FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Inleiding tot Kunstmatige Intelligentie

Jan Cnops
25 februari 2019

UNIVERSITEIT
GENT

Academiejaar 2018-2019

INHOUDSOPGAVE

HOOFDSTUK 1
KUNSTMATIGE INTELLIGENTIE

1.1 INLEIDING

Getting a machine learning system to be 99% correct is relatively easy,
but getting it to be 99.9999% correct, which is where it ultimately needs to be,
is vastly more difficult.
(tesla.com.)

Deze cursus is de inleiding tot het zeer grote vakgebied van kunstmatige intelligentie (AI:**A**rtificial **I**ntelligence). Het is dan ook niet mogelijk gebleken om een overzicht te geven van het hele vakgebied zonder te vervallen in een vage oppervlakkigheid. In de plaats daarvan is ervoor gekozen om enkele specifieke onderwerpen vrij uitgebreid aan bod te laten komen omdat ze zeer typisch zijn. Daarnaast komen een aantal onderwerpen korter aan bod, sommige enkel in het kader van één enkele specifieke toepassing. Dit geeft, zonder al te veel plaats in te nemen, een meer concrete kennismaking dan een theoretisch overzicht. Voorbeelden van deze benadering zijn de bespreking van genetische algoritmen, toegepast op actieve regelbanken, en Bayesiaanse leermethodes toegepast op spraakherkenning.

De twee voornaamste onderwerpen in deze cursus zijn expertsystemen en neurale netwerken. Er zijn andere technieken dan deze in AI, maar alle hebben ze veel gemeen met een van de twee, of met beide, te bespreken onderwerpen. Men kan in AI namelijk een onderscheid maken tussen twee vormen om kennis in te brengen in een computersysteem: ofwel brengt men de kennis expliciet in (in de vorm van code of van data), ofwel maakt men programmatuur die de kennis op een of andere manier zelf kan verwerven, m.a.w. programmatuur die zelf kan leren. Elk systeem in AI maakt gebruik van een van deze twee vormen, eventueel van een mengeling. Expertsystemen zijn het typevoorbeeld van expliciete inbreng van kennis, neurale netwerken van lerende systemen, en daarom hebben we deze gekozen als onderwerpen. Maar eerst geven we een kort overzicht van wat kunstmatige intelligentie juist is.

AI poogt om machines –en in het bijzonder computers– intelligent gedrag te doen vertonen. Het doel hiervan is tweeledig:

- Het laten overnemen, door machines, van taken waarvoor intelligentie vereist is.

- De studie van natuurlijke intelligentie.

Wat het laatste betreft gaat men theorieën over het denken van mensen en dieren uittesten door dit denken te simuleren met machines. Er is een samenhang met de eerste doelstelling: als we denken kunnen simuleren op een machine, dan kunnen we dit gebruiken om bepaalde taken uit te voeren. Maar er is een verschil: als we een methode vinden om een machine intelligent gedrag te laten vertonen die niet overeenkomt met menselijk (of dierlijk) denken, dan draagt dit bij tot de eerste doelstelling, maar niet tot de tweede. We gaan verder alleen in op het eerste doel: machines intelligente taken laten uitvoeren.

1.2 KUNNEN MACHINES DENKEN?

Dit brengt ons bij de vraag of machines intelligent kunnen zijn, m.a.w. of ze kunnen *denken*. We willen niet diep op deze vraag ingaan, maar toch enige elementen daarover aanhalen. Het grootste probleem bij de hele discussie is dat er geen eenduidige definitie is van wat denken eigenlijk is. Tot halverwege de twintigste eeuw was denken iets dat men alleen aantrof bij mensen en—in zeer beperkte mate—bij dieren die voldoende mensachtig waren. Klassieke definities gaan dan ook uit van bepaalde menselijke capaciteiten. Zo geeft Van Dale de definitie: "het verstand gebruiken, [...] een reeks voorstellingen van de geest bewust op elkaar doen volgen [...]". Met zo'n definitie wordt het de vraag of machines verstand, een geest en bewustzijn hebben (antwoord: ja, tenminste als het gaat om machines die kunnen denken). Als het gaat om de vraag of machines kunnen denken geven de meeste mensen eerst een antwoord op deze vraag, en leiden dan een definitie van denken af die hun antwoord bevestigt. We geven hier een overzicht van enkele mogelijke definities van denken zonder diep in te gaan op de consequenties van die definities, die trouwens meestal duidelijk zijn, of op de argumenten die gegeven worden. Bijna elke definitie van denken maakt impliciet of expliciet gebruik van de idee van complexe informatieverwerking, meestal in de vorm van het manipuleren en ordenen van ideeën. Als we dit als enige criterium nemen dan is het duidelijk dat een computer kan denken: een bezigheid zoals schaakspelen veronderstelt duidelijk de manipulatie van vrij complexe patronen. Maar meestal gaat men ervan uit dat denken meer is dan gewone manipulatie van informatiepatronen. Definities die meer poneren dan alleen deze manipulatie zijn echter zeer problematisch: dikwijls worden interne eigenschappen verondersteld waarvan we niet weten wat er mee bedoeld wordt.

Nemen we een voorbeeld: het roemruchte programma ELIZA (Weizenbaum, 1966). ELIZA was een programma waarmee je kon chatten, en dat zich voordeed als een psychotherapeut. In Figuur 1.1 vind je een voorbeeld van tot wat ELIZA in staat was. Het bleek dat ELIZA veel mensen overtuigde, alhoewel het programma bedoeld was als een parodie op een Rogeriaans psychotherapeut. Veel mensen die een praatje hadden geslagen met ELIZA waren dan ook diep teleurgesteld als ze te horen kregen hoe ELIZA werkt: met een aantal simpele vervangingsregels (Figuur 1.2). Sinds ELIZA probeert men altijd definities op te stellen van denken en intelligentie op zo'n manier dat ELIZA er niet aan voldoet.

Eén van de denkrichtingen hier stelt dat schijnbaar intelligent gedrag onvoldoende is om van denken te kunnen spreken. Voor sommigen is het nodig dat een entiteit *begrijpt* wat ze doet voor er van intelligentie sprake is. Het verschil tussen begrijpen en niet-begrijpen wordt meestal geïllustreerd aan de hand van de zogenaamde *Chinese Kamer* (Searle, 1980), die een fictieve situatie aangeeft waar schijnbaar intelligent gedrag en niet-begrijpen samen gaan. Iemand die geen Chinees kent wordt in een kamer gebracht. Door een luik krijgt hij briefjes in het Chinees aangereikt, en de bedoeling is dat hij daar schriftelijk een zinnige repliek op teruggeeft. In de kamer is er een aantal handboeken in een taal die de proefpersoon begrijpt. Als hij de regels in die handboeken volgt, dan zullen de replieken die hij

Person:	Men are all alike.
ELIZA:	In what way?
Person:	They're always bugging us about something or other.
ELIZA:	Can you think of a specific example?
Person:	Well, my boyfriend made me come here.
ELIZA:	Your boyfriend made you come here?
Person:	He says I'm depressed much of the time.
ELIZA:	I am sorry to hear that you are depressed.
Person:	It's true. I am unhappy.
ELIZA:	Do you think coming here will help you not to be unhappy?
Person:	I need some help, that much seems certain.
ELIZA:	What would it mean to you if you got some help?
Person:	Perhaps I could learn to get along with my mother.
ELIZA:	Tell me more about your family.

Figuur 1.1. Chatten met ELIZA

```
(X me Y) ---> (X you Y)
(I remember X) ---> (Why do you remember X just now?)
(My {family-member} is Y) ---> (Who else in your family is Y?)
(X {family-member} Y) ---> (Tell me more about your family)
(No) ---> (You are being a bit negative)
```

Figuur 1.2. Achter de schermen van ELIZA

optekent voor iemand die Chinees kent een teken zijn van intelligent gedrag. De proefpersoon echter heeft mechanisch de regels in de handboeken gevolgd, en heeft geen idee wat de boodschappen eigenlijk betekenen. Alhoewel zijn gedrag intelligent lijkt, is er hier geen sprake van denken omdat alle begrip ontbreekt. Begrijpen als criterium is echter problematisch, omdat het bijna onmogelijk is om vast te stellen wat het betekent dat een entiteit iets begrijpt. We kunnen het niet vragen: als we de Chinese Kamer een briefje geven met de vraag (in het Chinees) "begrijpt u wat op dit briefje staat?", zal het antwoord ongetwijfeld "ja" zijn (het antwoord "nee" zou bizar zijn in de context).

Een tweede denkrichting is gebaseerd op de weerstand tegen de idee dat machines kunnen denken, of dat er ooit machines zullen zijn die dat kunnen. Een definitie van denken die deze weerstand ondersteunt is dat denken elke vorm van complexe informatieverwerking is die niet mechanisch gedetermineerd is. Duidelijk kan een computer volgens deze definitie niet denken, vermits de werking door de programmatuur gedetermineerd is[1]. Dergelijke

[1] Hardwarefouten kunnen dit proces onvoorspelbaar maken. Dit impliceert dat kapotte computers misschien wel kunnen denken.

definitie doet echter een nieuwe vraag rijzen, die gesteld werd door de bekende behavioristische psycholoog Burrhus Skinner: "het is niet de vraag of machines kunnen denken, het is de vraag of mensen dat kunnen".

Vaak blijkt dat mensen bepaald gedrag als bewijs van denken willen aanvaarden, mits ze ervan overtuigd zijn dat een computer het nooit zal vertonen. Rond 1960 zijn er een aantal voorbeelden vernoemd van gedrag dat "ongetwijfeld" intelligent zou genoemd worden. Ondertussen hebben computerprogramma's bijna alle items op de lijst uitgevoerd. Het gaat om dingen zoals schaak spelen op grootmeesterniveau, gedichten schrijven, het bewijzen van wiskundige stellingen, menselijke taal begrijpen (op beperkte schaal weliswaar), enzovoorts. Het blijkt dat, telkens er een nieuwe vorm van intelligent gedrag wordt vertoond door een machine, gewoon de norm voor wat intelligent is verlegd wordt, zodat nu nog steeds kan gezegd worden dat machines niet intelligent zijn en niet kunnen denken. Blijkbaar is er een onderliggende definitie die nooit wordt uitgesproken, maar die als volgt luidt:

> Denken is elke vorm van complexe informatieverwerking waarvan de onderliggende mechanismen niet volledig gekend zijn.

Met deze definitie is het duidelijk dat machines niet kunnen denken, en dat dit nog een hele tijd zo zal blijven. We kunnen ons wel inbeelden dat er op den duur machines zijn die andere machines maken waarvan de juiste werking niet gekend is door de mens, en die volgens deze definitie dus wel kunnen denken. Overigens is deze definitie uitstekend om het element van begrijpen te omvatten, daar niemand weet hoe begrijpen in zijn werk gaat. Nadeel van deze definitie is dat ze verandert met de tijd, en er uiteindelijk toe kan leiden dat ze niet meer op de mens toepasselijk is. We kunnen dit toepassen op de intelligentie van ELIZA: mensen die niet weten hoe ELIZA werkt vinden ELIZA intelligent; nadat ze de werking leren kennen is dat niet meer het geval.

In dit verband kunnen we niet voorbijgaan aan de zogenaamde *Turingtest* uit 1950, die in zekere zin een toetssteen is voor het onderzoek. Er is overigens nog nooit een serieuze poging gedaan om een machine een Turingtest te laten doorstaan: daar zijn we (nog) niet aan toe. De bedoeling van Turing was om een criterium te geven dat zo moeilijk te bereiken was dat iedereen het erover eens zou zijn dat er intelligentie nodig was om eraan te voldoen.

Bij de Turingtest kan een proefpersoon contact maken met twee entiteiten. Aan de ene kant een mens, aan de andere kant de machine die getest wordt. Het contact verloopt niet rechtstreeks: er kunnen alleen geschreven boodschappen worden uitgewisseld. De proefpersoon kan vragen en zeggen wat hij wil. Als hij, na zijn ondervraging, niet kan uitmaken wat de machine was, en wat de mens, dan is de machine geslaagd voor de Turingtest. De Turingtest is wel heel veeleisend: een machine moet, om eraan te voldoen, intelligent gedrag vertonen dat evenwaardig is aan dat van een mens. Meestal zullen we met minder tevreden zijn. Belangrijk is echter dat de idee van de Turingtest aantoont dat het in principe denkbaar is dat een machine intelligent gedrag vertoont[2].

1.3 TOEPASSINGEN VAN AI EN DATA MINING

Een belangrijk kenmerk van de meeste intelligente taken die we aan een machine zouden kunnen overlaten is dat we wel weten *wat* er moet gebeuren, maar dat we niet goed weten *hoe* dit moet gebeuren. Een paar voorbeelden van taken die reeds, met meer of minder succes, door machines worden uitgevoerd:

[2] In bepaalde omstandigheden kan een machine uiteraard wel mensachtig gedrag vertonen. Een chatbot zoals *Cleverbot* slaagt er uitstekend in om doelloos geleuter te produceren dat nauwelijks te onderscheiden is van menselijk doelloos geleuter. Er wordt geclaimd dat Cleverbot geslaagd is voor een versie van de Turingtest alhoewel hij niet is in staat is een gesprek over welk onderwerp dan ook te voeren.

- Het stellen van medische diagnoses.
- Kwaliteitscontrole. Een voorbeeld hiervan is de controle van geweven textiel. Hierbij neemt een camera een beeld op van het weefsel, en een computer bepaalt dan hoeveel onregelmatigheden, zoals knopen, er zijn, en ook hoe regelmatig het weefsel is.
- Spelen allerhande. Het bekendste is het schaakspel, waarbij computers het kunnen opnemen tegen de beste menselijke spelers.
- Allerhande toepassingen in de robotica. Spectaculair zijn natuurlijk demonstratiemodellen die kunstjes kunnen uithalen zoals piano spelen (gemakkelijk te maken) of wandelen (moeilijk).
- Aanbevelingssystemen.

In het begin richtte AI zich voornamelijk op de analyse van redeneringen. Men spreekt in dit verband bijvoorbeeld van *regelbanken*: dit zijn systemen waar het gedrag van de machine bepaald wordt door een verzameling expliciet ingevoerde regels (als dit, doe dat). Een voorbeeld van toepassing is dat van *expertsystemen*: dit zijn systemen die een beoordeling door een expert vervangen. Medischediagnoseprogramma's zijn een voorbeeld van zo'n expertsysteem. Er zijn heel wat successen geboekt met deze manier van werken, maar door het werken met expliciete regels is men verplicht deze regels eerst te weten te komen voor men ze kan invoeren. Regelbanken zijn een voorbeeld van het werken met *zoekruimten*. Een zoekruimte is een geheel van (abstracte) plaatsen, waarbij acties horen die toelaten om van de ene plaats naar de andere te gaan. Bedoeling is om van een beginplaats (of vanuit een verzameling beginplaatsen) naar een doel te gaan. Pogingen om systemen die op deze basis werken te laten leren zijn nooit zeer succesvol gebleken.

Een belangrijke component bij bijna alle AI-toepassingen is dat van *classificatie*. Hierbij hebben we een gegeven stel klassen. Als we nu een bepaalde invoer krijgen, moeten we deze invoer verbinden met een bepaald klasse. Het meest klassieke voorbeeld is dat van OCR (**O**ptical **C**haracter **R**ecognition). Een computer krijgt een scan van een afgedrukte tekst aangeboden. Anderzijds heeft hij een beperkt alfabet (dat ook leestekens e.d. kan bevatten) waaruit hij kan kiezen. Hij moet nu de symbolen die hij 'ziet' herkennen als een van de letters van het alfabet. We vereenvoudigen een beetje: eerst moet hij bepalen waar het ene symbool eindigt en het volgende begint. Ook de taken in ons bovenstaande lijstje zijn taken van classificatie:

- Het stellen van medische diagnoses is het indelen van patiënten in klassen naargelang hun kwaal.
- Kwaliteitscontrole van geweven textiel. Hier moeten de onregelmatigheden op het beeld van het weefsel herkend worden.
- Schaak. Hierbij moet men goede en slechte mogelijke zetten van elkaar onderscheiden.
- Robotica. De machine heeft een arsenaal aan mogelijke bewegingen. Deze komen overeen met de patronen. De machine evalueert haar situatie, en voert de bijbehorende beweging uit.

Bij classificatie kan het gaan om *harde classificatie*, waarbij er een beperkt aantal duidelijk van elkaar gescheiden klassen is en om *zachte classificatie*, waarbij er een continue overgang is. Bij het schaakvoorbeeld is er een continue overgang van zeer goede zetten over minder goede zetten, ... tot aan slechte zetten. Vooral bij harde classificatie spreekt men ook van *patroonherkenning*.

Meestal gaat het over taken die al uitgevoerd worden door mensen, maar dit is niet altijd het geval. Een voorbeeld uit de praktijk is het herkennen van patronen in DNA-structuren. DNA bestaat uit een reeks van basen (A, C, G en T). Men kan relatief eenvoudig het genoom van een organisme bepalen: dit is de opeenvolging van de basen binnen het DNA. Verschillende delen van het DNA hebben verschillende functies. Er zijn de verschillende genen, maar een gen is niet noodzakelijk een aaneengesloten stuk van het genoom, en ook binnen een gen zijn verschillende deelfuncties te herkennen. Het bepalen van de functies is een zeer duur procedé. Als men echter een goede gok kan doen wat de functie is, kan men een aantal tests overslaan, en ook alleen de meest interessante stukken onderzoeken. Van een aantal stukken DNA kent men al de functie. Dit kunnen stukken zijn van het genoom van het organisme zelf, maar ook van andere organismen. Men poogt nu de onbekende stukken van het genoom te vergelijken met deze gekende stukken. Mensen zijn daar helemaal niet goed in, computers wel. Er wordt zowel gebruik gemaakt van expliciete algoritmes als van massief lerende programma's om deze taak uit te voeren.

Oorspronkelijk was AI voornamelijk *probleemgestuurd.* Uitgaande van een probleem probeerde men een oplossing te zoeken. Door de toegenomen informatisering van de laatste decennia heeft men meer en meer de beschikking over collecties van gegevens, die zeer groot kunnen zijn. Dit heeft geleid tot een een *datagestuurde* benadering. Uitgaande van een datacollectie stelt men de vraag welke problemen men ermee kan oplossen. Deze datagestuurde benadering noemt men *data mining*. De basis is een collectie gegevens die verkregen wordt los van enige AI-toepassing. Een typisch voorbeeld in een commerciële omgeving is een een databank met gegevens over alle transacties met klanten die is opgebouwd door een gewone administratieve toepassing. Uitgaande van zo'n gegevensverzameling gaat men zich afvragen welke nuttige informatie hieruit kan gehaald worden. Dit leidt dan tot typische toepassingen als aanbevelingssystemen, waarbij men van elke klant een profiel aanmeet en aanbevelingen doet op basis van aankopen van klanten met een gelijkaardig profiel, of risicodetectie waarbij men probeert klanten met een risicoprofiel (bijvoorbeeld met een risico van wanbetaling) te identificeren. Men spreekt daarbij over het omzetten van data (een verzameling gegevens over losse feiten, zoals de voornoemde transacties) naar informatie (nuttige data samengesteld uit losse gegevens door deze te structureren). De methodes om gegevens te structureren zijn dezelfde als deze gebruikt bij probleemgerichte AI. Het voornaamste verschil tussen data mining en klassieke AI is dan ook dat de gegevensverzameling vaak eerst moet gereorganiseerd worden voor men de eigenlijke informatie kan extraheren.

1.4 LEREN

Daarmee komen we bij lerende systemen. Hoe leert een computerprogramma? Heel algemeen gezegd heeft zo een programma parameters die het kan aanpassen. In sommige gevallen gaat het over een klein aantal parameters, die door de ontwerper zo gekozen zijn dat ze een duidelijke betekenis hebben. Een voorbeeld is dat van een router in een computernetwerk, die zijn routertabellen zelf aanpast: de betekenis van een element van zo'n routertabel is ook voor de programmeur duidelijk. In dit hoofdstuk zullen we een vorm van dit soort leren zien wanneer we beslissingsbomen behandelen.

Een meer moderne richting in AI houdt zich bezig met systemen met een zeer groot aantal aanpasbare parameters. We spreken hierbij van *massief lerende* systemen. Hier probeert men niet meer een bepaalde betekenis aan een parameter te hechten. Door het groot aantal parameters is het niet meer mogelijk ze met de hand in te voeren en een betekenisvol programma te krijgen. In de plaats daarvan laat men het systeem leren, dikwijls aan de hand van voorbeelden. Er zijn veel parameters, en men hoopt dat er combinaties zijn

van waarden die een efficiënt programma opleveren, en men hoopt dat het programma zo'n combinatie vindt.

Er zijn verschillende types van zulke systemen. Het type dat het meest lijkt op biologische denksystemen is dat van neurale netwerken (merk op dat hier de tweede doelstelling van AI, dat van de studie van natuurlijke intelligentie, hier weer bovenkomt). Vermelden we dat er nog andere massief lerende systemen zijn, zoals HMM's (**H**idden **M**arkov **M**odel), die gebruikt worden bij de analyse van allerhande sequenties. HMM's vinden we dan ook bij spraakanalyse (een opeenvolging van klanken) en bij de studie van DNA (sequenties van symbolen). HMM's liggen eigenlijk op in het randgebied tussen massief leren met niet-specifieke parameters en gericht leren met betekenisvolle parameters: er zijn vaak zeer veel parameters in een HMM, maar ze hebben nog altijd een zekere betekenis.

Naast dit onderscheid gebaseerd op de werking van de lerende entiteit kunnen we ook een indeling maken gebaseerd op de manier waarop de leerstof wordt aangeboden. Als voorbeeld nemen we drie gevallen waarin een kind een bepaalde vaardigheid aanleert.

- Leren optellen van grote getallen. De leraar legt uit (en doet voor) hoe men twee getallen onder elkaar schrijft, en dan rechts begint met twee onder elkaar staande cijfers op te tellen (daarvoor moet het kind eerst de tabellen om cijfers op te tellen van buiten leren). Als het resultaat groter is dan negen, moet men de 1 overdragen, en zo verder. Dit is een voorbeeld van *algoritmisch leren*: de leerling wordt uitgelegd *hoe* hij een optelling moet uitvoeren. Dit wordt daarna ingeoefend.

- Leren lezen. De leraar schrijft een 'a' op het bord, leest deze 'a', laat eventueel de klas dit nazeggen, en zo verder. Dit is leren met voorbeelden, ook *leren met supervisie* genoemd. De leerling krijgt geen algoritme om te zien wat een 'a' is, wel krijgt hij een voorbeeld te zien van wat hij moet leren herkennen, en krijgt ook het resultaat voorgeschoteld.

- Leren fietsen. Gedeeltelijk gebeurt dit algoritmisch: de leerling wordt uitgelegd waarvoor de pedalen dienen, enzovoorts. Maar wat het evenwicht betreft verloopt het leren anders. Zeer weinig mensen weten hoe ze met een fiets een bocht moeten nemen, en uiteindelijk probeert de leerling verschillende zaken zelf, door op de fiets te rijden, eventueel gesteund door de leraar. Hij ziet wat er gebeurt als hij aan zijn stuur draait, van positie verandert, enzovoorts. Op den duur weet hij wat hij moet doen om recht te blijven en om te draaien. In het algemeen weet de leerling dus wel wat zijn doel is. Het leren gebeurt dus niet met voorbeelden, maar uit eigen ervaring: dit heet *leren zonder supervisie*.

De eerste vorm van leren, algoritmisch leren, komt goed overeen met het programmeren van een computer. Essentieel is hier dat de leraar weet hoe de taak wordt uitgevoerd, en dit ook meedeelt aan de leerling.

Bij de andere twee vormen verloopt het proces anders. De leerling moet zelf ontdekken hoe de taak wordt uitgevoerd. Bij leren met supervisie worden wel vrij eenvoudige taken aangeleerd waarbij het juiste resultaat aan de leerling wordt meegedeeld. Dit is bij uitstek een geschikte manier van leren voor een associatief geheugen dat een link moet leggen tussen bepaalde stimuli en een bepaald resultaat. Bij supervisie hebben we een *leerverzameling*: een collectie voorbeelden met het gewenste resultaat. Bij leren zonder supervisie is het verband veel losser; er wordt geen gewenst resultaat opgegeven voor elke deeltaak, maar wel is er natuurlijk een algemeen idee van wat er moet aangeleerd worden.

Essentieel om de mogelijkheden van en de moeilijkheden met leren te begrijpen is de idee van *zinvolle veralgemening*. Bij leren zonder supervisie gaat het over een veralgemening in de vorm van *gelijkaardigheid*: gelijkaardige items worden samen gegroepeerd.

Bij leren met supervisie gaat het over de veralgemening van de indeling in klassen van de leerverzameling naar items erbuiten. Ook hier gaat het over gelijkaardigheid: we klasseren items op dezelfde manier als gelijkaardige items binnen de leerverzameling. Een veralgemening is alleen zinnig met het oog op een bepaald doel. Zinnige veralgemening is daarmee geen typische eigenschap van intelligente systemen, maar wel van levende entiteiten. Een boom bijvoorbeeld is genetisch geprogrammeerd om op een bepaald moment in de lente blaadjes te krijgen, afhankelijk van het weer. Dit is een veralgemening (er is nog nooit een lente geweest met exact hetzelfde weer). Deze is zinnig omdat ze een hoge kans op overleven en voortplanting biedt.

Bij leren gaan we een model opstellen van de werkelijkheid dat veralgemeent. We hopen dat deze veralgemening zinvol is. Of dit zo is hangt af van het probleem: het is bijvoorbeeld mogelijk dat we proberen een classificatie op te stellen zonder dat we voldoende relevante eigenschappen van de items kennen.

1.5 CLASSIFICATIE

Zoals vermeld is een van de belangrijkste taken in AI deze van patroonherkenning. In abstracte zin is dit het probleem waarbij we een of ander object of situatie moeten aanwijzen als behorende tot een bepaalde klasse; met elke klasse komt dan een patroon overeen.

We zullen spreken van een *item* dat we moeten klasseren. Nemen we een voorbeeld: je hebt een financiële instelling die leningen verstrekt aan klanten. Nu blijkt dat een aantal klanten hun lening probleemloos afbetalen, terwijl er bij anderen vertragingen zijn, en dat je bij anderen soms veel moeite hebt om je geld terug te krijgen. Uiteraard zou je graag hebben dat je, bij de beslissing om al dan niet een lening toe te staan, weet of de klant tot de risicogroep behoort.

Voor een programma wordt het object dat, of de situatie die, moet geklasseerd worden gekarakteriseerd door een aantal *meetwaarden*. Het hoeft niet te gaan om echte metingen. Voor het voorbeeld heb je van elke klant een dossier met een aantal gegevens, zoals leeftijd, beroepscategorie, strafblad, en zo verder. Er zijn twee soorten meetwaarden. Sommige metingen kunnen een groot aantal waarden opleveren. We veronderstellen dat deze kunnen uitgedrukt worden door een getal (geheel of met vlottende komma). Andere hebben maar een beperkt aantal mogelijkheden, zoals de beroepscategorie. Deze waarden kunnen we altijd omzetten naar de antwoorden op een aantal ja-neevragen. Als er bijvoorbeeld vijf beroepscategorieën zijn dan kunnen de vragen zijn “Is het de eerste categorie?”, “Is het de tweede categorie?”, en zo verder. De antwoorden kunnen we de waarde 1 of 0 geven. Op deze manier zijn de meetwaarden altijd getallen, wat voor een aantal toepassingen zeer handig is. Op deze manier krijgt het classificatieprobleem zijn standaardvorm: de computer heeft een aantal klassen en moet een getallenrij (die de meetwaarden voor een bepaald item bevat) toewijzen aan een van de klassen. Typische voorbeelden:

- Spraakherkenning: van een geluid is de sterkte bekend bij verschillende frequenties. Men moet bepalen over welk foneem het gaat (dit is een vereenvoudiging: wijzigingen in de tijd spelen ook een rol).
- OCR: van het beeld van een letter is de lichtintensiteit van elke pixel bekend. Welke letter is het?

Merken we op dat bij bepaalde methodes we geen gebruik kunnen maken van getallen, maar dat de methode eist dat we een reeks ja-neevragen hebben. Zojuist hebben we een logische waarde omgezet in een numerieke, nu moeten we het omgekeerde doen. Dit gebeurt

natuurlijk ook in een computer, met de binaire voorstelling van gehele en vlottendekommagetallen, maar zoals we zullen zien is dit voor de meeste methodes geen goede manier, en we zullen later zien hoe we dit probleem oplossen.

We voeren eerst een paar notaties in om onze getallenrijen te kunnen weergeven. De n-dimensionale Euclidische ruimte is de verzameling vectoren met n reële coördinaten. Deze vectoren stellen we voor door vette letters, dus bv. $\mathbf{v} = (v_1, \ldots, v_n)$. Soms beginnen we ook vanaf nul te tellen, zodat we $n+1$-dimensionale vectoren krijgen, $\mathbf{v} = (v_0, v_1, \ldots, v_n)$; v_0 heeft dan een speciale betekenis. 'Losse' reële getallen stellen we meestal voor door hoofdletters: A, B enzovoorts. Er is een nulvector $0 = (0, \ldots, 0)$. Op de ruimte is een inproduct gedefinieerd:

$$\mathbf{v} \cdot \mathbf{u} = \sum_{i=1}^{n} v_i u_i. \tag{1.1}$$

Dit inproduct is lineair: $(A\mathbf{v}) \cdot \mathbf{u} = A(\mathbf{v} \cdot \mathbf{u})$ en $(\mathbf{u} + \mathbf{v}) \cdot \mathbf{x} = \mathbf{u} \cdot \mathbf{x} + \mathbf{v} \cdot \mathbf{x}$. Uit dit product volgt een norm $|| \cdot ||$ gegeven door

$$||\mathbf{u}||^2 = \mathbf{u} \cdot \mathbf{u}. \tag{1.2}$$

$||\mathbf{u} - \mathbf{v}||$ is de afstand tussen $\mathbf{u}$ en $\mathbf{v}$, dit is de lengte van de kortste weg van $\mathbf{u}$ naar $\mathbf{v}$, en wordt ook geschreven als $d(\mathbf{u}, \mathbf{v})$. De weg tussen $\mathbf{u}$ en $-\mathbf{v}$ via 0 is zeker niet korter dan de kortste weg, en dus is $||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$. Het kwadraat van de twee kanten nemen geeft, vermits $(\mathbf{u} + \mathbf{v}) \cdot (\mathbf{u} + \mathbf{v}) = \mathbf{u} \cdot \mathbf{u} + 2\mathbf{u} \cdot \mathbf{v} + \mathbf{v} \cdot \mathbf{v}$,

$$\mathbf{u} \cdot \mathbf{v} \leq ||\mathbf{u}|| \, ||\mathbf{v}||.$$

De ongelijkheid van Cauchy. Als $\mathbf{u} \neq 0 \neq \mathbf{v}$, dan is

$$-1 \leq \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \, ||\mathbf{v}||} \leq 1.$$

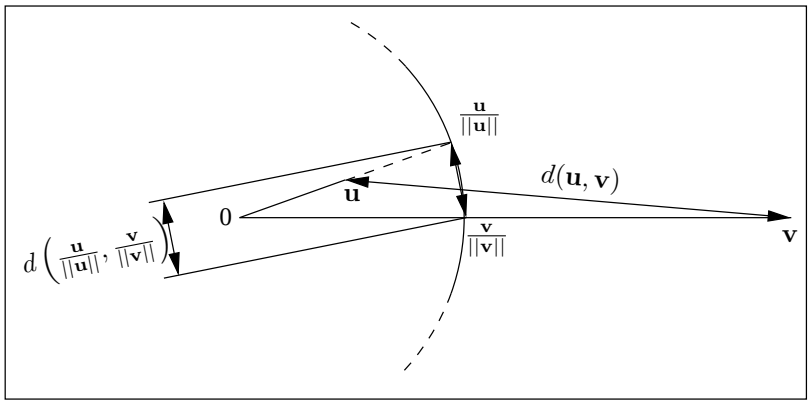
De uitdrukking in het midden noemt men de cosinus van de hoek tussen $\mathbf{u}$ en $\mathbf{v}$.

Het blijkt dat de afstand en de cosinus in veel gevallen een goede indruk geven in hoeverre twee vectoren op elkaar lijken: welke van de twee toepasbaar is hangt van het probleem af. De cosinus geeft een goede maat voor de afstand tussen twee *genormaliseerde* vectoren. Immers,

$$\begin{aligned} d\left(\frac{\mathbf{u}}{||\mathbf{u}||}, \frac{\mathbf{v}}{||\mathbf{v}||}\right)^2 &= \left(\frac{\mathbf{u}}{||\mathbf{u}||}\right)^2 - 2\frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}|| \, ||\mathbf{v}||} + \left(\frac{\mathbf{v}}{||\mathbf{v}||}\right)^2 \\ &= 2 - 2\cos(\mathbf{u}, \mathbf{v}), \end{aligned}$$

vermits $(\mathbf{u}/||\mathbf{u}||)^2 = 1$, en analoog voor $\mathbf{v}$. Soms is het een goed idee om de absolute meetwaarden te bekijken: als we het weer bestuderen lijkt een dag met een temperatuur van 20 ° en 10 uren zon niet op een dag met een temperatuur van 2 ° en 1 uur zon. Hebben we echter twee afbeeldingen waarbij de helderheid van elke pixel van de tweede een tiende is van de overeenkomstige waarde bij de eerste, dan zijn de twee afbeeldingen gelijkend: ze tonen hetzelfde, alleen is de tweede afbeelding veel donkerder. Hier is het dus passend de genormaliseerde vectoren te vergelijken.

De leerverzameling bij leren met supervisie is dus een verzameling met vectoren van meetwaarden, waarbij voor elke vector bekend is tot welke klasse hij behoort. Essentieel is dat het programma *veralgemeent*. Immers, van de vectoren uit de leerverzameling weet je tot welke klasse ze behoren, dus daarvoor moet je geen programma bouwen: je kan gewoon opzoeken in de verzameling. Het is voor nieuwe meetwaarden dat je een classificator wil hebben. De veralgemening moet zinvol zijn. Je zou je kunnen voorstellen



Figuur 1.3. genormaliseerde en niet-genormaliseerde vectoren.

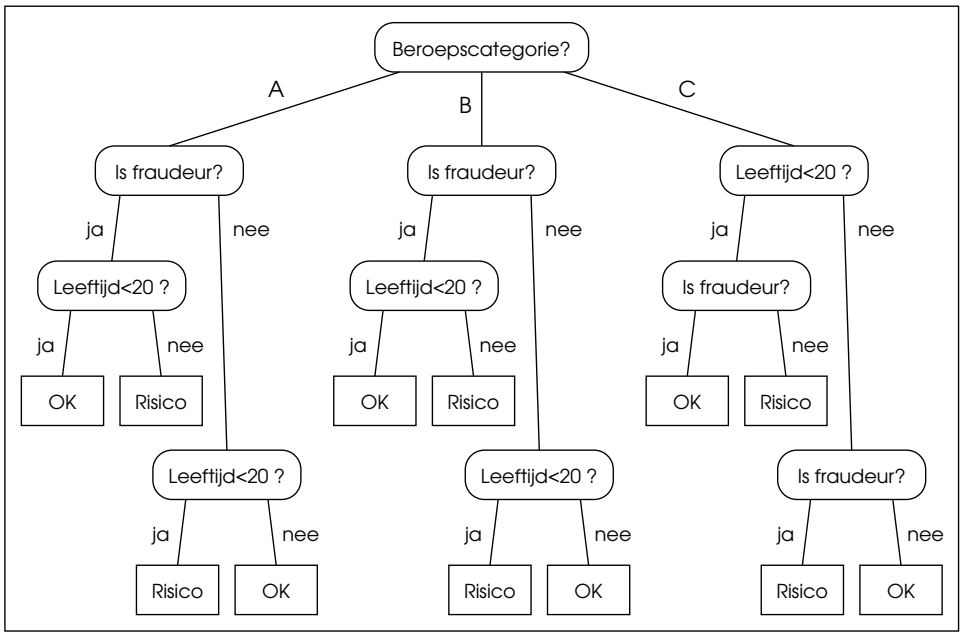
dat een programma gewoon de leerverzameling onthoudt. Als het daarna een vector binnenkrijgt die tot de leerverzameling behoort dan klasseert het deze zoals het onthouden heeft, als de nieuwe vector er niet toe behoort wordt hij bij de eerste klasse ingedeeld. Dit is uiteraard geen nuttig programma, omdat het niet op een zinnige manier veralgemeent. Wat zinnig veralgemenen dan wel is hangt vrij sterk af van het probleem. Om een zicht te krijgen op de kwaliteiten van een klasserend programma als het wordt toegepast op een specifiek probleem gaat men vaak een gedeelte van de leerverzameling apart houden. Heeft de leerverzameling bijvoorbeeld 1.000 elementen, dan zet men 200 elementen apart. Het programma leert met de overblijvende 800 elementen. Daarna laat men het de 200 opzijgezette elementen klasseren. Op deze manier krijgt men een beeld van de mate waarin het programma zinnig veralgemeent.

1.6 INFORMATIE EN BESLISSINGSBOMEN

Een van de klassieke hulpmiddelen bij classificatie is de *beslissingsboom*. Bij zo een boom bevat elke knoop die geen blad is een vraag met een beperkt aantal mogelijke antwoorden. Elk mogelijk antwoord verwijst naar een kind van de knoop. Wil men een item klasseren dan gaat men vanuit de wortel naar beneden en volgt steeds het antwoord dat bij het item hoort tot men bij een blad uitkomt. In het blad staat genoteerd tot welke klasse het item behoort. Beslissingsbomen kunnen gemakkelijk omgezet worden naar programmacode in de vorm van vernestelde `if-then-else`- en `switch-case`structuren of naar een vragenlijst zoals afgebeeld in Figuur 1.5. Het is duidelijk dat de beslissingsboom van Figuur 1.4 niet echt optimaal is. De vraag over de beroepscategorie is volkomen overbodig, vermits het antwoord alleen afhangt van de leeftijd en de fraudebereidheid van de aanvrager. Voor kleine bomen kan het opstellen met de hand gebeuren, maar soms zijn er honderden of duizenden attributen. Er bestaat een algoritme om, voor een gegeven leerverzameling, een goede beslissingsboom op te stellen. Om deze methode te begrijpen is het echter nodig om de notie van *informatie-inhoud* van een bericht te definiëren.

Een bericht heeft alleen maar zin als er iemand (of iets) is die of dat het bericht ontvangt. De ontvanger kan echter alleen iets aanvangen met een bericht als hij het kan interpreteren: hij moet weten waar het over gaat. Ergens heeft hij dus een idee van wat hij zal krijgen. Zo krijgen we de belangrijke elementen voor de informatie-inhoud van een bericht:

- het *bericht* zelf en



Figuur 1.4. Een beslissingsboom.

Vraag1: Wat is de beroepscategorie van de betrokkene? Categorie A: ga naar vraag 2. Categorie B: ga naar vraag 5. Categorie C: ga naar vraag 8.
Vraag2: is de betrokkene jonger dan twintig jaar? Ja: ga naar vraag 3. Nee: ga naar vraag 4.
Vraag 3: is de betrokkene ooit veroordeeld wegens fraude? Ja: terugbetaling van de lening is veilig. Nee: betrokkene vormt een risico.
...

Figuur 1.5. Een vragenlijst.

- de *kennis* van de ontvanger.

Met zijn kennis kan de ontvanger aan elk mogelijk bericht B dat hij kan krijgen een waarschijnlijkheid $p(B)$ hechten. De informatie-inhoud van het bericht wordt dan gedefinieerd

door

$$-\log_2(p(B)) \text{ bits}.$$

De informatie-inhoud is nooit negatief, want de waarschijnlijkheid dat iets gebeurt is nooit groter dan 1. Het grensgeval is dit waarbij $p(B) = 1$. Dit is een zekere gebeurtenis: de ontvanger wist op voorhand dat dit bericht ging binnenkomen. De informatie-inhoud is 0 bits. Dit is logisch: de ontvanger heeft niets bijgeleerd van dit bericht. De informatie-inhoud van een bericht geeft aan in welke mate de ontvanger verrast is door het bericht.

Voorbeeld 1.3

Ik verwacht een byte doorgestuurd te krijgen, maar heb geen flauw idee welke. Alle bytes zijn dus even waarschijnlijk, en hebben een waarschijnlijkheid van $1/256$*. De informatie-inhoud van de byte die ik binnenkrijg is dus* $-\log_2(1/256)$ bits $= 8$ bits*.*

Merk op dat de informatie-inhoud van een bericht niet altijd een geheel getal is. Het is dus zeker niet zo dat een bericht uit evenveel $1/0$-bittekens bestaat als zijn informatie-inhoud aangeeft.

Voorbeeld 1.4

We hebben een alfabet van 4 letters: A, C, G en T. De waarschijnlijkheid dat ze voorkomen is voor A 70,71 %, voor C 12,50 %, voor G 8,39 % en voor T 8,39 %. Aangenomen dat de ontvanger dit weet, dan is de informatie-inhoud van de letters:

$$\begin{array}{lrrl} A: & -\log_2(0{,}7071) & = -\log_2(\sqrt{2}/2) & = 0{,}5 \\ C: & -\log_2(0{,}1250) & = -\log_2(1/8) & = 3{,}0 \\ G: & -\log_2(0{,}0839) & = 3{,}575 & \\ T: & -\log_2(0{,}0839) & = 3{,}575 & \end{array}$$

Merk op dat de informatie-inhoud van een bericht additief is. Stel dat ik bijvoorbeeld eerst een G ontvang en daarna een C. De kans op zo'n combinatie krijg ik door de waarschijnlijkheden te vermenigvuldigen: in dit geval is de waarschijnlijkheid $0{,}1250 \times 0{,}0839 = 0{,}0105$*. Maar de logaritme van een product is de som van de logaritmen, en dus is de informatie-inhoud* $6{,}575$ *bits. Nemen we nu een bericht van 10.000 tekens, met een verdeling zoals opgegeven, en die gekend is door de ontvanger. Wat is nu de informatie-inhoud van dit bericht?*

7071 keer A	*3.536 bits*
1250 keer C	*3.750 bits*
839 keer G	*3.000 bits*
839 keer T	*3.000 bits*
samen	*13.286*

We wisten al voor de ontvangst van het bericht welke letters het bevatte, en deze 13.286 bits zijn de informatie die vervat zit in de volgorde *van de letters. Vandaar dat men ook spreekt van de* entropie *van een bericht. Entropie is een term uit de thermodynamica die het gebrek aan ordening van een dynamisch systeem meet. Hier is de entropie dan een maat voor de mate waarin informatie ontbreekt als de ordening niet gekend is.*

Keren we nu terug naar onze beslissingsbomen. Hier hebben we verzamelingen items die moeten ingedeeld worden in klassen. Als de indeling ontbreekt kennen we alleen de relatieve frequentie van de verschillende klassen; de bedoeling is een beslissingsboom op te stellen die de ontbrekende informatie oplevert. Hoeveel informatie is dit? Neem aan dat

er k klassen zijn, K_1, K_2, ..., K_k. Voor een verzameling S van items is $A(S,i)$ het aantal elementen behorende tot K_i in de verzameling, en $|S| = \sum_{i=1}^{k} A(S,i)$ het totaal aantal elementen van S. De informatie geleverd door een correcte klassering van alle elementen is dan

$$\begin{aligned} E(S) &= \sum_{i=1}^{k} A(S,i) \left(-\log_2 \left(\frac{A(S,i)}{|S|} \right) \right) \\ &= |S| \log_2(|S|) + \sum_{i=1}^{k} A(S,i) \left(-\log_2 \left(A(S,i) \right) \right). \end{aligned}$$

Het ID3-algoritme (ID3 staat voor *Iterative Dichotomiser 3*) voor het opstellen van een beslissingsboom gaat nu inhalig te werk. Het kiest voor de vraag in de wortel juist het attribuut dat het meeste informatie oplevert. Natuurlijk kunnen we alleen een schatting uitvoeren door de leerverzameling te gebruiken. Deelt het j-de attribuut de leerverzameling L in de deelverzamelingen $L_{j,1}$, $L_{j,2}$, ..., L_{j,n_j} dan is de informatie geleverd door dit attribuut gelijk aan

$$I(j) = E(L) - \sum_{m=1}^{n_j} E(L_{j,m}).$$

Informeel kan men zeggen dat $I(j)$ groot is als de verdeling van de verschillende klassen over de $L_{j,m}$ heel anders is dan de verdeling in L zelf. Als de verdeling hetzelfde is dan is $I(j)$ gelijk aan nul, en men kan aantonen dat $I(j)$ nooit kleiner is dan nul. Het algoritme kiest nu die j waarvoor $I(j)$ maximaal is voor de wortel, tenzij het maximum gelijk is aan nul. Als $I(j) = 0$ voor alle j dan behoren ofwel alle items tot dezelfde klasse, ofwel is er op basis van de attributen geen klassering te maken, ofwel faalt het algoritme. Nadat de wortel van de beslissingsboom is geconstrueerd moeten we nog de n_j deelbomen construeren.

Alhoewel de idee achter ID3 logisch is, kan het soms toch tot resultaten leiden die niet optimaal zijn. Dit is het geval als er attributen zijn die alleen *gecombineerd* informatie opleveren. We illustreren het algoritme en zijn beperkingen aan de hand van een voorbeeld. We gaan weer uit van het voorbeeld van het leenrisico, en veronderstellen dat we een leerverzameling hebben met de volgende frequenties:

Beroepscategorie	Jonger dan 20?	Fraudeur?	risico?	frequentie
A	ja	ja	veilig	10
A	ja	nee	riskant	11
A	nee	ja	riskant	18
A	nee	nee	veilig	100
B	ja	ja	veilig	180
B	ja	nee	riskant	8
B	nee	ja	riskant	1
B	nee	nee	veilig	90
C	ja	ja	veilig	50
C	ja	nee	riskant	5
C	nee	ja	riskant	5
C	nee	nee	veilig	50
		Totaal veilig:		480
		Totaal riskant:		48
		Algemeen totaal:		528

Nu gaan we voor elk van de drie attributen na hoe de resulterende verdeling eruit ziet:

Attribuut	waarde	# veilig	# riskant
Beroepscategorie	A	110	29
	B	270	9
	C	100	10
Jonger dan 20?	ja	240	24
	nee	240	24
Fraudeur?	ja	240	24
	nee	240	24

We zien dat noch leeftijd, noch fraudeurschap op zich genomen enige informatie geven over het risico: de verhouding veilige/risicohoudende klanten is steeds 10:1. De beroepscategorieën geven wel een goede indicatie: categorie A is duidelijk veel riskanter dan categorie B. Ter controle berekenen we even $I(1)$ (beroepscategorie is het eerste attribuut):

$$\begin{aligned}
E(L) &= 480(-\log_2(480/528)) + 48(-\log_2(48/528)) = \\
&\quad 480 \times 0.137504 + 48 \times 3.45943 = 232.054 \\
E(L_A) &= 110(-\log_2(110/139)) + 29(-\log_2(29/139)) = \\
&\quad 110 \times 0.337581 + 29 \times 2.26096 = 102.702 \\
E(L_B) &= 270(-\log_2(270/279)) + 9(-\log_2(9/279)) = \\
&\quad 270 \times 0.0473057 + 9 \times 4.9542 = 57.3603 \\
E(L_C) &= 100(-\log_2(100/110)) + 10(-\log_2(10/110)) = \\
&\quad 100 \times 0.137504 + 10 \times 3.45943 = 48.3447 \\
I(1) &= 232.054 - 102.702 - 57.3603 - 48.3447 = 23.647
\end{aligned}$$

Waarbij L_A, L_B en L_C de opsplitsingen zijn van L naargelang beroepscategorie. Als we ook $I(2)$ en $I(3)$ berekenen zien we dat deze nul zijn. $I(1)$ is de grootste van de drie en dus nemen we beroepscategorie als criterium voor de wortel.

Gaan we nu verder met L_A om de meest linkse deelbeslissingsboom op te stellen. De tabel wordt nu

Jonger dan 20?	Fraudeur?	risico?	frequentie
ja	ja	veilig	10
ja	nee	riskant	11
nee	ja	riskant	18
nee	nee	veilig	100
	Totaal veilig:		110
	Totaal riskant:		29
	Algemeen totaal:		139

.

De frequenties voor de attributen worden

Attribuut	waarde	# veilig	# riskant
Jonger dan 20?	ja	10	11
	nee	100	18
Fraudeur?	ja	10	18
	nee	100	11

.

Berekening van de informatiewinst ("jo" staat voor "jonger dan 20?", "fr" staat voor "fraude"; voor $E(L_A)$ hebben we direct de waarde 102.702 ingevuld):

$$\begin{aligned}
E(L_{\text{jo,ja}}) &= 10(-\log_2(10/21)) + 11(-\log_2(11/21)) = \\
&\quad 10 \times 1.07039 + 11 \times 0.932886 = 20.9656 \\
E(L_{\text{jo,nee}}) &= 100(-\log_2(100/118)) + 18(-\log_2(18/118)) = \\
&\quad 100 \times 0.238787 + 18 \times 2.71272 = 72.7076 \\
I(\text{jo}) &= 102.702 - 20.9656 - 72.7076 = 9.0288 \\
E(L_{\text{fr,ja}}) &= 10(-\log_2(10/28)) + 18(-\log_2(18/28)) = \\
&\quad 10 \times 1.48543 + 18 \times 0.63743 = 26.328 \\
E(L_{\text{fr,nee}}) &= 100(-\log_2(100/111)) + 11(-\log_2(11/111)) = \\
&\quad 100 \times 0.15056 + 11 \times 3.33498 = 51.7408 \\
I(\text{fr}) &= 102.702 - 26.328 - 51.7408 = 24.6332.
\end{aligned}$$

Bijgevolg wordt voor het tweede niveau voor het fraudeursattribuut gekozen, en zal bij het volgende niveau het leeftijdsattribuut gekozen worden. De behandeling van L_B is analoog. Bij L_C duikt er echter een probleem op. De frequentietabel wordt

Attribuut	waarde	# veilig	# riskant
Jonger dan 20?	ja	50	5
	nee	50	5
Fraudeur?	ja	50	5
	nee	50	5

Geen enkele van de twee attributen levert op zichzelf informatie op, zodat het algoritme faalt. Als we nu willekeurig het leeftijdsattribuut kiezen, dan krijgen we uiteindelijk weer de beslissingsboom uit Figuur 1.4.

Het voorbeeld dat we bekeken hebben is vrij kunstmatig. In de praktijk komt het bijna nooit voor dat verschillende attributen interageren op zo'n manier dat ze elk afzonderlijk helemaal geen informatie opleveren. Wel bestaat het gevaar dat informatie gegeven door een attribuut wel aanwezig is in combinaties van andere attributen. Als dit attribuut dan gekozen wordt voor de anderen, dan krijgen we de situatie van hierboven, waarbij de boom een overbodige knoop heeft. Bovendien zal men in veel gevallen rekening houden met mogelijke ruis op de meetwaarden (waardoor ze niet exact gekend zijn), en de mogelijkheid dat de gekende attributen eenvoudigweg niet voldoende zijn om een volledige classificatie door te voeren. Daarom worden een aantal verfijningen doorgevoerd:

(1) Er wordt een drempelwaarde ingevoerd voor de informatiewinst: als deze te klein is gaat men de verzameling niet verder opsplitsen. Men krijgt dan een blad waarbij het item tot verschillende klassen kan behoren, elk met zijn waarschijnlijkheid. Dit voorkomt dat men een zinloze opdeling maakt die door de ruis toch geen betekenis heeft.

(2) Ter compensatie gaat men voor elk mogelijk *paar* van attributen de informatiewinst berekenen en, als deze groot is, knopen maken met twee attributen in (en dus navenant meer kinderen).

(3) Als men attributen heeft die een getal opleveren zoals een fysische afmeting, dan gaat men eventueel knopen invoeren met een drempelwaarde voor dit attribuut: alle items met een waarde kleiner dan de drempel gaan naar links, de andere naar rechts. Men moet dan voor elke mogelijke drempelwaarde de informatiewinst berekenen.

ID3 wordt vooral gebruikt om relevante attributen te vinden: men heeft dan duizenden attributen en gebruikt ID3 om relevante attributen of combinaties ervan te vinden. Vaak gebruikt men dan een andere methode om met deze attributen de classificatie door te voeren.

1.7 KLASSEREN ZONDER LEREN

Ook zonder dat er sprake is van leren is er vaak sprake van klasseren. Hier zijn echter de klassen niet vooraf gegeven, zodat er ook geen leerverzameling is. Het programma krijgt een verzameling items die het moet indelen in samenhangende groepen (clustering). Een belangrijk gebied van toepassing is hier beeldanalyse. Het probleem is hier, gegeven een beeld (bijvoorbeeld een foto), te beslissen wat er op staat. Men kan hier denken aan een tweevoudig algoritme:

(1) Een ongesuperviseerd programma groepeert pixels in verschillende clusters.
(2) Een programma dat gesuperviseerd een aantal vormen of combinaties van vormen geleerd heeft klasseert de clusters.

Merk op dat de eerste taak van een heel andere natuur is dan de tweede. Er is hier geen sprake meer van het opsplitsen van de werking van het programma in een leerfase met voorbeelden en een gebruiksfase[3].

Voor we beginnen moeten we een goede definitie hebben van groep. We gebruiken voorlopig de volgende definitie:

Definitie 1 *Een* **groep** *van punten is dat wat door een expert als een groep beschouwd wordt.*

Dit is zonder twijfel geen erg praktische definitie. Ze legt echter de nadruk op een belangrijke eigenschap van het begrip: het is niet erg duidelijk. Bovendien kan het erg afhangen van het terrein van toepassing. We proberen iets meer vat te krijgen door twee redelijke criteria voor te stellen:

(1) Twee punten behoren (waarschijnlijk) tot dezelfde groep als ze zeer dicht bij elkaar liggen (maar: we hebben een expert nodig om de afstandsfunctie te definiëren).
(2) Deze eigenschap wordt transitief verdergezet. Als er, voor twee punten $\mathbf{x}$ en $\mathbf{z}$, een rij punten $\mathbf{y}_1, \ldots, \mathbf{y}_n$ bestaat zodanig dat $\mathbf{x}$ zeer dicht bij $\mathbf{y}_1$ ligt, $\mathbf{y}_1$ zeer dicht bij $\mathbf{y}_2$ ligt, ..., en $\mathbf{y}_n$ zeer dicht bij $\mathbf{z}$ ligt, dan behoren $\mathbf{x}$ en $\mathbf{z}$ tot dezelfde groep.

Bij veel problemen zit er echter ruis op de gegevens, waardoor de verschillende klassen niet duidelijk afgebakend zijn. Daarom past men de transitiviteitsregel niet altijd strikt toe.

Meer dan het geval was bij leren is de voorstelling van de punten van onze verzameling zeer belangrijk. Bij beeldanalyse is het zeker niet voldoende de coördinaten van de pixels op te geven: volgens onze criteria behoren dan alle pixels van een beeld altijd tot dezelfde groep! Iets beter gaat het wanneer we de intensiteit (voor de eenvoud gaan we uit van een zwart-witbeeld met grijswaarden) in rekening brengen. Punten voldoen dan aan criterium (1) wanneer ze én dicht bij elkaar in het beeld liggen, én een gelijkaardige grijswaarde hebben. De ervaring leert dat het ook nuttig is om een maat te gebruiken voor de vraag of de pixel een grenspunt is. Hiervoor kijkt men naar de buren (bv. op hoogstens 3 pixels afstand) van het punt. Als deze allemaal dezelfde grijswaarde hebben als het punt zelf is het zeker geen grenspunt; naarmate het er meer variatie is vergroot de kans dat het een grenspunt is.

[3] Nochtans kan er ook hier sprake zijn van leren: De gesuperviseerde module kan vormen herkennen en op basis daarvan de ongesuperviseerde module meedelen dat sommige pixels verkeerd geklasseerd zijn.

Nadat een passende voorstelling voor de punten geconstrueerd is begint het eigenlijke werk. Eerst maakt men een indeling in clusters, daarna wordt de kwaliteit van de groepering nagegaan. Zeer dikwijls wordt dan besloten om verschillende groepen die niet duidelijk gescheiden zijn samen te voegen. De gebruikte methode hangt af van algoritme tot algoritme.

1.7.1 k zwaartepunten

Bij zeer veel algoritmen dient men vooraf op te geven in hoeveel groepen men de leerverzameling $\mathcal{L}$ wil opdelen. Dit is duidelijk een zwakte van deze algoritmen, vermits men op voorhand uiteraard niet weet hoeveel groepen er zouden moeten zijn. Er kunnen twee fouten optreden:

- Er wordt een te groot aantal opgegeven. Hierdoor wordt een samenhangende groep (of verschillende) opgesplitst. Dit kan worden opgelost door samenhorende groepen op het einde samen te nemen.
- Het opgegeven aantal is te klein, en verschillende groepen worden samengenomen. Men probeert dit op te lossen door het algoritme verschillende malen los te laten op het probleem, met verschillende initialisaties. Men gaat er dan van uit dat niet altijd dezelfde groepen zullen worden samengenomen, en dat men de fout zal opmerken omdat er verschillende groeperingen gemaakt worden.

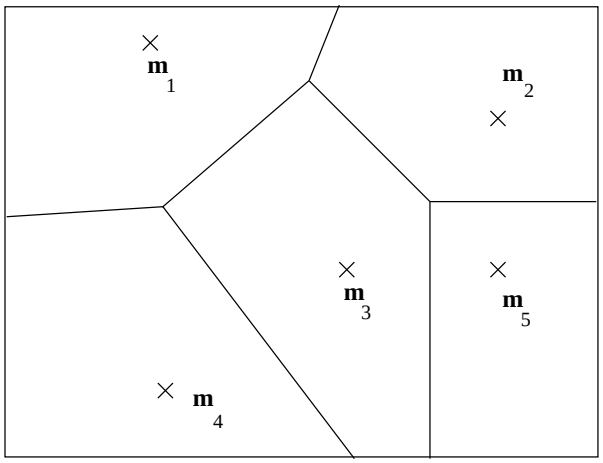
Soms is er ook een voordeel. Het kan immers zijn dat er op verschillende niveaus verschillende groepen zijn. Men heeft bijvoorbeeld drie grote groepen I, II en III, terwijl deze nog verder kunnen worden opgedeeld in kleinere groepen: Ia, Ib, Ic, IIa, IIb, enzovoorts. Door als aantal klassen 3 te kiezen krijgt men de groepen I, II en III. Door meer groepen te kiezen komen de deelgroepen te voorschijn.

De eenvoudigste methode in deze klasse is deze van de k zwaartepunten. Bij deze methode is er geen leren in de echte zin. Ze gaat ervan uit dat de gegevens op een zinnige manier worden voorgesteld: en verkeerde voorstelling, met bijvoorbeeld niet-relevante attributen, leidt dus tot slechte resultaten. Bij deze methode poogt men $\mathcal{L}$ op te delen in k groepen $G_1, \ldots, G_k$, waarbij k vooraf opgegeven is. Elke klasse wordt voorgesteld door haar zwaartepunt $\mathbf{m}_i$. Formeel:

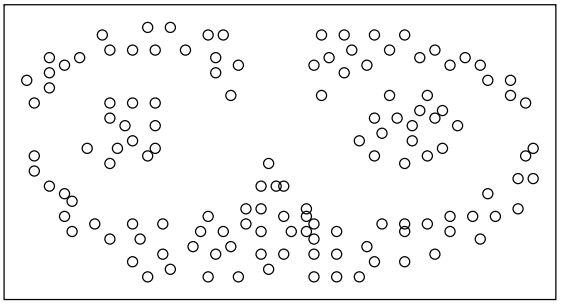
$$\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x} \in G_i} \mathbf{x}.$$

Hier is n_i het aantal vectoren in G_i. Een punt $\mathbf{z}$ wordt toegewezen aan een groep als volgt: Zoek uit de zwaartepunten $\mathbf{m}_1, \ldots, \mathbf{m}_k$ datgene dat het dichtst bij $\mathbf{z}$ ligt. $\mathbf{z}$ wordt dan toebedeeld aan de bijbehorende klasse. Hierdoor krijgt men een indeling die bekend staat als een *Voronojdiagram*, zoals op Figuur 1.6. Elk vakje komt overeen met een groep. Het is duidelijk dat de vorm van een groep hier nogal beperkt is: zo zijn alle vakjes convex. De methode heeft dan ook last bij groepen die niet convex zijn, zoals die gegeven in Figuur 1.7. Het algoritme is gebaseerd op de volgende idee: stel dat onze kandidaten voor de zwaartepunten allemaal ver van een groep liggen. Dan zullen toch vele punten van deze groep bij hetzelfde kandidaat-zwaartepunt geklasseerd worden. Daardoor schuift dit op in de goede richting. Na een tijdje komt elk kandidaat-zwaartepunt in de buurt van 'zijn' groep. Er worden nog maar weinig punten verkeerd geklasseerd, en daardoor wordt de schatting van de zwaartepunten steeds beter. Als de leerverzameling braaf is (de groepen liggen ver uit elkaar zonder veel ruis van punten tussenin, de vorm past goed bij een Voronojdiagram) werkt dit allemaal vrij vlot.

1. Deel de punten willekeurig in, bv. door k punten als voorlopige zwaartepunten te kiezen en naar stap 2b te springen.



Figuur 1.6. Voronojdiagram.



Figuur 1.7. Niet-convexe groepen.

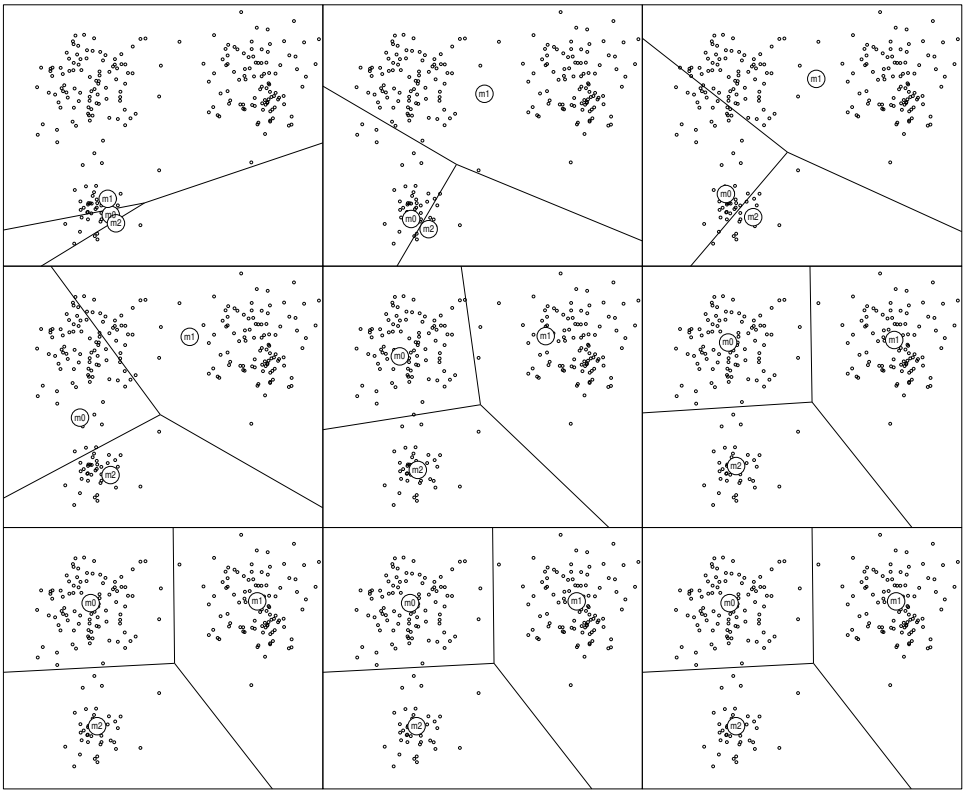
2. Herhaal de volgende stappen, tot er geen veranderingen meer optreden in stap 2b, of tot een maximaal aantal iteraties bereikt is:
   2a. Bereken voor alle groepen het zwaartepunt.
   2b. Deel alle punten van $\mathcal{L}$ opnieuw in.
   2c. Als er een groep is met te weinig elementen, schrap deze groep en deel de punten die tot die groep behoorden opnieuw in.

Opmerking: een vaak gebruikte methode voor indeling voor punt 1. is om gewoon k punten uit de verzameling te nemen als middelpunten voor de verschillende groepen.

1.7.2 Een toepassing

Dit algoritme wordt, met een kleine wijziging, gebruikt om afbeeldingen op te slaan in het GIF-formaat. Bij GIF is er het palet met hoogstens 256 kleurwaarden, opgeslagen in 24-bitnotatie (rood, groen en blauw, 8 bits per kleur). Bij elke pixel wordt aangegeven welke kleur van het palet hij heeft: daar hebben we dus 1 byte voor nodig.

Stel nu dat we een afbeelding hebben met 24 bits per pixel. Als we deze als GIF willen opslaan, moeten we 256 kleuren kiezen voor het palet, en voor elke pixel kiezen welke van deze kleuren we krijgen, en dit met zo weinig mogelijk kwaliteitsverlies.



Figuur 1.8. k-zwaartepuntenmethode in actie

We gebruiken nu alle 24-bitswaarden en beschouwen deze als punten in een driedimensionale ruimte (1 dimensie voor elke kleur). Daarna passen we de k-zwaartepuntenmethode toe, met $k = 256$. Wel met een kleine wijziging: stap (2c) passen we niet toe, zodat we steeds 256 zwaartepunten houden. Daarna ronden we de verkregen coördinaten af, zodat we de zwaartepunten met 24 bits kunnen voorstellen, en steken deze in het palet. Elke pixel krijgt nu de paletkleur die het dichtst bij de originele kleur ligt.

