Transparanten bij het vak Inleiding Adaptieve Systemen: Neurale Netwerken. M. Wiering

---

Neurale netwerken

Leerdoelen:

- Weten wanneer neurale netwerken toepasbaar zijn
- De Delta-leerregel kennen
- Kunnen uitrekenen wat gewichtenveranderingen in een lineair netwerk zijn gegeven een leervoorbeeld
- Weten wat multi-layer feedforward neurale netwerken zijn
- De backpropagation leerregel kunnen opschrijven en uitleggen
- Weten wat recurrente neurale netwerken zijn

Neurale netwerken

Kunstmatige neurale netwerken (KNN) bestaan uit een verzameling neuronen (rekeneenheden) welke verbonden zijn in netwerken (McCulloch en Pitts, 1943).

Ze bezitten nuttige computationele eigenschappen (bv. ze kunnen alle continue functies benaderen met 1 hidden laag en alle functies met 2 hidden lagen)

Ze bieden ons de mogelijkheid om te leren hoe de hersenen werken.

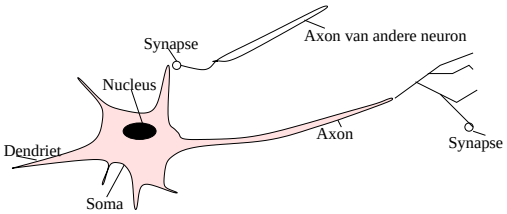
Een **neuron** of zenuwcel is de fundamentele bouwsteen van het brein.

Een neuron bestaat uit een cellichaam : de **soma**

Uit het cellichaam vertakken **dendrieten** en een **axon**. Een axon verbindt zich met dendrieten van andere neuronen in **synapses**, de verbindingspunten.

Een menselijk neuraal netwerk

Chemische transmitter vloeistoffen worden vrijgegeven in de synapses en stromen de dendrieten binnen.



Dit heeft als effect dat de **actie-potentiaal** in de soma vemindert of vermeerdert.

Wanneer de actie-potentiaal een bepaalde drempelwaarde overschrijdt, wordt een electrische pulse doorgegeven naar de axon (de neuron **vuurt**).

Synapses welke de actie potentiaal laten toenemen heten **excitatory**.

Synapses welke de actie potentiaal laten afnemen heten **inhibitory**.

Kunstmatige neurale netwerken

Een neuraal netwerk bestaat uit een aantal **neuronen** (units) en verbindingen tussen de neuronen.

Elke verbinding heeft een **gewicht** eraan geassocieerd (een getal).

Het leren gebeurt gewoonlijk door de gewichten bij te stellen.

Elke neuron heeft een aantal ingaande verbindingen van andere neuronen, een aantal uitgaande verbindingen en een **activatie nivo**.

Het idee is dat elke neuron een lokale berekening uitvoert, gebruikmakende van zijn inkomende verbindingen.

Om een neuraal netwerk te bouwen moet men de topologie van het netwerk instellen (hoe zijn neuronen verbonden).

Gewichten worden meestal willekeurig geinitialiseerd.

Vergelijking KNN en Biologische NN

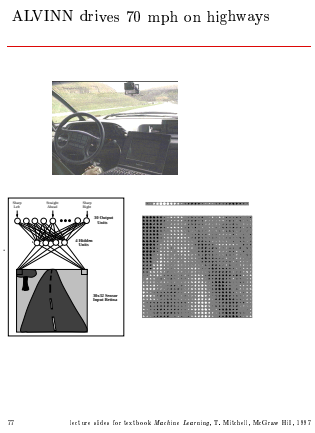
Beschouw mensen:

- Neuron switch tijd: .001 seconde
- Aantal neuronen: 10^{10-11}
- Connecties per neuron: 10^{4-5}

- Visuele herkennings tijd : 0.1 seconde
- 100 inferentie stappen lijkt niet genoeg → Veel parallelle computatie

Eigenschappen van neurale netwerken (KNN)

- Veel neuron-achtige drempel switch units
- Veel gewogen connecties tussen units
- In hoge mate parallel, gedistribueerd proces
- Nadruk op automatisch leren van gewichten



Wanneer kunnen Neurale Netwerken gebruikt worden?

