FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Kunstmatige intelligentie

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

Gecompileerd op 18 juni 2019

UNIVERSITEIT
GENT

Inhoudsopgave

Hoofdstuk 1

Kunstmatige intelligentie

- In deze cursus:
  - Genetische algoritmen toegepast op actieve regelbanken.
  - Bayesiaanse leermethodes toegepast op spraakherkenning.
- Twee doelen van kunstmatige intelligentie:
  - Het laten overnemen, door machines, van taken waarvoor intelligentie vereist is.
  - Studie van natuurlijke intelligentie.
- Twee vormen om kennis in te brengen in een computersysteem:
  - Expliciete kennis (expertsystemen).
  - Kennis kan zelf verworven worden.

1.1 Kunnen machines denken?

- Twee voorbeelden.
  - ELIZA:
    - Computerprogramma dat zich voordoet als een pyschotherapeut.
    - Maakt gebruik van simpele vervangingsregels.
    - Probeert de conversatie zo te sturen zodat de echte persoon het meest moet vertellen.
  - Chinese kamer:
    - Denkrichting die aantoont dat een entiteit eerst iets moet begrijpen, vooraleer er van intelligentie sprake is.
      1. Iemand die geen Chinees kent wordt in een kamer gebracht.
      2. Door een luik krijgt hij briefjes in het Chinees aangereikt, en de bedoeling is dat hij daar schriftelijk een zinnige antwoord op teruggeeft.
      3. De persoon krijgt handboeken waarin conversieregels staan.
    - De proefpersoon volgt mechanisch de regels vanuit het handboek, zodat hij wel intelligent gedrag vertoont, maar de berichten niet begrijpt.
- **Denken is elke vorm van complexe informatieverwerking waarvan de onderliggende mechanismen niet volledig gekend zijn.**

- **Turingtest**:
  - Proefpersoon kan contact maken met twee entiteiten: een mens en een machine, maar hij weet niet wie de mens of machine is.
  - De proefpersoon kan eender welke vragen stellen aan beide entiteiten.
  - Als de proefpersoon er niet in slaagt om na zijn vragenronde de entiteit aan te duiden die een machine is, dan is de machine geslaagd voor de Turingtest.

1.2 Toepassingen van AI en data mining

- **Classificatie:**
  - Stel een verzameling van k klassen.
  - Een bepaalde invoer met gelinkt worden aan één van die klassen.
  - Harde classificatie: beperkt aantal duidelijk van elkaar gescheiden klassen. Hier spreekt men ook van patroonherkenning.
  - Zachte classificatie: continue overgang van de klassen.
- Toepassingen:
  - Aanbevelingssystemen.
  - Kwaliteitscontrole.
- Probleemgestuurd: uitgaande van een probleem een oplossing zoeken.
- Datagestuurd: vanuit bestaande informatie problemen zoeken die ermee opgelost kunnen worden. Dit wordt ook data mining genoemd. Vaak moet de data eerst gereorganiseerd worden vooraleer de informatie nuttig wordt.

1.3 Leren

- Moderne AI houdt zich bezig met systemen met een zeer groot aantal aanpasbare parameters. Zulke systemen noemt men **massief lerende systemen**.
- **Voorbeelden** van massief lerende systemen:
  - Neurale netwerken: trachten het biologische denksysteem na te bootsen.
  - Hidden Markov Model: wordt gebruikt bij de analyse van allerhande sequenties, waarbij de toestand soms onbekend is.
- Parameters hebben niet noodzakelijk een betekenis, en is daarom ook onmogelijk om ze met de hand in te voeren. Daarom laat men een systeem leren, met behulp van **drie methoden**:
  - **Algoritmisch leren**: Er wordt gedemonstreerd hoe een bepaalde actie moet uitgevoerd worden. Het systeem kan hierna deze actie inoefenen door het herhalen van deze instructies. Deze vorm komt goed overeen met het programmeren van een computer.
  - **Leren met supervisie**: Hier wordt er geen gebruik gemaakt van een algoritme maar eerder van voorbeelden. Deze voorbeelden worden een leerverzameling genoemd en bevatten inputgegevens die het systeem moet leren herkennen, met de daarbij horende resultaten. Er wordt een verband opgelegd tussen een bepaalde input en output.

- **Leren zonder supervisie**: Dit gebeurt gedeeltelijk algoritmisch aangezien er enige instructies nodig zijn om de machine op gang te krijgen. De machine zal nadien zelf experimenteren wat er gebeurd bij het aanpassen van verschillende parameters. Het leren gebeurt dus niet met voorbeelden, maar uit eigen ervaring. Hier is er dan ook geen verband tussen het resultaat en de verschillende deeltaken, maar er is wel een algemeen idee wat er aangeleerd moet worden.

1.4 Classificatie

- Classificatie is het mappen van een bepaalde input op een klasse.
- We spreken van een **item** dat we moeten klasseren.
- Dit item wordt gekarakteriseerd door een aantal **meetwaarden**.
- Twee soorten meetwaarden:
  - Sommige metingen kunnen een groot aantal waarden opleveren die voorgesteld kunnen worden als een getal.
  - Andere metingen hebben maar een beperkt aantal waarden, zoals de indeling van categorieën.
    - Deze kunnen omgezet worden zodat ze antwoorden zijn op ja-nee vragen, zodat ze geconverteerd kunnen worden naar 0 of 1.
    - ✓ Nu zijn alle meetwaarden getallen.
- Aangezien dat alle meetwaarden getallen zijn → standaardvorm: de computer heeft een aantal klassen en moet een getallenrij die de meetwaarden voor een bepaal item bevat toewijzen aan één van de klassen.
- Hoe worden de getallenrijen weergegeven? Een aantal notaties:
  - De n-dimensionale Euclidische ruimte is de de verzameling vectoren met n reële coördinaten.
  - Zo een vector wordt voorgesteld door een vette letter: $\mathbf{v} = (v_1, ..., v_n)$.
  - Soms vanaf 0 beginnen, zodat we $n+1$-dimensionale vectoren hebben: $\mathbf{v} = (v_0, v_1, ..., v_n)$. De waarde v_0 krijgt een speciale betekenis.
  - Reeële getallen die niet deel uitmaken van een vector worden weergegeven met hoofdletters: A, B, ...
  - De nulvector: $0 = (0, ..., 0)$.
  - Het inproduct:

$$\mathbf{v} \cdot \mathbf{u} = \sum_{i}^{n} v_i u_i$$

    Het inproduct is lineair:
    - $(A\mathbf{v}) \cdot \mathbf{u} = A(\mathbf{v} \cdot \mathbf{u})$
    - $(\mathbf{u} + \mathbf{v}) \cdot \mathbf{x} = \mathbf{u} \cdot \mathbf{x} + \mathbf{v} \cdot \mathbf{x}$

    Het kan ook gedefinieerd worden als

$$\mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}||\ ||\mathbf{u}|| \cos\theta$$

    met θ de hoek tussen vector $\mathbf{v}$ en $\mathbf{u}$.
    Als $\mathbf{v}$ en $\mathbf{u}$ orthogonaal zijn ($\theta = \pi/2$), dan is het inwendig product:

$$\mathbf{v} \cdot \mathbf{u} = 0$$

Als $\theta = 0$, dan

$$\mathbf{v} \cdot \mathbf{u} = ||\mathbf{v}||\ ||\mathbf{u}||$$

Hieruit volgt

$$\mathbf{u} \cdot \mathbf{u} = ||\mathbf{u}||^2$$

- $d(u, v) = ||\mathbf{u} - \mathbf{v}||$ is de lengte van de kortste weg van $\mathbf{u}$ naar $\mathbf{v}$. Hieruit volgt:

$$||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$$

Het kwadraat van beide kanten geeft:

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

Aangezien dat

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{||\mathbf{u}||\ ||\mathbf{v}||}$$

kan dit omgevormd worden tot de volgende ongelijkheid:

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

- De afstand en de cosinus geven vaak een goede indruk in hoeverre twee vectoren op elkaar lijken. De cosinus geeft een goede maat voor de afstand tussen twee genormaliseerde vectoren:

$$d\left(\frac{\mathbf{u}}{||\mathbf{u}||}, \frac{\mathbf{v}}{||\mathbf{v}||}\right)^2 = 2 - 2\cos(\mathbf{u}, \mathbf{v})$$

1.5 Informatie en beslissingsbomen

1.5.1 Informatie-inhoud

- Een bericht is enkel nuttig indien ontvanger een betekenis kan geven aan het bericht. De belangrijke elementen voor de informatie-inhoud is dus het bericht zelf en de kennis van de ontvanger.
- Met de kennis kan aan elk mogelijk bericht B een waarschijnlijkheid $P(B)$ toekennen. De informatie-inhoud wordt dan gedefinieerd door

$$-\log_2(P(B)) \text{ bits}$$

Voor $P(B) = 1$ is de informatie-inhoud 0 bits, wat logisch is aangezien de ontvanger niets heeft bijgeleerd van dit bericht.

! De informatie-inhoud van een bericht is niet altijd een geheel getal.

! De informatie-inhoud is nooit negatief.

- Voorbeeld: Stel dat een byte verwacht wordt, maar er is geen idee welke byte. Elke byte is even waarschijnlijk met kans $1/256$. De informatie-inhoud van de byte die dan binnenkomt is $-\log_2(1/256)$ bits $= 8$ bits.
- Voorbeeld: Stel een alfabet van 4 letters: A, C, G en T. De waarschijnlijkheid dat ze voorkomen wordt weergegeven in tabel 1.1.

  Als de ontvanger dit weet dan wordt de informatie-inhoud voor elke letter:

$$A : -\log_2(0,7071) = 0,5$$
$$C : -\log_2(0,1250) = 3,0$$
$$G : -\log_2(0,0839) = 3,575$$
$$T : -\log_2(0,0839) = 3,575$$

A	70,71 %
C	12,50 %
G	8,39 %
T	8,39 %

Tabel 1.1: De waarschijnlijkheden voor de letters A, C, G en T.

1.5.2 Beslissingsboom

- Elke knoop dat geen blad is bevat een vraag met een beperkt mogelijk aantal antwoorden.
- Elk mogelijk antwoord verwijst naar een kind van de knoop.
- Een item klasseren is een pad vanuit de wortel naar een blad, waarin de klasse staat.
- Hoeveel informatie kan een beslissingsboom geven?
  - Stel k klassen $K_1, K_2, ...K_k$.
  - Stel een verzameling S van items waarbij:
    - $A(S,i)$ het aantal elementen horend bij K_i is in de verzameling en,
    - $|S| = \sum_{i=1}^{k} A(S,i)$ het totaal aantal element is van S.
  - De informatie geleverd door een correcte klassering van alle element is dan:
    (groene formules moeten niet van buiten geleerd worden. Ze worden gegeven bij de examenvraag)

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

    Hierbij is $\frac{A(S,i)}{|S|}$ de kans om een item te hebben in klasse i.

- Het **Iterative Dichotomiser 3 (ID3)** is een inhalig algoritme dat een beslissingsboom opstelt vanuit een bepaalde dataset.
  - De wortel bevat de vraag die het meeste informatie oplevert.
  - Als het j-de attribuut de leerverzameling L in de deelverzamelingen $L_{j,1}, L_{j,2}, ..., L_{j,n}$ opdeelt, dan is de informatie geleverd door dit attribuut gelijk aan:

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

    Hierbij is m de verschillende waarden dat dit attribuut kan aannemen.
  - Het attribuut wordt gekozen waarvoor $I(j)$ maximaal wordt.
  - ! Als $I(j) = 0$, dan behoren ofwel alle items tot dezelfde klasse, ofwel kan er op basis van het attribuut geen klassering gemaakt worden.
  - Na de constructie van de wortel moeten nog n_j deelbomen geconstrueerd worden.
  - Voorbeeld:
    - Veronderstel volgende informatie (tabel 1.2):
    - Voor elk attribuut kan de resulterende verdeling afgeleidt worden (tabel 1.3):

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

Tabel 1.2:

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

Tabel 1.3:

◇ Voor elk attribuut kan nu $I(j)$ berekent worden, hier uitgewerkt voor de beroepscategorie:

$$
\begin{aligned}
I(1) &= E(L) - \sum_{m=1}^{n_1} E(L_{1,m}) \\
&= E(L) - (E(L_A) + E(L_B) + E(L_C)) \\
&\text{met} \\
E(L) &= 480(-\log_2(480/528)) + 48(-log_2(48/528)) \\
&= 232.054 \\
E(L_A) &= 110(-log_2(110/139)) + 29(-log_2(29/139)) \\
&= 102.702 \\
E(L_B) &= 270(-log_2(270/279)) + 9(-log_2(9/279)) \\
&= 57.3603 \\
E(L_C) &= 100(-log_2(100/110)) + 10(-log_2(10/110)) \\
&= 48.3447 \\
&\text{zodat} \\
I(1) &= E(L) - (E(L_A) + E(L_B) + E(L_C)) \\
&= 232.054 - 102.702 - 57.3603 - 48.3447 = 23.647
\end{aligned}
$$

◇ Voor $I(2)$ en $I(3)$ bedragen beide uitkomsten 0, aangezien er op basis van die attributen geen informatie kan achterhaald worden. Dit is logisch aangezien de verhouding

veilige/risicohoudende klanten voor zowel leeftijd als fraudeur 10:1 is.

- ◇ Als wortel van de beslissingsboom wordt als criterium de beroepscategorie genomen.
- ◇ Voor zowel L_A, L_B en L_C moet apart dezelfde methode uitgevoerd worden. Het kan perfect mogelijk zijn dat op het tweede niveau bij keuze A eerst wordt gecheckt op fraude, maar bij keuze C eerst op leeftijd.
- ! Bij L_C levert geen enkel van de twee attributen informatie op, zodat er random moet gekozen worden:

Attribuut	waarde	# veilig	# riskant
(2) Jonger dan 20?	ja	50	5
	nee	50	5
(3) Fraudeur?	ja	50	5
	nee	50	5

- ◇ Verfijningen:
  - ∗ Invoeren van een drempelwaarde voor de informatiewinst. Als deze te klein is wordt er niet meer opgesplitst. Een blad kan dan meerdere klassen bevatten, elk met een waarschijnlijkheid.
  - ∗ Voor elk mogelijk paar van attributen de informatiewinst berekenen en, als deze te groot is, knopen maken met twee attributen.
  - ∗ Bij effectieve getallen kan ook een drempelwaarde ingevoerd worden. Alle items met een waarde kleiner dan deze drempelwaarde gaan naar links, de andere naar rechts. Voor elke mogelijke drempelwaarde moet de informatiewinst berekend worden.

1.6 Klasseren zonder leren

- Klassen worden niet vooraf gegeven.
- ! Geen leerverzameling aanwezig.
- Een **groep** van punten is wat door een expert als een groep beschouwd wordt.
  1. Twee punten behoren waarschijnlijk tot dezelfde groep als ze zeer dicht bij elkaar liggen. De expert definieert de afstandsfunctie.
  2. Deze eigenschap wordt **transitief** verdergezet. Twee punten $\mathbf{x}$ en $\mathbf{z}$ behoren tot dezelfde groep als een rij punten $\mathbf{y_1}, ..., \mathbf{y_n}$ 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.