1.8 EEN TOEPASSING: WATSON

In februari 2011 won voor het eerst een computercluster een tv-quiz. **Watson**, een door IBM speciaal voor deze gelegenheid ontwikkelde eenheid, versloeg op overtuigende wijze twee menselijke kandidaten in de quiz *Jeopardy!* Dit is met voorsprong het dichtste dat een computer ooit is geraakt bij het benaderen van menselijk gedrag zoals bedoeld in de Turingtest. De twee meest in het oog springende kenmerken van deze prestatie waren:

(1) Het gebruik van *natuurlijke taal*.
(2) Het feit dat de quiz ging over een zeer breed kennisgebied.

Het is dan ook interessant om eens van naderbij te kijken wat de sterktes en de zwaktes zijn van de gebruikte technieken.

We beginnen met de in- en uitvoer. Watson krijgt zijn vragen in digitale vorm en moet dus geen gebruik maken van spraak- of beeldherkenning. Dit verkleint enigszins de kans

op fouten maar is niet essentieel. Watson kan wel spreken. Hierbij beperkt hij zich echter tot frasen die niet veel verder gaan dan wat Eliza produceert: losse frasen waarin stukken kunnen worden ingevuld. Zo zijn bijvoorbeeld antwoorden van Watson altijd van de vorm 'Who is: <persoonsaanduiding>' of 'What is: <zaakaanduiding>'. Watson herkent wel meervoudsvormen; zo produceerde hij op een gegeven moment het antwoord 'What are: tools'.

Interessanter is natuurlijk om te zien hoe hij vragen beantwoordt. Een belangrijke bron van informatie wordt gevormd door de foutieve antwoorden die Watson geeft. Zoals de presentator van de quiz ook al opmerkt: Watson maakt niet veel fouten, maar de fouten die hij maakt komen vaak heel raar over. Zeer typisch is dat een verkeerd antwoord van Watson zeer vaak wel te maken heeft heeft met elementen van de vraag maar tot de verkeerde categorie behoort. Zo beantwoordt Watson een vraag naar een bepaalde kunstperiode met Picasso (zie verder).

Twee essentiële punten zijn hoe Watson informatie uit de vragen, die immers in natuurlijke taal gesteld zijn, haalt en hoe Watson aan de antwoorden raakt.

We beginnen met het laatste. Watson heeft een geheugenopslag van 15 terabyte. Ter vergelijking: alle teksten op het internet namen in 2005 ongeveer 20 terabyte in beslag. De Engelstalige Wikipedia bevat zowat alle antwoorden op Jeopardy!-vragen en beslaat 27 gigabyte aan tekst. We mogen er dus van uitgaan dat Watson de beschikking heeft over zowat alle relevante internetteksten. Uit de antwoorden van Watson, ook de onjuiste, blijkt bijna altijd dat Watson de juiste informatie wel ter beschikking heeft, maar dat hij niet in staat is om deze informatie op een correcte manier aan de vraag te linken. Het probleem is dus niet de hoeveelheid data maar wel het niet 'begrijpen' van ofwel de vraag ofwel van de tekst die de oplossing bevat.

De enorme hoeveelheid gegevens sluit uit dat deze teksten intensief manueel bewerkt of naar een machinevriendelijke vorm worden omgezet. Wel kunnen automatische bewerkingen, zoals het opbouwen van indexen, op voorhand gebeuren. Eveneens kunnen bepaalde groepen van teksten manueel aangeduid worden zodat ze een speciale functie hebben: zo kan een woordenboek gebruikt worden waarbij de teksten verklaringen zijn, kunnen er synoniemenwoordenboeken zijn die Watson de mogelijkheid geven , en zo voorts. 'Gewone teksten' zijn dan teksten die feiten en verbanden tussen begrippen aangeven.

Het opzoeken van het antwoord op een vraag gebeurt in twee stappen. Eerst dienen de voor de vraag relevante teksten worden opgezocht, daarna dient het antwoord in de teksten gevonden. Voor het opzoeken gebruikt Watson technieken die analoog zijn aan deze die door Google gebruikt worden. Watson selecteert een aantal kernwoorden en -frasen uit de vraag (op dit proces komen we later nog terug) en zoekt teksten waarin deze voorkomen. Zoals bij Google wordt daarbij een rangschikking opgemaakt van de teksten op basis van een aantal criteria en alleen de interessantste worden behouden. In deze teksten wordt dan het begrip opgezocht dat het beste aansluit uit de begrippen uit de vraag.

Om te beginnen dient opgemerkt te worden dat Watson *alleen antwoorden kan geven die uit een enkel begrip of standaardfrase bestaan*. Een goede illustratie hiervan wordt gegeven door de volgende vraag:

Vraag: *(Category: Olympic Oddities) It was the anatomical oddity of U.S. Gymnast George Eyser, who won a gold medal on the parallel bars in 1904.*
Watson: What is: leg. (correct is: He had only one leg).

Het correcte antwoord legt het verband tussen verschillende begrippen en gaat dus Watsons petje te boven. Om de basismanier van werken duidelijk te maken kijken we naar een ander voorbeeld:

Vraag: *(Category: The Art of the Steal) In May 2010 5 paintings worth $125 million by Bracque, Matisse & three others left Paris' museum of this art period.*
Watson: What is: Picasso (correct is: Modern arts).

Het is vrij gemakkelijk om de manier van werken van Watson te emuleren door in Google een aantal kernwoorden van de vraag in te tikken: 'theft', 'Paris', 'museum', '2010', 'Bracque' en 'Matisse' geven als eerste tekst een nieuwsbericht over de diefstal. Hierin wordt één keer de naam van het museum (Museum of Modern Arts) vermeld en drie keer dat de belangrijkste schilders Bracque, Matisse en Picasso waren. Die drie vermeldingen van Picasso vlak bij twee sleutelbegrippen uit de vraag wegen voor Watson meer door dan de enkele vermelding van Modern Arts bij één sleutelwoord. Watson mist hier duidelijk de aanwijzing uit de vraag dat er een kunstperiode gezocht wordt (de twee andere deelnemers uit de quiz gaven ook een verkeerd antwoord, respectievelijk 'cubism' en 'impressionism', wat veel beter aansluit bij het begrip kunstperiode).

Ook een vraag waar hetzelfde mechanisme de mist in gaat is

Vraag: *(Category: EU) As of 2010, Croatia and Macedonia are candidates but this is the only former Yugoslav republic in the EU.*
Watson: Serbia

Het correcte antwoord was hier Slovenië, Servië was zelfs nog geen kandidaat-lid op het ogenblik van de quiz. Maar waarschijnlijk wordt Servië in meer teksten als 'former Yugoslav Republic' betiteld dan Slovenië en wordt in veel teksten gezegd dat Servië *geen* lid of kandidaat-lid van de EU is.

Dat opzoeken van nauw verwante begrippen het hoofdinstrument van Watson is maakt dat hij dan ook vaak antwoorden geeft die vaag iets met de vraag te maken hebben maar in een volledig verkeerde categorie zitten:

Vraag: *This trusted friend was the first non-dairy powdered creamer:*
Watson: milk (juist is: Coffee mate).

Vraag: *(Category: A buck or less) A 15 ounce VO5 moisture milks conditioner from this manufacturer averages a buck online.*
Watson: butter (met lage score; VO5 is een deelmerk van Alberto's, het juiste antwoord).

Watson is echter niet helemaal blind voor aanwijzingen in de vraag die het soort begrip dat gezocht wordt aanduiden. We mogen ervan uitgaan dat Watson geen complexe grammaticale analyse van een vraag kan maken. Dit blijkt bijvoorbeeld uit de volgende vraag:

Vraag: *It's a 4-letter word for a summit; the first 3 letters mean a type of simian.*
Watson: (1) peak, (2) ACME, (3) APEX

Blijkbaar heeft Watson wel door dat hij een woord van 4 letters moet geven, maar dat er een 3-letterwoord dat 'aap' betekent in verscholen is ziet hij niet. Merk op dat van de drie mogelijke oplossingen die Watson geeft er twee zijn die acroniemen (Engels: letter words) zijn, wat blijkt uit het feit dat ze met hoofdletters geschreven zijn; ACME en APEX zijn beide bedrijven.

Een aantal dingen pikt hij toch op. Dit blijkt bijvoorbeeld uit het feit dat hij een onderscheid kan maken tussen 'who is'-vragen en 'what is'-vragen. Hoe zwak hij hier echter in is blijkt uit de vier antwoorden die hij gaf in de 'Beatles People'-categorie, waarin telkens een persoon gezocht werd in de titel van een Beatlesliedje:

Vraag: *(Category: Beatles people) "Anytime you feel the pain, hey" this guy "refrain ..."*
Watson: Who is: Jude.

Vraag: *(Category: Beatles people) This title gal, "Children at your feet..."*
Watson: Who is: Lady Madonna

Vraag: *(Category: Beatles people) "Bang bang" his "silver hammer came down upon her head"*
Watson: What is: Maxwell's Silver Hammer.

Vraag: *(Category: Beatles people) She "died in the church and was buried alone with her name. Nobody came"*
Watson: What is: Eleanor Rigby.

Alsof het een geplande demonstratie was van Watsons zwakheden, krijgen we twee gevallen waarbij de aanduiding van de persoon gelijk is aan de titel ('Eleanor Rigby' en 'Lady Madonna') en twee waarbij de aanduiding maar een deel van de titel is. In één van die twee gevallen isoleert Watson de persoonsaanduiding uit de titel (Jude); in het tweede lukt het niet: hij geeft de hele titel 'Maxwell's Silver Hammer' die een object aanduidt, en geeft dan ook aan dat het voor hem een 'what is'-vraag is. Waar de persoonsaanduiding de gehele titel is heeft Watson één keer door dat het over een persoon gaat (Who is: Lady Madonna) en één keer niet (What is: Eleanor Rigby).

Aangenomen mag worden dat Watson geen volledige grammaticale analyse van een zin maakt, maar afgaat op fragmenten van zinnen en frasen. Kenmerkend voor de vragen van de quiz is dat er niet zozeer een vraag wordt opgegeven maar een verklarende zin waarin het gevraagde wordt aangeduid door een voornaamwoord ('this . . . ', 'She . . . ', his . . .). Bij persoonlijke voornaamwoorden kan Watson uit het gebruikte woord afleiden of het om een persoon (she/he) of een voorwerp/begrip (it) gaat; na this komt een woord dat het gevraagde nader bepaalt ('this guy', 'this city'). Door gebruik van een woordenboek kan in het laatste geval vaak een categorie bepaald worden en kent Watson een hogere probabiliteit toe aan begrippen die in die categorie zitten. Waarschijnlijk gebruikt hij hiervoor een omgekeerd woordenboek dat voor elke eigenschap een lijst van woorden met die eigenschap bevat. Vreemd is dat het mechanisme dat duidelijk gebruikt wordt (bij de 'Hey Jude' vraag ziet Watson dat het een 'who is'-vraag is en isoleert 'Jude' uit de titel) soms niet werkt (Watson vindt 'Eleanor Rigby' een 'what is'-antwoord). Waarschijnlijk is het systeem redelijk onbetrouwbaar en krijgt daarom niet al te veel gewicht. Enkele keren gaat het opvallend goed en slaagt Watson er zelfs in de frase met het voornaamwoord om te bouwen tot een antwoord. In het voorbeeld wordt de frase 'dialect of this' uit de vraag aangevuld tot 'a dialect of Arabic'.

Vraag: *While Maltese borrows many words from Italian, it developed from a dialect of this Semitic language.*
Watson: What is: a dialect of Arabic.

Noteer wel dat dit strikt genomen een *verkeerd* antwoord oplevert omdat de gevraagde taal Arabisch is, maar dat een mens ook geneigd zou zijn om dezelfde fout te maken.

Een ander zwak punt van Watson is dat hij moeite heeft om te weten of twee referenties naar hetzelfde voorwerp verwijzen. Een voorbeeld: bij één vraag heeft Watson als favoriete antwoord Steve Wynn en als alternatief Stephen A. Wynn. Ook bij de volgende vraag gaat het mis:

Vraag: *(Category: US cities) Its largest airport is named for a WWII hero; its second largest, for a WWII Battle.*
Watson: Toronto.

Toronto, Canada, heeft in elk geval een vliegveld genoemd naar een oorlogsheld. Er zijn ook een stuk of vijf Toronto's in de Verenigde Staten, maar die zijn allemaal te klein om een vliegveld te hebben. Misschien heeft Watson zelfs Toronto boven Chicago, het juiste antwoord, gekozen omdat er meer Toronto's zijn in de VS dan Chicago's.

Tenslotte moeten we de vraag stellen in hoeverre Watson in staat is gegevens uit verschillende teksten te combineren. Nemen we het voorbeeld van de vragencategorie *Name the Decade*. Bij deze categorie krijgt de kandidaat twee gebeurtenissen uit hetzelfde decennium en moet dit decennium aanduiden. Op basis van de hierboven geschetste mechanismen is dit onmogelijk te bereiken met gewone teksten, die immers zeer zelden bij een historische gebeurtenis het decennium vermelden. Hoe kan Watson de link leggen tussen een gebeurtenis uit 1922, een uit 1926 en het decennium van de twintiger jaren? De eenvoudigste manier om Watson dit aan te leren is door de teksten te prepareren en eenvoudigweg overal bij elk jaartal het decennium toe te voegen. Als Watson dan twee teksten heeft, een over elke gebeurtenis, zal dit decennium de enige term zijn die in beide teksten voorkomt. Hoewel Watson erin slaagde sommige vragen in deze categorie met een decennium te beantwoordden, faalde het systeem regelmatig. Voorbeeld:

Vraag: *(Category: Name the Decade) Klaus Barbie is sentenced to life in prison & DNA is first used to convict a criminal.*
Watson: (1) 2002, (2) 1987, (3) Lyon (allen met zeer lage score).

Barbie was bekend als 'de slachter van Lyon' en werd veroordeeld in 1987. Waar 2002 vandaan komt is een raadsel.

Sommige vragen bevatten twee aanwijzingen die via verschillende wegen tot het antwoord leiden. Voor een mens is het heel gemakkelijk om te zien of een antwoord juist is: als iets aan beide aanwijzingen beantwoordt zal het wel kloppen. Mensen gebruiken dan ook een strategie die inhoudt dat men eerst voor beide problemen een reeks juiste antwoorden (noem ze R_1 en R_2) zoekt en daarna het (hopelijk unieke) antwoord te nemen dat in beide reeksen voorkomt. Is Watson in staat om iets dergelijks te doen? Een cruciale vraag is of Watson in staat is de twee gedeeltes van de vraag te scheiden. Immers, als de sleutelwoorden in het eerste deel A, B en C zijn en in het tweede deel D, E en F, dan moet Watson teksten opzoeken die ofwel A, B, C ofwel D, E, F bevatten, maar *geen* teksten met combinaties zoals A, C, F. Slaagt hij daarin dan krijgt hij twee groepen relevante teksten. De ene groep zal punten toekennen aan de antwoorden uit reeks R_1, de andere zal punten geven aan R_2, en uiteindelijk zal het juiste antwoord datgene zijn dat van alle teksten een hoge score krijgt. Laat ons een paar voorbeelden van dit soort vragen bekijken:

Vraag: *(Category: Final Frontiers) From the Latin for "end", this is where trains can also originate.*
Watson: (1) finis (97%) (2) Constantinople (13%) (3) Pig Latin (10%)

Het juiste antwoord is *terminal*. Het is niet echt verbazingwekkend dat Watson het juiste antwoord niet vindt: het bedoelde Latijnse woord is *terminus* en Watson begrijpt waarschijnlijk niet dat *terminus* en *terminal* verwante woorden zijn. Wel is opvallend dat Watson zo'n hoge score toekent aan een antwoord dat verbonden is met twee van de sleutelwoorden (*Latin* en *"end"*) maar niet met de twee andere (*trains* en *originate*). Merk wel op dat Watson correct inschat dat het Latijn bij "end" hoort. Het is een raadsel waar Constantinopel vandaan komt: misschien is dat omdat Constantinopel het *"end"* was van een van de beroemdste treinlijnen ter wereld, de Orient Express. Het is overigens niet de enige vraag waarbij Watson een van de twee aanwijzingen compleet negeert.

Vraag: *(Category: Alternate meanings) A thief, or the bent part of an arm*
Watson: (1) Knee (40%) (2) waist (10%) (3) crook (5%)

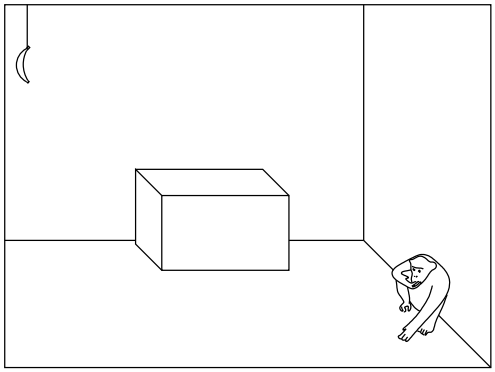
Erg moeilijke vraag voor Watson. Zeer korte vraag met alleen veel voorkomende woorden. Eigenaardig is de hoge score voor *knee*, waar je eerder *elbow* zou verwachten. Toch is Watson erin geslaagd het juiste antwoord (*crook*) op te vissen, zij het met een vrij lage score.

Vraag: *(Category: Final Frontiers) To push one of these paper products is to stretch established limits*
Watson: Envelope

Hier slaagt Watson wel in om het juiste antwoord te vinden, maar het is niet echt duidelijk of hij de twee aanwijzingen gescheiden gebruikt.

De eerste toepassingen in de 'echte wereld' van de technologie achter Watson liggen in de medische gebied. De algemene tekstbasis die gebruikt werd voor Jeopardy werd vervangen door een kleiner corpus van 600.000 medische dossiers in verband met de behandeling van longkanker. Op deze manier kan Watson analyseren wat op een gegeven ogenblik de beste behandeling is voor een patiënt. Merk op dat deze opgave, vanuit het standpunt van AI, eenvoudiger is. Het gaat hier om teksten met een specifiek onderwerp en de vraagstelling is zeer beperkt.

HOOFDSTUK 2
ZOEKEN IN ZOEKRUIMTEN



Figuur 2.1. Een zoekruimte

De oudste tak van AI houdt zich bezig met zoekmethodes in zogenaamde zoekruimten. Dit is het model dat het beste aansluit bij de manier waarop mensen (en bepaalde hogere diersoorten) problemen oplossen. Nemen we een klassiek voorbeeld, waarbij een aap in een kooi wordt gebracht waar aan het plafond een banaan hangt. De banaan hangt te hoog om er direct bij te kunnen, maar er is wel een doos aanwezig. Om bij de banaan te kunnen moet de aap de doos verschuiven en er op klimmen. Hij moet dus een actie uitvoeren (de doos verschuiven) die niet direct resultaat oplevert, maar wel de uiteindelijke actie (de banaan nemen) mogelijk maakt. Apen kunnen dit soort probleem oplossen indien er maar één tussenstap is, of enkele; mensen kunnen een vrij groot aantal tussenstappen aan. Om dit te formaliseren heeft men het begrip *zoekruimte* ingevoerd. De zoekruimte van een probleem is de verzameling van alle mogelijke relevante toestanden. In het banaanprobleem wordt de toestand bepaald door de positie van de banaan, de aap en de doos. Merk op dat de aap het probleem al grotendeels heeft opgelost wanneer hij beseft dat de doos relevant is. In elke toestand is er een aantal *acties* die kunnen ondernomen worden. Elke actie leidt tot een nieuwe toestand. In sommige toepassingen is de nieuwe toestand gekend; in andere is er een kleine of grote mate van onzekerheid. Alles bij elkaar hebben we dus de volgende elementen:

(1) Een gerichte graaf, waarin de knopen de toestanden zijn, en waarin er een verbinding van toestand a naar toestand b is als en slechts als er in a een actie mogelijk is die tot b leidt.
(2) Een begintoestand.
(3) Een doel, dat bestaat uit een verzameling van toestanden.

Merk op dat het doel meestal meer dan één toestand bevat. Zo behoort in het banaanprobleem elke toestand waarin de aap de banaan kan opeten tot het doel, wat ook de positie van de aap of de doos is. De bedoeling is nu om een reeks van acties te construeren die ervoor zorgt dat we één van de doeltoestanden bereiken. In sommige gevallen is het belangrijk dat het doel bereikt wordt met zo weinig mogelijk inspanning. Aan elke actie wordt dan een kost gehecht, en de bedoeling is om de totale kost zo klein mogelijk te maken. In andere gevallen heeft het weinig belang hoe lang de weg naar het doel is. Dit is bijvoorbeeld het geval bij diagnoseprogramma's. Het doel is hier de verzameling toestanden waarin een

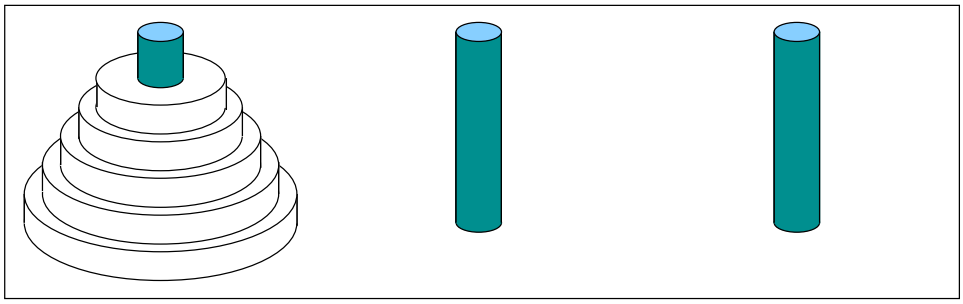
diagnose gekend is. Het belangrijkste is om te weten *welke* diagnose bereikt wordt, niet *hoe* ze bereikt wordt. Natuurlijk is het wel belangrijk dat het doel bereikt wordt (en liefst zo snel mogelijk), en dus zijn ook hier efficiënte zoekmethodes belangrijk.

Een van de eerste toepassingen van dit formalisme was het bewijzen van stellingen uit de formele logica. Een toestand wordt hier beschreven door het geheel van formules die reeds bewezen zijn (axioma's[1] en stellingen). De acties zijn hier afleidingsregels. Deze geven aan hoe uit één of meerdere bewezen formules een nieuwe kan afgeleid worden. Het doel wordt opgegeven in de vorm van een te bewijzen hypothese. Volgens onze definitie is het doel een verzameling toestanden: in dit geval is het de verzameling van toestanden waarin de hypothese bewezen is. Alhoewel het bewijzen van stellingen uit de logica een zeer intelligente bezigheid lijkt, is het probleem vrij eenvoudig en was er al snel een systeem geconstrueerd dat werkte: het aantal afleidingsregels is klein, en de stellingen die het systeem aankon hadden een vrij kort bewijs, zodat blindelings breedte-eerst zoeken een oplossing gaf.

Voor we gaan kijken hoe we op een intelligente manier kunnen zoeken in een zoekruimte, gaan we eerst aangeven hoe men een probleem kan beschrijven.

2.1 STRIPS

STRIPS (STanford Research Institute Problem Solver) is een algemene probleemoplosser. Bij dit programma hoort een taal om het op te lossen probleem te beschrijven. We zullen deze beschrijven aan de hand van een voorbeeld: de welgekende torens van Hanoi. Bij



Figuur 2.2. De torens van Hanoi

dit probleem zijn er vijf[2] doorboorde schijven van verschillende grootte die alle rond een pin zijn aangebracht. De stapel moet worden overgebracht naar een andere pin, en er is een derde pin die als hulp dienst doet. Men mag maar één schijf tegelijk verplaatsen, en er mag nooit een schijf bovenop een kleinere komen te liggen. Om dit probleem met STRIPS op te lossen moeten we het formeel beschrijven. Dit doen we aan de hand van *positieve uitspraken*. Deze zijn er op twee niveaus:

(1) Nuldeordepredicaten. Dit zijn gewoon strings, zoals *DeLuchtIsBlauw* of *veilig*.
(2) eersteordepredicaten. Deze zijn functies met een aantal parameters, die objecten aanduiden. Voorbeelden *HangtAan(schijf4,pin2)*.

[1] We kunnen een axioma opvatten als een bewijs voor zichzelf.

[2] Het verhaal dat bij het probleem hoort is dat er in Hanoi een tempel zou zijn waar de monniken zich voortdurend bezighouden met het verplaatsen van gouden schijven volgens de opgegeven regels. Volgens dit verhaal vergaat de wereld als de opdracht volbracht is. Gelukkig hebben zij honderd schijven in plaats van vijf.

Merk op dat er geen mogelijkheid is om uitspraken te combineren of een negatie uit te drukken. Een *toestand* is een lijst van positieve uitspraken, waarvan verondersteld wordt dat ze waar zijn. Het is gebruikelijk om deze voor te stellen door het enteken $\wedge$. Zo kunnen we dan bijvoorbeeld de begintoestand van ons probleem voorstellen door

HangtAan(schijf1,pin1)$\wedge$
HangtAan(schijf2,pin1) $\wedge$ *HangtAan(schijf3,pin1)* $\wedge$
HangtAan(schijf4,pin1) $\wedge$ *HangtAan(schijf5,pin1)* $\wedge$
NietIs(pin1,pin2) $\wedge$ *NietIs(pin1,pin3)* $\wedge$ *NietIs(pin2,pin1)* $\wedge$
NietIs(pin2,pin3) $\wedge$ *NietIs(pin3,pin1)* $\wedge$ *NietIs(pin3,pin2)*

STRIPS kent het begrip gelijkheid of ongelijkheid niet, en dus moeten we dit formeel definiëren. Vermits STRIPS niets weet over de inhoud van een uitspraak moeten we expliciet zeggen dat *NietIs(pin1,pin2)* waar is én dat *NietIs(pin2,pin1)* waar is. Een *doel is*, zoals gezegd, een verzameling van toestanden. Deze wordt gespecificeerd door een lijst van positieve uitspraken. Een toestand is een doeltoestand als alle uitspraken uit de lijst waar zijn in de toestand. Bij ons probleem is het doel een volledige beschrijving van één toestand, waarin alle schijven aan *pin3* hangen. In het algemeen kunnen er meerdere doeltoestanden zijn. Tenslotte zijn er de acties die een overgang van een toestand naar een andere uitmaken. Een actie kan parameters hebben. Deze worden bij de uitvoering van de actie ingevuld. De beschrijving van een actie bestaat uit twee delen: de *preconditie* en het *effect*. De preconditie is een lijst van uitspraken die waar moeten zijn in de toestand van waaruit we vertrekken. Het effect geeft aan welke uitspraken aan de oude moeten toegevoegd worden, en welke eruit verwijderd moeten worden om tot de nieuwe toestand te komen. Merk op dat een uitspraak verwijderen uit een toestand die de uitspraak niet bevat, of een uitspraak toevoegen aan een toestand waarin hij waar is geen verandering oplevert. Duiden we de preconditie aan met **P**, de lijst van toe te voegen uitspraken met **+** en de lijst met de te verwijderen uitspraken met **–**, dan kunnen we bijvoorbeeld het verplaatsen van de tweede schijf (we tellen van klein naar groot) beschrijven als

beweegschijf2(van, naar, tussen):
P : *NietIs(van,tussen)* $\wedge$ *NietIs(van,naar)* $\wedge$ *NietIs(tussen,naar)* $\wedge$
HangtAan(schijf1,tussen) $\wedge$ *HangtAan(schijf2,van)*
+ : *HangtAan(schijf2,naar)*
- : *HangtAan(schijf2,van)*

Merk op dat we vijf verschillende acties moeten definiëren, één voor elke schijf. We zien hier een voorbeeld van het verschil tussen begrijpen en het vertonen van intelligent gedrag: STRIPS is in staat om het Hanoiprobleem op te lossen, toch als het aantal schijven beperkt is, maar heeft uiteraard geen enkel idee van schijven die moeten verplaatst worden.

De definitietaal van STRIPS is ten zeerste beperkt. Dit heeft als voordeel dat de probleemoplosser zelf vrij eenvoudig is. Nadeel is dat de beschrijving van een bepaald probleem soms ingewikkeld en langdradig wordt. Na STRIPS zijn er dan ook probleemoplossers ontwikkeld die een meer uitgebreide beschrijvingstaal hebben.

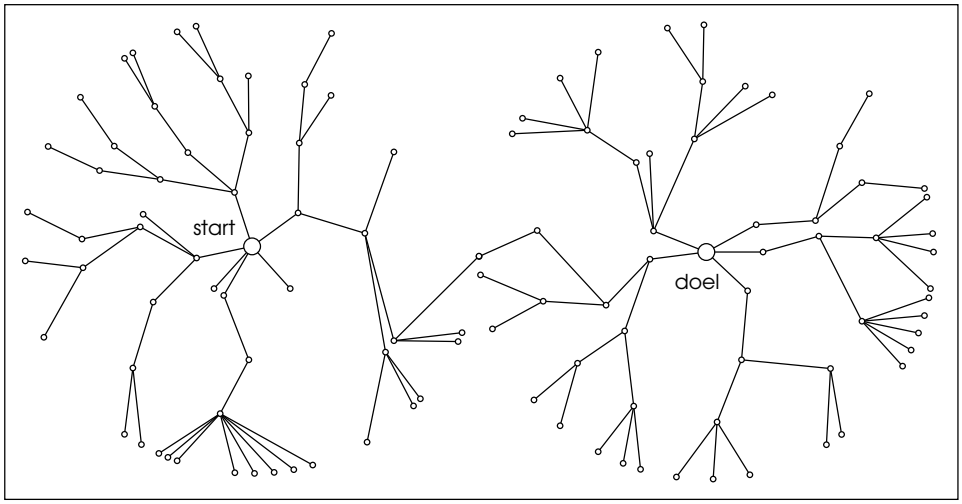
2.2 EFFICIËNT ZOEKEN IN ZOEKRUIMTEN

In principe kan men de zoekruimte blindelings aflopen, bijvoorbeeld met breedte-eerst zoeken. Dit kan echter alleen als de zoekruimte klein is, of indien doel en startpunt dicht bij

elkaar liggen. Een belangrijke maat hier is de *vertakkingsfactor* van een probleem. De vertakkingsfactor is het gemiddeld aantal nieuwe, nog niet bekeken buren van een onderzochte knoop en wordt voorgesteld door b, naar het Engelse *branching factor*. Bij de torens van Hanoi is $b \approx 1,5$. Immers, tenzij alle schijven op elkaar staan zijn er drie mogelijke zetten. De kleinste schijf kan naar een van de twee andere pinnen gaan. Verder is er nog één mogelijke beweging. Echter, één van de drie zetten is gewoon de omkering van de vorige zet en levert dus de ouder van de huidige toestand op. Bovendien is in één geval op twee de kleinste schijf juist verzet. Ze nogmaals verzetten levert een toestand op die bereikbaar is vanuit de ouder van de huidige toestand, en deze is ook niet nieuw. In het algemeen heeft niet elke knoop evenveel buren, en bovendien kunnen sommige buren al ontwikkeld zijn: de vertakkingsfactor is dus geen exact gegeven: hij kan pas na het oplossen van het probleem berekend worden. Toch geeft hij een goed beeld van wat er zich afspeelt bij breedte-eerst zoeken in een graaf. Als we dit doen tot we op een diepte d zitten, dan bezoeken we ongeveer

$$1 + b + b^2 + \ldots + b^d = \frac{b^{d+1} - 1}{b - 1}$$

knopen. Voor b voldoende groter dan 1 is dit ongeveer b^d. Bij schaken bijvoorbeeld kan een speler gemiddeld ongeveer zes zetten doen. Dit betekent dat we, als we tien zetten voor elke speler willen vooruitkijken we zowat $6^{20} \approx 3.6 \times 10^{15}$, dat is 3,6 biljard, knopen moeten ontwikkelen.



Figuur 2.3. Bidirectioneel zoeken

In sommige gevallen –maar niet bij schaken– kan men het aantal onderzochte knopen beperken door vooruit te zoeken vanuit het startpunt, en achteruit te zoeken vanuit het doel, beide met breedte-eerst zoeken. Dit is bijvoorbeeld mogelijk bij de torens van Hanoi, waar er één doeltoestand is en men de zetten kan omkeren. In plaats van ongeveer b^d knopen te ontwikkelen moet men er dan ongeveer $2b^{d/2}$ bekijken, wat een aanzienlijke besparing is. Rekenen we het uit voor $b = 2$ en $d = 20$, dan hebben we met breedte-eerst zoeken een miljoen knopen, en met bidirectioneel zoeken nog ongeveer 2000.

Voorts valt het op dat bij een vertakkingsfactor groter dan 2 de meeste knopen op het diepste niveau zitten (bij ons voorbeeld van daarnet: 3 van de 3,6 biljard knopen zitten

op het diepste niveau). Maar we ontwikkelen niet alle knopen op het diepste niveau: we stoppen bij een oplossing. Als we een methode hebben om de interessantste knopen eerst te zetten, dan vergroot de kans dat we een oplossing vinden als we nog maar weinig knopen op het diepste niveau ontwikkeld hebben en dan kunnen we ook weer werk uitsparen. Hiermee zij we aangekomen bij het heuristieken. Bij het zoeken in de graaf worden er aan elke knoop k twee waarden gehecht:

(1) De *gekende kost* $g(k)$ van de knoop en
(2) de *heuristische kost* $h(k)$ van de knoop.

Wat de gekende kost is van de knoop hangt af van de toepassing. In sommige toepassingen is het alleen belangrijk om een doeltoestand te bereiken, zoals bij diagnoseprogramma's. De gekende kost is dan steeds nul, omdat het niets kost om naar een gekende knoop te zoeken. In andere gevallen is het pad wel belangrijk. Bij planningsproblemen zoals de torens van Hanoi gaat het erom om een reeks acties te bepalen die het doel bereikt, en de bedoeling is om die reeks acties zo goedkoop mogelijk te maken. Bij het Hanoiprobleem is de kost van alle acties even groot, en is dus het aantal acties belangrijk. Bij andere problemen is dit niet het geval. Het klassieke voorbeeld is hier een routeplanner, waarbij het criterium kan zijn om de weg te vinden met de kortste afstand in kilometers (maar ook het soort weg is belangrijk: voor wagens zal de 'kost' voor een kilometer autosnelweg veel minder zijn dan die van een kilometer stadsweg); bij gebruik van openbaar vervoer kan ook tijd het criterium zijn. De gekende kost $g(k)$ van een knoop k wordt berekend door bij de kost van zijn voorganger v de kost $c(v \rightarrow k)$ van de actie die v omzet in k op te tellen. Merk op dat $g(k)$ een schatting is van de reële kost $g^*(k)$ van het kortste pad van de startknoop naar k, en dat steeds $g(k) \geq g^*(k)$. De heuristische kost $h(k)$ is een schatting van $h^*(k)$, de kost om vanuit de knoop het doel te bereiken via het kortste pad. De twee waarden samen geven de totale geschatte kost

$$f(k) = g(k) + h(k).$$

f wordt de evaluatiefunctie genoemd. Een interessante knoop heeft een kleine f-waarde: er is veel kans dat hij op het optimale pad ligt. De f-waarde wordt gebruikt om te bepalen welke knoop eerst ontwikkeld wordt in het zogenaamde beste-eerst zoeken. We hebben een verzamelingen van knopen bij dit algoritme: NOK bevat alle nog niet ontwikkelde knopen waarvan het interessant kan zijn om ze te ontwikkelen.

(1) Steek de startknoop s in de verzameling niet-ontwikkelde knopen NOK met $g(s) = 0$. Geef alle andere knopen een voorlopige schatting $g(k) = \infty$.
(2) Als NOK leeg is stop dan zonder oplossing, ga anders naar (3).
(3) Zoek de knoop k uit NOK met de laagste evaluatiewaarde en verwijder hem uit NOK.
(4) Als k in het doel zit, geef het pad naar k terug en stop; ga anders naar (5).
(5) Voor elke buur b van k: bereken $g_k(b) = g(k) + c(k \rightarrow b)$ en vergelijk $g_k(b)$ met de voorlopige waarde die $g(b)$ al gekregen had. Als $g_k(b) < g(b)$, vervang dan $g(b)$ door $g_k(b)$ en steek b in NOK als b daar niet in zit.
(6) Ga terug naar (2).

Merk op dat breedte-eerst zoeken een speciale vorm is van dit algoritme waarin $h(k) = 0$ en $g(k)$ het aantal stappen is vanuit de startknoop om k te bereiken. Eenmaal een knoop ontwikkeld is kan hij nooit terug in NOK terechtkomen: als we een knoop k ontwikkelen is reeds $g(k) = g^*(k)$ en dit kan niet meer verkleinen. Ook diepte-eerst lijkt op beste-eerst zoeken: ook hier is $h(k) = 0$ maar $g(k)$ is het aantal genomen stappen vermenigvuldigd met -1. Wel mogen we hier $g(k)$, eenmaal gevonden, nooit meer aanpassen.

Waar g in principe bepaald is door het probleem, is dit niet het geval voor h. Het gebeurt maar zeer zelden dat we een exacte waarde kunnen geven voor de *werkelijke* kost om vanuit k het doel te bereiken. Als we dit kunnen hebben alle knopen op optimale pad van start naar doel dezelfde evaluatiewaarde, terwijl alle andere een grotere evaluatiewaarde hebben. Beste-eerst zoeken gaat dan rechtstreeks op het doel af.

We veronderstellen nu dat ons probleem oplosbaar is, en dat er dus een optimaal pad van start naar doel bestaat. We noemen een heuristiek *toelaatbaar* als hij een pad vindt en als het gevonden pad naar het doel optimaal is. Een heuristiek is A^* als voor elke knoop $h(k) \leq h^*(k)$.

Stelling 1 *Bij een probleem waarbij slechts een eindig aantal knopen k_i is met $g^*(k_i) \leq g^*(D)$ is elke A^*-heuristiek toelaatbaar.*

Bewijs
We nemen een A^*-heuristiek h en bewijzen dat er, tot we een doelknoop ontwikkelen, steeds een knoop k' is die voldoet aan de volgende voorwaarden:

(1) k' zit in NOK.
(2) k' ligt op een optimaal pad van s naar D.
(3) De schatting $g(k')$ is correct, m.a.w. $g(k') = g^*(k')$.

Dit is zeker zo bij het begin, wanneer k' de startknoop is. k' kan alleen uit NOK verdwijnen door hem te ontwikkelen, maar dan wordt zijn plaats ingenomen door zijn opvolger k'' op het optimale pad. k'' voldoet aan (3), want

$$g^*(k'') = g^*(k') + c(k' \to k'') = g_{k'}(k'').$$

Alsk' de laatste knoop op het pad is die aan de voorwaarden voldoet dan is de nieuwe $g(k'')$-waarde kleiner dan de vorige en komt k'' zeker in NOK. Omdat h een A^*-heuristiek is geldt bovendien dat

$$f(k') = g(k') + h(k') \leq g^*(k') + h^*(k') = g^*(D).$$

Bijgevolg kan, zolang er een k' in NOK die aan de voorwaarden voldoet, geen enkele knoop k ontwikkeld worden met $g(k) > g^*(D)$, want dan zou $f(k) > f(k')$ zijn. Nu is er een maar eindig aantal knopen waarvoor $f(k)$ kleiner kan worden dan, of gelijk worden aan $g^*(D)$, en kan elke knoop maar een eindig aantal keer ontwikkeld worden. Immers, elke keer we een knoop k ontwikkelen hebben we een kleinere waarde voor $g(k)$ die overeenkomt met de kost van een pad van s naar k, en er is voor k maar een eindig aantal van zulke waarden[3]. Vroeg of laat zal dus een doelknoop d gekozen worden voor ontwikkeling. Deze heeft $g(d) = f(d) \leq g^*(D)$, en bijgevolg is een optimaal pad gevonden. ∎

Het bewijs geeft aan waarom we voorzichtig moeten zijn en waarom in het algoritme reeds ontwikkelde knopen terug in NOK gestoken kunnen worden. Immers, een knoop kan ontwikkeld worden terwijl $g(k) > g^*(k)$, zoals te zien is in Figuur 2.4. Dit voorbeeld toont ook waarom pas gestopt wordt als een doelknoop *uit* NOK gehaald wordt, en niet als er een in gestoken wordt.
Het voorbeeld van Figuur 2.4 toont aan dat een heuristiek, zelfs al is hij een A^*-heuristiek, misleidend kan zijn: het is de heuristiek die ervoor zorgt dat c ontwikkeld wordt voor a. De simpelste heuristiek, $h = 0$, zou wel de juiste volgorde hebben aangehouden. Daardoor zou b maar een keer moeten ontwikkeld worden en dus zou deze heuristiek sneller geweest zijn.

[3] Er kan wel een oneindig aantal *paden* zijn van s naar k als er lussen in de graaf zitten, maar vermits de kost van een actie nooit negatief is kan een lus de waarde van de schatting $g(k)$ niet verminderen.

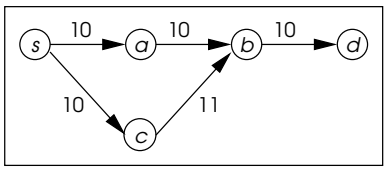
Er is echter een soort heuristieken die dit probleem niet hebben, en dat zijn de monotone heuristieken. Een heuristiek is *monotoon* als

(1) voor elk paar knopen v en k geldt dat $h(v) - h(k) \leq c(v \to k)$, waarbij $c(v \to k) = \infty$ als er geen actie is die v in k omzet.
(2) $h(d) = 0$ voor alle d in de doelverzameling D.

Dit leidt direct tot de volgende eigenschap:

Stelling 2 *Neem een willekeurig pad $k_0, k_1, \ldots, k_n$ tussen twee knopen k_0 en k_n. Dan is de functie $g(k_i) + h(k_i)$ monotoon stijgend (niet noodzakelijk strikt stijgend) als functie van i (vandaar de naam monotone heuristiek). Hierin wordt $g(k_i)$ gerekend langs het pad vanaf het startpunt k_0.*

Inderdaad, voor $i < n$ geldt dat $g(k_i) + c(k_i \to k_{i+1}) = g(k_{i+1})$ terwijl $h(k_i) \leq h(k_{i+1}) + c(k_i \to k_{i+1})$. De twee optellen en de term $c(k_i \to k_{i+1})$ aan beide zijden schrappen levert $g(k_i) + h(k_i) \leq g(k_{i+1}) + h(k_{i+1})$ ∎



Figuur 2.4. Met de waarden $h(a) = 20$ en $h(b) = h(c) = 0$ wordt b twee keer ontwikkeld voor het doel gevonden is. Met de waarden $h(a) = 20$ en $h(b) = h(c) = 10$ wordt b gevonden langs een niet-optimale weg, maar slechts één keer ontwikkeld.

Deze stelling heeft twee gevolgen:

(1) Een monotone heuristiek is A^*. Neem een willekeurige knoop $k = k_0$ en een optimaal pad $k_0, k_1, \ldots, k_n$ naar het doel D, waarbij dus $k_n \in D$. Dan geldt, door de monotoniciteit $g^*(k_0) + h(k_0) \leq g^*(k_n) + h(k_n)$. Nu is $g^*(k_0) = 0$ en $h(k_n) = 0$ (want $k_n \in D$). Wat overblijft is $h(k_0) \leq g^*(k_n)$.
(2) Een knoop k wordt nooit ontwikkeld voor $g^*(k)$ gekend is. Immers, veronderstellen we dat er een knoop k is die als *eerste* ontwikkeld wordt terwijl $g(k) > g^*(k)$. Bekijken we een optimaal pad $s = k_0, k_1, \ldots, k_n = k$. $n > 0$, want k kan uiteraard niet gelijk zijn aan s. Neem op dit pad de laatste knoop die al ontwikkeld werd, k_i. Uit de veronderstelling volgt dat $i < n$ en dat $g^*(k_i)$ gekend was toen k_i ontwikkeld werd. Daaruit volgt dat $g^*(k_{i+1})$ gekend is (en dus $i+1 < n$), maar

$$g^*(k_{i+1}) + h(k_{i+1}) \leq g^*(k_n) + h(k_n) < g(k_n) + h(k_n),$$

wat zou betekenen dat k_{i+1} nog altijd voor k_n in de wachtrij staat.
Vermits een knoop pas ontwikkeld wordt als $g^*(k)$ gekend is kan hij nooit een tweede keer ontwikkeld worden, want $g(k)$ kan nooit meer verkleinen. Wel kan het zijn dat een knoop *ontdekt* wordt met een voorlopige schatting voor $g(k)$ die te groot is: dit is het geval in Figuur 2.4 met de monotone heuristiek $h(a) = 20$ en $h(b) = h(c) = 10$. voor knoop c. Bij een monotone heuristiek is echter gegarandeerd dat deze schatting gecorrigeerd wordt voor de knoop ontwikkeld wordt, een garantie die we niet hebben bij een algemene A^*-heuristiek.

We hebben een zeer slechte monotone heuristiek gezien, namelijk $h = 0$, die resulteert in breedte-eerst zoeken. We hebben ook de optimale heuristiek gezien, $h = h^*$, die rechtstreeks naar het doel leidt, en die ook monotoon is. We kunnen ons afvragen of we een methode hebben om van twee monotone heuristieken te bepalen welke de beste

is. Daarvoor gaan we kijken hoeveel knopen we ontwikkelen[4]. We weten dat de laatste knoop die we ontwikkelen een doelknoop d is met $g^*(d) = g^*(D)$ en $h(d) = 0$. Dit betekent dat we, met een monotone heuristiek, alle knopen moeten ontwikkelen waarvoor $g^*(k) + h(k) < g^*(D)$, en geen enkele waarvoor $g^*(k) + h(k) > g^*(D)$. Met uitzondering van de randgevallen is het dus zo voor twee monotone heuristieken h_1 en h_2, dat als $h_1(k) \geq h_2(k)$ dan h_1 minder knopen doet ontwikkelen dan h_2. Indien voor sommige k geldt dat $h_1(k) < h_2(k)$ en voor andere $h_1(k) > h_2(k)$, dan kunnen we zelfs een betere heuristiek maken die monotoon is en groter dan zowel h_1 als h_2, namelijk h_3 gedefinieerd door

$$h_3(k) = \max\{h_1(k), h_2(k)\}.$$

De snelheidswinst geboekt door een goede heuristiek kan zeer groot zijn. Het is echter wel zo dat op den duur het gebruiken van een ingewikkelde heuristiek meer vertraging kan opleveren (door het werk om de heuristiek uit te rekenen) dan winst.

Een belangrijke klasse van heuristieken is die van de zogenaamde *relaxatieheuristieken*. Deze bekomt men door het probleem te vereenvoudigen op een specifieke manier: men maakt de precondities van de acties minder streng. Nemen we het voorbeeld van het Hanoiprobleem, met de beschrijving van de actie *beweegschijf2*:

beweegschijf2(van, naar, tussen):

P : *NietIs(van,tussen)* $\wedge$ *NietIs(van,naar)* $\wedge$ *NietIs(tussen,naar)* $\wedge$
HangtAan(schijf1,tussen) $\wedge$ *HangtAan(schijf2,van)*

+ : *HangtAan(schijf2,naar)*

- : *HangtAan(schijf2,van)*

We bekomen een vereenvoudigd probleem door in de preconditie *HangtAan(schijf1,tussen)* te schrappen. Doen we iets analoogs voor de andere acties dan bekomen we een nieuw probleem waarin het toegelaten is een grotere schijf op een kleinere te leggen. Doel en kost van acties in het vereenvoudigd probleem zijn hetzelfde als in het oorspronkelijke.

De heuristiek wordt nu gedefinieerd als volgt: om $h(k)$ te bekomen lost men het vereenvoudigd probleem op met k als startknoop, en neemt voor $h(k)$ de kost van de optimale oplossing. We zien dat dit een A^*-heuristiek is. Als we de graaf van toestanden bekijken, dan betekent relaxatie het toevoegen van verbindingen, en eventueel van nieuwe knopen. Uiteraard kunnen daardoor kortere paden ontstaan van k naar de doelverzameling, maar er kunnen nooit paden verdwijnen. Bijgevolg is het optimale pad van k naar D in het oorspronkelijke probleem zeker ook een geldig pad in het vereenvoudigd probleem, en dus is het optimale pad in het vereenvoudigd probleem zeker niet duurder. In het Hanoiprobleem krijgen we zo als heuristiek voor een toestand het aantal schijven dat niet aan de doelpin *pin3* hangt. Het definiëren van de relaxatie dient wel met enige zorg te gebeuren. Het heeft bijvoorbeeld geen zin om in de acties van het Hanoiprobleem de voorwaarden die stellen dat de pinnen *van* en *naar* verschillend moeten zijn te schrappen, omdat het nieuwe probleem dan geen kortere oplossing heeft dan het oude. Bepalen van de heuristiek voor de startknoop houdt dan in dat men het probleem oplost met breedte-eerst zoeken: dit zal zeker het oplossen niet vereenvoudigen.

In het tweede deel van de cursus zullen we massief lerende systemen bekijken in de vorm van neurale netwerken. Is er nu een klasse van problemen waarbij de zoekruimte altijd dezelfde is maar waarin start en doel kunnen variëren, dan kunnen we deze netwerken laten leren door ze opgeloste problemen in de juiste vorm voor te schotelen. Op deze manier zullen ze in staat zijn om te leren voorspellen, aan de hand van een aantal kenmerken

[4] De schatting die we geven werkt niet voor niet-monotone heuristieken, waar we sommige knopen herhaalde malen ontwikkelen. Het aantal knopen klopt wel, maar het aantal ontwikkelingen niet.

van toestanden, wat h^* is. Deze voorspellingen zijn niet exact, maar kunnen vaak goede benaderingen opleveren. Ze hebben wel het nadeel dat er geen enkele garantie is dat ze monotoon zijn, of zelfs maar A^*. Hierdoor is het mogelijk dat het gevonden pad niet optimaal is. Aan de andere kant, als de schatting door een neuraal netwerk dicht bij de realiteit ligt, zal er zeer snel een pad gevonden worden met een kost die niet veel groter is dan het optimale pad. Dit is te prefereren boven een systeem dat de optimale oplossing in principe vindt, maar hiervoor zoveel tijd en geheugenruimte nodig heeft dat het in de praktijk niet werkt.

2.3 SPELBOMEN

In de voorgaande paragrafen zijn we er steeds van uit gegaan dat we volledige kennis hadden over het resultaat van acties. Bij veel problemen is dit niet het geval. Dit kan verschillende oorzaken hebben. Eén klasse van zulke problemen is dat van spelen, waarbij niet op voorhand gekend is welke acties de tegenstander onderneemt. We bekijken hier alleen maar nulsomspelen met 2 spelers (Eng: *two player zero sum games*)[5]. Bij een nulsomspel kan iemand alleen winst boeken ten koste van de tegenstander, en de winst van de ene is gelijk aan het verlies van de ander. Bij het spelen van een nulsomspel gaan we ervan uit dat elke speler probeert zijn winst te optimaliseren.

Nemen we het volgende, uiterst domme maar zeer instructieve nulsomspel. Er zijn twee spelers, speler 1 en speler 2. Het spel gaat als volgt: speler 1 mag een willekeurig (positief) bedrag kiezen. Daarna beslist speler 2 of speler 2 dit bedrag moet betalen aan speler 1, of dat speler 1 de helft van dit bedrag moet betalen aan speler 2. Als speler 1 zeer dom is, kiest hij een zeer hoog bedrag in de hoop dat speler 2 kiest dat speler 2 moet betalen. Maar een redelijke speler 1 zal ervan uitgaan dat speler 2 steeds het alternatief kiest dat voor speler 2 het meest voordelig is, en kiest dus een bedrag van[6] €0. Een iets minder dom spel is een gewijzigde versie van Nim. Op de tafel tussen de twee spelers liggen een aantal stapels jetons. De speler die aan zet is moet een stapel nemen, en deze verdelen in twee ongelijke stapels (dus als de oorspronkelijke stapel 4 jetons had mag 3+1 wel, maar 2+2 niet). De speler die niet meer kan zetten (omdat alle stapels 1 of 2 jetons bevatten) verliest. De winnaar krijgt als winst een aantal punten gelijk aan het aantal overblijvende stapels[7].

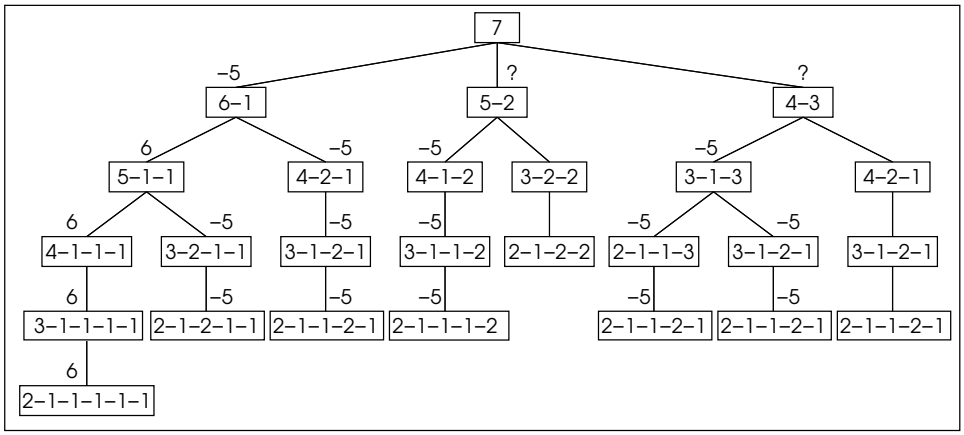
We kunnen nu een boom opstellen waarin getoond wordt wat de mogelijke zetten zijn, zoals getoond in Figuur 2.5. We noemen nu de speler die begint Max en de andere speler Min. Bij een aantal eindposities hebben we de winst of het verlies van Max uitgetekend. Hoe bepalen we nu de te verwachten score voor andere posities? Dit gaat van onder naar boven. Onderstel dat de score voor alle kinderen van een knoop bekend is dan

(1) Als Max aan zet is (zo'n knoop noemen we een maxknoop) dan nemen we de grootste score. Max probeert immers zijn winst te maximaliseren.
(2) Als Min aan zet is (een minknoop) dan nemen we de kleinste score, want Min wil de winst van Max minimaliseren.

[5] Een spel is elke situatie waarin een aantal actoren (de spelers) proberen zo veel mogelijk winst of zo weinig mogelijk verlies te realiseren voor zichzelf. Een typisch voorbeeld van een spel met som verschillend van nul is een beurscrash. Als de aandelenkoersen beginnen ineen te storten en alle aandeelhouders zouden weigeren te verkopen dan blijft het verlies beperkt. Als echter iedereen verkoopt en enkele aandeelhouders weigeren te verkopen dan lijden deze laatsten nog meer verlies dan de anderen. Dus verkoopt toch iedereen. Dit is, in drie zinnen, de reden waarom John Nash de Nobelprijs economie heeft gekregen.

[6] We gaan ervan uit dat het spel maar één keer gespeeld wordt. Anders kan speler 2 proberen een strategie te ontwikkelen om, door kleine bedragen zelf te betalen, speler 1 aan te zetten een zeer groot bedrag op te noemen, en dit dan te innen.

[7] Dit is geen nulsomspel. De analyse blijft echter gelijk. Als we aannemen dat de verliezer evenveel punten verliest als de winnaar er wint, is het namelijk weer wel een nulsomspel.



Figuur 2.5. Een spelboom voor de gewijzigde versie van Nim.

Dergelijke evaluatie heet minimax. Als een programma een bepaalde toestand analyseert om tot een beste zet te komen, dan is het niet geïnteresseerd om te weten wat de score is voor de andere zetten. Hiervan wordt gebruik gemaakt met de techniek die bekend staat als alfa-betasnoeien (Eng. *alpha-beta pruning*). De basisidee is dat, als de speler al weet van een bepaalde zet dat het niet de beste is die hij kan doen, dat hij dan niet alle mogelijkheden meer afgaat. Nemen we als voorbeeld een schaakspel, waarbij speler 1 ziet dat zijn tegenstrever een repliek heeft op een zet X die speler 1 duidelijk zou doen verliezen (terwijl hij betere zetten kent). Speler 1 gaat deze zet dan zeker niet doen, en gaat niet alle mogelijke domme reacties van speler 2 op zet X verder uitwerken.

We zien in Figuur 2.5 dat de zet 7→6-1 een score oplevert van -5. Van de zet 7→5-2 weten we niet hoeveel hij oplevert. We weten wel dat Min erop kan repliceren met een zet die een score van -5 oplevert. Het kan zijn dat Min een (voor hem) nog betere zet heeft, en dan is de waarde van de zet 7→5-2 minder dan -5, maar in geen geval is de waarde van de zet groter dan -5. Bijgevolg is deze zet niet beter dan de zet 7→6-1, en heeft het geen zin om de exacte score van deze zet te zoeken.

Meer algemeen zoeken we een pad naar een blad waarbij zowel Max als Min steeds de beste zet kiezen. Op dit pad is de score van alle knopen gelijk. Laat ons deze s_{opt} noemen, de optimale score.

Moeten we alle knopen volledig evalueren om de optimale score te vinden, samen met het kind van de wortel dat voor die score zorgt?

Om dit probleem op te lossen voeren we het begrip *voorbroer* in van een knoop. Een knoop b is een voorbroer van k als hij een broer is van k, of een broer van een van de voorouders van k (zie Figuur 2.6).

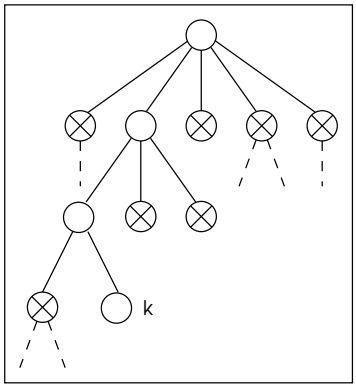
Nemen we een knoop k en veronderstel dat we $s(k)$ vervangen door een schatting f. Hierdoor veranderen de getalwaarden bij nul of meer voorouders van die knoop. Veronderstellen we even dat f groter is dan $s(k)$. Er zijn twee mogelijkheden:

(1) k is een minknoop. Als f groter is dan het maximum van zijn broers, dan wordt de s-waarde van de ouderknoop vervangen door de grotere waarde f, waardoor we in geval 2 belanden. Is er echter een broer met s-waarde groter dan f verandert er niets.

(2) k is een maxknoop (en niet de wortel). Als we de echte waarde van $s(k)$ niet kennen dan weten we niet of hier iets verandert bij de ouder. In elk geval is de nieuwe s-waarde voor de ouder hoogstens f en kan ze alleen maar groter zijn dan de echte

s-waarde, zodat we terug naar geval 1 gaan.

Samenvattend: de te grote schatting f levert geen probleem als er een voorbroer is die een minknoop is met een s-waarde groter dan f. Door de nodige tekens te veranderen krijgen we dat we de s-waarde van een knoop mogen vervangen door een te kleine schatting f als er een voorbroer is die een maxknoop is met s-waarde kleiner dan f. Om deze criteria te kunnen toepassen moeten we wel weten of f te groot of te klein is, maar $s(k)$ zelf moeten we niet kennen.



Figuur 2.6. De voorbroers van een knoop

Wanneer werkt dit? Nemen we een minknoop. De score van deze knoop is het minimum van de scores van de kinderen. Als we dus een gedeelte van de kinderen geëvalueerd hebben, dan hebben we een voorlopig minimum f. Er kunnen nog kinderen zijn met een nog lagere score: het voorlopig minimum is dus een bovengrens voor de score van de minknoop. Als f verkeerd is dan is f te groot: we mogen stoppen als er een voorbroer is die een minknoop is met gekende s-waarde groter dan f. Stellen we dus

$$\alpha = \max_{b \in B^-(k)} s(b),$$

waarin $B^-(k)$ de verzameling is van alle linkse voorbroers van k die minknopen zijn. dan mogen we stoppen met kinderen van k te evalueren als $f < \alpha$.

Bij een maxknoop krijgen we een ondergrens f en mogen we stoppen als f groter is dan

$$\beta = \min_{b \in B^+(k)} s(b),$$

waar $B^+(k)$ de verzameling voorbroers is die maxknopen zijn. Natuurlijk zijn alleen de voorbroers die links van het pad naar k staan reeds behandeld.

Soms zijn er meerder optimale paden. Als we al deze optimale paden willen kennen moeten we de strikte ongelijkheid in de stopvoorwaarde $f < \alpha$ of $f > \beta$ gebruiken. Als we alleen *een* optimaal pad willen mogen we ook stoppen bij $f = \alpha$ of $f = \beta$.

In ons Nimvoorbeeld is, na evaluatie van knoop 6-1, α gelijk aan -5. Nadat kind 4-1-2 van 5-2 geëvalueerd is heeft 5-2 een bovengrens van -5 gekregen, en wordt dus niet verder onderzocht.

Het is duidelijk dat de efficiëntie van alfa-betasnoeien beïnvloed wordt door de volgorde waarin we de kinderen evalueren. Immers, als het eerste kind van een maxknoop een zeer goede zet is dan wordt α meteen zeer groot, en dan is de kans op een waarde van $s(k)$ die kleiner is dan deze α ook groot. Op zijn best zal alfa-betasnoeien voor veel knopen alleen het eerste kind evalueren, op zijn slechtst levert alfa-betasnoeien geen besparingen op omdat de beste zet voor een speler pas op het einde gevonden wordt. Noemen we een *ronde* een combinatie van een zet van Max gevolgd door een zet van Min (normaal heet een ronde ook een zet, maar dat is verwarrend) en noemen we b de vertakkingsfactor per ronde dan is, als we een volledige boom maken voor k rondes, het totaal aantal te evalueren bladeren voor minimax van de orde $O(b^k)$ en bij optimale alfa-betasnoei van de orde $O(b^{k/2})$. Voor onze tien rondes in het schaakspel gaat onze schatting dan van 3,6 biljard naar 60 miljoen. Maar zelfs met alfa-betasnoeien zal men er niet in slagen een volledige boom te ontwikkelen voor ingewikkelde spelen zoals schaakspelen: het gemiddelde schaakspel telt meer dan 10 rondes.

Men gaat dan de spelboom slechts ontwikkelen tot op een zekere vooraf ingestelde diepte. Voor de knopen op die diepte gaat men een schatting maken van de score aan de hand van een aantal kenmerken van de positie. Deze functie noemt men de *statische evaluatie* van de knoop. In paragraaf 9.2 zullen we zien hoe zo een statische-evaluatiefunctie kan geleerd worden door een neuraal netwerk. Dynamische evaluatie is dan het bepalen van de evaluatie van de andere knopen door minimax met snoeien. Hierbij worden de knopen geordend door middel van de statische-evaluatiefunctie, of door een vereenvoudigde versie daarvan. Hierbij moet men een afweging maken. Men kan soms de beste resultaten bekomen met een diepe boom en een slechte evaluatiefunctie, maar soms is een betere evaluatiefunctie met een ondiepe boom beter. Maar zo'n betere functie zal meestal meer rekenwerk meebrengen, en dus moet de diepte inderdaad verkleind worden. Voor een spel als schaken kan men dan nog een tweede heuristiek invoeren die aangeeft hoe betrouwbaar de statische evaluatie van een knoop is. In het schaakspel zijn er indicaties dat een fase interessant is, zoals bij het slaan van een stuk. In dergelijke situaties is statische evaluatie niet betrouwbaar. Veel schaakprogramma's zullen dan ook knopen die op de normale maximale diepte liggen maar die interessant zijn nog enkele zetten verder ontwikkelen.

HOOFDSTUK 3
EXPERTSYSTEMEN

Zoals we gezien hebben zijn met expertsystemen de eerste successen geboekt op het gebied van kunstmatige intelligentie. Bij een expertsysteem wordt de kennis van een expert bij het oplossen van bepaalde problemen opgenomen in het programma. Het spreekt vanzelf dat de *gebruiker* van een expertsysteem zelf een zekere kennis van het probleemgebied moet hebben: hij moet de conclusies van het systeem kunnen begrijpen en toepassen. Als voorbeeld kunnen we MYCIN aanhalen. MYCIN is een expertsysteem dat ontwikkeld werd om te helpen bij de diagnose van bepaalde besmettelijke bloedziekten. MYCIN is beter in het stellen van een diagnose dan een arts die geen specialist is in het gebied, maar de gebruiker moet wel een arts zijn die de diagnose begrijpt en de juiste behandeling kan voorschrijven.

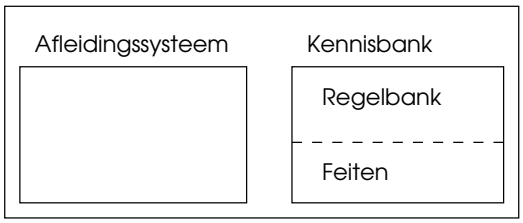
Het eerste werkelijk bruikbare systeem was DENDRAL, dat rond 1970 ontwikkeld werd. De bedoeling was een systeem te ontwikkelen dat de moleculaire structuur van een stof kon bepalen aan de hand van gegevens verkregen met een massaspectrograaf. Een van de toepassingen was het bepalen van de structuur van de bodem van de planeet Mars. Het probleem was niet met traditionele methodes op te lossen. Met een klassiek programma zou men alle mogelijke moleculaire structuren onderzoeken. Het bleek echter dat een expert gebruik maakt van een aantal vuistregels die het aantal relevante mogelijkheden sterk beperkt. In DENDRAL zijn deze vuistregels geformaliseerd. Sinds 1970 zijn ettelijke duizenden expertsystemen ontwikkeld, waarbij een aantal problemen geanalyseerd en opgelost werden. Door de jaren heen is er dan ook nogal wat evolutie in de opvattingen over en de complexiteit van de systemen. De eerste generatie expertsystemen had alleen een vrij ongeordende verzameling van regels (regelbanken). Later werden er verfijningen ingevoerd. De twee voornaamste zijn:

(1) Het werken met onzekere conclusies.
(2) De invoering van zogenaamde *frames*.