- Input is hoog-dimensionaal discreet of continu (b.v. ruwe sensor input)
- Output is discreet of continu
- Output is een vector van waarden
- Mogelijk ruisige data
- Vorm van doelfunctie is onbekend
- Menselijke leesbaarheid van gevonden oplossing is onbelangrijk

Voorbeelden:

- Spraak herkenning
- Beeld classificatie (gezichts herkenning)
- Financiële voorspelling
- Patroon herkenning (postcodes)

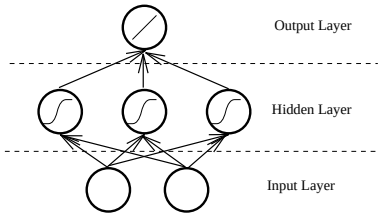
Voorbeeld: autorijden

Feedforward neurale netwerken

Vormen een gelaagde structuur. Alle verbindingen gaan van 1 laag, naar de volgende laag.

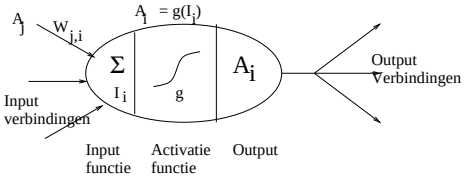
We onderscheiden de volgende lagen:

- De **Input** laag: hier worden de inputs van het netwerk naartoe gecopieerd.
- De **Hidden** laag: hier worden interne (niet-lineaire) berekeningen uitgevoerd.
- De **Output** laag: hier worden de waarden van de outputs van het netwerk berekend.



Neuron

In een netwerk ziet een individuele neuron er als volgt uit:



Leren gaat door gewichten bij te stellen aan de hand van de fout op leervoorbeelden.

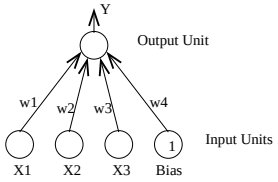
Voorbeeld: de output van een netwerk is 0.9. De gewenste output is 1.0. Verhoog de gewich-

ten die de output van het netwerk doen toenemen. Verlaag de gewichten die de output doen afnemen.

Een lineair neuraal netwerk

Het simpelste neurale netwerk is een lineair neuraal netwerk. Deze bestaat uit enkel een input en output laag.

Een lineair neuraal netwerk ziet er als volgt uit:



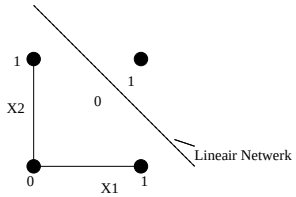
Er wordt een bias-unit gebruikt om alle lineaire functies te kunnen representeren. Deze kan als extra waarde (1) aan de inputvector meegegeven worden.

Het lineaire netwerk wordt gezien als een functiemapping van de inputs $X_1, \ldots, X_N$ naar output Y:

$$Y = \sum_i w_i X_i$$

Representeren

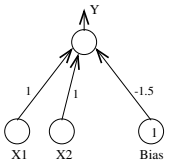
Een lineair netwerk kan bijvoorbeeld de AND functie representeren:



Het volgende netwerk (de Perceptron) doet dit (als de output > 0 dan $Y = 1$, anders $Y = 0$):

Leren

Een initieel netwerk wordt gemaakt met random gewichten (bv. tussen -0.5 en 0.5)



Het leren gaat als volgt:

- Definieer de fout als het kwadratische verschil tussen de gewenste uitkomst D en de verkregen uitkomst Y voor een voorbeeld: $X_1, \ldots, X_N \rightarrow D$:

$$E = \frac{1}{2}(D - Y)^2$$

- We willen nu de afgeleidde van de fout E naar de gewichten $w_1, \ldots, w_N$ berekenen:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial w_i} = -(D - Y)X_i$$

- Nu "updaten" we de gewichten met leersnelheid $\alpha > 0$ om de fout te verkleinen. De **Delta-leerregel** ziet er als volgt uit:

$$w_i = w_i + \alpha(D - Y)X_i$$

- We stoppen als de totale fout over alle leervoorbeelden klein genoeg is.