1.6.1 k zwaartepunten

- Op voorhand opgeven dat er k klassen zijn.
- ! Een zwakte, aangezien men nu moet weten hoeveel klassen er op voorhand zijn. Twee problemen:
  - ○ Het aantal gekozen klassen is te groot zodat samenhangende groepen worden opgesplitst. Dit kan opgelost worden samenhorende groepen op het einde samen te nemen.
  - ○ Het aantal gekozen klassen is te klein zodat verschillende groepen samen worden genomen. Dit wordt opgelost door het algoritme meerdere malen uit te voeren met verschillende initialisaties.

- k **zwaartepunten** poogt een leerverzameling L op te delen in k groepen $G_1, ..., G_k$, waarbij k vooraf opgegeven is. Een klasse wordt voorgesteld door haar zwaartepunt m_i, waarbij n_i het aantal vectoren in G_i is:

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

- Een punt $\mathbf{z}$ wordt toegewezen aan een groep als volgt:
  1. Zoek uit de zwaartepunten $\mathbf{m_1}, ..., \mathbf{m_k}$ datgene dat het dichtst bij $\mathbf{z}$ ligt.
  2. $\mathbf{z}$ wordt dan toebedeeld aan de bijhorende klasse.
- Het resultaat is een **Voronoi-diagram**. Het negatieve aan zo een diagram is dat het enkele convexe groepen toelaat.

! Groepen die niet convex zijn worden door dit algoritme niet goed ingedeeld.

- Gebaseerd

1.7 Een toepassing: Watson

onbelangrijk

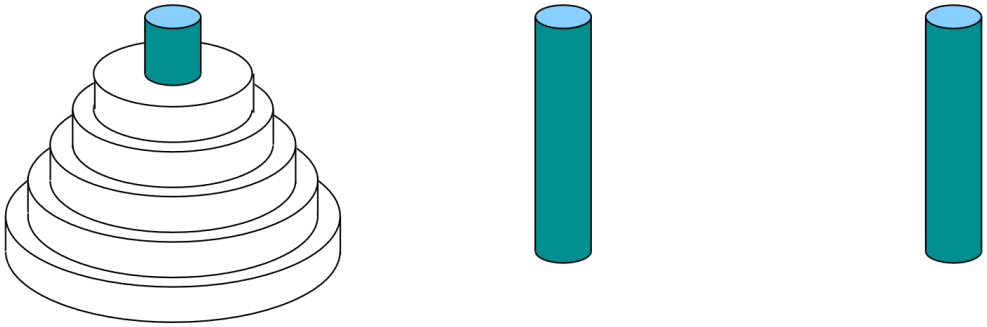
Hoofdstuk 2

Zoeken in zoekruimten

- Een zoekruimte tracht de manier van het menselijk redeneren na te bootsen.
- Drie elementen:
  - Een gerichte graaf, waarin de knopen toestanden zijn, en waarin er een verbinding van toestand a naar toestand b is als er in a een actie mogelijk is die tot b leidt.
  - Een begintoestand.
  - Een doel, dat bestaat uit een verzameling van toestanden.
- Soms is het afgelegde pad belangrijk, bijvoorbeeld wanneer de kost ook belangrijk is.

2.1 STRIPS

- **STanford Research Institute Problem Solver (STRIPS)** = een algemene probleemoplosser.
- Met behulp van een taal wordt het op te lossen probleem beschreven.
- Voorbeeld: de torens van Hanoi (Figuur 2.1).



Figuur 2.1: De torens van Hanoi.

  - Twee niveaus van positieve uitspraken:
    1. Nuldeordepredicaten: Dit zijn strings, zoals *DeLuchtIsBlauw* of *Veilig*.

2. Eersteordepredicaten: Deze zijn functies met een aantal parameters, die objecten aanduiden.

- Elke schijf hangt aan een bepaalde pin, er kan dus een predicaat van de eerste orde ingevoerd worden zoals bijvoorbeeld:

$$\texttt{HangtAan(schijf1, pin1)}$$

- Ook moet duidelijk gemaakt worden dat twee verschillende pinnen effectief verschillend zijn:

$$\texttt{IsNiet(pin1, pin2)}$$

- De begintoestand van de torens van Hanoi kan nu beschreven worden als volgt:

$$\begin{aligned}
&\texttt{HangtAan(schijf1, pin1)} \wedge \\
&\texttt{HangtAan(schijf2, pin1)} \wedge \\
&\texttt{HangtAan(schijf3, pin1)} \wedge \\
&\texttt{HangtAan(schijf4, pin1)} \wedge \\
&\texttt{HangtAan(schijf5, pin1)} \wedge \\
&\quad\texttt{IsNiet(pin1, pin2)} \wedge \\
&\quad\texttt{IsNiet(pin1, pin3)} \wedge \\
&\quad\texttt{IsNiet(pin2, pin1)} \wedge \\
&\quad\texttt{IsNiet(pin2, pin3)} \wedge \\
&\quad\texttt{IsNiet(pin3, pin1)} \wedge \\
&\quad\texttt{IsNiet(pin3, pin2)}
\end{aligned}$$

  De predicaten worden gebonden met een logische EN. Ook is het nodig om de symmetrie te definiëren, aangezien STRIPS hier niets van af weet.

- Nu kan een actie gedefinieerd worden om een schijf te bewegen. Voor elke schijf moet dit apart gedaan worden. Hier wordt het uitgewerkt voor schijf 2:

  $\texttt{BeweegSchijf2(van, naar, tussen)}:$

  $P:$ $\texttt{IsNiet(van, naar)} \wedge \texttt{IsNiet(van, tussen)} \wedge \texttt{IsNiet(naar, tussen)}$
  $\texttt{HangtAan(schijf1, tussen)} \wedge \texttt{HangtAan(schijf2, van)}$

  $+:$ $\texttt{HangtAan(schijf2, naar)}$

  $-:$ $\texttt{HangtAan(schijf2, van)}$

  Elke actie bestaat uit een premisse P die voldaan moet zijn vooraleer de actie kan uitgevoerd worden. De lijst van de toe te voegen uitspraken wordt aangeduid met het $+$ symbool en de lijst met de te verwijderen uitspraken worden aangeduid met het $-$ symbool.

- Om de symmetrie van de begintoestand op te lossen zou een actie `IsSymmetrie(A, B)` aangemaakt kunnen worden als volgt:

  $\texttt{IsSymmetrie(A, B)}:$

  $P:$ $\texttt{IsNiet(A, B)}$

  $+:$ $\texttt{IsNiet(B, A)}$

  $-:$ /

2.2 Efficiënt zoeken in zoekruimten

2.2.1 Breedte-eerst zoeken

- De **vertakkingsfactor** is het gemiddeld aantal nieuwe, nog niet bekeken buren van een onderzochte knoop en wordt voorgesteld door b (branching factor).
- Bij de torens van Hanoi is $b \approx 1,5$.
- Voor een diepte d worden ongeveer

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

  knopen bezocht. Voor b voldoende groter dan 1 is dit ongeveer b^d.
- Als we tien zetten voor elke speler willen vooruitkijken bij schaken, moeten er $6^{20} \approx 3.6 \cdot 10^{15} = 3,6$ biljard knopen ontwikkeld worden.
- Indien zowel start- als eindtoestand bekend zijn, kan **bidirectioneel zoeken** toegepast worden. Er wordt vooruit gezocht vanuit het startpunt, en achteruit vanuit het doel, beide met breedte-eerst zoeken. Er moet dan in plaats van b^d knopen slechts $2b^{d/2}$ knopen ontwikkeld worden. Voor $b = 2$ en $d = 20$ krijgen we met breedte-eerst zoeken 10^6 knopen en met bidirectioneel zoeken ongeveer 2000.

2.2.2 Heuristieken

In plaats van breedte-eerst zoeken te gebruiken zou ook een manier kunnen gebruikt worden die de meest interessante knoop volgt, in plaats van ze blindelings af te lopen.

- Bij het zoeken in een graaf worden er aan elke knoop k twee waarden toegekend:
  1. De gekende kost $g(k)$ van de knoop:
     - Deze functie is een schatting van de werkelijke kost $g^*(k)$ van het kortste pad van de startknoop naar k.
     - Het wordt berekend door bij de kost van zijn voorganger v de kost $c(v \to k)$ op te tellen; de kost van de actie die v omzet in k.
     - Er geldt steeds $g(k) \geq g^*(k)$.
  2. De heuristische kost $h(k)$ van de knoop:
     - Deze functie is een schatting van $h^*(k)$, de kost om vanuit knoop k het doel te bereiken via het kortste pad.
- De som van deze twee waarden geven de geschatte kost, verpakt door de evaluatiefunctie f:

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

! Interessante knopen heben een lage f waarde.

- **Beste-eerst zoeken**:

  (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 $g_k(b) = g(k) + c(k \to b)$ 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).

- De functie g is afhankelijk van het probleem, maar h hoeft dit niet te zijn. Er zijn verschillende mogelijkheden voor h. Er is wel de voorwaarde dat ze **toelaatbaar** moeten zijn. Dit wil zeggen dat het een pad vindt en dat dit pad optimaal is.

A* heuristieken

- Een heuristiek is $A*$ als $h(k) \leq h^*(k)$.
- Bewijs: indien er slechts een eindig aantal knopen k_i, met $g^*(k_i) \leq g^*(D)$ zijn, is een $A*$-heuristiek toelaatbaar.
  - Neem een $A*$ heuristiek h en bewijs dat er 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: $g(k') = g^*(k')$.
  - Dit is geldig indien k' de startknoop is. Er is minstens één buur k'' die ook op het optimale pad ligt. Door k' te ontwikkelen wordt zijn plaats ingenomen door zijn opvolger k'' op het optimale pad. k'' voldoet aan (3), want
    $$g^*(k'') = g(k') + c(k' \to k'') = g_{k'}(k'')$$
  - Als k' 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. Bovendien geldt
    $$f(k') = g(k') + h(k') \leq g^*(k') + h^*(k') = g^*(D)$$
  - Er kan geen enkele k ontwikkeld worden met $g(k) > g^*(D)$, want dan zou $f(k) > f(k')$ zijn.
  - Iedere keer als een knoop k ontwikkeld wordt, is er een kleinere waarde voor $g(k)$ die overeenkomt met de kost van een pad s naar k.
  - Ooit zal een doelknoop d gekozen worden voor ontwikkeling en die heeft $g(d) = f(d) \leq g^*(D)$, en is bijgevolg optimaal.

Monotone heuristieken

- Een heuristiek is monotoon als
  1. voor elk paar knopen geldt dat $h(v) - h(k) \leq c(v \to k)$.
  2. $h(d) = 0$ voor alle d in de doelverzameling D.
- Bewijs: Neem een willekeurig pad $k_0k_1, ..., k_n$ tussen twee knopen k_0 en k_n. Dan is de functie $g(k_i) + h(k_i)$ monotoon stijgend als functie van i. Hierin wordt $g(k_i)$ gerekend langs het pad vanaf het startpunt k_0.
  - Voor $i < n$ geldt dat $g(k_i) + c(k_i \to k_{i+1}) = g(k_{i+1})$
  - Ook geldt er dat $h(k_i) \leq h(k_{i+1}) + c(k_i \to k_{i+1})$.

- Deze twee termen optellen en de term $c(k_i \to k_{i+1})$ schrappen levert

$$g(k_i) + h(k_i) \leq g(k_{i+1}) + h(k_{i+1})$$

- Twee gevolgen:
  1. Een monotone heuristiek is $A*$.
  2. Een knoop k wordt nooit ontwikkeld voor $g^*(k)$ gekend is.
- Efficiëntie
  - Voor $h = 0$ krijgen we breedte-eerst zoeken.
  - Voor $h = h*$ krijgen we de optimale heuristiek, die rechtstreeks naar het doel leidt, en die ook monotoon is.
  - Hoe bepalen of een willekeurige monotone heuristiek beter is dan een andere?
    - Laatste ontwikkelde knoop is altijd d met $g^*(d) = g^*(D)$ en $h(d) = 0$.
    - Voor een monotone heuristiek moeten alle knopen ontwikkeld worden waarvoor $g^*(k)+h(k) < g^*(D)$, en geen enkele waarvoor $g^*(k) + h(k) > g^*(D)$.
    - Een monotone heuristiek h_1 ontwikkelt minder knopen dan een monotone heuristiek h_2 als $h_1(k) > h_2(k)$.
    - Als voor sommige k geldt dat $h_1(k) < h_2(k)$ en voor andere k $h_1(k) > h_2(k)$, kan een betere heuristiek opgebouwd worden:

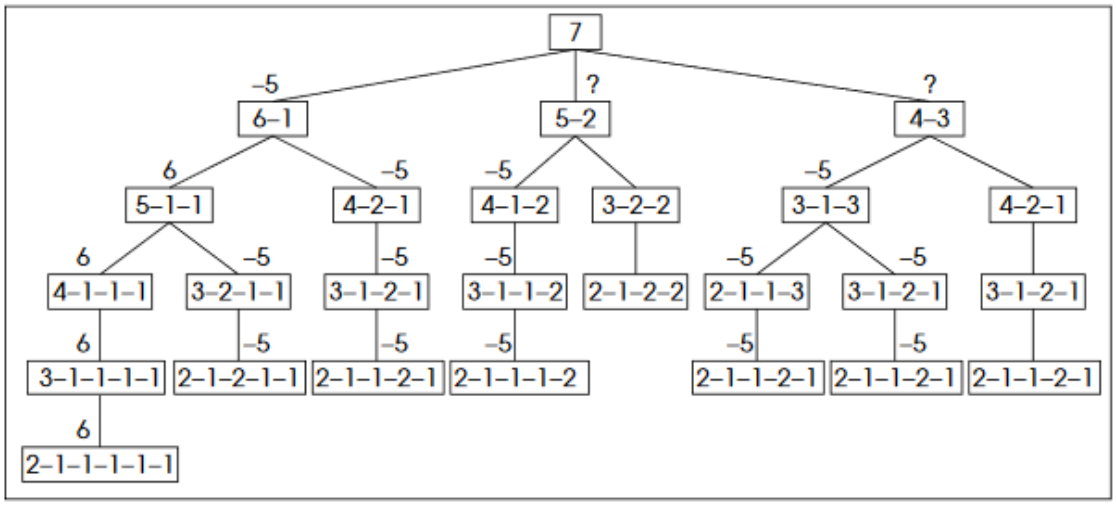
$$h_3 = \max\{h_1(k), h_2(k)\}$$

Relaxatieheuristieken

- Worden bekomen door het originele probleem sterk te vereenvoudigen.
- Er worden verbindingen toegevoegd in de graaf, wat overeenkomt met het schrappen van acties in de premisse.
- Het kortste pad in nieuwe graaf is zeker niet langer dan het kortste pad in de originele graaf. Deze heuristiek is dan ook A^*.

