UNIVERSITY OF APPLIED SCIENCES UTRECHT

Delta learning & Backpropagation

TCTI-VKAAI-17: Applied Artificial Intelligence

Huib Aldewereld

Leerdoelen

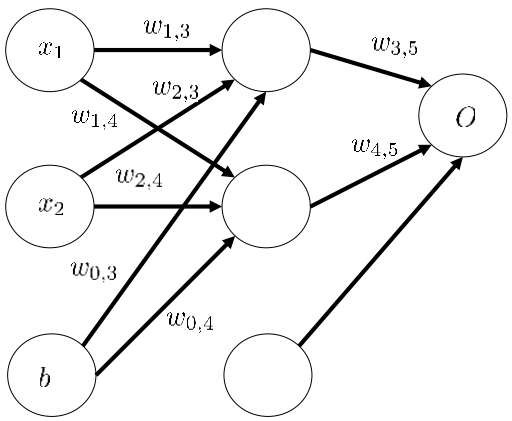
- Na deze les kan de student:
  - Leeralgoritmen voor neurale netwerken uitleggen en toepassen.
  - Weloverwogen keuzes maken met betrekking tot netwerktopologie, initialisatie, gewichten en leeralgoritme in het bouwen van een neuraal netwerk voor een specifiek doel.

Inhoudsopgave

3

Herhaling

- Forward propagation



- Output op basis van gewogen som van inputs (per neuron)
- Activatie liefst gradueel, dus d.m.v. sigmoid of tanh functie

$$\sigma(z) = \frac{1}{(1+e^{-z})}$$

$$tanh(z) = 2\sigma(2z) - 1$$

4

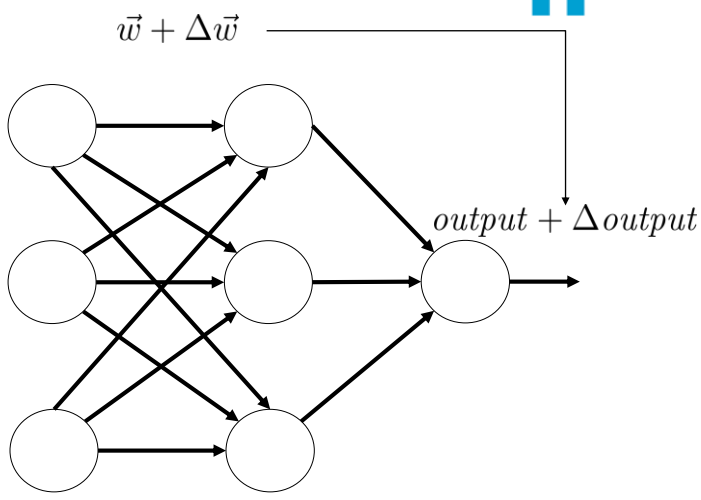
Inhoudsopgave

- Korte herhaling
- Trainen NN
  - Cost functies
  - Gradient descent
- Delta regel
- Backpropagation

5

Leren van NN

- Maak een aanpassing aan de gewichten, zodanig dat de output *dichter* bij de gewenste output komt
- De threshold functie (step-function) die we nu gebruiken maakt dit erg lastig!



6

Cost functie

- Mate om te bepalen hoe goed de classifier presteert op een bepaalde input
  - Supervised (output bekend)
  - Wat is de afwijking t.o.v. verwachte output?
- Afstand tussen werkelijke uitkomst en verwachte uitkomst

$$\vec{a}(\vec{x}_i) \longleftrightarrow \vec{y}_i$$

Activatie van output neuronen (vector) bij trainingsitem x_i

Label (verwachte output vector) bij trainingsitem i

- Meest gebruikte cost functie is Mean Squared Error (MSE)

7

Mean Squared Error

- Gekwadrateerde afstand tussen verwachte output en werkelijke output
  - Gekwadrateerd om min-tekens kwijt te raken

$$C(\vec{w}) = MSE = \frac{1}{2n} \sum_{i=1}^{n} |\vec{y}_i - \vec{a}(\vec{x}_i)|^2$$