Naarmate de omvang van de systemen groeide werd men geconfronteerd met twee problemen, en de twee zojuist vermelde verfijningen bieden oplossingen hiervoor. Het eerste probleem is dat van de aanwezigheid van contradicties: verschillende regels kunnen een tegenstrijdig resultaat opleveren. Het tweede probleem heeft voornamelijk te maken met de ordening van grote hoeveelheden kennis. We bespreken de twee problemen en hun oplossingen, nadat we eerst de basisprincipes van een eenvoudig systeem bekeken hebben.

3.1 EENVOUDIGE SYSTEMEN

In zijn eenvoudigste vorm bestaat een expertsysteem uit twee hoofdcomponenten: een afleidingsmechanisme en een kennisbank (Eng.: *knowledge base*). De kennisbank bevat een aantal *regels* en een aantal *feiten*. Een feit is een uitspraak die waar is. De feiten in de



Figuur 3.1. Kennisbank.

kennisbank zijn die uitspraken waarvan we weten dat ze waar zijn. Bij eenvoudigste vorm van een uitspraak gaat het om de waarde van een variabele. Variabelennamen zullen we schrijven met hoofdletters, toegekende waarden door kleine letters, of cijfers. Voorbeeld:

AS=rond

Een regel heeft altijd de vorm **als**. . . **dan**. . . . Voorbeeld:

als AS=rond **en** MATERIAAL=staal
dan SMERING=olie

Het gedeelte tussen de **als** en de **dan** heet de *premisse*. Een premisse bestaat uit een reeks van uitspraken, gescheiden door het trefwoord **en**. Het deel na de **dan** is de conclusie. Ook deze bestaat uit een lijst van uitspraken, gescheiden door het trefwoord **en**[1].

Een probleem bestaat uit de gegevens en een doel. De gegevens vormen een lijst van feiten die alle gekend zijn. Ook het doel is een lijst van uitspraken. Het doel is bereikt als één uitspraak uit de lijst (en niet alle uitspraken) is afgeleid uit de gegevens. Zo kan het doel zijn de waarde van een bepaalde variabele, bijvoorbeeld RESULTAAT, te bepalen. Het doel is dan de verzameling van alle uitspraken van de vorm RESULTAAT=. . . , en het doel is bereikt als een uitspraak van deze vorm is afgeleid uit de gegevens.

Eenmaal een probleem is opgegeven aan het expertsysteem, komt het afleidingsmechanisme in werking. Fundamenteel zijn er twee mogelijkheden om een probleem op te lossen: *voorwaarts redeneren* en *terugredeneren*.

Bij voorwaarts redeneren wordt uitgegaan van de gegevens. Het afleidingsmechanisme overloopt alle regels tot het een regel vindt waarvoor alle uitspraken uit de premisse tot de gegevens behoren. Het voegt dan alle uitspraken uit de conclusie toe aan de gegevens. Als een van de uitspraken uit de conclusie tot het doel behoort is het probleem opgelost: het systeem geeft dan ook het gevonden uitspraak terug als resultaat. Als geen uitspraak uit de conclusie tot het doel behoort, dient verder te worden gezocht. Het afleidingsmechanisme zoekt een nieuwe regel die het kan toepassen, en zo verder. Merk op dat het geen zin heeft een regel die al gebruikt is nog te onderzoeken. Immers, alle uitspraken uit de conclusie zijn al opgenomen in de gegevensverzameling, en gebruik van de regel zou dus geen nieuwe feiten opleveren. Anderzijds kan er een regel zijn die nog niet gebruikt kon worden omdat niet alle uitspraken uit de premisse tot de gegevens behoorden, en die nu wel bruikbaar is geworden: er zijn immers feiten aan de gegevens toegevoegd.

Het kan natuurlijk zijn dat er na verloop van tijd geen regel meer kan gevonden worden die kan worden toegepast. In dat geval kan het systeem het probleem niet oplossen. In pseudocode ziet het algoritme voor voorwaarts redeneren er als volgt uit :

[1] Een *Hornclausule* is een regel met hoogstens één uitspraak in de conclusie. Men kan met Hornclausules werken: een regel met n uitspraken in de conclusie is dan equivalent met n Hornclausules. Een Hornclausule kan ook nul uitspraken in de conclusie hebben. Dit wordt opgevat als de waarde *onwaar*. Zulke regels duiden aan dat hun premisse nooit waar mag zijn.

1. Initialiseer de gegevensverzameling en de doelverzameling.
2. Zolang de gegevensverzameling en het doel geen gemeenschappelijk element bevatten, en er toepasbare regels zijn, doe het volgende:
   2a. Neem een regel in de regelbank waarvoor de premisse een deel is van de uitsprakenverzameling.
   2b. Voeg de uitspraken uit de conclusie toe aan de gegevens, en verwijder de regel uit de regelbank.
3. Deel het resultaat (of het gebrek eraan) mee aan de gebruiker.

Merk op dat we de logische operator **of** niet gebruiken. Dit is logisch, als er een **of** zou staan in de premisse van een regel, dan kunnen we de regel vervangen door twee regels zonder **of**. De regel

als AS=vierkant **of** AS=driehoekig
dan KANDRAAIEN=nee

is gewoon een verkorte notatie voor

als AS=vierkant
dan KANDRAAIEN=nee

als AS=driehoekig
dan KANDRAAIEN=nee

In een reëel systeem zal de verkorte notatie voor deze twee regels dikwijls wel toegelaten zijn. In een conclusie is de **of** niet bruikbaar. Immers, als de premisse van de 'regel'

als AS=rond
dan SMERING=olie **of** SMERING=vet

waar is, dan weten we niet welke van de twee uitspraken in de conclusie aan de gegevens mag worden toegevoegd.

Merken we op dat voorwaarts redeneren kan gebeuren in $O(g)$, waarin g de grootte van de regelbank is (de grootte van de regelbank is de som van de groottes van de regels; de grootte van een regel is het aantal uitspraken in premisse en conclusie samen). Hiervoor hebben we twee extra gegevensstructuren nodig:

(1) Een structuur *LijstMetRegels* die voor elke uitspraak bijhoudt in welke regels ze in de premisse voorkomt.
(2) Een structuur *premisseteller* die voor elke regel bijhoudt hoeveel uitspraken in de premisse nog niet gevalideerd zijn.

Met deze gegevensstructuren kan het algoritme van *voorwaarts aaneenschakelen* (Eng: *forward chaining*) als volgt gegeven worden: :

1. Initialiseer de gegevensverzameling en de doelverzameling.
2. Voor elke regel r, initialiseer $premisseteller(r)$ op het aantal uitspraken in de premisse.
3. Zolang de gegevensverzameling en het doel geen gemeenschappelijk element bevatten, en de gegevensverzameling niet leeg is:
   3a. Neem een uitspraak u uit de gegevensverzameling (d.w.z. verwijder ze eruit)
   3b. Voor elke regel r in $LijstMetRegels(u)$, verminder $premisseteller(r)$. Als deze daardoor nul wordt, zet elke uitspraak uit de conclusie die nog niet behandeld is in de gegevensverzameling.

3. Deel het resultaat (of het gebrek eraan) mee aan de gebruiker.

Dikwijls zal een expertsysteem niet alleen het resultaat meedelen aan de gebruiker, maar ook de redenering waardoor het tot dit resultaat gekomen is. Het geeft dan een lijst van alle regels die gebruikt werden om het resultaat te bereiken. Uiteraard geeft het alleen de regels die het nodig had. Het is immers mogelijk dat er regels werden toegepast zonder dat dit nodig was om het resultaat te vinden. Het systeem geeft dus:

1. De gebruikte regel waarbij het resultaat tot de conclusie behoorde.
2. Als niet alle uitspraken uit de premisse tot de oorspronkelijke gegevensverzameling behoorden, welke gebruikte regels deze uitspraken in de conclusie hadden.
3. En zo verder, tot men geen tussenliggende uitspraken meer heeft.

Bij voorwaarts redeneren gebeurt het vaak dat men allerhande regels gebruikt en feiten toevoegt aan de gegevens, zonder dat dit helpt bij het oplossen van het probleem. Men probeert dit op te lossen door de techniek van terugredeneren te gebruiken. Deze methode is veel ingewikkelder, maar dikwijls ook veel efficiënter. Bij terugredeneren gaat men uit van het doel in plaats van de gegevens. We werken hier met drie uitsprakenverzamelingen: de gegevens, het doel, en de verzameling van tussendoelen. In het begin is de verzameling van tussendoelen leeg. Er is een verschil tussen een tussendoel en een gewoon doel. Als men één enkel doel kan aantonen, dan heeft men gevonden wat men zoekt. Maar als men een tussendoel aantoont dan is dit *niet* noodzakelijk voldoende: het kan zijn dat er in de premisse van een te gebruiken regel nog andere tussendoelen zitten die niet zijn aangetoond. Nemen we als voorbeeld een tussenregel

als A **en** B **en** C
dan D,

waarbij D een doel is en A gegeven. B en C komen in de tussendoelverzameling, maar als we B kunnen afleiden door een andere regel te gebruiken maar C niet, dan helpt dit ons niet vooruit.

Men heeft nu ook twee verzamelingen van regels: de regelbank, en de verzameling van tussenregels. Ook de tussenregelverzameling is in het begin leeg.

Men zoekt nu alle regels op wiens conclusie een uitspraak uit het doel bevat. Als alle uitspraken uit de premisse van zo een regel gegevens zijn, dan is een oplossing gevonden. Zo niet, dan voegt men de uitspraken uit de premisse toe aan de verzameling van tussendoelen, en de regel aan de verzameling van tussenregels. Men probeert nu ook de tussendoelen af te leiden uit de gegevens. Als men verschillende regels heeft die men kan toepassen moet men een keuze maken welke men eerst neemt. Zowel bij voorwaarts als bij terugredeneren kan het zijn dat men regels gebruikt die niet bijdragen aan de oplossing. Men probeert uiteraard steeds zo snel mogelijk tot een oplossing te komen, en dus de 'beste' regel te nemen. Er zijn verschillende strategieën ontwikkeld om dit zoveel mogelijk te beperken, zoals we in een vorig hoofdstuk gezien hebben. In het bijzonder is het ook hier mogelijk een relaxatieheuristiek te gebruiken. Het vereenvoudigde probleem bestaat eruit dat men regels mag toepassen waarvoor niet alle feiten uit de premisse reeds aangetoond zijn, maar dat men daarvoor een extra kost aanrekent gelijk aan het aantal nog niet bewezen uitspraken in de premisse. Vermits men om een uitspraak aan te tonen de toepassing van minstens één regel nodig heeft, leidt dit zeker tot een A^*-heuristiek (als men dubbeltellingen vermijdt: onbewezen uitspraken kunnen in de premissen van verschillende gebruikte regels zitten). In pseudocode ziet het volledige algoritme er als volgt uit:

A. Behandeling voor een regel waarbij alle uitspraken uit de premisse gegevens zijn:

A1. Voeg alle uitspraken uit de conclusie toe aan de gegevens.
A2. Voor alle tussenregels voor wie, op deze manier, alle uitspraken uit de premisse gegevens zijn geworden, behandel ze volgens A.
A3. Verwijder de regel uit de regelbank of uit de verzameling tussenregels.

B. Behandeling voor een regel waarbij niet alle uitspraken uit de premisse gegevens zijn:

B1. Voeg alle uitspraken uit de premisse die geen gegevens zijn toe aan de tussendoelverzameling.
B2. Verwijder de regel uit de regelbank en voeg hem toe aan de tussenregels.

C. Oplossingsalgoritme:

1. Initialiseer de gegevensverzameling en de doelverzameling. Initialiseer ook de verzamelingen van tussendoelen en tussenregels: deze zijn leeg.
2. Zolang er geen resultaat is gevonden, en er een regel is uit de regelbank wiens conclusie een doel of een tussendoel bevat, doe het volgende.

2a. Neem zo'n regel.
2b. Als alle uitspraken uit de premisse gegevens zijn, behandel volgens A, en anders volgens B.

Ook hier kan men een implementatie geven die in $O(g)$ werkt: dit heet achterwaarts aaneenschakelen (Eng: *backward chaining*).

3.2 DE CONSTRUCTIE VAN EEN EXPERTSYSTEEM

Bij de constructie van een expertsysteem heeft men twee soorten kennis nodig: kennis over het probleemgebied en kennis over expertsystemen. Experts in het toepassingsgebied weten hoe ze problemen oplossen, maar zijn meestal niet in staat deze kennis te vertalen in de vorm die een expertsysteem nodig heeft. Daarom wordt een expertsysteem meestal gemaakt door een team dat naast deze experts ook kennisingenieurs (Eng: *knowledge engineers*) omvat. Deze zijn ervoor verantwoordelijk te bepalen welke kennis er juist nodig is voor de constructie van het systeem, en om deze kennis in de juiste vorm te gieten. Uiteraard moeten deze twee met mekaar zinvol kunnen overleggen, zodat de experts vaak genoodzaakt zijn een basiskennis over expertsystemen te verwerven, terwijl de kennisingenieurs genoeg van het vakgebied moeten afweten om de experts te begrijpen.

We hebben er al op gewezen dat de kern van een expertsysteem uit twee delen bestaat: het afleidingsmechanisme en de kennisbank. Als men een expertsysteem voor een nieuw probleemgebied wil bouwen, dan is het mogelijk om een bestaand expertsysteem te nemen en de kennisbank te vervangen. Men maakt dan gebruik van het afleidingsmechanisme (samen met andere delen, zoals de user interface). Dit heeft geleid tot de ontwikkeling van *shells*. Een shell is een omgeving voor het ontwikkelen van expertsystemen. Dit houdt dan niet alleen het afleidingsmechanisme, en alle nodige programmatuur voor het gebruik van het systeem in, maar ook een ontwikkelingsomgeving voor het opstellen, testen en verifiëren van de kennisbank. Zowat de eerste shell was EMYCIN (Empty MYCIN). Zoals de naam aangeeft is dit het diagnoseprogramma MYCIN waaruit de medische kennis verwijderd is.

In deze context dient ook de programmeertaal Prolog vermeld. Prolog is uitermate geschikt voor het ontwikkelen van expertsystemen. Zo kunnen **als**-**dan**regels onmiddellijk worden ingevoerd, en is het mechanisme voor voorwaarts redeneren standaard aanwezig.

3.3 ONZEKERHEID

In realiteit zal het vaak voorkomen dat de relatie tussen premisse en conclusie van een regel niet geheel zeker is. Laten we een voorbeeld nemen van een expertsysteem dat de oorzaak zoekt van problemen in een computer. Dergelijk systeem kan een reparateur begeleiden die een defecte pc moet repareren. De reparateur zal aan het systeem melden welke de symptomen zijn, het systeem moet dan de actie bepalen die moet ondernomen worden om het probleem op te lossen. Het doel is in dit geval om de waarde te vinden van de variabele ACTIE. De regelbank bevat regels zoals

> **als** SYMPTOOM=computer doet niets bij opstarten
> **dan** PROBLEEM=voeding defect
>
> **als** PROBLEEM=voeding defect
> **dan** ACTIE=vervang voeding,

enzoverder. Maar niet bij alle problemen zal de link tussen symptoom en actie zo direct zijn. Zo kan een probleem verschillende oorzaken hebben. Nemen we het voorbeeld van een pc met meer dan een OS, die blijft hangen als de bootloader wordt opgestart. Dit kan verschillende oorzaken hebben: de harde schijf werkt niet (en dat kan zijn omdat de schijf defect is, maar ook omdat het kabeltje los zit), maar ook omdat de bootloader niet correct is geïnstalleerd. Dit zou dan een regel opleveren van de vorm

> **als** PROBLEEM=bootloader hangt
> **dan** OORZAAK=schijf defect **of** OORZAAK=bootloaderinstallatie **of** ...

Maar zoals we gezien hebben kunnen we bij een eenvoudig systeem geen **of** gebruiken in de conclusie. Bovendien zijn niet alle mogelijkheden hier equivalent: sommige zijn meer waarschijnlijk dan de andere. Zo zal een expert waarschijnlijk zeggen dat, als er een vreemd gebrom is als de computer opstaat, dit hoogstwaarschijnlijk een defecte ventilator is, en misschien een defecte harde schijf. Men lost dit op door aan elke uitspraak een numerieke waarde toe te kennen, in plaats van een logische waarde (waar of onwaar). Elke uitspraak uit de gegevensverzameling is niet gewoonweg waar, maar heeft nu zo'n numerieke factor, waarbij bijvoorbeeld de waarde 1 betekent 'zeker waar'. De conclusie van een regel bestaat nu uit een lijst van uitspraken die niet meer verbonden zijn met de **en**operator, maar waar bij elke uitspraak uit de lijst een numerieke waarde staat, die aangeeft hoe waar een uitspraak is als de premisse waarde 1 heeft. Maar we moeten nu een manier vinden om een numerieke waarde te bepalen voor elke uitspraak in de conclusie als de waarde bij de premisse verschilt van 1. Er zijn drie belangrijke systemen voor dergelijke numerieke waarden:

1. Bayesiaans redeneren,
2. theorie van zekerheidsfactoren,
3. vage verzamelingen.

De theorie van vage verzamelingen ligt buiten het bestek van deze cursus. Bayesiaans redeneren gaat uit van waarschijnlijkheidsrekening. De waarschijnlijkheid van een uitspraak ligt hier tussen 0 en 1, en geeft de kans aan dat de uitspraak waar is. Voor het gebruik van regels met een onzekere premisse gebruikt men nu waarschijnlijkheidsrekening. Nemen we het voorbeeld van weersvoorspelling. Voor een bepaald gebied gelden de volgende regels:

> **als** het regent op dag x
> **dan** is de kans dat het op dag $x + 1$ regent 0,8
>
> **als** het niet regent op dag x
> **dan** is de kans dat het op dag $x + 1$ regent 0,3

Als we weten dat het vandaag regent, dan is de kans dat het morgen regent 0,8. Maar wat is de kans dat het overmorgen regent? Er zijn twee mogelijkheden:

- Het regent morgen wel, en overmorgen ook: de kans hierop is 0,8×0,8=0,64.
- Het regent morgen niet, maar overmorgen wel: de kans hierop is 0,2×0,3=0,06.

De totale kans dat het overmorgen regent is dus 0,7. Statistisch redeneren wordt toegepast in expertsystemen, maar alleen in die gevallen waar er veel statistische informatie is. Er is immers een probleem. Als a en b twee uitspraken zijn, met waarschijnlijkheden $p(a) = 0{,}8$ en $p(b) = 0{,}2$, wat is dan de waarschijnlijkheid van de premisse a **en** b? Daar kunnen we niet veel over zeggen: $p(a$ **en** $b)$ kan elke waarde tussen 0 en 0,2 aannemen. Als bijvoorbeeld a =het regent morgen en b = het regent niet morgen, dan kunnen a en b onmogelijk beide samen waar zijn, en dan is $p(a$ **en** $b) = 0$. Maar het kan ook zijn dat a altijd waar is als b waar is, bijvoorbeeld als a =het regent morgen en b = het regent morgenochtend. In dat geval is $p(a$ **en** $b) = 0{,}2$. Er is nog een ander probleem: experts denken in termen van onzekerheid, maar kunnen niet goed inschatten wat de waarschijnlijkheid van iets is. Iemand die al vijf jaar niets anders doet dan pc's repareren, zal waarschijnlijk niet kunnen zeggen wat juist de kans is dat een harde schijf defect is, zelfs al kan hij efficiënt de fout in een computer zoeken.

De makers van MYCIN hebben een systeem ontwikkeld dat beide moeilijkheden omzeilt, maar toch goede resultaten oplevert: ze ontwikkelden de theorie van zekerheidsfactoren (Eng: *certainty factors*). De zekerheidsfactor, die door een expert in woorden wordt uitgedrukt, wordt vertaald naar een numerieke factor met waarde tussen -1 en +1, volgens de volgende tabel:

Term	zekerheidsfactor
zeker niet	-1,0
bijna zeker niet	-0,8
waarschijnlijk niet	-0,6
misschien niet	-0,4
onbekend	-0,2 tot +0,2
misschien	+0,4
waarschijnlijk	+0,6
bijna zeker	+0,8
zeker	+1,0

Als a een uitspraak is, dan noteren we de zekerheidsfactor van die uitspraak als zf(a). Uit de tabel volgt dat zf$(a) = -$ zf$(\neg a)$, waarbij $\neg a$ staat voor de negatie van a.

We veronderstellen nu dat elke regel slechts één uitspraak als conclusie heeft: we zullen in het vervolg dan ook spreken over 'conclusie' als we 'uitspraak uit de conclusie' bedoelen. Hoe berekenen we nu de zekerheid van een conclusie als de premisse van een regel zelf onzeker is? We nemen eerst aan dat de conclusie a maar in een enkele regel voorkomt, bijvoorbeeld

als b
dan $a \quad (x)$

Hier is x de zekerheidsfactor van de regel. Er geldt dan dat zf$(a) = zf(b).x$. Als de premisse meer dan een uitspraak bevat, dan nemen we als zekerheidsfactor voor de premisse het minimum van de zekerheidsfactoren van de uitspraken: zf$(b$ **en** c **en** $d) =$ min(zf(b), zf(c), zf(d)). Hebben we dus bijvoorbeeld de regel

als b **en** c **en** d

dan $a \quad (x)$

dan krijgen we $\text{zf}(a) = \min(\text{zf}(b), \text{zf}(c), \text{zf}(d)).x$. Het is echter ook mogelijk dat een uitspraak voorkomt als conclusie van twee verschillende regels. Nemen we bijvoorbeeld dat we twee regels hebben

als e
dan $a \quad (x)$

als f
dan $a \quad (y)$

Volgens de eerste regel moeten we aan a de zekerheidsfactor $z_1 = \text{zf}(e).x$ toekennen, volgens de tweede regel $z_2 = \text{zf}(f).y$. We gaan nu een combinatiefunctie $\text{zf}(z_1, z_2)$ nemen die al zeker aan de volgende voorwaarden voldoet:

(1) $\text{zf}(z_1, z_2) = \text{zf}(z_2, z_1)$. Symmetrie.
(2) $\text{zf}(z_1, 0) = z_1$. Een nulwaarde draagt niets bij aan de zekerheidsfactor.
(3) Als $z_3 > z_2$ dan is $\text{zf}(z_1, z_3) \geq \text{zf}(z_1, z_2)$, met gelijkheid als en slechts als $z_1 = 1$ (en $z_2 > -1$) of $z_1 = -1$.
(4) $\text{zf}(1, z_2) = 1$ voor $z_2 > -1$: een zekere uitspraak kan niet veranderd worden.
(5) $\text{zf}(-1, 1)$ is onbepaald, want duidt een contradictie aan.

Nemen we eerst aan dat beide waarden positief zijn. We zouden nu de grootste van de twee waarden kunnen nemen, maar dat is niet echt een goed idee en overtreedt regel 3. Immers, beide waarden zijn positief, wat wil zeggen dat uitspraak a bevestigd wordt door de twee regels. We willen dus een zekerheidsfactor die groter is dan de grootste van de twee waarden z_1 en z_2 (tenzij we al 1 hebben). Men gebruikt de formule

$$\text{zf}(a) = z_1 + z_2 - z_1.z_2 \qquad (z_1 \geq 0, z_2 \geq 0).$$

Merk op dat we dit ook kunnen schrijven als $z_1 + z_2(1 - z_1)$, of als $z_2 + z_1(1 - z_2)$. Nu is $(1 - z_1)$ positief (want $z_1 \leq 1$) en dus is daarmee $\text{zf}(a) > z_1$, behalve als $z_1 = 1$. Bovendien is $\text{zf}(a)$ zeker niet groter dan 1.

Als z_1 en z_2 beide negatief zijn, dan zijn $-z_1$ en $-z_2$ kandidaat-zekerheidsfactoren voor $\neg a$, en ze zijn beide positief. We krijgen dus $\text{zf}(\neg a) = -z_1 - z_2 - z_1.z_2$, en dus

$$\text{zf}(a) = z_1 + z_2 + z_1.z_2 \qquad (z_1 \leq 0, z_2 \leq 0).$$

Als tenslotte z_1 en z_2 tegengesteld teken hebben, dan spreken de twee regels elkaar tegen. Als $|z_1|$ en $|z_2|$ ongeveer even groot zijn, dan wegen de twee regels tegen elkaar op. Als ze heel erg verschillen, dan weegt een van de twee regels meer door. Nu willen we dat de combinatie van 'zeker wel' en 'misschien niet' toch nog 'zeker wel' oplevert. Dus nemen we niet gewoon de som van de twee waarden, maar wel

$$\text{zf}(a) = \frac{z_1 + z_2}{1 - \min(|z_1|, |z_2|)} \qquad (z_1.z_2 \leq 0).$$

Het is gemakkelijk te controleren dat als een van de twee waarden +1 of -1 is, het resultaat ook +1 dan wel -1 is. Als echter de ene waarde -1 is en de andere +1, dan delen we door nul. Dit duidt op een fout in de kennisbank: volgens de ene regel is a zeker waar, volgens de andere is a zeker onwaar.

Bij de implementatie van zekerheidsfactoren moeten we aanpassingen doen aan het algoritme van de afleiding. Bekijken we bijvoorbeeld het voorwaarts redeneren bij een eenvoudig expertsysteem (zie het algoritme op pagina 38). Als we een regel behandelden, verwijderden we deze uit de regelbank. Dit was toegelaten, omdat de premisse waar was, en

altijd waar zou blijven. In het nieuwe systeem is de situatie anders. Als de zekerheidsfactor van een uitspraak b uit de gegevensverzameling verschilt van +1 en van -1, dan kan het zijn dat hij nog verandert door toepassing van een andere regel. Als dit gebeurt, moeten alle regels waarbij b in de premisse zit, opnieuw geëvalueerd worden. Nemen we als voorbeeld het geval waarbij de regel

als b
dan a (0,8)

behandeld wordt op een ogenblik dat zf(b) = 0,1. We krijgen dan zf(a) = 0,08. Even later vinden we een regel die de waarde van zf(b) op 0,9 brengt. Daarmee is a veel waarschijnlijker geworden: we moeten de nieuwe waarde zf(a) = 0,72 gebruiken. Hierbij worden hoge eisen gesteld aan de structuur van de regelbank, omdat er geen lussen meer mogen voorkomen. In een eenvoudige regelbank, zonder onzekerheidsfactoren, mag best een paar regels voorkomen van de vorm

als b
dan a

als a
dan b

Dit betekent gewoon dat a waar is als en slechts als b waar is. Maar in een systeem met onzekerheden leidt het paar

als b
dan a (0,8)

als a
dan b (0,8)

tot problemen. Immers, als een andere regel zf(b) positief maakt, zal de eerste regel zf(a) positief maken. Door de tweede regel vergroot dan zf(b), waarna de eerste zf(a) vergroot, en zo verder. Uiteindelijk bekomen we zf(a) = zf(b) = 1, wat natuurlijk niet de bedoeling kan zijn. In dit voorbeeld is de lus eenvoudig te vinden, maar in een regelbank met honderden of zelfs duizenden regels is dit niet langer het geval.

Nog een opmerking die we moeten maken is de volgende. In een eenvoudig systeem zoals we tot hier bekeken hebben kan een variabele maar één bepaalde waarde aannemen. Werken we met zekerheidsfactoren dan is het best mogelijk dat twee tegenstrijdige uitspraken beide een positieve zekerheidsfactor hebben, bijvoorbeeld

- 'VAR=a' heeft zekerheidsfactor 0,4'
- 'VAR=b' heeft zekerheidsfactor 0,5',

waarin VAR een of andere variabele is. In het voorbeeld van ons diagnoseprogramma voor pc's kan dit de variabele ACTIE zijn, terwijl het juist de waarde van ACTIE moeten zoeken. Het systeem zal dan de mogelijke acties als uitvoer geven, en hierbij de zekerheidsfactoren opgeven, zodat de gebruiker kan beslissen welke actie hij onderneemt, en daarbij een idee heeft wat de kansen op succes zijn. Voor de structuur van het programma wil dit zeggen dat, als er een variabele is die verschillende waarden kan aannemen en waarbij onzekerheid kan optreden, we voor elke mogelijke waarde moeten bijhouden wat de zekerheidsfactor is voor die waarde. Dit betekent dat we bij de constructie van de regelbank we twee soorten variabelen hebben:

- Variabelen met onzekerheid. Zo'n variabele kan slechts een beperkt aantal waarden aannemen, en de mogelijke waarden moeten expliciet beschreven zijn

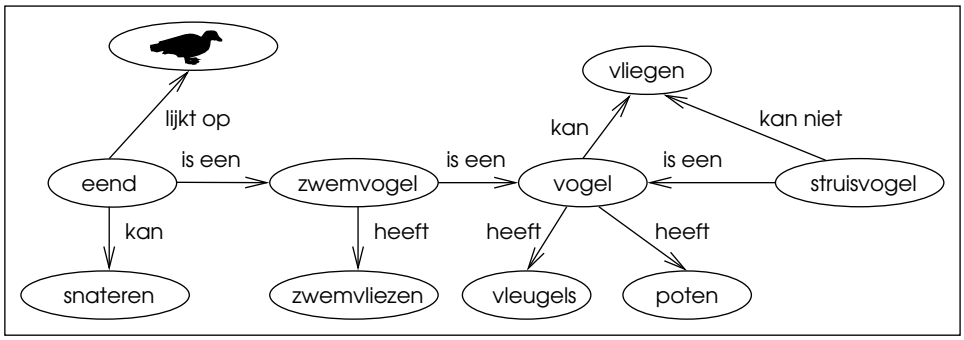
- Variabelen waarbij geen onzekerheid kan bestaan: ofwel zijn ze ingevuld, ofwel zijn ze niet ingevuld. Alleen dergelijke waarden kunnen een groot domein hebben.

In ons voorbeeld moeten dus alle mogelijke waarden voor de variabele ACTIE op voorhand gekend zijn. Bij zekere variabelen kan het voldoende zijn om –bijvoorbeeld– te specifiëren dat ze een willekeurige gehele waarde hebben.

3.4 FRAMES EN REGELSCHEMA'S

In een eenvoudig expertsysteem bestaat de kennisbank uit een regelbank en feiten. Voor grotere systemen is er noodzaak om de kennis in te delen. Hiervoor wordt gebruik gemaakt van *frames*. Frames zijn ontwikkeld in dezelfde periode als objecten voor objectgericht programmeren, en vertonen er ook zeer veel gelijkenis mee. Soms wordt het woord frame zelfs als synoniem beschouwd voor het woord klasse. Beide zijn ontwikkeld uit de theorie van het *semantisch net*, en het is passend om dit begrip even toe te lichten.

De theorie van het semantisch net is in de zestiger jaren van de vorige eeuw ontwikkeld als model van de opslag van de betekenis van woorden in het menselijk brein. Het semantisch netwerk bestaat uit een geëtiketteerde gerichte graaf[2] (dat wil zeggen dat de verbindingen zelf namen hebben). Elke knoop is een *begrip*, en elk begrip is verbonden met andere begrippen. Een klein voorbeeld zal dit duidelijk maken. We zijn uitgegaan van het begrip eend, en hebben een klein gedeelte van het semantisch net getekend in Figuur 3.2. Zoals het menselijk brein, is het semantisch net zeer weinig georganiseerd: het is



Figuur 3.2. Semantisch net.

gewoon een verzameling begrippen. Sommige begrippen hebben veel, andere veel minder verbindingen. Het net kan gebruikt worden om allerhande informatie op te zoeken. Nemen we enkele voorbeeldjes, waarbij men gebruik maakt van het bovenstaande net om vragen over eenden op te lossen:

- *Kan een eend snateren?* Het antwoord wordt gevonden door vanuit 'eend' te vertrekken en via de 'kan'-relatie naar 'snateren' te gaan. Het antwoord is 'ja'.
- *Heeft een eend zwemvliezen?* Via de 'is een'-relatie vindt men dat een eend een zwemvogel is, en via de 'heeft'-relatie ontdekt men dat een zwemvogel zwemvliezen heeft. Het antwoord is 'ja'.

[2] Strikt genomen gaat het om een *multigraaf*, omdat er meer dan één tak van een knoop naar een andere kan gaan.

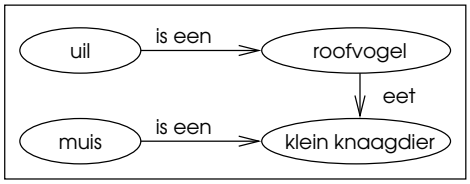
- *Heeft een eend vleugels?* Via de twee 'is een'-relaties vindt men dat de eend een vogel is, via de 'heeft'-relatie ontdekt men dat een vogel vleugels heeft. Het antwoord is 'ja'.

- *Eet een eend sla?* Na het netwerk te hebben afgezocht vindt men geen relatie tussen sla en eenden. Het antwoord is 'geen idee'.

In welke richting zoekt men het antwoord op een vraag? Er zijn twee regels:

(1) Volg de takken waarvan de naam in de vraag staat (maar alleen in de goede richting: het is niet de bedoeling dat we op de vraag "wat eten kikkers?" het antwoord "ooievaars" krijgen).
(2) Volg de takken met "is een" als naam.

Dat de structuur van het semantisch net er ongeveer uitziet zoals hierboven geschetst, is experimenteel bevestigd. Men ziet dat bij de vier voorbeeldvragen hierboven er steeds meer relaties moeten worden onderzocht voor het antwoord gevonden wordt. En inderdaad blijkt in de praktijk, als men zulke vragen stelt, dat proefpersonen meer tijd nodig hebben om te antwoorden als er meer relaties onderzocht moeten worden. Een tweede opmerkelijk feit is het belang van de 'is een'-relatie. Inderdaad gaan de meeste onderzoekingen via dergelijke relaties. Als we een eigenschap van een begrip willen kennen, en het antwoord op de vraag niet rechtstreeks vinden, dan gaan we via de 'is een'-relaties naar meer algemene begrippen, en zien we of we daar de informatie over de eigenschap vinden. Merk op dat we dit alleen doen als we geen informatie vinden zonder de 'is een'-relatie. Het antwoord op de vraag *kan een struisvogel vliegen?* is volgens dit semantisch net 'neen'. Een struisvogel is een vogel, en een vogel kan vliegen, maar dit onrechtstreeks antwoord wordt niet gezocht (en dus ook niet gevonden), omdat het juiste antwoord, via de 'kan niet'-relatie, eerder gevonden is. Merk op dat de 'is een'-relatie in twee richtingen kan gevolgd worden. Als we met het onderstaande semantisch net de vraag willen oplossen *eten uilen muizen?* kunnen we van 'uil' naar 'muis' gaan door de 'is een'-relatie van 'klein knaagdier' naar 'muis' in de omgekeerde richting te volgen.



Figuur 3.3. Uilen en muizen.

Omdat de 'is een'-relatie zo belangrijk is en bovendien een speciale status heeft omdat ze wordt afgezocht ook als ze niet in de vraag voorkomt, heeft men ze centraal gesteld bij meer geordende voorstellingen van kennis, zoals objectgericht programmeren en het werken met frames: ze is de basis voor *overerving* in beide formalismen. Het feit (zoals de niet-vliegende struisvogel) dat een eigenschap van een begrip anders kan zijn als die van een meer algemeen begrip, leidde bij objectgericht programmeren tot het overschrijven van methodes.

Een tweede eigenschap van het semantisch net is het bestaan van speciale begrippen die gekenmerkt worden door twee eigenschappen:

- In dergelijke begrippen komen veel pijlen aan, maar vertrekken er weinig.

- Bijna alle aankomende pijlen hebben hetzelfde etiket, en dit etiket is niet 'is een'.

Een voorbeeld hiervan is het begrip 'blauw'. Hier komen veel pijlen aan met het etiket 'is': er is een blauwe hemel, er zijn blauwe hemden, blauwe maandagen, enzovoorts. Dergelijke begrippen hebben bij objectgericht programmeren geleid tot de notie van *attributen*, die bij een object worden genoteerd. In plaats van een groot aantal begrippen die een 'is'-relatie

hebben naar blauw, zoals bij het semantisch net, zijn er dan objecten met een kleurattribuut[3].

Tot slot van onze bespreking van het semantisch net moeten we opmerken dat dit net enkel informatie bevat, en geen regels voor de verwerking ervan. Als we de vergelijking met objecten maken, dan heeft het semantisch net wel het equivalent van attributen en relaties, maar niet van methodes.

Zowel frames als klassen worden gebruikt om structuur te brengen in gegevens, en zijn voornamelijk interessant als er gelijkaardige groepen van gegevens zijn. Er zijn echter een aantal verschillen, die we verder nader bekijken. De theorie van objecten in verband met objectgeoriënteerd programmeren is tamelijk goed gestandaardiseerd; dit is niet het geval bij frames. Er zijn heel wat varianten en de beschrijving die we hier geven is maar een algemeen kader: de naamgeving hangt af van toepassing tot toepassing, en er zijn ook structurele verschillen. De beschrijving die we hier geven is dan ook een soort van gemiddelde.

Zoals bij objecten en klassen, is er bij frames een hiërarchisch systeem dat gebruik maakt van overerving. We gaan hier uit van een systeem waarin geen meervoudige overerving voorkomt, hoewel er systemen zijn die daarvan wel gebruik maken. Een systeem bestaat dan uit een of meerdere bomen bestaande uit frames. In tegenstelling tot wat bij objectgericht programmeren gebeurt, wordt er hier geen onderscheid gemaakt tussen klassen en objecten: de frames die de bladeren van de boom of bomen vormen, komen overeen met objecten, de andere met klassen.

Laten we het voorbeeld van het expertsysteem voor reparatie van pc's nogmaals bekijken. We zouden een boom kunnen hebben van computermodellen. Bovenaan is er een frame, pc. De kinderen van dit frame komen overeen met verschillende fabrikanten van pc's. Elk merk heeft verschillende modellen, en voor elk model zouden we weer een frame kunnen voorzien. Maar ook de acties zouden we op dergelijke manier kunnen beschrijven. In ons eenvoudig systeem was er een variabele ACTIE, die verschillende waarden kon aannemen. In een systeem met frames kunnen we dan een frame ACTIE hebben. Elke mogelijke actie zou dan een frame krijgen dat een kind is van het ACTIE-frame, en in dit frame zouden we dan gegevens over de actie kunnen opnemen, zoals de kost en de nodige tijd.

Frames hebben *slots*. Een slot kan een attribuut zijn, maar ook verwijzen naar een andere frames, zodat relaties kunnen gelegd worden tussen verschillende frames. Dit komt niet helemaal overeen met de associaties tussen objecten in objectgericht programmeren: immers een frame kan kinderen hebben in de hiërarchie, en dus overeenkomen met een klasse, en tegelijkertijd attribuutslots hebben (en dus heeft het een toestand en is het een object), en kan daarbij ook nog relatieslots hebben. Een slot (in termen van objecten) kan dus verwijzen naar een object, maar ook naar een klasse.

In de algemene zin zijn er geen operaties in frames, hoewel er in een aantal systemen wel code aan een frame gehecht wordt, vooral in de vorm van zogenaamde *facetten*. Deze zullen we later bespreken, maar eerst gaan bekijken hoe gegevens behandeld worden in een systeem met frames.

Een frame heeft dus slots, en in zo'n slot kan een waarde ingevuld zijn. Een slot kan ook niet ingevuld zijn, maar het is handig om een waarde *niet_ingevuld* in te voeren, zodat een slot per definitie een waarde bevat. Zoals vermeld vormen de frames in een systeem een of meerdere bomen, waarin elk frame, behalve de wortel van een boom, een ouder heeft. Frames erven slots volgens de volgende regels:

1. Als de ouder van een frame een slot heeft, dan heeft automatisch het frame zelf ook

[3] Opgemerkt moet worden dat er naast het attribuut ook het object blauw bestaat, dat wel relaties kan aangaan: zo maakt de kleur blauw deel uit van de regenboog, en zo verder.

een slot met dezelfde naam.

2. De waarde van de slot wordt overgeërfd: als de slot van de ouder een bepaalde waarde heeft, en de slot bij het kind heeft de waarde *niet_ingevuld*, dan wordt bij raadpleging de waarde van de slot van de ouder genomen. De slot van het kind kan echter overschreven worden, zodat de waarde bij ouder en kind verschillen.

Een frame kan ook bijkomende slots hebben, die niet overeenkomen met slots bij de ouder.

Bij een eenvoudig systeem bekeken we uitspraken van de vorm ⟨**variabelennaam**⟩ = ⟨**waarde**⟩. Nu we werken met frames hebben we niet genoeg aan deze vorm. Verschillende frames kunnen slots met dezelfde naam hebben, en vaak gaat het over frames die we niet kennen, maar zoeken. Dit gebeurt als we vragen stellen zoals:

1. Is er een frame waarbij slot ⟨**variabelennaam**⟩ een bepaalde gegeven waarde heeft?
2. Welk frame past het best bij een bepaald profiel?

Nemen we als voorbeeld een beeldanalyseprogramma, dat probeert vormen te herkennen op een beeld. Als het een blauwe pixel vindt, dan kan het de vraag stellen voor welke frames KLEUR=blauw geldt (vraag 1). Uiteindelijk zal het proberen een aantal kenmerken te herkennen, en aan de hand daarvan proberen te bepalen welk voorwerp er op het beeld te zien is (vraag 2).

Bovendien moeten we met relaties kunnen werken. Een voorbeeld: persoon A is een grootouder van persoon B als er een persoon is die tegelijk kind is van A en ouder van B. Deze relatie moeten we in een regel kunnen stoppen als we een expertsysteem voor familierelaties willen maken. Hiervoor maken we gebruik van *predikatenlogica*. In wat volgt werken we meteen met frames. We gebruiken de notatie die bij frames hoort, en onze aanpak zal er dus anders uitzien dan die van een klassieke uiteenzetting over logica. Nemen we aan dat we drie frames hebben, Anna, Bart en Chris. Bekijken we de volgende drie uitspraken:

1. Anna is een man.
2. Bart is een man.
3. Chris is een man.

Deze uitspraken kunnen waar of onwaar zijn. Maar wat opvalt is dat ze een gemeenschappelijke structuur hebben, waarin telkens een andere naam is ingevuld. Er is dus een schema, en dergelijk schema zullen we een *predikaat* noemen. In zo'n schema moeten we kunnen aanduiden waar we de naam van een frame kunnen invullen. Dit doen we met een variabele: deze noteren we als een vraagteken gevolgd door een letter. Het predikaat dat hoort bij de vorige uitspraken is dan

$?x$ is een man.

Dit is een voorbeeld van een predikaat met één vrije variabele (we leggen later uit wat een vrije variabele is; er zijn ook gebonden variabelen), maar het is mogelijk om predikaten met meerdere variabelen in te voeren, zoals

$?x$ is een broer van $?y$.

Het aantal vrije variabelen in een predikaat is de *graad* van het predikaat. Een predikaat met nul vrije variabelen is een uitspraak. Een uitspraak die we verkrijgen door in een predikaat met graad groter dan nul alle variabelen te vervangen door de naam van een frame is een *instantiatie* van het predikaat. Omdat een predikaat een schema is voor een uitspraak, kunnen we hier ook allerhande logische operatoren op predikaten toepassen, zoals **en** of $\neg$, bijvoorbeeld

$\neg ?x$ is een broer van $?y$ **en** $?x$ is een man.

Voor de waarde van een slot gebruiken we de puntnotatie zoals die in programmeertalen zoals C++ en Java gebruikelijk is. Het predikaat '$?x$ is een man' kunnen we dan ook schrijven als '$?x$.GESLACHT=man'.

Maar dit is nog niet voldoende. Wat moeten we doen met een uitspraak als 'er is een frame dat een man is'? Hiervoor gebruiken we zogenaamde kwantoren. Er zijn twee kwantoren: $\exists$ (betekent: er is een ...) en $\forall$ (betekent: voor alle). Een voorbeeld:

$\exists ?x$:$?x$ is een man.

Een kwantor wordt gevolgd door een variabele, daarna een dubbelpunt en daarna een predikaat waarin dezelfde variabele als vrije variabele voorkomt. Het resultaat is opnieuw een predikaat, waarvan de graad één lager is dan de graad van het oorspronkelijke predikaat. De variabele na de kwantor is *gebonden* door het predikaat, en is dus geen vrije variabele meer. Nog een voorbeeld:

$?x$ is een grootouder van $?y$.
$\exists ?z$:$?x$ is een ouder van $?z$ **en** $?z$ is een ouder van $?y$.

Dit zijn allebei predikaten van graad 2. In het tweede predikaat zijn er drie variabelen, $?x$, $?y$ en $?z$. De eerste twee zijn vrij, de derde is gebonden.

Keren we nu terug naar regelbanken. Een predikaat is een uitspraakschema, en naast regels (met uitspraken in premisse en conclusie) krijgen we ook regelschema's van de vorm

als ⟨**predikaat1**⟩
dan ⟨**predikaat2**⟩

waarbij ⟨**predikaat1**⟩ en ⟨**predikaat2**⟩ predikaten zijn, meestal van dezelfde graad, en *met dezelfde vrije variabelen*. Als voorbeeld kunnen we de definitie van een grootouder geven:

als $\exists ?z$:$?x$ is een ouder van $?z$ **en** $?z$ is een ouder van $?y$
dan $?x$ is een grootouder van $?y$.

en een definitie van neef:

als $\exists ?z$:($?z$ is grootouder van $?x$ **en** $?z$ is ouder van $?y$) **en** $?x$.GESLACHT=man
dan $?x$ is neef van $?y$.

Merk op dat dit maar halve definities zijn: we hebben wel de **als**...**dan**-kant, maar niet de omkering. De regel zegt wel dat we een grootouderrelatie hebben als de premisse voldaan is, maar niet dat de premisse voldaan is als we een grootouder hebben. Dit wordt duidelijk met het volgende voorbeeld:

als $?x$ is koeienmelk **en** $?x$ is gestremd
dan $?x$ is kaas

geeft geen definitie van kaas: er is ook kaas gemaakt van geitenmelk, schapenmelk enzovoorts. De overerving in framesystemen is uiteraard belangrijk. Vandaar dat een aantal predikaten in een aantal systemen vast gespecificeerd zijn:

- isdirecteen. '$?x$ isdirecteen $?y$' is waar als $?x$ direct onder $?y$ komt in de overervingshiërarchie. (Dit komt overeen met de 'is een'-relatie uit het semantisch netwerk).

- iseen. Dit wordt gedefinieerd met de regels

  als $?x$ isdirecteen $?y$
  dan $?x$ iseen $?y$

als $\exists ?z : (?x$ iseen $?z$ **en** $?z$ isdirecteen $?y)$
dan $?x$ iseen $?y$.

Merk op dat deze definitie recursief is, zodat '$?x$ iseen $?y$' waar is als $?x$ onder $?y$ komt in de overervingshiërarchie met een willekeurig aantal tussenstappen.

- iseenobject. Gedefinieerd met

  als $\neg\exists ?y$:$?y$ iseen $?x$
  dan $?x$ iseenobject.

Een predikaat is een schema om uitspraken te genereren. Regels die predikaten bevatten met graad groter dan nul zijn dus ook eigenlijk schema's om regels te genereren. Hebben we zo'n regelschema, dan krijgen we een instantiatie van dit schema door alle vrije variabelen te vervangen door de naam van een frame. Uiteraard moet zo een variabele zowel in de premisse als in de conclusie door dezelfde naam vervangen worden.

Als we nu terugkijken naar het algoritme dat gebruikt wordt bij voorwaarts redeneren in een eenvoudig regelsysteem dan zien we dat er een aantal aanpassingen nodig zijn. We nemen aan dat de verzameling frames tijdens de redenering constant is, d.w.z. dat er tijdens het redeneren geen frames gecreëerd of vernietigd worden. We kunnen dan de gegevensverzameling beschouwen als het geheel van alle frames. Hierin kunnen sommige slots de waarde *niet_ingevuld* hebben.

Het algoritme op pagina 38 moet nu als volgt worden aangepast:

- In plaats van alleen maar een toepasbare regel te zoeken, kijken we nu ook naar de regelschema's. Als er een instantiatie van het regelschema bestaat waardoor de premisse alleen gegevens bevat, kunnen we die gebruiken.
- Als we een gewone regel (die alleen predikaten van graad nul gebruikt) gebruiken, dan verwijderen we deze uit de regelbank. Als we echter een regelschema hebben, dan verwijderen we alleen de gebruikte instantiatie. In de praktijk gebeurt dit door bij het schema te noteren welke variabelencombinatie gebruikt is.

Nemen we bij het voorbeeld van familierelaties aan dat we reeds de feiten

Karel is ouder van Joost.
Karel is ouder van Geert.

als gegevens hebben en dat er het regelschema

als $\exists ?z : (?z$ is een ouder van $?x$ **en** $?z$ is een ouder van $?y)$ **en**$?x$.GESLACHT=man
dan $?x$ is een broer van $?y$

gebruikt wordt, met $?x$ vervangen door Geert, en $?y$ vervangen Joost, zodat we de conclusie 'Geert is een broer van Joost' krijgen. Bij de regel noteren we dat we het paar (Geert, Joost) al gebruikt hebben. Merk op dat de volgorde belangrijk is: we kunnen het schema nogmaals gebruiken om de conclusie 'Joost is een broer van Geert' te krijgen.

Het doel bij frames wordt dikwijls ook uitgedrukt als een predikaat. Voor ons diagnosesysteem kunnen we dit als volgt doen: de kennisbank bestaat onder andere uit een hiërarchie van types van pc's. Noemen we nu de pc die we onderzoeken depc, dan voegen we één frame toe, dat overerft van het type pc dat we onderzoeken. Dit frame bevat onder andere een slot SYMPTOOM, dat we invullen (in een reëel systeem zijn meerdere symptomen mogelijk, zodat we meerdere slots daarvoor voorzien), en een slot TEONDERNEMENACTIE. Wat we zoeken kunnen we uitdrukken als een predikaat:

$?x$ iseen ACTIE **en** depc.TEONDERNEMENACTIE=$?x$.

Het doel is een uitsprakenverzameling, namelijk de verzameling van instantiaties van dit predikaat. Het resultaat van het expertsysteem is dus een uitspraak van bovenstaande vorm.

Het is zeer goed mogelijk om te werken met zekerheidsfactoren als we gebruik maken van frames. We hebben al opgemerkt dat slots zowel attributen als verwijzingen naar andere frames kunnen zijn. Voor een attribuut geldt wat we reeds gezegd hadden bij gewone verzamelingen: als een attribuut onderworpen is aan zekerheidsfactoren, mag er slechts een beperkt aantal waarden zijn. Als een slot een verwijzing bevat, is de waardenverzameling natuurlijk beperkt (er zijn meestal voorwaarden die maar een beperkt aantal frames toestaan), en mag hij aan zekerheidsfactoren gekoppeld worden.

Een belangrijke toepassing van framesystemen is het verrichten van intelligente queries in een databank. Nemen we als voorbeeld een website waarop boeken te koop worden aangeboden. Een gebruiker doet een query op de website waarbij hij een aantal parameters opgeeft, zoals onderwerp, taal, prijsklasse en zo verder. Het doel wordt hier uitgedrukt door het predikaat

$?x$ iseen BOEK **en** $?x$ is interessant voor de gebruiker.

Eventueel kan het expertsysteem gebruik maken van voorafgaande kennis van de gebruiker, zoals voorgaande aankopen. Het expertsysteem kan nu voor elk boek, gebaseerd op de gegevens die het heeft, een zekerheidsfactor berekenen voor de uitspraak 'dit boek is interessant voor de gebruiker', en de boeken tonen, gesorteerd op deze zekerheidsfactor (grootste zekerheidsfactor eerst). Op deze manier krijgt de gebruiker een overzicht dat zoveel mogelijk is aangepast aan zijn wensen.

Sommige systemen kunnen *impliciet* de vorm hebben van een regelbank met onzekerheidsfactoren die zeer weinig regelschema's bevat. Nemen we als voorbeeld een aanbevelingssysteem met de regels

als $\exists ?p \exists ?z$:$?p$ vindt $?z$ leuk en $?q$ vind $?z$ leuk en $?p$ vindt $?y$ leuk
dan $?q$ vindt $?y$ leuk (0.00000001)

als $\exists ?p \exists ?z$: $\big(?p$ vindt $?z$ leuk$\big) \Leftrightarrow \big(?q$ vind $?z$ niet leuk$\big)$ en $?p$ vindt $?y$ leuk
dan $?q$ vindt $?y$ leuk (-0.00000001)

Als het aantal items klein genoeg is wordt hier het aantal gemeenschappelijke interesses gewoon opgeteld om aanbevelingen te krijgen.

Facetten

Het framesysteem zoals we dit tot nu toe besproken hebben is, juist zoals het semantisch net, enkel een opslagplaats van gegevens. Reeds snel ontstond de behoefte om ook regels bij frames onder te brengen, zodat ook de regelbank meer structuur gaat vertonen. In sommige systemen heeft men een structuur geconstrueerd die zeer dicht aanleunt bij objecten. Maar er is ook een structuur die specifiek is voor expertsystemen. Men werkt dan met *facetten*. In tegenstelling tot OGP, waar operaties bij klassen worden ingedeeld, horen facetten bij slots. Dit is logisch: uitspraken gaan over de waarde die een slot aanneemt, en het doel is meestal het bepalen van de waarde van een slot. Facetten bepalen dan hoe deze waarden moeten toegekend worden. Een slot kan de volgende facetten hebben:

- Een waardefacet. Dit geeft de waarde die op een expliciete manier is gegeven, bijvoorbeeld in de gegevensverzameling.
- Een defaultfacet. Dit bevat de waarde die wordt toegekend tenzij expliciet een andere waarde is ingevuld.

- Een toekenfacet, vaak WHEN NEEDED genoemd. Indien de slot geen ingevulde waarde heeft, bepaalt dit facet hoe de waarde kan afgeleid worden uit andere waarden. Dit facet is typisch voor terugredeneren.
- Een veranderdfacet, vaak WHEN CHANGED genoemd. Indien de slot wordt ingevuld, of de waarde verandert, bepaalt dit facet welke slots (in hetzelfde frame of in een ander frame) ook veranderd moet worden. Dit facet is typisch voor voorwaarts redeneren.
- Een vraagfacet (prompt facet). Soms is het nodig om aan de gebruiker extra informatie te vragen, omdat het probleem met de gegeven informatie niet kan opgelost worden. Om een voorbeeld te geven: in ons diagnoseprogramma voor pc's kan het systeem besluiten dat er verschillende mogelijke oorzaken zijn, en een actie voorstellen om tussen deze oorzaken een keuze te kunnen maken. Het vraagfacet bepaalt dan in welke omstandigheden het verantwoord is om zo'n vraag te stellen. Het heeft immers geen zin om vragen te stellen die niet bijdragen aan de oplossing van het probleem.

Een slot die niet is ingevuld kan dus een waarde krijgen op verschillende manieren: ofwel door een of ander facet (defaultfacet, toekenfacet, vraagfacet), ofwel door overerving vanuit een frame hoger in de hiërarchie. Het is dus nodig om een volgorde op te stellen om de waarde van zo'n slot in te vullen.

HOOFDSTUK 4
DE REGEL VAN BAYES EN MARKOVKETENS

In dit hoofdstuk maken we kennis met een aantal methodes van AI die gebruik maken van statistiek, en meer in het bijzonder de wet van Bayes. In het hoofdstuk over expert-systemen hebben we gezien hoe we kunnen omgaan met subjectieve onzekerheid, zoals bijvoorbeeld aanwezig is in de kennis van een expert. We kunnen statistische methodes alleen gaan gebruiken als er voldoende statistische informatie is. De voorbeeldtoepassing is hier spraakherkenning. Hierbij maken we ook kennis met Markovketens, die gebruikt worden om tijdsafhankelijke fenomenen te modelleren.

4.1 PROBABILITEIT

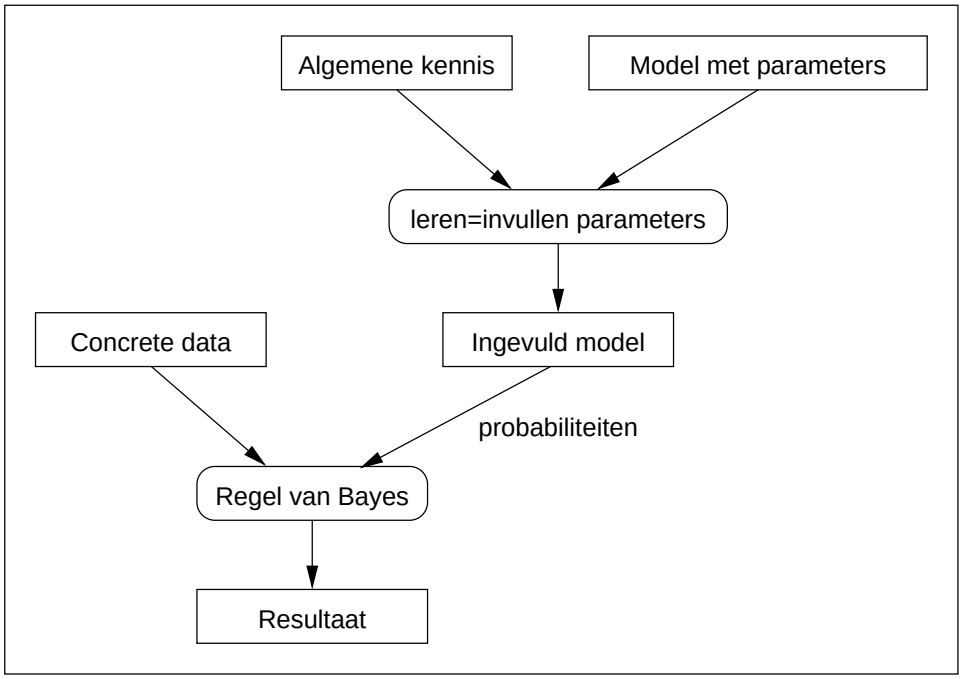
Er bestaan nogal wat opvattingen over probabiliteit. We gebruiken deze die voor onze doelstelling het eenvoudigste is. Aan de ene kant hebben we een aantal uitspraken, en aan de andere kant hebben we een universum van mogelijke toestanden. In elke toestand is er een aantal uitspraken die waar zijn, en een aantal die onwaar zijn. We veronderstellen niet dat bij elke mogelijke combinatie van uitspraken er een toestand is waar juist deze uitspraken waar zijn en andere niet, en we veronderstellen ook niet dat twee toestanden verschillen in hun verzamelingen ware uitspraken. Wel veronderstellen we dat het universum slechts een eindig aantal toestanden bevat. Als we het hebben over een medisch diagnoseprogramma, dan zou bijvoorbeeld het universum in een eerste benadering kunnen bestaan uit alle reeds onderzochte patiënten, waarbij de uitspraken gaan over persoonlijke gegevens (leeftijd, beroep, roker/niet-roker, ...), symptomen en ziektes. Is nu a een uitspraak. Dan is de *probabiliteit* van a, genoteerd als $p(a)$ en vaak ook de *kans* van a genoemd, het quotiënt van het aantal toestanden waarin a waar is en van het aantal toestanden in het universum.

Deze eerste benadering is niet bruikbaar. Stel dat we een 83-jarige patiënt hebben met hoofdpijn en dat er in ons universum geen 83-jarige patiënt met hoofdpijn zit. Dan, volgens onze definitie van waarschijnlijkheid, is het probleem van deze patiënt dat hij niet kan bestaan. Een volgende patiënt is een man van 43, niet-roker en metselaar. In ons universum zit één metselaar van 43 die niet rookte, en die had last van psoriasis. Onze diagnose is dan dat onze nieuwe patiënt psoriasis heeft. En zo voorts.

Het probleem hier is fundamenteel hetzelfde als wat we hadden bij lerende classificatieprogramma's. Daar hadden we een *zinvolle veralgemening* nodig, en deze werd geleverd door het lerend programma (zie paragraaf 1.5). Hier hebben we een feitenverzameling (vroegere ziektegevallen of, voor spraakherkenning, een verzameling van uitspraken). Het lerend programma gaat een *model* gebruiken waarbij een aantal parameters moet worden ingevuld. Met de waarden van de parameters kan het model een zinnige veralgemening geven van probabiliteit. Hebben we nu een item (een patiënt, of een geluidsopname) dan kunnen we met deze veralgemeende probabiliteit en de regel van Bayes uitspraken doen over dit item.

Het is dus best mogelijk dat er nooit een 83-jarige patiënt met hoofdpijn is geweest, maar met dit model kunnen we een zinnige probabiliteit geven aan de diagnose dat hij bijvoorbeeld malaria heeft.

4.2 DE REGEL VAN BAYES



Figuur 4.1. De regel van Bayes.

De *voorwaardelijke kans* $p(a|b)$ is de kans dat a waar is gegeven dat b waar is. Noteren we $\#(a)$ (spreek uit: aantal a) voor het aantal toestanden waarin a waar is, en $\#U$ het aantal toestanden in het (veralgemeende) universum. Dan is

$$p(a) = \frac{\#(a)}{\#U}, \qquad p(a|b) = \frac{p(a\&b)}{p(b)} = \frac{\#(a\&b)}{\#(b)},$$

waarin & de logische-**en**operator is. We kunnen de rechtse gelijkheid ook schrijven in de vorm $p(b)p(a|b) = p(a\&b)$. Nu is duidelijk dat $p(a\&b) = p(b\&a)$, en dus is

$$p(b)p(a|b) = p(a)p(b|a).$$

Nemen we een klein numeriek voorbeeldje: we nemen een universum met 10.000 vogels. 1000 daarvan zijn raven, en er zijn in totaal 2 witte vogels. Er is exact 1 witte raaf. Duidelijk is $p(\text{raaf}) = 0{,}1$, $p(\text{wit}) = 0{,}0002$, $p(\text{raaf}|\text{wit}) = 0{,}5$ en $p(\text{wit}|\text{raaf}) = 0{,}001$. Bijgevolg is $p(\text{wit})p(\text{raaf}|\text{wit}) = p(\text{raaf})p(\text{wit}|\text{raaf}) = 0{,}0001$. Merk op dat de voorwaardelijke waarschijnlijkheden op zich niet symmetrisch zijn: in ons voorbeeld is het zeer onwaarschijnlijk dat een raaf die we aanduiden wit is, maar het is wel waarschijnlijk dat een witte vogel die

we aanduiden een raaf is. De gelijkheid slaat alleen op het voorkomen van de combinatie in de algemene populatie. De gelijkheid is de wet van Bayes, die meestal wordt uitgedrukt in een niet-symmetrische vorm:

$$p(a|b) = \frac{p(a)p(b|a)}{p(b)}.$$

In deze vorm kan ze gebruikt worden om $p(a|b)$ te berekenen. Dit lijkt een beetje bizar, omdat het rechterlid er veel ingewikkelder uitziet dan het linkerlid. Toch zijn er omstandigheden waarin voldoende uitdrukkingen in het rechterlid gekend zijn, terwijl het linkerlid relevante informatie geeft.

Meer bepaald is dit het geval bij problemen die we kunnen beschouwen als een classificatieprobleem met zeer veel klassen. In dit geval hebben we een aantal vooraf opgegeven hypothetische uitspraken h_i, $i = 1, 2, \ldots$. We veronderstellen dat voor een willekeurig item exact één hypothetische uitspraak waar is. Al wat we hebben is het resultaat van een aantal metingen op het item, die resulteren in een uitspraak d (d staat voor data). Wat is nu de kans dat een hypothese h_i waar is, gegeven de data d? de regel van Bayes geeft ons

$$p(h_i|d) = \frac{p(h_i)p(d|h_i)}{p(d)}. \tag{4.1}$$

Als voorbeeld kunnen we een medische diagnose nemen. Er is een groot aantal ziektes (en combinaties van ziektes). Deze leiden tot de lijst van hypothesen, waarbij h_i dan staat voor de uitspraak "De patiënt heeft het i-de ziektepatroon". Er is een groot aantal mogelijke symptomen. Als we een patiënt hebben, dan kunnen we de symtomen vaststellen, wat ons leidt tot een uitspraak d over de symptomen. We willen dan graag weten welk ziektepatroon het meest waarschijnlijk is (en of er verschillende ziektepatronen zijn die redelijk waarschijnlijk zijn), en we leiden dit af uit vergelijking (4.1). $p(h_i)$ is de kans dat ziektepatroon i voorkomt in het universum van patiënten, en dit is waarschijnlijk bekend uit de medische literatuur (of uit de krant, als het bijvoorbeeld gaat over een griepepidemie). $p(d|h_i)$ is de kans dat ziektepatroon i juist de opgegeven symptomen veroorzaakt, en ook dat, veronderstellen we, behoort tot de medische kennis. Blijft de factor $p(d)$. Theoretisch kunnen we deze berekenen uit de gegevens: als we veronderstellen dat er bij elke patiënt juist één i is waarvoor h_i waar is, dan is

$$p(d) = \sum_i p(h_i \& d) = \sum_i p(h_i)p(d|h_i).$$

Maar we hebben de expliciete waarde van $p(d)$ niet nodig. Immers, als we de i zoeken met de grootste waarde voor $p(h_i|d)$, dan is dit equivalent met het zoeken van de i met de grootste waarde voor $p(h_i)p(d|h_i)$: de factor $p(d)$ is immers onafhankelijk van i. Het is ook logisch dat de waarde $p(d)$ onbelangrijk is. Immers, we weten dat d waar is uit de waarnemingen.