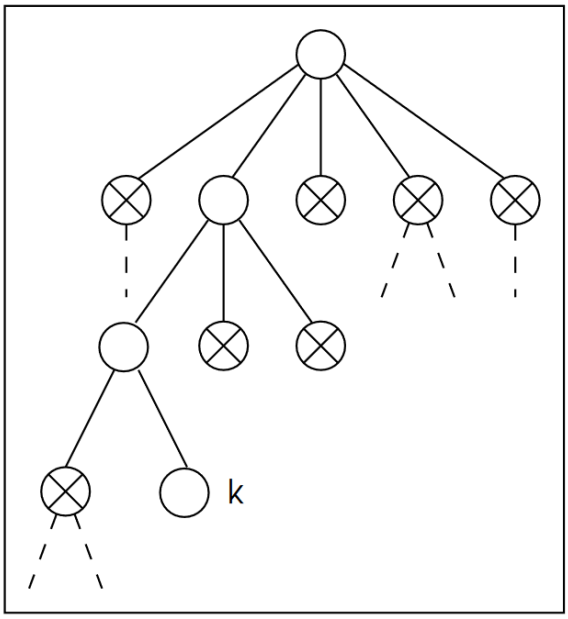
2.3 Spelbomen

- Wat doen indien er geen volledige kennis is over het resultaat van acties?
- Een **nulsomspel** (Engels: two player zero sum game)
  - Twee spelers.
  - Enkel winst mogelijk ten koste van de andere speler.
  - De winst van de ene speler is gelijk aan het verlies van de andere speler.
  - Elke speler probeert zijn winst te maximaliseren.
- Voorbeeld van een nulsomspel:
  - Op een tafel tussen twee spelers liggen een aantal stapels jetons.
  - De speler die aan de zet is moet een stapel nemen, en deze verdelen in twee ongelijke stapels.
  - De speler die niet meer kan zetten verliest.



Figuur 2.2: Een spelboom waarbij de startsituatie een stapel van 7 jetons is.

- De winnaar krijgt als winst het aantal punten gelijk aan het aantal overblijvende stapels, en de verliezer verliest dan ditzelfde aantal punten.
- Er kan een bijhorende boom (figuur 2.2) opgesteld worden die alle mogelijke zetten beschrijft.
- Noem de speler die begint *Max*, en de andere speler *Min*.
- Bij elke eindpositie is de winst of verlies van Max berekent.
- De overige posities kunnen van onder naar boven opgesteld worden:
  1. Als Max aan zet is dan nemen we de grootste score.
  2. Als Min aan zet is nemen we de kleinste score.
- Dit heet **minimax**.
- Er wordt gebruik gemaakt van $\alpha - \beta$ **snoeien** om oninteressante zetten (deelbomen) zeker niet te evalueren in de spelboom. Op die manier is er minder geheugen nodig.
  - De zet $7 \to 6 - 1$ levert een score van -5 op.
  - Van de zet $7 \to 5 - 2$ weten we niet wat de score is.
    - We weten wel dat Min erop kan reageren met een zet die een score van -5 geeft.
    - Het kan zijn dat de andere deelboom een betere score oplevert voor Min, maar het zal zeker geen lagere zijn (want die zet zou hij niet nemen).
    - Deze zet is niet beter dan $7 \to 6 - 1$
  - Algemeen wordt er gezocht naar een pad naar een blad waarbij zowel Max als Min steeds de beste zet kiezen.
    - Op dit pad is de kost van alle knopen gelijk.
    - De kost is s_{opt}, de optimale score.
- Moeten alle knopen ge-evalueerd worden om de optimale score te vinden?
  - Een **voorbroer** b van een knoop k is een broer van k, of een broer van één van de voorouders van k (Figuur 2.3).
  - Neem een knoop k en veronderstel dat $s(k)$ kan vervangen worden door een schatting f.
  - De waarden van de voorouders van k kunnen ook nu veranderd worden. Stel $f > s(k)$:



Figuur 2.3: De voorbroers van een knoop k worden gemarkeerd met een kruis.

1. k is een minknoop. Als $f > s(b)\ \forall\ b$ broer van k, dan wordt de s-waarde van de ouderknoop vervangen door f. Nu wordt naar geval 2 gegaan.
2. k is een maxknoop en geen wortel. De nieuwe $s-$waarde van de ouder is hoogstens f. Nu wordt naar geval 1 gegaan.

Hoofdstuk 3

Expertsystemen

- Kennis van een **echte expert** overnemen in een programma.
- De **gebruiker** moet ook enigszins kennis hebben. Hij moet de conclusies begrijpen en toepassen.
- **MYCIN**: een expertsysteem om te helpen bij de diagnose van bepaalde besmettelijke bloedziekten.
  - Is in staat om een betere diagnose te stellen dan een arts die geen expert is in het gebied.
  - De gebruiker moet wel een arts zijn die de diagnose begrijpt.
- Een expertsysteem maakt gebruik van **vuistregels**, waarop later nog **verfijningen** ingevoerd worden:
  1. werken met **onzekere conclusies**. Dit wordt ingevoerd om contradicties te behandelen indien deze voorkomen.
  2. De invoering van **frames**. Zorgt voor een betere orderning voor grote hoeveelheden kennis.

3.1 Eenvoudige systemen

- Een expertsysteem bestaat uit twee hoofdcomponenten
  1. Een **kennisbank**:
     - bevat **feiten**: dit zijn uitspraken die waar zijn.
     - bevat **regels**: dit heeft de vorm **als**<**premisse**>**dan**<**conclusie**>

       als AS=rond **en** MATERIAAL=staal
       dan SMERING=olie
  2. Een **afleidingssysteem**:
     - Elk probleem bestaat uit begingegevens en een doel.
     - Deze gegevens vormen een lijst van feiten die gekend zijn.
     - Het doel is ook een lijst van uitspraken.
     - Het doel is bereikt als één uitspraak uit de lijst is afgeleid uit de gegevens.
- Het afleidingssysteem kan op twee manieren werken.

- **Forward chaining:**
  - Kan in $O(g)$, met g de grootte van de regelbank.
  - Twee gegevensstructuren:
    - **LijstMetRegels(u)** die voor elke uitspraak u bijhoudt in welke regels ze in de premisse voorkomt.
    - **Premisseteller(r)** die voor elke regel r bijhoudt hoeveel uitspraken in de premisse nog niet gevalideerd zijn.
  - Werking:
    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:
       (a) Neem een uitspraak u uit de gegevensverzameling.
       (b) Voor elke regel r in *LijstMetRegels(r)*, verminder *Premisseteller(r)*. Als deze daardoor nul wordt, zet elke uitspraak uit de conclusie die nog niet behandeld is in de gegevensverzameling.
    4. Deel het resultaat mee aan de gebruiker.
  - Voorbeeld:
    - Stel volgende regelbank:

      als X kwettert en X zingt
      dan X is een kanarie

      als X kwaakt en X eet vliegen
      dan Dan X is een kikker

      als X is een kikker
      dan X is groen

      als X is een kanarie
      dan X is geel

    - Stel nu dat we een huisdier hebben, Fritz, en er zijn twee dingen bekend over hem: hij kwaakt en eet vliegen. We willen nu de kleur weten.
    - Fritz wordt gesubstitueerd voor X in regel 1, maar hij voldoet niet aan de premisse. Hij wordt nu gesubstitueerd voor X in regel 2, zodat er besloten wordt dat hij een kikker is. Dit wordt dan toegevoegd aan de gegevens. We weten nu dat Fritz kwaakt, dat hij vliegen eet en dat hij een kikker is.
    - Fritz kan nu gesubstitueerd worden in regel 3, zodat er besloten wordt dat hij groen is.
  - Forward Chaining werkt van links naar rechts, startend vanaf de gekende feiten om tot een conclusie te komen.
  - **!** In het algoritme kunnen regels toegepast worden die uiteindelijk niet bijdragen tot de oplossing. Men probeert dit efficiënter op te lossen met backwards chaining.
- **Backward chaining:**
  - Kan in $O(g)$, met g de grootte van de regelbank.
  - Werking:
    A. Behandeling voor een regel waarbij alle uitspraken uit de premisse gegevens zijn:
    *A*1. Voeg alle uitspraken uit de conclusie toe aan de gegevens.

*A*2 Voor alle tussenregels voor wie, op deze manier, alle uitspraken uit de premisse gegevens zijn geworden, behandel ze volgens *A*.

*A*3. 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:

*B*1. Voeg alle uitspraken uit de premisse die geen gegevens zijn toe aan de tussendoelverzameling.

*B*2. Verwijder de regel uit de regelbank en voeg hem toe aan de tussenregels.

C. Oplossingsalgoritme:

(a) Initialiseer de gegevensverzameling en doelverzameling. Initialiseer ook de verzamelingen van tussendoelen en tussenregels: deze zijn leeg.

(b) 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:

i. Neem zo een regel.

ii. Als alle uitspraken uit de premisse gegevens zijn, behandel volgens *A*, anders volgende *B*.

◇ Voorbeeld:

* Veronderstel dezelfde regelbank als bij forward chaining.
* Stel nu dat we een huisdier hebben, Fritz, en er zijn twee dingen bekend over hem: hij kwaakt en eet vliegen. We willen nu de kleur weten.
* Fritz wordt gesubstitueerd voor *X* in regel 3. Men moet dan bewijzen dat Frits een kikker is.

3.2 De constructie van een expertsysteem

- Twee soorten kennis nodig:
  - Kennis over het probleemgebied. Experten in het toepassingsgebied weten hoe ze problemen moeten oplossen, maar kunnen deze kennis niet vertalen in de vorm die een expertsysteem nodig heeft.
  - Kennis over expertsystemen. Een team die de kennis van de experten kan filteren en omvormen naar de juiste vorm.

3.3 Onzekerheid

- De relatie tussen premisse en conclusie van een regel is niet altijd zeker.
- Voorbeeld:
  - Een expertsysteem dat de oorzaak zoekt van problemen in een computer. Dit systeem kan een reparateur begeleiden die een defecte computer moet repareren.
  - De reparateur moet melden wat de symptomen zijn. Het systeem moet de actie bepalen die moet ondernomen worden om het probleem op te lossen. Mogelijke regels zijn bijvoorbeeld:

als SYMPTOOM=computer doet niets bij opstarten
dan PROBLEEM=voeding defect

als SYMPTOOM=voeding defect
dan PROBLEEM=vervang voeding

- Maar soms kan een probleem meerdere oorzaken hebben, waarbij sommige meer waarschijnlijkheid hebben dan anderen:

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

- Er wordt dan een numerieke factor toegekend, die de waarschijnlijkheid aanduidt.
- **Hoe wordt de numerieke waarde bepaalt?**
  1. Vage verzamelingen: **Niet te kennen**.
  2. Bayesiaans redeneren:
     - Stel 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

     - Statistisch redeneren kan toegepast worden in expertsystemen als er voldoende statistische informatie is:
       - De kans dat het morgen en overmorgen regent: $0,8 \times 0,8 = 0,64$.
       - Da kans dat het morgen niet regent, maar overmorgen wel: $0,2 \times 0,3 = 0,06$.
     - Stel dat we nu twee uitspraken a en b hebben met $p(a) = 0,8$ en $p(b) = 0,2$.
     - Wat is de kans op $p(a$ **en** $b)$? Deze waarde kan tussen 0 en 0,2 liggen.
       Als a = het regent morgen en b = het regent niet morgen, dan is $p(a$ **en** $b) = 0$.
       Als a = het regent morgen en b = het regent morgenochtend, dan is $p(a$ **en** $b) = 0,2$
     - **!** Bayesiaans redeneren is geen aangewezen methode.
  3. Theorie van zekerheidsfactoren:
     - Er wordt een zekerheidsfactor berekent op basis van woorden die een expert uitdrukt:

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

     - Voor een uitspraak a wordt de zekerheidsfactor van die uitspraak $z(a)$. Hierbij geldt $z(a) = -z(\neg a)$.
     - Hoe berekenen van de zekerheid van een conclusie als de premisse van een regel zelf onzeker is?

       als b
       dan a (x)

       $z(a) = z(b) \times x$

als b **en** c **en** d
dan a (x)

$$z(b \textbf{ en } c \textbf{ en } d) = \min(z(b), z(c), z(d)) \times x$$

als e
dan a (x)

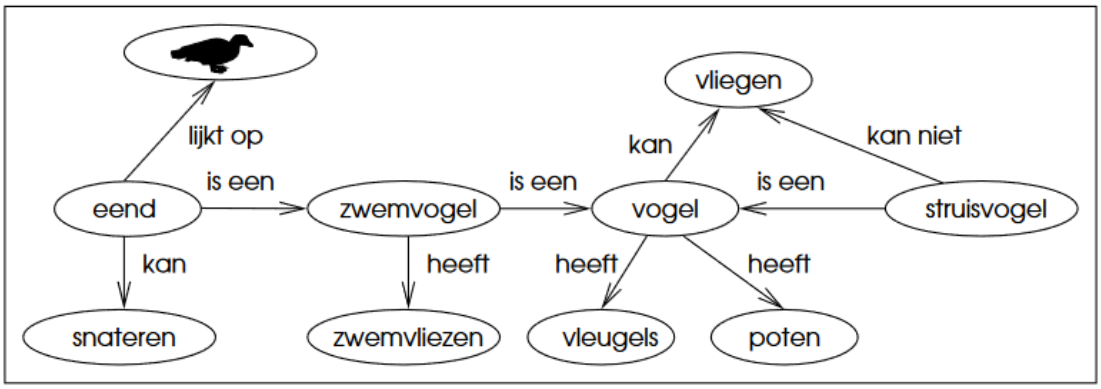
als f
dan a (y)

$$z_1 = z(e) \times x, \qquad z_2 = z(f) \times y$$

$$z(a) = \frac{z_1 + z_2}{1 - \min(|z_1|, |z_2|)}$$

3.4 Frames en regelschema's

- Noodzaak om kennis in te delen.
- Een **frame** kan vergeleken worden met een klasse uit object georiënteerde programeertalen.
- Komt uit de theorie van **semantische netten**.



Figuur 3.1: Een semantisch net.

  - Een semantisch netwerk bestaat uit een gelabelde gerichte multigraaf.
  - Elke knoop is een begrip, en elk begrip is verbonden met andere begrippen.
- Opzoeken van informatie:
  - *Kan een eend snateren?* Vanuit 'eend' wordt de 'kan'-relatie naar 'snateren' genomen. Het antwoord is 'ja'.
  - *Heeft een eend zwemvliezen?* Via de 'is een'-relatie weet men dat een eend een zwemvogel is. De zwemvogel bevat een 'heeft'-relatie naar 'zwemvliezen'. Het antwoord is 'ja'.

- *Heeft een eend vleugels?* Via de twee 'is een'-relaties weet men dat een eend een vogel is. Een vogel bevat een 'heeft'-relatie naar 'vleugels'. Het antwoord is 'ja'.
- *Eet een eend sla?* Na het netwerk te hebben afgezocht is er geen verband tussen sla en eenden gevonden.