Cost gegeven een gewichten vector

d.w.z. hoe slecht presteert het netwerk gegeven deze gewichten

Mean = gemiddelde, dus delen door *n*

Deling door 2 wegens berekenen afgeleide

Afstand tussen verwacht en berekend
Gekwadrateerd om teken kwijt te raken

$$|\vec{v}| = \sqrt{v_1^2 + \ldots + v_n^2}$$

- Doel: minimaliseren van de Cost

8

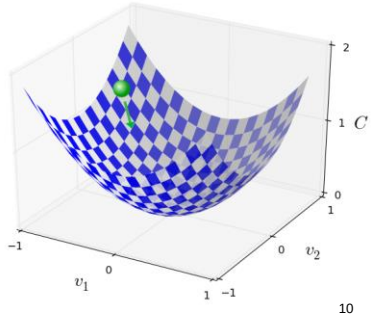
MSE vs. # correct answers

- Waarom gebruiken we MSE en tellen we niet gewoon hoeveel items goed worden geclassificeerd (zoals bij kNN)?
- Probleem met tellen correcte antwoorden is dat die functie niet gradueel (smooth) is
- Een abrupte functie (zoals tellen van goede antwoorden) maakt het lastig om te weten in welke richting je moet bewegen om een verbetering te krijgen
  - Maken van kleine veranderingen heeft geen invloed

9

Gradient Descend

- Berekenen wanneer $C(\vec{w})$ minimaal is, is exponentieel omdat $\vec{w}$ erg groot kan worden
  - Voor praktische netwerken zijn duizenden gewichten niet ongewoon
- Gradient Descend als oplossing
  - Optimalisatie techniek die (locale) minima van een functie benadert door stapjes te nemen in de richting van de grootste (negatieve) helling
  - Vergelijk met een bal in een vlak



10

Gradient Descend, vervolg

- Probleem: minimum niet te berekenen
  - Functie is complex, multi-dimensioneel, en niet uniform (grillig)
- Benader het met het GD algoritme:
  - Begin op een random plek; en
  - Zet telkens een stapje in een richting waardoor je Cost kleiner wordt
- Ofwel vindt $\Delta\vec{w}$ zodanig dat $-\Delta C(\vec{w})$
- Blijf dat doen totdat alle $\Delta\vec{w}$ een $+\Delta C(\vec{w})$ oplevert
  - Geen stapjes meer mogelijk om de Cost te laten dalen

11

Gradient Descend, vervolg

- De stapjes $\Delta\vec{w}$ kunnen worden berekend door middel van afgeleides
- Bereken de gradient (helling) op onze huidige locatie $\vec{w}$

$$\nabla C = \left(\frac{\partial C}{\partial w_1}, \ldots, \frac{\partial C}{\partial w_n}\right)^T$$

(Transpose: verandert rij-vector in kolom-vector)

Gradient vector
Collection of gradients

Verandering in Cost bij een verandering in gewicht

- Volgens gradient descent zouden we nu dus stapjes moeten zetten in de richting van $-\nabla C$

Geven afgeleides je hoofdpijn, of weet je niet meer wat het is; bekijk dan eens de eerste 3 episodes van “Essence of Calculus” van Youtuber 3Blue1Brown: https://youtu.be/WUvTyaaNkzM

12

Gradient Descent, vervolg

- Met gebruik van de gradient vector kunnen we nu de update regel specificeren

$$\vec{w}' = \vec{w} - \Delta\vec{w} = \vec{w} - \eta\nabla C$$

- Of gesplitst per gewicht

$$w_i' = w_i - \Delta w_i = w_i - \eta\frac{\partial C}{\partial w_i}$$

leerrate

Nieuwe waarde gewicht

Oude waarde gewicht

Benodigde verandering

Verandering van C t.o.v. veranderingen in w_i

Leerrate = learning rate
Mate van verandering (grootte van de stap).
Te groot → overshoot
Te klein → traag

13

Inhoudsopgave

- Korte herhaling
- Trainen NN
  - Cost functies
  - Gradient descent
- Delta regel
- Backpropagation