De waarschijnlijkheidsverdeling gegeven door de waarden $p(h_i)$ noemt men de *a priori*verdeling. Zij duidt de waarschijnlijkheid aan van de hypothesen vooraleer de data bekend zijn. De waarden $p(h_i|d)$ noemt men de *a posteriori*verdeling die de waarschijnlijkheid aangeeft nadat de data bekend geworden zijn.

In een aantal gevallen is de a prioriverdeling niet gekend, en moeten we deze schatten. De klassieke methode is dan om alle hypothesen dezelfde waarschijnlijkheid toe te kennen, de zogenaamde methode van de uniforme waarschijnlijkheid. Als een fenomeen wordt beschreven door een aantal variabelen, wordt voor elke variabele elke mogelijke waarde even waarschijnlijk geacht. Dit lijkt een redelijke veronderstelling, maar deze methode is echter een noodoplossing die tot eigenaardige contradicties leidt. Nemen we bijvoorbeeld dat we

een punt in een vlak hebben. We kunnen twee soorten coördinaten hebben, carthesische en poolcoördinaten, en deze leiden tot verschillende waarschijnlijkheidsverdelingen. Vaak beïnvloedt de keuze van een mogelijke voorstellingen het resultaat echter niet of weinig.

Gegeven een a prioriverdeling voor de hypotheses en de waarschijnlijkheden $p(d|h_i)$ voor de de data kunnen we nu in principe de waarschijnlijkheid van elke hypothese berekenen. In de praktijk kan dit echter tegenvallen, bijvoorbeeld als er verschrikkelijk veel hypothesen zijn. Als voorbeeld kunnen we spraakherkenning nemen. Bij gewone spraak is er niet altijd een duidelijke afscheiding tussen de opeenvolgende woorden. Bijgevolg moeten we een lange reeks klanken samennemen en dan proberen uit te vinden welke zin er uitgesproken is. Met elke mogelijke zin komt er een hypothese overeen, en er zijn zeer veel mogelijke zinnen[1]. We kunnen niet eens alle mogelijke zinnen opsommen, laat staan dat we voor elke zin de waarschijnlijkheid kunnen berekenen. We moeten dus in dit geval een model ontwerpen dat alle mogelijk zinnen beschrijft, met een benaderende waarde voor hun waarschijnlijkheid. Bovendien gaan we niet de waarde $p(h_i|d)$ berekenen voor alle hypothesen, maar gaan we ons concentreren op het vinden van de hypothese die $p(h_i|d)p(h_i)$ optimaliseert. Hiervoor zullen we gebruik maken van zogenaamde Markovketens.

4.3 MARKOVKETENS EN -MODELLEN

Een Markovketen is een speciaal soort automaat, die veel gebruikt wordt in het modelleren van tijdsafhankelijke processen. Er zijn twee essentiële verschillen tussen een Markovketen en een klassieke automaat. Het eerste is dat een Markovketen een *producerende* eenheid is die uitvoer genereert zonder uit te gaan van een specifieke invoer: een Markovketen is dus autonoom. Het tweede verschil is dat een Markovketen een stochastische werking heeft[2]. Bij het werken met Markovketens maken we gebruik van discrete tijd met ogenblikken $t = 0, 1, \ldots$.

Formeel wordt een Markovketen gekarakteriseerd door de volgende elementen:

(1) Een eindige verzameling van *staten* $S = \{s_1, \ldots, s_n\}$. Een van deze staten is de beginstaat (nemen we aan dat deze s_1 is). Er kan eventueel een eindstaat zijn; als deze er is veronderstellen we dat deze s_n is.
(2) Een *transitiematrix* T. Dit is een $n \times n$-matrix van reële getallen.
(3) Een *uitvoeralfabet* $A = \{a_1, \ldots, a_{k-1}\} \cup \{a_k\}$. Hierin staat a_k voor het lege symbool.
(4) Een *uitvoermatrix* U. dit is een $n \times k$-matrix van reële getallen.

Zoals we zullen zien kent een Markovketen een probabiliteit toe aan elke string van karakters in $\{a_1, \ldots, a_{k-1}\}*$. Als we een Markovketen in gang zetten zal hij zo een string produceren. Vermits een Markovketen niet-deterministisch is hoeft dit niet altijd dezelfde string te zijn, en we kunnen de kans berekenen dat een Markovketen een bepaald resultaat geeft. Op elk ogenblik is de Markovketen in een bepaalde staat; op $t = 0$ is dit de beginstaat s_1. Als we de keten activeren krijgen we dus een functie $i : \mathbb{N} \to \{1, \ldots, n\}$ die voor elk moment t de index van de staat op dat moment geeft. Meer formeel is op elk gegeven ogenblik t de staat $s_{i(t)}$. De evolutie van de keten is zoals gezegd niet-deterministisch. Is op een bepaald ogenblik de keten in een bepaalde staat $s_{i(t)}$, dan is de staat op het volgende moment niet uniek bepaald: de Markovketen heeft verschillende mogelijkheden, elk met

[1] Linguïsten gaan bij hun beschrijving van taal uit van sprekers en luisteraars met een oneindig groot werkgeheugen voor taal. Hieruit leiden ze modellen af waarin het aantal mogelijke zinnen oneindig is. Overigens steunen ze zich dan op deze modellen om te argumenteren dat mensen een oneindig groot werkgeheugen hebben voor taal.

[2] In tegenstelling tot een niet-deterministische automaat is een Markovketen dus niet-deterministisch.

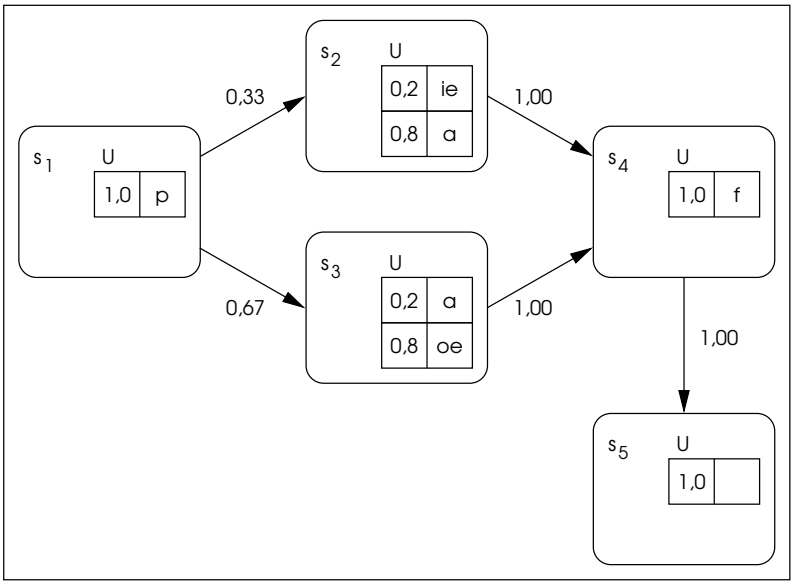
zijn eigen probabiliteit. Deze probabiliteiten worden gegeven door T:

$$T_{kj} = p(i(t+1) = j | i(t) = k).$$

Het is altijd zo dat voor willekeurige k

$$\sum_j T_{kj} = 1$$

Uit de formule volgt dat de probabiliteit onafhankelijk is van de tijd t, en dat de waarschijnlijkheid om op tijdstip $t+1$ een bepaalde staat te bereiken alleen afhangt van de staat op het vorige tijdstip[3]. Het is duidelijk dat alle T_{ij} niet-negatief moeten zijn en dat $\sum_j T_{ij} = 1$ voor willekeurige i. De uitvoer van de Markovketen kunnen we ook beschouwen als een functie van de tijd, zodat we $a_{j(t)}$ kunnen schrijven. Deze uitvoer is ook een probabilistische functie van de staat, waarbij de probabiliteiten gegeven worden door de matrix U. Is op een bepaald ogenblik t de staat $s(t) = s_i$, dan is de waarschijnlijkheid dat symbool a_j wordt buiten gebracht, m.a.w. dat $j(t) = j$, gelijk aan U_{ij}. Ook hier geldt dat $U_{ij} \geq 0$ en dat $\sum_j U_{ij} = 1$. Bij een eenvoudige Markovketen is de uitvoer uniek bepaald door de staat, en vice versa: we kunnen dus de staat afleiden uit de uitvoer van de keten. In ons formalisme beteken dit dat $k = n$ en dat $U_{ii} = 1$ (en dus dat $U_{ij} = 0$ als $i \neq j$). Bij een



Figuur 4.2. HMM.

verborgen Markovmodel (Eng: HMM, wat staat voor Hidden Markov Model) is er geen één-éénrelatie tussen staat en uitvoer: een staat kan verschillende mogelijke uitvoeren hebben, en omgekeerd kan een uitvoer horen bij verschillende staten. Vandaar dat het model verborgen blijft achter de uitvoer.

We zeggen dat de keten een eindstaat heeft (en deze eindstaat is dan s_n) als $T_{nn} = 1$, en $U_{nk} = 1$, waarbij a_k het lege symbool is. Komt zulke keten in de eindstaat dan genereert hij geen uitvoer meer. Hebben we twee Markovketens M_1 en M_2, en heeft M_1 een eindstaat,

[3] Er zijn Markovketens die ook staten op vroegere tijdstippen in acht nemen, maar het is mogelijk deze te herleiden tot de eersteordeketens die we hier bekijken door extra staten in te voeren.

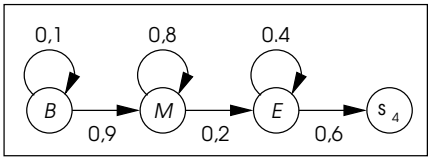
dan kunnen we de concatenatie doorvoeren en dan krijgen we een nieuwe Markovketen $M_1 M_2$ door de eindstaat van M_1 te identificeren met de beginstaat van M_2. Het kan handig zijn om ook voor de beginstaat geen uitvoer te voorzien, maar dit wordt niet altijd zo gedaan.

Kijken we hoe de theorie van HMM's wordt toegepast op spraakherkenning. We zullen HMM's hebben op drie niveaus:

(1) We zullen een HMM hebben voor elk *foneem*.
(2) We zullen een HMM hebben voor elk *woord*.
(3) We zullen een HMM hebben voor alle mogelijke zinnen samen.

Het kan lijken alsof we een niveau vergeten zijn, namelijk dat waar een er een HMM is voor elke zin. We zullen later zien waarom dit niveau zinloos is.

We gaan ervan uit dat elk woord een vaste opeenvolging is van *fonemen*, dit zijn betekenisvolle klankenreeksen die soms met een letter overeenkomen (maar de a in kat is natuurlijk een ander foneem dan de a in kater), soms met een lettergroep (zoals ch, eu, ...). Sommige woorden kunnen uitgesproken worden met verschillende fonemen (sommige mensen zeggen tr*a*m, anderen tr*e*m): we zullen voor de eenvoud deze als verschillende woorden beschouwen. Vermits we met klanken werken beschouwen we homoniemen als hetzelfde 'woord': meid, mijd, mijdt en mijt zijn voor ons hetzelfde woord.



Figuur 4.3. Het HMM voor het foneem 't'.

Wie fonemen ontleedt ontdekt dat ze uit meer dan één klank bestaan. Dit is het duidelijkst bij plofklanken: zo bestaat het 't'-foneem uit een scherpe tik gevolgd door een sissend geluid. We gaan niet diep in op het begrip klank; laten we volstaan met te zeggen dat het bij spraakherkenning gebruikelijk is het ontvangen geluidssignaal op te delen in frames van zowat 10 ms en voor elk frame de klank te bepalen door frequentieanalyse. Hetzelfde foneem kan traag of snel worden uitgesproken, zodat de klanken ook langer of korter zijn. Het meest gebruikelijke model gaat uit van drie staten voor het HMM voor elk foneem: B, M en E (Begin, Midden en Eind), gevolgd door de eindstaat s_4. De variatie in lengte van de klanken wordt aangegeven door de probabiliteit om in dezelfde staat te blijven. Nemen we het voorbeeld van het 't'-foneem, zoals afgebeeld in Figuur 4.3. De probabiliteit van de overgang $B \to B$ is bijna nul, zodat het HMM zelden langer dan één frame in B blijft. De overgang $M \to M$ is veel waarschijnlijker: gemiddeld zal het HMM een viertal frames in de staat M blijven voor het naar staat E overgaat. Met elk van de staten komt een probabilistische uitvoer overeen, gegeven door de matrix U. Zo zijn er verschillende soorten plofklank die met de B-staat kunnen overeenkomen, elk met haar eigen probabiliteit.

Het HMM voor een woord wordt gevormd door de HMM's voor zijn fonemen aaneen te schakelen. Heeft woord w_i f_i fonemen, dan heeft het HMM $3f_i$ staten (de eindstaten rekenen we niet mee). Tenslotte gaan we nog een model opstellen voor de waarschijnlijkheid van zinnen. Onze hersenen voeren hierbij een grammaticale analyse uit. Dergelijke grammaticale analyse is echter zeer complex, en men slaat dan ook meestal de andere denkpiste in, die van de *n-grammodellen*. Een n-gram is een sequentie van n woorden. Als men de probabiliteit van alle n-grammen kent, dan kan men dit gebruiken om uitgaande van $n-1$

			opvolger							
		unigram	de	een	het	systeem	code	we	is	zullen
voorganger	de	9207	0	4	0	0	120	0	11	0
	een	7584	9	6	0	66	5	2	7	0
	het	5645	48	26	5	285	0	0	307	1
	systeem	493	9	10	13	0	0	1	13	1
	code	454	9	6	13	0	0	2	20	0
	we	2706	172	251	110	0	1	0	0	35
	is	4228	337	381	464	0	6	19	22	0
	zullen	136	6	3	1	0	0	41	0	0

Figuur 4.4. Een gedeelte van een unigram en een bigram, berekend op een totaal van 177.000 woorden.

woorden het volgende woord te voorspellen. Men neemt hiervoor gewoon alle n-grammen die beginnen met de gegeven $n-1$ woorden. Op deze manier krijgt men het unigrammodel, dat gewoon de relatieve frequentie van alle woorden omvat, het bigrammodel dat alle mogelijke combinaties van twee opeenvolgende woorden bekijkt, enzoverder. Op deze manier kunnen we een HMM opstellen voor de ganse taal. Het eenvoudigst is dit uit te leggen voor het bigrammodel.

Eerst is er een beginknoop. Die is verbonden met alle beginknopen van woorden en genereert zelf geen uitvoer. De overgangswaarschijnlijkheid hier wordt gegeven door het unigrammodel. Het uiteindelijke HMM is de unie van de HMM's voor alle woorden. De overgangswaarschijnlijkheid van de eindknoop van een woord naar de beginknoop van een volgend woord is gegeven door de relatieve frequentie van het bijbehorende bigram. Als we bijvoorbeeld de gegevens uit Figuur 4.3 bekijken, dan zien we dat 'de' 9207 keer voorkomt. 120 keer daarvan wordt 'de' gevolgd door'code' zodat de overgangswaarschijnlijkheid 'de→code' is gelijk aan $120/9207$. Is nu W het aantal woorden en f het gemiddeld aantal fonemen per woord dan is het totale aantal staten van dit HMM $3fW$. Voor een taal zoals het Nederlands is f ongeveer 11, terwijl het aantal woorden W (zoals te vinden is in een woordenlijst zoals het Groene Boekje, dat niet eens alle woordvormen bevat) ongeveer 140.000 is.

Het spreekt vanzelf dat we niet meer de waarden gegeven door de regel van Bayes in (4.1) kunnen berekenen voor elke mogelijke sequentie van woorden, gewoon omdat er veel te veel mogelijke sequenties zijn (als we ons beperken tot alle mogelijke sequenties van 10 woorden dan zijn dit er ongeveer 3×10^{51}). Wat we wel kunnen doen is het *optimale pad* berekenen. Wat is dit optimale pad? We hebben een HMM, en we kunnen dit gebruiken om de waarschijnlijkheid van een bepaalde uitvoer te berekenen. Maar bij spraakherkenning hebben we een uitvoer gekregen en we willen weten hoe die uitvoer gegenereerd is. We kennen dus de functie $j(t)$, die voor elk tijdstip aangeeft dat de uitvoer $a_{j(t)}$ was. Hieruit kunnen we echter niet altijd eenduidig de functie $i(t)$, die de staat geeft als functie van de tijd, afleiden. Immers, als op een bepaald tijdstip t de uitvoer gelijk is aan a_j, dan weten we alleen dat op dat tijdstip de staat een staat s_i moet zijn met $U_{ij} \neq 0$. We zoeken nu de indexfunctie $i(\cdot)$ die het meest waarschijnlijke is, gegeven de uitvoer $j(\cdot)$. Meer formeel: we zoeken $i(\cdot)$ die $p(i(\cdot)|j(\cdot))$ zo groot mogelijk maakt. Volgens de regel van Bayes geldt

$$p(i(\cdot)|j(\cdot)) = \frac{p(i(\cdot))p(j(\cdot)|i(\cdot))}{p(j(\cdot))}.$$

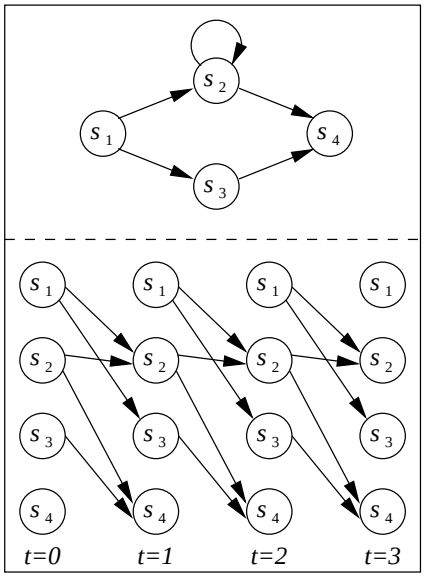
De term $p(j(\cdot))$ is niet belangrijk (want is hetzelfde voor alle mogelijke sequenties). $p(i(\cdot))$

is gemakkelijk te berekenen: het is gewoon het product van de overgangswaarschijnlijkheden $T_{i(t),i(t+1)}$. Ook $p(j(\cdot)|i(\cdot))$ is gemakkelijk te berekenen: de waarschijnlijkheid van een bepaalde uitvoer $j(t)$ op tijdstip t, gegeven dat de staat $i(t)$ is wordt gegeven door $U_{i(t)j(t)}$. Bijgevolg geldt

$$p(i(\cdot)|j(\cdot)) = \alpha \prod_{t=0}^{t=T-1} T_{i(t),i(t+1)} \prod_{t=0}^{t=T} U_{i(t)j(t)}, \tag{4.2}$$

waarin T het totaal aantal tijdstappen is, en α de factor $1/p(j(\cdot))$ is die onafhankelijk is van $i(\cdot)$. We kunnen het probleem om de functie $i(\cdot)$ te vinden opvatten als het probleem om een optimaal pad te vinden in een bepaalde gerichte graaf. De graaf construeren we als volgt:

- De knopenverzameling van de graaf is de verzameling koppels (s_i, t), waarbij $0 \leq t \leq T$. (Merk op dat we een tijdstip $t = 0$ toevoegen om een begintoestand te hebben).
- Er is een verbinding van (s_i, t_1) naar (s_j, t_2) als en slechts als $t_2 = t_1 + 1$ én er een verbinding is van s_i naar s_j in het HMM, met andere woorden als $T_{ij} \neq 0$.



Figuur 4.5. Een HMM (boven) en de bijbehorende tijdgraaf (onder).

Een tekening van een voorbeeld (Figuur 4.5) zal dit duidelijker maken. De probabiliteiten hebben we weggelaten. Bij het klassieke optimaal pad maken we de totale kost zo klein mogelijk (en de kost is additief), terwijl we hier de probabiliteit zo groot mogelijk willen maken (en de probabiliteit is multiplicatief). Het is dus logisch om de kost van een overgang te definiëren door de logaritme te nemen van de probabiliteit (additief in plaats van multiplicatief) en een minteken bij te voegen (minimum i.p.v. maximum):

$$c((s_i, t) \rightarrow (s_k, t{+}1)) = -\log_2 \left(T_{ik}.U_{kj(t+1)}\right).$$

Merk op dat de kost niet negatief is: zowel T_{ik} als $U_{kj(t)}$ zijn hoogstens gelijk aan 1, en dat we een waarde oneindig krijgen als de overgang van s_i naar s_k onmogelijk is ($T_{ik} = 0$), of als de staat s_k niet kan leiden tot de uitvoer $a_{j(t+1)}$. Indien het systeem ook nog aan foutcorrectie moet kunnen doen wordt $U_{kj} > 0$ genomen voor alle j en k, zodat het laatste geval niet kan voorkomen. We kunnen hier weer een interpretatie geven in termen van informatie-inhoud: de kost om van een staat naar een andere te gaan is de benodigde informatie-inhoud van de overgang plus deze om de uitvoer aan te geven op dat ogenblik. Het startpunt voor ons pad is $(s_1, 0)$. Deze startknoop genereert geen uitvoer, zodat er geen kost voor deze uitvoer moet worden berekend. Het doel bestaat uit uit alle staten op het tijdstip $t = T$.

In principe is het nu mogelijk om het klassieke zoeken in grafen toe te passen. Helaas zijn er tot nu toe geen interessante heuristieken gevonden om een A^*-methode toe te passen. Het dient opgemerkt te worden dat de graaf en speciale vorm heeft, waardoor het zoeken op enigszins andere manier kan verlopen als gewoonlijk. Het gebruikte algoritme heeft dan ook een andere naam gekregen: het is het zgn. Viterbialgoritme. Bij dit algoritme wordt

de graaf tijdslaag per tijdslaag overlopen. Immers, als we voor een bepaalde waarde van t alle knopen (s_i, t) ontwikkeld hebben, dan weten we dat we voor alle knopen van de vorm $(s_i, t+1)$ een correcte waarde hebben voor g, de afschatting voor de kortste weg van de startknoop naar de huidige knoop. Als we het Viterbialgoritme uitdrukken in termen van probabiliteiten, dan nemen we de functie e zodanig dat $\log_2(e((s_i,t))) = g((s_i,t))$. Het algoritme ziet er dan als volgt uit:

1. Voor $t = 0$, initialiseer $e((s_1, 0)) = 1$ en $e((s_i, 0)) = 0$ voor $i \neq 1$.
2. Voor $t = 1, 2, \ldots$, en voor alle staten s_i, zoek de k die de functie $e((s_k, t-1))T_{ki}$ maximaliseert, en stel

$$e((s_i,t)) = e((s_k,t-1))T_{ki}U_{ij(t)}.$$

Het is duidelijk dat dit algoritme zich uitstekend leent tot parallellisatie. Immers, voor een bepaalde tijdstap zijn de berekeningen voor alle staten onafhankelijk van elkaar. Bovendien is het aantal parallel uit te voeren berekeningen onafhankelijk van de tijdstap, want er is één berekening per staat. Nadat we de e-waarden berekend hebben voor het tijdstip $t = T$ krijgen we een staat (s_i, T) met de grootste waarschijnlijkheid: dit komt overeen met het kennen van het laatste woord in de zin. Om de andere woorden in de zin te kennen moeten we, op het ogenblik dat we de e-waarden berekenen voor alle tijdstippen, voor elke knoop een wijzer bijhouden naar de voorganger die de beste e-waarde opleverde. Op deze manier kunnen we het optimale pad reconstrueren.

4.4 LEREN VAN HET MODEL

Bij het aanleren van de HMM's voor dit taalmodel gaat men uit van een corpus van teksten waarin men de mogelijke woordcombinaties telt. Zo zal men bij het bigrammodel de probabiliteit[4] $p(w(t) = w_j | w(t-1) = w_i)$ gaan schatten door $n(w_i w_j)/n(w_i)$, waarbij $n(w_i w_j)$ het aantal keer is dat w_j voorkomt vlak na w_i, en $n(w_i)$ het aantal keer dat w_i voorkomt in de tekst. Er is hier sprake van een groot aantal parameters: als we een schatting aannemen van 140.000 woorden in de taal, dan zijn er voor het bigrammodel zowat 20 miljard mogelijke woordcombinaties. Toch is er hier geen sprake van het massief leren zoals bedoeld in het inleidende hoofdstuk van deze cursus. Immers, elke parameter heeft een duidelijke en vooraf vastgelegde betekenis.

Het is duidelijk dat er, zelfs in een zeer groot corpus van teksten, een aantal mogelijke woordsequenties niet gaan voorkomen. In ons voorbeeld uit Figuur 4.3 komt de combinatie 'we is' niet voor. 'We' is een voornaamwoord in het meervoud, en 'is' staat in het enkelvoud: bijgevolg is de combinatie 'we is' onwaarschijnlijk, maar ze komt wel voor in deze zin én in de vorige. Bijgevolg gaat men bij het aanleren van het bigrammodel niet beginnen met alle waarschijnlijkheden op nul te zetten, maar wel met ze een zeer kleine waarde te geven.

Men kan ook waarschijnlijkheden aangeven in functie van de *grammaticale categorie* van woorden. Zo worden lidwoorden vaak gevolgd door een substantief of door een adjectief, maar veel minder door een persoonlijk voornaamwoord. Eventueel kan men hier gebruik maken van verbuigingen en vervoegingen. In plaats van een probabiliteit $p(w(t) = w_j | w(t-1) = w_i)$ voor woorden krijgen we dan een probabiliteit $p(g(t) = g_j | g(t-1) = g_i)$ voor grammatische categorieën g_i. Om terug te kunnen overgaan naar woorden, wat nodig is voor ons herkenningsmodel, gebruiken we dan de benaderingsfor-

[4] Het gebruik van t in deze paragraaf wijkt af van dat zoals gebruikt voor het fonetisch model. Hier gaan we uit van één tijdstap per woord, en niet van één tijdstap per klankframe.

mule

$$p(w(t) = w_j | w(t-1) = w_i) = \\ p(g(t) = g(w_j) | g(t-1) = g(w_i)).\frac{p(w_j)}{p(g(w_j))}.$$

De laatste factor is nodig om ervoor te zorgen dat weinig voorkomende woorden in een veel voorkomende grammaticale categorie toch de juiste, kleine, waarschijnlijkheid krijgen. Het nadeel van deze methode is dat men nood heeft aan een *geannoteerd* corpus, waar bij elk woord de grammaticale categorie staat aangeduid. Sommige woorden (zoals slaap, even) kunnen tot verschillende categorieën behoren. In het ideale geval gaat men dan elke keer dat het woord voorkomt in het corpus een annotatie aanbrengen, maar er zijn andere oplossingen mogelijk.

Dit model houdt geen rekening met de betekenissen van een woord. Als we rechtstreeks een bigrammodel berekenen uitgaande van een corpus dat groot genoeg is dan zal dit model de informatie niet bevatten dat 'blauwe lucht' een meer waarschijnlijke combinatie is dan 'blauwe woensdag'. Men kan dit opvangen door meer nauwkeurige woordcategorieën te gebruiken (en bijvoorbeeld dingen die een kleur hebben apart te klasseren). Dit lost het probleem niet volledig op: 'paarse lucht' is veel zeldzamer dan 'blauwe maandag'. Bovendien is het zo dat als men het aantal categorieën gaat vermeerderen het annoteringswerk navenant toeneemt.

Het grammatisch bigrammodel kan men gaan gebruiken op twee manieren. Het kan gebruikt worden om de ontbrekende waarschijnlijkheden uit het gewone bigrammodel in te vullen, maar ook kan men dit model rechtstreeks gaan gebruiken als model met minder parameters.

Ten slotte kan men de efficiëntie van de methode vergroten in het geval men een zeer specifieke toepassing heeft (denken we aan het voorbeeld van een routeplanner). Men heeft dan een beperkte, zeer specifieke woordenschat (zo zal het woord 'bestemming' bij een routeplanner veel meer voorkomen dan in het algemene taalgebruik). Spraakherkenning scoort dan ook veel beter bij zulke toepassingen dan bij algemeen gebruik.

Een gelijkaardige opmerking kan gemaakt worden bij het fonetisch model. Elke spreker klinkt anders, en men kan de performantie verbeteren door het fonetisch model in te stellen op een specifieke spreker. Hiertoe laat men de spreker een vooraf vastgelegde tekst inlezen. Vermits de tekst gekend is kan men daarmee de parameters voor het fonetisch model (zowel de overgangswaarschijnlijkheden als de uitvoerwaarschijnlijkheden) schatten met de regel van Bayes. Dit houdt dan natuurlijk wel in dat het lerend programma de invoer van de proefpersoon kan synchroniseren met de tekst. Het algoritme gaat dan ook uit van een algemeen spraakherkennend model, en ziet er ongeveer als volgt uit:

0. Stel een HMM op voor de in te lezen tekst (dit maakt dus geen gebruik van de probabiliteiten van bepaalde woordsequenties: men zet gewoon de HMM's voor de woorden in de tekst op een rij).
1. Laat de proefpersoon de tekst één of meerdere keren inlezen.
2. Voor elke inleesbeurt, bepaal het optimale pad met het Viterbialgoritme.
3. Voor elk foneem (men zal er bij het maken van de voor te lezen tekst ervoor zorgen dat elk foneem voldoende voorkomt in de tekst), neem de delen van het optimale pad die instaan voor dit foneem, en bepaal de overgangswaarschijnlijkheden. Dit gebeurt als volgt. Elk deelpad zal bestaan uit een sequentie van B-, M- en E-staten, en de enige mogelijke overgangen zijn $B \to B$, $B \to M$, $M \to M$, $M \to E$, $E \to E$ en $E \to s_4$. De homogene overgangen ($B \to B$, $M \to M$, ...) kunnen 0 of meer keren voorkomen, de heterogene overgangen ($B \to M$, $M \to E$, ...) exact eenmaal voor elke

keer dat het foneem voorkomt. De overgangswaarschijnlijkheden volgen uit de verhoudingen tussen de frequenties.

4. Vermits we voor elk tijdstip t de staat kennen (zoals bepaald voor het optimale pad) en de uitvoer, kunnen ook de uitvoerwaarschijnlijkheden U_{ij} worden ingeschat.

Om een idee te geven van de kracht van de bi- en trigrammodellen, volgt hier een voorbeeld van uitvoer voor het bigrammodel opgesteld aan de hand van de cursustekst:
computer waarbij de gewone regel in het heeft voor de waarde we hebben is dit gebruiken gegeven door een antwoord van vectoren de functie heeft bij te zijn een zeer groot aantal a en bekijken voor de conclusie is de regelbank de conclusie zijn die vaak kan een enkele de premisse en dus de waarde van de energie voor een neuraal de staat hier tussen verschillende eigenschappen van en ook nog niet over welke op zo klein de mogelijke zet is immers een aantal mogelijke zetten wel is of een groot aantal gewone methode om er geen problemen een binaire uitvoer voorkomt heeft Watson geen informatie die kunnen sommige problemen omdat de kans op een voorbeeld van de gewichten bepaald door de is een hond a op een groter is we de verzameling van de andere programma zijn veel te maken de tweede regel dat het netwerk kan denken in het heeft een voorbeeld van verbindingen het gebruik van het semantisch net dezelfde gegevens van een bepaald wordt ons netwerk dit kan immers de kans dat men kan het nieuwe toestand op elkaar de gewichten van een pad of de leerverzameling de

HOOFDSTUK 5
ACTIEVE SYSTEMEN

Klassieke regelbanken worden gebruikt voor expertsystemen, die toelaten om conclusies te trekken uit bepaalde gegevens, eventueel met een zekere mate van onzekerheid. In dit geval is het resultaat *feitenkennis*, al kan deze feitenkennis natuurlijk gebruikt worden om acties te sturen.

Het is echter ook mogelijk om systemen te ontwikkelen bij wie het geheel van de kennis (dus niet alleen de zojuist vermelde feitenkennis maar ook de algemene kennis) zo geformuleerd is dat ze rechtstreeks een actieve eenheid (de *agent*[1]) kunnen aansturen. Het resultaat van gegevensverwerking wordt niet meer geformuleerd als feitenkennis maar rechtstreeks omgezet in een actie. Daarmee wordt het gedrag van een actieve eenheid in een bepaalde omgeving gestuurd. Deze eenheid kan bijvoorbeeld een robot zijn, maar ook een mens die zich laat leiden door de regelbank. De omgeving waarin de eenheid zich bevindt word *de wereld* genoemd. Dit is zelfs zo als het gaat om een robot die zich op een andere planeet bevindt. De wereld bevindt zich in een bepaalde *toestand*. Deze toestand kan veranderen door de dynamiek van de wereld zelf, of door bepaalde acties van de actieve eenheid. Meestal heeft de regelbank niet de beschikking over een volledige beschrijving van de toestand. Op een of andere manier (bijvoorbeeld door sensoren) kent de regelbank een aantal gegevens over de toestand: deze verzameling van gegevens wordt de *visie* van de regelbank genoemd. Op basis van deze visie bepaalt de regelbank een actie voor de eenheid. Hierdoor ontstaat een nieuwe toestand, die leidt tot een nieuwe visie, waarna weer een nieuwe actie kan worden voorgesteld. Uiteraard kan, in sommige systemen, de actie eruit bestaan om niets te doen. Dit model kan in veel gevallen worden toegepast. Enige voorbeelden:

- Bij een chemisch productieproces kan de wereld bestaan uit de inhoud van een bepaalde tank. De toestand van deze wereld verandert door chemische reacties. De regelbank kan een aantal acties voorstellen, zoals het veranderen van druk of temperatuur, het toevoegen van bepaalde stoffen, en zo verder, die deze reacties beïnvloeden. De visie op de wereld wordt bepaald door gegevens van bepaalde sensoren in de tank.

- Een klassiek model is dit van een wereld die bestaat uit een bord met verschillende vakjes. Op elk van deze vakjes kan zich een van de volgende items bevinden:

  - De eenheid zelf.
  - Een schat.
  - Een muur, waardoor het vakje niet bezocht kan worden.

[1] Geen politieagent. Het woord agent wordt hier op zijn Engels uitgesproken.

  - Een of meer monsters.
  - Allerhande hulpmiddelen.

  De bedoeling is dat de eenheid op zoek gaat naar de schat, zonder door een monster te worden opgegeten. In sommige versies kunnen de monsters bewegen, in andere niet. De visie op de wereld bestaat in dit geval uit informatie over de omgeving van de eenheid.

- Sinds een aantal jaren zijn er ook een aantal robotjes op de markt gekomen die een specifieke taak uitvoeren, zoals het gras maaien of stofzuigen. Deze robotjes moeten in staat zijn hun omgeving te exploreren om hun taak uit te voeren, zonder dat ze een plan krijgen van die omgeving. Zo zal een stofzuigrobot die in een kamer geplaatst wordt door de hele kamer rijden en waar nodig stofzuigen. Als zijn batterij leeg geraakt zal hij een stopcontact zoeken om zichzelf op te laden. Hierbij moet hij obstakels vermijden, ervoor zorgen dat hij niet van de trap valt, enzovoorts. Merk op dat de visie ook eigenschappen van de actieve eenheid zelf kan aangeven, zoals de toestand van de batterij. Dat is een complicatie die we in deze cursus niet formeel zullen uitwerken.

- Een programma dat resources verdeelt over verschillende aanvragers is een actieve agent.

- Een (voorlopig nog alleen gedeeltelijk) zelfrijdende auto valt ook grotendeels onder de noemer actieve eenheid, al heeft zo'n wagen natuurlijk ook nog een reisdoel.

Een spel waarbij een spelboom kan worden geanalyseerd is een speciaal geval van zo'n actief systeem. De spelboom is een methode om te voorspellen wat de reactie gaat zijn van de wereld (de tegenstrever) op een actie van de agent. In het geval van een kleine spelboom konden we daarmee exact het verloop van het spel met een perfect spelende tegenstander beschrijven. In het algemeen geval zullen we een andere methode moeten gebruiken om het gedrag van de buitenwereld te beschrijven.

5.1 EENVOUDIGE SYSTEMEN

In deze paragraaf bekijken we systemen die zo eenvoudig zijn dat het mogelijk is een opsomming te maken van alle mogelijke visies, acties en hun combinaties. We gaan uit van een model met discrete tijdsstappen. Bij elke tijdsstap analyseert de actieve eenheid de visie die ze krijgt en kiest op basis daarvan een actie. De buitenwereld reageert op deze actie en presenteert daarmee een nieuwe visie aan de eenheid.

Omdat elke visie slechts een gedeeltelijke beschrijving van de wereld geeft, zitten we ook hier met een mate van onzekerheid. Als model zoeken we dus iets dat lijkt op een HMM. Hierbij zal het uitvoeralfabet van het model overeenkomen met de verzameling van mogelijke visies die de actieve eenheid kan hebben. Er moeten echter twee aanpassingen gebeuren:

(1) Actieve eenheden hebben een doel. We moeten dus een maat invoeren die ons toelaat om te meten in hoeverre de actieve eenheid slaagt in de doelstelling.
(2) Actieve eenheden nemen beslissingen die de wereld beïnvloeden. Ons HMM moet dus uitgebreid worden zodat het invoer kan ontvangen en verwerken.

Het eerste is gemakkelijk te verwezenlijken. Ingebouwd in de visie kunnen we een *beloning* opnemen. Dit stellen we voor door een reëel getal. De bedoeling is de beloning te

optimaliseren. We gaan daar later nog op in. Voorlopig merken we al wel op dat de beloning negatief kan zijn, wat overeenkomt met een verlies. Het tweede is iets ingewikkelder. Een HMM heeft een transitiematrix T die, uitgaande van een bepaalde staat, uitgeeft wat de waarschijnlijkheid is om op het volgende moment een bepaalde staat te hebben. Nu hebben de uitgevoerde acties echter invloed op deze waarschijnlijkheden. Bijgevolg moeten we voor elke beslissing d een verschillende transitiematrix $T(d)$ gebruiken. Dit leidt tot de formele definitie voor een HMDM (*H*idden *M*arkov *D*ecision *M*odel). Zo'n HMDM omvat het volgende.

(1) Een eindige verzameling van *staten* $S = \{s_1, \ldots, s_n\}$. Een van deze staten is de beginstaat (nemen we aan dat deze s_1 is). Er kan eventueel een eindstaat zijn; als deze er is veronderstellen we dat deze s_n is.
(2) Een eindige verzameling $D = \{d_1, \ldots, d_\ell\}$ van *beslissingen*.
(3) Voor elke $d \in D$ een *transitiematrix* $T(d)$.
(4) Een *uitvoeralfabet* $A = \{a_1, \ldots, a_k\}$. Dit is het alfabet van visies.
(5) Een *beloningsfunctie* $b : A \to \mathbf{R}$. r staat voor *beloning*. Beloningen kunnen negatief zijn en dus kosten representeren.
(6) Een *uitvoermatrix* U. dit is een $n \times k$-matrix van reële getallen.

In veel gevallen (denk bijvoorbeeld aan onze grasmaairobot) is er een niet-eindigend proces van interactie tussen buitenwereld en actieve eenheid. Beloningen kunnen op elk moment worden toegekend. In het vervolg gaan we uit van systemen waar er eindtoestanden zijn. We spreken dan van een *run* die doorloopt tot we een eindtoestand bereiken: we kunnen dan de totale beloning voor deze run berekenen. Dit maakt geen essentieel verschil voor de leermethodes, maar maakt de uiteenzetting eenvoudiger.

Een actieve eenheid moet nu een *strategie* τ hebben. Deze kent aan elke letter a van het uitvoeralfabet een beslissing $\tau(a)$ toe. Later zullen we ook nog *exploratiestrategieën* bekijken. Daarbij is er sprake van randomisatie zodat de eenheid mogelijkheden kan uittesten.

In het eenvoudigste geval hebben we geen onzekerheid:

(1) De staat komt overeen met de visie. Er is dus een één-éénrelatie tussen A en S, waarbij $k = n$ en $U_{ij} = \delta_{ij}$. Hier is δ_{ij} de Diracdelta gegeven door

$$\delta_{ij} = \begin{cases} 1 & i = j, \\ 0 & i \neq j. \end{cases}$$

We kunnen dus zonder problemen $\tau(s) = d$ noteren als τ in staat s beslissing d neemt.
(2) De overgang naar de volgende staat is volledig gedetermineerd door de beslissing: elke $T(d)_{ij}$ is ofwel 1 ofwel 0. We kunnen de transitie dus uitdrukken als functie van d en s. We noteren

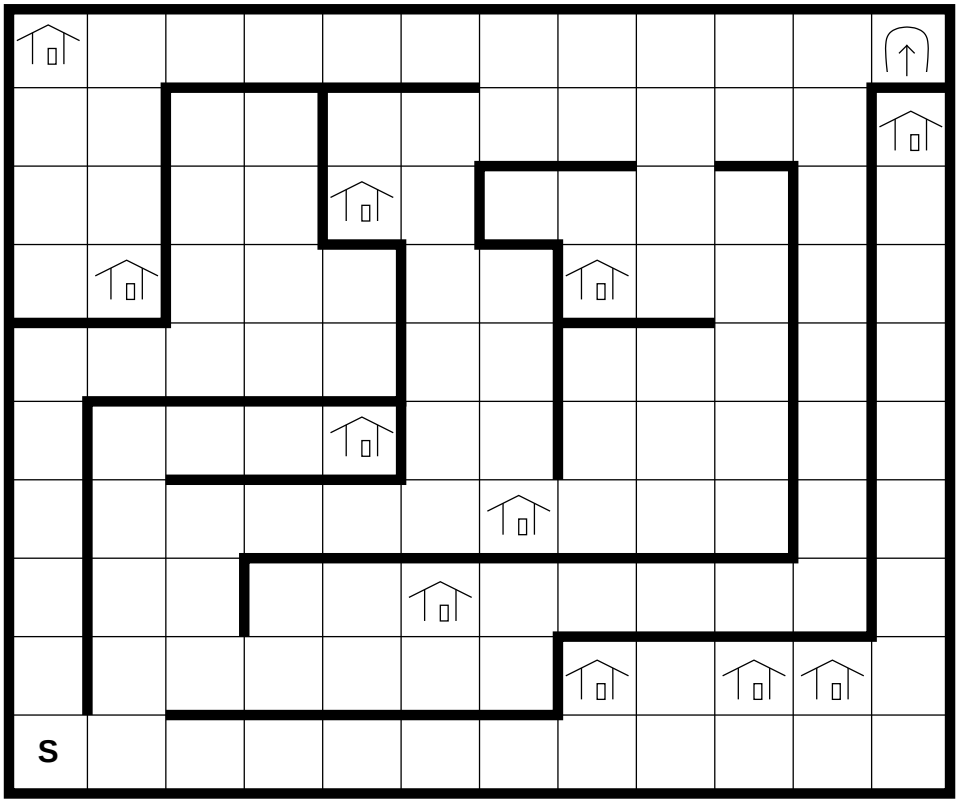
$$T(d, s) = s'$$

als d leidt tot een overgang naar staat s'.

De eerste voorwaarde zegt dat de staat getoond wordt door de visie en dat de H uit HMDM dus mag wegvallen. De tweede zegt dat er geen onzekerheid is bij de overgang naar de volgende staat. Dit zou zinloos zijn bij een Markovmodel, maar hier heeft het wel zin.

We krijgen een deterministisch systeem, waarbij een staat s in één tijdsstap overgaat in de staat $T(\tau(s), s)$.

Bekijken we even een voorbeeld van een eenvoudige wereld. We hebben een doolhof met startpositie s. Op elke plaats kan de eenheid beslissen één vakje verder te gaan (niet schuin), maar ze kan niet door de dikke lijnen heen. Dat kost haar één dag. In elke



Figuur 5.1. Een eenvoudige leerwereld.

schuilhut is er een onbeperkte voorraad proviand, maar de eenheid kan maar voor tien dagen eten en drank meenemen. Haar beginvoorraad is ook maar genoeg voor tien dagen. Is haar voorraad op dan sterft ze, opbrengst 0. Zaak is met zoveel mogelijk voorraad naar de uitgang linksboven te geraken (opbrengst: 1 plus de overblijvende hoeveelheid proviand). Om hiervan een MDM te maken nemen we voor elk vakje van het doolhof 11 staten, aangegeven door de coördinaten van het vakje (beginnend linksbeneden) aangevuld met de voorraad van de eenheid: de beginstaat is dus gegeven door $((0,0),10)$. In een simpel MDM kunnen we in principe aan elke staat een waarde geven, die aangeeft wat een run die vertrekt vanuit die staat opbrengt. Voor de eindstaten waar de eenheid sterft (voorraad 0 en geen uitgang of schuilhut) is de waarde -10. Voor de eindstaten aan de uitgang, $((11,9),i)$ is de waarde bijvoorbeeld $i+1$ (we nemen aan dat aankomen bij de uitgang met voorraad 0 nog altijd een winst oplevert).

Voor de andere vakjes is er een verband tussen de waardering voor de staat door de eenheid en de gevolgde strategie.

Een eenheid die bijvoorbeeld in staat $((10,9),1)$ terechtkomt en dan naar links gaat sterft; gaat ze naar rechts krijgt ze eindwaarde 1. Voor een eindstaat is de waardering niet afhankelijk van de strategie; voor het algemene geval krijgen we de formule

$$w_\pi(s) = b(s) + w_\pi(T(\pi(s), s)). \tag{5.1}$$

Gegeven een strategie π krijgen we dus voor elke staat s een waardering $w_\pi(s)$, die overeenkomt met het resultaat als we π toepassen op s.

Het is duidelijk dat er een optimale strategie τ_{opt} bestaat zodanig dat $w_{\tau_{opt}}(s) \geq w_\tau(s)$ voor elke andere strategie τ. Bovendien zien we gemakkelijk dat de beslissing $\tau_{opt}(s)$ leidt tot een staat s' die voldoet aan

$$w_{\tau_{opt}}(s) = b(s) + \max_{d \in D} w_{\tau_{opt}}(T(d, s)). \tag{5.2}$$

Het kan zijn dat er verschillende optimale strategieën bestaan omdat bepaalde beslissingen gelijkwaardig zijn. Ze leveren echter allemaal dezelfde waardering voor elke staat, zodat we elke staat een waardering w kunnen geven, waarbij de gebruikte optimale strategie geen rol speelt,

$$w^*(s) = b(s) + \max_{\tau} w_\tau(T(d, s)), \tag{5.3}$$

waarbij het maximum genomen wordt over alle strategieën. Voor elke optimale strategie τ_{opt} geldt dat $w_{\tau_{opt}}(s) = w^*(s)$.

Veronderstellen we even dat er een reeks beslissingen zou zijn die een winstgevende lus oplevert. We wijzigen bijvoorbeeld het probleem zo dat elk bezoek aan een schuilhut een bonuspunt oplevert. Dan kan een eenheid heen en weer beginnen te lopen tussen twee schuilhutten om bonuspunten te verzamelen zodat we staten krijgen met een oneindige waardering[2]. We gaan er in het vervolg van uit dat dergelijk lussen niet voorkomen. Ook als alle lussen verlieslatend zijn is het bovendien zo dat een optimaal pad eindig is.

5.1.1 Exploratie zonder onzekerheid

In veel gevallen is het zo dat onze eenheid kan leren door een *exploratiestrategie* te volgen en te zien wat er gebeurt. Een exploratiestrategie is geen klassieke strategie. Immers, een gewone strategie neemt elke keer ze in een bepaalde staat komt dezelfde beslissing, zodat ze niets leert over andere mogelijkheden. Een exploratiestrategie ξ hecht aan elke combinatie (s, d) van een staat en een beslissing een probabiliteit $\xi(s, d)$, waarbij uiteraard

$$\sum_D \xi(s, d) = 1.$$

Komt de eenheid in staat s dan wordt de beslissing genomen met deze probabiliteit, waarbij we voor de eenvoud aannemen dat ze lussen vermijdt. Op deze manier komt ze vroeg of laat in een eindstaat. Daar kan de totale opbrengst W voor deze run bepaald worden.

We gaan er eerst van uit dat de nieuwe staat elke keer gedetermineerd is door de vorige staat en de genomen beslissing. We kunnen nu, voor alle staten die de eenheid gepasseerd is, de benaderde schatting $w(s)$ van $w^*(s)$ updaten door $w(s)$ te vervangen door

$$\max\{W, w(s)\}.$$

Merk op dat de waardering nu niet meer noodzakelijk voldoet aan (5.1). Immers, het is best mogelijk dat er een staat s is die niet behoort tot de huidige run, maar zodanig dat $T(\tau(s), s) = s'$, waarbij s' wel behoort tot de huidige run. Dit houdt in dat de schatting $w_\tau(s')$ misschien verandert, maar $w_\tau(s)$ zeker niet.

Op basis van deze schatting kunnen we een voorlopige optimale strategie opstellen: deze zal ervoor zorgen dat we vanuit een staat s steeds overgaan naar een nieuwe staat met dezelfde waardering. Merk op dat w altijd een onderschatting is van w^*, $\forall s : w(s) \leq w^*(s)$.

Als we kijken naar Figuur 5.1 dan zien we dat er gebieden zijn die nooit tot goede resultaten zullen leiden (bijvoorbeeld de uiterst rechtse doorgang naar boven, die afgesloten is van de uitgang). Aan de andere kant is er een optimale oplossing, maar het is vrij

[2] Dit doet niet toevallig denken aan het niet-definieerbaar zijn van een goedkoopste pad in grafen met lussen met negatief gewicht.

onwaarschijnlijk dat onze lukraak gekozen exploratiestrategie veel kans heeft om daarop uit te komen. Vandaar dat men na verloop van tijd de exploratiestrategie aanpast. Hierbij wordt ervan uitgegaan dat staten met een hoge geschatte waarde veel kans maken om, door verdere exploratie, een nog hogere waardering te krijgen, terwijl staten met een lage waardering waarschijnlijk niet op een goed pad liggen. Een mogelijke methode is hier het zgn. ε-leren. Hierbij kiest men een kleine waarde ε. Komt men in een bepaalde staat, dan kiest men met waarschijnlijkheid $1 - \varepsilon$ voor de beslissing die volgens de gekende schattingen het beste resultaat geeft, en met waarschijnlijkheid ε keert men terug naar de exploratiestrategie ξ. Uiteraard zijn hier tal van variaties op: zo kan men ook een beslissing uitkiezen met gewogen randomisatie, waar het gewicht afhangt van de schatting van de waardering die de beslissing oplevert.

5.1.2 Exploratie met onzekerheid

Als er geen 1-1-relatie is tussen staat en visie of de overgang naar een nieuwe staat is slechts probabilistisch afhankelijk van de genomen beslissing, dan zijn er meer parameters die we moeten leren. We bekijken alleen het geval waar de overgangen probabilistisch zijn: gegeven een staat s_i en een beslissing d, dan is de waarschijnlijkheid om over te gaan naar staat s_j gelijk aan $T(d)_{i,j}$. Dit heeft als gevolg dat we, zelfs als we het hele systeem kennen, niet juist kunnen voorspellen wat de winst van een strategie π zal zijn als we vanuit een bepaald staat s vertrekken. Wel kunnen we een *verwachtingswaarde* berekenen:

$$w_\pi(s_i) = b(s_i) + \sum_j T(\pi(s_i))_{ij} w_\pi(s_j). \tag{5.4}$$

Maar een lerende eenheid kent het systeem niet, het is dit aan het exploreren. Het moet nu niet alleen een schatting maken van de waarde van alle staten, maar ook van de overgangswaarschijnlijkheden $T(d)_{ij}$. Dit doet ze door de frequenties bij te houden van de staten die ze passeert. Als A_{ij} het totaal aantal keer is dat de eenheid in staat s_j geraakt is vanuit staat s_i door d te gebruiken, dan gebruikt ze de schatting

$$T(d)_{ij} \sim \frac{A_{ij}}{\sum_{k=1}^{n} A_{ik}},$$

waarbij, zoals vroeger, n het totale aantal staten is. Op basis van deze schattingen kan men dus weer een optimale strategie π bepalen die de uitdrukking (5.4) voor alle staten zo groot mogelijk maakt.

5.2 COMPLEXE SYSTEMEN

Bij de eenvoudige systemen die we tot nu toe gezien hebben hielden we informatie bij over elke staat. In het geval van systemen met onzekerheid moesten we zelfs voor elke combinatie (s, d) van een staat met een beslissing de overgangswaarschijnlijkheid naar elke andere staat bijhouden. Dit is alleen haalbaar als we maar een beperkt aantal staten hebben. Voor grotere systemen is het zelfs niet mogelijk een strategie, die voor elke visie een overeenkomstige actie[3] voorziet, op te slaan in een associatieve container die voor elke staat een aparte entry heeft. Wat we wel kunnen doen is een *actieve regelbank* invoeren. Deze bevat niet, zoals een gewone regelbanken afleidingsregels die uitspraken afleiden uit andere uitspraken, maar wel regels van de vorm

[3] Het is in deze context gebruikelijk om van acties te spreken in plaats van van beslissingen.

als premisse
dan voer actie uit.

waarbij de premisse een uitspraak is die, toegepast op een visie, waar of vals kan zijn. Het gebruik van zulke actieve regelbank is meestal zeer eenvoudig, omdat er in dit geval geen sprake kan zijn van een opeenvolging van gebruikte regels in een bepaalde situatie. Een actie kan geen deel uitmaken van de premisse van een andere regel: een premisse is immers een lijst van uitspraken, en acties zijn geen uitspraken. Natuurlijk is het mogelijk dat een regelbank twee soorten regels bevat: een regel van de eerste soort heeft dan uitspraken in zijn conclusie, de tweede soort heeft een actie als conclusie. Dergelijke regelbank heet een *hybridische* regelbank. We hebben reeds voorbeelden gezien van zulke hybridische systemen. Bepaalde expertsystemen kunnen verdere onderzoeken voorstellen, waarmee gegevens verzameld worden die tot een diagnose kunnen leiden. Bij medische expertsystemen is een diagnose maar een tussenstap om een behandeling van de patiënt voor te stellen: zulke behandeling kan ook beschouwd worden als een actie.

In de rest van dit hoofdstuk zullen we alleen zuiver actiegerichte regelbanken bekijken. Dergelijke regelbanken zijn zeer geschikt om te leren. Eén zulke leermethode maakt gebruik van *genetische algoritmen*. Bij een genetisch algoritme maakt men gebruik van begrippen ontleend aan de evolutietheorie. Bij natuurlijke evolutie heeft men een verzameling individuen van een bepaalde soort, en door bepaalde mechanismen evolueert deze, over verschillende generaties, naar een verzameling individuen die beter zijn aangepast aan hun omgeving. Deze idee kan men toepassen op een zeer diverse groep van problemen. Omdat we dit echter specifiek zullen uitwerken met het voorbeeld van een actieve regelbank, zullen we ons hoertoe beperken.

De verzameling van alle mogelijke visies kan dus zeer groot worden, en voor elke mogelijke visie moet er een actie voorzien worden. In het algemeen is het niet wenselijk om met elke mogelijke visie een regel te laten overeenkomen. Als bijvoorbeeld een visie van een bepaalde wereld bestaat uit vijf variabelen die elk honderd verschillende waarde kunnen aannemen, zouden we tien miljard regels moeten hebben. De normale gang van zaken is dat de premisse van een regel de visie maar gedeeltelijk beschrijft. Vaak gebeurt dit in de vorm van *wildcards*: bepaalde variabelen mogen een willekeurige waarde aannemen. Deze variabelen worden dan eenvoudigweg niet vermeld in de premisse. Uiteraard is het zo dat de waarde van een numerieke variabele niet altijd exact wordt opgegeven: soms wordt er met groter-dan- en kleiner-danoperatoren gewerkt.

Om de regelbank zo klein mogelijk te houden verdient het dan aanbeveling om regels zo algemeen mogelijk te nemen. Dit kan natuurlijk leiden tot conflicten. Stel dat we twee regels hebben:

als $a > 3$ en $a \leq 5$
dan voer actie 1 uit.

als $b == 2$
dan voer actie 2 uit.

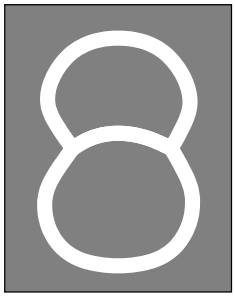
Geldt in een visie $a == 5$ en $b == 2$, welke actie moeten we dan uitvoeren? Het is niet wenselijk te werken met meer gedetailleerde premissen (type: $a > 3$ en $a \leq 5$ en $b \neq 2$). Dit zou het onderhoud van de regelbank ten zeerste bemoeilijken, en bovendien is het mogelijk dat de regelbank dan zeer veel regels zou krijgen.

Het probleem wordt opgelost door regels een verschillende prioriteit te geven. Merk op dat deze prioriteit een andere interpretatie heeft dan de zekerheidsfactoren uit het voorgaande hoofdstuk. Daar ging het erom te werken in een omgeving met onzekerheden, hier gaat het erom om conflicten tussen regels op te lossen. Uitgaande van haar visie op de wereld zoekt de regelbank de regels die overeenkomen met deze visie, en kiest daaruit de

regel met de hoogste prioriteit. Als er verschillende regels zijn met dezelfde prioriteit, kan het probleem op verschillende manieren worden opgelost: de regelbank kan de eerste regel nemen die ze tegenkwam, of de laatste, enzovoorts. Een goed idee is om dan de regel te nemen die het *minst* algemeen is. Immers, als een regel zeer specifiek is behandelt hij waarschijnlijk een speciaal geval dat door algemene regels over het hoofd wordt gezien. Nemen we als voorbeeld de regels

als $a < 10$
dan voer actie 1 uit.

als $a == 5$
dan voer actie 2 uit.



Figuur 5.2. Een parcours waar variatie nodig is

Als we de altijd de meest algemene regel zouden nemen, dan wordt de tweede regel nooit uitgevoerd. De tweede regel heeft dus alleen zin als hij een uitzonderingsgeval aangeeft.

In sommige gevallen is het nodig dat er enige variatie in het gedrag van het systeem zit, bijvoorbeeld als het systeem een speler is in een spel met verschillende spelers. Het gedrag mag dan niet al te voorspelbaar zijn, omdat andere spelers dan gebruik zouden kunnen maken van deze voorspelbaarheid. In dat geval wordt een mechanisme ingebouwd waardoor het systeem niet altijd de regel kiest met de hoogste prioriteit. Ook stofzuig- en grasmaairobots moeten variaties vertonen in hun gedrag, om te voorkomen dat ze altijd dezelfde plaatsen aandoen en andere overslaan. Als voorbeeld kunnen we Figuur 5.2 nemen. Een robot die dit parcours moet schoonhouden, mag aan een Y-punt niet altijd dezelfde kant kiezen.

5.3 GENETISCHE ALGORITMEN

Voor het oplossen van bepaalde problemen maakt men gebruik van een simulatie van natuurlijke evolutie. We illustreren deze techniek aan de hand van actieve regelbanken. We hadden een strategie gedefinieerd als een functie die met elke visie een actie associeert. Als een actieve regelbank een strategie moet kunnen weergeven moet ze een geheel van regels bevatten dat voor *elke* mogelijke visie een actie teruggeeft. Voor de eenvoud noemen we dergelijke regelbank ook een strategie.

Als we een willekeurige verzameling regels nemen, dan kunnen we deze altijd aanvullen tot een strategie, door het invoeren van één nieuwe regel. Deze geeft aan wat er moet gebeuren als er geen andere regel van toepassing is. Formeel gezien heeft deze nieuwe als premisse 'waar', en hij krijgt de laagste prioriteit. Degelijke regel heet de defaultregel.

We zoeken nu een methode waarbij de regelbank leert om een goede strategie te vinden. Een mogelijke denkrichting zou hier zijn om te beginnen met een willekeurig aangemaakte strategie, en deze dan te verbeteren door individuele regels toe te voegen, te verwijderen of te wijzigen. Dit is ongeveer de manier waarop een mens zou proberen het probleem op te lossen. We komen later nog terug op deze methode. Een genetisch algoritme gaat heel anders te werk. Hier werken we met een grote groep van strategieën, de zogenaamde *populatie*. We pogen nu de populatie zo te wijzigen, door strategieën te verwijderen of toe te voegen, dat er op den duur een strategie inzit die efficiënt is. Hiertoe werken we met verschillende *generaties*. Elke generatie bestaat uit een populatie die even groot is, bijvoorbeeld honderd strategieën. Om de volgende generatie te bekomen hebben we twee stappen nodig:

1. In de eerste stap dunnen we de populatie uit door een aantal strategieën te verwijderen. We houden alleen de beste strategieën over. In de woorden van Darwin heet dit *the survival of the fittest*.
2. Ons baserend op de overblijvende strategieën vullen we de populatie terug aan tot haar oorspronkelijke grootte.

Bij de eerste stap is het essentieel dat we een goede methode hebben om het onderscheid te maken tussen goede en slechte strategieën. Immers, als we de overblijvende strategieën lukraak uitkiezen kunnen we niet hopen om een verbetering te krijgen in opeenvolgende generaties. In de meeste klassieke systemen werkt men met een *geschiktheidsfunctie*. Deze functie hecht aan iedere mogelijke strategie een getal dat aangeeft hoe goed de strategie wel is. De manier waarop de geschiktheidsfunctie bepaald wordt hangt af van geval tot geval. Als het doel is om een strategie te ontwikkelen om een bordwereld te verkennen, kan men bijvoorbeeld de strategie een aantal problemen laten oplossen en kijken naar het resultaat: hoeveel voedsel vindt de strategie, en hoe snel? Als men een strategie voor een tweespelersspel wil ontwikkelen, kan men ook eenvoudigweg twee strategieën tegen elkaar laten spelen en de verliezer verwijderen uit de populatie.

De tweede stap bestaat uit twee delen: kruising en mutatie. Bij kruising neemt men twee exemplaren uit de populatie, en vermengt deze, zodat er een nakomeling ontstaat die eigenschappen heeft van de beide ouders. De juiste vorm die kruising aanneemt hangt weer af van probleem tot probleem. Als we veronderstellen dat elke strategie in de populatie evenveel regels heeft, zouden we de helft van de regels van de eerste en de helft van de regels van de tweede strategie kunnen samenvoegen tot een nieuwe strategie, die dan het kind van de twee eerste strategieën is. Merk op dat het niet interessant is om een strategie te hebben waarbij de premissen van regels elkaar al te veel overlappen. Stel dat we twee regels hebben,

als $a > 5$ en $a < 10$
dan actie 1 (prioriteit 30)

als $a > 4$ en $a < 20$
dan actie 2 (prioriteit 40),

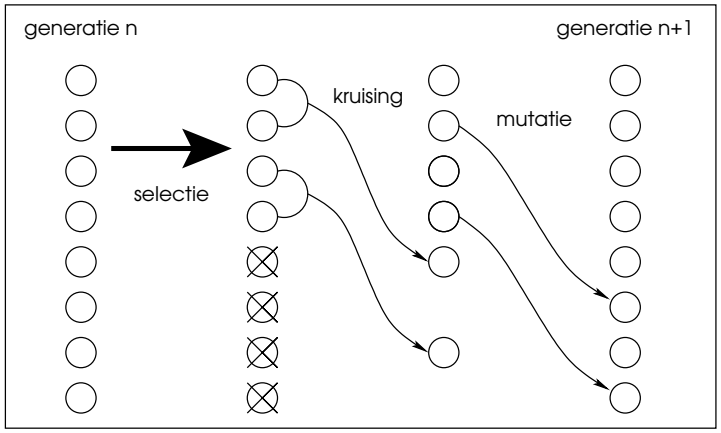
dan zal de eerste regel nooit gebruikt worden. Immers als zijn premisse waar is, dan is deze van de tweede regel ook waar, en de tweede regel heeft een hogere prioriteit. Een realistisch schema in dit geval zou er zo kunnen uitzien:

- Bepaal een maat voor de overlapping van twee regels. Als het aantal mogelijke visies eindig is, kunnen we gewoon het aantal mogelijke visies nemen waarin de premissen van beide regels waar zijn.
- Om twee strategieën te kruisen, doe nu het volgende, zolang er nog regels in de twee strategieën overblijven:
    1. Zoek de regel in strategie A en de regel in strategie B zodanig dat de overlapping tussen deze twee regels de grootste is van elke mogelijke combinatie van twee regels.
    2. Verwijder beide regels uit A en B, en voeg een ervan toe aan de kindstrategie.

Merk op dat deze werkwijze garandeert dat een van de defaultregels in het kind terechtkomt. Immers, de defaultregel uit A en deze uit B hebben de grootst mogelijke overlapping: voor elke mogelijke visie is de premisse van beide regels waar.

Bij mutatie neemt men een strategie uit de populatie, en voert daarin een kleine wijziging door. Hiervoor neemt men één enkele regel en wijzigt deze. Men kan bijvoorbeeld

de actie uit de conclusie vervangen door een andere actie, of in de premisse een uitspraak wijzigen of verwijderen, of een willekeurig gegenereerde uitspraak toevoegen, of de prioriteit aanpassen. Er zijn uiteraard nogal wat variaties mogelijk. In sommige varianten gooit



Figuur 5.3. Evolutie in één generatie.

men de ouders na kruising terug in de populatie, in andere is dat niet het geval (strategieën kunnen dan wel ouder zijn van meerdere kinderen, anders zou de populatie kleiner worden). Ook bij mutatie kan men ervoor kiezen de oorspronkelijke strategie te verwijderen of te behouden. In elk geval wordt ervoor gezorgd dat na kruising en mutatie de populatie terug het oorspronkelijke aantal bereikt.

Tot slot nog een specifieke opmerking over de geschiktheidsfunctie. Genetische algoritmen worden vaak gebruikt bij het zoeken naar een oplossing van ingewikkelde problemen. Vaak is het dan niet alleen moeilijk om een goede oplossing voor het probleem te vinden, maar is het moeilijk om een oplossing te vinden die aan alle eisen voldoet. De geschiktheidsfunctie moet er dan voor zorgen dat er ook een onderscheid kan gemaakt worden tussen verschillende niet-oplossingen. We maken dit duidelijk aan de hand van een voorbeeldje dat geen gebruik maakt van een actieve regelbank: het uurroosterprobleem.