- Twee regels om informatie op te zoeken:
  1. Volg de takken waarvan de naam in de vraag staat.
  2. Volg de takken met 'is een' als naam. Deze zorgt ervoor dat het probleem veralgemeend wordt (eend wordt zwemvogel en dan vogel), maar het wordt enkel gebruikt als er geen andere informatie meer beschikbaar is via andere relaties.

- *Kan een struisvogel vliegen?* We vinden informatie met de 'kan niet'-relatie zodat het antwoord 'nee' is.

! Het net bevat enkel informatie en geen regels voor de verwerking ervan.

- Frames versus objectgeoriënteerd programmeren:
  1. Ze hebben beiden een hiërarchisch systeem dat gebruik maakt van (geen meervoudige) overerving.
  2. In een semantisch net wordt er geen onderscheid gemaakt tussen klassen en objecten.

Hoofdstuk 4

De regel van Bayes en Markovketens

4.1 Probabiliteit

- Een universum U bevat een eindig aantal toestanden.
- De kans op een uitspraak a wordt genoteerd als $p(a)$ en is gelijk aan:

$$p(a) = \frac{\#(a)}{\#U}$$

met $\#U$ het aantal toestanden in het universum en $\#(a)$ het aantal toestanden waarin a waar is binnen dit universum.

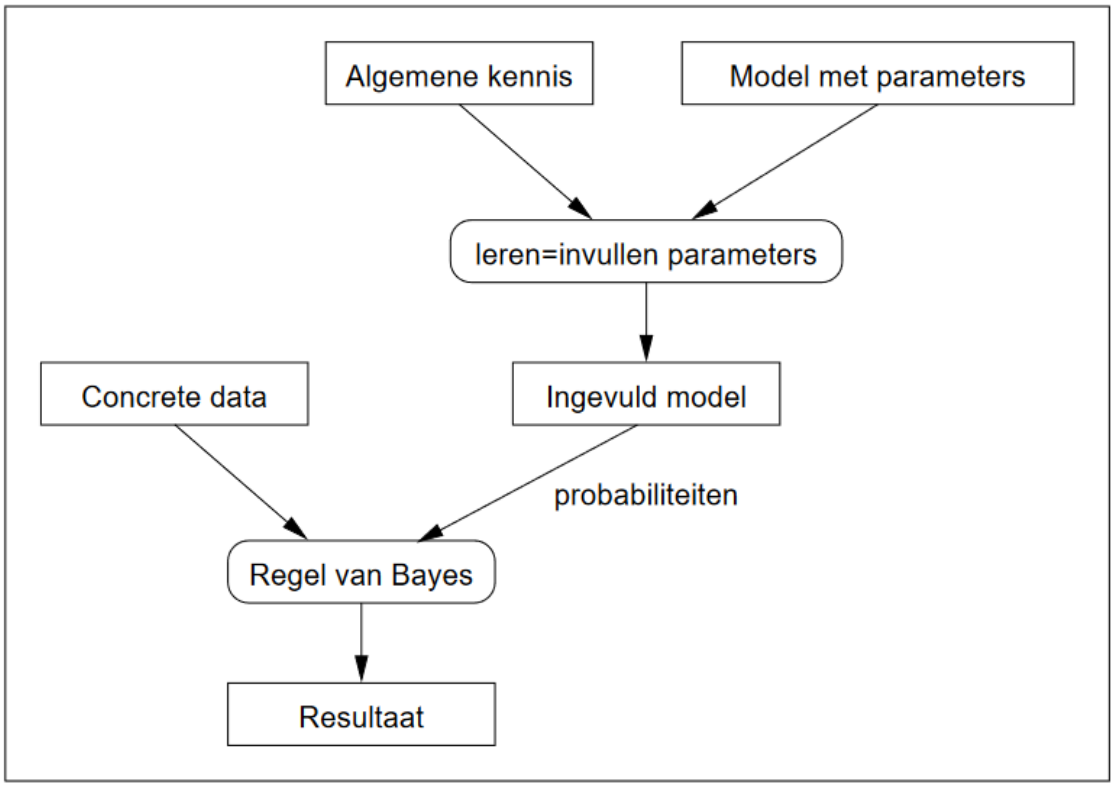
! Niet bruikbaar: het systeem kan enkel de kans bepalen van reeds bestaande gevallen.

1. ◇ Stel een observatie: 83-jarige patiënt met hoofdpijn.
   ◇ In het universum zit er geen 83-jarige patiënt met hoofdpijn.
   ◇ De patiënt kan dus niet bestaan.
2. ◇ Stel 43-jarige patiënt die niet rookt en metselaar is.
   ◇ In het universum zit één metselaar die 43 jaar is, niet rookte en psoriasis heeft.
   ◇ De patiënt heeft dus psoriasis.

- Er is een **zinvolle veralgemening** nodig.
- Het lerend programma moet een **model** gebruiken waarbij een aantal parameters moet worden ingevuld.
- De waarden van de parameters van dit model kan een zinnige veralgemening geven van probailiteit, via **de regel van Bayes**.

4.2 De regel van Bayes

- De voorwaardelijke kans $p(a|b)$ is de kans dat a waar is gegeven dat b waar is.
- Stel $\#U$ het aantal toestanden in het universum en $\#(a)$ het aantal toestanden waarin a waar is:



Figuur 4.1: De regel van Bayes bij artificiële intelligentie.

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

- Dit kan herschreven worden:

$$p(b)p(a|b) = p(a\&b)$$

zodat

$$p(a\&b) = p(b\&a)$$

of

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

- Voorbeeld:
  - Universum = 10.000 vogels
  - 1.000 van deze vogels zijn raven: $p(raaf) = 0,1$
  - 2 van deze vogels zijn wit: $p(wit) = 0,0002$
  - Er is slechts 1 witte raaf: $p(raaf|wit) = 0,5$ en $p(wit|raaf) = 0,001$
  - **Dus:** $p(wit)p(raaf|wit) = p(raaf)p(wit|raaf) = 0,0001$
- Dit kan veralgemeend worden met **de regel van Bayes**:

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

- Veronderstel hypothetische uitspraken $h_i, i = 1, 2, ...$
  - Voor een willekeurig item is exact één hypothetische uitspraak waar.
  - Het resultaat van een aantal metingen op dat item resulteren in een uitspraak d.
  - Wat is de kans dat een hypothese h_i waar is gegeven de data d?

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

- Voorbeeld:
  - Er is een groot aantal ziektes.
  - Deze leiden tot de lijst van hypothesen, waarbij h_i staat voor de uitspraak "De patiënt heeft het $i-$de ziektepatroon".
  - Is er een groot aantal mogelijke symptomen.
  - Bij een patiënt kunnen de symptomen vastgesteld worden, die resulteren in een uitspraak d.
  - Hoe bepalen welk ziektepatroon het meest waarschijnlijk is?
    - $p(h_i)$ is de kans dat ziektepatroon i voorkomt in het universum. Dit is verondersteld gekend te zijn.
    - $p(d|h_i)$ is de kans dat ziektepatroon i de opgegeven symptomen veroorzaakt. Dit is verondersteld gekend te zijn.
    - $p(d)$ kan berekend worden uit de vorige gegevens, als er verondersteld wordt dat er bij elke patiënt juist één i is waarvoor h_i waar is:

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

    - De expliciete waarde van $p(d)$ is eigenlijk niet nodig aangezien deze onafhankelijk is van i. We weten dat d waar is uit de waarnemingen en is dus een constante factor.
- $p(h_i)$ is de **a prioriverdeling**: de waarschijnlijkheid dat h_i waar is vooraleer de data d bekend is.
  - Als deze niet gekend is, moet ze geschat worden.
  - Bijvoorbeeld via uniforme verdeling, elke hypothese krijgt dezelfde waarschijnlijkheid.
    - ! Slechts een noodoplossing.
- $p(h_i|d)$ is de **a posterioriverdeling**: de waarschijnlijkheid dat h_i waar is wanneer de data d gekend is.

4.3 Markovketens en -modellen

- Een speciaal soort automaat met twee essentiële verschillen:
  1. Een Markovketen is een **producerende** eenheid: het genereert uitvoer zonder een specifieke invoer;
  2. Een Markovketen heeft een **stochastische** werking: de overgang naar een andere staat is gebaseerd op kans.
- Wordt gebruikt om tijdsafhankelijke processen te modelleren.
- Formele definitie van een **Markov model**:

1. Een eindige verzameling van **staten** $S = \{s_1, ..., s_n\}$. Er is een beginstaat (s_1) en eventueel een eindstaat (s_n).
2. Een **transitiematrix** T. Dit is een $n \times n$ matrix die de probabiliteit van een toestand s_i naar een andere toestand s_j bevat.
3. Een **uitvoeralfabet** $A = \{a_1, ..., a_{k-1}\} \cup \{a_k\}$. Hier is a_k het lege symbool.
4. Een **uitvoermatrix** U. Een $a \times k$ matrix. Voor i, de huidige staat, en t, het huidige tijdstip, dan geeft U_{ij} de kans op j als uitvoer.

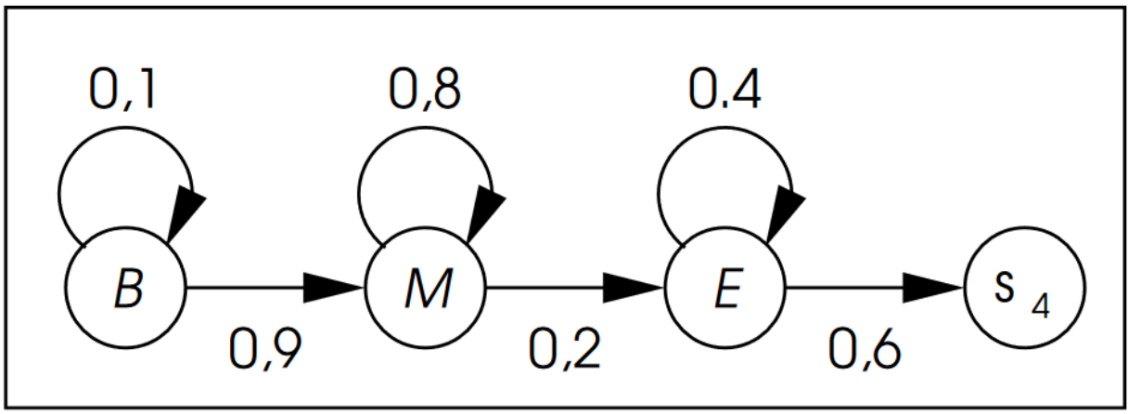
- Een Markovketen kent een probabiliteit toe aan elke string van karakters in A.
- Een Markovketen produceert een string, maar niet noodzakelijk altijd dezelfde. Het is **niet-deterministisch**, maar de kans op een bepaalde string kan wel berekent worden.
- Op elk ogenblik is de Markovketen in een bepaalde staat. Voor $t = 0$ is dit s_1.
- De keten activeren levert een functie $i : \mathcal{N} \to \{1, ..., n\}$ op die voor elk moment t de index van de staat op dat moment geeft. Op een ogenblik t is de staat $s_{i(t)}$.
- De probabiliteiten om van één staat naar een andere te gaan worden in de transitiematrix T beschreven:

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

  Voor een willekeurige k geldt (som der kansen):

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

- De probabiliteit is onafhankelijk van de tijd t.
- De probabiliteit om op tijdstip $t + 1$ een bepaalde staat te bereiken hangt alleen af van de staat op tijdstip t.
- De uitvoer van een Markovketen wordt geschreven als $a_{j(t)}$.
- Als op tijdstip t de staat s_i is, dan is de waarschijnlijkheid dat a_j de uitvoer is gelijk aan U_{ij}.
- Er bestaat dus een één-op-één relatie tussen staat en uitvoer. Hieruit volgt $k = n$, $U_{ii} = 1$ en $U_{ij} = 0$ als $i \neq j$.
- Bij een **Hidden Markov Model (HMM)** kan een staat verschillende mogelijke uitvoeren hebben, en kan een uitvoer horen bij verschillende staten.
- Voorbeeld: Een HMM toegepast op spraakherkenning.
  - Er bestaat een HMM op drie niveaus:
    1. Voor elk *foneem*. Dit zijn betekenisvolle klankenreeksen die soms met een letter overeenkomen en soms met een lettergroep.
    2. Voor elk *woord*. Een woord is een opeenvolging van fonemen.
    3. Voor alle mogelijke zinnen samen.
  - Een aantal veronderstellingen:
    - Sommige woorden kunnen uitgesproken worden met verschillende fonemen. Hier worden deze als verschillende woorden beschouwd.
    - Homoniemen worden als hetzelfde woord beschouwd: meid, mijd, mijdt, mijt, ...
    - Hetzelfde foneem kan traag of snel uitgesproken worden. Deze variatie van lengte wordt aangegeven als de probabiliteit om in dezelfde staat te blijven.



Figuur 4.2: Het HMM voor het foneem 't'.

1. Het meest gebruikelijke modelvoor een foneem maakt gebruik van drie staten: B, M en E (Begin, Midden en Eind), gevolgd door een eindstaat s_4. Figuur 4.2 toont het HMM voor het foneem 't'.
2. Het HMM voor een woord wordt gevormd door de HMM's voor zijn fonemen aaneen te schakelen. Als een woord w_i de fonemen f_i heeft, dan heeft het HMM 3 f_i staten (eindstaat wordt niet mee gerekend).
3. Er moet ook nog een model opgebouwd worden om de waarschijnlijkheid van zinnen te modelleren. Dit is echter zeer complex en hierbij wordt er gebruik gemaakt van ***n*-grammodellen**.
   - ◇ Een n-gram is een sequentie van n woorden.
   - ◇ Als de probabiliteit van alle n-grammen gekent is, dan kan dit gebruikt worden om bij $n-1$ woorden het volgende woord te voorspellen.
   - ◇ Een **unigram** is een 1-gram dat dan de relatieve frequentie van alle woorden bevat.
   - ◇ Een **bigram** is een 2-gram bevat alle mogelijke combinaties van twee opeenvolgende woorden.
   - ◇ Een bigram heeft een beginknoop. Deze beginknoop is verbonden met alle beginknopen van woorden en genereert zelf geen uitvoer. De overgangswaarschijnlijkheid wordt gegeven door de unigrammen. 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.
   - ◇ Stel W het aantal woorden en f het gemiddeld aantal fonemen per woord, dan is het totaal aantal staten $3fW$.