Voorbeeld

Gegeven leervoorbeeld $(0.5, 0.5 \rightarrow 1)$.

We maken een lineair netwerk met initiele gewichten 0.3 en 0.5 en 0.0 (voor de bias).

We kiezen een leersnelheid, bv: $\alpha = 0.5$

Nu kunnen we de gewichten aanpassen:

$Y = 0.3 * 0.5 + 0.5 * 0.5 + 0.0 * 1.0 = 0.4.$
$E = 1/2(1.0 - 0.4)^2 = 0.18$
$w_1 = 0.3 + 0.5 * 0.6 * 0.5 = 0.45$
$w_2 = 0.5 + 0.5 * 0.6 * 0.5 = 0.65$
$w_3 = 0.0 + 0.5 * 0.6 * 1.0 = 0.30$

Bij een volgende presentatie van het leervoorbeeld is de nieuwe uitkomst:
$Y' = 0.45 * 0.5 + 0.65 * 0.5 + 0.3 * 1.0 = 0.85.$

Vraag: Stel hetzelfde leervoorbeeld wordt nogmaals gepresenteerd. Bereken de nieuwe gewichten.

Batch vs Stochastic Gradient Descent

Er zijn in principe 2 methodes om met de data om te gaan:

- Batch-leren: probeert de fout in 1 keer voor alle voorbeelden in de leerverzameling te verminderen. Hiervoor wordt de totale gradient berekend en in 1 keer bijgesteld:

$$E = \frac{1}{2}\sum_p (D^p - Y^p)^2$$

Dus:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial w_i} = \sum_p -(D^p - Y^p)X_i^p$$

- Online-leren: stelt de fout na elk leervoorbeeld bij. Maakt dus stochastische stapjes in het foutlandschap (de totale fout kan verminderd of vermeerderd worden):

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial Y^p}\frac{\partial Y^p}{\partial w_i} = -(D^p - Y^p)X_i^p$$

Meestal wordt online learning gebruikt. Dit kan enkele orders van magnitude sneller convergeren (10 a 100 keer zo snel).

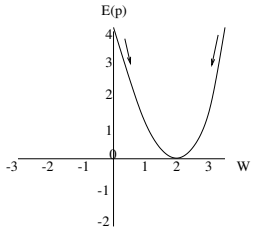
Intuitie van leerproces

Fout van leervoorbeeld heeft afgeleidde naar elk gewicht. Minimaliseer de fout door de afgeleidde (gradient) af te gaan.

Voorbeeld doelfunctie : $Y = 2X$.

Leervoorbeeld $p = (1, 2)$.

Foutlandschap voorbeeld:



Berekenen van optimale gewichten

We kunnen ook Lineaire Algebra gebruiken om de optimale gewichten voor een lineair netwerk te berekenen.

We hebben : $Y = W^T X$, voor alle voorbeelden. We zetten de voorbeelden in de matrices X en Y en doen dat als volgt:

$Y = [Y^1, Y^2, Y^3, \ldots, Y^N]$, en $X = [X^1, X^2, X^3, \ldots, X^N]$.