Bij dit probleem heeft men een aantal studentengroepen, die elk een aantal cursussen volgen die gegeven worden door bepaalde docenten. Verder zijn er klaslokalen. Men moet nu een uurrooster opmaken dat aan een aantal eisen voldoet. Er zijn twee soorten eisen:

1. Harde eisen. Elke cursus moet gegeven worden, en uiteraard kan een docent geen twee lessen tegelijkertijd geven en kan een studentengroep geen twee lessen tegelijk volgen. Verder moet elke les doorgaan in een lokaal dat geschikt is voor de les (het lokaal moet bijvoorbeeld groot genoeg zijn voor de groep studenten, een les kan maar doorgaan in een lokaal met geschikte uitrusting, en zo verder), en kunnen er geen twee lessen tegelijk doorgaan in hetzelfde lokaal.
2. Zachte eisen. Een uurrooster kan beter zijn dan een andere die ook aan alle harde eisen voldoet. Criteria kunnen zijn de afwezigheid van springuren voor studentengroepen of docenten, het beperken van verandering van lokalen, en zo verder.

Als we gebruik maken van een genetisch algoritme, zal het zeer moeilijk zijn om een beginpopulatie van uurroosters te maken die aan alle harde eisen voldoet. Bovendien kunnen er, door kruising of mutatie, uurroosters ontstaan die niet aan deze eisen voldoen. Dit probleem lost men op door onmogelijke uurroosters toe te laten, maar deze een slechte geschiktheid te geven. Zo kan een geschiktheid genomen worden die start vanaf nul, en die

voor elke eis wordt aangepast als volgt:

1. Voor elke harde eis die niet voldaan is worden duizend strafpunten afgetrokken. Dit gebeurt als bijvoorbeeld een studentengroep twee verschillende lessen op hetzelfde moment heeft.
2. Voor elke zachte eis die niet voldaan is wordt één punt afgetrokken.

Bij de eerste generaties zullen alle uurroosters nog harde fouten hebben. Geschikte uurroosters hebben dan minder harde fouten dan andere. Naarmate het aantal harde fouten afneemt, worden de zachte eisen meer zichtbaar: er zullen vaak paren uurroosters zijn met evenveel harde fouten, en dan is het rooster met het kleinste aantal zachte fouten beter dan het andere.

5.4 OPTIMALISATIE VAN ÉÉN STRATEGIE

Een geheel andere benadering probeert een verzameling actieregels te optimaliseren tot één goede strategie. Bij kleine systemen gebruikten we daarvoor een waardering voor elke staat apart. De verzameling visies is hier echter te groot en daarom moeten we ons hier beperken tot een waardebepaling voor elke regel. Net zoals vroeger hangt die waarde meestal af van de rest van de strategie. Een gegeven regel heeft wel een waarde op zich als hij altijd rechtstreeks leidt tot een dezelfde eindtoestand. In het algemeen echter kan een verandering in andere regels ook de waarde van de gegeven regel veranderen. Gegeven dus een regel r krijgen we een waardering $w(r)$ die aangeeft wat de te verwachten winst is als we r toepassen, ervan uitgaande dat we voor de rest de gegeven strategie volgen.

5.4.1 Het leerproces

Bij het normaal gebruik van een strategie heeft het geen zin als er regels zijn met overlappende premissen, maar hier is dat wel zo. Immers, de bedoeling van ons leerproces is regels met hoge waarde een hoge prioriteit te geven en regels met een lage waarde een lage prioriteit, of ze zelfs te verwijderen. Hiervoor is het goed als er veel overlappende regels zijn die tegen elkaar kunnen concurreren. Hebben we bijvoorbeeld twee regels met dezelfde premisse maar verschillende acties, dan willen we alleen de regel overhouden met de beste actie. Hebben we twee regels met dezelfde actie, waarbij de premisse van de eerste regel meer algemeen is dan die van de tweede, dan willen we weten of bij visies waarbij de tweede regel niet toepasbaar is de actie efficiënt is: zoja, dan is de eerste regel de beste, zonee de tweede.

Na het leerproces dat we dadelijk gaan bekijken hebben we een regelverzameling, waarbij elke regel een waardering heeft. De strategie die we daaruit afleiden gebruikt de waardering om de prioriteit te bepalen: als verschillende regels toepasbaar zijn op een visie wordt die met de hoogste waardering gekozen.

Het leerproces kent twee fasen, die elkaar afwisselen. In de ene fase worden de waarderingen van de regels aangepast aan de hand van informatie verkregen door exploratie. We noemen ze dan ook de *exploratiefase*. In de andere fase wordt de regelverzameling zelf gewijzigd. We noemen ze de *regelaanpassingsfase*.

In de exploratiefase is de *regelverzameling* (Eng: action set) $A(v)$ van een visie belangrijk. Het is de verzameling regels waarvan de premisse beantwoordt aan v. $A(v)$ is op twee manieren belangrijk:

(1) Als we aan een bepaalde visie v beland zijn moeten we beslissen welke actie we gaan nemen. Hiervoor kiezen we een regel uit $A(v)$. Het is de bedoeling om goede regels te bevoordelen, bijvoorbeeld door een gewogen random keuze met de waarderingen van de regels als gewicht. We voeren de bijbehorende actie a uit.

(2) Het resultaat is een nieuwe visie v'. Om nu een schatting te maken hoe goed het resultaat is moeten we een waarde toekennen aan v'. Als v' een eindstaat is dan kunnen we rechtstreeks zo'n waarde geven, namelijk de beloning die vervat is in de visie. Maar als v' geen eindstaat is dan kunnen we verder gaan en nog beloningen (positief of negatief) verzamelen. Omdat we geen waarderingen hebben voor staten, maar alleen voor regels behelpen we ons met de regelverzameling $A(v')$ en kennen we een waarde $w(v')$ toe met de regel

$$w(v') = b(v') + \max_{r \in A(v')} w(r).$$

Hierin is $b(v')$ de beloning die rechtstreeks in de visie v' zit.
We splitsen nu de verzameling $A(v)$ op in twee delen:

a. Voor een regel r in $A(v)$ die als actie a voorstelt (dat is vaak niet alleen de regel die we gekozen hadden), is $w(v')$ de verwachte winst in deze run. Maar door de onzekerheid die we hebben kunnen we niet zeker zijn dat $w(v')$ altijd de winst is die we mogen verwachten als we r toepassen. We gaan de oude waarde van $w(r)$ niet zomaar weggooien, maar alleen een aanpassing in de richting van $w(s')$ doorvoeren. Hiervoor gebruiken we een vergeetfactor α, $0 < \alpha < 1$, en we stellen

$$w(r) \rightarrow (1 - \alpha)w(r) + \alpha w(v')$$

b. Een regel r in $A(v)$ die een andere actie dan a voorstelt was niet succesvol. We verminderen dus zijn waardering met een vergeetfactor β (die typisch kleiner is dan α). Voor zo'n regel krijgen we

$$w(r) \rightarrow (1 - \beta)w(r).$$

Na enige exploratie is het tijd om de regelverzameling zelf aan te passen. De bedoeling is om regels te verwijderen met een lage waardering, omdat deze toch geen bijdrage leveren aan de uiteindelijke strategie, en deze te vervangen door nieuwe, potentieel waardevolle regels. Vaak gebeurt dit door regels niet te verwijderen, maar ze gewoon te veranderen.

Op basis van de waardering kunnen we beslissen welke regels we veranderen en verwijderen. We kunnen ook bijhouden hoe dikwijls regels *gebruikt* werden in de exploratiefase. Een regel die nauwelijks gebruikt werd kan toch een hoge waardering krijgen omdat er overlappende regels gebruikt werden die dezelfde actie voorstelden. Toch is de regel waarschijnlijk niet nuttig, juist omdat die overlappende regels er ook zijn.

Er zijn veel manieren om een slechte regel te veranderen in een regel die hopelijk beter is. Zo kunnen we bijvoorbeeld de premisse behouden en de actie vervangen door een andere.

Men kan ook de *variantie* van de opbrengst van regels gebruiken. Als een regel wel een goed gemiddelde haalt, maar meestal of heel goed of heel slecht presteert dan kan men de regel *splitsen* door de voorwaarde in de premisse op vrij willekeurige wijze over twee regels te verdelen. Hebben we bijvoorbeeld een regel

als $3 \leq x \leq 5$
dan Actie A

dan nemen we random een andere variabele (bijvoorbeeld y) uit de visie met een willekeurige grenswaarde (10, in ons voorbeeld) en splitsen de regel in

als $3 \leq x \leq 5$ en $y < 10$
dan Actie A

als $3 \leq x \leq 5$ en $y \geq 10$

dan Actie A

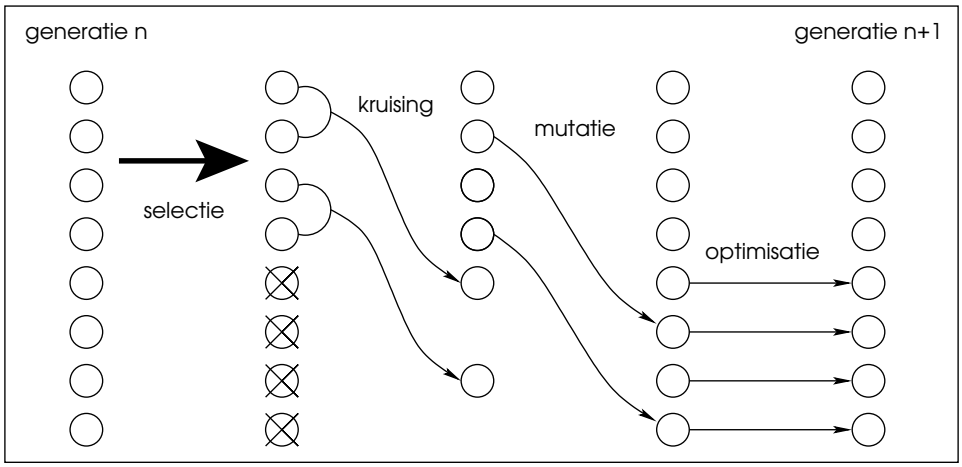
Op deze manier kan men proberen het slechte en het goede deel van de regel van elkaar te scheiden.

Men kan ook, in plaats van te splitsen, twee regels *samenvoegen*. Dit kan nuttig zijn als de twee regels dezelfde actie voorstellen, en grote overlapping hebben. Ook kan men een goed presterende regel uitbreiden door zijn premisse te verruimen. Samenvoegen en verruimen doet men bij regels die succesvol zijn: de oorspronkelijke regels blijven dan ook meestal behouden.

5.5 COMBINATIES

In een voorgaande paragraaf hebben we bekeken hoe we de klassieke metaheuristiek van het genetische algoritme kunnen toepassen op strategieën. Men mag echter verwachten dat deze methode zeer traag zal zijn. Dat heeft twee redenen. Ten eerste is de evaluatie van een strategie zeer tijdrovend. Immers, deze gebeurt meestal door te zien hoe een strategie werkt, dus niet door een simpele berekening maar door een simulatie van de omgeving. Vermits we werken met situaties met onzekerheid moet die simulatie herhaalde malen worden uitgevoerd. Ten tweede is het zo dat één verkeerde regel de efficiëntie van een strategie volledig teniet kan doen. Vermits een genetisch algoritme vrij blindelings zoekt, kan het dus lang duren voor juist die regel wordt aangepast of verwijderd.

Daarom wordt er vaak gewerkt met een aangepast schema, waarbij voor elke nieuwe strategie in een generatie een individuele optimalisatie wordt uitgevoerd. Dit schema wordt vaak Lamarckiaanse[4] evolutie genoemd.



Figuur 5.4. Lamarckiaanse evolutie.

De optimalisatie zelf verloopt op een manier gelijkaardig aan de methode beschreven in de vorige paragraaf, alleen gaan we hier uit van een strategie die verkregen is door mutatie of kruising en niet van een randomstrategie. Hierbij kunnen we opmerken dat, als we een strategie die al eens geoptimaliseerd is geweest nemen, deze gaan muteren met een kleine

[4] Jean-Baptiste Pierre Antoine de Monet, Chevalier de Lamarck was de eerste, zowat een halve eeuw voor Wallace en Darwin, die een evolutietheorie ontwikkelde. Volgens hem gaf een individu eigenschappen die het verwierf tijdens zijn leven door aan zijn nakomelingen.

wijziging en daarna terug optimalizeren, er veel kans is dat we weer dezelfde strategie krijgen.

Ten slotte dient nog opgemerkt dat, ook binnen het genetisch algoritme, men meestal niet uitgaat van compleet random mutaties en kruisingen. Men gebruikt dan methodes zoals beschreven in de vorige paragraaf om de regelverzameling te wijzigen.

HOOFDSTUK 6
NEURALE NETTEN

De cursussen in de opleiding Informatica handelen voornamelijk over een bepaald soort informatieverwerkende eenheid: de klassieke Von Neumannarchitectuur. Neurale netwerken zijn een heel andere vorm van informatieverwerkende eenheden. Klassiek zijn er natuurlijk biologische neurale netten: de zenuwstelsels, inclusief eventuele hersenen van mensen en dieren. Daarnaast zijn er ook nog kunstmatige NN'en (NN = neuraal netwerk), die ruwweg op dezelfde principes gebaseerd zijn als biologische. We beginnen met een beknopte definitie van een neuraal netwerk:

Een neuraal netwerk is een informatieverwerkend geheel dat de vorm heeft van een **gewogen gerichte graaf***. De knopen van deze graaf heten* **neuronen***. Ze sturen een signaal naar alle verdere knopen in een graaf met een sterkte die afhangt van de som van de invoersignalen. De gewichten van de takken van de graaf geven aan hoeveel het signaal dat langs die tak loopt versterkt wordt.*

De gewichten van de graaf kunnen positief of negatief zijn. Ze kunnen ook veranderen, zodat een neuraal netwerk verschillende taken kan vervullen. De topologie van de graaf (welke neuronen zijn verbonden met welke) is onveranderlijk, evenals de manier waarop elk neuron de sterkte van zijn uitvoersignaal laat afhangen van zijn invoersignaal.

Zoals we al gezien hebben is een kunstmatig neuraal netwerk een voorbeeld van een *massief lerend systeem*. Als we een NN simuleren gaat het over een relatief beperkt aantal verbindingen (hoogstens enkele honderdduizenden, meestal veel minder), bij biologische netten gaat het over veel meer. Het is niet zo dat we aan elk verbindingsgewicht een specifieke functie geven: door te leren worden de gewichten zo veranderd dat het netwerk zijn taak aankan.

Het belang van NN'en voor de informatica ligt voornamelijk in twee gebieden. Eén ervan is de communicatie met biologische NN'en, het tweede is dat van leerprocessen voor taken waarbij geen exact algoritme gegeven is.

De klassieke communicatie tussen de computer en zijn omgeving is volledig bepaald door de manier waarop computers werken: ze verloopt in een vorm waarbij kleine, exact omschreven berichten worden uitgewisseld. Een druk op de toets van een klavier, een muisklik, een tekst op een scherm bestaande uit letters van bepaalde grootte en vorm zijn voorbeelden van zulke berichten. NN'en zijn echter ingericht om informatie te verwerken die vaag, gevarieerd en ingewikkeld is. Stilaan verwacht men dat een computer ook dergelijke informatie kan verwerken. Spraaktechnologie is daar een voorbeeld van: de communicatie wordt aangepast aan de vereisten van de mens, in plaats van aan de machine. Maar ook OCR, Optical Character Recognition, hoort hierbij: een tekst wordt aangeboden in de vorm van een afbeelding waar de vorm en grootte van letters niet exact gekend zijn door de machine. Ook wordt meer en meer gebruik gemaakt van een computer om allerhande afbeeldingen te analyseren (bijvoorbeeld beelden gemaakt door een bewakingscamera). Zelfs wordt er reeds een begin gemaakt met rechtstreekse communicatie tussen computer en (menselijk zenuwstelsel) d.m.v. elektroden. Zo bestaan er kunstmatige oren die aangesloten worden op de gehoorzenuwen, en zijn er (nog zeer primitieve) kunstmatige

ogen die lichtprikkels simuleren in de oogzenuw. Men kan verwachten dat deze technieken steeds meer in belang zullen toenemen.

Wat leren betreft is een NN ten zeerste geschikt om een computer taken te laten vervullen waarvoor men geen algoritme kent. Zonder algoritme kan men dus geen programma schrijven, en de computer moet zelf leren hoe de taak te vervullen. De verwerking van invoer van een computer in situaties zoals hierboven is hiervan een typisch voorbeeld. Maar ook bij expertsystemen die beslissingen nemen aan de hand van vrij vage invoer is dit het geval. Een typisch voorbeeld wordt gegeven door spelcomputers of -programma's die bijvoorbeeld schaak spelen. Men heeft hier wel een goed zicht op wat een goede zet is (een goede zet zorgt voor winst), maar men weet niet direct hoe men tot een goede zet komt. We gaan hier in deze cursus nog verder op in.

6.1 VERGELIJKING MET COMPUTERS

Een traditionele computer is opgebouwd uit twee delen: een centrale verwerkingseenheid en een geheugen (dat verder contact kan hebben met de buitenwereld). De verwerkingseenheid kan een aantal welbepaalde opdrachten uitvoeren. Als een computer een taak moet uitvoeren laadt men een *programma* met gedetailleerde instructies in het geheugen. Dit programma vertelt de computer welke gegevens hij van buitenuit moet ophalen, en wat hij ermee moet doen (inclusief het naar buiten brengen van resultaten).

Centraal staat hier het begrip *algoritme*: een exacte beschrijving van een procedure die de reacties van de computer op zijn omgeving bepaalt. We kunnen hier als voorbeeld een compiler voor een bepaalde taal nemen. er is een taaldefinitie die exact aangeeft hoe een bronprogramma moet vertaald worden naar uitvoerbare code. De invoer is hier een reeks symbolen (het bronprogramma). Voor een mens (althans voor een programmeur) geven deze symbolen *ideeën* of *concepten* weer. Voor de computer, dankzij het compilerprogramma, is het mogelijk deze symbolen te verwerken tot een uitvoerbaar programma. Het is volkomen irrelevant voor de computer waar dit programma voor dient (het kan een programma zijn voor een ander type computer, dat onze computer niet kan uitvoeren). Essentiële karakteristieken van deze manier van werken zijn:

1. Men moet de machine tot op het laatste detail vertellen wat het algoritme is om de invoer te verwerken. Dit houdt automatisch in dat er ergens iemand moet geweest zijn die dit algoritme kende. Dit gebeurt *vooraleer* er informatie verwerkt wordt.
2. De computer is zeer gevoelig voor onverwachte invoer: hij kan alleen een fout in de invoer herstellen als in het algoritme deze fout voorzien is.
3. De kleinste fout in de apparatuur of in de programmatuur kan ertoe leiden dat de uitvoer zelfs niet meer lijkt op wat de bedoeling was: kleine onregelmatigheden in het systeem kunnen het volledig onbruikbaar maken.
4. Elk begrip en object van de verwerking (getallen, strings) is duidelijk *localiseerbaar* in de hardware. Zo is er bijvoorbeeld ergens in het geheugen een lijst van de variabelen die voorkomen in het te compileren programma.

Zoals we zullen zien gaan eigenschappen (3) en (4) in het geheel niet op voor biologische neurale netten. Maar eigenschappen (1) en (2) zijn de belangrijkste obstakels voor het klassieke gebruik van de computer voor bepaalde taken. Het eerste punt is hierbij het belangrijkste. Voor sommige taken die we zonder problemen uitvoeren kennen we het algoritme niet, en het blijkt dat het bijzonder moeilijk is om er een op te stellen. Nemen we als voorbeeld lezen. We hebben dit allemaal geleerd, zonder dat ons expliciet een algoritme hebben gekregen om dit te doen. We hebben er nauwelijks een idee van op basis van wat we beslissen dat een bepaalde krabbel dit of dat woord weergeeft, laat staan dat we dit

zomaar in een computerprogramma kunnen gieten. We zijn zelfs in staat handschrift te lezen, terwijl elke handgeschreven letter verschillend is (punt (2)). Het is belangrijk er aan te denken dat we dit wel degelijk *geleerd* hebben: ons neuraal netwerk begint met een toestand waarin het niet kan lezen, en is zelfs niet speciaal geconstrueerd om te kunnen lezen: lezen is een vaardigheid die hoogstens enkele duizenden jaren bestaat, en heeft dus zeker nauwelijks invloed kunnen hebben op de evolutie van de mens.

Aan de hand van het leren van mensen en dieren kunnen we naast de eigenschappen van een computer dus de volgende eigenschappen plaatsen:

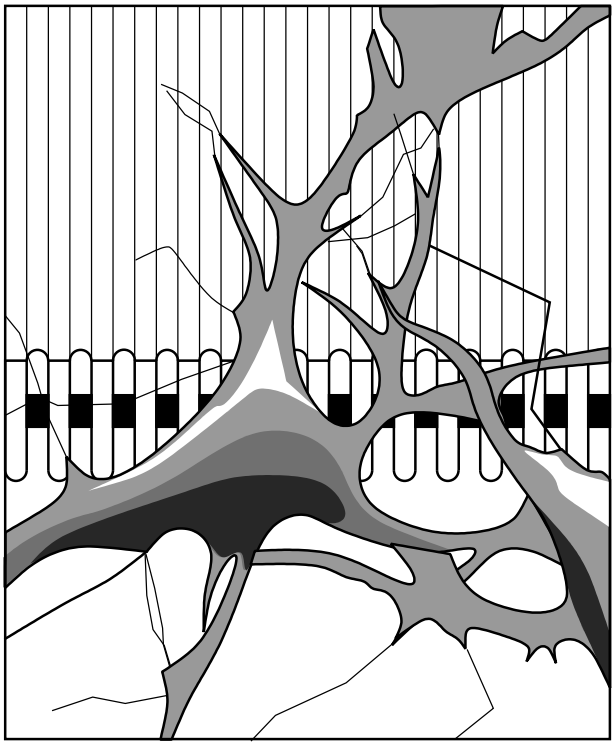
1. Een neuraal netwerk wordt niet geprogrammeerd maar *leert*. Dit leren gebeurt door voorbeelden en associaties, zonder dat een eventueel algoritme wordt gegeven. Leren kan gebeuren voordat informatie verwerkt wordt, maar ook zonder expliciete leerfase: *terwijl* het neuraal netwerk informatie verwerkt leert het bij.
2. Een neuraal netwerk is zeer goed in veralgemenen: het kan invoer herkennen die lijkt op al geziene invoer.
3. Een neuraal netwerk is zeer ongevoelig voor beschadigingen: een gedeeltelijk vernietigd neuraal netwerk kan meestal zijn taak nog vrij goed aan.
4. Begrippen en objecten zijn niet exact te lokaliseren. In de menselijke hersenen is er wel een spraakcentrum, een auditief centrum, enzovoorts, maar het is onmogelijk om (bijvoorbeeld) het woord ‘oor’ uit de hersenen te snijden zonder heel wat andere informatie mee te hebben. Dit houdt verband met het massief leren met parameters zonder bepaalde betekenis: het zijn *combinaties* van parameterwaarden die een betekenis opleveren.

Er zijn twee manieren in gebruik om kunstmatige neurale netwerken te maken. De eerste is hardewarematig. Men construeert (veelal elektronische) componenten die dezelfde functies hebben als een natuurlijk neuron. De gewichten van de takken veranderen op een manier die overeenkomt met het biologische voorbeeld. Deze methode was vrij populair in de beginfase van het onderzoek naar NN’en, de jaren ’50 van de vorige eeuw, toen er van digitale computers nog nauwelijks sprake was. Het eerste kunstmatige net bestond was SNARC van Minsky en Edmonds uit 1951. Het simuleerde een net met 40 neuronen en bestond o.a. uit 3000 vacuümbuizen en het automatischepilootmechanisme van een B-24 bommenwerper.

De tweede methode is softwarematig: men simuleert de werking van een NN met een programma. Deze methode heeft een aantal voordelen: men kan de computers gebruiken die er toch al zijn, en het netwerk kan vlot communiceren met andere programma’s. Men moet zich echter op deze manier tot zeer eenvoudige en kleine netwerken beperken (met een aantal knopen en takken dat enkele duizenden of honderdduizenden kan bedragen), zodat het best mogelijk is dat hardwarenetwerken terug hun intrede doen. In elk geval zijn ook de taken die de huidige NN’en vervullen vrij eenvoudig. Meestal beperken ze zich tot een *classificatieprobleem* met een beperkt aantal klassen, zoals OCR of spraakherkenning.

Recent is er hernieuwde belangstelling voor hardwarenetwerken. Door het gebruik van technieken die ook in computers gebruikt worden (VLSI-chips) is het mogelijk vrij grote neurale netwerken op te bouwen. Het ontwikkelen van toepassingen voor deze netten staat echter nog in de kinderschoenen.

Bijkomend dienen ook nog de zgn. neurochips vermeld. Hierbij laat men neuronen groeien bovenop een chip, zodat verbindingen van de neuronen met de banen op de chip ontstaan. Deze techniek is nog volledig experimenteel.



Figuur 6.1. Neurochip

6.2 DE BIOLOGISCHE GRONDSLAGEN

De term neuraal netwerk wordt, in de computerwetenschappen, gebruikt voor programma's met een structuur die overeenkomt met biologische neurale netwerken, zoals het zenuwstelsel (inclusief de hersenen) van hogere diersoorten. In dit hoofdstuk bekijken we deze biologische netwerken voor zover deze voor ons interessant zijn. We gaan niet in op de biochemische processen die de grondslag vormen van de werking van neuronen, noch op functionele verschillen tussen verschillende neuronen. Wel houden we ons bezig met de werking van 'typische' neuronen, en bekijken we in het kort hoe de invoer en uitvoer van een neuraal netwerk gebeurt.

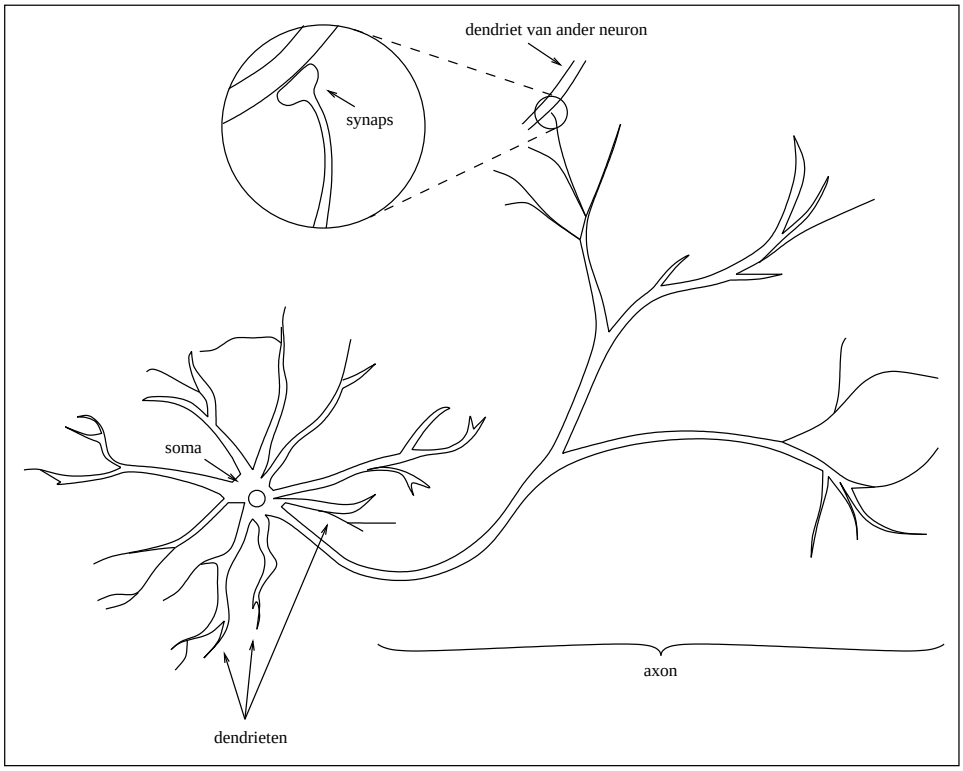
6.2.1 Neuronen

Het zenuwstelsel bestaat uit zenuwcellen. Soms wordt de term zenuwcel gebruikt als synoniem van neuron. Er zijn drie soorten zenuwcellen. De indeling in drie soorten gebeurt naargelang de functie:

1. Waarnemingscellen (invoerneuronen): deze zorgen voor de omzetting van fenomenen uit de buitenwereld (en de buitenwereld is alles wat niet het neuraal netwerk is) in signalen die verwerkt kunnen worden door andere zenuwcellen. We spreken ook wel van invoerneuronen. Typische voorbeelden zijn de lichtreceptoren in de ogen, smaakpapillen enzovoorts,
2. Gewone neuronen, ook interneuronen genoemd: deze verwerken informatie. Ze zijn enkel verbonden met andere zenuwcellen.

3. Uitvoercellen: deze geven informatie door aan de buitenwereld. De meest bekende vorm van uitvoer is deze waarbij een zenuwprikkel een spier doet samentrekken. Een uitvoerzenuw kan echter ook andere processen op gang trekken of onderbreken. Deze cellen noemen we ook uitvoerneuronen.

Wat de structuur van zenuwcellen betreft kijken we eerst naar de gewone neuronen. Het centrale deel heet soma (van het Grieks $\sigma\tilde{\omega}\mu\alpha$, lichaam). Vanuit de soma vertrekken een of meer dendrieten (Grieks: $\delta\acute{\epsilon}\nu\delta\rho o\nu$, boom). Zoals de naam aangeeft is een dendriet een lange sliert die kan vertakken. De dendrieten doen dienst als receptoren van signalen van andere neuronen. Uit de soma vertrekt ook nog het axon (Grieks: $\acute{\alpha}\xi\omega\nu$, spil). Ook dit



Figuur 6.2. Schema van een neuron.

axon vertakt zich, maar veel minder opvallend dan de dendrieten. Dit axon kan zeer lang zijn, tot wel een meter lengte (bij grotere dieren dan de mens nog meer). Het is langs dit axon dat het neuron zijn signalen uitzendt. Elk van de vertakkingen eindigt in een synaps (Grieks $\sigma\upsilon\nu\acute{\alpha}\pi\tau\omega$: ik verbind). Deze zit vast op een dendriet van een ander neuron. Bij een typisch menselijk neuron kan het aantal synapsen gaan van een paar honderd tot meer dan honderdduizend; het gemiddelde blijkt ruim 3000 te zijn.

Een synaps is een verbindingspunt tussen twee zenuwcellen, en heeft dus aan de ene kant een axon en aan de andere kant een dendriet (van een andere zenuwcel). Een signaal dat wordt uitgestuurd door de soma van een zenuwcel bereikt via haar axon de synapsen, en wordt door de dendrieten aan de andere kant doorgestuurd naar de soma's van de bijbehorende zenuwcellen. Meestal zijn twee zenuwcellen door hoogstens enkele synapsen verbonden: een typisch neuron krijgt dus invoer van een aantal zenuwcellen dat ongeveer gelijk is aan het aantal dendritische vertakkingen van het neuron, dat vrij hoog kan liggen.

Bij waarnemingscellen ontbreken de dendrieten. De vorm van een waarnemingscel hangt af van de prikkel waarvoor ze gevoelig is, maar we kunnen hier ook spreken van een soma. Deze zendt signalen door als ze geprikkeld wordt op de manier waar ze gevoelig voor is. Zo zal een cel in het oog signalen uitzenden als er licht op valt, zal een smaakpapil signalen doorsturen als ze bepaalde stoffen waarneemt, en zo verder. Voor de eenvoud van ons model veronderstellen we bij een waarnemingscel alle dendrieten vervangen zijn door een enkele sensor.

De uitvoercellen geven hun signalen door, niet aan andere zenuwcellen, maar aan de buitenwereld. In de plaats van het axon met zijn vertakkingen is er dus een verbinding die specifiek is voor de functie van de zenuwcel. Sommige uitvoercellen prikkelen spieren zodat deze samentrekken; andere dan weer zetten andere processen in werking (bijvoorbeeld deze die de speekselklieren prikkelen).

Weer gaan we er voor de eenvoud van uit dat er een enkele uitvoer per uitvoercel bestaat.

Zenuwcellen sturen signalen door in de vorm van pulsen. Elk neuron (en elke waarnemingscel) kan op elk gegeven moment besluiten om een puls af te vuren. Elke puls van een gegeven neuron is gelijk qua intensiteit en tijdsduur. Dit heeft geleid tot de misvatting dat de uitvoer van een neuron essentieel binair is, vermits op een gegeven moment een neuron ofwel in rust is, ofwel een puls afvuurt. Het is dus nodig om uitdrukkelijk te stellen dat *de uitvoer van een neuron een analoog signaal is*. De sterkte van dit signaal wordt niet bepaald door de sterkte van de pulsen (die niet varieert) maar door de tijdsduur die er verloopt tussen verschillende pulsen.

De puls is een elektrische stroomstoot. Deze loopt door het gehele axon en komt aan bij alle synapsen van dit axon. Er zijn geen onderverdelingen in het axon, en het neuron kan dus niet beslissen een impuls naar een gedeelte van zijn uitgangen te sturen: alle neuronen waarvan een dendriet verbonden is met het axon van het afvurende neuron ontvangen dus de puls.

De puls wordt op chemische wijze voorbij de synaps geleid. Sommige synapsen laten beter pulsen door dan andere, en op langere termijn kan de doorlaatbaarheid van een synaps veranderen. Dit zal het neurale netwerk in staat stellen om te leren, waarover later meer. Langs de dendriet wordt de puls dan naar de soma gevoerd. Het hele proces dat begint met het vertrek van de puls in de zendende soma, en eindigt met de aankomst van de puls in de ontvangende soma duurt ongeveer 1 tot 2 milliseconden (ter vergelijking: dit zijn 3 tot 6 miljoen klokpulsen van een 3GHz-processor). Op basis van wat vuurt een neuron nu een puls af? Bekijken we eerst het geval van een neuron in rust. Vermits de fysische achtergrond hier niet ter zake doet, kunnen we de soma opvatten als een soort reservoir (of een condensator, voor wie in termen van elektriciteit wil denken). Naargelang het neuron waaruit de puls vertrekt zijn er twee mogelijkheden:

1. een *excitatorische* of prikkelende puls vult dit reservoir bij met een hoeveelheid die evenredig is met zijn sterkte (welke op haar beurt bepaald wordt door de doorlaatbaarheid van de synaps waardoor hij passeert);
2. een *inhibitorische* of remmende puls vermindert de inhoud van het reservoir met een hoeveelheid die evenredig is met zijn sterkte.

Of een signaal excitatorisch dan wel inhibitorisch is hangt af van het soort synaps. Tenslotte is het reservoir lek (in elektrische termen: er staat een weerstand parallel met de condensator) zodat het leegloopt en binnenkomende pulsen geleidelijk aan vergeet.

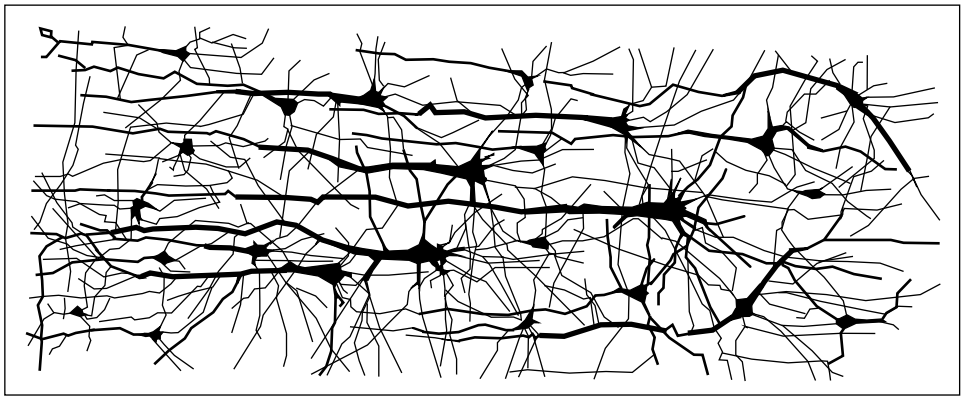
Als het niveau in het reservoir een bepaalde grenswaarde overschrijdt, wordt een puls afgevuurd en wordt het reservoir geledigd. Door het lekken is dit alleen mogelijk als er op vrij korte tijd (typisch veel minder dan een seconde) voldoende excitatorische pulsen binnenkomen die niet door inhibitorische pulsen worden teniet gedaan.

Na het afvuren van een puls heeft het neuron een zekere tijd nodig om zich te herstellen.

De absolute herstelperiode is weer 1 tot 2 milliseconden lang. Gedurende deze periode kan het neuron geen puls afvuren. Bijgevolg is de maximale pulsfrequentie van een neuron 500 à 1000 Hz, maar meestal ligt de frequentie lager. Een actief neuron vuurt typisch enkele tientallen tot enkele honderden pulsen af in een seconde; een neuron in rust uiteraard geen, of bijna geen. Na de absolute herstelperiode is er een relatieve herstelperiode van 5 tot 7 milliseconden waarbij de grenswaarde voor het afvuren van een puls daalt tot haar normale waarde.

Leren vindt plaats in de synapsen. Naargelang wat er gebeurt in de twee neuronen die hij verbindt, kan een synaps meer of minder doorlaatbaar worden. Deze verandering kan tijdelijk zijn (enkele seconden of langer) maar ook permanent (langetermijngeheugen). Sommige synapsen kunnen niet leren, en hoe de verandering gebeurt hangt af van een aantal factoren. Zo zal een excitatorische synaps die signalen van neuron A aan neuron B doorgeeft meer signaal gaan doorgeven als B afvuurt vlak na A.

6.2.2 Neurale netwerken



Figuur 6.3. Een deeltje van het menselijk brein. De axonen wijzen naar rechts.

Een menselijk brein heeft, volgens de meest recente schattingen zowat 30 miljard (3×10^{10}) neuronen en dus zowat 100 biljoen (10^{14}) synapsen. Er is geen speciale globale structuur in dit netwerk. Al deze neuronen werken uiteraard parallel. Op deze manier wordt de traagheid van de individuele componenten gecompenseerd door een massief parallellisme: er kunnen veel componenten tegelijk bezig zijn. Vergelijk dit met een klassieke computer, waarbij slechts een zeer beperkt aantal schakelingen tegelijk actief is. Zo zijn er hoogstens enkele geheugenplaatsen tegelijk actief. Vandaar dat we dus hebben:
Een biologisch net is op microniveau zeer traag, maar op macroniveau zeer snel.

HOOFDSTUK 7
KUNSTMATIGE NETWERKEN

Of het nu gaat om programma- of apparatuursimulaties van netten, er zijn een aantal gelijkaardige eigenschappen die in verschillende vormen steeds terugkomen. Zoals bij biologische netten hebben we steeds een gewogen gerichte graaf met (kunstmatige) neuronen als knopen. De invoerneuronen zetten signalen van buitenuit om. Dit hangt uiteraard af van de toepassing, en op dit punt gaan we er niet verder op in, noch op de uitvoerneuronen. Binnenin een netwerk zijn (meestal) alle neuronen van dezelfde opbouw. Een signaal dat uit een neuron vertrekt wordt vermenigvuldigd met het gewicht van de tak voor het aankomt bij het volgende neuron. Bovendien reageren neuronen altijd alleen op de *som van alle invoer* en is er een leerproces mogelijk dat de gewichten van de takken wijzigt.

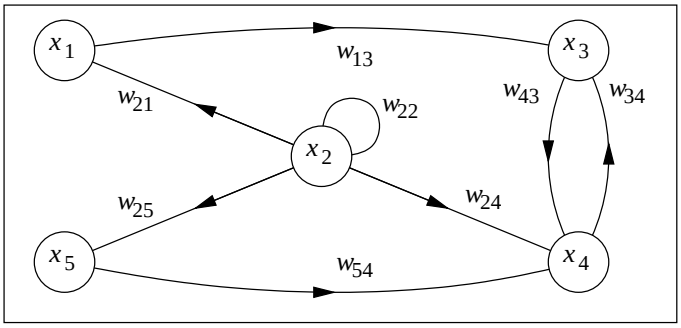
Er is echter wel nogal wat verschil tussen de verschillende toepassingen. Een eerste aandachtspunt is het verschil tussen analoge en digitale netwerken, waarbij de laatste nog twee vormen kunnen hebben (binaire en polaire) die niet essentieel van mekaar verschillen.

Bij analoge netwerken kan de uitvoer van een neuron continu variërende waarden aannemen, bij digitale netwerken slechts twee (men zou aan tussenvormen kunnen denken, maar deze hebben weinig praktisch nut; hier en daar treft men wel eens drie uitvoerwaarden aan). Bij een *binair* netwerk heeft men de waarden 1 en 0, bij een *polair* netwerk de waarden 1 en -1. Men kan altijd de ene vorm transformeren in de andere, maar soms is de ene vorm praktischer dan de andere. Merk op dat de *gewichten* van het net niet beperkt zijn tot twee waarden. Biologische netten zijn uiteraard altijd analoge netten.

Verder is er ook nog een verschil in hoe de factor *tijd* in het netwerk verwerkt wordt. Dit is vooral belangrijk in netwerken met terugkoppeling, waar de uitvoer van een neuron rechtstreeks of onrechtstreeks (via andere neuronen) terug deel kan uitmaken van de invoer van hetzelfde neuron. Tenslotte zijn er nog verschillende manieren van leren, die we in een later hoofdstuk in detail zullen bespreken. Voor de rest van deze cursus gebruiken we (bijna) altijd de volgende notaties:

1. De neuronen van het netwerk worden aangeduid met de letters x, y of z. Ze worden genummerd door een onderindex. Als er maar een soort neuronen is hebben we dus de neuronen $x_1, \ldots, x_n$, waarbij n het aantal neuronen in het net is.
2. De takken krijgen meestal geen apart symbool. Het gewicht van de tak die van x_i naar x_j gaat noteren we w_{ij}; als er geen tak is stellen we altijd $w_{ij} = 0$. Het gewicht van een bestaande tak kan ook nul zijn op een bepaald ogenblik, maar kan door leren veranderen; als er geen tak is moet w_{ij} altijd nul blijven.
3. De uitvoer van neuron x_i duiden we aan met u_i.

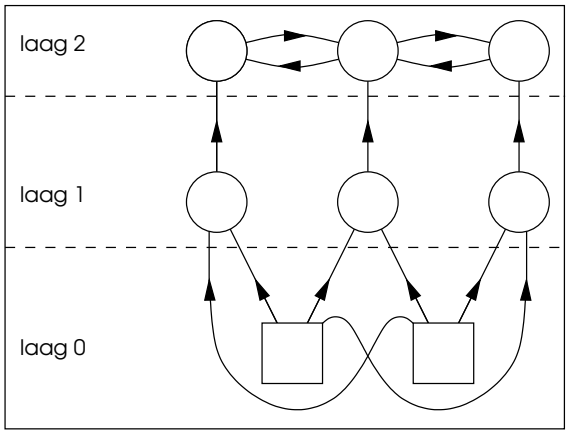
We gebruiken ook de vectornotatie uit Hoofdstuk I waarbij we een rij getallen kort schrijven met een vette letter. We schrijven dus $\mathbf{v}$ in plaats van $(v_1, \ldots, v_k)$, $\mathbf{u}$ in plaats van $(u_1, \ldots, u_n)$ enzoverder.



Figuur 7.1. Schema van een neuraal net.

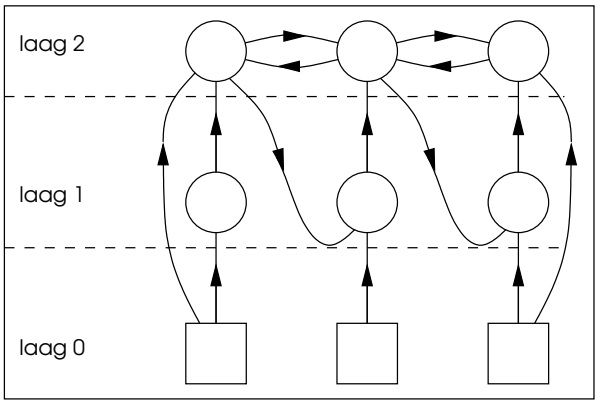
Hoe uitgebreider een netwerk is, en hoe ingewikkelder de structuur van de geassocieerde graaf, hoe moeilijker het wordt om iets zinnigs te voorspellen over het gedrag van het netwerk. Vandaar dat de best bestudeerde netwerken een duidelijke structuur hebben. Het belangrijkste kenmerk van een netwerk is of er ja dan neen terugkoppeling is. Terugkoppeling heeft specifieke functies, zoals kortetermijngeheugen. Verder is een net vaak ingedeeld in lagen (meer correct: de neuronen zijn verdeeld over verschillende lagen). Hierbij heeft elke laag een specifieke functie. Naargelang deze functie kan een laag al dan niet terugkoppeling hebben tussen de neuronen van de laag onderling: we hebben het hier dus niet over terugkoppelingslussen die over verschillende lagen gespreid zijn. De invoerneuronen worden niet meegerekend, vermits die meestal geen informatie bewerken, maar ze enkel doorgeven: zij vormen de nulde laag. We duiden ze dan ook niet aan met een cirkeltje, zoals de gewone neuronen, maar met een vierkantje. De laatste laag wordt steeds gevormd door de uitvoerneuronen. We onderscheiden volgende vormen:

- Gesloten gelaagde netten. Bij deze structuur zijn er alleen verbindingen tussen opeenvolgende lagen van het net. Er zijn dus takken van laag k naar laag $k + 1$ en, bij tussenlaagse terugkoppeling, van $k + 1$ naar k, maar niet verder in het net. Een voorbeeld een gesloten gelaagd net zonder tussenlaagse terugkoppeling zien we in Figuur 7.2.



Figuur 7.2. Gesloten gelaagd net.

- Open gelaagde netten: hierbij kunnen er verbindingen zijn tussen twee willekeurige lagen. De naam gelaagd net is hier eerder een conventie die men gebruikt als men de neuronen kan groeperen naargelang de functie. Een voorbeeld een open gelaagd net met tussenlaagse terugkoppeling vinden we in Figuur 7.3.



Figuur 7.3. Open gelaagd net.

7.1 ARTIFICIËLE NEURONEN

Alle kunstmatige neuronen (behalve invoerneuronen) lijken zeer sterk op mekaar. Er zijn echter drie belangrijke kenmerken die van soort tot soort verschillen:

1. De reactiefunctie, ook excitatiefunctie genoemd. Deze geeft aan wat de uitvoer is in functie van de invoer.
2. Hoe de verandering in uitvoer wordt doorgevoerd. Meestal verandert de uitvoer direct als de invoer verandert. Bij sommige architecturen echter, zoals bij de Hopfieldnetten die we zullen bespreken, behoudt een neuron zijn uitvoer tot het een extern signaal krijgt. Op dat ogenblik wordt de uitvoer herberekend, en deze uitvoer blijft dan weer behouden tot het neuron weer gekozen wordt.
3. De factor tijd. In biologische netwerken speelt deze een belangrijke rol. In artificiele netwerken wordt vaak verondersteld dat alle wijzigingen ogenblikkelijk gebeuren. Ook hier gaan we verder op in.

7.2 TLU'S

TLU staat voor **T**hreshold **L**ogic **U**nit. Historisch is dit het eerste model dat grondig bestudeerd werd (McCulloch en Pitts, 1943). Ze worden gebruikt in binaire netwerken; de uitvoer is dus altijd 0 of 1. Naast de gewichten wordt de werking van het neuron bepaald door een drempelwaarde (threshold), die voor elk neuron verschillend kan zijn, en net zoals de gewichten kan veranderen bij leren. Als de drempelwaarde van neuron x_i gelijk is aan T_i wordt de uitvoer gegeven door

$$u_i = \begin{cases} 1 & \text{als} \quad \sum_j w_{ji} u_j - T_i \geq 0 \\ 0 & \text{als} \quad \sum_j w_{ji} u_j - T_i < 0 \end{cases} .$$

Omwille van de symmetrie laat men vaak de drempelwaarden weg. Men voegt dan een neuron x_0 toe aan het netwerk dat geen ingangen heeft. Vermits de invoer van dit neuron nul is, is de uitvoer altijd 1. x_0 stuurt zijn uitvoer naar alle andere neuronen, waarbij het gewicht van de verbinding gegeven wordt door $w_{0i} = -T_i$ (zie Figuur 8.1). De excitatiefunctie wordt dan voor alle neuronen gelijk, en wordt gegeven door

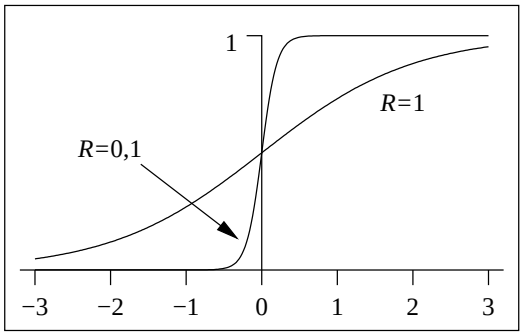
$$f(r) = \begin{cases} 1 & \text{als} \quad r \geq 0 \\ 0 & \text{als} \quad r < 0 \end{cases} .$$

7.3 ANALOGE NETTEN

Systemen met TLU's zijn gemakkelijk te bestuderen omdat hun uitvoer zeer eenvoudig kan beschreven worden. Analoge netten zijn echter veel flexibeler, en leunen nauwer aan bij het biologische voorbeeld. De uitvoer van een analoog neuron kan continu veranderen. Wel neemt men aan dat de functie stijgend is: als de invoer van het neuron vergroot, vergroot ook zijn uitvoer. Meestal, maar niet altijd, is de uitvoer van een neuron begrensd. Een zeer populaire functie is de sigmoide[1] functie σ gegeven door

$$\sigma(r) = \frac{1}{1 + \exp\left(\frac{-r}{R}\right)}.$$

Hierin is R een parameter die aangeeft hoe snel de functie verloopt. Voor $R \to 0$ wordt de functie bijna binair, en benaderen we een TLU. voor R zeer groot verloopt de functie vlak. Soms gebruikt men andere functies, zoals de boogtangens, die analoge eigenschappen heb-



Figuur 7.4. Sigmoide.

ben.

7.4 TIJD

Zoals we gezien hebben speelt tijd geen echte rol bij netwerken zonder terugkoppeling. We geven zo een netwerk een bepaalde invoer, en na een zekere tijd is de uitvoer van elk neuron een exacte weerspiegeling van zijn invoer: het netwerk is stabiel geworden. Als de invoer verandert in functie van de tijd is er reeds een zekere complicatie. Als we een gesloten gelaagd netwerk hebben, en de neuronen hebben een zekere inertie, dan komt de uitvoer

[1] De naam komt van de Griekse letter sigma (kleine letter: σ, hoofdletter Σ). Op het einde van een woord wordt deze geschreven als ς: de grafiek van de functie heeft de vorm van een langgerekte ς.

van het netwerk op een bepaald ogenblik overeen met de invoer van enige tijd tevoren. Is het netwerk open, dan wordt het nog ingewikkelder. Bij terugkoppeling wordt het pas echt interessant. Er zijn verschillende methodes om tijd te modelleren, en deze leveren verschillende resultaten op. Bekijken we hiervoor een netwerk dat uit twee TLU's x_1 en x_2 bestaat, met gewicht $w_{12} = w_{21} = 1$ en $w_{11} = w_{22} = 0$, en drempels $T_1 = T_2 = 0.5$. De uitvoer van elk neuron is dus gelijk aan de invoer.

We veronderstellen eerst dat de reactie van elk neuron ogenblikkelijk is, en dat beide neuronen als invoer kunnen dienen. Hiermee bedoelen we dat de invoer vanuit het andere neuron even kan onderdrukt worden, en vervangen door een ander signaal. We krijgen dan we het volgende beeld: als we in een van de twee neuronen een signaal (1 of 0) invoeren, dan geven beide neuronen ogenblikkelijk dit signaal als uitvoer, en het net blijft in deze toestand tot het weer onderbroken wordt.

Een eerste methode voor het modelleren van tijd is het invoeren van een discrete tijd. De tijd wordt, zoals bij een computer, door het tikken van de klok verdeeld in momenten $t_0, \ldots, t_i, \ldots$. Elk neuron geeft op ogenblik t_i de uitvoer die overeenkomt met zijn invoer op t_{i-1}. In ons voorbeeld is de uitvoer van elk neuron de uitvoer van het andere neuron op het vorige moment. Is de toestand op t_0 bijvoorbeeld 01, dan is hij op t_1, t_3, t_5, …, 10, en op de even momenten 01. Voeren we een signaal in een neuron in, dan wordt de uitvoer van het andere neuron onderdrukt. Voeren we bv. een 1 in aan neuron x_1 op t_0, dan verdwijnt de nul uit de toestand, en op alle verdere momenten is de uitvoer 11.

Een andere methode is dat neuronen alleen veranderen als we ze aanduiden. Is de begintoestand van ons netwerk 01, dan blijft deze zo tot we een neuron aanduiden. Duiden we het eerste aan dan wordt de toestand 11. De vorige methode is een speciaal geval hiervan: we duiden bij elke tik van de klok alle neuronen tegelijk aan.

Deze twee methodes kunnen we gebruiken zowel bij TLU's als bij analoge neuronen. Beide zijn gemakkelijk te programmeren. De tweede methode is iets sneller en vraagt minder geheugen. Bij analoge neuronen kunnen we nog een continue overgang gebruiken. Is de invoer van neuron x_i op ogenblik t gelijk aan $s_i = \sum_{j=1}^{n} w_{ji} u_j$, dan wordt de verandering van de uitvoer gemodelleerd met een interne toestand a_i die afhangt van de tijd. De uitvoer wordt dan gegeven door

$$u_i(t) = f(a_i(t)) \qquad \frac{da_i}{dt} = -a_i(t) + s_i(t).$$

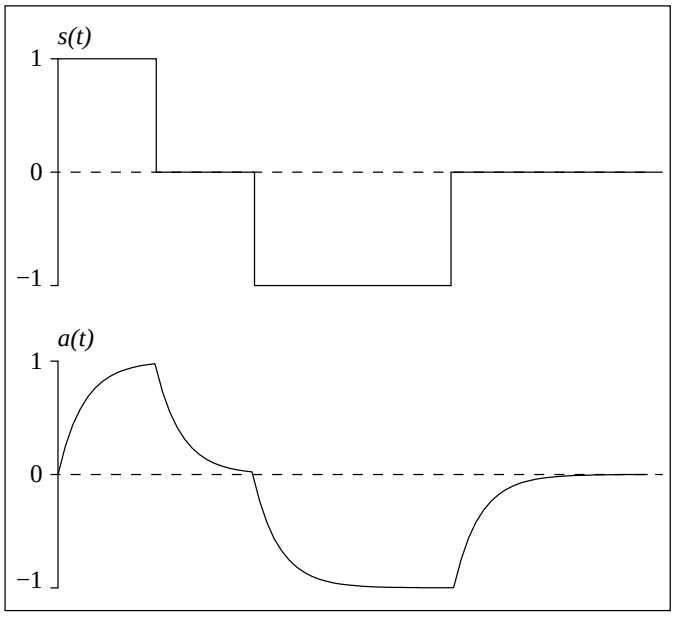
Hierin is f de excitatiefunctie. De tweede vergelijking zegt dat a_i verkleint als a_i groter is dan s_i, evenredig met het verschil. Als $a_i < s_i$, dan wordt a_i groter. Deze derde methode is een nauwkeurig model van een biologisch netwerk. Simulatie op een computer is zeer traag, maar voor hardwarenetten werkt deze methode zeer goed.

7.5 LEREN

Er zijn verschillende manieren om soorten leren in te delen. Een eerste indeling hebben we reeds gezien: we hadden algoritmisch leren, leren met supervisie en leren zonder supervisie. Dit is onderscheid dat kan gemaakt worden bij het leren door mensen of dieren, maar ook bij het verwerven van vaardigheden door een computer: algoritmisch leren komt dan overeen met klassiek programmeren. De twee andere vormen werken met aanpasbare parameters.

Een tweede indeling is specifiek voor het leren van massief lerende programma's, zoals artificiële neurale netten, en heeft te maken met hoe het netwerk het leeraanbod verwerkt.

Bij biologische netten is er maar een manier waarop het net kan leren. Elk gewicht wordt aangepast –of, om exact te zijn, de doorlaatbaarheid van de synaps verandert– geba-



Figuur 7.5. Continue tijdsafhankelijkheid.

seerd op de activiteit van de twee verbonden neuronen. Bij een artificieel net veranderen de gewichten ook[2], maar er zijn verschillende manieren waarop dit kan gebeuren.

In tegenstelling tot biologische netten, waar er geen onderscheid is tussen gebruiken en leren, hebben de meeste artificiële netten een leer- en een gebruiksfase. In de gebruiksfase verandert de waarde van de gewichten niet, en het netwerk voert een of andere taak uit. In de leerfase worden de gewichten bepaald. We gaan hier uit van een leerproces met supervisie. De meest voorkomende vorm gaat uit van een verzameling invoergegevens met de bijbehorende uitvoergegevens. We proberen nu de gewichten op zodanige manier aan te passen dat de gegeven invoer de gevraagde uitvoer geeft. We onderscheiden volgende methodes:

- Elk in- en uitvoergegeven wordt apart aan het net gepresenteerd en de gewichten worden aangepast 'in de goede richting', waarbij de gewichten tussen twee neuronen worden vergroot in directe functie van de activiteit van de twee neuronen. Deze manier sluit zeer nauw aan bij het biologische leerproces. Men noemt deze methode leren volgens de *regel van Hebb*[3].

- Men kan ook graduele aanpassing gebruiken die de volgende regel niet volgt, maar afgeleid is uit kennis van het probleem. Dit gebeurt o.a. bij systemen van steunvectoren. De gewone methode maakt gebruik van een soort foutfunctie die geminimaliseerd moet worden. Bij elke stap wordt geprobeerd een kleine aanpassing aan de gewichten door te voeren die de fout verkleint.

- Soms worden de gewichten bepaald uit vergelijkingen die een optimale waarde voor de gewichten geven. Deze manier staat het verste af van biologisch leren.

[2] Er zijn ook leermethodes waarbij de structuur van het net wordt aangepast, waarbij neuronen en verbindingen worden toegevoegd of weggelaten. We gaan niet dieper in op deze methodes.

[3] Donald Hebb, Amerikaans psycholoog, speelde een belangrijke rol in de studie van de leermethodes van biologische neurale systemen.

Van elk van de drie vermelde vormen bestaan er heel wat varianten. Welke er gebruikt wordt hangt af van het probleem, en ook van het doel. Als men poogt menselijk of dierlijk leren te simuleren zal men uiteraard werken met een regel van Hebb; voor sommige specifieke problemen kan het echter zijn dat er een meer efficiënte leermethode bestaat.

HOOFDSTUK 8
INFORMATIEVERWERKING MET NEURALE NETTEN

In dit hoofdstuk bekijken we een aantal vormen van informatieverwerking zoals deze door een neuraal net kan gebeuren. Om het eenvoudig te houden bekijken we alleen netwerken met TLU's, het model dat het verste afstaat van biologische neurale netten.

We beginnen met een tijdsonafhankelijk probleem: gegeven een functie van binaire patronen naar binaire patronen, kunnen we dan een netwerk maken dat deze functie realiseert? Het blijkt dat het antwoord ja is, en dat we zelfs met een vrij eenvoudige structuur kunnen werken: een netwerk zonder terugkoppeling en hoogstens twee lagen. Wel kan het netwerk vrij veel neuronen bevatten: typisch is de grootte $O(s)$, waarbij s het aantal mogelijke invoercombinaties is. Deze methode is dus alleen bruikbaar voor kleine problemen, maar is een belangrijke hulp om bijvoorbeeld de in- of uitvoer van een complex net in de juiste vorm te gieten.

Een tweede vorm van informatieverwerking die we behandelen gebeurt door middel van automaten. Automaten worden ingedeeld in verschillende klassen, die allemaal met neurale netten kunnen worden geïmplementeerd. Zij zijn vooral belangrijk in de context van talen beschreven door generatieve grammatica's. Het gaat dan zowel over door de mens gemaakte, formeel gedefinieerde, talen (zoals programmeertalen) als over natuurlijke talen (zoals Nederlands of Swahili) als over talen die bij bepaalde problemen horen (de opbouw van DNA gehoorzaamt aan een formele, nog niet geheel gekende, grammatica). Automaten kunnen gebruikt worden bij de analyse van 'zinnen' uit de taal. Als de taal een programmeertaal is, is zo'n 'zin' bijvoorbeeld een programma of een implementatie van een klasse, bij een natuurlijke taal gaat het over echte zinnen, en bij DNA-analyse gaat het over een geheel chromosoom of een kleinere eenheid, zoals een gen. Maar ook het construeren van 'zinnen' kan gebeuren door een automaat, die dan naast de formele grammatica gebruikt maakt van semantische gegevens die de inhoud van de te construeren zin bepaalt. We behandelen hier slechts een vorm van automaat, de zogenaamde Mooreautomaat, maar voor de verschillende vormen van automaten die gebruikt worden voor verschillende taken, en die horen bij verschillende soorten grammatica's, kan een realisatie in een neuraal netwerk gegeven worden.

Een derde vorm is deze van associatieve geheugens. Bij een klassiek computergeheugen vraagt men gegevens op door een adres op te geven. Dit adres heeft geen verband met de inhoud van het geheugen: als het adres van een 4 bytes grote geheugenplaats `0x001A08FC` is, dan zegt dit niets over welke bitcombinatie op die plaats te vinden is. Een associatief geheugen werkt volgens een heel ander principe: men geeft een gedeelte van de gegevens op, en dit wordt aangevuld door het geheugen. Ook voor associatieve geheugens bespreken we slechts een enkele vorm: het zogenaamde *Hopfieldnet*. Meer algemene associatieve netten hebben eigenschappen die lijken op deze Hopfieldnetten, maar zijn zeer moeilijk theoretisch te bestuderen. We bekijken associatieve geheugens in Hoofdstuk 11.

Het gebruik van TLU's in dit hoofdstuk maakt het gemakkelijk om de structuur van het

netwerk te beschrijven. Maar deze vereenvoudiging heeft een belangrijk nadeel: het levert geen realistisch model op van de manier waarop zo'n netwerk *leert*. Met uitzondering van het Hopfieldnet, en van het eenlagig perceptron dat een speciale vorm is van de realisatie van een binaire functie, kan eigenlijk geen leermethode gegeven worden. Daarom zullen we bij het bespreken van leren vrij snel overgaan tot het gebruik van analoge neuronen.

8.1 BINAIRE FUNCTIES

Zoals reeds vermeld werden TLU's, met een uitvoer die alleen de waarden 0 en 1 kan aannemen, ingevoerd door McCulloch en Pitts in 1943. Zij legden de basis voor de meeste ideeën die in dit hoofdstuk besproken worden. Ze deden dit in een vorm die heden ten dage vreemd aandoet. Zo bestond er in 1943 nog geen theorie van automaten en formele talen[1], zodat ze een eigen formalisme moesten gebruiken. Alleen de behandeling van binaire functies staat in een vorm die nog altijd (min of meer) overeenkomt met deze die nu gebruikt wordt. De stelling dat alle binaire functies kunnen gerealiseerd worden met een tweelagig net van TLU's wordt dan ook algemeen aangeduid als *de* stelling van McCulloch en Pitts, alhoewel hun ideeën veel verder gaan dan deze stelling.

De titel van het artikel is *'A Logical Calculus of Ideas Immanent in Nervous Activity'*, wat al enigszins aangeeft in welke richting de ideeën van McCulloch en Pitts gaan. De basisidee is logisch denken, en dit wordt beschouwd als het evalueren van logische uitdrukkingen. Hebben we een aantal elementaire logische uitdrukkingen, $v_1, \ldots, v_n$, dan kunnen we hiermee een meer complexe uitdrukking maken, zoals bijvoorbeeld $\neg v_1 \wedge (v_2 \vee v_3) \Rightarrow (v_1 \Rightarrow \neg v3)$. Voor gegeven waarden van de v_i kan de uitdrukking waar of onwaar zijn. McCulloch en Pitts onderzochten of het mogelijk was om voor zo een uitspraak een neuraal net te construeren dat kon nagaan of ze waar of onwaar was. Zo een net zou bestaan uit TLU's en de waarden van de v_i als invoer krijgen in de vorm van 1 of 0, en moest zelf 1 of 0 geven als uitvoer naargelang de samengestelde uitdrukking waar was of niet. Meer algemeen onderzochten McCulloch en Pitts netwerken met meer dan een uitvoerneuron, en hun stelling beperkt zich tot netwerken zonder terugkoppeling[2].

Een binaire functie is een functie F van de verzameling binaire getallen met n cijfers, $\{0,1\}^n$ naar deze van binaire getallen met k cijfers, $\{0,1\}^k$, of meer formeel

$$F : \{0,1\}^n \rightarrow \{0,1\}^k,$$

waarbij n en k vooraf bepaalde positieve gehele getallen zijn. Het is duidelijk, als we een neuraal net nemen zonder terugkoppelingslussen en met n ingangen en k uitgangen we een tijdsonafhankelijk probleem hebben: we mogen veronderstellen dat elk neuron ogenblikkelijk reageert op zijn invoer. Als we de invoer aan het net geven, kunnen we wachten tot alle neuronen een stabiele toestand bereikt hebben, en zo is de uitvoer alleen afhankelijk van de invoer. Er is m.a.w. een binaire functie die de uitvoer beschrijft als functie van de invoer. We zeggen dat het net deze functie realiseert.

