AI introductie voor testers

De basis van deep learning

TestNet werkgroep ‘Testen met AI’

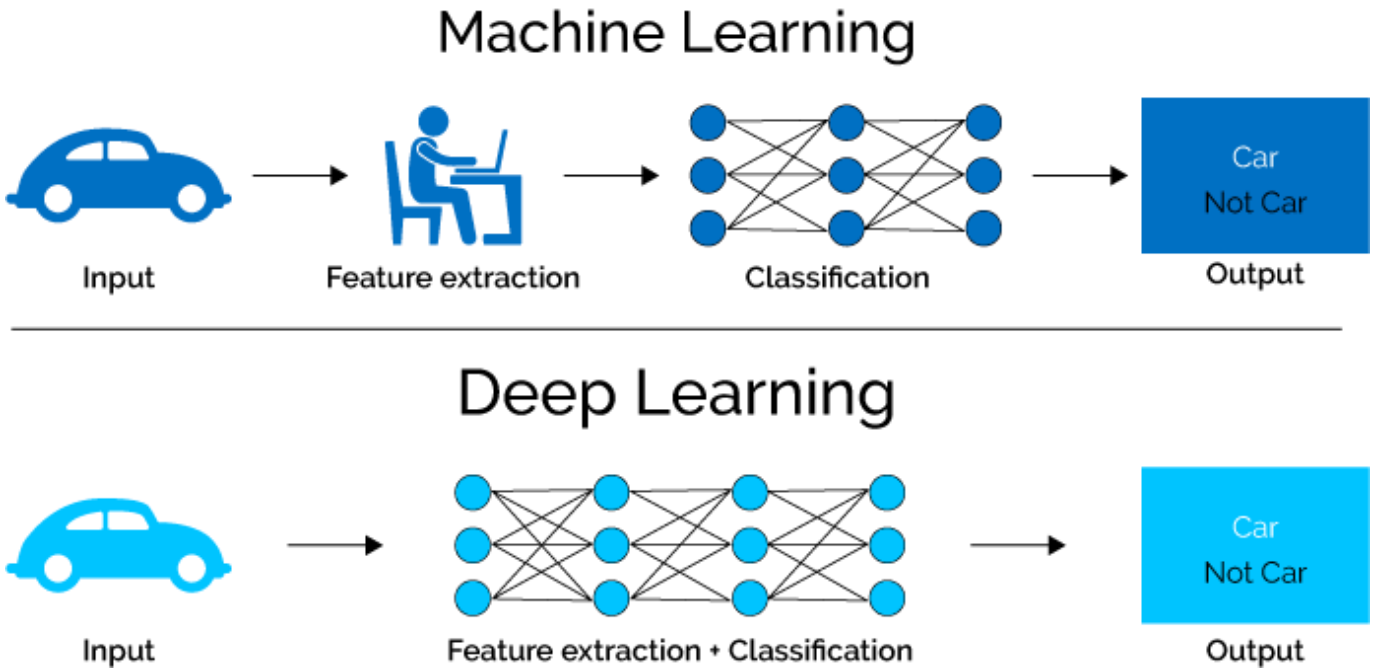
Martin van Helden
Sander Mol

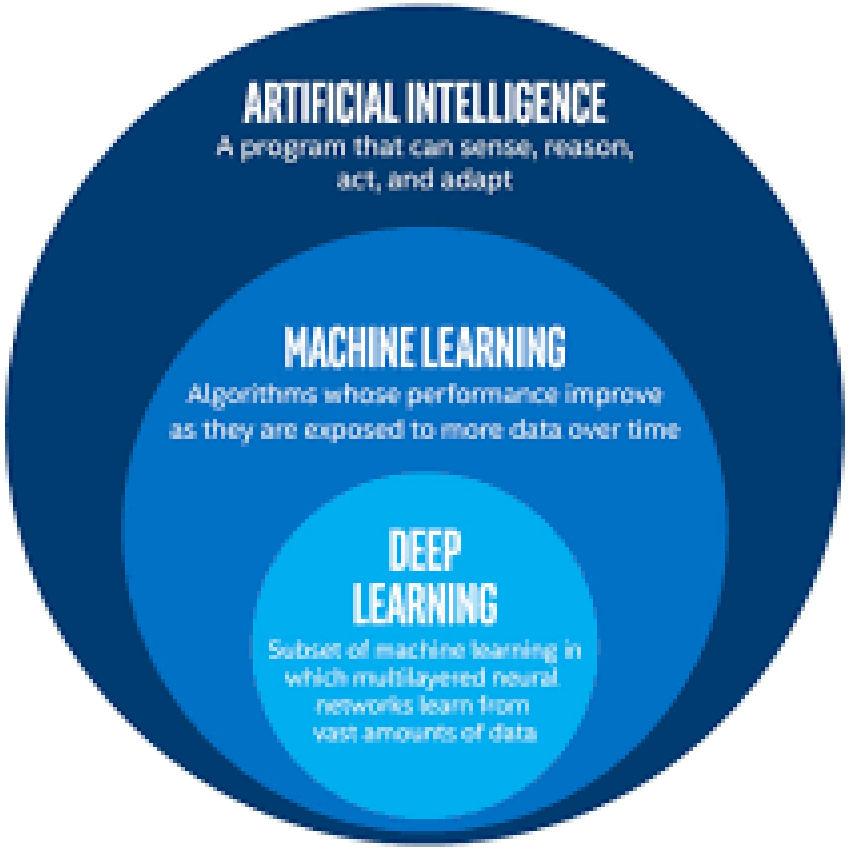
Introductie

Artificial Intelligence (AI) is anders dan traditioneel programmeren. Traditioneel moet je alle functies zelf programmeren. In AI wordt de functionaliteit geleerd door te trainen op gegevens.

Hoe programmeer je het verschil
tussen een appel en een sinaasappel?

Machine Learning en Deep learning



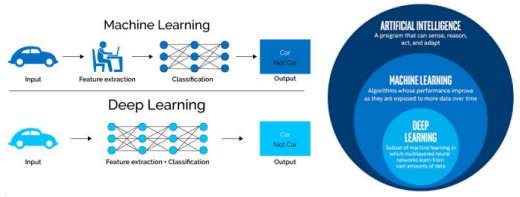


Machine Learning en Deep learning