- Een HMM kan gebruikt worden om de waarschijnlijkheid van een bepaalde uitvoer te berekenen. Maar wat als we willen weten hoe de uitvoer gegenereerd is?
  - ◦ De functie $j(t)$ is gekend, die voor tijdstip t aangeeft dat de uitvoer $a_{j(t)}$ was.
  - ◦ De functie $i(t)$, die de staat geeft als functie van de tijd t, kan hier niet uit afgeleidt worden.
  - ◦ Op een tijdstip t is de uitvoer a_j, dus weten we enkel dat op tijdstip t de staat s_i moet zijn met $U_{ij} \neq 0$.
  - ◦ We zoeken een indexfunctie $i(.)$ die de meest waarschijnlijke is, gegeven de uitvoer $j(.)$.
  - ◦ We zoeken $i(.)$ die $p(i(.)|j(.))$ maximaliseert. Volgens Bayes:

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

- $p(j(.))$ is onbelangrijk aangezien deze hetzelfde is voor alle mogelijke sequenties.
- $p(i(.))$ kan berekend worden door het product te nemen van de overgangswaarschijnlijkheden $T_{i(t),i(t+1)}$ en T het totaal aantal tijdstappen.
- $p(j(.)|i(.))$ is de waarde $U_{i(t)j(t)}$

- De formule van bayes kan dan gesubstitueerd worden, met $\alpha = \frac{1}{p(j(.))}$:

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

- Om $p(i(.)|j(.))$ te maximaliseren kan het probleem opgevat worden als het zoeken van een optimaal pad doorheen een gerichte graaf.

- Constructie graaf:
  - De knopenverzameling van de graaf is de verzameling koppels (s_i, t) waarbij $0 \leq t \leq T$.
  - Er is een verbinding van (s_i, t_1) naar (s_j, t_2) als $t_2 = t_1 + 1$ en er een verbinding is van s_i naar s_j in het HMM (dus $T_{ij} \neq 0$).
  - Probleem: in klassiek optimaal pad wordt de kost zo klein mogelijk gemaakt, terwijl we hier een zo groot mogelijke probabiliteit willen bereiken. De kost tussen twee knopen kan berekend worden via:

$$c((s_i, t) \to (s_k, t+1)) = -\log_2(T_{ik} \cdot U_{kj(t+1)})$$

  De graaf doorzoeken gebeurd via het **Viterbialgoritme**:
  - Loop de graaf tijdslaag per tijdslaag af.
  - Als we voor een waarde t alle knopen (s_i, t) ontwikkeld hebben, dan kennen we voor elke knoop $(s_i, t+1)$ de geschatte kost g voor de kortste weg van de startknoop naar de huidige knoop.
  - We zoeken een functie e zodanig dat $log_2(e((s_i, t))) = g((s_i, t))$.
    1. Voor $t = 0$, initialiseer $e((s_1, 0)) = 1$ en $e((s_i, 0)) = 0$ voor $i \neq 1$.
    2. Voor $t = 1, 2, ...,$ en voor alle staten s_i, zoek de k die de functie $e((s_k, t-1))T_{ki}$ maximaliseert, en stel

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

4.4 Leren van het model

Hoofdstuk 5

Actieve Systemen

- Klassieke regelbanken worden gebruikt voor expertsystemen, die toelaten om conclusies te trekken uit bepaalde gegevens.
- Er kan ook een systeem ontwikkeld worden waar het geheel

5.1 Eenvoudige systemen

- Bij een eenvoudig systeem is het mogelijk een opsomming te maken van alle visies, acties en hun combinaties.
- Bij elke tijdsstap analyseert de actieve eenheid de visie en kiest op basis daarvan een actie. De buitenwereld reageert op deze actie en genereert een nieuwe visie.

! Een visie geeft slechts gedeeltelijke beschrijving van de wereld. Een **Hidden Markov Decision Model (HMDM)**, wat een uitbreiding is op een **HMM**, tracht dit te modelleren en bestaat uit het volgende:

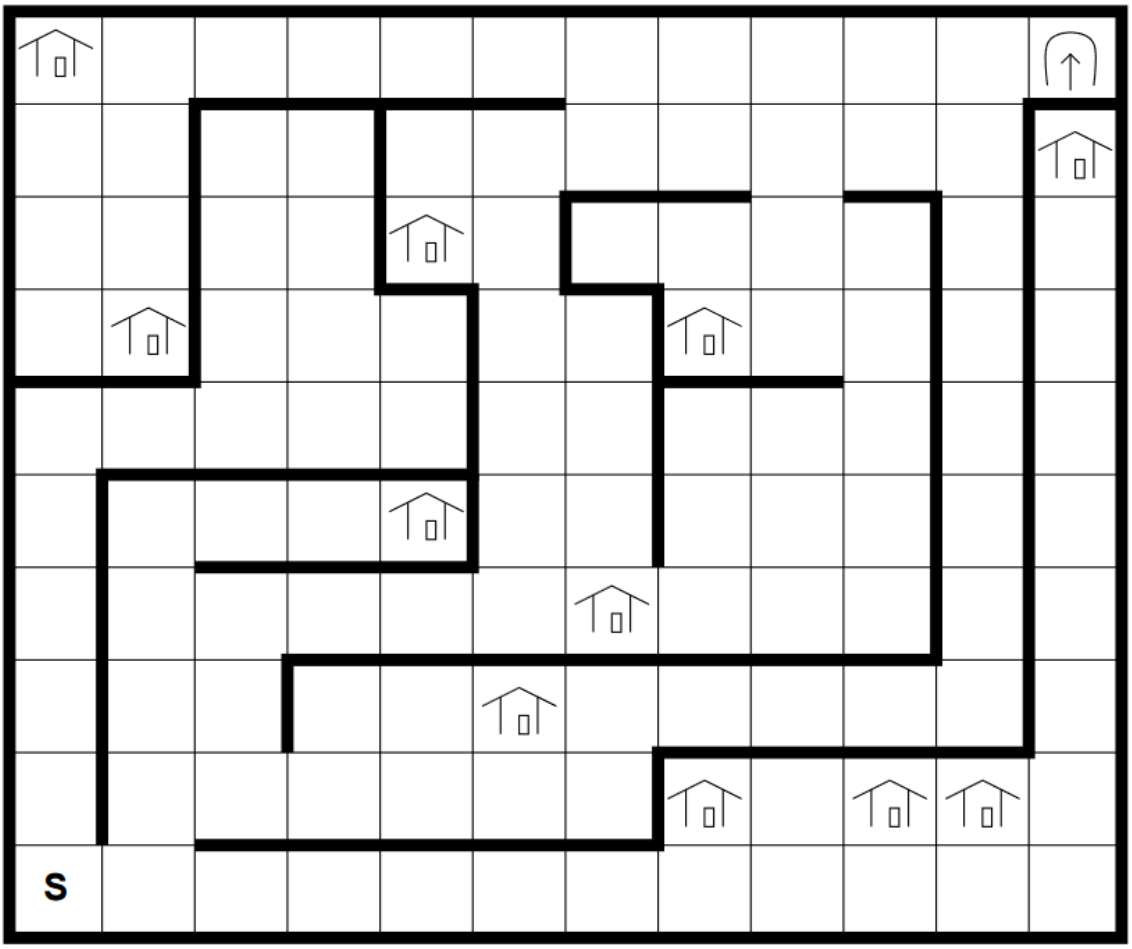
  - Een eindige verzameling van **staten** $S = \{s_1, ..., s_n\}$. Er is een beginstaat (s_1) en een eindstaat (s_n).
  - Een eindige verzameling $D = \{d_1, ..., d_l\}$ van **beslissingen**.
  - Voor elke $d \in D$ een **transitiematrix** $T(d)$.
  - Een **uitvoeralfabet** $A = \{a_1, ..., a_k\}$. Dit is het alfabet van visies.
  - Een **beloningsfunctie** $b : A \to \mathcal{R}$. Een beloning kan negatief zijn.
  - Een $n \times k$ **uitvoermatrix** U.

- Een **run** is het hele proces van begintoestand en eindtoestand.
- Een actieve eenheid moet een **strategie** τ hebben. Aan elke letter a van het uitvoeralfabet is er dan een beslissing $\tau(a)$.
- Actieve eenheden zonder onzekerheid: twee voorwaarden

  1.
     - De staat komt overeen met de visie.
     - Er is een één op één relatie tussen A en S, waarbij $k = n$ en U een diagonaalmatrix waarbij alle diagonaalelementen 1 zijn.
     - Noteer $\tau(s) = d$ als een strategie τ die in staat s beslissing d neemt.
     - Dit zorgt ervoor dat dat de H uit HMDM mag wegvallen.

2. ◦ De overgang naar de volgende staat is helemaal gedetermineerd door de beslissing.
   ◦ Elke $T(d)_{ij}$ is ofwel 1 ofwel 0.
   ◦ De transitie kan uitgedrukt worden als $T(d, s) = s'$ als d leidt tot een overgang naar staat s'.
   ◦ We krijgen een deterministisch systeem waarbij een staat s in één tijdsstap overgaat in de staat $T(\tau(s), s)$.

- Voorbeeld: figuur 5.1



Figuur 5.1: Een eenvoudige leerwereld.

◦ Een doolhof met startpositie s.
◦ Op elke plaats kan de eenheid beslissen één vakje verder te gaan in horizontale of verticale richting. De eenheid kan niet door dikke lijnen heen.
◦ Het verplaatsen van een vakje naar een volgend vakje kost 1 dag.
◦ De eenheid kan maar voor 10 dagen verplaatsingen doen.
◦ Een hut kan onbeperkt het aantal dagen terug opvullen tot 10.
! We gaan ervan uit dat er geen lussen zijn in het optimale pad. De eenheid zal dus niet telkens twee hutten oneindig lang blijven bezoeken.
◦ De eenheid moet nu met zoveel mogelijk dagen de uitgang (rechtsboven) weten te bereiken.

- Om een MDM te maken wordt voor elk vakje 11 staten bijgehouden. Bij elke staat wordt de coördinaten van het vakje aangevuld met het aantal dagen van de eenheid bijgehouden.
- De beginstaat is dan $((0,0),10)$.
- Er kan aan elke staat een waarde gegeven worden:
  - Voor de eindstaten waar de eenheid sterft is de waarde -10.
  - Voor de eindstaten aan de uitgang, $((11,9),i)$ is de waarde $i+1$.
  - Voor de andere vakjes is er een verband tussen de waardering voor de staat door eenheid en de gevolgde strategie.
- In het algemeen geval wordt de formule gegeven:

$$\omega_\pi(s) = b(s) + \omega_\pi(T(\pi(s), s))$$

  - De strategie π beïnvloedt de waarde $\omega_\pi(s)$ als π toegepast wordt op een staat s.
- Aangezien er geen onzekerheid is kan een optimale strategie τ_{opt} gevonden worden zodat $\omega_{\tau_{opt}}(s) \geq \omega_\tau(s)$. De beslissing $\tau_{opt}(s)$ leidt dan tot een staat s' die voldoet aan

$$\omega_{\tau_{opt}}(s) = b(s) + \max_{d \in D} \omega_{\tau_{opt}}(T(d, s))$$

- Het kan zijn dat er meerdere optimale strategieën bestaan. De formule kan dan herschreven worden door het maximum van alle strategieën te pakken, zodat de keuze van strategie niet meer uitmaakt:

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

Voor elke optimale strategie geldt dat $\omega_{\tau_{opt}}(s) = \omega^*(s)$

5.1.1 Exploratie zonder onzekerheid

- Een **exploratiestrategie** laat de eenheid leren door te zien wat er gebeurt.
- Een klassieke strategie neemt elke keer dezelfde beslissing voor elke staat.
- Een exploratiestrategie ξ kent aan elke combinatie (s,d) een probabiliteit $\xi(s,d)$ toe. Hierbij geldt:

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

- Als de eenheid in staat s terechtkomt, wordt beslissing d gekozen op basis van de probabiliteit.
- De totale opbrengst W kan voor een specifieke run bepaald worden. Voor alle staten die de eenheid gepasseerd is, kan de benaderde schatting $\omega(s)$ van $\omega^*(s)$ vervangen worden door

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

5.1.2 Exploratie met onzekerheid

- Hierbij is er geen één op één relatie tussen staat en visie. De overgang naar een nieuwe staat is slechts probabilistisch afhankelijk van de genomen beslissing.
- Gegeven een staat s_i en een beslissing d, dan is de waarschijnlijkheid om over te gaan naar s_j gelijk aan $T(d)_{ij}$.

! Dit heeft als gevolg dat de winst van een strategie π niet exact kan voorspeld worden als we uit een staat s vertrekken.

○ Er kan wel een **verwachtingswaarde** berekend worden:

$$\omega_\pi(s_i) = b(s_i) + \sum_j T(\pi(s_i))_{ij}\omega_\pi(s_j)$$

- Een lerende eenheid kent het systeem echter niet. Het moet niet alleen een schatting maken van de waarde van alle staten, maar ook van de overgangswaarschijnlijkheden $T(d)_{ij}$. Stel A_{ij} het totaal aantal keer dat de eenheid in staat s_j is geraakt vanuit staat s_i door d te gebruiken, dan wordt volgende schatting gebruikt:

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

5.2 Complexe systemen

- Bij eenvoudige systemen is er informatie beschikbaar over elke staat. Bij systemen met onzekerheid moet zelf voor elke combinatie (s, d) de overgangswaarschijnlijkheid naar elke staat bijgehouden worden.

! Niet haalbaar voor een groot aantal staten.

- Invoering van **actieve regelbanken**, deze bevat regels van de vorm

als premisse
dan voer actie uit

✓ Eenvoudiger dan klassieke regelbanken: een actie kan geen deel uitmaken van de premisse van een andere regel.

- Om de actieve regelbank zo klein mogelijk te houden worden regels zo algemeen mogelijk genomen. In het geval van conflicten zoals

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

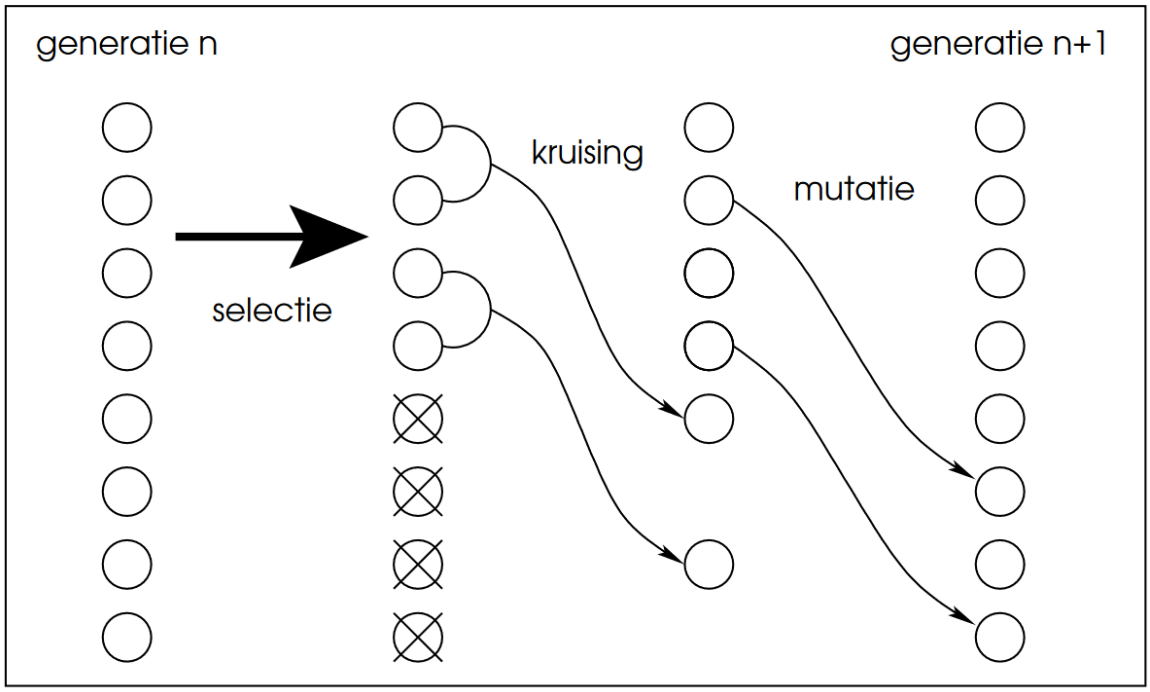
als $b == 2$
dan voer actie 2 uit

wordt meestal de regel gekozen die het minst algemeen is. In sommige gevallen wordt er ook randomisatie gebruikt om variatie te hebben tussen acties.