De stelling behandelt nu de omgekeerde vraag: gegeven een binaire functie F, kunnen we een net vinden dat dit realiseert? Dit blijkt inderdaad zo te zijn. Merk eerst op dat we alleen moeten kijken naar het geval $k = 1$. Inderdaad, als $k > 1$, dan kunnen we k netwerken nemen met 1 uitvoerneuron. Elk van deze netwerken zorgt dan voor 1 component van F. Een voorbeeldje: stel $n = k = 2$ en bekijk de functies $F : \{0,1\}^2 \rightarrow \{0,1\}^2$,

[1] Toch niet in de vorm zoals wij die kennen.

[2] Het tweede deel van het artikel behandelt terugkoppeling.

$F_1 : \{0,1\}^2 \to \{0,1\}$, $F_2 : \{0,1\}^2 \to \{0,1\}$, gegeven door

$$\begin{array}{lll} F(0,0) = (0,1) & F_1(0,0) = 0 & F_2(0,0) = 1 \\ F(0,1) = (1,1) & F_1(0,1) = 1 & F_2(0,1) = 1 \\ F(1,0) = (1,1) & F_1(1,0) = 1 & F_2(1,0) = 1 \\ F(1,1) = (0,0) & F_1(1,1) = 0 & F_2(1,1) = 0 \end{array} .$$

Samenvoegen van F_1 en F_2 geeft duidelijk F.

Nemen we een binaire functie F en zoeken we een net dat, als een invoer $(b_1, \ldots, b_n)$ van bits gegeven is, als uitvoer $F(b_1, \ldots, b_n)$ geeft. Het net dat we zoeken heeft dus n invoerneuronen en k uitvoerneuronen. Merk nog eens op dat de laag van invoerneuronen die de rij $(b_1, \ldots, b_n)$ doorgeeft, niet wordt beschouwd als een laag van het net (of anders als de nulde laag van het net), terwijl de uitvoerneuronen wel een laag van het net vormen. Als we dus spreken van een tweelagig net hebben we dus tussen de in- en de uitvoerneuronen nog één laag van tussenliggende neuronen.

McCulloch en Pitts bewezen dat voor *elke* binaire functie er een net bestaat dat de functie realiseert. Het is mogelijk dat twee verschillende netten (met verschillend verbonden neuronen of met verschillende gewichten) dezelfde functie realiseren. De stelling geeft niet noodzakelijk het meest efficiënte net voor de functie: in veel gevallen bestaan er kleinere netten die de gegeven functie realiseren. Voor we de stelling en haar bewijs geven, kijken we eerst naar twee functies die met een eenlagig net kunnen worden gerealiseerd:

(1) De OF-functie, die $(0, \ldots, 0)$ op 0 afbeeldt, en alle andere combinaties op 1. Deze kan worden gerealiseerd door een neuron met n ingangen waarbij elk gewicht 1 is, en een drempelwaarde van 0.5.

(2) De selectiefunctie die een vooropgegeven bitcombinatie $(b_1, \ldots, b_n)$ op 1 afbeeldt, en alle andere op 0. Hiervoor nemen we een neuron met n ingangen, waarbij elk gewicht gegeven wordt door $w_i = 2b_i - 1$, dus $w_i = 1$ als $b_i = 1$, en $w_i = -1$ als $b_i = 0$. Stel nu dat er k bits in $(b_1, \ldots, b_n)$ gelijk zijn aan 1, en neem een willekeurige bitcombinatie $(c_1, \ldots, c_n)$. Dan is

$$\sum_i w_i c_i = k - v,$$

waarbij v het aantal bits is waarin $(c_1, \ldots, c_n)$ verschilt van $(b_1, \ldots, b_n)$. Inderdaad, als voor een index i geldt dat $c_i = 0$ en $b_i = 1$, dan is $w_i b_i = 1$, terwijl $w_i c_i = 0$, dus 1 minder. Als voor de index i geldt dat $c_i = 1$ en $b_i = 0$, dan is $w_i b_i = 0$, terwijl $w_i c_i = -1$, dus ook 1 minder. Als we als drempelwaarde $T = k - 0.5$ nemen, voldoet ons neuron aan de voorwaarden.

Maar niet alle binaire functies met één uitgang kunnen met één enkel neuron gerealiseerd worden. De functie F_1 die we zojuist bekeken is de XOF-functie (exclusieve of), en dit is een voorbeeld van een functie die niet kan gerealiseerd worden op deze manier. Inderdaad, zo een neuron zou twee ingangen hebben, stel v_1 en v_2, twee gewichten w_1 en w_2, en een drempelwaarde T. Kunnen we nu gewichten en een drempelwaarde vinden die de functie F_1 weergeeft? Het antwoord is neen. Immers, zo'n waarden moeten voldoen aan

$$\begin{array}{lcccc} F_1(0,0) = 0 & \Rightarrow & -T < 0 & \Rightarrow & T > 0 \\ F_1(0,1) = 1 & \Rightarrow & w_2 - T \geq 0 & \Rightarrow & T \leq w_2 \\ F_1(1,0) = 1 & \Rightarrow & w_1 - T \geq 0 & \Rightarrow & T \leq w_1 \\ F_1(1,1) = 0 & \Rightarrow & w_1 + w_2 - T < 0 & \Rightarrow & T > w_1 + w_2. \end{array}$$

Dit leidt tot een contradictie.

Stelling 3 (McCulloch en Pitts) *Zij $F : \{0,1\}^n \to \{0,1\}^k$ een willekeurige functie. Dan is er een gesloten gelaagd netwerk met ten hoogste twee lagen dat F weergeeft. Anderzijds bestaan er functies die niet door een netwerk met één laag kunnen worden weergegeven.*

Bewijs
Hoe kunnen we nu een willekeurige functie F met een tweelagig net realiseren? Stel dat F ℓ bitcombinaties op 1 afbeeldt. We construeren nu een tweelagig net als volgt:

1. De eerste laag bevat ℓ neuronen. Elk daarvan realiseert de selectiefunctie voor een bitcombinatie die op 1 moet worden afgebeeld.
2. De tweede laag bestaat uit een neuron met ℓ ingangen die de OF-functie realiseert.

Dit netwerk realiseert de functie F. ∎

Als we binaire functies opvatten als logische uitdrukkingen van de invoeren $v_1, \ldots, v_n$, dan krijgen we verschillende niveaus:

1. Basiszinnen. Dit zijn atomaire uitdrukkingen (v_i) of de negatie ervan ($\neg v_i$).
2. Eenvoudige samenstellingen van basiszinnen.

   2a. Conjuncties van basiszinnen. Deze worden gerealiseerd met de selectiefunctie. Zo wordt de conjunctie $v_1 \wedge \neg v_2 \wedge \neg v_3$ gerealiseerd met de selectie van het patroon 100. Dit geeft aan dat we de selectiefunctie kunnen gebruiken voor eenvoudige negatie en voor de EN-functie.

   2b. Negatie van zulke conjuncties. Deze krijgen we door de gewichten (inclusief de drempel) met -1 te vermenigvuldigen.

   2c. Disjuncties van basiszinnen. Zulk een disjunctie is de negatie van een conjunctie (wetten van de Morgan): bijvoorbeeld is $\neg v_1 \vee v_2 \vee v_3 = \neg(v_1 \wedge \neg v_2 \wedge \neg v_3)$, en worden dus gerealiseerd met de methode beschreven in 2b.

Het is een algemene stelling uit de logica dat elke logische uitdrukking kan geschreven worden als een disjunctie van zulke conjuncties, of als een conjunctie van zulke disjuncties (disjunctieve en conjunctieve normaalvormen). Dit levert alternatieve methodes om te bewijzen dat de logische uitdrukking kan gerealiseerd worden met een tweelagig neuraal net.

8.2 LEREN

In ons bewijs hebben we, uitgaande van een gegeven binaire functie, een netwerk geconstrueerd dat de functie exact weergeeft. Dit is natuurlijk niet zeer nuttig: als we de functie al kennen moeten we er geen net voor bouwen. In praktische toepassingen van een net hebben we meestal een iets andere situatie (we gaan er weer van uit dat de uitvoerdimensie k gelijk is aan 1):

Gegeven een verzameling invoergegevens van n-bitgetallen $\mathcal{L}$. Voor elk van deze getallen weten we of de uitvoer 0 of 1 moet zijn. Gegeven is ook een neuron met n invoerkanalen. Bepaal de gewichten zodanig dat dit neuron zo goed mogelijk de juiste uitvoer geeft.

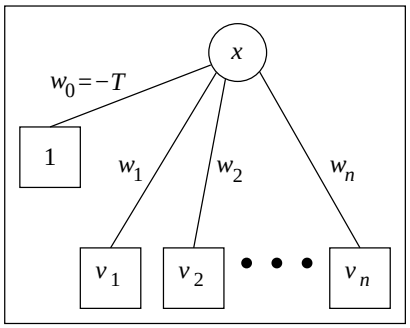
Merk op dat we niet veronderstellen dat $\mathcal{L}$ 2^n elementen bevat. Op de getallen die niet in $\mathcal{L}$ zitten is de functie niet gedefinieerd: het neuron mag teruggeven wat het wil. Maar het probeert wel te veralgemenen: als het bijvoorbeeld als leerverzameling

$$\begin{aligned} 000\ldots0 &\to 0 \\ 010\ldots0 &\to 1 \\ 001\ldots0 &\to 1 \end{aligned}$$

$$\begin{array}{l}\ldots\\ 000\ldots1 \quad \rightarrow 1\end{array}$$

opgegeven krijgt, zal het de OF-functie leren.

We hebben dus een neuron x met n invoerkanalen, waaraan we een invoerkanaal toevoegen dat altijd 1 geeft. Daarbij horen de gewichten $w_1, \ldots, w_n$, met als bijkomend gewicht w_0. We moeten de gewichten w_0,... w_n bepalen. We beginnen door al onze gewichten een willekeurige waarde te geven (bijvoorbeeld 0). Daarna voeren we de invoergegevens een voor een aan het netwerk en vergelijken de gewenste uitvoer met de reële uitvoer van het net. We passen de gewichten aan door een regel van Hebb te gebruiken. Gegeven een invoer $v_1, \ldots, v_n$ waarvoor de gewenste uitvoer y is (er geldt dus steeds dat $y = 1$ of $y = 0$), en de reële uitvoer u.



Figuur 8.1. Netwerk met één neuron.

Als $y = u$, dan veranderen we de gewichten niet. Als $y \neq u$ dan veranderen we elk gewicht zo:

$$w_i \rightarrow w_i + (y - u)v_i. \tag{8.1}$$

De invoer van x verandert nu, en het verschil is

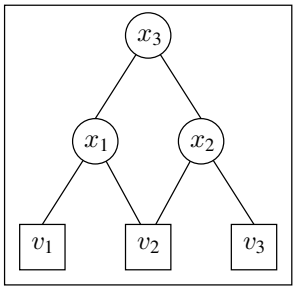
$$\sum_{i=0}^{n}(w_i+(y-u)v_i)v_i-\sum_{i=0}^{n} w_i v_i = (y-u)\sum_{i=0}^{n} v_i^2.$$

Nu is $v_0 = 1$, dus de som is $\sum_i v_i^2$ nooit 0, maar altijd strikt positief. Als nu $y-u$ positief is, en dus gelijk aan 1, dan vergroot de totale invoer van x, en wordt minder negatief, of zelfs positief waardoor de fout verdwijnt. Als $y - u$ negatief is verkleint de invoer van x. Merk echter op dat de wijziging niet garandeert dat $\mathbf{v}$ nu wel juist geklasseerd wordt.

Als we aan het einde van onze leerverzameling komen heeft ons neuron al iets geleerd, maar dat wil nog niet zeggen dat het niets meer kan bijleren. We beginnen dus opnieuw aan het begin van de leerverzameling en doorlopen deze opnieuw. Het kan zijn dat we na een aantal herhalingen vinden dat er geen fouten meer zijn, en dus geen correcties: het neuron heeft de testverzameling volledig onder de knie. Maar het kan zijn dat dit onmogelijk kan bereikt worden, want we hebben gezien dat niet elke functie met een eenlagig netwerk kan gerealiseerd worden. We moeten dus een ander stopcriterium hanteren, bijvoorbeeld het aantal keren dat we door de leerverzameling lopen. Ons leeralgoritme wordt dus:

initialiseer de gewichten
zolang (geen foutloos parcours en niet te veel pogingen){
  voor elke invoergegeven $(\mathbf{v}, t)$\{
    bereken de uitvoer van het neuron voor deze $\mathbf{v}$
    als ($y \neq$ uitvoer) pas de gewichten aan
  }
}

Kunnen we nu een tweelagig netwerk op deze manier laten leren? Hier zitten we met een probleem: moeten we de gewichten van verbindingen tussen lagen 0 en 1 aanpassen, of deze tussen laag 1 en 2, of een combinatie?



Figuur 8.2. Een tweelagig net

Nemen we als voorbeeld het netwerk uit Figuur 8.2 Stel dat we een functie moeten leren met $F(101) = 0$ en $F(011) = 1$, maar dat, als we (101) of (011) aan het net presenteren, we twee keer $u_1 = 0$ en $u_2 = 1$ krijgen. Het is dan duidelijk dat we de gewichten tussen de invoerlaag en laag 1 moeten aanpassen, maar we weten niet hoe, omdat we geen idee hebben wat 'correcte' waarden voor u_1 en u_2 zijn.

Het heeft lang geduurd voor men een goede oplossing vond voor dit probleem, dat ook bekend staat als dat van het terugvoeren van de fouten (*error backpropagation*). Hoe moeten we fouten in de uitvoer terugvoeren op fouten in de gewichten? Dit probleem heeft het onderzoek en gebruik van neurale netten decennia lang opgehouden. Pas in 1974 werd een oplossing gevonden, die pas midden de jaren '80 wijde bekendheid kreeg. We komen er later gedetailleerd op terug. Vermelden we hier dat deze methode niet bruikbaar is voor een netwerk met TLU's: we hebben analoge neuronen nodig om een meerlagig netwerk te construeren dat kan leren.

8.3 AUTOMATEN

Zoals reeds vermeld zijn er verschillende vormen van automaten die kunnen gerealiseerd worden met een neuraal netwerk. De vorm die we hier behandelen is die van Mooreautomaten. Een Mooreautomaat bevindt zich altijd in een bepaalde *staat*, en verwerkt invoer van een bepaalde vorm letter per letter.

Elke keer er een letter binnenkomt, kan de automaat naar een andere staat overgaan. Als ze dit doet, zendt ze ook een signaal uit. Meer formeel heeft een Mooreautomaat de volgende kenmerken:

- Een eindige verzameling staten,

  $$Q = \{q_1, \ldots, q_{|Q|}\}.$$

  Een Mooreautomaat bevindt zich steeds in één enkele staat: het is dus een deterministische automaat.

- Een invoeralfabet, dit is een eindige verzameling van mogelijke invoeren,

  $$S = \{s_1, \ldots, s_{|S|}\}.$$

- Een uitvoeralfabet, dit is een eindige verzameling van mogelijke uitvoeren,

  $$G = \{g_1, \ldots, g_{|G|}\}.$$

- Een transitiefunctie

  $$d : Q \times S \to Q.$$

  Deze geeft aan naar welke staat de Mooreautomaat gaat als ze een bepaald symbool binnenkrijgt. Als ze zich in een staat q_i bevindt en dan het symbool s_j binnenkrijgt dan gaat ze naar de staat $d(q_i, s_j)$.

- Een uitvoerfunctie

  $$\ell : Q \to G.$$

  Deze definieert de uitvoer van de automaat: als de automaat in de staat q_i aankomt, geeft ze als uitvoer $\ell(q_i)$.

- Een beginstaat $q_I \in Q$ die aangeeft van waaruit de automaat vertrekt. Merk op dat de functiewaarde $\ell(q_I)$ nooit gebruikt wordt. De index I in q_I staat voor *i*nitiële toestand.

Een Mooreautomaat lijkt goed op een eindige deterministische automaat zoals die gebruikt wordt om zinnen van een formele grammatica te herkennen. Vermits er een eindig aantal staten is, heeft een Mooreautomaat een eindig geheugen: de automaat herinnert zich iets van invoer uit het verleden, maar niet alles. Invoer die letter per letter wordt aangeboden moet niet noodzakelijk iets met tekst te maken hebben. Zo kan een Mooreautomaat een strategie implementeren. Het invoeralfabet S is hierbij dan de verzameling van mogelijke visies op de wereld, en het uitvoeralfabet G is de verzameling van acties.

Omdat we werken met TLU's zullen we steeds veronderstellen dat zowel het in- als het uitvoeralfabet bestaat uit binaire strings van een vaste lengte, met andere woorden dat er n en k bestaan waarvoor

$$S \subset \{0,1\}^n \qquad G \subset \{0,1\}^k.$$

Als we nu een neuraal model willen maken van een Mooreautomaat, dan moeten we met tijdsafhankelijke neuronen werken. We gebruiken een model waarbij de uitvoer van een neuron op een bepaald ogenblik t afhangt van zijn totale invoer op het ogenblik $t-1$. We zullen zien dat elke invoer in ons model door twee neuronen na elkaar moet worden verwerkt[3]. We gaan er dan ook van uit dat de opeenvolgende symbolen worden aangevoerd op de tijdstippen $t = 0, 2, 4, \ldots$. De oneven tijdstippen geven de automaat de kans voor de verwerking te zorgen.

Is nu een Mooreautomaat, met Q, S, G, d, ℓ, en q_I gegeven. We maken twee veronderstellingen:

- Het invoeralfabet is exact de verzameling van binaire strings van lengte n bestaande uit $n-1$ nullen en één 1.
- Het uitvoeralfabet is exact de verzameling van binaire strings van lengte k bestaande uit $k-1$ nullen en één 1.

Deze veronderstellingen zijn niet echt beperkingen. Veronderstel dat S een willekeurige verzameling is van binaire strings van lengte n. We kunnen dan de invoerlaag vervangen door een laag van $|S|$ neuronen waarbij het i-de neuron de selectiefunctie realiseert voor s_i. Merk op dat deze verwerking van de invoer parallel gebeurt met de werking van de rest van de automaat, en dus niet vereist dat de invoer trager wordt ingegeven. Wel zit er een vertraging van 1 tijdseenheid op de uitvoer.

Voor de uitvoer kunnen we ook een vertaaleenheid maken. Inderdaad, is G een willekeurige deelverzameling van $\{0,1\}^k$ met $|G|$ elementen, dan bestaat er volgens de stelling van McCulloch en Pitts een tweelagig netwerk dat, voor gegeven i $(b_1, \ldots, b_{|G|})$, waarin $b_i = 1$ en $b_j = 0$ voor $j \neq i$ afbeeldt op g_j.

Het neuraal netwerk heeft nu $|Q|$ neuronen $\{x_1, \ldots, x_{|Q|}\}$ die elk overeenkomen met een staat. Het is handig om een extra invoerkanaal in te voeren dat enkel verbonden is met de startstaat q_I. Het startsignaal wordt dan gegeven op $t = -1$, dus juist voor de eerste echte invoer wordt ingegeven.

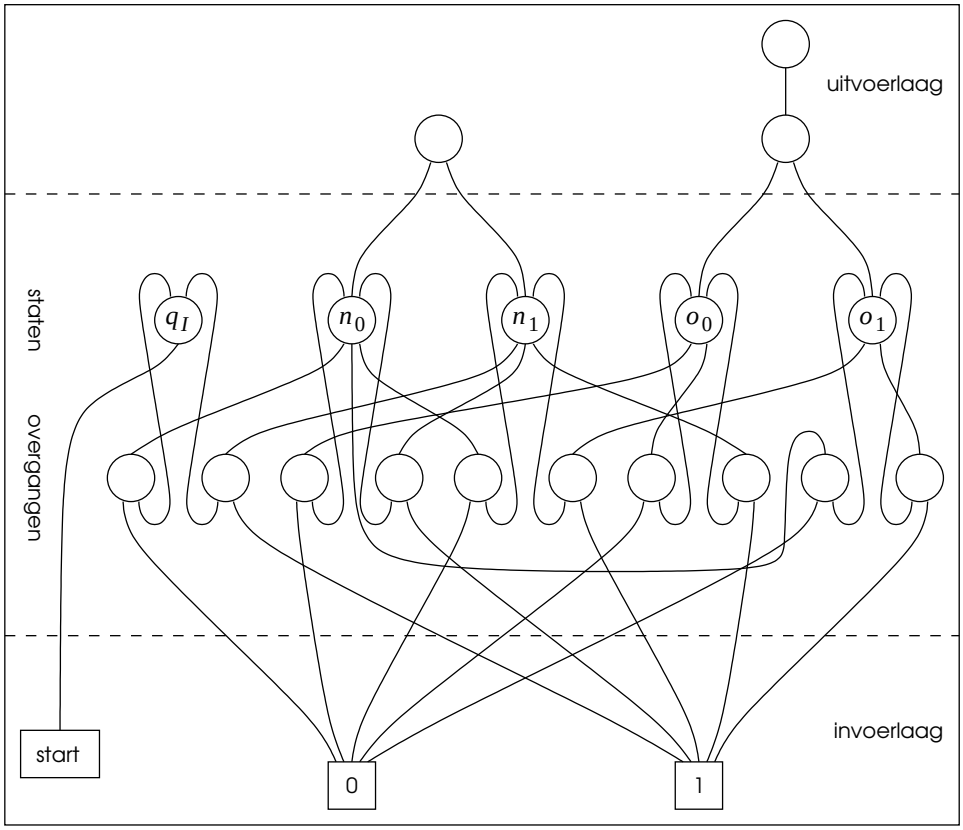
De transitiefunctie d kan beschouwd worden als een tabel met $|Q|$ rijen en n kolommen. Op de i-de rij en de j-de kolom staat dan aangegeven wat $d(q_i, s_j)$ is. De functie wordt gerealiseerd met $n|Q|$ neuronen, die elk een EN-functie met twee ingangen implementeren. Als $d(q_i, s_j) = q_k$ dan worden de twee ingangen van het overeenkomstig neuron verbonden

[3] Dit is niet echt nodig: de neuronen die overeenkomen met een staat kunnen worden weggelaten. Dit geeft echter een minder duidelijk model.

met de uitgang van x_i en met het j-de invoerneuron, terwijl zijn uitgang verbonden wordt met de ingang van x_k, met gewicht 1.

De uitvoerfunctie wordt als volgt gerealiseerd: als $\ell(q_i) = g_j$, dan wordt de uitgang van x_i verbonden met de ingang van het j-de uitvoerneuron, met gewicht 1.

Merk op dat alle gewichten in het netwerk 1 zijn, en dat elk neuron voor een overgang drempel 1,5 heeft (EN-functie), terwijl elk statenneuron drempel 0,5 heeft (OF-functie). Op elk ogenblik heeft ofwel juist één statenneuron uitvoer 1 (de transitieneuronen geven allemaal nul, dit gebeurt op de even tijdstippen $t = 0, 2, \ldots$) ofwel één transitieneuron (alle statenneuronen geven 0, dit gebeurt op de oneven tijdstippen). De totale invoer van een statenneuron is dus op elk ogenblik ofwel 1 ofwel 0, en een statenneuron geeft gewoon deze invoer door. Eigenlijk kunnen ze gewoon weggelaten worden: we hebben ze enkel omwille van de duidelijkheid ingevoerd.



Figuur 8.3. Mooreautomaat

Als voorbeeld nemen we een Mooreautomaat die bit per bit inleest. Als de ingelezen bit de eerste is, of verschilt van zijn voorganger, stuurt hij de bit 1 uit, in het andere geval een 0. Zo krijgen we bijvoorbeeld

$$0001111010100001111111110100011101110011101110111 \quad \rightarrow$$
$$1001000111110001000000001110010011001010011001100$$

Het invoeralfabet is dus $\{0, 1\}$, maar dit wordt met de bovenvermelde techniek omgezet naar $\{(01), (10)\}$. Er zijn vijf staten, $Q = \{q_I, n0, n1, o0, o1\}$. Deze hebben de volgende betekenis:

- q_I is de startstaat.
- $n0$: de laatst ingelezen bit was een 0 die niet voorafgegaan werd door een andere 0.
- $n1$: de laatst ingelezen bit was een 1 die niet voorafgegaan werd door een andere 1.
- $o0$: de laatst ingelezen bit was een 0, voorafgegaan door een 0.
- $o1$: de laatst ingelezen bit was een 1, voorafgegaan door een 1.

De transitiefunctie wordt dan gegeven door

	0	1
q_I	$n0$	$n1$
$n0$	$o0$	$n1$
$n1$	$n0$	$o1$
$o0$	$o0$	$n1$
$o1$	$n0$	$o1$

terwijl de uitvoerfunctie gedefinieerd wordt door $\ell(n0) = \ell(n1) = 1$ en $\ell(o0) = \ell(o1) = 0$. Dit alles levert het neurale net van de volgende bladzijde op. Enkele opmerkingen hierbij:

- Het uitvoergedeelte in het diagram bestaat hier uit drie neuronen in twee lagen, terwijl één neuron voldoende zou zijn. We hebben een overbodig neuron getekend dat de uitvoer 0 symboliseert omdat dit overeenkomt met het schema waarbij elk letter van het uitvoeralfabet een neuron krijgt.
- De invoerlaag heeft drie invoerneuronen. Eén om het startsignaal door te geven en twee andere voor de eigenlijke invoer. De bijbehorende selectiefuncties hebben we niet getekend.

Een Mooreautomaat heeft een zeer beperkte geheugencapaciteit. Immers: op elk moment heeft hoogstens één van de statenneuronen uitvoer 1: er zijn evenveel statenneuronnen als er staten zijn. Vaak kan men echter verschillende automaten combineren. Het aantal mogelijke staten is dan het product van de aantallen van de individuele automaten.

8.4 REGULIERE UITDRUKKINGEN

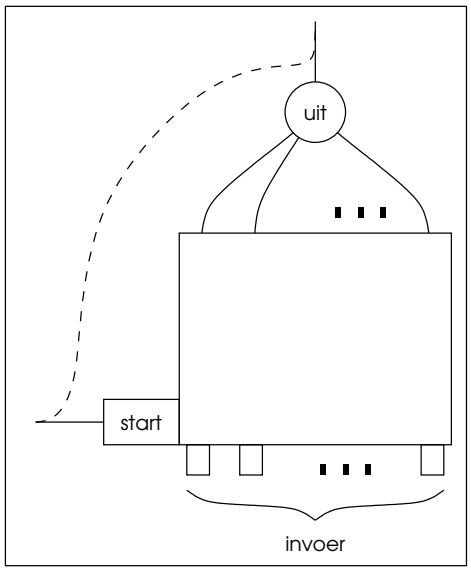
Reguliere uitdrukkingen (regexps) zijn een gekend werktuig voor informatici. Minder gekend is het feit dat ze zijn uitgevonden in de context van neurale netwerken. In 1956 publiceerde Stephen Kleene het artikel *'Representation of Events in Nerve Nets and Finite Automata'* waarin hij de vraag behandelde welke uitdrukkingen konden ontdekt worden door een neuraal net[4].

Uitgangspunt is een neuraal netwerk dat een stroom gegevens te verwerken krijgt, zoals een Moore-automaat, maar dat maar één uitvoerneuron heeft. De vraag is om te beschrijven onder welke omstandigheden de uitvoer de waarde 1 zal hebben. Het antwoord is dat de invoer moet voldoen aan een bepaalde regexp, die uiteraard afhangt van het net.

We kunnen het net opvatten als een detector. Juist zoals bij een Moore-automaat veronderstellen we een extra invoerkanaal voor het startsignaal. Dit startsignaal wordt gegeven

[4] McCullochs en Pitts claimden in hun artikel dat ze dit probleem hadden opgelost. Het stuk (Part III) waarin ze hum oplossing geven is echter zo goed als onbegrijpelijk. Zo merkt Kleene in zijn artikel op: "This [...] discouraged us from further attempts to decipher Part III". Overigens is het niet alleen bijna onbegrijpelijk, het staat ook vol fouten, en de gegeven oplossing is dan ook niet correct.

samen met de eerste invoer, en niet, zoals bij een Mooreautomaat, één tijdstap vroeger. Er zijn invoerkanalen, zoals bij een Moore-automaat, maar het is niet noodzakelijk dat de invoer alleen bestaat uit bitreeksen die exact één 1-bit bevatten. Zoals bij de Moore-automaten veronderstellen we dat een letter aan het netwerk gepresenteerd worden met één tijdstap tussenruimte (dus op de even tijdstippen $t = 0, 2, \ldots$); de oneven tijdstippen worden voorbehouden voor berekeningen[5]. Op het schema duiden we het inwendige niet aan. Het uitvoerneuron is altijd een OF-neuron, dat we apart getekend hebben. Het kan



Figuur 8.4. Een detector.

van één of meer inwendige neuronen tegelijkertijd een signaal krijgen. Als de detector een instantie van de regexp detecteert die begint op het ogenblik dat het startsignaal gegeven is, dan zal de detector een 1 als uitvoer hebben 2 tijdstappen na het voorkomen van de laatste letter (dus op het moment dat een volgende letter wordt aangeboden). Als de regexp geldig is voor een lege string, is er ook een directe verbinding die het startsignaal doorgeeft naar de uitvoer (getekend met een stippellijn in de diagrammen). De definitie van reguliere uitdrukkingen verloopt recursief. Voor een gegeven alfabet zijn de volgende uitdrukkingen regexps:

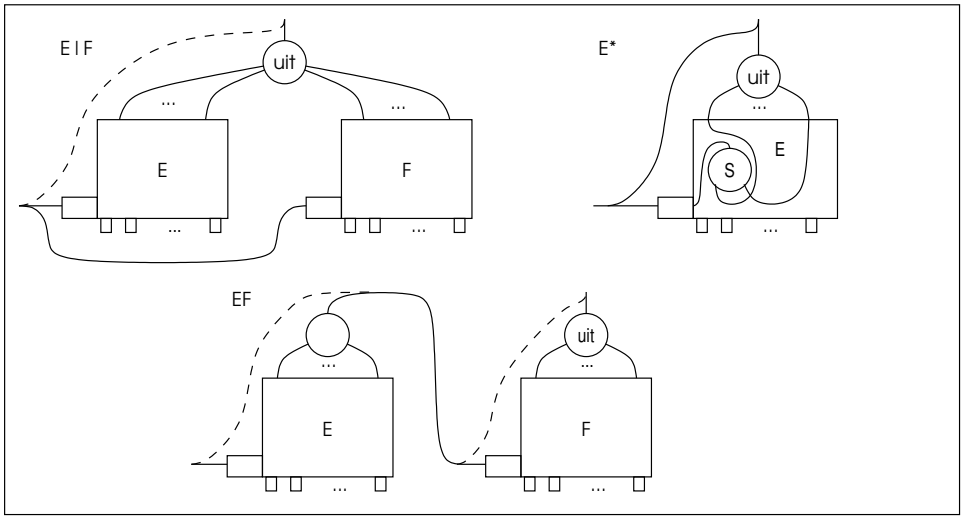
(1) De lege string, genoteerd λ.
(2) De strings bestaande uit één letter van het alfabet.
(3) Als E en F regexps zijn, dan is $E|F$ een regexp (gelezen *E of F*), evenals EF (gelezen *E voor F*, of gewoon als E F).
(4) Als E een regexp is, dan is $E*$ een regexp, de zogenaamde *Kleenesluiting* van E.

Bij veel praktische implementaties zijn er uitbreidingen, die tot deze gevallen kunnen herleid worden, zoals $E+$ voor $EE*$, en $E?$ voor $E|\lambda$.

Stelling 4 (Kleene) *Voor elke regexp over een gegeven alfabet kan een neurale detector geconstrueerd worden.*

[5] Ook hier is dit niet echt noodzakelijk: de constructie van Kleene zelf gebruikt maar 1 tijdstap per letter. Ze is echter behoorlijk ingewikkeld.

Bewijs
De detector voor λ wordt geconstrueerd door het startkanaal rechtstreeks door te verbinden met de uitvoer; deze voor een enkele letter door een selectieneuron, zoals reeds beschreven (het OF-neuron heeft dan slechts één ingang, en dient alleen voor synchronisatie). Als de detectoren voor E en F gekend zijn, dan kunnen deze voor $E|F$, $E*$ en EF geconstrueerd worden zoals getekend in Figuur 8.5. Merk op dat bij $E|F$we de twee uitvoerneuronen samenvoegen, en niet een nieuw neuron toevoegen: dit is nodig om de juiste synchronisatie te behouden. Bij $E*$ hebben we steeds een directe verbinding van start naar uitvoer,



Figuur 8.5. De detectoren voor $E|F$, $E*$ en EF.

omdat de lege string toegestaan is (dit is niet het geval bij $E+ = E*E$). We moeten hier ook een verandering aanbrengen binnen de detector: we voegen een neuron S toe (S staat voor schaduwneuron) dat de actie van het uitvoerneuron schaduwt. Het heeft dezelfde invoerverbindingen als dit neuron en dus ook dezelfde uitvoer, maar het stuurt deze terug naar het startkanaal.

∎

Kleene bewees ook het omgekeerde van de stelling: dat voor elke neurale detector er een regexp kan gevonden worden die de detector beschrijft. Het bewijs hiervan maakt essentieel gebruik van grafentheorie, en valt dus buiten het domein van deze cursus.

HOOFDSTUK 9
HET CLASSIFICATIEPROBLEEM

Zoals we gezien hebben staat het classificatieprobleem centraal in AI. Een zeer verwant probleem is dat van ordening. Voorbeeld: van schaakposities zijn een aantal parameters bekend (hoeveel pionnen zijn er voor elke kleur? hoeveel stukken staan niet verdedigd? ...). Welke positie is beter voor een speler dan andere?

Ter vereenvoudiging veronderstellen we hier dat het gaat om een classificatieprobleem met twee klassen. Dit is geen essentiële beperking, want een probleem met meer dan twee klassen kunnen we beschouwen als een combinatie van tweeklassenproblemen. Hebben we bijvoorbeeld drie klassen, A, B en C dan kunnen we classificatie uitvoeren door eerst het tweeklassenprobleem A of niet-A op te lossen, en daarna voor de niet-A het tweeklassenprobleem B of C op te lossen.

Traditioneel wordt een netwerk bestaande uit één neuron dat classificatie in twee klassen uitvoert een *perceptron* genoemd. Soms wordt de naam perceptron ook gebruikt voor meerlagige netten met binaire classificatie, maar dat doen we hier niet: een perceptron bestaat uit één enkel neuron.

Zulk een neuron (zie Figuur 8.1 op p. 97) deelt de n-dimensionale ruimte in twee stukken, die men halfruimten noemt ($n = 2$ halfvlak, $n = 1$ halfrechte). De scheiding tussen deze twee halfruimten is een verzameling van de vorm

$$\{\mathbf{x} : \mathbf{x} \cdot \mathbf{w} - T = 0\},$$

waarbij we veronderstellen dat de gewichtenvector $\mathbf{w}$ verschilt van nul. Dergelijke verzameling noemen we een hypervlak. Als $n = 3$ noemt men dit soms ook een vlak, als $n = 2$ soms wel een rechte, en bij $n = 1$ eventueel een punt.

9.1 HARDE CLASSIFICATIE

We spreken van harde classificatie als we werken met een TLU, en er dus slechts twee mogelijke uitvoerwaarden zijn. Is nu $\mathcal{L}$ (de leerverzameling) een verzameling van vectoren verdeeld in twee klassen, $\mathcal{L}^+$ en $\mathcal{L}^-$. We zeggen dat $\mathcal{L}$ lineair scheidbaar is als en slechts als er een halfruimte bestaat die alle punten van $\mathcal{L}^+$ bevat, en geen enkel van $\mathcal{L}^-$, terwijl er ook geen punten op de rand mogen liggen. Met andere woorden: er bestaat een vector $\mathbf{w}$ en een getal T zodanig dat voor alle $\mathbf{x} \in \mathcal{L}$ geldt

$$\mathbf{x} \cdot \mathbf{w} \begin{cases} > T & \text{als } \mathbf{x} \in \mathcal{L}^+ \\ < T & \text{als } \mathbf{x} \in \mathcal{L}^- \end{cases}$$

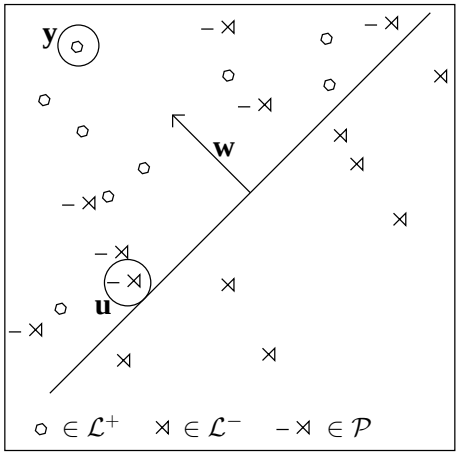
We kunnen dit nu ook anders formuleren: $\mathcal{L}$ is lineair scheidbaar als en slechts als er een neuron bestaat met n invoerkanalen, elk met gewicht w_i, en met drempel T zodanig dat de invoer van dit neuron strikt positief is als we een punt uit $\mathcal{L}^+$ aanbieden, en strikt negatief voor een punt uit $\mathcal{L}^-$. Als we voor zo'n neuron een polaire TLU nemen, dan is de uitvoer

dus $+1$ voor elementen uit $\mathcal{L}^+$, en -1 voor elementen uit $\mathcal{L}^-$. Dit is de eenvoudigste vorm van het perceptron, dit voor twee klassen.

Hoe laten we nu het perceptron leren wat de juiste gewichten zijn? We gebruiken hier een vorm van Hebbregel die –uitermate passend– de perceptronleerregel genoemd wordt. Om onze berekeningen te vergemakkelijken beginnen we met onze drempel T weg te werken door een extra dimensie toe te voegen. We maken dus weer een 0-de plaats in al onze vectoren, en zetten daar een 1; de T wordt nu vervangen door het gewicht w_0, met $T = -w_0$. We gebruiken nu nog altijd de inproductnotatie en de norm, maar nu voor de ruimte $\mathbb{R}^{n+1}$. We zoeken dus een vector $\mathbf{w}$ in $\mathbb{R}^{n+1}$ zodanig dat

$$\mathbf{x} \cdot \mathbf{w} > 0 \text{ als } \mathbf{x} \in \mathcal{L}^+ \qquad -\mathbf{x} \cdot \mathbf{w} > 0 \text{ als } \mathbf{x} \in \mathcal{L}^-.$$

We vervangen nu $\mathcal{L}$ door de verzameling $\mathcal{P}$. Deze bevat $\mathbf{x}$ als $\mathbf{x} \in \mathcal{L}^+$, en $-\mathbf{x}$ als $\mathbf{x} \in \mathcal{L}^-$. Dit vereenvoudigt ons probleem: als we een vector $\mathbf{x}$ uit $\mathcal{P}$ aanbieden, dan moet ons perceptron 1 als uitvoer hebben. Denk eraan dat we hier een neuron zonder drempel



Figuur 9.1. Lineair scheidingsprobleem

hebben, anders konden we gewoon de drempel bij $-\infty$ nemen om dit op te lossen. Ons leeralgoritme lijkt nu bijzonder goed op dat van het netwerk van McCulloch en Pitts:

initialiseer de gewichten (bijvoorbeeld op nul)
zolang (geen foutloos parcours en niet te veel pogingen){
  voor elke vector $\mathbf{x}$ uit $\mathcal{P}${
    als $\mathbf{x} \cdot \mathbf{w} \leq 0$
    vervang $\mathbf{w}$ door $\mathbf{w} + \mathbf{x}$
  }
}

De reden van deze vervanging is zeer eenvoudig: we weten dat $\mathbf{x} \cdot (\mathbf{w} + \mathbf{x}) > \mathbf{x} \cdot \mathbf{w}$. Immers het verschil is $\mathbf{x} \cdot \mathbf{x} = ||\mathbf{x}||^2$, en dit is groter dan nul. Om juist te zijn: het is zelfs minstens gelijk aan 1, want de eerste coördinaat is ofwel $+1$ ofwel -1.

Essentieel is dit algoritme hetzelfde als het algoritme voor het leren van een binaire functie $F : \{0,1\}^n \to \{0,1\}$. Inderdaad, daar hadden we twee klassen van bitstrings: zij die op 1 moeten worden afgebeeld, en zij die op 0 moeten worden afgebeeld. Daar gebruikten we de formule

$$\mathbf{w} \to (y - u)(1, b_1, \ldots, b_n).$$

Maar deze formule betekent hetzelfde als bovenstaande:

- Als de gewenste uitvoer $y = 1$, en de vector wordt verkeerd geklasseerd, dan is $(y - u) = 1$, en $(1, b_1, \ldots, b_n) \in \mathcal{P}$.
- Als de gewenste uitvoer $y = 0$, en de vector wordt verkeerd geklasseerd, dan is $(y - u) = -1$, en $-(1, b_1, \ldots, b_n) \in \mathcal{P}$.

Alleen konden bij binaire functies de coördinaten van de vectoren alleen 0 of 1 zijn, terwijl we hier werken met vectoren van reële waarden. Merk op dat we nu ook kunnen aangeven welke binaire functies met een eenlagig net kunnen gerealiseerd worden: Het zijn de functies die lineair scheidbaar zijn.

We hebben nu een leeralgoritme gegeven, maar we hebben nog niet aangetoond dat dit algoritme zinvol is. Immers, als we $\mathbf{w}$ vervangen door $\mathbf{w} + \mathbf{x}$ kan het zijn dat $\mathbf{x} \cdot (\mathbf{x} + \mathbf{w})$ nog altijd negatief is. Ook kan het zijn dat een vector $\mathbf{v}$ uit $\mathcal{P}$, die al correct geklasseerd was, $\mathbf{v} \cdot \mathbf{w} > 0$, nu verkeerd wordt gelezen. Immers, het zou best kunnen dat $\mathbf{x} \cdot \mathbf{v} < 0$, en dan is $\mathbf{v} \cdot (\mathbf{w} + \mathbf{x}) < \mathbf{v} \cdot \mathbf{w}$. We moeten dus de testverzameling $\mathcal{P}$ verschillende malen overlopen. Eindigt het dan nooit? De kracht van de methode volgt uit de volgende stelling:

Stelling 5 *Als er voor $\mathcal{P}$ een vector $\mathbf{w}^*$ bestaat zodanig dat $\mathbf{x} \cdot \mathbf{w}^* > 0$ voor alle $\mathbf{x} \in \mathcal{P}$ –met andere woorden: $\mathcal{L}$ is lineair scheidbaar–, en $\mathcal{P}$ is een eindige verzameling, dan zal de perceptronleerregel na een eindig aantal stappen een $\mathbf{w}$ vinden die het probleem oplost, en dus $\mathbf{x} \cdot \mathbf{w} > 0$ voor alle $\mathbf{x}$ in $\mathcal{P}$.*

Bewijs
We beginnen met alle gewichten op nul te zetten, en nummeren de keren dat we een verkeerd geklasseerde vector tegenkomen. De vector die bij de k-de keer hoort noemen we $\mathbf{x}(k)$ (dezelfde vector kan natuurlijk verschillende keren in de rij voorkomen). De gewichtenvector waar we mee beginnen noemen we $\mathbf{w}(0)$, deze die we hebben na de k-de stap $\mathbf{w}(k)$. Volgens ons algoritme is dus $\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{x}(k)$ voor $k > 0$. We weten dat $\mathbf{x}(k) \cdot \mathbf{w}(k-1) \leq 0$, anders zouden we $\mathbf{w}$ niet aanpassen. Bijgevolg groeit de norm van $\mathbf{w}(k)$ niet zeer snel. Immers

$$\begin{aligned} ||\mathbf{w}(k)||^2 &= (\mathbf{w}(k-1) + \mathbf{x}(k)) \cdot (\mathbf{w}(k-1) + \mathbf{x}(k)) \\ &= \mathbf{w}(k-1) \cdot \mathbf{w}(k-1) + 2\mathbf{w}(k-1) \cdot \mathbf{x}(k) + \mathbf{x}(k) \cdot \mathbf{x}(k) \\ &\leq ||\mathbf{w}(k-1)||^2 + ||\mathbf{x}(k)||^2 \\ &\leq \ldots \\ &\leq ||\mathbf{x}(1)||^2 + \ldots + ||\mathbf{x}(k)||^2 \end{aligned}$$

Immers, de middelste term op de tweede lijn is negatief. Nu is $\mathcal{P}$ eindig, en dus is er een vector $\mathbf{y}$ in $\mathcal{P}$ met grootste norm, $||\mathbf{y}||^2 = M$, en $||\mathbf{x}||^2 \leq M$ voor alle $\mathbf{x}$ in $\mathcal{P}$. We hebben dus

$$||\mathbf{w}(k)||^2 \leq kM.$$

Anderzijds gaat $\mathbf{w}(k)$ altijd meer en meer op $\mathbf{w}^*$ lijken, met andere woorden: de cosinus tussen $\mathbf{w}(k)$ en $\mathbf{w}^*$ wordt meestal groter als k groter wordt. Hoe tonen we dit aan? Eenvoudig: $\mathbf{x} \cdot \mathbf{w}^* > 0$ voor alle $\mathbf{x}$ in $\mathcal{P}$. Als we beginnen met $\mathbf{w}(0) = 0$ krijgen we

$$\mathbf{w}(k) = \mathbf{x}(1) + \mathbf{x}(2) + \cdots + \mathbf{x}(k)$$

Omdat $\mathbf{x}\cdot\mathbf{w}^* > 0$ op $\mathcal{P}$ en omdat $\mathcal{P}$ eindig is, is er een vector $\mathbf{u}$ in $\mathcal{P}$ waarvoor dit inproduct het kleinste wordt: $\mathbf{u}\cdot\mathbf{w}^* = m > 0$, en $\mathbf{x}\cdot\mathbf{w}^* \geq m$ voor alle $\mathbf{x}\in\mathcal{P}$. We krijgen

$$\begin{aligned}\mathbf{w}(k)\cdot\mathbf{w}^* &= \mathbf{x}(1)\cdot\mathbf{w}^* + \cdots + \mathbf{x}(k)\cdot\mathbf{w}^* &(9.1)\\ &\geq m + \cdots + m = km. &(9.2)\end{aligned}$$

Uit de ongelijkheid van Cauchy ($\mathbf{w}(k)\cdot\mathbf{w}^* \leq \|\mathbf{w}(k)\|\,\|\mathbf{w}^*\|$), die we kwadrateren, volgt dan

$$||\mathbf{w}(k)||^2 \geq \frac{k^2m^2}{||\mathbf{w}^*||^2}.$$

Bijgevolg kan k maar beperkt groeien. Inderdaad, uit de twee ongelijkheden voor $||\mathbf{w}(k)||^2$ volgt dat

$$kM \geq \frac{k^2m^2}{||\mathbf{w}^*||^2},$$

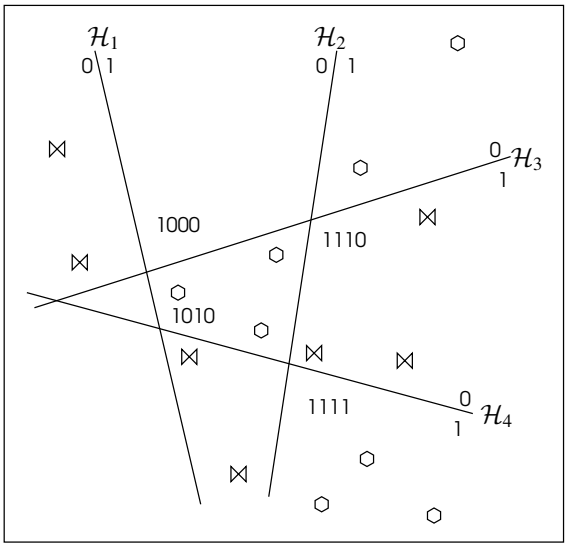
en dus dat $M||\mathbf{w}^*||^2/m^2 \geq k$. Het is dus onmogelijk dat we meer dan $M||\mathbf{w}^*||^2/m^2$ keer onze gewichtenvector moeten aanpassen. ∎

Natuurlijk, meestal weten we niet of ons probleem lineair scheidbaar is. Een eenvoudig tegenvoorbeeld waar het niet lukt: $\mathcal{L}^+ = \{(0,1),(1,0)\}$ en $\mathcal{L}^- = \{(1,1),(0,0)\}$. Bovendien kennen we in onze schatting voor het aantal stappen de grootheid $||\mathbf{w}^*||/m$ niet: moesten we $\mathbf{w}^*$ kennen zouden we geen leermethode moeten toepassen. In de praktijk stopt men dus na een zeker aantal stappen, waarbij het best mogelijk is dat een aantal punten verkeerd geklasseerd worden. Voor problemen die men niet lineair kan scheiden gebruikt men vaak meerlagige netwerken, of ook stemmachines, die in volgende hoofdstukken aan bod komen.

Wel kunnen we aantonen dat we, gegeven een willekeurige leerverzameling $\mathcal{L}$ waarvoor $\mathcal{L}^+$ en $\mathcal{L}^-$ geen gemeenschappelijke elementen hebben, een drielagig net kunnen construeren dat een scheiding van $\mathcal{L}$ kan uitvoeren. Er is een groot verschil met de situatie bij McCulloch en Pitts. Een enkele TLU kan de ruimte in twee delen verdelen. Bij McCulloch en Pitts was dit voldoende om een selectiefunctie te hebben, omdat de invoervectoren binair waren: bij elke mogelijke bitcombinatie die als invoer kan dienen kunnen we een hypervlak construeren dat dit bitpatroon van de andere afscheidt. Hier werken we echter met reële vectoren, en dan is zo'n afscheiding met één hypervlak niet meer mogelijk. Vandaar dat we een drielagig netwerk in plaats van een tweelagig netwerk nodig hebben.

Stelling 6 *Is een willekeurige eindige leerverzameling $\mathcal{L}$ gegeven, zo dat $\mathcal{L}^+ \cap \mathcal{L}^- = \emptyset$. Dan is er een drielagig net dat de classificatie kan doorvoeren.*

Bewijs
We weten al dat we voor een willekeurig hypervlak een neuron kunnen opbouwen dat als uitvoer langs de ene kant 0 geeft, en aan de andere kant 1. Als we een hele verzameling hypervlakken nemen (stel met k elementen), dan verdelen deze de hele ruimte in vakjes. Elk vakje kunnen we aanduiden met een binair getal met k cijfers: we nemen gewoon een willekeurige vector in het vakje, en kijken wat de uitvoer is van de k neuronen. Elke vector in datzelfde vakje geeft immers dezelfde uitvoer. We kunnen er nu voor zorgen dat geen enkele vector van $\mathcal{L}$ op een van de hypervlakken ligt, en dat alle elementen van $\mathcal{L}$ die in een gegeven vakje zitten tot dezelfde klasse (dus ofwel $\mathcal{L}^+$, ofwel $\mathcal{L}^-$) behoren. Inderdaad, desnoods zorgen we ervoor dat rond elk punt van $\mathcal{L}$ een klein kubusje zit dat alleen dat ene punt bevat. De k neuronen vormen de eerste laag van ons netwerk. We nemen nu een



Figuur 9.2. Niet-lineaire scheiding met drielagig net.

binaire functie $F : \{0,1\}^k \to \{0,1\}$ die gedefinieerd is als volgt:

$$F(b_1,\ldots,b_k) = \begin{cases} 1 & \text{als er een element van } \mathcal{L}^+ \\ & \text{in het vakje met binair getal } b_1 \ldots b_k \text{ zit} \\ 0 & \text{als er een element van } \mathcal{L}^- \\ & \text{in het vakje met binair getal } b_1 \ldots b_k \text{ zit} \end{cases} .$$

Als het overeenkomstige vakje leeg is, kunnen we kiezen of de waarde 0 of 1 moet zijn. Er is nu zeker een tweelagig netwerk van TLU's dat deze functie realiseert. We gebruiken de uitvoer van onze eerste laag als invoer voor dit netwerk, dat duidelijk het classificatieprobleem voor $\mathcal{L}$ oplost. ∎

9.2 ZACHTE CLASSIFICATIE

Als we werken met TLU's is de uitvoer van ons net altijd 0 of 1. In een aantal gevallen is dit niet gewenst, en hebben we liever een continue uitvoer die ook tussenliggende waarden kan aannemen.

- Meetwaarden die aan een neuraal net ter classificatie worden aangeboden bevatten dikwijls meetfouten. Daarom is de classificatie van punten dicht bij het scheidende hypervlak twijfelachtig. Meestal werkt men dan met een net dat uitvoerwaarden in het interval $[-1,1]$ geeft. Uitvoerwaarden die dicht bij 0 liggen worden dan als onzeker geklasseerd.
- Soms willen we een ordening doorvoeren. Als we bijvoorbeeld schaak spelen willen we de mogelijke zetten niet indelen in twee klassen (goede en slechte zetten), maar willen we een ordening maken van zeer slechte tot zeer goede zetten.

Als voorbeeld gebruiken we het schaakprobleem. We willen een neuraal netwerk ontwerpen dat, gegeven een aantal mogelijke zetten, ons vertelt wat de beste zet is. Op deze

manier krijgen we een evaluatiefunctie die we dan weer kunnen gebruiken in een spelboom met alfa-betasnoeien.

Om de beste zet te bepalen moeten we alle posities waar we naartoe kunnen gaan ordenen. Hoe doen we dit? We analyseren een groot aantal gespeelde partijen. We weten natuurlijk dat in realiteit spelers niet altijd de best mogelijke zet doen: anders zou elke schaakpartij dezelfde uitkomst hebben.

Immers, de minimaxboom voor schaken bestaat in principe, ook al is hij veel te groot om hem expliciet uit te rekenen. Bijgevolg is er een optimaal pad of misschien zelfs meerdere. Het is dus best mogelijk dat perfecte spelers op bepaalde ogenblikken verschillende opties hebben, maar elk optimaal pad eindigt met dezelfde uitslag.

We kunnen dus slechts een schatting maken van de kwaliteit van een positie aan de hand van de resultaten van het spel. Vermits we het resultaat van de partij kennen weten we welke zetten we "goed" moeten noemen: deze van de winnaar. Maar omdat we niet de volledige spelboom analyseren hebben we geen exacte evaluatie van de positie, maar alleen een benadering. Hoe verder we van het einde van de partij verwijderd zijn, hoe groter de kans op fouten. Nabij het einde van het spel wordt onze schatting beter: we geven dus een grote waarde aan spelen dicht bij het einde. We definiëren dan ook de kwaliteit $q(P)$ van een positie P in een partij als volgt:

1. 0 als de partij in remise eindigde;
2. $1/n$ als de speler die juist gezet had wint. Hier is n het aantal zetten van de positie tot het einde van de partij.
3. $-1/n$ als de speler die juist gezet had verliest, met dezelfde betekenis voor n.

Merk op dat het best mogelijk is dat een positie in verschillende partijen voorkomt, met verschillende kwaliteit. Dit is het soort gegevens dat klassieke computerprogramma's tot waanzin drijft, terwijl een neuraal netwerk er niet te veel last mee heeft.

Anderzijds moeten we een aantal metingen hebben, zodat we aan P een vector kunnen hechten. Typische maten zijn:

1. Het aantal pionnen, lopers, ..., van elke kleur.
2. Hoeveel velden er kunnen aangevallen worden.
3. Hoeveel stukken van elke kleur gedekt zijn.
4. ...

Om te voorkomen dat we een drempel in ons neuron nodig hebben zorgen we er voor dat de 0-de meting altijd 1 oplevert. In dit geval leveren de meeste metingen een positief geheel getal op, maar dit is niet echt belangrijk. We nemen aan dat er in totaal $n + 1$ metingen zijn. Op deze manier gaan we dus van een positie P over naar een meetvector $\mathbf{v}$, waarbij het mogelijk is dat verschillende posities leiden tot dezelfde meetvector.

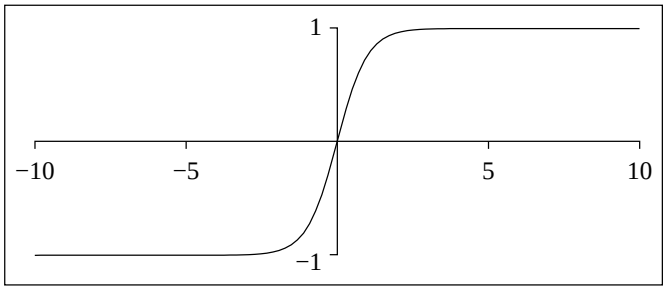
Merken we hier op –en dit is een algemeen kenmerk van klasserende netten– dat er een algemeen ontwerpprobleem is voor de invoer. Bij elk reëel probleem moeten we een passende verzameling van meetgegevens vinden. Het kan zijn dat we een vooropgegeven verzameling hebben van metingen (en dan moeten we ons afvragen welke we gaan gebruiken) of, zoals bij ons schaakprobleem, dat we zelf een verzameling meetwaarden moeten opstellen. De prestaties van een net kunnen soms sterk verbeterd worden door andere metingen in te voeren. Merk op dat het bij een perceptron geen kwaad kan om *te veel* metingen op te nemen. Als een of ander meetgegeven niets te maken heeft met de classificatie zal het perceptron dit zelf ontdekken en het overeenkomende gewicht gewoon op nul zetten. Wel is het zo dat het toevoegen van niet-relevante meetgegevens het leerproces vertraagt.

9.3 DE DELTAREGEL

Voor de eerste versie van de deltaregel gaan we uit van een netwerk dat bestaat uit één neuron, dat een analoog neuron moet zijn. We hebben nu een leerverzameling $\mathcal{L}$ die bestaat uit een aantal meetvectoren $\mathbf{v}$, samen met een gewenste uitvoer $y(\mathbf{v})$. We veronderstellen dat de gewenste uitvoerwaarden liggen tussen -1 en $+1$, en dus kunnen we een neuron nemen met een reactiefunctie die tussen deze waarden ligt, bijvoorbeeld

$$f(r) = \frac{e^r - e^{-r}}{e^r + e^{-r}}.$$

Het is belangrijk dat de functie strikt stijgend is en overal een afgeleide heeft, en dat deze



Figuur 9.3. Reactiefunctie met extrema -1 en $+1$.

verschilt van nul. De bovenstaande functie, die overigens de hyperbolische tangens heet, wordt dikwijls gebruikt omdat de afgeleide gemakkelijk te berekenen is. We duiden de afgeleide aan met f' en krijgen

$$\begin{aligned} f'(r) &= \frac{e^r + e^{-r}}{e^r + e^{-r}} - \frac{(e^r - e^{-r})^2}{(e^r + e^{-r})^2} \\ &= 1 - (f(r))^2. \end{aligned}$$

Hoe trainen we nu ons netwerk? Hiervoor gebruiken we de deltaregel[1].

We nemen een meetvector $\mathbf{v}$, en voeren deze in ons netwerk in. Het resultaat is een uitvoer u, waarvan we aannemen dat hij verschillend is van $y(\mathbf{v})$: er is dus een uitvoerfout $e = y(\mathbf{v}) - u$. Eerst berekenen we nu de fout op de invoer van ons neuron (dit is dus een vorm van terugvoeren).

De invoer van het neuron was $a = \sum_i w_i v_i$, met $f(a) = u$. Bij benadering geldt voor kleine waarden van δ dat

$$f(a + \delta) \sim f(a) + \delta f'(a).$$

Nemen we dus $\delta = e/f'(a)$, dan wordt $f(a + \delta) \sim y(\mathbf{v})$. Nu voeren we wijzigingen in de gewichten door die de fout opheffen: we vervangen $\mathbf{w}$ door $\mathbf{w} + \Delta\mathbf{w}$ zodanig dat

- De nieuwe invoer van het neuron gelijk is aan $a + \delta$.
- De wijziging zo klein mogelijk is, $||\Delta\mathbf{w}||$ is minimaal.

[1] de Griekse letter delta (kleine letter: δ, hoofdletter: Δ) wordt in de wiskunde heel dikwijls gebruikt voor kleine verschillen. De deltaregel is dus 'de regel van de kleine verschillen'.

We krijgen dus dat $(\mathbf{w} + \Delta\mathbf{w}) \cdot \mathbf{v} = a + \delta$, en bijgevolg dat $\Delta\mathbf{w} \cdot \mathbf{v} = \delta$. Schrijven we nu $\Delta\mathbf{w}$ als

$$\Delta\mathbf{w} = A\mathbf{v} + \mathbf{y},$$

met A reëel en $\mathbf{y}$ loodrecht op $\mathbf{v}$, dus $\mathbf{y} \cdot \mathbf{v} = 0$. A is uniek bepaald, want $A\mathbf{v} \cdot \mathbf{v} = \delta$, waaruit volgt dat $A = \delta/||\mathbf{v}||^2$. Hoe groot is $\mathbf{y}$? We weten dat $||\Delta\mathbf{w}||^2 = A^2||\mathbf{v}||^2 + ||\mathbf{y}||^2$. De kleinst mogelijke waarde krijgen we bij $\mathbf{y} = 0$, en dus stellen we uiteindelijk

$$\Delta\mathbf{w} = \frac{y(\mathbf{v}) - u}{f'(a)||\mathbf{v}||^2}\mathbf{v}. \tag{9.3}$$

Merk op dat dit zinvol is, want $||\mathbf{v}||^2$ is nooit nul (de eerste component is 1) en $f'(a)$ is ook nooit nul. Vergelijking (9.3) noemen we de deltaregel voor dit neuron.

We kunnen de deltaregel nog op een andere, minder klassieke manier beschrijven. Gegeven is weer de invoervector $\mathbf{v}$. Stel dat we nu een kleine wijziging τ doorvoeren in een of ander gewicht w_i. Hoe wijzigt nu u, de uitvoer van ons netwerk? De invoer van het neuron wijzigt, en wordt $a + \tau v_i$ in plaats van a, zodat de uitvoer verandert in $f(a + \tau v_i) \sim f(a) + f'(a)\tau v_i$. De factor $f'(a)v_i$ is de *partiële afgeleide* van u naar w_i, die we ook noteren als $\partial_{w_i} u$. We kunnen al deze afgeleiden samenvoegen in een vector, die we als $\partial_{\mathbf{w}} u$ noteren, dus

$$\partial_{\mathbf{w}} u = (\partial_{w_0} u, \ldots \partial_{w_n} u).$$

In het geval van één enkel neuron dat we al behandeld hebben is gewoon $\partial_{\mathbf{w}} u = f'(a)\mathbf{v}$. Zodoende kunnen we de deltaregel voor dit net herschrijven[2]:

$$\Delta\mathbf{w} = \frac{e}{||\partial_{\mathbf{w}} u||^2}\partial_{\mathbf{w}} u. \tag{9.4}$$

Deze tweede vorm van de deltaregel kan ook gebruikt worden om een meerlagig net zonder terugkoppeling te laten leren. Inderdaad, in zo'n net kunnen we, voor elke tak afzonderlijk, een kleine wijziging in het gewicht doorvoeren en zien hoe de uitvoer verandert. Als we nu deze takken ook gewoon nummeren met één enkele index (en niet met twee zoals vroeger), dan is op deze manier elke $\partial_{w_i} u$ goed bepaald.

Het spreekt vanzelf dat deze regel alleen kan toegepast worden op een neuron met een continue uitvoer. Maar zelfs dan nog kan hij tot problemen leiden. Hij is immers gebaseerd op de regel $f(a + \delta) \sim f'(a) + \delta f'(a)$. Als nu de fout e groot is, dan wordt de δ die we nodig hebben ook groot, en dan gaat de regel niet meer op. Dit is zeker zo als $|a|$ zeer groot is. Een numeriek voorbeeldje: stel dat $a = 2.6477$ zodat $f(a) = 0.99$ en $f'(a) = 0.0199$. De uitvoer zou nul moeten zijn, zodat $e = -0.99$. Uiteraard moet dan, om dit te corrigeren, $\delta = -a$ worden. Maar $e/f'(a) = 49,75$ als we deze waarde aannemen krijgen we $f(a + \delta) = -1$ (op ettelijke cijfers na de komma): de fout is groter geworden dan ze was. Bij dit enkele neuron kan men, door expliciet de inverse functie te nemen, dit probleem omzeilen; bij meerlagige netwerken is dit echter niet gemakkelijk. Men wijzigt dan de regel dikwijls in

$$\Delta\mathbf{w} = A\frac{e}{||\partial_{\mathbf{w}} u||^2}\partial_{\mathbf{w}} u,$$

waar A de zogenaamde leerfactor is. Deze wordt dan vrij klein genomen. Dit kan het leren vertragen, maar vermijdt al te grote schommelingen. Hij is ook nuttig om te voorkomen dat het net door deze grote schommelingen te veel 'vergeet' wat het vroeger geleerd heeft.

[2] Eigenlijk is de formule afkomstig van de methode van de steilste helling (Eng.: method of steepest descent), die gebruikt wordt om parameters zo aan te passen dat een bepaalde functie een minimumwaarde bereikt. Men past de parameters aan in de richting waarin de afname het sterkst is, en die gegeven wordt door $\partial_{\mathbf{w}} u$.

Dit is bijvoorbeeld duidelijk in ons schaaknet. Stel dat een positie P in de leerverzameling voorkomt met verschillende kwaliteit. Als $A = 1$ dan zal alleen de laatste kwaliteit onthouden worden; voor $A < 1$ blijven de vorige waarden gedeeltelijk behouden.

HOOFDSTUK 10
STEUNVECTOREN

We hebben gezien dat lineaire separatieproblemen kunnen opgelost worden met een enkel neuron. Als we een niet-lineair probleem moeten oplossen, weten we dat er een meerlagig netwerk bestaat dat de scheiding kan realiseren. Maar de stellingen waar we op steunen geven niet aan hoe dit netwerk er moet uit zien: we weten niet hoeveel neuronen er nodig zijn in de eerste en de tweede laag van het netwerk.

