uitgebreid behandeld. De keuze van de onderwerpen is nogal beperkt. Te vinden in de UB.

[Fu94] Redelijk boek voor gebruik bij een college neurale netwerken voor α-informatici. Volledig, actueel, inleidend en verdiepend, maar soms erg beknopt. In verdieping ligt de nadruk op toepassingen in kunstmatige intelligentie. De *top down* benadering die zo kenmerkend is voor het vakgebied kunstmatige intelligentie (en zijn grootste beperking) doet het essentiële *bottom up* kenmerk van neurale netwerken geweld aan. Opzet en niveau is vergelijkbaar met [RK91] (zelfde uitgever).

[Was89] Zeer geschikt voor gebruik bij een college neurale netwerken. Weinig gebruik van wiskunde, maar genoeg details om zelf simulaties te kunnen ontwikkelen. Te vinden in bibliotheek letteren.

[CB90] Goed overzicht van het vakgebied vanuit een *Artificial Intelligence* oogpunt. Bevat een uitgebreide verklarende woordenlijst. Niveau is goed te volgen voor studenten α-informatica maar het boek bevat te weinig details om zelf simulaties te kunnen schrijven. Gebrek aan details leidt ook vaak tot beperkt inzicht in de werking van bepaalde modellen. Een literatuurlijst ontbreekt. Te vinden in bibliotheek letteren.

[Has95] Dik boek met grondig overzicht van alle belangrijke aspecten van neurale netwerken. Veel wiskunde. Zeer goed als naslagwerk te gebruiken, minder geschikt als studieboek.

[Kha90] Dun boekje met wiskundige beschrijving van alle belangrijke modellen in de neuroinformatica. Waarschijnlijk te moeilijk voor studenten α-informatica. Bevat een verklarende woordenlijst. Te vinden in bibliotheek letteren.

[FS91] Boek over het zelf programmeren van simulaties van neurale netwerken. Vrij volledig boek, te gebruiken als studieboek. Uitgangspunt is de werking en implementatie van de meest belangrijke soorten netwerken, er is zeer weinig aandacht voor de achterliggende motivaties. Bevat pseudo-codes die implementatie mogelijk moeten maken. Naar mijn mening zijn de algoritmen hiervoor vaak onnodig complex. Een ander nadeel is dat de code in een soort *pascal* is geschreven wat nog eens extra bijdraagt aan de omslachtigheid. *Simple recurrent networks* (Jordan, Elman) en genetische algoritmen ontbreken. Te vinden in bibliotheek letteren.

[Koh89] Pittig boek, maar wel geschikt als leerboek. Gaat voornamelijk over geheugens en zelforganisatie (o.a. Kohonen-kaarten), het onderwerp dat in andere inleidende boeken vaak onderbelicht wordt. Bevat vrij pittige inleiding in de wiskunde die je in veel artikelen over neurale netwerken tegenkomt. Te vinden bij (?) bibliotheek wiskunde of natuurkunde.

[Gro88] Dit is een omvangrijk boek, niet geschikt voor beginners. Maar wie verder wil dan 'speelgoed'-problemen oplossen met backpropagation moet dit boek zeker eens inkijken. Het geeft een goede indruk van wat er verwacht wordt van mensen die zich op een hoog niveau met neurale netwerken willen bezighouden. Neurofysiologie, psychologie, wiskunde en technische kennis spelen een grote rol. De auteurs houden zich bezig met primaire functies voor intelligente systemen. Waarnemen, bewegen, coördinatie komen uitgebreid aan bod. *Adaptive Resonance Theory* (ART1) wordt behandeld.

[Mii93] Een boek over FGREP en de toepassing hiervan in een groter systeem: DISCERN. Boeiend en zeer goed te lezen.

[Arb95] Zeer lijvig boekwerk met verzameling artikel over alle mogelijke deelgebieden van neuro-informatica. Te vinden in de bibliotheek letteren (niet uitleenbaar).