- Actieve regelbanken worden gebruikt bij genetische algoritmen.

5.3 Genetische algoritmen

- Start vanuit een grote groep van strategieën, de **populatie**.
- Wijzig nu de populatie door strategieën te verwijderen, toe te voegen of aan te passen zodat uiteindelijk een strategie gevonden is die efficiënt is.



- We maken gebruik van **generaties**. Elke generatie bestaat uit een populatie die even groot is. De volgende generatie wordt bekomen in twee stappen:
  1. De populatie wordt uitgedund zodat enkel de beste strategieën overblijven.
     - Er moet een **geschiktheidsfunctie** gedefinieerd worden.
  2. Gebaseerd op de overblijvende strategieën wordt de populatie terug aangevuld. Hier kan er kruising of mutatie toegepast worden.
     - Een **kruising** neemt twee exemplaren uit de populatie en vermengt deze, zodat de nakomeling eigenschappen heeft van de ouders.
     - Een **mutatie** voert een kleine wijziging door aan een strategie uit de populatie.
- Voorbeeld: uurroosterprobleem
  - Er zijn een aantal studentengroepen die elk een aantal cursussen volgen die gegeven worden door bepaalde docenten. Verder zijn er klaslokalen.
  - Er moet een uurrooster opgemaakt worden dat aan een aantal eisen voldoet. Er zijn twee soorten eisen:
    1. **Harde eisen**:
       - Elke cursus moet gegeven worden.
       - Een docent kan maar één cursus geven op een moment.
       - Een studentengroep kan maar les volgen op één moment.
       - Elke les moet doorgaan in een lokaal dat geschikt is voor de les.
       - Er kunnen geen twee lessen tegelijk doorgaan in hetzelfde lokaal.
    2. **Zachte eisen**:
       - Er moeten zo weinig mogelijk springuren zijn.
       - De lokaalwissels moeten beperkt blijven.
       - enz...
  - De beginpopulatie kan nooit aan alle voorwaarden voldoen. Daarom worden onmogelijke uurroosters toegelaten, maar deze krijgen een slechte geschiktheid.

1. Voor elke harde eis die niet voldaan is worden duizend strafpunten afgetrokken.
2. Voor elke zachte eis die niet voldaan is wordt één punt afgetrokken.

5.4 Optimalisatie van één strategie

- Elke regel krijgt een waardering $\omega(r)$, die aangeeft wat de verwachte winst is wanneer r toegepast wordt, als ervan uitgegaan wordt dat voor de rest van de oplossing dezelfde strategie gebruikt wordt.
- Een regel die uitgevoerd wordt kan ook de waarde van andere regels wijzigen.
- Het **leerproces**:
  - Normaal gezien zijn overlappende regels slecht, maar hier zijn ze nuttig:
    - Als er twee regels zijn met dezelfde premise maar andere actie, wordt diegene bijgehouden die de hoogste waardering oplevert.
    - Als er twee regels zijn met verschillende premissen maar dezelfde acties, waarbij premisse 1 meer algemeen is dan premisse 2, dan behouden we premisse 1 als er visies zien waarbij de tweede regel niet van toepassing is, anders premisse 2.
  - Twee fasen:
    1. De **exploratiefase**:
       - Er is een **regelverzameling** $A(v)$ van een visie.
       - Dit is de verzameling regels waarvan de premisse beantwoord aan v.
       - Als we in visie v zitten nemen we een regel uit $A(v)$, en voeren de daarbijhorende actie α uit. De regel kan gekozen worden door bijvoorbeeld een random keuze te maken met de waarderingen van de regels als gewicht.
       - De actie α zorgt voor een nieuwe visie v'.
       - Om na te gaan hoe goed deze actie is, moet er een waarde toegekend worden aan v'.
       $$\omega(v') = b(v') + \max_{r \in A(v')} \omega(r)$$
       - De verzameling $A(v)$ wordt in twee delen opgesplitst:
       1. Elke regel r in $A(v)$ die als actie a voorstelt, kan de waardering $\omega(r)$ door onzekerheid aangepast worden met een vergeetfactor $\alpha, 0 < \alpha < 1$:
       $$\omega(r) \to (1 - \alpha)\omega(r) + \alpha\omega(v')$$
       2. Voor elke andere regel r in $A(v)$ die een andere actie al a voorstelt, vermindert de waardering met een vergeetfactor β die kleiner is dan α:
       $$\omega(r) \to (1 - \beta)\omega(r)$$
    2. De **regelaanpassingsfase**:
       - Regels met een lage waardering worden verwijderd, vaak door ze te veranderen.
       - Er kan bijgehouden worden hoe vaak regels gebruikt werden in de exploratiefase. Deze hebben minder kans om verwijderd te worden.
       - Als een regel soms heel goed en soms heel slecht presteert, dan kan de regel **gesplitst** worden:

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

Er kan een variabele ingevoerd die tracht het goede en het slechte deel te scheiden:

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

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

- Regels kunnen ook **samengevoegd** worden. Als twee regels dezelfde actie voorstellen, en een grote overlapping hebben kan dit nuttig zijn.
- De premissen van regels die goed presteren kunnen ook verruimd worden.

5.5 Combinaties

Hoofdstuk 6

Neurale netten

- Een neural netwerk is een informatieverwerkend geheel dat de vorm heeft van een **gewogen gerichte graaf**.
- De knopen van deze graaf worden **neuronen** genoemd.
- Deze knopen sturen signalen naar alle verdere knopen in de 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.

6.1 Vergelijking met computers

- Bij een **computer**:
  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 iemand moet geweest zijn dat die algoritme kende.
  2. De computer is zeer gevoelig voor onverwachte invoer: een fout in de invoer kan enkel herstelt worden als het algoritme op deze fout voorzien is.
  3. De kleinste fout in apparatuur kan ertoe leiden dat de uitvoer corrupt wordt.
  4. Elk begrip en object van de verwerking is duidelijk localiseerbaar in het geheugen.
- Bij een **neuraal net**:
  1. Een neuraal netwerk wordt niet geprogrammeerd meer leert.
  2. Een neuraal netwerk is zeer goed in veralgemenen: het kan invoer verwerken die lijkt op de al geziene invoer.
  3. Een neuraal netwerk is ongevoelig voor beschadigingen.
  4. Objecten zijn niet localiseerbaar.
- Twee manieren om kunstmatige neurale netwerken te maken:
  1. **Hardwarematig**: er worden elektronische componenten geconstrueerd die dezelfde functies hebben als een natuurlijk neuron.
  2. **Softwarematig**: de werking van een neuraal netwerk wordt gesimuleerd met een programma.

6.2 De biologische grondslagen

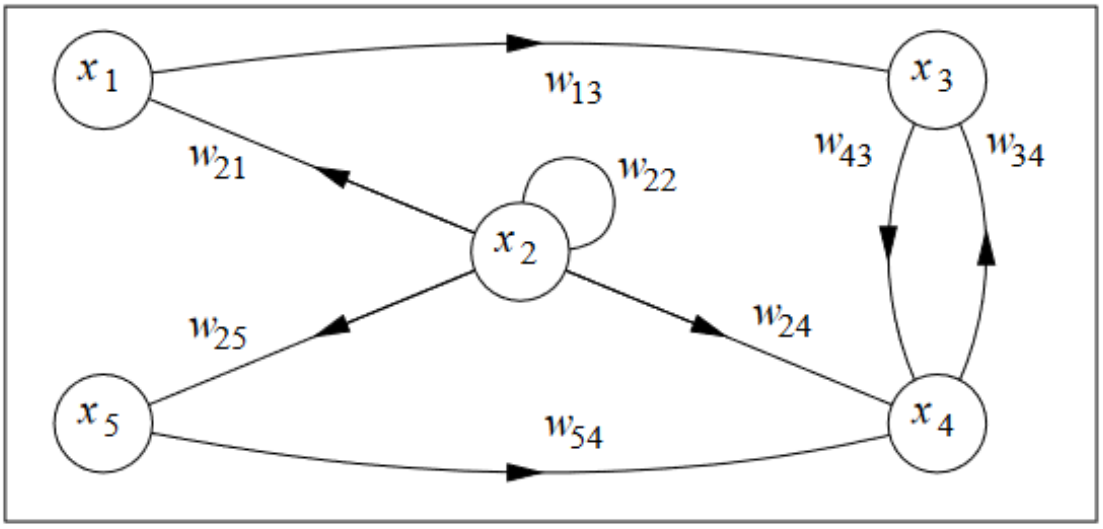
- Zenuwstelsel bestaat uit zenuwcellen (neuronen).
- Er zijn drie soorten neuronen:
  1. **Invoerneuronen**: Deze zetten de signalen van de buitenwereld om in signalen die verwerkt kunnen worden door andere neuronen.
  2. **Interneuronen**: Deze verwerken de informatie.
  3. **Uitvoerneuronen**: Deze geven informatie door aan de buitenwereld.
- Structuur van een interne neuron:
  - De **soma** is het centrale deel.
  - De **dendrieten** zijn vertakkingen die vertrekken uit de soma.
  - De **axon** is een lange, enkelvoudige, vertakking die ook vertrekt uit de soma.
    - Uit de axon kunnen andere vertakkingen zich voordoen.
    - Elke vertakking eindigt met een synaps, die de vertakking verbindt met de dendriet van een ander neuron.
- Structuur van een invoerneuron:
  - Idem interne neuron maar zonder dendrieten.
- Structuur van een uitvoerneuron:
  - Idem interne neuron maar de axon is vervangen met een specifieke verbinding voor de functie van die zenuwcel.
- De puls is een elektronische stroomstoot. Deze loopt doorheen de axon en komt aan bij alle synapsen van dit axon.
- Er zijn twee situaties wanneer een neuron een puls kan afvuren:
  1. Een **excitatorische** of **prikkelende** puls ??
  2. Een **inhibitorische** of **remmende** puls ??

Hoofdstuk 7

Kunstmatige Netwerken

Enkele notaties:

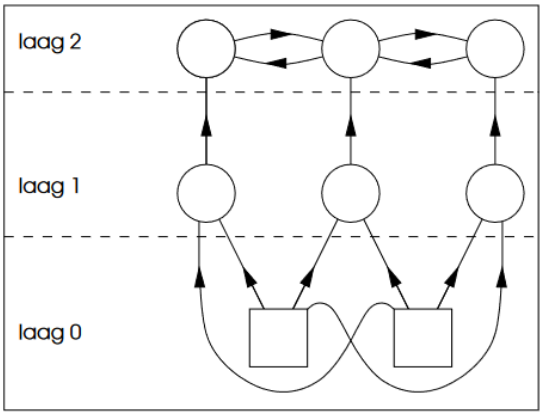
- De neuronen van het netwerk worden aangeduid met de letters x, y of z. Deze worden genummerd door een onderindex i.
- De verbindingen in de graaf krijgen geen apart symbool. Het gewicht van de tak die van x_i naar x_j gaat noteren we als w_{ij}; in het geval dat er geen verbinding is, is $w_{ij} = 0$.
- De uitvoer van neuron x_i duiden we aan met u_i.
- Een rij getallen die een vector voorstellen worden geschreven met een vette letter: $\mathbf{v} = (v_1, ...v_n)$.



Figuur 7.1: Schema van een neuraal net.

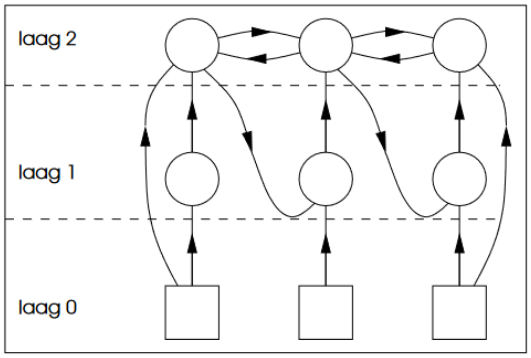
Een neuraal net is vaak ingedeeld in verschillende lagen, waarin elke laag een verschillende functie heeft. De nulde laag bestaat uit de invoercellen. Er zijn twee vormen:

1. **Gesloten gelaagde netten**: Er zijn enkel verbindingen mogelijk tussen opeenvolgende lagen van het net. Van laag k naar $k + 1$, of met terugkoppeling, van $k + 1$ naar k.



Figuur 7.2: Gesloten gelaagd net

2. **Open gelaagde netten**: Er zijn verbindingen mogelijk tussen twee willekeurige lagen.



Figuur 7.3: Open gelaagd net

7.1 Artificiële neuronen

Er zijn drie belangrijke kenmerken die kunstmatige neuronen kunnen onderscheiden:

1. De **reactiefunctie**: deze geeft aan wat de uitvoer is in functie van de invoer.
2. De **uitvoeringsmethode**: beschrijft hoe de verandering in uitvoer wordt doorgevoerd. Twee opties:
   (a) Direct.
   (b) Wachten op een extern signaal (bv. Hopfieldnetten).
3. De **tijd**: vaak wordt er verondersteld dat alle wijzigingen ogenblikkelijk gebeuren.

7.2 TLU's

TLU = Threshold Logic Unit

- Wordt gebruikt in binaire netwerken.
- De werking van een neuron wordt bepaald door een drempelwaarde, die voor elke neuron verschillend kan zijn.
- Als de drempelwaarde van neuron x_i gelijk is aan T_i, wordt de uitvoer gegeven door

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

! Ook mogelijk om extra neuron x_0 in te voeren, die verbonden is met elke andere neuron met gewicht $w_{0i} = -T_i$. Op die manier wordt de reactiefunctie gelijk voor alle neuronen:

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

7.3 Analoge netten

- Leunt meer aan met biologische netwerken.
- De uitvoer van een analoog neuron kan continu veranderen: als de invoer van het neuron vergroot, vergroot ook zijn uitvoer.
- De uitvoer is wel begrensd
- Vaak wordt stijgende functie gebruikt:

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

  Voor $R \to 0$ is deze functie bijna binair.

  Voor $R \to$ inf verloopt de functie vlakker.
- Andere functies met analoge eigenschappen: boogtangens.