Nu hebben we (pseudo-inverse):

$$\begin{aligned} Y &= W^T X \qquad (1)\\ YX^T &= W^T XX^T \\ YX^T(XX^T)^{-1} &= W^T \\ W^T &= YX^T(XX^T)^{-1} \end{aligned}$$

Voorbeeld pseudo inverse

Voorbeeld: lineair netwerk met 1 input unit en 1 bias (met constante waarde 1). Data-Voorbeelden:
$(0 \to 1)$
$(1 \to 2)$
$(2 \to 3)$

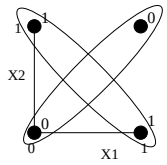
$$XX^T = \begin{vmatrix} 0 & 1 & 2 \\ 1 & 1 & 1 \end{vmatrix} \begin{vmatrix} 0 & 1 \\ 1 & 1 \\ 2 & 1 \end{vmatrix} = \begin{vmatrix} 5 & 3 \\ 3 & 3 \end{vmatrix}$$

$$(XX^T)^{-1} = \begin{vmatrix} \frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{5}{6} \end{vmatrix}$$

$$X^T(XX^T)^{-1} = \begin{vmatrix} -\frac{1}{2} & \frac{5}{6} \\ 0 & \frac{1}{3} \\ \frac{1}{2} & -\frac{1}{6} \end{vmatrix}$$

$$YX^T(XX^T)^{-1} = \begin{vmatrix} 1 & 2 & 3 \end{vmatrix} \begin{vmatrix} -\frac{1}{2} & \frac{5}{6} \\ 0 & \frac{1}{3} \\ \frac{1}{2} & -\frac{1}{6} \end{vmatrix} = \begin{vmatrix} 1 & 1 \end{vmatrix}$$

Beperkingen van lineaire netwerken



Een lineair netwerk kan de X-OR functie niet representeren. De voorbeelden van de X-OR functie zijn niet-lineair scheidbaar.

Na het verschijnen van het book Perceptrons van Minsky en Papert (1969) waarin deze problemen aangetoond werden, was het aanvankelijke enthousiasme voor neurale netwerken verdwenen.

Een kleine groep onderzoekers ging wel door. Dit leidde tot een aantal verschillende neurale netwerken.

In 1986 werden neurale netwerken weer populair na het uitvinden van het **backpropagation** algoritme, waarmee door gebruik van de kettingregel ook niet-lineaire (multi-layer) feedforward neurale netwerken geleerd konden worden.

Representatie in multi-layer feedforward neurale netwerken

We representeren het netwerk in een gerichte graaf. De optimale representatie kan een willekeurig kleine fout hebben voor een bepaalde doel functie.

Gewoonlijk wordt de topologie van het netwerk van te voren gekozen.

Hierdoor onstaat er echter een representatie fout (zelfs de optimale gewichten in een gekozen representatie hebben een bepaalde fout)

Ook lukt het vaak niet om de optimale gewichten te vinden (leerfout) door de lokale minima.

We onderscheiden 2 stappen: voorwaartste propagatie (gebruik) en terugwaartse propagatie (leren).

Voorwaartse propagatie in multi-layer feedforward neurale netwerken

- Clamp input vector $\vec{a}$
- Bereken gesommeerde input hidden units:

$$i_i = \sum_j w_{ji} a_j + b_i$$

- Bereken activatie hidden units (andere $\vec{a}$):

$$a_i = F_i(i_i) = \frac{1}{1 + e^{-i_i}}$$

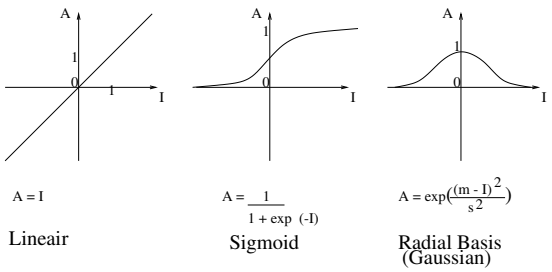
- Bereken activatie output units s:

$$s_i = F_i(\sum_j w_{ji} a_j + b_i)$$

Activatie Functies

Er kunnen meerdere activatie functies gebruikt worden.