Artificial Intelligence is het vakgebied, waarin een programma zelfstandig keuzes maakt op basis van de input die het krijgt. Computerspellen hebben bijvoorbeeld al jaren een AI, het concept bestaat zelfs al sinds de jaren ‘50.

Machine learning is de AI variant, waarbij het programma op basis van nieuwe gegevens steeds betere keuzes leert maken.

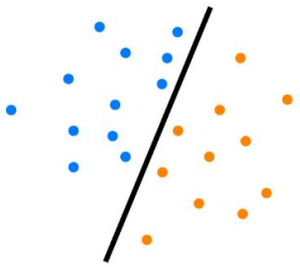
Deep learning is de sterke variant van machine learning, waarbij het programma door gebruik van meer gegevens zelfstandiger keuzes kan maken.

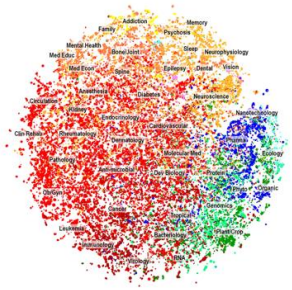


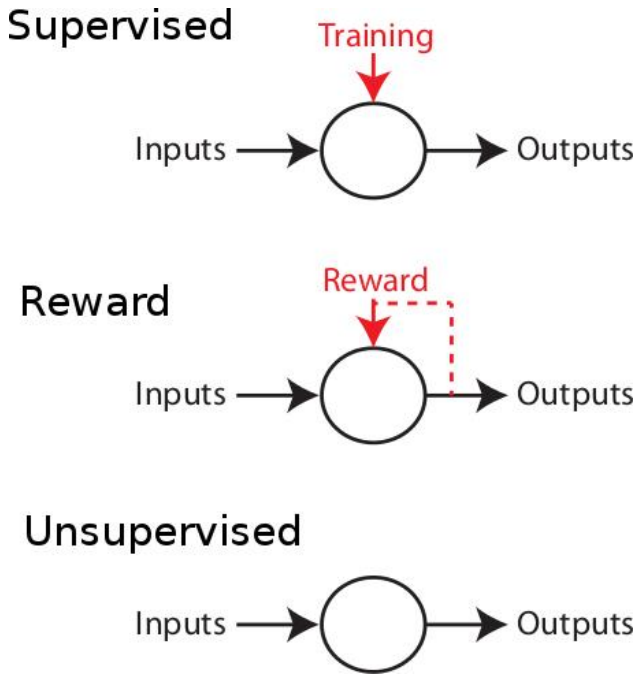
Soorten machine learning

Supervised -> leren van gelabelde gegevens
Unsupervised-> leren door te clusteren
Reinforcement -> leer van de resultaten (score)