14

Delta regel

- Cost functie hiervoor gebruikt alle trainingsinputs
  - Veel berekeningen voor slechts 1 stap
  - Mogelijk om een stap te zetten na elk trainingsvoorbeeld
    - (algoritme wordt sneller, maar ook iets minder precies)

$$C(\vec{w}) = \tfrac{1}{2}|\vec{y} - \vec{a}(\vec{x})|^2$$

- We kunnen nu de partiele afgeleide berekenen per gewicht

$$\Delta w'_{i,j} = \eta \frac{\partial C}{\partial w_{i,j}} = \eta \frac{\partial(\frac{1}{2}|\vec{y} - \vec{a}(\vec{x})|^2)}{\partial w_{i,j}} = -\eta\, a_i\, g'(in_j)(y_k - a_j)$$

15

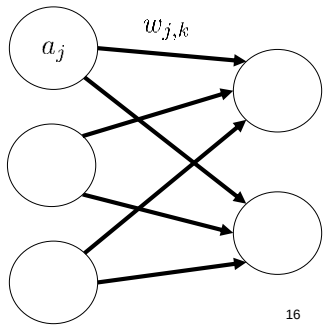
Delta regel, vervolg

- In enkellaags netwerk heeft elke input direct invloed op de output (= perceptron)
- Invloed van elk gewicht is dus ‘makkelijk’ te achterhalen

$$w'_{j,k} = w_{j,k} - \Delta w_{j,k} = w_{j,k} + \eta\, a_j\, g'(in_k)(y_k - a_k)$$

($-\eta\, a_j\, g'(in_k)(y_k - a_k)$)

- a_j is de activatiewaarde van neuron j



16

Delta regel, vervolg

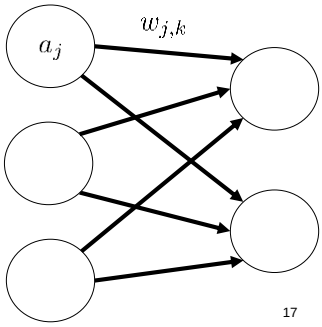
- In enkellaags netwerk heeft elke input direct invloed op de output (= perceptron)
- Invloed van elk gewicht is dus ‘makkelijk’ te achterhalen

$$w'_{j,k} = w_{j,k} - \Delta w_{j,k} = w_{j,k} + \eta\, a_j\, \underbrace{g'(in_k)}(y_k - a_k)$$

- g' is de afgeleide van de activatiefunctie
  Voor de sigmoid functie:

$$g' = \sigma(z)(1 - \sigma(z))$$

$(tanh'(z) = 1 - tanh^2(z))$



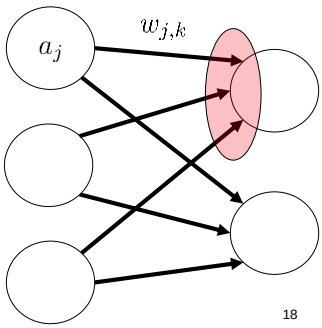
17

Delta regel, vervolg

- In enkellaags netwerk heeft elke input direct invloed op de output (= perceptron)
- Invloed van elk gewicht is dus ‘makkelijk’ te achterhalen

$$w'_{j,k} = w_{j,k} - \Delta w_{j,k} = w_{j,k} + \eta\, a_j\, \underbrace{g'(in_k)}(y_k - a_k)$$

- in_k is gewogen som van alle inputs van neuron k



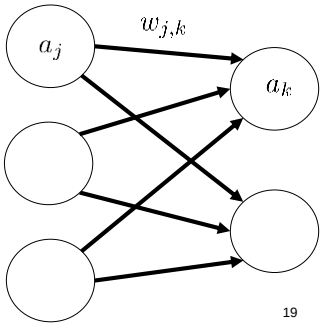
18

Delta regel, vervolg

- In enkellaags netwerk heeft elke input direct invloed op de output (= perceptron)
- Invloed van elk gewicht is dus ‘makkelijk’ te achterhalen

$$w'_{j,k} = w_{j,k} - \Delta w_{j,k} = w_{j,k} + \eta\, a_j\, g'(in_k)\underbrace{(y_k - a_k)}$$