Verschillende activatie functies zijn nuttig voor representeren bepaalde functie (voorkennis)



De hidden laag gebruikt meestal sigmoid functies of Radial Basis functies (meer lokaal)

De output laag gebruikt meestal een lineaire activatie functie (zodat alle functies gerepresenteerd kunnen worden)

Backpropagation

Minimaliseer error functie:

$$E = \frac{1}{2} \sum_i (d_i - s_i)^2$$

Door gewichten aan te passen m.b.v. gradient descent:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial i_i} \frac{\partial i_i}{\partial w_{ji}}$$

- Leerregel met leersnelheid α :

$$\Delta w_{ji} = -\alpha \frac{\partial E}{\partial w_{ji}} = \alpha \delta_i a_j$$

- Output unit:

$$\delta_i = -\frac{\partial E}{\partial i_i} = (d_i - s_i) F_i'(i_i)$$

- Hidden unit:

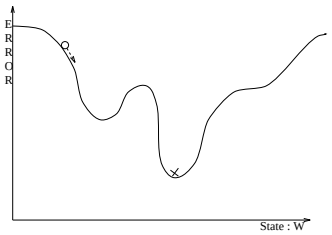
$$\delta_i = F_i'(i_i) \sum_{o \in Outputs} \delta_o w_{io}$$

Hier is $F_i'(i_i) = (1 - a_i)a_i$, als F de sigmoid functie is.

en $F_i'(i_i) = 1$, als F de lineaire functie is.

Leren als zoeken

Gradient descent op het foutlandschap werkt als volgt:



Problemen:

- **Lokale minima**. Als het netwerk in een lokaal minimum komt, kan het niet meer verbeterd worden met gradient descent.
- **Lange even vlaktes.** Als het foutlandschap ergens heel vlak is, gaat het leren erg langzaam (de gradient is zeer klein).
- Het leren van een optimaal netwerk is een NP-moeilijk probleem.

Meer over backpropagation

- Gradient descent over gehele netwerk gewichten vector
- Makkelijk generaliseerbaar naar willekeurige gerichte grafen.
- Zal lokaal minimum vinden en niet noodzakelijk globaal minimum. Kan met meerdere restarts toch goed werken.
- Gebruikt soms een momentum term:

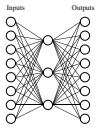
$$\Delta w_{ij}(t) = \gamma \delta_j x_i + \mu \Delta w_{ij}(t-1)$$

- Minimaliseert fout over alle trainings voorbeelden, zal het goed generaliseren naar opvolgende voorbeelden?
  - Pas op met teveel hidden units → overfitting
  - Werkt goed met genoeg voorbeelden: Vuistregel: aantal leervoorbeelden is veelvoud van aantal gewichten.
- Leren kan duizenden iteraties duren → traag!
- Gebruik van geleerd netwerk gaat snel.

Representatie van hidden units

Learning Hidden Layer Representations

A network:



Learned hidden layer representation:

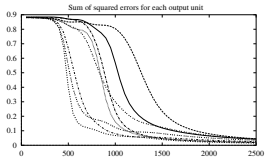
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

94 lecture slides for textbook *Machine Learning*, T. Mitchell, McGraw Hill, 1997

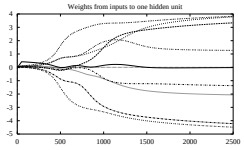
Evolutie van leerproces

Evolutie van leerproces (2)

Training



Training



Recurrente neurale netwerken

Feedforward neurale netwerken worden meest gebruikt: voor patroon herkenning zijn ze uitstekend

Recurrente neurale netwerken zijn geschikt voor problemen waarin tijdspredictie (bv. Spraakherkenning) een rol speelt.

Recurrente netwerken kunnen vorige inputs mee laten tellen in hun predictie van de huidige toestand van het systeem.