In de praktijk zit er bovendien ruis op onze gegevens: niet alle gegevens zijn correct. Daarom is het erg belangrijk dat de oplossingsmethode die we gebruiken daarmee rekening houdt. Dit heeft twee aspecten. Ten eerste proberen we, als onze leerverzameling kan gescheiden worden door de gekozen methode, om die scheiding zo duidelijk mogelijk te maken. Bij lineaire scheiding eisen we zo dat het scheidend hypervlak zo ver mogelijk van de leerverzameling verwijderd is. Maar door de ruis kan het zijn dat we verkeerde waarden in onze leerverzameling zitten. Vandaar dat men toelaat dat een aantal punten verkeerd geklasseerd wordt: men neemt een netwerk van een zekere grootte, en probeert de gegevens zo goed mogelijk te klasseren: als een enkel punt van $\mathcal{L}^-$ in een grote groep van $\mathcal{L}^+$-vectoren ligt, dan wordt aangenomen dat dit een fout is en houdt men er weinig of geen rekening mee.

Er zijn verschillende manieren om deze twee punten te realiseren. Naast meerlagige netten, gebruikt men ook eenvoudige netten waarbij de neuronen speciale eigenschappen hebben. Deze speciale neuronen doen dan dienst als invoerneuronen. Dit ligt in de lijn van biologische netten, waar ook de invoerneuronen speciaal zijn aangepast aan hun taak. De vorm die we hier gaan bespreken, het zogenaamde steunvectorenmechanisme (SVM: **S**upport **V**ector **M**achine) is er een van. Oorspronkelijk zijn SVM's uitgevonden als alternatief mechanisme voor lineaire problemen. Later heeft men ingezien dat ze ook voor andere problemen kunnen gebruikt worden.

We bekijken hier alleen SVM's die de vorm hebben van een *stemmachine*. Bij deze vorm is het originele verband tussen SVM's en perceptrons niet meer duidelijk.

10.1 BASISPRINCIPES

Zoals gewoonlijk hebben we een leerverzameling $\mathcal{L} = \mathcal{L}^+ \cup \mathcal{L}^-$. We nummeren deze leerverzameling doorlopend,

$$\mathcal{L} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}.$$

Hierbij is het niet noodzakelijk zo dat alle elementen van $\mathcal{L}^+$ vooraan staan. Het zal handig zijn bij elke $\mathbf{x}_i$ een getal y_i te voegen, dat de waarde 1 heeft als $\mathbf{x}_i \in \mathbf{L}^+$ en de waarde -1 als $\mathbf{x}_i \in \mathbf{L}^-$. Op deze manier kunnen we bijvoorbeeld schrijven dat

$$\mathcal{L}^- = \{\mathbf{x}_i \in \mathbf{L} : y_i = -1\}.$$

Een stemmachine maakt gebruik van een *gelijkaardigheidsfunctie* g. Deze geeft aan hoe goed twee mogelijke items op mekaar lijken. Voor gegeven $\mathbf{x}$ en variërende $\mathbf{z}$ heeft

$g(\mathbf{x}, \mathbf{z})$ de grootst mogelijke waarde voor $\mathbf{z} = \mathbf{x}$; naarmate $\mathbf{z}$ minder op $\mathbf{x}$ begint te lijken neemt $g(\mathbf{x}, \mathbf{z})$ af. Een mogelijke gelijkaardigheidsfunctie is

$$g(\mathbf{x}, \mathbf{z}) = -\|\mathbf{x} - \mathbf{z}\|^2. \tag{10.1}$$

Het mag eigenaardig lijken dat de maximale waarde 0 is en dat de functie niet beperkt is naar onder toe, maar dat is geen probleem. Wel moeten we aannemen dat g symmetrisch is, $g(\mathbf{x}, \mathbf{z}) = g(\mathbf{z}, x)$ voor alle mogelijke $\mathbf{x}$ en $\mathbf{z}$.

We gaan nu ons classificatiesysteem eens op een andere manier bekijken. De basisidee is dat vectoren die meer lijken op vectoren uit $\mathcal{L}^+$ dan op vectoren uit $\mathcal{L}^-$ als positief moeten geklasseerd worden, en de andere als negatief. Een schema om dit uit te voeren gaat als volgt:

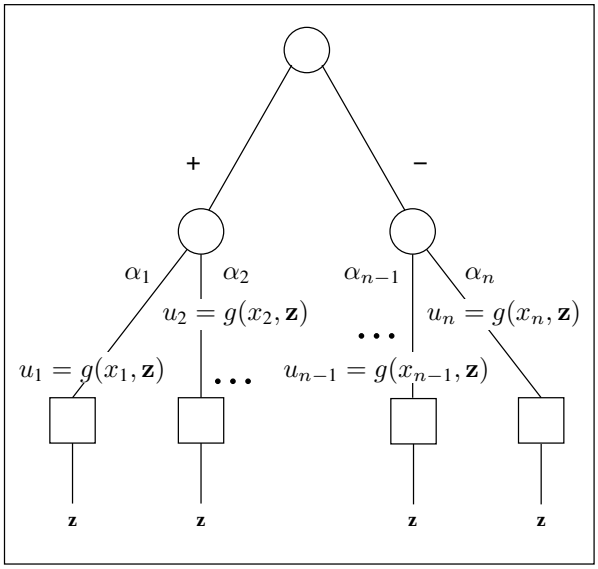
1. (Leerfase)
   Elke klasse geeft aan elk van zijn vectoren in $\mathcal{L}$ een gewicht dat aangeeft hoe belangrijk de vector voor de klasse is. Er is een beperking: de som van de gewichten (binnen een klasse) moet 1 zijn en de gewichten mogen niet negatief worden. Ook moet een drempelwaarde T voor de scheiding bepaald worden.
2. (Gebruiksfase)

2a. Voor een onbekende vector $\mathbf{z}$ zendt elke vector $\mathbf{x}_i$ in $\mathcal{L}$ een signaal uit dat aangeeft hoe sterk $\mathbf{z}$ op $\mathbf{x}$ gelijkt.

2b. De klasse telt deze signalen voor zijn klasse op, rekening houdend met de gewichten.

2c. $\mathbf{z}$ wordt toegewezen aan de klasse met het sterkste signaal.

We krijgen dus een neuraal netwerk met de vorm van Figuur 10.1. We kunnen elk



Figuur 10.1. Stemmachine.

element van $\mathcal{L}$ beschouwen als een invoerneuron dat, gegeven een item $\mathbf{z}$, een signaal uitstuurt dat aangeeft hoe goed $\mathbf{z}$ op $\mathbf{x}$ gelijkt. Beide klassen hebben een neuron dat de outputs van alle invoerneuronen voor die klasse verzamelt. In tegenstelling tot vroeger verzamelen deze neuronen enkel de signalen en hebben ze geen begrensde uitvoer; men kan ze beschrijven door de reactiefunctie $f(r) = r$. De gewichten noteren we als $\alpha_1, \ldots, \alpha_n$,

zodat de totale invoer van het uitvoerneuron kan geschreven worden als

$$a(\mathbf{z}) = \sum_{i=1}^{n} \alpha_i y_i g(\mathbf{x}_i, \mathbf{z}).$$

$\mathbf{z}$ wordt dus bij $\mathcal{L}+$ geklasseerd als $a(\mathbf{z}) \geq T$ en bij $\mathcal{L}^-$ als $a(\mathbf{z}) < t$.

De gewichten voldoen aan de randvoorwaarden

$$\sum_{y_i=1} \alpha_i = 1 \qquad \sum_{y_i=-1} \alpha_i = 1 \qquad \alpha_i \geq 0. \tag{10.2}$$

De vectoren $\mathbf{x}_i$ waarvoor $\alpha_i \neq 0$ heten de *steunvectoren* van het systeem.

In de leerfase moeten we de gewichten α_i en de drempelwaarde T bepalen. Hierbij spelen de twee grootheden

$$m^+ = \min_{y_i=1} a(\mathbf{x}_i) \qquad m^- = \min_{y_i=-1} a(\mathbf{x}_i)$$

een belangrijke rol. De regels zijn:

(1) De α_i horend bij $\mathcal{L}^+$ worden zo gekozen dat m^+ zo groot mogelijk is.
(2) De α_i horend bij $\mathcal{L}^-$ worden zo gekozen dat m^- zo klein mogelijk is.
(3) De drempelwaarde wordt vastgelegd op $(m^+ + m^-)/2$.

Het spreekt vanzelf dat de waarden van α_i horend bij $\mathcal{L}^-$ voorwaarde (1) beïnvloeden, terwijl ze zelf bepaald worden door voorwaarde (2). Deze wordt dan weer beïnvloed door de waaren van α_i horend bij $\mathcal{L}^+$, die bepaald worden door voorwaarde (1). Men kan aantonen dat deze circulariteit geen probleem is en dat er een optimale oplossing is die aan (1) en (2) voldoet. Deze oplossing is niet uniek, maar alle oplossingen leiden tot dezelfde functie $a(\mathbf{z})$.

Om een optimale oplossing te vinden maken we gebruik van de energiefunctie

$$\mathcal{F}(\alpha_1, \ldots, \alpha_n) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j g(\mathbf{x}_i, \mathbf{x}_j).$$

We beweren nu het volgende:

Stelling 7 *Voor waarden van $\alpha_1, \ldots, \alpha_n$ die een oplossing van ons probleem opleveren is $\mathcal{F}$ minimaal.*

Bewijs
We bekijken eerst alleen de α's die gehecht zijn aan een vector in $\mathcal{L}^+$. Wat gebeurt er met $\mathcal{F}$ als we een van de αs zouden veranderen? Daartoe moeten we de afgeleide berekenen. Neem k vast, zodanig dat $y_k = 1$. De termen in $\mathcal{F}$ die afhangen van α_k zijn

$$\alpha_k \left(\sum_{i \neq k} \alpha_i y_i g(\mathbf{x}_i, \mathbf{x}_k) \right) + \frac{1}{2} \alpha_k^2 g(\mathbf{x}_k, \mathbf{x}_k).$$

Dit is een gewone tweedegraadsveelterm in α_k, en dus is de afgeleide gemakkelijk te berekenen:

$$\begin{aligned} \partial_{\alpha_k} \mathcal{F} &= \left(\sum_{i \neq k} \alpha_i y_i g(\mathbf{x}_i, \mathbf{x}_k) \right) + \alpha_k g(\mathbf{x}_k, \mathbf{x}_k) \\ &= \left(\sum_{i} \alpha_i y_i g(\mathbf{x}_i, \mathbf{x}_k) \right) \\ &= a(\mathbf{x}_k). \end{aligned}$$

Als we nu gewichten willen veranderen aan de $\mathcal{L}^+$-kant, dan moet $\sum_{y_i=1} \alpha_i$ gelijk blijven aan 1. Als we een gewicht verkleinen (zeg α_k), dan moet er dus een ander zijn (zeg α_j) dat groter wordt. Bovendien moet α_k strikt positief zijn, want de gewichten mogen niet kleiner worden dan 0. Bovendien wordt $\mathcal{F}$ alleen maar kleiner als $\partial_{\alpha_j}\mathcal{F} < \partial_{\alpha_k}\mathcal{F}$. We kunnen dus $\mathcal{F}$ niet meer verkleinen langs de $\mathcal{L}^+$-kant als

1. $\partial_{\alpha_k}\mathcal{F}$ gelijk is voor alle k waarvoor $\alpha_k > 0$.
2. $\partial_{\alpha_j}\mathcal{F}$ is zeker niet kleiner als $\alpha_j = 0$.

Maar vermits $\partial_{\alpha_j}\mathcal{F} = a(\mathbf{x}_j)$ zijn dit juist de voorwaarden om m^+ zo groot mogelijk te maken. Inderdaad, het is vrij gemakkelijk in te zien dat, als we α_k met een term τ verminderen en dus α_j met een term τ doen toenemen, dat dan $a(\mathbf{x}_k)$ afneemt met een term $\tau\ (g(\mathbf{x}_k, \mathbf{x}_k) - g(\mathbf{x}_i, x_k))$ en dat $a(\mathbf{x}_j)$ toeneemt met dezelfde term. Onder de voorwaarden 1. en 2. hierboven is er geen verbetering van m^+ meer mogelijk. Immers, met die voorwaarden is m^+ gelijk aan $a(\mathbf{x}_k)$ voor alle vectoren $\mathbf{x}_k$ met $\alpha_k = 0$ en elke wijziging zou $a(\mathbf{x}_k)$ verminderen.

Een analoge redenering levert een analoog resultaat in voor $\mathcal{L}^-$. ∎

Deze stelling levert ons de mogelijkheid om een leeralgoritme te formuleren: we nemen een of andere beginwaarde voor de α's. Daarna maken we een lus zodanig dat elke uitvoering van de lus de waarde van $\mathcal{F}$ verkleint. Dit garandeert ons dat we dichter in de buurt van een oplossing komen. Als deze verbetering niet te klein is, komen we op de duur dicht genoeg bij de oplossing om te stoppen. In het volgende algoritme is n_+ het aantal vectoren in $\mathcal{L}^+$ en n_- het aantal vectoren in $\mathcal{L}^-$.

1. initialiseer de gewichten op zo'n manier dat ze aan de voorwaarden voldoen. Bijvoorbeeld: $\alpha_k = 1/n_+$ als $y_k = 1$, $\alpha_k = 1/n_-$ als $y_k = -1$.
2. zolang er een $\mathbf{x}_k \in \mathbf{L}^+$ is met $\alpha_k > 0$ en $a(x_k) > m^+$ of een $\mathbf{x}_s \in \mathbf{L}^-$ met $\alpha_s > 0$ en $a(x_s) < m^-$:
   a. zoek de x_j in $\mathcal{L}^+$ waarvoor $g(\mathbf{x}_j)$ minimaal is.
   b. zoek de x_k in $\mathcal{L}^+$ waarvoor $g(\mathbf{x}_k)$ maximaal is terwijl $\alpha_k > 0$.
   c. zoek de waarde van t, met $0 \leq t \leq \alpha_k$, die de kleinste waarde oplevert voor $\mathcal{F}$ als we α_j vervangen door $\alpha_j + t$, en α_k vervangen door $\alpha_k - t$.
   d. zoek de x_r in $\mathcal{L}^-$ waarvoor $g(\mathbf{x}_r)$ maximaal is.
   e. zoek de x_s in $\mathcal{L}^-$ waarvoor $g(\mathbf{x}_s)$ minimaal is terwijl $\alpha_s > 0$.
   f. pas α_r en α_s op analoge wijze aan.

Met de eindwaarden voor $\alpha_1, \ldots, \alpha_n$ kunnen we nu de drempelwaarde

$$T = \frac{(m^+ + m^-)}{2}$$

nemen, zodat de discriminatiefunctie gelijk wordt aan

$$a(\mathbf{z}) = \left(\sum_i y_i g(\mathbf{x}_i, \mathbf{z}) \right) - T. \tag{10.3}$$

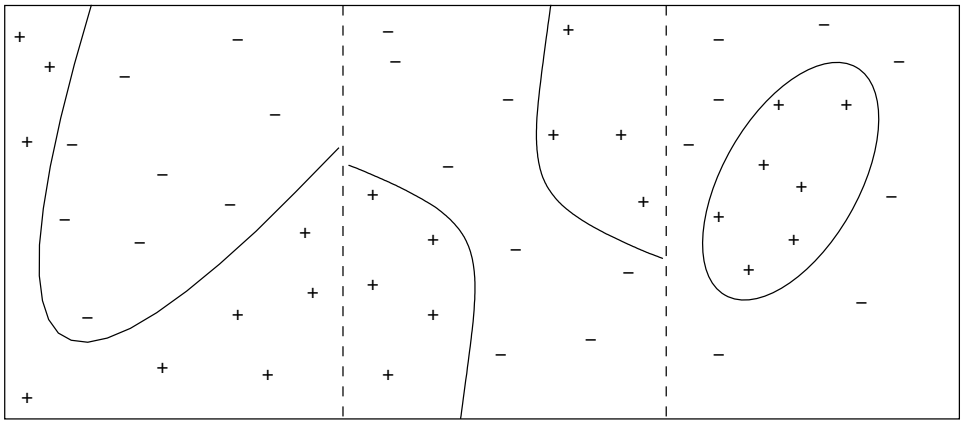
10.2 NIET-LINEAIRE CLASSIFICATIE

Zoals we gezien hebben kunnen niet alle classificatieproblemen worden opgelost met een

enkel neuron zonder de invoer te wijzigen. Als aanloop op het algemene geval bekijken we een iets algemenere klasse van problemen: de kwadratisch oplosbare.

Definitie 2 *Een verzameling $\mathcal{L} = \mathcal{L}^+ \cup \mathcal{L}^-$ heet kwadratisch scheidbaar als er een tweedegraadsveelterm p_2 bestaat zodanig dat $p_2(\mathbf{z}) > 0$ voor alle $\mathbf{z}$ in $\mathcal{L}^+$ en $p_2(\mathbf{z}) < 0$ voor $\mathbf{z}$ in $\mathcal{L}^-$.*

In Figuur 10.2 zien we enkele voorbeelden van kwadratisch scheidbare verzamelingen.



Figuur 10.2. Kwadratisch scheidbare verzamelingen.

Het spreekt vanzelf dat we er problemen zijn die we ook niet met een kwadratische functie kunnen oplossen, omdat er problemen zijn waar de scheiding een zeer grillige vorm heeft.

Kunnen we nu met een stemmachine dergelijke problemen behandelen? Laten we terugkeren naar de gelijkaardigheidsfunctie die we als voorbeeld hebben genomen, de functie uit (10.1) $g(\mathbf{x}, \mathbf{z}) = -\|\mathbf{x} - \mathbf{z}\|^2$. Vermits we deze ook kunnen schrijven als

$$g(\mathbf{x}, \mathbf{z}) = -\|\mathbf{x}\|^2 - \|\mathbf{z}\|^2 + 2\mathbf{x} \cdot \mathbf{z},$$

krijgen we voor de discriminatiefunctie $a(\mathbf{z})$ dat

$$\begin{aligned}
a(\mathbf{z}) &= \sum_{i=1}^{n} \alpha_i y_i g(\mathbf{x}_i, \mathbf{z}) \\
&= \sum_{i=1}^{n} \alpha_i y_i \left(-\|\mathbf{x}\|^2 - \|\mathbf{z}\|^2 + 2\mathbf{x} \cdot \mathbf{z}\right) \\
&= -\sum_{i=1}^{n} \alpha_i y_i \|\mathbf{x}_i\|^2 - \sum_{i=1}^{n} \alpha_i y_i \|\mathbf{z}\|^2 + 2\sum_{i=1}^{n} \alpha_i y_i \mathbf{x} \cdot \mathbf{z} \\
&= -\sum_{i=1}^{n} \alpha_i y_i \|\mathbf{x}_i\|^2 - \left(\sum_{y_i=1}^{n} \alpha_i\right) \|\mathbf{z}\|^2 + \left(\sum_{y_i=-1}^{n} \alpha_i\right) \|\mathbf{z}\|^2 + 2\sum_{i=1}^{n} \alpha_i y_i \mathbf{x} \cdot \mathbf{z}.
\end{aligned}$$

De eerste som uit de laatste lijn is duidelijk onafhankelijk van $\mathbf{z}$. De som in de tweede term is gelijk aan 1, evenals die in de derde term, zodat beide termen elkaars tegengestelde zijn. Alleen de laatste term hangt af van $\mathbf{z}$ en die term is lineair. Dit toont twee dingen:

(1) Met de gelijkaardigheidsfunctie uit (10.1) kunnen we alleen lineair scheidbare problemen oplossen.
(2) Als we het formalisme hadden gebruikt met de functie $g(\mathbf{x}, \mathbf{z}) = 2\mathbf{x} \cdot \mathbf{z}$ hadden we met dezelfde waarden voor $\alpha_1, \ldots, \alpha_n$ een discriminatiefunctie bekomen die, op een constante na, dezelfde was als die voor de functie uit (10.1). Door de drempel T dan met dezelfde constante aan te passen krijgen we daarmee dezelfde classificatie.

In veel gevallen hoort er bij een gelijkaardigheidsfunctie zo'n tweede functie, de zogenaamde *kernfunctie*. Vaak is de kernfunctie gemakkelijker te gebruiken dan de gelijkwaardigheidsfunctie, ook al is het bij zo'n functie niet direct duidelijk wat de maat voor gelijkaardigheid is. Zo kan men aantonen dat men met de kernfunctie

$$K_2(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^2$$

alle kwadratisch scheidbare problemen kan oplossen en ook alleen deze, zelfs al is de functie zelf van de vierde graad.

Sommige problemen zijn ook niet kwadratisch scheidbaar. In het vlak bijvoorbeeld kunnen we als scheidingslijn enkel een tweedegraadskromme hebben, dus een cirkel, een ellips, een hyperbool of een parabool. Als de punten heel onregelmatig verdeeld zijn kan het zijn dat we een veelterm van hogere orde nodig hebben: een derdegraads-, een vierdegraads-, een k-degraadskromme, of zelfs een heel andere vorm van functie. De afbeeldingsfunctie g kan hier zeer ingewikkeld worden, maar de kernfunctie blijft vrij eenvoudig. Zo gebruikt men voor een k-degraadskromme de kernfunctie

$$K_k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^k.$$

Een andere kernfunctie die vaak gebruikt wordt is

$$K(\mathbf{x}, \mathbf{z}) = e^{-||\mathbf{x}-\mathbf{z}||^2}.$$

Bij deze functie is het verband met gelijkaardigheid terug duidelijk.

HOOFDSTUK 11
ASSOCIATIEVE GEHEUGENS

Een klassiek computergeheugen is adresgestuurd: men geeft een adres op en men krijgt de inhoud van de geheugenplaats terug. Dit adres moet correct zijn, of men krijgt een totaal verkeerde inhoud terug.

Bij een associatief geheugen daarentegen geeft men een gedeelte van de inhoud op en krijgt men de daarmee geassocieerde volledige inhoud terug. Door de associatie is deze manier van werken veel robuuster qua input.

Dit geheugen speelt bij de mens (en bij dieren) een belangrijke rol. Het ophalen van een herinnering verloopt typisch op deze manier. Opvallend is overigens dat geuren hierbij een belangrijke rol spelen: men spreekt van het olfactorisch (geurgebonden) geheugen. In het algemeen is het zo dat het meemaken van een bepaalde situatie herinneringen oproept aan eerder meegemaakte, gelijkaardige situaties, inclusief de handelingen die het subject daarbij heeft gesteld en de resultaten daarvan. In dit verband kunnen we verwijzen naar de experimenten van Pavlov.

Ivan Petrovitsj Pavlov (1849-1936), Russisch fysioloog. Zijn bekendste experiment ging over geconditioneerde (aangeleerde) reflexen met honden, het zogenaamde hond-van-Pavloveffect. Bij dit experiment kreeg een hond, telkens vlak voor hij eten kreeg, een bel te horen. Het bleek dat na enige tijd de hond begon te kwijlen telkens hij de bel hoorde, zelfs als hij geen eten kreeg.

Dit illustreert de typische werking van een associatief geheugen. Bij de hond van Pavlov wordt de invoer ‘bel’ aangevuld met het gegeven ‘voedsel’, dat in het associatief geheugen wordt bewaard, en daar gekoppeld is met het belsignaal. Het gedeeltelijke patroon ‘bel’ wordt op deze manier aangevuld tot het volledige patroon ‘bel+voedsel’, zelfs in het geval waar de hond geen prikkels krijgt die rechtstreeks met voedsel samenhangen.

Maar een associatief geheugen zorgt niet alleen voor eenvoudige associatie, maar is ook de basis van abstractie. Het is het systeem waarmee we tot algemene begrippen komen. Nemen we het begrip ‘paard’. Er zijn massa’s beesten die paarden kunnen genoemd worden, maar deze zijn alle verschillend. Een associatief geheugen kan echter uit waarnemingen van dieren (paarden en andere dieren) tot een algemeen beeld komen, en op deze manier het begrip ‘paard’ inhoud geven.

11.1 HOPFIELDNETTEN

Een model voor een biologisch associatief geheugen wordt gegeven door zogenaamde Hopfieldnetten. In hun eenvoudigste vorm kunnen deze een binair patroon opslaan. De neuronen in een Hopfieldnet zijn dan ook TLU’s, en ze hebben drempelwaarde nul.

Bij een Hopfieldnet zijn er geen aparte in- en uitvoerneuronen. We gaan uit van een bitpatroon dat evenveel bits heeft als er neuronen zijn in het net. Elk neuron wordt in een toestand gebracht waarin het als uitvoer de overeenkomstige bitwaarde heeft. Daarna

duiden we willekeurige neuronen aan voor herberekening, en we gaan zo door tot we een stabiele toestand bereiken. Deze toestand is dan de geassocieerde eindtoestand. Het is een toestand die 'goed lijkt' op de begintoestand. In het voorbeeld van onze hond zijn er twee stabiele toestanden: 'geen bel en geen eten' en 'bel en eten'; als hij de bel hoort gaat hij over naar de tweede toestand.

Het is in deze context handiger om polaire neuronen (uitvoer -1 of $+1$) te gebruiken dan klassieke binaire (uitvoer 0 of 1), maar dit is dan weer niet zo handig bij bitpatronen. We zullen er in het vervolg altijd van uitgaan dat een 0 in een bitpatroon overeenkomt met een uitvoer -1 van het bijbehorende neuron, en een bitpatroon soms ook schrijven met plus- en mintekens ($(+++--+)$ in plaats van $(111001)_2$, bijvoorbeeld).

Om de analyse te vereenvoudigen veronderstellen we dat een neuron alleen kan omkippen (uitvoer $+1$ naar uitvoer -1, of omgekeerd) als we het aanwijzen. Anders blijft een neuron altijd dezelfde uitvoer geven. Bij een Hopfieldnet zijn alle neuronen verbonden met alle andere neuronen, behalve met zichzelf, en bovendien zijn de gewichten symmetrisch, dus

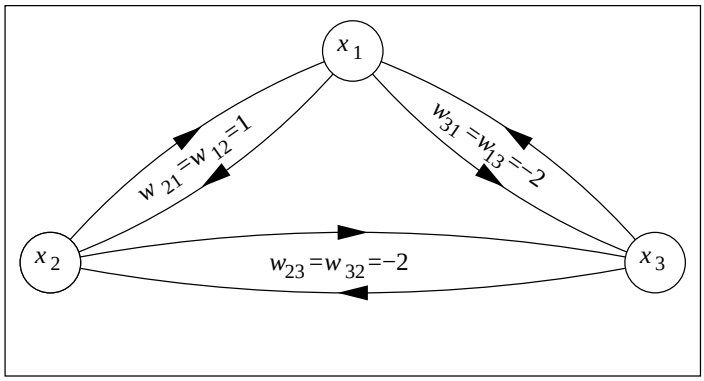
$$w_{ij} = w_{ji} \text{ en } w_{ii} = 0.$$

11.1.1 Een voorbeeld

Bekijken we de werking van zo'n net aan de hand van een voorbeeldje met drie neuronen, x_1, x_2 en x_3, en met gewichtenmatrix

$$W = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 0 & -2 \\ -2 & -2 & 0 \end{pmatrix}.$$

De toestand beschrijven we door de uitvoer van de drie neuronen als een binair getal te

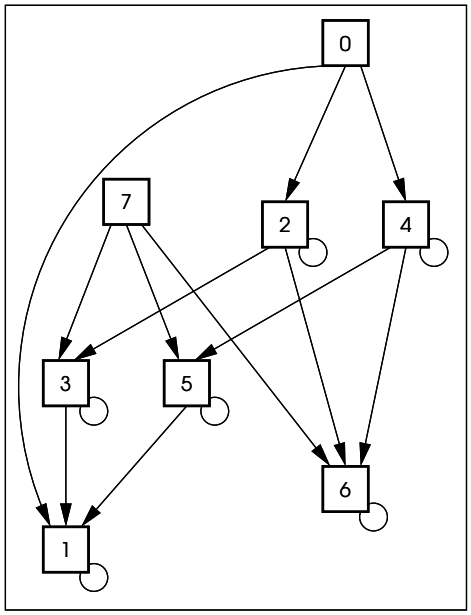


Figuur 11.1. Hopfieldnet met drie neuronen.

beschouwen. Zo is de toestand $5 = (+ - +)$ die toestand waarbij x_1 en x_3 uitvoer 1 hebben, en x_2 uitvoer -1. Wat gebeurt er nu als we in deze toestand x_2 aanwijzen voor verandering? De invoer voor x_2 is $1 \cdot 1 - 2 \cdot 1 = -1$, zodat x_2 dezelfde uitvoer houdt, en het net in toestand 5 blijft. We kunnen nu een tabel maken van de mogelijke overgangen:

toestand	aangewezen neuron		
	x_1	x_2	x_3
0=(– – –)	4	2	1
1=(– – +)	1	1	1
2=(– + –)	6	2	3
3=(– + +)	3	1	3
4=(+ – –)	4	6	5
5=(+ – +)	1	5	5
6=(+ + –)	6	6	6
7=(+ + +)	3	5	6

Wat hier opvalt is dat er twee toestanden zijn (6 en 1) die stabiel zijn: welk neuron we hier ook aanduiden, nooit zal het omkippen. Als we vertrekken uit een andere toestand zullen we op den duur altijd in een van de twee eindtoestanden 1 of 6 terecht komen. Merk op dat het bij bepaalde toestanden van de volgorde van het omkippen van neuronen afhangt in welke van de twee we aankomen. Het diagram van mogelijke overgangen zien we in Figuur 11.2. De twee stabiele toestanden 1 en 6 komen overeen met de twee patronen



Figuur 11.2. Overgangsdiagram van het netwerk van Figuur 11.1.

die het netwerk heeft opgeslagen. Als we een begintoestand opleggen en het netwerk laten evolueren gaat men naar een van de herinnerde patronen, en wel datgene dat het beste lijkt op het beginpatroon.

11.1.2 Algemene Hopfieldnetten

Natuurlijk kunnen we bij dit kleine netwerk direct zien wat er allemaal gebeurt. Als we het hebben over een groter netwerk met bijvoorbeeld duizend neuronen, dan zijn er 2^{1000} toestanden en kunnen we geen overgangstabel meer tekenen. We weten dus niet of er wel stabiele toestanden zijn, en nog veel minder of we er vanuit elke beginpositie een bereiken.

Om toch nog te zien wat er gebeurt definiëren we de *energie* van het netwerk. Als er n neuronen zijn wordt de energie gegeven door

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}u_i u_j.$$

Bekijken we dit even van dichtbij. Het product $u_i u_j$ is altijd gelijk aan ± 1. We hebben volgende gevallen:

- Als w_{ij} positief is, dan probeert de verbinding ervoor te zorgen dat u_i en u_j hetzelfde teken krijgen. De toestand voor die verbinding is dan normaal als $u_i u_j = +1$ (dit levert een negatieve bijdrage aan de energie) en vreemd als $u_i u_j = -1$ (met een positieve bijdrage aan de energie). Hoe groter de waarde van w_{ij}, hoe meer het resultaat als vreemd dan wel als normaal wordt beschouwd.
- Als w_{ij} negatief is moeten we alles omdraaien: tegengestelde waarden voor u_i en u_j zijn normaal, gelijke waarden vreemd.

De energie van het net is dus een maat voor hoe vreemd de toestand is van het net, en het net zal zoveel mogelijk proberen de vreemde uitvoeren recht te trekken.

Duiden we nu een neuron x_k aan voor herberekening. Er zijn twee mogelijkheden. De eerste is dat het neuron, en dus ook het net, dezelfde toestand behoudt: de energie blijft dan uiteraard gelijk. De tweede is dat het neuron omkipt. Wat is nu de invloed van het omkippen van x_k op de energie? We schrijven de termen die afhangen van u_k apart en houden rekening met het feit dat W symmetrisch is, $w_{ij} = w_{ji}$ en dat $w_{kk} = 0$. We krijgen

$$\begin{aligned} E &= -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n} w_{ij}u_i u_j - \frac{1}{2}\sum_{j} w_{kj}u_k u_j - \frac{1}{2}\sum_{i} w_{ik}u_i u_k \\ &= -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n} w_{ij}u_i u_j - u_k\sum_{i\neq k} w_{ik}u_i. \end{aligned}$$

De verandering van het energieniveau door het omkippen van x_k is dus

$$E(\text{na}) - E(\text{voor}) = (u_k(\text{voor}) - u_k(\text{na}))\sum_{i\neq k} w_{ik}u_i.$$

Maar de factor $\sum_{i\neq k} w_{ik}u_i$ is niets anders dan de invoer van x_k. Er zijn weer twee mogelijkheden:

1. Het neuron kipt om van -1 naar $+1$. Dit gebeurt alleen als de invoer van het neuron groter dan of gelijk aan nul is, $\sum_{i\neq k} w_{ik}u_i \geq 0$. anderzijds geldt dat $u_k(\text{voor}) - u_k(\text{na}) = -2$. De energie is dus afgenomen of, in een randgeval, gelijk gebleven.
2. Het neuron kipt om van $+1$ naar -1. In dit geval is de invoer kleiner dan nul, in symbolen $\sum_{i\neq k} w_{ik}u_i < 0$. Anderzijds geldt dat $u_k(\text{voor}) - u_k(\text{na}) = 2$. De energie neemt dus af.

We zien dus dat de energie nooit toeneemt, en bijna altijd afneemt telkens als er een neuron omkipt. De energie kan enkel gelijk blijven als er een neuron omkipt van -1 naar $+1$. Bijgevolg, als er een neuron omkipt vermindert ofwel de energie, ofwel het aantal minenen, (ofwel allebei). Bijgevolg kan het systeem nooit teruggaan naar een toestand waarin het al geweest is. Uiteindelijk moet het dus wel in een stabiele toestand terechtkomen (er is een eindig aantal toestanden). Als voorbeeld geven we de energiewaarden voor het voorbeeldnet:

toestand	energie	toestand	energie
0=(– – –)	6	7=(+ + +)	6
1=(– – +)	-10	6=(+ + –)	-10
2=(– + –)	2	5=(+ – +)	2
3=(– + +)	2	4=(+ – –)	2

Samenvattend kunnen we dus zeggen dat een Hopfieldnet een groot aantal toestanden heeft. Een aantal daarvan is stabiel: welk neuron we ook aanduiden, het zal niet omkippen. Deze toestanden hebben *lokaal* minimale energie, maar deze hoeft niet voor alle stabiele toestanden dezelfde te zijn. Als we uit een andere toestand vertrekken komen we, nadat we genoeg neuronen hebben aangeduid, in een van de stabiele toestanden terecht, en wel deze die goed 'lijkt' op de begintoestand: deze toestand is de herinnering die hoort bij de begintoestand.

Het is zo dat de energie van een gegeven toestand dezelfde is als die van zijn negatie, zodat een Hopfield net altijd patronen onthoudt in symmetrische paren. Dit is duidelijk in ons voorbeeld, waar de twee stabiele toestanden elkaars complement zijn.

Om dit probleem op te lossen kan men een extra neuron toevoegen dat altijd uitvoer $+1$ heeft, en dat nooit wordt aangeduid om te veranderen. Dit komt overeen met de techniek die we gebruikt hebben om de drempelwaarde bij TLU's te simuleren.

11.2 LEREN BIJ HOPFIELDNETTEN

Leren bij een Hopfieldnet is vrij eenvoudig en gebeurt volgens de regel van Hebb. Als we de kans willen vergroten dat een bepaald patroon $U = (u_1, \ldots, u_n)$ als uitvoer voorkomt, dan verminderen we gewoon de energie van dit patroon. We veranderen de gewichten "in de richting" van het product van de uitvoeren:

$$w_{ij} \rightarrow w_{ij} + u_i u_j \text{ voor } i \neq j.$$

Er staat een minteken in de formule voor de energie, dus daarmee verkleinen we de energie voor dit patroon.

De wijziging in de energie van U is gemakkelijk te berekenen:

$$\begin{aligned} E_{\text{NIEUW}}(U) &= -\frac{1}{2}\sum_{i,j} u_i u_j (w_{ij} + \Delta w_{ij}) = E_{\text{OUD}}(U) - \frac{1}{2}\sum_{i,j} u_i u_j \Delta w_{ij} \\ &= E_{\text{OUD}}(U) - \frac{1}{2}\sum_{i,j} 1 = E_{\text{OUD}}(U) - \frac{n^2 - n}{2} \end{aligned}$$

Maar ook voor patronen die goed op U lijken daalt de energie. Berekenen we de energieverandering voor een ander patroon, $V = (v_1, \ldots, v_n)$, waarbij we veronderstellen dat V in k bits verschilt van U. Weer geldt dat

$$E_{\text{NIEUW}}(V) = -\frac{1}{2}\sum_{i,j} v_i v_j (w_{ij} + \Delta w_{ij}) = E_{\text{OUD}}(V) - \frac{1}{2}\sum_{i,j} v_i v_j \Delta w_{ij}.$$

nu geldt dat

$$v_i v_j w_{ij} = \begin{cases} 1 & \text{als } v_i = u_i \text{ en } v_j = u_j & (n-k)(n-k-1) \text{ gevallen} \\ 1 & \text{als } v_i = -u_i \text{ en } v_j = -u_j & k(k-1) \text{ gevallen} \\ -1 & \text{als } v_i = -u_i \text{ en } v_j = u_j & k(n-k) \text{ gevallen} \\ -1 & \text{als } v_i = u_i \text{ en } v_j = -u_j & (n-k)k \text{ gevallen.} \end{cases}$$

Optellen geeft

$$E_{\text{NIEUW}}(V) = E_{\text{OUD}}(V) - \frac{(2k-n)(2k-n)-n}{2}.$$

We zien dat de energiewijziging het grootst is als $k = 0$ en afneemt als k groter wordt en het patroon minder op U begint te lijken. De energiewijziging is minimaal als $k \sim n/2$. Dat de energiewijziging terug toeneemt als k nog groter is, is normaal: denk eraan dat voor een Hopfieldnet een patroon V dezelfde energie heeft als het patroon $-V$.

Keren we terug naar ons voorbeeld met drie neuronen. Als we dit netwerk het patroon $3 = (- + +)$ willen aanleren, dan verhogen we de gewichten van de verbindingen tussen het tweede en het derde neuron met 1, en verminderen alle gewichten van verbindingen van en naar het eerste neuron met 1. We krijgen dan de gewichtenmatrix

$$W = \begin{pmatrix} 0 & 0 & -3 \\ 0 & 0 & -1 \\ -3 & -1 & 0 \end{pmatrix}.$$

Merk op dat het patroon 3 hiermee *geen* stabiel patroon geworden is: de stabiele patronen zijn nog altijd 1 en 6. Het Hopfieldnet is dus zijn vorige herinneringen niet vergeten, maar wel heeft het patroon 3 nu een lagere energie. Moesten we het net dit patroon herhaaldelijke malen laten leren, zou het uiteindelijk wel een stabiel patroon worden.

We zouden een 1-neuron x_0 kunnen toevoegen, zoals we bij het perceptron gedaan hebben. Dit neuron duiden we nooit aan voor herberekening, zodat u_0 altijd 1 blijft. De verbindingen vanuit dit neuron (en formeel gezien ook deze naar het neuron, maar dit heeft geen praktisch belang) leren mee op dezelfde manier als de andere. Elk patroon U dat we gebruiken heeft dan als steeds als eerste waarde $u_0 = 1$.

Zonder het toevoegen van u_0 komen stabiele patronen altijd voor in paren: als U stabiel is dan is $-U$ het ook. Door het toevoegen van u_0 is dit niet meer het geval: als U een stabiel patroon is dan is $-U$ ongeldig. Maar het is nog altijd wel zo dat, als we een patroon U aanleren, er patronen die zijn weinig verschillen van $-U$ en wiens energie veel afneemt. Dit is niet gewenst. We kunnen dit voorkomen door, in plaats van één extra neuron dat altijd uitvoer 1 heeft, er evenveel in te voeren als er normale neuronen zijn in het net. Als n het totaal aantal neuronen is (en dus $n = 2m$ met m het aantal bits in de oorspronkelijke patronen) dan is het aantal verschillende bits tussen twee geldige patronen U en V, dat we hierboven k noemden, hoogstens gelijk aan m en dus is $k \leq n/2$. Als zo een net een patroon U aanleert dan zal de energieafname van een toegelaten patroon V dus een dalende functie zijn van het verschil tussen U en V.

We bekijken nu twee toepassingen van Hopfieldnetten. Bij de tweede toepassing gaat het om *patroonvervollediging*, bij de eerste om classificatie zonder supervisie.

11.3 ASSOCIATIEVE GROEPERING

In realiteit gebruikt men grotere Hopfieldnetten, of andere associatieve geheugens (biologische netten hebben niet exact de vorm van een Hopfieldnet, maar hun werking lijkt er wel op). Het zal dan bijna nooit voorkomen dat dezelfde invoer herhaalde malen voorkomt in het leerproces. De stabiele eindtoestanden zijn dan een soort gemiddelde van groepen invoertoestanden die bij elkaar horen. Bijgevolg kunnen we Hopfieldnetten gebruiken om te groeperen zonder supervisie.

Deze methode is deze die het meest lijkt op de clustering zoals ze uitgevoerd wordt bij biologische netwerken. We zetten eventuele continue meetwaarden om naar een bitpatroon zoals later zal beschreven worden, zodat we een binair associatief geheugen kunnen

gebruiken. Verder verbinden we deze TLU's tot een associatief geheugen. Ons classificatiealgoritme gaat nu als volgt:

1. Zet alle gewichten van het associatief geheugen op nul.
2. Train het associatief geheugen op de gebruikelijke wijze met de gehele verzameling van punten.
3. Houdt nu de gewichten van het associatief geheugen vast en presenteer weer alle punten. Punten die dezelfde uitvoer geven worden in dezelfde groep gestoken.

Merk op dat het hier zinloos is om de leerverzameling herhaald te presenteren omdat er geen sprake is van fouten en de gewichten bij elk leerobject worden aangepast.

In een iets meer gesofisticeerde versie van het algoritme wordt elk punt in de derde fase verschillende malen aan het netwerk gepresenteerd, terwijl de neuronen in willekeurige volgorde worden aangeduid voor omkippen. Hierdoor kan het gebeuren dat een punt bij verschillende presentaties verschillende uitvoer geeft. Zo'n punt wordt dan aangeduid als een grenspunt (het kan een grenspunt zijn tussen twee groepen, maar ook tussen drie of meer groepen). Daarna bekijkt men de groepen twee aan twee. Als twee groepen meer grenspunten gemeen hebben dan een bepaald vooropgegeven getal, dan worden ze samengevoegd. Dit komt overeen met de transitiviteitsregel die we vooropgesteld hadden bij klasseren zonder supervisie: als $\mathbf{x}$ zeer dicht bij $\mathbf{y}_1$ ligt, $\mathbf{y}_1$ zeer dicht bij $\mathbf{y}_2$ ligt, ..., en $\mathbf{y}_n$ zeer dicht bij $\mathbf{z}$ ligt, dan behoren $\mathbf{x}$ en $\mathbf{z}$ tot dezelfde groep.

Een zeer specifiek geval van associatieve groepering is het zogenaamde *winner-takes-all*netwerk dat ingebed wordt in een groter netwerk en dat dient om ambiguïteiten op te lossen. Dit maakt essentieel gebruik van analoge neuronen, en is dus geen Hopfieldnet, maar lijkt er wel sterk op.

Stel dat we n alternatieven hebben die elkaar uitsluiten. We nemen n analoge neuronen met uitvoerwaarden tussen 0 en 1 op zo'n manier dat het omringende netwerk ervoor zorgt dat de activatie van de neuronen een functie is van de waarschijnlijkheid van elk van de alternatieven, uitgaande van de informatie die aanwezig is in het net. Alle n neuronen zijn nu onderling verbonden met negatieve gewichten. Indien nu één alternatief een sterker signaal krijgt dan de andere, zal het door de negatieve verbindingen deze anderen hun activatie ontnemen, waardoor het eigen signaal sterker wordt. Uiteindelijk zal dit sterkste alternatief het enige zijn met een grote uitvoer, terwijl de andere neuronen bijna inactief zullen worden. Dergelijke netten kunnen gebruikt worden bij patroonherkenning. Hier hebben we n klassen voor de indeling, elk met een eigen herkenningsnetwerk, en we geven de n uitvoeren door aan een winner-takes-allnetwerk.

11.4 PATROONVERVOLLEDIGING

Bij bepaalde problemen krijgen we een gedeeltelijk patroon dat we moeten aanvullen. Dit is een bekend probleem uit de theorie van gegevensstructuren, waarbij dikwijls gegevens moeten opgezocht worden aan de hand van een sleutel. Bij AI gaat het echter om sleutels die niet exact gekend zijn. Een voorbeeld is beveiligingssoftware die in een beeld verkregen uit een bewakingscamera een gelaat moet halen en zo een persoon herkennen. Gegeven is een gedeeltelijk patroon (de afbeelding van het gelaat), gevraagd is dit te vervolledigen met de naam van die persoon. Dit kan niet op de klassieke manier gebeuren omdat de afbeelding van het gelaat niet exact op voorhand gekend is. Een tweede voorbeeld van toepassing vinden we in langetermijnweersvoorspellingen[1]. Laten we aannemen dat het

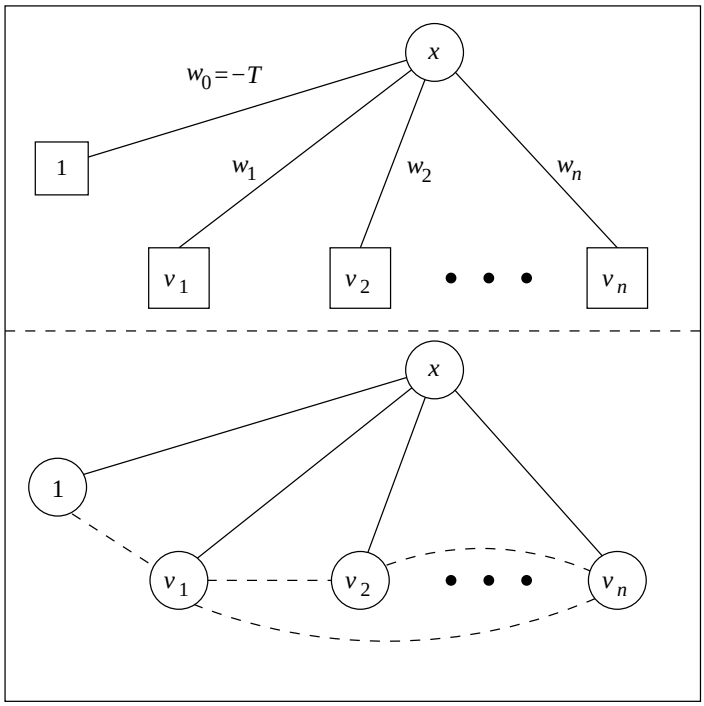
[1] Het voorbeeld is gekozen om educatieve redenen. Er is geen reden om aan te nemen dat het resultaat zinvol is.

einde maart is, en dat we een algemene voorspelling voor de maand april willen maken. Het gedeeltelijk patroon is het weer tot einde maart, en dit moet aangevuld worden met de voorspelling voor april. Merk op dat in ons eerste voorbeeld de aanvulling van het patroon een exact gegeven is, terwijl we in het tweede voorbeeld ook een vaag resultaat hebben.

Een Hopfieldnet werkt met binaire patronen. Bij dit soort problemen komen dikwijls meetwaarden voor, zoals hier de gemiddelde temperatuur, die nog moeten in een binair patroon gegoten worden. Dit moet natuurlijk op zo'n manier gebeuren dat gelijkaardige meetwaarden een gelijkaardig bitpatroon opleveren. Gaat het bijvoorbeeld om gehele getallen, dan moeten 7 en 8 bijna hetzelfde patroon vormen. Dit bereiken we niet met een gewone binaire voorstelling: in achtbitnotatie krijgen we de patronen $(00001000)_2$ en $(00000111)_2$ die niet echt op elkaar lijken. Beter is het om het aantal éénbits te laten toenemen met de meetwaarde. Nemen we voor de eenvoud aan dat elke parameter een waarde tussen 0 en 1 kan aannemen. Deze presenteren we dan aan k TLU's met één invoerkanaal (gewicht 1), en met drempelwaarde $1/k$, $2/k$, ..., 1 (deze veranderen nooit). Dit levert een bitpatroon op waarin het aantal 1-bits evenredig is met de sterkte van de invoer. In het geval $k = 10$ krijgen we dat bijvoorbeeld 0,7 en 0,8 genoteerd worden als $+ + + + + + + - --$ en $+ + + + + + + + --$, twee patronen die wel op elkaar lijken. Merk op dat de k neuronen een klein deelnet zullen vormen. De onderlinge gewichten zijn dan zo dat de stabiele toestanden de patronen zijn met $+$-en vooraan en $-$-en achteraan.

We hebben een probleem voor gesuperviseerd leren, en we moeten dus een verzameling hebben van complete patronen. Voor ons weerprobleem gebruiken we weersgegevens van een voorbije periode (de cijfers van het KMI gaan terug tot 1835, dus we hebben wel iets om op te bouwen). We delen de gegevens op in jaren, die we laten beginnen in mei en eindigen in april (april is de maand die we willen voorspellen, en dus is het zinvol deze als laatste maand te nemen). Laat ons zeggen dat we een jaar karakteriseren door voor elke maand de gemiddelde temperatuur, totale neerslag, aantal uren zonneschijn, enzoverder op te geven (hoe meer gegevens hoe beter). Deze gegevens worden gebruikt voor de leerfase zoals hierboven beschreven. Als we nu het weer voor april willen voorspellen, voeren we de gegevens van mei tot maart in, en laten het associatieve net werken. Het heeft geen zin om neuronen aan te duiden die te maken hebben met de maanden mei tot en met maart: die data zijn gegeven. We duiden dus alleen neuronen aan die te maken hebben met het ontbrekende deel van het patroon. Op deze manier wordt ons oorspronkelijke patroon aangevuld. We krijgen zelfs een maat voor de betrouwbaarheid van zo een voorspelling: als de energie van de voorspelling veel lager is dan die van gelijkaardige toestanden (dat is: met dezelfde gegevens van de voorgaande maanden, maar met een andere voorspelling voor de komende maand), dan is de voorspelling betrouwbaar. Als de energie voor verschillende voorspellingen nauwelijks verschilt, dan weten we dat het netwerk eigenlijk niets geleerd heeft.

Bij de bespreking van TLU's hadden we een leerregel opgesteld voor één neuron dat bitpatronen van lengte n afbeeldt op 1 of 0. Een alternatieve oplossing voor dit pobleem zou het gebruik van de bovenstaande techniek voor Hopfieldnetten kunnen zijn. Immers, we kunnen de invoervector $\mathbf{v}$ beschouwen als een gedeelte van het bitpatroon dat ook $f(\mathbf{v})$ bevat. Als we, zoals we in paragraaf 8.2 ook nog een 1 toevoegen aan het volledige patroon om een drempel te vermijden, dan is het volledige patroon $(1, \mathbf{v}, f(\mathbf{v}))$. Dit wordt gezocht als 1 en $\mathbf{v}$ gegeven zijn, en daarmee is het een probleem van patroonvervollediging geworden. Op het eerste gezicht gebruiken we een heel andere techniek bij een Hopfieldnet dan bij een netwerk van één neuron zoals in paragraaf 8.2. Het blijkt echter dat de twee technieken toch op mekaar lijken. Bekijken we hiervoor Figuur 11.3. Bovenaan hebben we het schema uit paragraaf 8.2 (Figuur 8.1 op pagina 97) nogmaals weergegeven, onderaan het overeenkomend Hopfieldnet. Als we werken met het Hopfieldnet en we hebben een nieuwe vector $\mathbf{z}$ die we willen klasseren, dan voeren we deze in het net in. Het enige neuron dat



Figuur 11.3. Classificator met één neuron en corresponderend Hopfieldnet.

bij het ontbrekende deel van het patroon hoort is gemarkeerd met x, en dit is dus het enige neuron dat we aanduiden voor herberekening. Daaruit volgt dat alleen verbindingen die leiden naar x (en hun gewichten) belang hebben. De andere zijn onbelangrijk: we hebben een aantal ervan aangeduid met stippellijnen. Zo blijkt dat de structuur van het perceptron is afgeleid uit een Hopfieldnet door alleen de relevante verbindingen te behouden.

Wat met leren? De leerregel van paragraaf 8.2 was gegeven door vergelijking (8.1): $w_i \rightarrow w_i + (y - u)v_i$. In ons Hopfieldnet veranderen de gewichten altijd, terwijl in paragraaf 8.2 alleen geleerd werd als $\mathbf{v}$ een verkeerde uitvoer kreeg, maar dit is het enige verschil. Merk op dat dit verschil kan opgevat worden als een modellering van *aandacht*. Als een mens leert aan de hand van voorbeelden zal hij meer aandacht besteden aan voorbeelden die door hem verkeerd geklasseerd worden. Hierdoor blijven deze voorbeelden langer in het Hopfieldnet, en beïnvloeden de gewichten meer dan voorbeelden die aan het verwachtingspatroon beantwoorden en waar dus snel wordt overgegaan.

HOOFDSTUK 12
KENNISREPRESENTATIE IN NEURALE NETTEN

In de vorige hoofdstukken hebben we neurale netten beschouwd als instrumenten voor classificatie. Het is echter duidelijk dat de mens zijn neuraal net gebruikt voor meer gesofisticeerde doelen, tot en met het het maken van redeneringen. In dit hoofdstuk gaan we dieper in op de grondslag ervan: de voorstelling van feiten en uitspraken. Immers, voor we kunnen te weten komen hoe een neuraal net uitspraken zoals *ooievaars eten kikkers* gebruikt in een redenering, moeten we weten in welke vorm zulk een uitspraak aanwezig kan zijn in het neurale net.

Taal kan beschouwd worden als een uitvinding van neurale netten om ideeën over te brengen van het ene neurale net naar het andere. Het wekt dan ook geen verwondering dat er een nauw verband is tussen de structuur van ideeën zoals ze in een neuraal net voorkomen en taalstructuren: zo komen types (die we verder definiëren) ongeveer overeen met substantieven, eigenschappen met adjectieven en de rollen van tokens met grammaticale rollen (onderwerp, lijdend voorwerp, ...). Alhoewel deze taalstructuren zeer algemeen voorkomen, zijn ze niet universeel. Zo heeft het Navaho nauwelijks substantieven en geen adjectieven, wat niet betekent dat men in het Navaho niet naar objecten zou kunnen refereren of geen eigenschappen van objecten beschrijven. De verbinding tussen eigenschap en type wordt in het Nederlands aangegeven door de adjectieven voor het substantief te zetten; in andere talen (Latijn bijvoorbeeld) wordt de verbinding door uitgangen gemaakt, en mogen adjectief en substantief verspreid zitten in een zin.

Om de gedachten te vestigen gaan we uit van een alwetende observator. We veronderstellen dat deze de volledige werking van het neurale net dat we bekijken kan beschouwen en doorgronden. De observator kan gebruik maken van notities: zo kan hij een etiket aanbrengen bij een neuron of een verbinding om aan te duiden waar dit element voor staat, en wat het juist doet. Maar deze etiketten maken geen deel uit van het net, en we moeten kunnen aangeven op basis van wat de observator bepaalt wat er op de etiketten moet staan. In dit verband moeten we wijzen op het verband met het semantisch net. Dit net was gebaseerd op de idee van neurale netwerken, maar het is duidelijk dat het semantisch net niet zomaar kan beschouwd worden als een model van kennisopslag in een neuraal net. Immers, we kunnen ons voorstellen dat de knopen van het semantisch net zouden overeenkomen met neuronen, maar essentieel aan het semantisch net is dat het een *geëtiketteerde* graaf is waarin de etiketten van de verbindingen een belangrijke rol spelen, o.a. bij het opzoeken van gegevens. Maar dergelijke etiketten zijn afwezig in een neuraal net. Om duidelijk te maken wat voor problemen dat met zich mee kan brengen: veronderstel dat je twee knopen in het net hebt met verschillende verbindingen (bijvoorbeeld: een visser *vangt* vissen, maar hij *verkoopt* ook vissen, en hij *eet* waarschijnlijk ook vissen). Als er zowel een visserneuron is als een visneuron, dan is het best mogelijk dat er drie synapsen zijn van het ene neuron naar het andere, maar we kunnen deze niet van elkaar onderscheiden.

Wat de *knopen* van het semantisch net betreft is er wel een rechtstreeks verband met

neurale netten. Met hersenscans heeft men kunnen aantonen dat het denken aan een bepaald begrip steeds een gelijkaardig activatiepatroon oplevert. Dit patroon is onafhankelijk van de manier waarop men het begrip onder de aandacht brengt: of men een proefpersoon een foto toont van een kat, hem het woord 'kat' laat lezen, of men hem het geluid van een kat laat horen: steeds krijgt men het zelfde patroon[1]. Dit leidt ons tot een eerste vaststelling: in ons neuraal net komt een knoop van het semantische net overeen met een (min of meer duidelijk afgebakend) geheel van neuronen, dat samen actief wordt als het overeenkomende concept actief voorkomt in een gedachte. Om bepaalde eigenschappen van semantisch net te kunnen verklaren (bijvoorbeeld het feit dat iemand die pas aan een object gedacht heeft gemakkelijk weer aan dit object denkt) is het nodig om de interne structuur van dit ensemble te bekijken. Voor onze doelstellingen is het echter voldoende te veronderstellen dat dit ensemble reageert als één neuron, en we zullen een knoop uit het semantisch net dan ook identificeren met een enkel neuron. Om duidelijk te maken dat we geen keuze maken tussen een enkel neuron of een ensemble zullen we echter meestal gewoon spreken van een knoop; voor zover we het onderscheid willen maken tussen het semantisch net en het neurale net spreken we dan van een semantische knoop of van een neurale knoop.

Dit leidt ons tot een eerste probleem: wordt elk begrip dat we kennen voorgesteld door één enkel neuron (of een samenwerkend ensemble), of geldt dit alleen voor enkelvoudige begrippen? Het lijkt logisch te veronderstellen dat er geen apart neuron is voor samengestelde begrippen, maar dan is de vraag welke begrippen basisbegrippen zijn die door één neuron worden voorgesteld. Is, bijvoorbeeld, het begrip 'grootouder' een samengesteld begrip, als combinatie van de begrippen 'grootvader' en 'grootmoeder', of is het omgekeerd 'grootmoeder' een samengesteld begrip, namelijk een 'vrouwelijke grootouder'? In dit verband mag aangehaald worden dat de theorie dat een enkelvoudig begrip overeenkomt met één neurale knoop bekend staat als *de theorie van het grootmoederneuron*, en dat er nog een andere mogelijke analyse is van 'grootmoeder' als niet-enkelvoudig begrip. In het Zweeds bijvoorbeeld bestaat er zelfs geen courant woord voor grootmoeder: men gebruikt de woorden *farmor* en *mormor* ('vaders moeder' en 'moeders moeder'). Aangenomen mag worden dat de grens tussen enkelvoudige begrippen met eigen neuronen en zuiver samengestelde begrippen zoals zowat alles bij neurale netwerken nogal vaag is.

In elk geval hebben we onze observator een manier gegeven hoe hij een neurale knoop die overeenkomt met een semantische knoop kan herkennen: hij laat het netwerk op verschillende manieren aan een begrip denken en noteert welk ensemble van neuronen altijd actief wordt. We mogen dus veronderstellen dat elk zo'n neurale knoop voorzien is van een etiket.

12.1 DENKEN EN WETEN

Het 'aanwezig zijn' van een uitspraak kan twee vormen aannemen:

- De uitspraak kan opgeslagen zijn in het neurale net.
- De uitspraak kan actief zijn in het net.

In het eerste geval gaat het om kennis die aanwezig is, ook op het ogenblik dat het net niets met deze kennis doet. Het gaat hier om *weten*; in het tweede geval gaat het om *denken*. In het vervolg gebruiken we de termen 'weten' en 'kennis' voor opgeslagen informatie, en 'denken' en 'gedachte' voor actieve informatie[2].

[1] Dit is niet helemaal correct: als men iemand een woord laten lezen wordt een leescentrum in de hersenen geactiveerd, als men een geluid laat horen het akoestisch centrum. Deze specifieke patronen moet men wegdenken.

[2] In het dagelijks taalgebruik is er bij het onderscheid tussen denken en weten het aspect van al-dan-niet correct

Er is uiteraard een verband tussen denken en kennis, en dit verband gaat in twee richtingen:

- Denken leidt tot weten.
- Weten leidt tot denken.

Bij het eerste gaan we ervan uit dat het (herhaaldelijk) denken van een bepaald feit leidt tot het opslaan ervan. Het is denkbaar, maar zeer onwaarschijnlijk, dat er een ingewikkeld mechanisme zou zijn in onze hersenen om deze opslag mogelijk te maken. Waarschijnlijker is echter dat de opslag gebeurt door een direct mechanisme dat beantwoordt aan de leerregels van Hebb, of een variant ervan. Merk op dat we enigszins vereenvoudigen, door de abstractie die deel uitmaakt van de opslag van het gegeven verwaarlozen. Zo zal een netwerk dat regelmatig geconfronteerd wordt met specifieke ooievaars die specifieke kikkers opeten uiteindelijk het feit *ooievaars eten kikkers* opslaan. Het mechanisme voor deze abstractie hebben reeds gezien toen we ongesuperviseerde classificatie bij Hopfieldnetten bekeken: daar werd er voor elke groep een archetype opgemaakt, dat het meest typische element van de groep aanduidde, en een soort gemiddelde van alle elementen van de groep was. Ook hier gebeurt iets analoogs: van een aantal zeer gelijkaardige gedachten wordt een archetypisch gemiddelde opgeslagen.

Het opslaan van een feit moet ertoe leiden dat dit feit gemakkelijk terug in gedachten kan gebracht worden. Zoals gebruikelijk bij een neuraal net gebeurt dit via associatie. Dit betekent dat een gedeelte van een feit genoeg moet zijn om het gehele feit tot een gedachte te maken. Merk op dat dit de typische manier is van vraagstelling aan het semantisch net, waaraan we vragen kunnen stellen zoals *wat doen ooievaars met kikkers?* of *wat eten ooievaars?* Aan de andere kant moeten we denken aan het fenomeen dat aangeduid wordt met de bijna onvertaalbare Engelse uitdrukking *suspension of disbelief*. Dit fenomeen doet zich voor als iemand tijdelijk dingen als waar aanneemt, terwijl hij weet dat ze niet waar kunnen zijn. Om in het in termen van kikkers en ooievaars uit te drukken: iemand die weet dat kikkers ooievaars opeten kan nog altijd denken dat ooievaars kikkers opeten.

12.1.1 Denkpatronen

In ons brein zijn een aantal synchronisatiemechanismen aanwezig, die ervoor zorgen dat de (des)activatie van neuronen synchroon verloopt. Deze synchronisatie zorgt voor de 'golven' die men ziet op een EEG, en waarvan de verschillende frequenties verband houden met verschillende activiteiten (zo worden de betagolven van 12-15 Hz gelinkt aan het denken). We kunnen denkactiviteit dan ook voorstellen als een opeenvolging van activatiepatronen. Dit betekent dat het tijdsmodel dat we besproken hebben waarbij elk neuron zijn uitvoer op tijdstip t bepaalt op basis van zijn invoer op tijdstip $t-1$ een goed beeld geeft van wat er in de hersenen gebeurt.

Nu kan een neuron verschillende niveau's van activatie hebben, maar in het vereenvoudigde model dat we opstellen gaan we ervan uit dat op een gegeven moment een neuron al dan niet actief is. Een activatiepatroon kan dan beschreven worden door de verzameling van actieve knopen, en in wat volgt zullen we een activatiepatroon voorstellen als de som van de etiketten van de actieve knopen, zoals bijvoorbeeld

'man'+'bijten'+'hond'.

Indien een neurale knoop overeenkomt met een semantische knoop is het etiket een woord

zijn. Zo zegt men dat dat iemand *weet* dat de aarde rond is, maar dat iemand anders *denkt* dat de aarde plat is. Dit onderscheid maken we niet: we zeggen dus dat iemand weet dat de aarde plat is als het als feit opgeslagen is in zijn hersens, en dat hij denkt dat de aarde rond is, als die gedachte bij hem opkomt. Ook gebruiken we 'feit' als synoniem voor 'uitspraak'. Ook een onwaar feit is dus een feit.

dat overeenkomt met het desbetreffende begrip. Strikt genomen is er natuurlijk geen 1-op-1-relatie tussen semantische knopen en woorden: één woord kan verschillende betekenissen hebben, en anderzijds bestaan er synoniemen. In de voorbeelden die we geven zullen er echter geen problemen zijn in die zin. Wel zullen we later nog neurale knopen tegenkomen die niet overeenkomen met semantische knopen. Hiervoor zullen we op het juiste moment een andere notatie invoeren, door gebruik van deze NOTATIE D.M.V. EEN ANDER LETTERTYPE.

12.2 *TYPES* EN *TOKENS*

Een eerste probleem dat we moeten aanpakken is de referentie naar objecten. Vaak is een uitspraak toepasselijk op een welbepaald object. Als we bijvoorbeeld een kat zien zitten, dan denken we *daar zit een kat*. Het gaat daarbij om één welbepaalde kat, namelijk de kat die we zien, en niet om een andere. Nu zal een neuraal netwerk misschien een aantal katten kennen, en dan heeft het een semantische knoop voor elk van die katten, maar het netwerk kan ook denken over een kat die het voor de eerste keer ontmoet. De referentie naar de kat in kwestie zal verbonden zijn met de semantische knoop 'kat'. Een semantische knoop die verwijst naar een of andere soort van objecten noemen we een enkelvoudig *type*. We zullen later samengestelde types definiëren; voorlopig zijn alle types enkelvoudig. De verwijzing naar een specifiek object noemen we een *token*[3]. Ons eerste probleem is om te bepalen wat zulk een token juist is.