- $(y_k - a_k)$ is afstand tussen gewenste (y_k) en geleverde uitkomst (a_k)



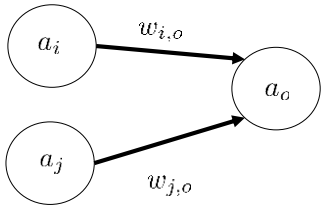
19

Delta regel, toepassing

- Start door random toewijzing van de gewichten
- Gebruik van de delta regel update de gewichten van de outputs (met respect tot de trainingsdata)
- Blijf updaten (trainingsdata voeren en vergelijken met output) totdat er geen veranderingen meer gemaakt wordt aan de gewichten
  - Kan erg lang duren, dus stop na een ‘redelijke’ tijd
- Merk op: de delta regel update alle gewichten tegelijkertijd
  - Alle $\Delta w_{j,k}$ worden berekend, en daarna worden ze pas geüpdatet

20

Voorbeeld: Training AND



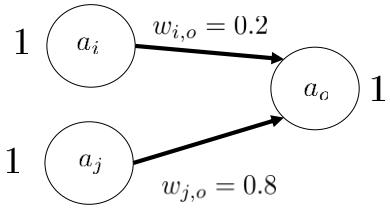
- Gegeven:
  - Modeleer een AND-functie
  - Perceptron, dus a_i, a_j inputs, a_o output
  - Hyperbool tangens functie als activatie g
  - Learnrate $\eta = 0.1$
- Gewichten 'random' geïnitieerd als

  $w_{i,o} = 0.2$ $\quad$ $w_{i,o} = 0.8$
- Training op drie voorbeelden

  $\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow 1, \begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow 0, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow 0$

21

Voorbeeld: Training AND $\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow 1$



- Delta regel:

  $w'_{j,k} = w_{j,k} + \eta\ a_j\ g'(in_k)(y_k - a_k)$

  $g'(x) = 1 - tanh(tanh(x))$

$a_o = g(a_i \times w_{i,o} + a_j \times w_{j,o}) = g(1.0 \times 0.2 + 1.0 \times 0.8) = tanh(1.0) = 0.76$

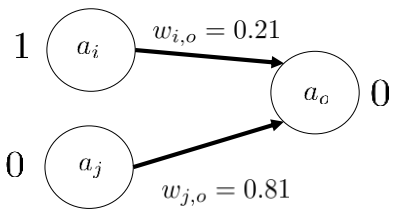
$w_{i,o} = 0.2 + 0.1 \times 1.0 \times g'(1.0 \times 0.2 + 1.0 \times 0.8)(1.0 - 0.76)$

$= 0.2 + 0.1 \times 1.0 \times (1 - tanh(tanh(1.0)))(1.0 - 0.76) = 0.21$

$w_{j,o} = 0.8 + 0.1 \times 1.0 \times g'(1.0 \times 0.2 + 1.0 \times 0.8)(1.0 - 0.76) = 0.81$

22

Voorbeeld: Training AND $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow 0$



- Delta regel:

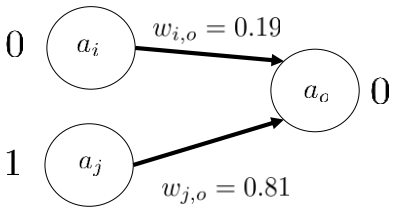
$$w'_{j,k} = w_{j,k} + \eta\ a_j\ g'(in_k)(y_k - a_k)$$
$$g'(x) = 1 - tanh(tanh(x))$$

$$a_o = g(a_i \times w_{i,o} + a_j \times w_{j,o}) = g(1.0 \times 0.21 + 0.0 \times 0.81) = tanh(0.21) = 0.21$$
$$w_{i,o} = 0.21 + 0.1 \times 1.0 \times g'(1.0 \times 0.21)(0.0 - 0.21) = 0.19$$

$$w_{j,o} = 0.81 + 0.1 \times 0.0 \times g'(0.21)(0.0 - 0.21) = 0.81$$

23

Voorbeeld: Training AND $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow 0$