[Tou89, Tou90, LMT91, MHL92, HCG93, CTA94, TTL95] Verzamelingen artikelen van de NIPS-congressen. Te vinden in de bibliotheek letteren.

Bibliografie

[Alt90] Gerry T. M. Altmann, editor. *Cognitive Models of Speech Processing: Psycholinguistic and Computational Perspectives.* The MIT Press, Cambridge, MA, 1990.

[Arb95] M. A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks.* The MIT Press, Cambridge, MA, 1995.

[Arn69] R. Arnheim. *Visual Thinking.* University of California Press, Berkely, CA, 1969.

[CA86] F. Crick and C. Asanuma. *Certain Aspects of the Anatomy and Physiology of the Cerebral Cortex*, chapter 20, pages 333–371. Volume 2: Psychological and Biological Models of PDP-2 [MRtPRG86], 1986.

[Cal] William H. Calvin. *De cerebrale symfonie.* Oorspronkelijke titel onbekend.

[Cal86] William H. Calvin. *The River that Flows Uphill.* 1986. Vertaald als: De rivier die tegen de berg opstroomt.

[Cal90] William H. Calvin. *The Ascent of Mind: Ice Age Climates and the Evolution of Intelligence.* 1990. Vertaald als: De opkomst van het intellect.

[CB90] Maureen Caudill and Charles Butler. *Naturally Intelligent Systems.* The MIT Press, Cambridge, MA, 1990.

[Cha95] Agnès Charrel. Tierra network version. Technical report, Institut National des Télécommunications, Advanced Telecommunications Research Institute, 1995.

[Cli91] Dave Cliff. Computational neuroethology: A provisional manifesto. In Meyer and Wilson [MW91], pages 29–39.

[CTA94] Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors. *Advances in Neural Information Processing Systems 6*, San Mateo, CA, 1994. Morgan Kaufmann Publishers.

[Elm90] Jeffrey L. Elman. Representation and structure in connectionist models. *Cognitive Science*, 14, 1990.

[Fah91] Scott E. Fahlman. The recurrent cascade-correlation architecture. In Lippmann et al. [LMT91], pages 190–196.

[FF87] Martin A. Fischler and Oscar Firschein. *Intelligence: The Eye, the Brain, and the Computer.* Addison-Wesley, Reading, MA, 1987.

[FL90] Scott E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In Touretzky [Tou90], pages 524–532.

[FS91] James A. Freeman and David M. Skapura. *Neural Networks: Algorithms, Applications, and Programming Techniques.* Addison-Wesley, Reading, MA, 1991.

[Fu94] LiMin Fu. *Neural Networks in Computer Intelligence.* McGraw-Hill, New York, 1994.

[Gol89] D. Goldberg. *Genetic Algorithms.* Addison-Wesley, Reading, MA, 1989.

[Gro88] Stephen Grossberg, editor. *Neural Networks and Natural Intelligence.* The MIT Press, Cambridge, MA, 1988.

[Has93] Mohamad H. Hassoun, editor. *Associative Neural Memories: Theory and Implementation.* Oxford University Press, New York, 1993.

[Has95] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks.* The MIT Press, Cambridge, MA, 1995.

[HCG93] Stephen Jose Hanson, Jack D. Cowan, and C. Lee Giles, editors. *Advances in Neural Information Processing Systems 5*, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[Heb49] Donald O. Hebb. *The Organization of Behavior.* Wiley, New York, 1949.

[HH91] P. Hogeweg and B. Hesper. Evolution as pattern processing: TODO as substrate for evolution. In Meyer and Wilson [MW91], pages 492–497.

[HKP91] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, Reading, MA, 1991.

[HL88] W. Y. Huang and R. P. Lippmann. Neural net and traditional classifiers. In D. Z. Anderson, editor, *Neural Information Processing Systems*, pages 387–396. American Institute of Physics, New York, 1988.

[HN87] R. Hecht-Nielsen. Counterpropagation networks. *Applied Optics*, 26:4979–4984, 1987.