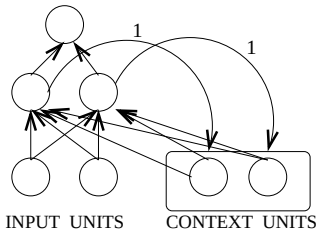
Recurrente netwerken kunnen ook gebruikt worden voor het infereren van een heel patroon op basis van een deelpatroon (pattern completion)

We onderscheiden de volgende recurrente netwerken:

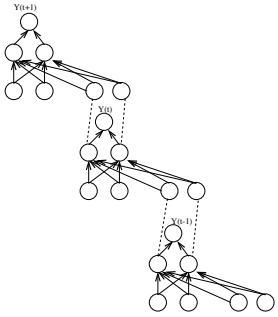
- Elman netwerken
- Jordan netwerken
- Time delay neurale netwerken (TDNN)
- Hopfield netwerken (Boltzmann machines)

Elman netwerken

Elman netwerken koppelen activatie van hidden units terug naar inputs: goed voor predictie waarin tijd belangrijke rol speelt.



Leeralgoritme: Recurrent backpropagation through time:

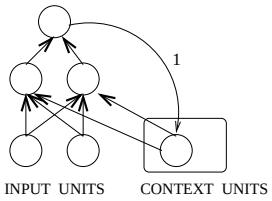


Jordan netwerken

Jordan netwerken koppelen activatie van output units terug naar inputs: goed voor predictie waarin tijd belangrijke rol speelt en sequentie van beslissingen een rol speelt.

Jordan en Elman netwerken werken ongeveer even goed.

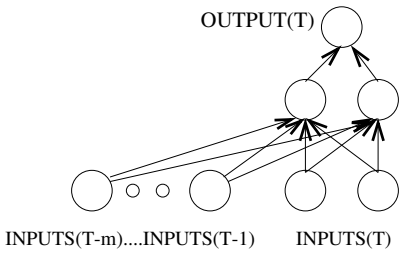
Ze hebben grote problemen als de gradient through time een erg zwak signaal wordt →
gewichten worden erg langzaam bijgesteld.



NB: Leren vaak veel trager dan feedforward netwerken. Alternatieve leermethode: Evolutionary computation.

Time Delay Neurale Networken (TDNN)

TDNN gebruiken inputs van voorgaande tijdstappen voor huidige predictie



Hebben problemen met Markov order (m) :

- Hoeveel voorgaande inputs moeten meegegeven worden?
- Kan inputs die langer geleden gezien zijn nooit mee laten tellen in beslissing.
- Veroorzaakt soms erg groot netwerk

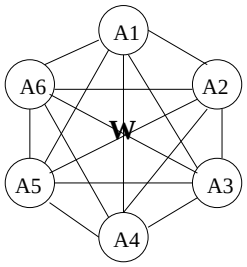
Hebben geen problemen met afnemende gradient (b.v. 100000001 $\rightarrow$ 1 en 000000001 $\rightarrow$ 0).

Hopfield Netwerk

Autoassociative Networks (Hopfield Netwerk, Boltzmann machine): soort geheugen voor opslaan patronen: goed voor pattern completion

Leerregels versterken verbindingen tussen inputs die gelijk aan staan. Als deelpatroon aangeboden wordt, zullen inputs die vaak gelijk met andere inputs voorkomen ook aan komen te staan

De nieuw geactiveerde inputs kunnen weer andere inputs activeren



Symmetrische gewichten / Asymmetrische gewichten. Lijken op Bayesiaanse geloofs netwerken.

Voor- en nadelen van Neurale netwerken

Nadelen:

- Belanden vaak in lokale minima.
- Geen directe manier om om te gaan met missende waarden
- Soms erg traag leerproces
- Soms vergeet het netwerk geleerde kennis als het getraind wordt op nieuwe kennis (leer-interferentie)
- Het kan veel experimenteertijd kosten om een goede topologie en leerparameters te vinden.
- Het is niet zo makkelijk om a-priori kennis in een netwerk te zetten
- Leren optimaal neuraal netwerk is NP-moeilijk probleem

Voordelen:

- Kan alle functies exact representeren
- Kan goed met ruis omgaan
- Kan goed met redundantie omgaan
- Kan goed met hoog dimensionale inputruimtes omgaan
- Kan direct continue functies benaderen
- Is robuust tegen wegvallen neuronen $\rightarrow$ graceful degradation