- Delta regel:

$$w'_{j,k} = w_{j,k} + \eta\ a_j\ g'(in_k)(y_k - a_k)$$
$$g'(x) = 1 - tanh(tanh(x))$$

$$a_o = g(a_i \times w_{i,o} + a_j \times w_{j,o}) = g(0.0 \times 0.19 + 1.0 \times 0.81) = tanh(0.81) = 0.67$$
$$w_{i,o} = 0.19 + 0.1 \times 0.0 \times g'(1.0 \times 0.81)(0.0 - 0.67) = 0.19$$

$$w_{j,o} = 0.81 + 0.1 \times 1.0 \times g'(0.81)(0.0 - 0.67) = 0.78$$

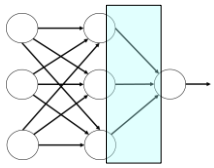
24

Inhoudsopgave

- Korte herhaling
- Trainen NN
  - Cost functies
  - Gradient descent
- Delta regel
- Backpropagation

25

Trainen MLP, output zijde



- Probleem: gewichten in hidden lagen hebben indirect invloed op output
  - Geen directe koppeling tussen input $\vec{x}$ en output $\vec{y}$
- Herschrijf updateregel om aan te geven wat een error is

$$\Delta_k = g'(in_k)(y_k - a_k)$$

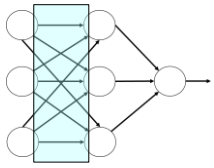
Error in de output van neuron *k*

$$w_{j,k} = w'_{j,k} + \eta a_j \Delta_k$$

Gebruik makend van die error kan je bepalen hoe de gewichten moeten worden aangepast

26

Trainen MLP, input zijde



- Betreffende de input i naar een hidden node j, zouden we de veranderingen in gewichten kunnen berekenen als we zouden weten Δ_j (de fout in output van de neuron)

$$w_{i,j} = w'_{i,j} + \eta a_i \Delta_j$$

- Neuron is proportioneel verantwoordelijk voor de fout van de volgende laag

$$\Delta_j = g'(in_j) \sum_p w_{j,p} \Delta_p$$

Fout neuron op deze laag

Proportioneel op gewichten

Fout neuron op volgende laag

27

Meerdere lagen: Backpropagation

- Mate van fout propageert terug door het netwerk:
  - Output (error) → Hidden (error) → Input (error)
- Voor een gewicht tussen neuron j en neuron p geldt

$$w_{i,j} = w'_{i,j} + \eta a_i \Delta_j$$

- Als j een hidden neuron:

$$\Delta_j = g'(in_j) \sum_p w_{j,p} \Delta_p$$

- Als j een output:

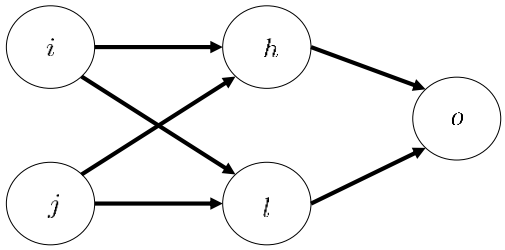
$$\Delta_j = g'(in_j)(y_j - a_j)$$

28

Voorbeeld: Training XOR

- XOR netwerk heeft 1 hidden laag
- Feitelijk een combinatie van OR + NOT AND op hidden laag, en AND op output laag
- Met volgende gewichten (en threshold activatie) levert dit een XOR:

$$\begin{aligned} w_{i,h} &= 1 \\ w_{j,h} &= 1 \\ w_{i,l} &= -1 \\ w_{j,l} &= -1 \\ w_{h,o} &= 1 \\ w_{l,o} &= 1 \end{aligned}$$