[HN88] R. Hecht-Nielsen. Applications of counterpropagation networks. *Neural Networks*, 1:131–139, 1988.

[Hol92] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The MIT Press, Cambridge, MA, 2nd edition, 1992.

[HS86] G. E. Hinton and T. J. Sejnowski. *Learning and Relearning in Boltzmann Machines*, chapter 7, pages 282–317. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[Jor86] M. I. Jordan. *An Introduction to Linear Algebra in Parallel Distributed Processing*, chapter 9, pages 365–422. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[Kal92] James W. Kalat. *Biological Psychology*. Brooks/Cole Publishing Company, Pacific Grove, California, 4th edition, 1992.

[Kan88] P. Kanerva. *Sparse Distributed Memory*. The MIT Press, Cambridge, MA, 1988.

[Kha90] Tarun Khanna. *Foundations of Neural Networks*. Addison-Wesley, Reading, MA, 1990.

[Koh72] Teuvo Kohonen. Correlation matrix memories. *IEEE Transactions on Computers*, C(21):353–359, 1972.

[Koh88] Teuvo Kohonen. The "neural" phonetic typewriter. *Computer*, 21(3):11–22, March 1988.

[Koh89] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer-Verlag, Berlin, 3rd edition, 1989.

[Law94] Diane Law. Searle, subsymbolic functionalism and synthetic intelligence. Technical Report AI93-222, Department of Computer Sciences, The University of Texas at Austin, 1994.

[Lin86] R Linsker. From basic network principles to neural architecture. In *Proceedings of the National Academy of Sciences, USA*, volume 83, pages 7508–7512, 8390–8394, 8779–8783, 1986.

[LM94] Diane Law and Risto Miikkulainen. Grounding robotic control with genetic neural networks. Technical Report AI93-223, Department of Computer Sciences, The University of Texas at Austin, 1994.

[LMT91] Richard P. Lippmann, John E. Moody, and David S. Touretzky, editors. *Advances in Neural Information Processing Systems 3*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[LTFR91] C. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors. *Artificial Life II*, volume X of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley, Redwood City, CA, 1991.

[McC81] J. L. McClelland. Retrieving general and specific information from stored knowledge of specifics. In *Proceedings of the Third Annual Meeting of the Cognitive Science Society*, pages 170–172, 1981.

[MD91] Risto Miikkulainen and Michael G. Dyer. Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15:343–399, 1991.

[MH69] J. McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, Edinburgh, 1969.

[MHL92] John E. Moody, Steve J. Hanson, and Richard P. Lippmann, editors. *Advances in Neural Information Processing Systems 4*, San Mateo, CA, 1992. Morgan Kaufmann Publishers.

[Mii93] Risto Miikkulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory.* The MIT Press, Cambridge, MA, 1993.

[MK95] Sean D. Murphy and Edward W. Kairiss. Model of a biological neuron as a temporal neural network. In Tesauro et al. [TTL95], pages 85–91.

[MM95] Mark Moll and Risto Miikkulainen. Convergence-zone episodic memory: Analysis and simulations. Technical Report AI95-227, Department of Computer Sciences, The University of Texas at Austin, 1995.

[MMss] David Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22, in press.

[MP88] Marvin Minsky and Seymour Papert. *Perceptrons: Expanded Edition.* The MIT Press, Cambridge, MA, 1988.

[MR88] James L. McClelland and David E. Rumelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* The MIT Press, Cambridge, MA, 1988.

[MRH86] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton. *The Appeal of Parallel Distributed Processing*, chapter 1, pages 3–44. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[MRtPRG86] James L. McClelland, David E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2: Psychological and Biological Models. The MIT Press, Cambridge, MA, 1986.

[MW91] Jean-Arcady Meyer and Stewart W. Wilson, editors. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, Cambridge, MA, 1991. The MIT Press.