7.4 Tijd

7.5 Leren

Hoofdstuk 8

Informatieverwerking met neurale netten

Drie vormen die zullen toegepast worden op TLU's:

- Via binaire functies.
- Via automaten.
- Via associatieve geheugens (Hopfieldnet).

Belangrijk nadeel aan TLU: het geeft geen realistisch model van de manier waarop het netwerk leert.

8.1 Binaire functies

- Alle binaire functies kunnen gerealiseerd worden met een tweelagig net van TLU's (Stelling van McCulloch en Pitts).
- Stel een aantal elementaire logische uitspraken $v_1, ...v_n$.
- Hier kunnen complexe uitdrukkingen mee gevormd worden: $\neg v_1 \wedge (v_2 \vee v_3) \Rightarrow (v_1 \Rightarrow \neg v_3)$
- Neuraal net opstellen om na te gaan of de uitspraak waar is of niet.
  - Het netwerk bestaat uit TLU's die de waarden v_i als invoer krijgen in de vorm van 0 of 1.
  - Als de stelling waar is geeft TLU 1 terug, anders 0.
- 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 $\{0,1\}^k$, waarbij n en k vooraf positief gehele getallen zijn.
- Een neuraal net heeft dan n ingangen en k uitgangen.
- Als de invoer aan het net gegeven wordt, wachten we tot alle neuronen een stabiele toestand bereiken, en zo is de uitvoer alleen afhankelijk van de invoer.
- Een net realiseert een binaire functie F als de uitvoer beschreven wordt als functie van de invoer.

- Gegeven een binaire functie F, kan er een net gevonden worden die deze functie realiseert?
  - We moeten alleen kijken naar $k = 1$ want als $k > 1$, dan kunnen k netwerken gekozen worden met 1 uitvoercomponent. Elk van deze netwerken zorgt dan voor 1 component van F.
    - Stel $n = k = 2, F : \{0,1\}^2 \to \{0,1\}^2, F_1 : \{0,1\} \to \{0,1\}^2, F_2 : \{0,1\}^2 \to \{0,1\}$

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

    - Samenvoegen van F_1 en F_2 geeft F.
  - Stel binaire functie F met invoer $(b_1, .., .b_n)$ en uitvoer $F(b_1, ..., b_n)$.
  - Het net dat we zoeken heeft n invoerneuronen en k uitvoerneuronen.
  - De laag van invoerneuronen (nulde laag) vormt geen deel van het neuraal net.
  - Dus een tweelagig net bevat de invoerneuronen, de uitvoerneuronen en nog één laag van tussenliggende neuronen.
  - Functies die met een éénlagig net kunnen gerealiseerd worden:
    1. De **OF**-functie, die (0, ..., 0) op 0 afbeeldt, en alle andere combinaties op 1 $\to$ neuron met n ingangen en $T = 0.5$.
    2. De **select**-functie die een vooropgegeven bitcombinatie $(b_1, ..., b_n)$ op 1 afbeeldt, en alle andere op 0 $\to$ neuron met n ingangen en $w_i = 2b_i - 1$.
  - **Stelling: (McCulloh en Pitts)** Zij $F : \{0,1\}^n \to \{0,1\}^k$ een willekeurige binaire 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.
    - Als F ℓ bitcombinaties op 1 afbeeldt:
      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.

8.2 Automaten

- De Mooreautomaat heeft volgende kenmerken:
  - Een eindige verzameling staten $Q = \{q_1, ..., q_{|Q|}\}$.
  - Een invoeralfabet $S = \{s_1, ..., s_{|S}\}$.
  - Een uitvoeralfabet $G = \{g_1, ..., g_{|G|}\}$.
  - Een transitiefunctie $d : Q \times S \to Q$.
  - Een uitvoerfunctie $\ell : Q \to g$.
  - Een beginstaat $q_I \in Q$.
- Aangezien we met TLU's werken is

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

- Het is tijdsafhankelijk: de uitvoer van een neuron op tijdstip t hangt af van de totale invoer op tijdstip $t-1$.
- De invoer wordt aangevoerd op even tijdstippen, $t = 0, 2, 4, ...$ De oneven tijdstippen geeft de automaat de kans om de invoer te verwerken.
- Twee veronderstellingen:
    1. Het invoeralfabet is exact de verzameling van binaire strings van lengte n bestaande uit $n-1$ nullen en één 1.
    2. Het invoeralfabet is exact de verzameling van binaire strings van lengte k bestaande uit $n-1$ nullen en één 1.

Hoofdstuk 9

Het Classificatieprobleem

- Slechts 2 klassen
  - Geen probleem indien meerdere klassen: ze kunnen beschouwd worden als een combinatie van tweeklassenproblemen.
  - Stel drie klassen A, B en C: los eerst classificatieprobleem op met A en niet$-A$, en daarna voor $niet - A$ het probleem B en C op te lossen.
- Een neuron deelt de n-dimensionale ruimte op in twee stukken, die men halfruimten noemnt.
- De scheiding tussen deze twee halfruimten is een verzameling van de vorm:

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

9.1 Harde classificatie

- Stel $\mathcal{L}$ de leerverzameling.
- Deze is verdeeld in twee klassen $\mathcal{L}^+$ en $\mathcal{L}^-$.
- $\mathcal{L}$ is lineair scheidbaar als er een halfruimte bestaat die alle punten van L^+ bevat, en geen enkele van L^-, terwijl er ook geen punten op de rand mogen liggen.

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

- Dit komt overeen met het bestaan van een neuron met n invoerkanalen, elk met gewicht w_i, en met een drempel T zodanig dat de invoer strikt positief is voor een punt uit $\mathcal{L}^+$ en strikt negatief voor een punt uit $\mathcal{L}^-$.
- Neem een polaire TLU, dan is de uitvoer +1 voor elementen uit $\mathcal{L}^+$ en -1 voor elementen uit $\mathcal{L}^-$. Dit is de eenvoudigste vorm van het **perceptron**.
- Hoe moeten de gewichten gekozen worden?
  - Om berekeningen te vergemakkelijken wordt de drempel T weggewerkt door een extra dimensie toe te voegen, waarbij elk component in die dimensie altijd de waarde 1 bevat en zodat $T = -\omega_0$.

- We zoeken nu een vector in $\mathcal{R}^{n+1}$ zodanig dat

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

- Vervang $\mathcal{L}$ door $\mathcal{P}$

$$\mathcal{P}_i = \begin{cases} \mathbf{x} & \text{als } \mathbf{x} \in \mathcal{L}_i^+ \\ -\mathbf{x} & \text{als } \mathbf{x} \in \mathcal{L}_i^- \end{cases}$$

- Als we een vector $\mathbf{x}$ uit $\mathcal{P}$ aanbieden aan perceptron, dan moet deze 1 als uitvoer hebben.
- Algoritme:
  1. Initialiseer de gewichten op 0
  2. Zolang (geen foutlees parcours en niet te veel pogingen)
  3. Voor elke vector $\mathbf{x}$ uit $\mathcal{P}$
     (a) Als $\mathbf{x} \cdot \mathbf{w} \leq 0$
     (b) Vervang $\mathbf{w}$ door $\mathbf{w} + \mathbf{x}$
- Eindigt dit algoritme?
  - ! Als we $\mathbf{w}$ vervangen door $\mathbf{w} + \mathbf{x}$ kan het zijn dat $\mathbf{x} \cdot (\mathbf{w} + \mathbf{x})$ nog altijd negatief is.
  - ! Het kan zijn dat een vector $\mathbf{v}$ uit $\mathcal{P}$, die al correct geklasseerd was, $\mathbf{v} \cdot \mathbf{x} > 0$,nu verkeerd wordt geklasseerd. Het zou kunnen dat $\mathbf{x} \cdot \mathbf{v} < 0$ en dan is $\mathbf{v} \cdot (\mathbf{w} + \mathbf{x}) < \mathbf{v} \cdot \mathbf{w}$.
  - **Stelling** Als $\mathcal{L}$ lineair scheidbaar is en $\mathcal{P}$ een eindige verzameling, dan zal het algoritme 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:
    - Stel alle gewichten op 0, en nummer het aantal keer dat een verkeerd geklasseerde vector is tegengekomen.
    - De vector bij de $k-$de keer hoort noemen we $\mathbf{x}(k)$.
    - De gewichtenvector waarmee we beginnen is $\mathbf{w}(0)$, die na de $k-$de keer $\mathbf{w}(k)$.
    - Ons algoritme stelt $\mathbf{w}(k) = \mathbf{w}(k-1) + \mathbf{x}(k)$ als $\mathbf{x}(k) \cdot \mathbf{w}(k-1) \leq 0$
    - De bovengrens wordt dan:

$$\begin{aligned} ||\mathbf{w}(k)||^2 &= \mathbf{w}(k) \cdot \mathbf{w}(k) \\ &= (\mathbf{w}(k-1) + \mathbf{x}(k)) \cdot (\mathbf{w}(k-1) + \mathbf{x}(k)) \\ &= \mathbf{w}(k-1) \cdot \mathbf{w}(k-1) + 2\mathbf{w}(k-1) \cdot \mathbf{x}(k) + \mathbf{x}(k) \cdot \mathbf{x}(k) \\ &= ||\mathbf{w}(k-1)||^2 + ||\mathbf{x}(k)||^2 \\ &= ||\mathbf{x}(1)||^2 + ...||\mathbf{x}(k)||^2 \end{aligned}$$

    - De term $2\mathbf{w}(k-1) \cdot \mathbf{x}(k)$ mag verwaarloosd worden, aangezien deze kleiner is dan nul en dus de bovengrens zeker niet zal beïnvloeden.
    - Er is nu een vector $\mathbf{y}$ in $\mathcal{P}$ met de grootste norm, $||\mathbf{y}||^2 = M$ en $||\mathbf{x}||^2 \leq M$ voor alle andere $\mathbf{x}$ in $\mathcal{P}$. Hieruit volgt

$$||\mathbf{w}(k)||^2 \leq kM$$

    - ...

! We weten niet of ons probleem lineair scheidbaar is.

✓ Wel weten we dat voor een leerverzameling $\mathcal{L}$ waarbij $\mathcal{L}^+$ en $\mathcal{L}^-$ geen gemeenschappelijke elementen hebben, dat er een drielagig net kan geconstrueerd worden dat een scheiding van $\mathcal{L}$ kan uitvoeren.

9.2 Zachte classificatie

9.3 De deltaregel

- Twee versies:

1.
   - Analoog netwerk, dat bestaat uit één neuron.
   - Leerverzameling $\mathcal{L}$ die bestaat uit meetvectoren $\mathbf{v}$ en gewenste uitvoer $y(\mathbf{v}) \in [-1, +1]$ waarbij y strikt stijgend is en overal een afgeleide heeft, en dat deze afgeleide verschilt van nul.
   - Bijvoorbeeld tangens hyperbolicus:
   $$f(r) = \frac{e^r - e^{-r}}{e^r + e^{-r}}$$
   - De afgeleide is:
   $$f'(r) = 1 - (f(r))^2$$
   - Veronderstel een meetvector $\mathbf{v}$, en een resultaat u die verschilt van $y(\mathbf{v})$.
   - De uitvoerfout is hier $e = y(\mathbf{v}) - u$.
   - De fout op de invoer van het neuron:
     - De invoer van het neuron is $a = \sum_i (w_i v_i)$ en $f(a) = u$.
     - Voor kleine waarden van δ geldt:
     $$f(a + \delta) \approx f(a) + \delta f'(a)$$
     - Neem $\delta = e/f'(a)$, dan is $f(a + \delta) \approx y(\mathbf{v})$.
     - Vervang elke $\mathbf{w}$ door $\mathbf{w} + \Delta\mathbf{w}$ zodat:
       - De nieuwe invoer van het neuron gelijk is aan $a + \delta$.
       - De wijziging zo klein mogelijk is, $||\Delta\mathbf{w}||$ is minimaal.
     - Hier volgt $(\mathbf{w} + \Delta\mathbf{w}) \cdot \mathbf{v} = a + \delta$.
     - Schrijf $\Delta\mathbf{w}$ als
     $$\Delta\mathbf{w} = A\mathbf{v} + \mathbf{y}$$
     met $\mathbf{y}$ loodrecht op $\mathbf{v}$ en (als gevolg) $A = \frac{\delta}{||\mathbf{v}||^2}$
     - $\mathbf{y}$ moet gekozen worden zodanig dat $\Delta\mathbf{w}$ minimaal is, en dat is bij $\mathbf{y} = 0$.
     - De (eerste versie) deltaregel van een neuron is:
     $$\Delta\mathbf{w} = \frac{y(\mathbf{v}) - u}{f'(a)||\mathbf{v}||^2}\mathbf{v}$$
   - Gegeven een invoervector $\mathbf{v}$.
   - Stel een kleine wijziging τ voor gewicht w_i.
   - Hoe wijzigt u, de uitvoer van het netwerk?
   - De invoer verandert: $a = a + \tau v_i$.
   - De uitvoer verandert: $f(a + \tau v_i) \approx f(a) + \tau v_i f'(a)$.
   - De factor $v_i f'(a)$ is de partiële afgeleide van u naar w_i, die genoteerd wordt als $\partial_{w_i} i$.
   $$\partial_{\mathbf{w}} u = (\partial_{w_0} u, ..., \partial_{w_n} u)$$
   - Hierbij is $\partial_{\mathbf{w}} u = f'(a)\mathbf{v}$
   - De (tweede versie) deltaregel van een neuron is:
   $$\Delta\mathbf{w} = \frac{y(\mathbf{v}) - u}{||\partial_{\mathbf{w}} u||^2}\partial_{\mathbf{w}} u$$
   - ✓ Kan ook werken om een meerlagig net zonder terugkoppeling te laten leren.
   - Eventueel een leerfactor A toevoegen, om al te grote schommelingen te vermijden.

Hoofdstuk 10

Steunvectoren

- Andere aanpak voor harde classificatie voor lineaire en niet-lineaire problemen.
- Soms is er ruis en is het onbepaald hoeveel neuronen er nodig zijn.
- Hier zien we SVM's (Support Vector Machine) die de vorm hebben van een stemmachine.