Een object kan eigenschappen hebben. Een aantal eigenschappen kunnen vervat zitten in het type, maar dit wordt in het semantisch net voorgesteld door relaties met andere knopen, dus kunnen we ervan uitgaan dat elke typeknoop op zich eigenschapsloos is.

Het blijkt uit neurologisch onderzoek dat visuele eigenschappen van een bepaald object dynamisch verbonden worden zo dat ze samen geactiveerd worden. Om duidelijk te maken wat er juist gebeurt is het noodzakelijk even de beeldverwerking in onze hersenen te beschrijven. Deze verwerking gebeurt grotendeels in de *visuele cortex*. Deze vormt een meerlagig netwerk, en analyse heeft aangetoond dat we, als we van laag naar hogere laag gaan neuronen tegenkomen die coderen voor steeds meer abstracte en minder gelokaliseerde eigenschappen. Zo zijn er in de onderste lagen neuronen voor lokale randdetectie (deze worden actief als er een groot contrastverschil is tussen dicht bij elkaar liggende fotocellen), terwijl in hogere lagen er neuronen zijn die lijnstukken herkennen met een bepaalde oriëntatie, of bepaalde bogen. Het blijkt dat neuronen die coderen voor eigenschappen uit het beeld die bij elkaar horen, zoals lijnstukken die op elkaar aansluiten, boogdelen die op dezelfde cirkel liggen of elementen die in dezelfde richting en even snel bewegen, zich synchroniseren en gezamenlijk actief worden. Dergelijke synchronisatie wordt teruggevonden in de hersenen van bepaalde zoogdieren zoals katten, en is dus niet specifiek menselijk.

Als we ervan uitgaan dat dit mechanisme ook werkt op niveau van het eigenlijke denken, dan kunnen we veronderstellen dat een netwerk dat nadenkt over een bepaald object met bepaalde eigenschappen zowel de typeknoop van dat object activeert als de semantische neuronen die eigenschappen aanduiden. Op deze manier kunnen verschillende eigenschappen tegelijkertijd worden aangeduid: als het netwerk denkt aan een zwarte kat, dan is dit bepaald door het patroon

'zwart'+'kat'

. Merk op dat het hierbij moet gaan over eigenschappen die worden aangeduid door één enkel begrip, meestal geëtiketteerd door een bijvoeglijk naamwoord, en dat combinaties mogelijk zijn, vermits ze ondubbelzinnig zijn: het patroon

[3] Dit is een onvertaalbaar Engels woord, dat een object of een symbool aanduidt dat verwijst naar iets anders.

'klein'+'zwart'+'kat'

duidt zonder problemen een kleine zwarte kat aan. Een dergelijk patroon dat bestaat uit een type en een aantal eigenschappen noemen we een token, omdat het juist kan verwijzen naar een welbepaald object.

Op deze manier weten we ongeveer hoe we naar één geïsoleerd object kunnen verwijzen, maar wat gebeurt er als we aan twee objecten tegelijk denken, bijvoorbeeld als we denken *daar zit een zwarte kat naast een kleine hond*? Als een patroon het deelpatroon

'klein'+'zwart'+'kat'+'hond'

bevat, gaat dit dan over een kleine kat en een zwarte hond, of over een zwarte kat en een kleine hond? We zouden ons kunnen voorstellen dat er voor elk type-eigenschappaar, zoals KLEIN/HOND en ZWART/KAT er een knoop bestaat. dergelijke knoop is dan geen knoop in het semantisch net, vandaar de notatie. Een patroon dat gaat over een kleine zwarte hond en een grote zwarte kat zou het patroon

KLEIN/HOND+ZWART/KAT+GROOT/KAT+ZWART/HOND

omvatten. Nu is het mogelijk een neuraal netwerk te construeren dat volgens dit principe werkt door dergelijke knopen voor te programmeren, maar zou een dergelijk net capaciteitsproblemen hebben. Als we bijvoorbeeld duizend eigenschappen hebben en duizend types, dan hebben we nood aan een miljoen combinatieknopen. Bovendien lijkt het onwaarschijnlijk dat een lerend neuraal net met dit principe kan werken. Immers, als we een eigenschap willen toevoegen aan dit net, dan zouden we evenveel knopen moeten toevoegen als er types zijn, en veel van die extra knopen zouden combinaties coderen die nooit zullen voorkomen. Een gevolg van de klassieke leerregels is echter dat knopen alleen kunnen ontstaan als ze gebruikt worden.

Een derde probleem bij het gebruik van combinatieknopen duikt op als we werken met twee tokens van hetzelfde type. Nemen we als voorbeeld twee katten, zoals de gedachte *daar zit een witte kat naast een grote kat*? Vermits we maar één 'kat'-type hebben is het niet duidelijk, als ons activatiepatroon de combinatie GROOT/KAT+WIT/KAT bevat, of we het hebben over één enkele kat die zowel groot als wit is, dan wel over twee verschillende katten waarvan de ene groot en de andere wit is. Een verwant probleem is dat van de onderscheiden rollen die men heeft bij een uitspraak, en hiermee komen we terug bij het semantisch net. Een verbinding in het semantisch net heeft een begin- en een eindknoop, maar heeft ook een etiket. Dit etiket is vaak een werkwoord. We kunnen aannemen dat er ook voor dit etiket een semantische knoop bestaat, maar we kunnen niet de gedachte vervat in een verbinding van het semantisch net uitdrukken door gewoon de drie knopen te activeren. Als we het patroon

'ooievaar'+'eten'+'kikker'

zouden hebben, dan kan onze observator niet vaststellen of het netwerk denkt dat ooievaars kikkers eten, ofwel dat het denkt dat kikkers ooievaars eten. Net zoals we, als we twee tokens vermengen, niet meer weten welke eigenschap bij welk token hoort, weten we hier niet meer welke rol bij welke knoop hoort, en combinatieknopen bieden hier ook geen oplossing.

Het is mogelijk om een schema te vinden waar het aantal extraknopen dat wordt toegevoegd beperkt is. Al deze schema's werken op het principe waarbij elke semantische knoop twee niet-semantische tegenhangers heeft. Als we deze met A en B aanduiden (bijvoorbeeld KAT.A en KAT.B, GROOT.A en GROOT.B), dan is een patroon als

KAT.A + HOND.B + GROOT.A + PURPER.A + KLEIN.B + DOORSCHIJNEND.B

ondubbelzinnig te interpreteren als een dat zowel het token voor een grote purperen kat bevat, als van dat van een kleine doorschijnende hond. Dergelijk schema lost het probleem op dat we hadden met verschillende tokens van hetzelfde type, maar voldoet toch niet. Ten eerste is werkt het alleen voor een vooraf opgegeven aantal tokens (als we drie tokens in één activatiepatroon willen, moeten we zorgen voor een A, een B en een C-knoop voor elk begrip). Bovendien is het creëren van de .A- en .B-knopen niet in overeenstemming te brengen met klassieke leerregels. Tenslotte is er het probleem van alignatie van de twee knopen. Hiermee bedoelen we dat de .A- en de .B-knopen gealigneerd moeten zijn in die zin dat ze gelijkaardige relaties moeten hebben met gelijkaardige knopen: HOND.A moet zich verhouden tot KAT.B zoals HOND.B zich verhoudt tot KAT.A. (HOND.A en KAT.A zijn bijna zeker niet gerelateerd, vermits er geen reden is waarom deze samen zouden actief zijn). Als, bijvoorbeeld, het semantisch net weet dat katten bang zijn van honden, dan moet het neuraal net ergens hebben opgeslagen dat KAT.A bang is van een HOND.B, maar ook dat KAT.B bang is van een HOND.A. Maar als het denkt aan een kat die bang is van een hond, dan is het ofwel een KAT.A die bang is van een HOND.B, ofwel een KAT.B die bang is van een HOND.A. Bijgevolg wordt het bang-zijn opgeslagen ofwel bij het (KAT.A, HOND.B)-paar ofwel bij het (KAT.B, HOND.A)-paar, maar niet bij beide paren tegelijk.

Het is dan ook duidelijk dat het vermengen van types onoverkomelijke problemen geeft: we stellen dan ook als regel dat een activatiepatroon hoogstens één type mag bevatten. Indien een activatiepatroon een type bevat, dan is er een token dat bestaat uit dit type en alle eigenschappen die in het activatiepatroon zitten. We kunnen dit dan ook *het* token van het activatiepatroon noemen. Een gedachte die handelt over meer dan een type dient dus automatisch uit verschillende activatiepatronen te bestaan.

12.3 PREDIKATEN

Keren we nu terug naar het probleem van de verschillende rollen die een token kan hebben bij een gedachte. We hadden al vastgesteld dat het patroon

‘ooievaar’+‘eten’+‘kikker’

niet duidelijk maakt of de ooievaar de kikker opeet, ofwel omgekeerd. Vermits twee types niet samen mogen voorkomen, is het duidelijk dat de gedachte uit minstens twee patronen moet bestaan:

‘ooievaar’+‘eten’+...
‘eten’+‘kikker’+...

Het zou kunnen dat de knoop ‘eten’ maar in een van beide patronen aanwezig is, of zelfs alleen maar in een derde patroon, maar dat is niet echt relevant. Wat wel belangrijk is is dat we nog steeds niet kunnen vaststellen wie de eter en wie de gegetene is. Hiervoor zijn er twee verschillende mogelijkheden:

- De rol wordt duidelijk gemaakt door de *volgorde* van de patronen.
- De rol wordt duidelijk gemaakt door extra *knopen*.

De eerste oplossing is ongeveer deze die gebruikt wordt in het Nederlands (vergelijk *man bijt hond* met *hond bijt man*). Een gedachte die bestaat uit verschillende patronen wordt echter gedacht door deze patronen cyclisch te herhalen, zodat er geen eerste of laatste patroon is: deze oplossing moeten we dus verwerpen. Blijven dus de bijkomende, niet-semantische knopen. Voor elke mogelijke rol moet er zo een knoop zijn. De verbindingen

in het semantisch net duiden predikaten aan met twee rollen: begin en einde van de verbinding. Meestal is het etiket van de verbinding een werkwoord dat een actie aanduidt, waarbij een van de twee knopen de UITVOERDER van de actie is en de andere degene die de actie ondergaat (wat we kortweg de ONDERGAANDER zullen noemen, bij gebrek aan een beter woord[4]. De gedachte vervat in de uitspraak *de grote paarse kat achtervolgt de kleine doorschijnende hond* wordt dan weergegeven door de twee patronen

'groot'+'kat'+'paars'+UITVOERDER
'doorschijnend'+'hond'+'klein'+ONDERGAANDER

Om te benadrukken dat de volgorde van de opsomming van de knopen onbelangrijk is hebben we ze alfabetisch gerangschikt. Merk op dat deze manier van werken een zeer beperkt aantal niet-semantische knopen veronderstelt, waarvan we hier UITVOERDER en ONDERGAANDER gebruikt hebben. De meeste werkwoorden in het Nederlands duiden acties aan die één, twee of drie rollen hebben:

- Onovergankelijke werkwoorden (of onovergankelijk gebruikte overgankelijke werkwoorden) hebben alleen een UITVOERDER: rennen, groeien, ...
- Veel overgankelijke werkwoorden hebben een UITVOERDER en een ONDERGAANDER.
- Een aantal werkwoorden duiden acties aan die ook nog een BENEFICIËNT hebben, grammaticaal uitgedrukt door een meewerkend voorwerp, zoals *aan de klant* in *hij geeft het boek aan de klant*.

Verder is het zo dat in de patronen die de gedachte voorstellen nog ruimte is om de actie zelf nader te omschrijven, een omschrijving die in de meeste Europese talen wordt uitgedrukt door *bijwoordelijke bepalingen*. Dat de grote paarse kat de kleine doorschijnende hond traag achtervolgt kan worden uitgedrukt door de twee patronen

'groot'+'kat'+'paars'+'traag(bijw.)'+UITVOERDER+'achtervolgen'
'doorschijnend'+'hond'+'klein'+ONDERGAANDER+'traag(bijw.)'+'achtervolgen'

We beschouwen 'traag(bijw.)' als een semantische knoop die verschilt van 'traag', die geëtiketteerd is met een adjectief. In veel talen wordt het onderscheid tussen bijwoord en het adjectief door een expliciete vormverandering, zoals in het Frans ('lent' tegenover 'lentement') of het Engels ('slow' tegenover 'slowly').

Met een systeem van opeenvolgende patronen zoals hier geschetst kunnen we enkelvoudige acties uitdrukken. Voor meer complexe gedachten, waarin verschillende acties voorkomen die elk hun rollen hebben (zoals in de uitspraak *de grote paarse kat achtervolgt de kleine doorschijnende hond om hem schrik aan te jagen*), schiet dit systeem tekort. Er wordt aangenomen dat het manipuleren van dergelijke uitspraken een speciale structuur vereist die de structuur van een algemeen neuraal net overschrijdt. Bij mensen vinden we deze structuur in de vorm het zogenaamde *werkgeheugen*. De werking hiervan valt buiten het bestek van deze cursus.

12.4 GEHEUGEN

De reeks van activatiepatronen die een gedachte uitmaken kan worden opgewekt op verschillende manieren. We hebben gezien dat in de visuele cortex er mechanismen zijn die

[4] Men zou kunnen denken aan de grammaticale termen ONDERWERP en LIJDEND VOORWERP, maar in een passieve zin is het onderwerp de ONDERGAANDER en in een actieve zin de UITVOERDER, zodat deze termen niet ondubbelzinnig zijn.

leiden tot het aanmaken van een of meerdere tokens, zodat de sprong naar de reeks activatiepatronen niet zeer groot is. Ook via taal kan een gedachte geactiveerd worden. Maar gedachten worden niet alleen maar geactiveerd door stimuli van buitenaf. We moeten nog uitleggen hoe feiten kunnen worden opgeslagen in de synapsen en hoe dit het opnieuw denken van de gedachte kan helpen, terwijl toch *suspension of disbelief* kan optreden.

Hiervoor moeten we de leerregels voor een neuraal net herbekijken. Zoals we al weten verandert bij de methode van leren voor een kunstmatig net die het meeste op biologisch leren lijkt het gewicht van een verbinding op een manier die alleen rekening houdt met de activatie van de betrokken neuronen. Dit leidde tot leerregels van de vorm zoals deze voor een Hopfieldnet:

$$w_{ij} \rightarrow w_{ij} + u_i u_j.$$

In het model dat we nu gebruiken hebben we een reeks van opeenvolgende activatiepatronen: elk neuron x_i heeft op een gegeven moment t een activatie $u_i(t)$. Hierin is t een discrete tijdsparameter, die bijvoorbeeld de waarden 0, 1, ... kan aannemen. Het lijkt nu logisch om de bovenvermelde regel van Hebb voor dit model te schrijven als

$$w_{ij}(t+1) = w_{ij}(t) + u_i(t)u_j(t).$$

Dergelijke regel kan echter niet dienen als basis voor een goed model. Immers, het is gemakkelijk in te zien dat, als we beginnen met een symmetrisch netwerk ($w_{ij} = w_{ji}$ voor alle i en j) het netwerk met deze leerregel steeds symmetrisch zal blijven. Maar biologische netwerken kunnen sequenties leren. Mensen leren vlot woorden, zinnen, grammatica: allemaal voorbeelden van geordende opeenvolgingen, en ook dieren kunnen sequenties leren: zo zijn er veel zangvogels die melodieën van hun soortgenoten overnemen, wat bewijst dat ze sequenties kunnen onthouden. Maar het onthouden van een sequentie is essentieel asymmetrisch: er is een verschil tussen onthouden dat A na B komt, en onthouden dat B na A komt. Vermits de activatie van x_i op een gegeven moment t via de verbinding met gewicht w_{ij} de activatie van x_j op het ogenblik $t+1$ beïnvloedt, is het logischer om een regel van de vorm

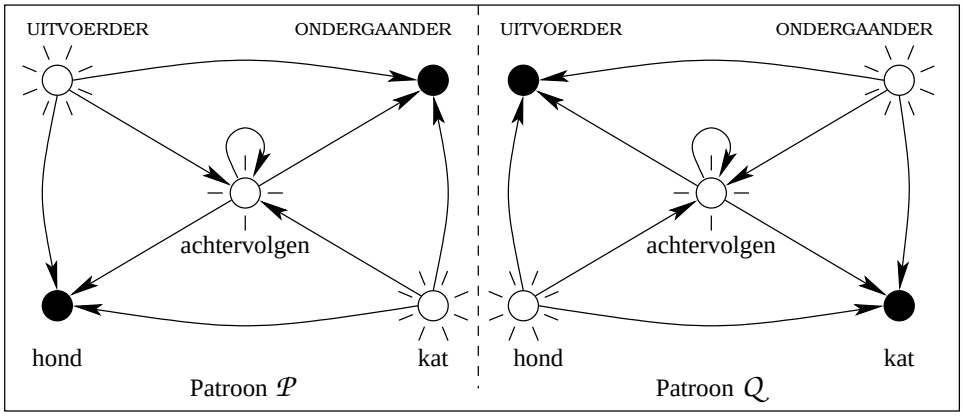
$$w_{ij}(t+1) = w_{ij}(t) + u_i(t)u_j(t+1)$$

voorop te stellen, die wel asymmetrisch is in de tijd[5]. We gaan geen gebruik maken van de numerieke waarden die volgen uit deze regel: in een biologisch net zijn er verschillende mechanismen die de leerreactie beïnvloeden, en die leiden tot verschillende evenwichten. De regel die we zullen gebruiken geeft alleen kwalitatieve veranderingen aan, en we formuleren deze regel als volgt:
Als een activatiepatroon $\mathcal{P}$ gevolgd wordt door het patroon $\mathcal{Q}$, dan veranderen de verbindingen op zo'n manier dat het waarschijnlijker wordt dat, als patroon $\mathcal{P}$ later nog eens voorkomt, het zal gevolgd worden door $\mathcal{Q}$.
Het resultaat zien we in Figuur 12.1. In deze figuur zijn de twee elkaar afwisselende patronen voor de gedachte *de kat achtervolgt de hond* te zien, namelijk de patronen

$\mathcal{P}$='kat'+UITVOERDER+'achtervolgen'
$\mathcal{Q}$="hond'+ONDERGAANDER+'achtervolgen'.

Op elk van de twee schema's zien we alleen de excitatorische verbindingen die vertrekken vanuit de geactiveerde knopen, en die dus het daaropvolgende patroon zullen bepalen. Inhibitorische verbindingen hebben we voor de duidelijkheid niet aangegeven. Deze zijn wel

[5] Deze leerregel is veel meer in overeenstemming met de theorieën van Donald Hebb dan de tijdsonafhankelijke. Volgens hem: "When one cell repeatedly assists in firing another, the axon of the first cell develops synaptic knobs (or enlarges them if they already exist) in contact with the soma of the second cell."Asymetrie is hier duidelijk aanwezig. Wel vermeldt Hebb alleen excitatorische synapsen.



Figuur 12.1. De verbindingen voor de gedachte *de kat achtervolgt de hond.*

belangrijk. Immers, indien een net met feedbacklussen alleen excitatorische verbindingen zou bevatten, dan zou de activatie, als ze boven een bepaalde drempel uitkomt, zichzelf steeds versterken zodat op den duur alle neuronen op volle kracht vuren, een toestand die grote gelijkenis vertoont met een epileptische aanval.

Als een gedachte slechts uit één patroon bestaat (*honden blaffen luid*), dan wordt dit patroon zelfbevestigend, en gedragen de verbindingen zich symmetrisch, zoals bij een Hopfieldnet. De leerregel voor een Hopfieldnet is dan ook de regel voor patronen die gedurende langere tijd actief zijn.

Het eerste wat we met dit model kunnen verklaren is de *abstractie* die leidt tot algemene uitspraken. Nemen we aan dat we een neuraal net hebben dat verschillende waarnemingen doet van katten die honden achtervolgen. We nemen aan dat de katten waar het over gaat allemaal verschillende eigenschappen hebben: sommige zijn groot, andere klein, sommige paars, andere oranje, sommige harig, sommige niet, en zo voorts. Ook de honden in kwestie hebben allerhande eigenschappen. Het resultaat zal zijn dat alleen de verbindingen getoond in Figuur 12.1 sterk zullen beïnvloed worden. De verbindingen van en naar de eigenschappen van de honden en katten veranderen slechts weinig en, als er inhibitorische verbindingen zijn dan kan het zijn dat er ook een tegengestelde werking is: we kunnen ze dus verwaarlozen. Op deze manier hebben we uit een aantal specifieke uitspraken een algemeen feit afgeleid, op een manier die gelijkaardig is aan de manier waarop een Hopfieldnet een stabiel minimum bepaalt uit een aantal gelijkaardige patronen.

De overblijvende verbindingen die de abstractie uitmaken zijn voldoende om de implementatie van het semantisch net te verklaren. Immers, elke geëtiketteerde verbinding duidt een relatie aan met een UITVOERDER- en een ONDERGAANDERrol (al is de benaming van de rollen misschien niet perfect gekozen). Een algemeen feit kan wel eigenschappen van objecten bevatten. Zouden er in de voorbeelden van katten die honden achtervolgen alleen honden voorkomen die doorschijnend zijn, maar voor de rest allerhande verschillende eigenschappen hebben, dan zouden de verbindingen naar de eigenschap ‘doorschijnend’ ook allemaal evenveel versterken, en zou deze eigenschap opgenomen worden als eigenschap kenmerkend voor honden die door katten achtervolgd worden. Het neurale model is dus verfijnder dan het semantischnetmodel, dat alleen relaties tussen types toestaat, en geen relaties tussen types-met-extra-relaties kan weergeven. Bovendien verklaart dit model hoe het semantisch net vragen kan beantwoorden, alhoewel we de rol van de ‘is een’-relatie met dit eenvoudig model nog niet kunnen verklaren. Later, als we samengestelde types

gedefinieerd hebben, zullen we zien hoe de 'is een'-relatie geïmplementeerd wordt, en hoe het netwerk vragen beantwoordt met gebruik van deze relatie.

We kunnen de verschillende mogelijke vragen over het elkaar achtervolgen van katten en honden die het semantisch net kan beantwoorden immers uitdrukken als een reeks van een of meerdere activatiepatronen die door de vraag gecreëerd worden:

Vraag	Patroon/patronen
Wie achtervolgt er honden?	$\mathcal{Q}$
Wie wordt er door katten achtervolgd?	$\mathcal{P}$
Wat doet een kat met een hond?	Afwisselend: - 'kat' + UITVOERDER - 'hond' + ONDERGAANDER

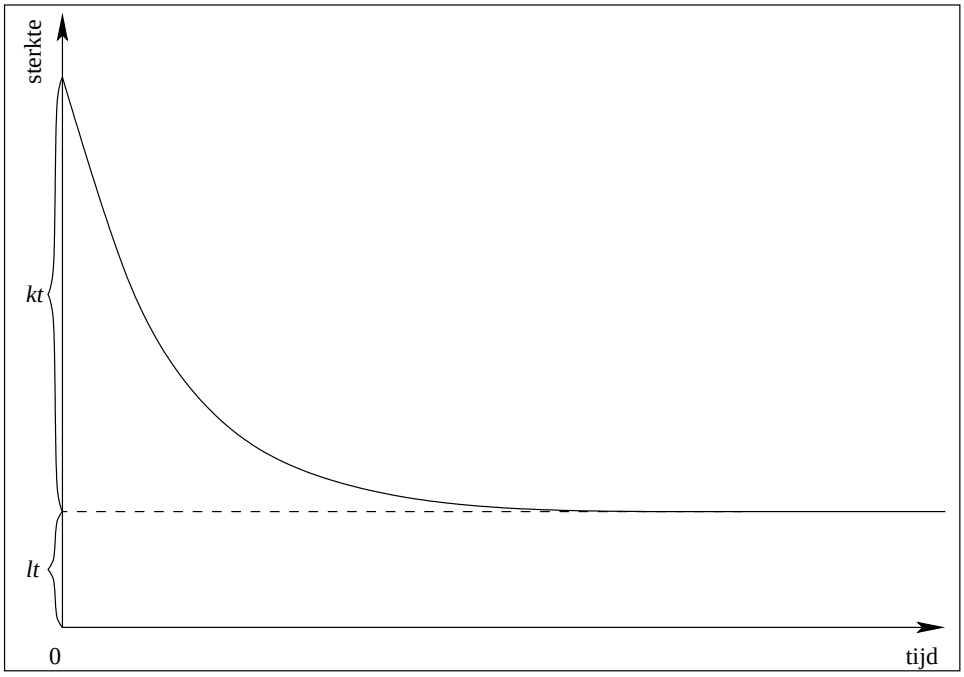
De vraagpatronen worden elke keer gevormd door de elementen van het antwoord die in de vraag vervat zitten. Dat de eerste twee vragen met het bijbehorende patroon tot het antwoord zullen leiden, is duidelijk. Het laatste vereist enige uitleg. Op een gegeven ogenblik leidt de vraag tot een patroon dat twee van de drie elementen van het patroon $\mathcal{P}$ bevat. Het volgende ogenblik activeert de vraag 'hond' + ONDERGAANDER, terwijl de gedeeltelijke activatie van $\mathcal{P}$ op het voorgaande moment leidt tot een (zwakke) activatie van $\mathcal{Q}$. Bijgevolg zijn 'hond' en ONDERGAANDER volledig actief, en 'achtervolgen' gedeeltelijk. Bij de volgende tijdstap worden weer 'kat' en UITVOERDER volledig geactiveerd, terwijl 'achtervolgen' iets sterker wordt geactiveerd dan in de vorige tijdstap (door de lusverbinding van 'achtervolgen' naar zichzelf). Op deze wijze convergeert de activatie naar het gewenste patronenpaar. Uiteraard is deze voorstelling van zaken een vereenvoudiging, omdat elke knoop deelneemt aan nog andere feiten, en er dus een soort van concurrentiestrijd kan ontstaan tussen verschillende patronen. Hier spelen de inhibitorische verbindingen een cruciale rol: zij moeten ervoor zorgen dat de ongewenste patronen gedesactiveerd worden.

Tenslotte is het nog belangrijk in dit perspectief te wijzen op het verschil tussen kortetermijn- en langetermijngeheugen. De hierboven beschreven leerregels worden in biologische netten gerealiseerd door middel van een aantal verschillende biochemische processen. Niet alle synapsen bieden plaats aan al deze processen (men vermoedt trouwens dat leren niet alleen door het wijzigen van synapsen gebeurt, maar dat in bepaalde omstandigheden synapsen systematisch gevormd kunnen worden, wat ook een vorm van leren is)[6]. Sommige synapsen kunnen helemaal niet leren, zodat ze een gedrag vertonen dat aangeboren is (zo is het hebben van pijn iets dat niet moet geleerd worden). Andere synapsen kunnen wel een of meerdere van deze leermechanismen bevatten. Deze mechanismen kunnen in een aantal opzichten van elkaar verschillen: of het mechanisme in één keer grote of kleinere veranderingen teweegbrengt, wat het verzadigingspunt is (dit is bereikt als de synaps zijn maximale sterkte bereikt heeft), in hoeverre het proces actief omkeerbaar is (bijvoorbeeld: verzwakt een excitatorische synaps van A naar B als A afvuurt en B niet?), en zo verder. Nog niet alle leermechanismen zijn gekend, maar verschillende delen van de hersenen bestaan uit een verschillende mix van neuronen en synapsen, waarvan men veronderstelt dat ze elk deel geschikt maken voor een bepaalde taak.

Een belangrijke eigenschap is de *duurzaamheid* van de aanpassing. Sommige processen leiden tot een permanente of semi-permanente aanpassing (en leiden tot kennisopslag die decennia kan overbruggen), andere hebben slechts een tijdelijk effect, dat van enkele seconden tot meerdere dagen kan voortduren. Men spreekt van korte- en langetermijngeheugen, maar de grens tussen de twee is niet duidelijk te trekken: decennia worden tot de

[6] In dit verband kan opgemerkt worden dat de mannelijke kanarie (vrouwelijke kanaries zingen niet) zowat de meest extreme vorm van vergeten in het dierenrijk vertoont. Het zangcentrum van deze vogel schrompelt in elkaar na het zangseizoen, en groeit het jaar daarop weer tot zijn normale grootte, met nieuwe neuronen.

lange termijn gerekend, seconden tot de korte termijn, en wat daartussen ligt is niet strikt bepaald. De superpositie van verschillende mechanismen kan het *suspension of disbelief*-verschijnsel uitleggen.



Figuur 12.2. Evolutie van de invloed van leren. kt: kortetermijnmechanismen, lt: langetermijnmechanismen

Het is bekend dat een gedachte kan aanhouden als actieve patronensequentie, zelfs als de prikkel die daartoe aanleiding gaf verdwenen is. Bovendien wordt een pas gedachte gedachte gemakkelijk terug actief, zelfs als ze onderbroken is door een andere gedachte. Beide fenomenen zijn moeilijk te rijmen met *suspension of disbelief*. Immers, bij dit fenomeen is de gedachte niet in overeenstemming met de structuur van de verbindingen. Als, bijvoorbeeld, het netwerk weet dat kikkers ooievaars eten, dan zal de excitatorische verbinding van 'kikker' naar ONDERGAANDER sterk zijn. Maar als het netwerk denkt dat ooievaars kikkers eten, dan moet deze verbinding juist zwak zijn. De vraag is dan hoe het mogelijk is dat een netwerk weet dat kikkers ooievaars eten (sterke verbinding aanwezig) terwijl het denkt dat ooievaars kikkers eten, en zelfs deze gedachte kan vasthouden, waarvoor een zwakke verbinding van 'kikker' naar ONDERGAANDER nodig is (en de andere verbindingen eveneens niet overeen mogen komen met wat er aanwezig is)? Het fenomeen kan verklaard worden in het geval dat kortetermijnmechanismen een sterkere werking hebben dan langetermijnmechanismen. Op het ogenblik dat externe prikkels het netwerk ertoe brengen te denken dat ooievaars kikkers eten is het effect van de korte termijn belangrijk, en halen de verbindingen voor deze gedachte de bovenhand. Op langere termijn verdwijnt dit effect grotendeels, zodat het oude geloof dat kikkers ooievaars eten terug de bovenhand haalt.

Dit kortetermijneffect zorgt er ook voor dat tokens een zekere identiteit krijgen. Indien een neuraal net een bepaalde gedachtegang volgt, dan creëert het op die manier een soort van klein semantisch net bovenop het klassieke semantisch net. Dit kleine net bestaat uit

kortetermijnverbindingen die sterker zijn dan deze van de langetermijnbasis. Dit tijdelijke net vormt de eigenlijke basis voor *suspension of disbelief*: wat waar is in dit net kan onwaar zijn in het permanente net. Indien een token voorkomt in meerdere patronen van de gedachtegang, en de elementen van dit token komen niet voor in andere tokens, dan hebben al deze elementen dezelfde relaties, toekomende zowel als vertrekkende, met andere knopen van het net, zodat dit token zich als een eenheid gedraagt. op deze manier wordt elk token binnen de gedachtegang een type binnen het tijdelijke semantische net. Indien verschillende tokens een overlapping hebben, dan zullen niet alle elementen van een token dezelfde verbindingen hebben met andere knopen. Immers: in tegenstelling tot knopen die uniek toebehoren aan één token, nemen deze elementen deel in verschillende patronen die niets met elkaar te maken hebben, waardoor er op hun verbindingen tegengestelde krachten worden uitgevoerd. Een voorbeeld zal dit misschien duidelijker maken: neem twee tokens '$rode'$ + '$ooievaar'$ en '$groene'$ + '$ooievaar'$. Als beide tokens deelnemen aan gelijkaardige gedachten ('een rode ooievaar eet kikkers' en 'een groene ooievaar eet kikkers') dan krijgen we, zoals reeds gezien, abstractie, en het resultaat is een algemene uitspraak. Hier zijn de verbindingen van de specifieke elementen (in dit geval de kleur) zwak, omdat ze soms wel en soms niet voorkomen. Als beide tokens deelnemen aan verschillende gedachten ('een rode ooievaar vliegt hoog' en 'een groene ooievaar vliegt laag') dan krijgen we dat de verbindingen van het gemeenschappelijk element zwak zijn (de verbinding tussen 'ooievaar' en 'hoog' is zwak, want soms wordt 'ooievaar' gevolgd door 'hoog', en soms niet), terwijl de verbindingen van de specifieke elementen sterk zijn ('rood' wordt nooit gevolgd door 'laag', en altijd door 'hoog'). Als we dit in een tabel gieten krijgen we

Element	Sterkte verbinding voor	
	Abstractie (... eet kikkers)	Onderscheid (... vliegt hoog/laag)
Gemeenschappelijk (ooievaar)	sterk	zwak
Specifiek (rood/groen)	zwak	sterk

Als tokens, of ze nu al dan niet overlappen, leiden tot tijdelijke types en zo, als deze tijdelijke types voldoende voorkomen in gedachten, tot permanente types, dan krijgen we een nieuwe vorm van het type-tokenprobleem. Wat gebeurt er namelijk als we twee objecten hebben met identieke eigenschappen? Wat als we twee grote witte katten hebben, en de ene achtervolgt de andere? De meest voorkomende uitweg is dat men op zoek gaat naar onderscheidende eigenschappen[7]: als een van de twee katten ook maar iets langer haar heeft dan de andere, dan wordt de ene de langharige kat en de andere de kortharige. Als dit niet lukt is er nog een noodoplossing, die ook gebruikt wordt als men geen gebruik *wil* maken van onderscheid. Dit is om een van de twee tokens te voorzien van de eigenschap 'het ene', en het andere van de eigenschap 'het andere', een oplossing die kan uitgebreid worden naar meer dan twee tokens (eerste, tweede, derde, ...).

In elk geval is het zo dat de nieuwe types die zo gevormd worden *samengestelde* types zijn, bestaande uit een ander type, met bijkomende eigenschappen. Tussen het nieuwe type en het oude type is er dan een 'is een'-relatie. Het oude type kan zelf een samengesteld type zijn: elke grote purperen kat is een purperen kat, en elke purperen kat is een kat. Merk op dat de 'is een'-relatie meer lijkt op de overerving bij frames dan op deze van objectgericht programmeren, vermits in het neurale net elk type de ouder kan worden van een nieuw type. Een token is dan een type dat naar slechts één object verwijst, en elk token kan op zijn beurt, door overerving, een type worden met meerdere deeltypes. Als we aannemen dat een netwerk een permanent type 'de kat van de buren' heeft, dat een token

[7] Mensen die contact hebben met een eeneïge tweeling kennen zowel het probleem als de oplossing.

is omdat de buren exact één kat hebben, dan kan dit token een niet-token worden als de buren een tweede kat in huis halen. Naarmate het neurale net de twee katten van elkaar leert onderscheiden, groeien de twee deeltypes in duidelijkheid.

We kunnen nu ook schetsen hoe het neurale net vragen kan oplossen waarbij de overerving relevant is. Keren we terug naar het semantische net zoals getekend op bladzijde 46 en laten het enkele vragen beantwoorden:

(1) *Heeft een eend vleugels?* Het antwoord is ja als de patrooncombinatie

'eend'+UITVOERDER+'hebben'
'vleugels'+ONDERGAANDER+'hebben'.

ondersteund wordt door de langetermijnverbindingen van het net. 'Eend' is een samengesteld type, dat als enkelvoudige type 'vogel' omvat. Bijgevolg kunnen de verbindingen leiden tot de patrooncombinatie

'vogel'+UITVOERDER+'hebben'
'vleugels'+ONDERGAANDER+'hebben'.

Deze patrooncombinatie is niet exact wat we willen (hierdoor wordt de activatie minder sterk, wat de werking van het net vertraagt), maar het antwoord is ja. Dit is een voorbeeld van overerving van een eigenschap.

(2) *kan een struisvogel vliegen?* 'Struisvogel' is een samengesteld type van de vorm 'struis'+'vogel'[8]. Door de activatie van de twee elementen ervan wordt *kan niet vliegen* sterker geactiveerd dan *kan wel vliegen*, want we hebben gezien dat de verbindingen van het onderscheidende element (in dit geval 'struis') sterker zijn dan die van gemeenschappelijk element. Het netwerk concludeert dan ook dat een struisvogel niet kan vliegen. Dit is een voorbeeld van het overschrijven van een eigenschap.

[8] Het woord 'struisvogel' is niet verwant met het Nederlandse woord 'struis', maar met het Griekse $\sigma\tau\rho\omega\theta\iota\omega\nu$, dat *mus* betekent.

HOOFDSTUK 13
EEN GEÏNTEGREERD NETWERK

Als toepassing op wat we vooraf gezien hebben bekijken we hier een model van een groot netwerk. Dit model sluit vrij nauw aan bij de beschrijvingen van Hebb van de werking van het menselijk brein. Het is de bedoeling de verschillende technieken die we in de vorige hoofdstukken bekeken hebben te illustreren met een reëel probleem: dat van vertaling. De beschrijving die we hier geven gaat voorbij aan een aantal details. Het is dan ook niet de bedoeling om een volledig werkend vertaalprogramma te construeren, wel om te tonen hoe verschillende technieken samen kunnen gebruikt worden.

13.1 HET VERTAALPROBLEEM

Reeds vanaf de jaren zestig is er onderzoek verricht naar de mogelijkheid vertaalprogramma's te maken. Na de eerste, beperkte successen, die gebaseerd waren op het bepalen van een aantal expliciete regels (als dit, doe dat), bleek dat het probleem van dubbelzinnigheid leidde tot problemen. Deze dubbelzinnigheid doet zich voor op drie niveaus:

1. Het eenvoudigste probleem is dat van de verschillende betekenissen van een woord. Bekijken we volgende Nederlandse zinnen:

   Het vertrek werd uitgesteld.
   De deur gaf uit op een vertrek zonder ramen.

   In de eerste zin is de Engelse vertaling van het woord 'vertrek', 'departure', in de tweede moet het als 'room' vertaald worden.
2. Het probleem van grammaticale ontleding, dat zich op twee niveaus afspeelt. Bekijk de volgende Engelse[1] zin, een klassieker uit het genre:

Time	flies	like	an	eagle.
(1)	(2)	(3)		

   Er zijn niet minder dan vier grammaticale mogelijkheden om deze zin te ontleden (en in één van de varianten is er ook nog een variatie met verschillende woordbetekenissen die vertalingen 3a en 3b hieronder geeft). Ambiguïteit wordt veroorzaakt door de mogelijke *grammaticale classificatie* van woorden. Op dit niveau zijn er drie mogelijkheden: elk van de eerste drie woorden kan als werkwoord worden beschouwd. Verder is er het probleem van *grammaticale combinatie*: als we (1) als werkwoord nemen kan 'like an eagle' zowel op 'Time' als op 'flies' slaan. We krijgen de vertalingen

   1a. Neem de tijd op van vliegen zoals een arend zou doen.
   1b. Neem de tijd op van arendachtige vliegen.
   2. De tijd vliegt als een arend.
   3a. Tijdvliegen houden van een arend.
   3b. Tijdvliegen vinden een arend lekker.

[1] Engels is de Europese taal met het minste grammaticale structuur. Vertalingen uit het Engels zijn veel moeilijker dan uit pakweg Duits en Russisch, die veel expliciete structuur hebben, zoals naamvallen.

Vertaling (2) is de normale vertaling, maar de andere vier zijn grammaticaal correct. (3a) en (3b) hebben dezelfde grammaticale structuur, alleen is het woord 'like' verschillend geïnterpreteerd. Een menselijke vertaler zal geen moeite hebben om (2) eruit te halen, maar bij computervertalingen vindt men vaak fouten die lijken op de andere vertalingen die we hier geven.

Een aantal van de problemen kunnen redelijk vlot door een computer worden opgelost: bepaalde mogelijke classificaties, bijvoorbeeld, leiden niet tot een zinnige zinsstructuur. In de zin 'Time is on my side' kunnen we 'Time' niet als werkwoord opvatten, want dan staan er twee werkwoordsvormen naast elkaar. Maar veel ambiguïteiten kunnen maar opgelost worden met een grondige dosis kennis van de wereld. Neem de volgende Nederlandse zinnen:

1. De kat speelde op de piano.
2. Tante Magda speelde op de piano.

De correcte Engelse vertalingen zijn (meestal) 'the cat played *on top of* the piano' and 'aunt Magda played the piano'. Maar algemene kennis van de wereld is nodig, ook kennis van de context (en de mogelijkheid deze te gebruiken): 'the cat played the piano' en 'aunt Magda played on top of the piano' kunnen correct zijn in een of ander sprookje.

Om deze ambiguïteit te verwerken is parallellisme nodig, zodat alle mogelijkheden samen kunnen verwerkt worden. Immers, combinatie van ambiguïteiten kan tot een zeer groot aantal mogelijkheden leiden, wat seriële verwerking vertraagt. Parallellisme is er voldoende in ons hoofd, zo is er een aparte woordherkenner voor elk woord, een aparte eenheid voor elke mogelijke betekenis, een aparte eenheid voor elke grammaticale structuur, enzovoorts.

13.2 OVERZICHT VAN HET NETWERK

We gaan uit van een gesproken tekst in een of ander brontaal die moet worden omgezet in de doeltaal. Wat er moet gebeuren is het volgende:

1. De binnenkomende geluiden moeten omgezet worden in een reeks van *fonemen*.
2. De fonemenreeksen moeten gesplitst worden in woorden. De woorden voegen we samen in zinnen.
3. Van deze zinnen moet de grammaticale structuur bepaald worden.
4. Uit de woorden en de grammaticale structuur moeten we betekenissen halen.
5. Deze betekenissen moeten omgezet worden in de doeltaal, door de passende grammaticale structuur in de doeltaal te vinden en de juiste woorden daarin in te passen.
6. De resulterende woordenreeks moet omgezet worden in een fonemenreeks en uitgestuurd worden.

We gaan niet alle stappen in detail bekijken. Stap (1) is een redelijk technisch onderwerp, hoewel het in principe een klassiek probleem van classificatie is. Stappen (5) en (6) zijn de omkering van de stappen (1) tot en met (4). In onze hersenen maken ze gebruik van de structuren die ook in deze stappen actief waren. Voor een computer zijn deze twee stappen relatief eenvoudig omdat er geen ambiguïteit is en omdat de 'spreker' zelf beslist welke structuren (welke grammaticale constructies, welke woorden,...) hij wil gebruiken. Aan de andere kant beschikken hedendaagse vertaal- en spraakprogramma's niet over een taalonafhankelijke abstracte kennisrepresentatie[2]. Dit maakt dat computervertalingen alleen werken als bron- en doeltaal ongeveer dezelfde grammaticale structuur hebben (bijvoor-

[2] Overigens wordt ons begrippensysteem ook beïnvloed door de taal of talen die we kennen.

beeld Nederlands naar Engels of Russisch of zoiets). Er is nog niemand in geslaagd om een computer geslaagde, zelfs eenvoudige, vertalingen te laten maken van bijvoorbeeld het Navaho[3] naar het Engels.

Verder zullen we ons netwerk modulair indelen. Een verzameling neuronen die instaan voor een begrip (een foneem, een woord, ...) zullen we een *cluster* noemen. In onze diagrammen stellen we zo'n cluster voor als een klein rechthoekje, dat verschillende in- en uitgangen kan hebben. De interne structuur zullen we dan niet meer bekijken. Het kan zelfs zijn dat een neuron deelneemt aan verschillende clusters. In onze hersenen komt dit volgens sommigen veel voor, maar voor de eenvoud gaan we er hier van uit dat dit niet het geval is.

De twee belangrijke vormen van clusters hebben we behandeld in de vorige hoofdstukken. Clusters die verband houden met tijdssequentiële data zullen min of meer de vorm aannemen van een automaat, terwijl clusters die simultane data moeten verwerken zich gedragen als associatieve geheugens.

Hierbij dient opgemerkt dat we een aantal kenmerken weglaten die essentieel zijn voor de goede werking van het systeem.

13.2.1 Automaten

In principe hoort bij elke sequentie die relevant is voor taalverwerking een automaat die deze sequentie kan herkennen. De automaten uit paragraaf 8.3 werkten aan een vaste snelheid en kregen een input per twee tijdsstappen. Onze automaten hebben nood aan een ingebouwd synchronisatiemechanisme. Synchronisatie is op alle niveaus belangrijk. Woorden kunnen op verschillende snelheid worden uitgesproken, zinsdelen in een grammaticale constructie kunnen meer of minder woorden bevatten, en zo voorts. Het bestaan van zo'n synchronisatiemechanisme is experimenteel vastgesteld, maar het zou te ver voeren om hier in te gaan op de structuur ervan. Ook dient te worden opgemerkt dat de automaten die instaan voor de herkenning van een bepaalde sequentie ook verantwoordelijk zijn voor de productie van die sequentie. Dit is een vrij ingewikkeld proces waarbij moet worden gekozen welke automaat juist wordt geactiveerd en waarbij op een gegeven ogenblik een hele hiërarchie van automaten kan actief zijn.

13.2.2 Associatieve geheugens

De Hopfieldnetten uit Hoofdstuk 11 werkten met TLU's. Bij de verwerking van taal is het echter heel belangrijk om rekening te houden met de verschillen in activatiegraad van verschillende clusters. Het belangrijkste is dit bij het oplossen van ambiguïteiten en bij het corrigeren van fouten. Zo moeten bijvoorbeeld op het laagste niveau verkeerd uitgesproken (of verkeerd begrepen) fonemen worden gecorrigeerd. Dit betekent dat ook de automaten van woorden die fonetisch lijken op het waargenomen woord een activatie moeten krijgen (ze zouden immers kunnen horen bij het correcte woord, terwijl het waargenomen woord verkeerd is). Het netwerk moet in staat zijn te beslissen wat de correcte interpretatie is. Dat doet het door op elk niveau verschillende concurrerende opties te vergelijken en die opties met de te kleine activatie te onderdrukken door middel van een zacht winner-takes-all-mechanisme.

13.2.3 Structuur

We kunnen in het netwerk drie eenheden onderscheiden:

(1) Een *woordherkenningseenheid.*

[3] Navaho kent geen substantieven: alles is er actie. Het is het typevoorbeeld van een taal die niet op een Europese taal lijkt.

(2) Een *grammaticale eenheid*.
(3) Een *begripseenheid*.

Deze drie eenheden communiceren voortdurend met elkaar.

13.3 VAN FONEEM NAAR WOORD

We krijgen een reeks fonemen aangeboden[4], de ene na de andere, en we moeten deze samenvoegen tot woorden. De tijdsvolgorde is belangrijk en we mogen veronderstellen dat er voor elk woord een aparte herkennende automaat is.

Terzijde weze opgemerkt dat bij lezen de volgorde van minder belang is, zoals wordt bewezen door het volgende citaat over plaatsvolgorde van letters:

luek om te wteen : Vglenos een odzeonrek aan de cmabridge uinervstieit, makat het neit uit in wkele vdgoorle de letters zcih in een worod bnedvien, het egine brelkganije is dat de eetrse en de latsate lteetr op de jutise piotsie satan[5].

Dit geeft aan dat we bij lezen elk woord als één geheel bekijken. Een woord wordt beschouwd als een cluster van letters, alhoewel blijkbaar de eerste en laatste letter apart worden genomen. In elk geval is ook hier voor elk woord een detectiecluster. we mogen aannemen dat elke cluster enige tijd nodig heeft om het woord te analyseren: de output van de cluster evolueert naar een duidelijk 'ja' of 'nee'.

Bij het menselijk gehoor worden lange woorden niet in één laag gevonden: er is een tussenlaag van korte woordstukjes[6].

Er is een oppervlakkige gelijkenis met de HMM's die we in Hoofdstuk 4 gezien hebben, maar deze gaat niet verder dan de lineaire structuur van het te herkennen woord: HMM's zijn stochastische producenten van sequenties, hier gaat het over een herkennende automaat.

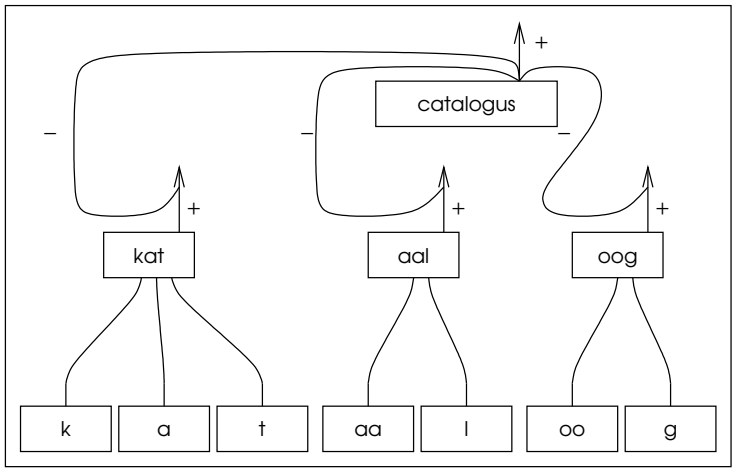
Alle woordautomaten werken parallel en proberen hun woord te herkennen in de invoerstroom. Het spreekt vanzelf dat er dan ook veel valse positieven zullen zijn, zeker voor korte woorden. Voor zover een kort woord een deel kan zijn van een langer woord zal detectie van het lange woord detectie van het korte woord onderdrukken. Het detectieschema voor 'catalogus' ziet er dan ongeveer zo uit als in Figuur 13.1.

In dit voorbeeld zien we reeds duidelijk dat verschillende woordclusters geactiveerd worden die niets met de tekst van doen hebben: de woorden 'kat', 'aal' en 'oog' zijn niet uitgesproken, maar wel gedetecteerd. Vandaar dat er terugkoppelingslussen zijn van cluster 'catalogus' naar deze woorden. Deze zijn gemarkeerd met een minteken. Het zijn immers negatieve terugkoppelingen zodat, als 'catalogus' wordt geactiveerd, deze woorden worden gedesactiveerd. Dit maakt het geheel tot een klein associatief geheugen dat vier patronen, namelijk de woorden 'kat', 'oog', 'aal' en 'catalogus' kan herkennen. Er is een zekere complicatie met synchronisatie: 'catalogus' wordt pas gedetecteerd als het woord volledige is uitgesproken, een tijdje nadat 'kat' is herkend. We gaan hier niet op in. In ons catalogusvoorbeeld worden ook de woorden 'kat', 'aal' en 'oog' geactiveerd, maar door de activatie van 'us' wordt deze activatie ongedaan gemaakt, waarbij de dubbelzinnigheid verdwijnt. Maar de dubbelzinnigheid is niet altijd opgelost. Als we overschakelen op Vlaams wordt met de zin 'Hij zag dat de kat al oog had voor de muis' het woord 'kataloog' foutief geactiveerd. Dit probleem wordt opgelost op dezelfde manier als het betekenisprobleem.

[4] We vereenvoudigen hier. In werkelijkheid is er nog andere informatie. De woordenreeks 'Jan en tomaten tomaten' heeft dezelfde fonemen als 'Jan en Tom aten tomaten', maar klinkt wel anders.

[5] `http://www.radio1.be/programma/jenw`, op 22/9/03.

[6] Wie een vreemde taal leert dient eerst deze woordstukjes te leren. Vandaar dat een woord als 'tsjoevstva' redelijk moeilijk is om uit te spreken voor wie geen Russisch kent, hoewel er geen vreemde klanken inzitten.



Figuur 13.1. Detectie-eenheid voor ‘catalogus’.

13.4 VAN WOORD NAAR BETEKENIS

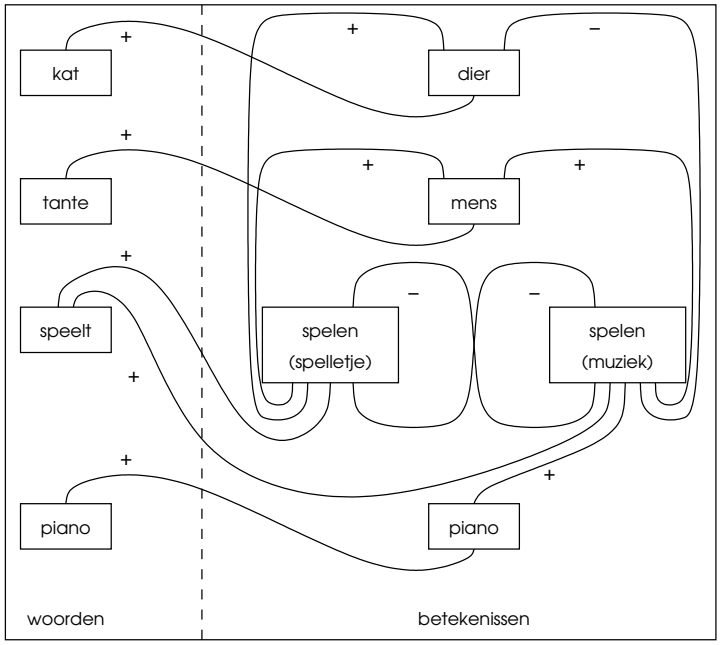
We hebben nu een stel woorden in een bepaalde volgorde. We gaan ervan uit dat die woorden eenduidig bepaald zijn. Dit is al een vereenvoudiging van de werkelijkheid. Normaal zullen op elk ogenblik heel wat woorden geactiveerd zijn. Onze automaten zijn geen TLU-automaten die wel of niet een signaal geven. Alleen bij zeer duidelijke spraak is er geen nood aan foutopheffing: bij tests blijken proefpersonen fouten en lacunes zeer goed te kunnen aalvullen, zelfs zonder zich er van bewust te zijn dat deze fouten en lacunes er zijn. Dat betekent dat we bij spraakherkenning keuzes moeten maken uit verschillende combinaties van verschillende woorden. Hiervoor maken we essentieel gebuik van grammaticale en semantische (betekenis-) informatie. Het is dus niet zo dat het woordherkenningsgebied kant-en-klare informatie aflevert. Bovendien zal ze ook input van de twee andere modules gebruiken om problemen op te lossen: woorden met betekenissen verwant aan al geïdentificeerde begrippen en woorden die passen in een grammaticale structuur, voor zover deze al gekend is, worden meer geactiveerd dan andere.

Het betekenisgebied van het neuraal net bestaat uit verschillende modules die verbonden zijn zoals een Hopfieldnet. Betekenissen die samen voorkomen zijn verbonden door positieve takken. Betekenissen die verbonden zijn met hetzelfde woord zijn verbonden met negatieve takken. Ditlaatste levert een winner-takes-allnetwerk op zoals we beschreven hebben in een vorig hoofdstuk. In het volgende voorbeeld hebben we voor de eenvoud een aantal elementen weggelaten, waarvan het belangrijkste het cruciale woordje ‘op’ is, met de verschillende betekenissen van plaats (bovenop) en middel (door middel van).

De negatieve terugkoppelingen tussen de twee betekenissen van spelen zijn zeer sterk: als een van de twee sterker geprikkeld wordt dan de andere, zal ze ervoor zorgen dat de andere betekenis niet geactiveerd wordt. Alleen als de twee betekenissen even sterk geprikkeld worden worden ze beide geactiveerd: dit heet twijfel.

13.5 GRAMMATICALE ANALYSE

De grammatica van menselijke taal kan min of meer voorgesteld worden door een context-



Figuur 13.2. Betekenisdetectie.

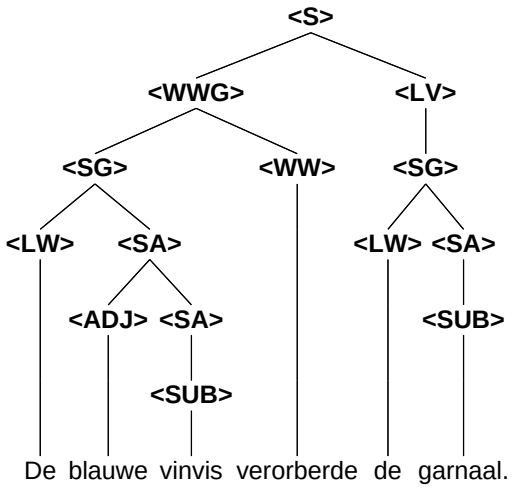
vrije grammatica[7]. Contextvrije grammatica's geven een vrij goed beeld van de grammaticale analyse zoals die gebeurt in de menselijke hersenen. Zoals bekend kan zo'n grammatica worden beschreven met een aantal zgn. *niet-terminale symbolen*, of kortweg niet-terminalen. Deze worden aangeduid met de notatie $\langle \mathbf{\ldots} \rangle$, waarin ... vervangen wordt door de naam van het niet-terminale symbool. Bij elke niet-terminaal hoort een *substitutieregel*, die de verschillende mogelijkheden geeft om het symbool te vervangen. Uitgegeen wordt van een startsymbool, meestal $\langle \mathbf{S} \rangle$, waarin S staat voor *sentence*, en dus een volledige zin aanduidt. Een klein voorbeeld:

$$
\begin{array}{lcl}
\langle \mathbf{S} \rangle & ::= & \langle \mathbf{WWG} \rangle \langle \mathbf{LV} \rangle \\
\langle \mathbf{WWG} \rangle & ::= & \langle \mathbf{SG} \rangle \langle \mathbf{WW} \rangle \\
\langle \mathbf{LV} \rangle & ::= & \langle \mathbf{SG} \rangle \\
\langle \mathbf{SG} \rangle & ::= & \langle \mathbf{LW} \rangle \langle \mathbf{SA} \rangle \\
\langle \mathbf{SA} \rangle & ::= & \langle \mathbf{ADJ} \rangle \langle \mathbf{SA} \rangle | \langle \mathbf{SUB} \rangle \\
\langle \mathbf{LW} \rangle & ::= & de \\
\langle \mathbf{ADJ} \rangle & ::= & blauwe \\
\langle \mathbf{SUB} \rangle & ::= & vinvis | garnaal \\
\langle \mathbf{WW} \rangle & ::= & verorberde
\end{array}
$$

Hierbij staan WWG, LV, SG, SA, LW, ADJ, SUB en WW voor werkwoordgroep, lijdend voorwerp, substantiefgroep, substantief-adjectief, lidwoord, adjectief, substantief en werkwoord.

Hebben we een zin die voldoet aan de grammatica dan kunnen we een syntaxboom opstellen zoals in Figuur 13.3.

[7] Linguïsten stellen dat een contextvrije grammatica niet voldoende is. We gaan hier niet verder op in, behalve om op te merken dat een algemene contextvrije gramatica nog te sterk is.



Figuur 13.3. Syntaxboom.

We gaan er nu vanuit dat voor elk alternatieve substitutie voor een symbool er een automaat is die een lineaire sequentie detecteert bestaande uit de opeenvolging van de activaties van zijn delen; als er meerdere alternatieven zijn worden deze verbonden door een OF-neuron.

De syntaxboom geeft aan welke van de automaten actief zijn op een gegeven ogenblik: men moet gewoon het pad volgen van de wortel tot aan het woord dat verwerkt wordt. Op het ogenblik dat 'blauwe' wordt verwerkt zijn dat bijvoorbeeld de automaten $\langle \mathbf{S} \rangle$, $\langle \mathbf{WWG} \rangle$, $\langle \mathbf{SG} \rangle$, $\langle \mathbf{ADJ} \rangle$ en *'blauwe'* actief.

Dit gaat goed als er geen pad is waarop dezelfde niet-terminaal twee keer voorkomt. Nu zijn er essentieel twee gevallen binnen een contextvrije grammatica waarbij dit lan gebeuren.

In het eerste geval hebben we linkse of rechtse recursie. Dit is in ons voorbeeld het geval door de regel $\langle \mathbf{SA} \rangle ::= \langle \mathbf{ADJ} \rangle \langle \mathbf{SA} \rangle | \langle \mathbf{SUB} \rangle$. Immers, het resultaat van de toepassing van deze regel is altijd van de vorm $\langle \mathbf{ADJ} \rangle * \langle \mathbf{SUB} \rangle$. Dit is een reguliere uitdrukking en dus bestaat er een neurale automaat voor.

Anders is het gesteld met centrale recursie, waarbij we impliciet of expliciet een regel hebben van de vorm $\langle \mathbf{A} \rangle ::= \langle \mathbf{B} \rangle \langle \mathbf{A} \rangle \langle \mathbf{C} \rangle | \langle \mathbf{D} \rangle$. Dergelijke regel maakt dat de taal niet meer regulier is en dus niet door aan automaat kan herkend worden. In menselijke taal komen constructies voor waarbij centrale recursie gebruikt wordt. Nemen we het voorbeeld

De man wandelde door de dreef.
De man die in het huis woonde wandelde door de dreef.
De man die in het huis dat op de berg stond woonde wandelde door de dreef.
De man die in het huis dat op de berg die vol bomen stond stond woonde wandelde door de dreef.
De man die in het huis dat op de berg die vol bomen die juist in bloei stonden stond stond woonde wandelde door de dreef.
....
De afhandeling van dergelijke recursie valt buiten het bestek van de cursus.

De grammaticale analyse staat toe om de rol toe te voegen aan de denkpatronen zoals we ze gedefinieerd hebben in Hoofdstuk 12.

13.6 VERTALING

De vertaling zelf is nu een relatief eenvoudig proces. We zijn ervan uit gegaan dat de grammaticale structuur van de doeltaal dezelfde is dan deze van de brontaal. Aan de hand van de geactiveerde betekenissen worden nu de in te vullen woorden gekozen. Uiteraard moeten nog de juiste woordvormen (verbuigingen en vervoegingen) doorgegeven worden. Het resultaat wordt naar de uitvoerneuronen gebracht, die de juiste klanken in de juiste volgorde produceren.

Hieronder volgt een lijstje van formules die niet van buiten gekend moeten zijn voor het examen. Mocht je er een van nodig hebben krijg je ze bij de desbetreffende opgave.

Varia:

- Hoeveelheid informatie van de classificatie van een verzameling S:

$$\begin{aligned} E(S) &= \sum_{i=1}^{k} A(S,i) \left(-\log_2 \left(\frac{A(S,i)}{|S|} \right) \right) \\ &= |S| \log_2(|S|) + \sum_{i=1}^{k} A(S,i) \left(-\log_2 \left(A(S,i) \right) \right). \end{aligned}$$

- Formule voor Viterbialgoritme:

$$p(i(\cdot)|j(\cdot)) = \alpha \prod_{t=0}^{t=T-1} T_{i(t),i(t+1)} \prod_{t=0}^{t=T} U_{i(t)j(t)},$$

- Actieve systemen.
  Waardering voor aan algemene strategie:

$$w_\pi(s) = b(s) + w_\pi(T(\pi(s), s)).$$

  Definiërende eigenschap voor een optimale strategie:

$$w_{\tau_{opt}}(s) = b(s) + \max_{d \in D} w_{\tau_{opt}}(T(d, s)).$$

  Verwachtingswaarde bij exploratie met onzekerheid:

$$w_\pi(s_i) = b(s_i) + \sum_j T(\pi(s_i))_{ij} w_\pi(s_j).$$

  .

- Continu tijdsmodel voor neuronen:

$$u_i(t) = f(a_i(t)) \qquad \frac{da_i}{dt} = -a_i(t) + s_i(t).$$

- Discriminatiefunctie stemmachine:

$$V(\mathbf{z}) = \sum_i^n y_i \alpha_i g(\mathbf{x}_i, \mathbf{z}) - T.$$

Leren:

- Regel van Hebb:

$$w_i \to w_i + (t-u)v_i.$$

  Met nieuwe totale invoer

$$\sum_{i=0}^{n}(w_i + (t-u)v_i)v_i - \sum_{i=0}^{n} w_i v_i = (t-u)\sum_{i=0}^{n} v_i^2.$$

- Deltaregel:

$$\Delta\mathbf{w} = \frac{y(\mathbf{v}) - u}{f'(a)||\mathbf{v}||^2}\mathbf{v} = \frac{e}{||\partial_{\mathbf{w}} u||^2}\partial_{\mathbf{w}} u.$$

Zekerheidsfactoren:

$$\begin{array}{ll} \text{zf}(a) = z_1 + z_2 - z_1.z_2 & (z_1 \geq 0, z_2 \geq 0) \\ \text{zf}(a) = z_1 + z_2 + z_1.z_2 & (z_1 \leq 0, z_2 \leq 0) \\ \text{zf}(a) = \frac{z_1+z_2}{1-\min(|z_1|,|z_2|)} & (z_1.z_2 \leq 0). \end{array}$$

Energiefuncties:

- Hopfieldnet:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}u_i u_j.$$

- Stemmachine:

$$\mathcal{F}(\alpha_1, \ldots, \alpha_n) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i\alpha_j y_i y_j g(\mathbf{x}_i, \mathbf{x}_j).$$

BIBLIOGRAFIE

[1] **Peter Dayan en L. Abbot:** *Theoretical Neuroscience*, MIT Press, 2003.

[2] **Laurene Fausett:** *Fundamentals of Neural Networks. Architectures, Algorithms and Applications*, Prentice Hall International editions, 1994.

[3] **Donald Hebb:** *A textbook of Psychology*, W. B. Saunders Company, 1966.

[4] **George Luger:** *Artificial Intelligence*, Addison Wesley, 2005.

[5] **Stuart Russel en Peter Norvig:** *Artificial Intelligence. A modern approach*, second edition, Prentice Hall, 2003.

[6] **J. Schürmann:** *Pattern Classification. A unified view of statistical and neural approaches*, John Wiley & Sons, 1996.

[7] **Richard Sutton en Andrew Barto:** *Reinforcement Learning: an introduction*, second edition, MIT Press, 2018.