[PA87] C. Peterson and J. R. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[Par85] D. B. Parker. Learning logic. Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[PB91] Martin Pickering and Guy Barry. Sentence processing without empty categories. *Language and Cognitive Processes*, 6(3):229–259, 1991.

[PL90] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.

[Ray91] Thomas S. Ray. An approach to the synthesis of life. In Langton et al. [LTFR91], pages 371–408.

[Ray92] Thomas S. Ray. How I created life in a virtual universe, 1992.

[Ray94a] Thomas S. Ray. An evolutionary approach to synthetic biology: Zen and the art of creating life. *Artificial Life*, 1(1/2):195–226, 1994.

[Ray94b] Thomas S. Ray. A proposal to create a network-wide biodiversity reserve for digital organisms, 1994.

[Res91] Mitchel Resnick. Animal simulations with ⋆logo: Massive parallelism for the masses. In Meyer and Wilson [MW91], pages 534–539.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, chapter 8, pages 318–362. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[RK91] Elaine Rich and Kevin Knight. *Artificial Intelligence*. McGraw-Hill, Inc., New York, 2nd edition, 1991.

[RM86] D. E. Rumelhart and J. L. McClelland. *PDP Models and General Issues in Cognitive Science*, chapter 4, pages 110–146. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[RMS89] Helge J. Ritter, Thomas M. Martinetz, and Klaus J. Schulten. Topology-conserving maps for learning visuo-motor-coordination. *Neural Networks*, 2(3):159–168, 1989.

[RMtPRG86] David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. The MIT Press, Cambridge, MA, 1986.

[RS86] H. Ritter and K. Schulten. Topology conserving mappings for learning motor tasks. In John S. Denker, editor, *Neural Networks for Computing*, pages 376–380. American Institute of Physics, New York, 1986.

[RS87] H. Ritter and K. Schulten. Extending Kohonen's self-organizing mapping algorithm to learn ballistic movements. In Rolf Eckmiller and Christoph v.d. Malsburg, editors, *Neural Computers*, pages 393–406. Springer-Verlag, Heidelberg, 1987.

[RZ86] D. E. Rumelhart and D. Zipser. *Feature Discovery by Competitive Learning*, chapter 5, pages 151–193. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[Sch91] Uwe Schnepf. Robot ethology: A proposal for the research into intelligent autonomous systems. In Meyer and Wilson [MW91], pages 465–474.

[Sea80] J. R. Searle. Minds, brains and programs. *The Behavioral and Brain Sciences*, 3:417–458, 1980.

[Shi86] Stuart M. Shieber. *An Introduction to Unification-Based Approaches to Grammar*. Centre for the Study of Language and Information, 1986.

[SM94] Joseph Sirosh and Risto Miikkulainen. Cooperative self-organization of afferent and lateral connections in cortical maps. *Biological Cybernetics*, 71:66–78, 1994.

[Smo86] P. Smolensky. *Information Processing in Dynamical Systems: Foundations of Harmony Theory*, chapter 6, pages 194–281. Volume 1: Foundations of PDP-1 [RMtPRG86], 1986.

[SR87] Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168, 1987.

[Tou89] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 1*, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Tou90] David S. Touretzky, editor. *Advances in Neural Information Processing Systems 2*, San Mateo, CA, 1990. Morgan Kaufmann Publishers.

[TR] Kurt Thearling and Thomas S. Ray. Evolving multi-cellular artificial life. In *Artificial Life IV conference proceedings*.

[TTL95] Gerald Tesauro, David Touretzky, and Todd Leen, editors. *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1995. The MIT Press.

[VBK+86] G. Verkerk, J. B. Broens, W. Kranendonk, F. J. v.d. Puijl, J. L. Sikkema, and C. W Stam. *Binas: Informatieboek VWO/HAVO voor het onderwijs in de natuurwetenschappen*. Wolters-Noordhof, Groningen, 2nd edition, 1986.

[Was89] Philip D. Wasserman. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, New York, 1989.

[Wer74] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.