Supervised	Unsupervised	Reinforcement
Learning known patterns	Learning unknown patterns	Generating data Learning patterns





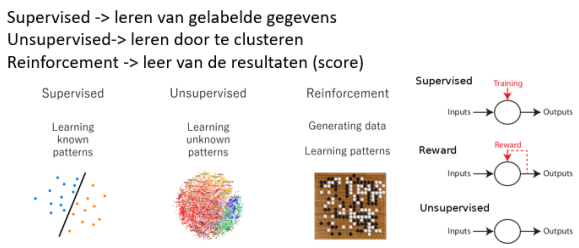


Soorten machine learning

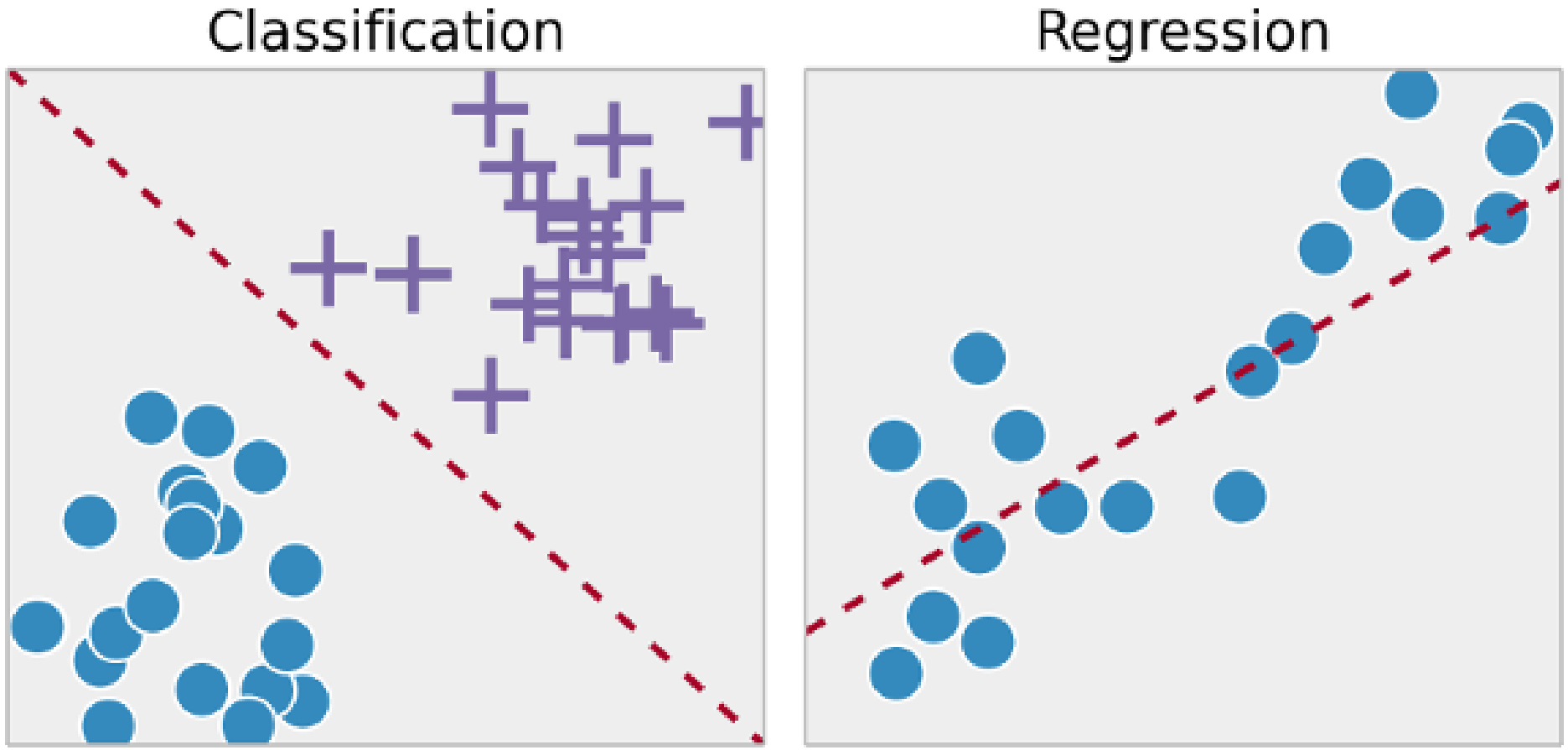
Bij supervised learning geeft de mens aan wat de verwachte uitkomst is, zoals ‘dit is een kat’. Via heel veel kattenplaatjes leert de AI de algemene eigenschappen van een kat te herkennen.