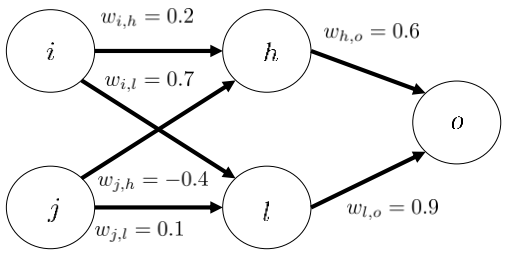
29

Voorbeeld: Training XOR

- Trainen
  - Kies ‘random’ gewichten
  - Leerrate = 0.1
  - Voorbeelden:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow 0 \qquad \begin{pmatrix} 0 \\ 0 \end{pmatrix} \rightarrow 0$$

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow 1 \qquad \begin{pmatrix} 0 \\ 1 \end{pmatrix} \rightarrow 1$$

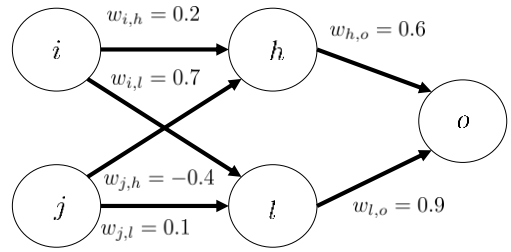


- Meerlaagsnetwerk → backpropagation

30

Voorbeeld: Training XOR $\begin{pmatrix} 1 \\ 1 \end{pmatrix} \rightarrow 0$

$$a_i = a_j = 1$$
$$a_h = g(0.2 - 0.4) = -0.2$$
$$a_l = g(0.7 + 0.1) = 0.66$$
$$a_o = g(0.6 \times -0.2 + 0.9 \times 0.66) = 0.44$$

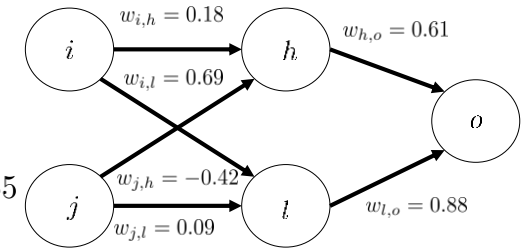


$$\Delta_o = g'(0.46) \times (0 - 0.44) = -0.26$$
$$\Delta_h = g'(-0.2) \times 0.6 \times -0.26 = -0.18$$
$$\Delta_l = g'(0.8) \times 0.9 \times -0.26 = -0.1$$

$$w_{h,o} = 0.6 + 0.1 \times -0.2 \times -0.26 = 0.61$$
$$w_{l,o} = 0.9 + 0.1 \times 0.66 \times -0.26 = 0.88$$
$$w_{i,h} = 0.2 + 0.1 \times 1.0 \times -0.18 = 0.18$$
$$w_{j,h} = -0.4 + 0.1 \times 1.0 \times -0.18 = -0.42$$
$$w_{i,l} = 0.7 + 0.1 \times 1.0 \times -0.1 = 0.69$$
$$w_{j,l} = 0.1 + 0.1 \times 1.0 \times -0.1 = 0.09$$

Voorbeeld: Training XOR $\begin{pmatrix} 1 \\ 0 \end{pmatrix} \rightarrow 1$

$$a_i = 1 \quad a_j = 0$$
$$a_h = g(0.17) = 0.17$$
$$a_l = g(0.69) = 0.60$$
$$a_o = g(0.61 \times 0.17 + 0.88 \times 0.6) = 0.55$$



$$\Delta_o = g'(0.63) \times (1 - 0.55) = 0.22$$
$$\Delta_h = g'(0.17) \times 0.61 \times 0.22 = 0.11$$
$$\Delta_l = g'(0.6) \times 0.88 \times 0.22 = 0.10$$

$$w_{h,o} = 0.61 + 0.1 \times 0.17 \times 0.22 = 0.61$$
$$w_{l,o} = 0.88 + 0.1 \times 0.6 \times 0.22 = 0.89$$
$$w_{i,h} = 0.18 + 0.1 \times 1.0 \times 0.11 = 0.18$$
$$w_{j,h} = -0.42 + 0.1 \times 0.0 \times 0.11 = -0.42$$
$$w_{i,l} = 0.69 + 0.1 \times 1.0 \times 0.1 = 0.70$$
$$w_{j,l} = 0.09 + 0.1 \times 0.0 \times 0.1 = 0.09$$