10.1 Basisprincipes

- Terug een leerverzameling $\mathcal{L} = \mathcal{L}^+ \cup \mathcal{L}^-$, of $\mathcal{L} = \{\mathbf{x_1}, ..., \mathbf{x_n}\}$
- Een stemmachine heeft een gelijkaardigheidsfunctie g.
  - Geeft aan hoe goed twee items op elkaar lijken.
  - Voor gegeven $\mathbf{x}$ en variërende $\mathbf{z}$ heeft $g(\mathbf{x}, \mathbf{z})$ de grootst mogelijke waarde voor $\mathbf{x} = \mathbf{z}$ en neemt de waarde af naarmate $\mathbf{z}$ minder op $\mathbf{x}$ begint te lijken.
  - Bijvoorbeeld:
    $$g(\mathbf{x}, \mathbf{z}) = -||\mathbf{x} - \mathbf{z}||^2$$
    waarbij $g(\mathbf{x}, \mathbf{z}) = g(\mathbf{z}, \mathbf{x})$
- Basisidee: Vectoren die meer lijken op vectoren uit $\mathcal{L}^+$ worden positief geklasseerd. Vectoren die meer lijken op vectoren uit $\mathcal{L}^-$ worden negatief geklasseerd.
  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. De som van de gewichten moet gelijk zijn aan 1 en de gewichten mogen niet negatief zijn. Ook moet een drempelwaarde T bepaald worden.
  2. (Gebruiksfase)
     (a) 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 $\mathbf{x}$ lijkt.
     (b) De klasse telt deze signalen voor zijn klasse op, rekening houdend met de gewichten.
     (c) $\mathbf{z}$ wordt toegewezen aan de klasse met het sterkste signaal.
- We kunnen elk element $\mathbf{x}$ uit $\mathcal{L}$ beschouwen als een invoerneuron dat, gegeven een item $\mathbf{z}$, een signaal uitstuurt hoe goed $\mathbf{z}$ lijkt op $\mathbf{x}$.
- Beide klassen hebben een neuron dat de outputs van alle invoerneuronen voor die klasse verzamelt. Ze kunnen beschreven worden door de reactiefunctie $f(r) = r$.

- De gewichten zijn $\alpha_1, ..., \alpha_n$, zodat de totale uitvoer van het uitvoerneuron kan beschreven worden als:
$$a(\mathbf{z}) = \sum_{i=1}^{n} \alpha_i y_i g(\mathbf{x}_i, \mathbf{z})$$
Hierbij is $y_i = 1$ als $\mathbf{x} \in \mathcal{L}^+$ en $y_i = -1$ als $\mathbf{x} \in \mathcal{L}^-$
- $\mathbf{z}$ wordt bij $\mathcal{L}^+$ geklasseerd als $a(\mathbf{z}) \geq T$, en anders bij $\mathcal{L}^-$ als $a(\mathbf{z}) < T$
- De vectoren $\mathbf{x}_i$ waarvoor $\alpha_i \neq 0$ worden de steunvectoren genoemd.
- Hoe worden de gewichten α_i en de drempelwaarde T bepaald?
  - Twee grootheden belangrijk:
$$m^+ = \min_{y_1=1} a(\mathbf{x}_i) \qquad m^- = \min_{y_1=-1} a(\mathbf{x}_i)$$
    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 $\frac{m^+ + m^-}{2}$
  - Om regel (1) en (2) optimaal te houden, kan de energiefunctie gebruikt worden:
$$\mathcal{F}(\alpha_1, ..., \alpha_n) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j g(\mathbf{x}_i, \mathbf{x}_j)$$
    - Beschouw de α die gehecht zijn aan een vector in $\mathcal{L}$.

10.2 Niet-lineaire classificatie

- Hier: kwadratisch oplosbare problemen
- Een verzameling $\mathcal{L} = \mathcal{L}^+ \cup \mathcal{L}^-$ is 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 alle $\mathbf{z}$ in $\mathcal{L}^-$.
- Kan een stemmachine diet probleem behandelen?
  - Stel $g(\mathbf{x}, \mathbf{z}) = -||\mathbf{x} - \mathbf{z}||^2$

$$\begin{aligned}
a(\mathbf{z}) &= \sum_{i=1}^{n} \alpha_i y_i g(\mathbf{x}_i, \mathbf{z}) \\
&= \sum_{i=1}^{n} \alpha_i y_i (-||\mathbf{x}_i||^2 - ||\mathbf{z}||^2 + 2\mathbf{x}_i \cdot \mathbf{z})) \\
&= -\sum_{i=1}^{n} \alpha_i y_i ||\mathbf{x}_i||^2 - \sum_{i=1}^{n} \alpha_i y_i ||\mathbf{z}||^2 + 2 \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{z} \\
&= -\sum_{i=1}^{n} \alpha_i y_i ||\mathbf{x}_i||^2 - ||z||^2 \left(\sum_{y_i=1}^{n} \alpha_i - \sum_{y_i=-1}^{n} \alpha_i \right) + 2 \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{z} \\
&= -\sum_{i=1}^{n} \alpha_i y_i ||\mathbf{x}_i||^2 + 2 \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \cdot \mathbf{z}
\end{aligned}$$

- Enkel de laatste term hangt af van $\mathbf{z}$ en is lineair:
  - De gelijkaardigheidsfunctie $g(\mathbf{x}, \mathbf{z}) = -||\mathbf{x} - \mathbf{z}||^2$ kan enkel lineair scheidbare problemen oplossen.

- Bij een gelijkaardigheidsfunctie hoort er best een **kernfunctie**. Voorbeeld voor de 2de graad:

$$\kappa_2(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^2$$

  Deze functie kan alle kwadratische problemen oplossen.

- Voor de $k-$de graad:

$$\kappa_2(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^k$$

  of

$$\kappa_2(\mathbf{x}, \mathbf{z}) = e^{-||\mathbf{x}-\mathbf{z}||^2}$$

Hoofdstuk 11

Associatieve Geheugens

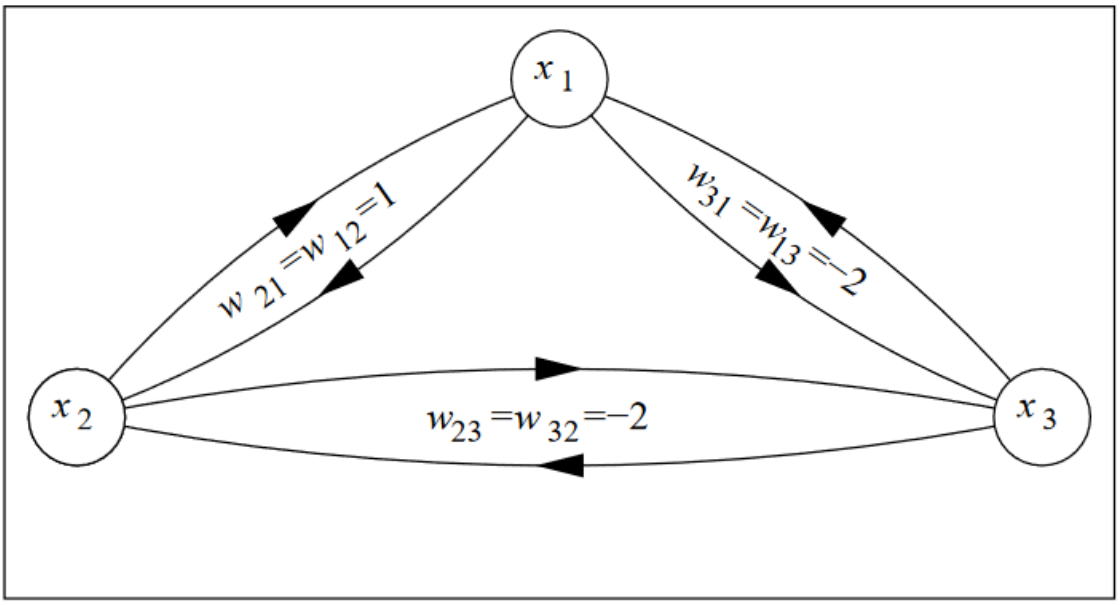
- Klassiek computergeheugen:
  - Geef een adres op en krijg de inhoud op die geheugenplaats.
  - ! Dit adres moet correct zijn, of verkeerde inhoud wordt teruggegeven.
- Associatief geheugen:
  - Geef een gedeelte van de inhoud op en krijg daarmee de geassocieerde volledige inhoud terug.
  - Speelt bij de mens een belangrijkere rol.

11.1 Hopfieldnetten

- Een model voor een biologisch associatief geheugen.
- Eenvoudigste vorm = binaire patronen opslaan.
- De neuronen in een Hopfieldnet zijn dan ook TLU's met drempelwaarde nul.
- Alle neuronen zijn met alle andere neuronen verbonden, behalve zichzelf en de gewichten zijn symmetrisch:

$$w_{ij} = w_{ji} \text{ en } w_{ii} = 0$$

- ! Geen aparte in- en uitvoerneuronen.
- Veronderstelling van bitpatroon dat evenveel bits heeft als neuronen in het netwerk.
- Elk neuron wordt in een toestand gebracht waarin het als uitvoer de overeenkomstige bitwaarde heeft.
- Sommige neuronen worden aangeduid voor herberekening, tot dat een stabiele toestand bereikt wordt (de geassocieerde eindtoestand).
- Voorbeeld:



Figuur 11.1: Hopfieldnet met drie neuronen.

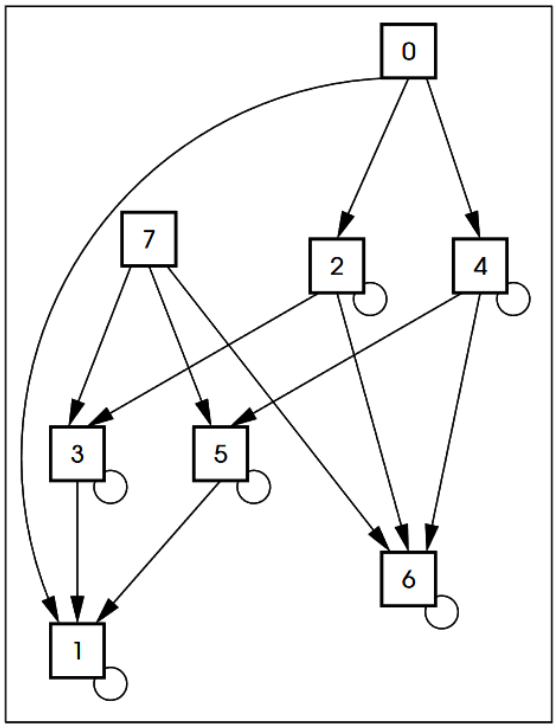
- Drie neuronen x_1, x_2 en x_3 met gewichtenmatrix

$$W = \begin{pmatrix} 0 & 1 & -2 \\ 1 & 0 & -2 \\ -2 & -2 & 0 \end{pmatrix}$$

- De toestand wordt beschreven door de uitvoer van de drie neuronen als een binair getal te beschouwen.
  - Toestand $5 = (+ - +)$ waarbij x_1 en x_3 als uitvoer 1 hebben en x_2 uitvoer -1.
  - Wat als we in deze toestand x_2 aanduiden voor herberekening.
  - De invoer voor x_2 is $1 \cdot 1 - 2 \cdot 1 = -1$, zodat x_2 dezelfde uitvoer behoudt, en het net in toestand 5 blijft.
- Er kan een transitietabel opgemaakt worden:

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

- Toestand 6 en 1 zijn stabiel. Elke andere toestand zal ooit in toestand 1 of 6 komen door willekeurig neuronen aan te duiden.
- Toestand 6 en 1 komen overeen met de twee patronen die het netwerk heeft opgeslagen. Bij een begintoestand (willekeurig bitpatroon van 3 bits) zal het netwerk evulueren naar één van deze twee toestanden, en wel diegene dat er het best op lijkt.



Figuur 11.2: Overgangsdiagram van het netwerk.

11.1.1 Algemene Hopfieldnetten

- Vorig voorbeeld was heel klein, stel nu een groter netwerk, met bijvoorbeeld duizend neuronen ($= 2^{1000}$ toestanden).
- De **energie** van een netwerk met n neuronen wordt gedefinieerd als:

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\omega_{ij}u_i u_j$$

  - Het product $u_i u_j$ is altijd gelijk aan ± 1.
  - Als w_{ij} positief is, dan probeert de verbinding ervoor te zorgen dat u_i en u_j hetzelfde teken krijgen. Hoe groter de waarde w_{ij}, hoe meer het resultaat als 'vreemd' beschouwd wordt.
  - Als w_{ij} negatief is, dan is alles omgekeerd.
- De energie van een net is een maat voor hoe vreemd de toestand is van het net.
- Duidt een neuron x_k aan voor herberekening. **Twee** mogelijkheden:
  1. De waarde van het neuron blijft hetzelfde zodat de toestand en energie behouden blijft.
  2. De waarde van het neuron verandert. Wat is nu de invloed hiervan op de energie van het netwerk?

$$\begin{aligned} E &= -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n}\omega_{ij}u_i u_j - \frac{1}{2}\sum_{j}\omega_{kj}u_k u_j - \frac{1}{2}\sum_{i}\omega_{ik}u_i u_k \\ &= -\frac{1}{2}\sum_{i\neq k}^{n}\sum_{j\neq k}^{n}\omega_{ij}u_i u_j - u_k\sum_{i\neq k}w_{ik}u_i \end{aligned}$$

3. De verandering van energieniveau is dan:

11.2 Leren bij Hopfieldnetten

- Via de regel van Hebb.
- Als we de kans willen vergroten dat een bepaald patroon $U = (u_1, ..., u_n)$ als uitvoer voorkomt, dan moet de energie van dit patroon vermindert worden.
  - Dit komt neer met het veranderen van de gewichten in de richting van de uitvoeren:

$$w_{ij} \to w_{ij} + u_i u_j \text{ voor } i \neq j$$

  - De wijziging in energie:

$$\begin{aligned} E_{nieuw}(U) &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\omega_{ij} + \Delta\omega_{ij})u_i u_j \\ &= E_{oud}(U) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta\omega_{ij}u_i u_j \\ &= E_{oud}(U) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}1 \\ &= E_{oud}(U) - \frac{n^2 - n}{2} \end{aligned}$$

- Voor een ander patroon $V = (v_1, ..., v_n)$ die k bits verschilt van U daalt de energie ook:
  -

$$\begin{aligned} E_{nieuw}(V) &= -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}(\omega_{ij} + \Delta\omega_{ij})v_i v_j \\ &= E_{oud}(V) - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta\omega_{ij}v_i v_j \end{aligned}$$

en

$$v_i v_j w_{ij} = \begin{cases} 1 & \text{als} \quad v_i = u_i \quad \text{en} \quad v_j = u_j & (n-k)(n-k-1) \text{ gevallen} \\ 1 & \text{als} \quad v_i = -u_i \quad \text{en} \quad v_j = -u_j & k(k-1) \text{ gevallen} \\ -1 & \text{als} \quad v_i = -u_i \quad \text{en} \quad v_j = u_j & k(n-k) \text{ gevallen} \\ -1 & \text{als} \quad v_i = u_i \quad \text{en} \quad v_j = -u_j & (n-k)k \text{ gevallen} \end{cases}$$

dus

$$E_{nieuw}(V) = E_{oud}(V) - \frac{(2k-n)^2 - n}{2}$$

11.3 Associatieve groepering

- Het komt bijna nooit voor dat dezelfde invoer herhaalde malen voorkomt in het leerproces.
- De stabiele eindtoestanden zijn een soort gemiddelde van groepen invoertoestanden die bij elkaar horen.
- Hopfieldnetten kunnen dan gebruikt worden om te classificeren zonder supervisie.
- Het algoritme:
  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.

11.4 Patroonvervollediging

Hoofdstuk 12

Kennisrepresentatie in neurale netten

Hoofdstuk 13

Een geïntegreerd netwerk