Bij unsupervised learning krijgt de AI geen verwachte resultaten mee. Door heel veel plaatjes van bijvoorbeeld katten en auto’s te beoordelen, gaat hij twee clusters van objecten herkennen, zonder dat hij het ‘kat’ of auto’ noemt.

Bij reinforcement learning wordt de AI gekoppeld aan iets dat handelingen uitvoert (zoals een toetsenbord), zodat hij zelf ervaart en leert welke keuzes succesvol zijn.



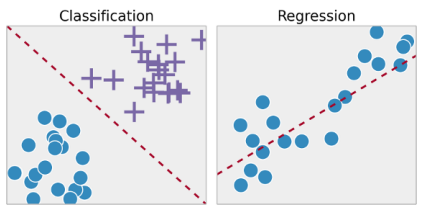
Classificatie vs Regressie



Classificatie vs Regressie

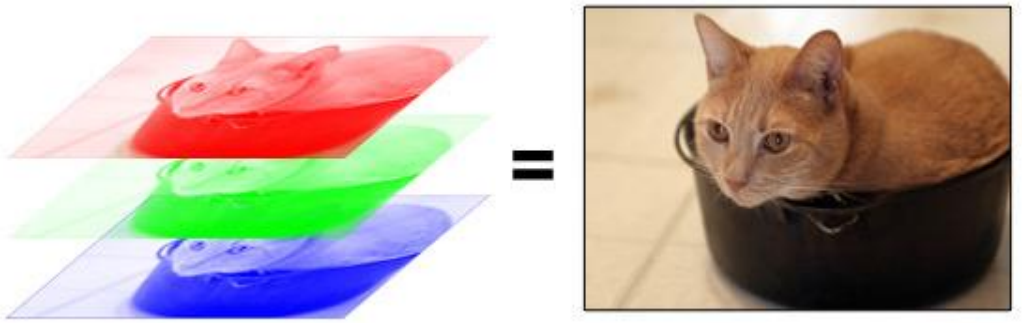
Classificatie is het opdelen van objecten in verschillende klassen, zoals de katten en auto's op de vorige sheet.

Regressie is het proberen om verschillende objecten van dezelfde soort (bijvoorbeeld veel foto's van het object 'kat') zo goed mogelijk op een eenduidige manier te voorspellen.

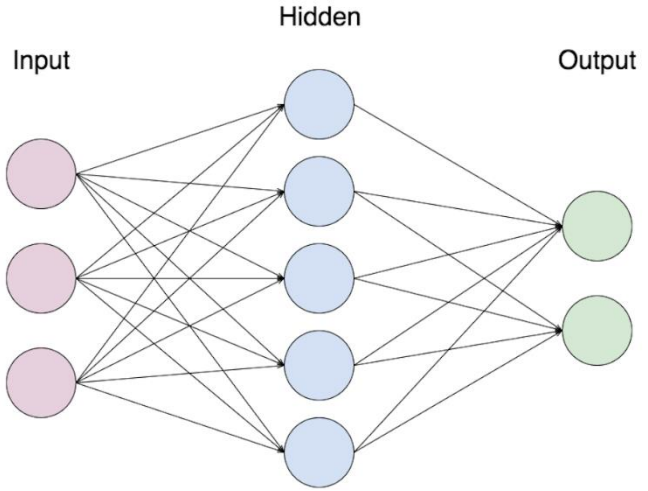


Deep Learning - voorbeeld

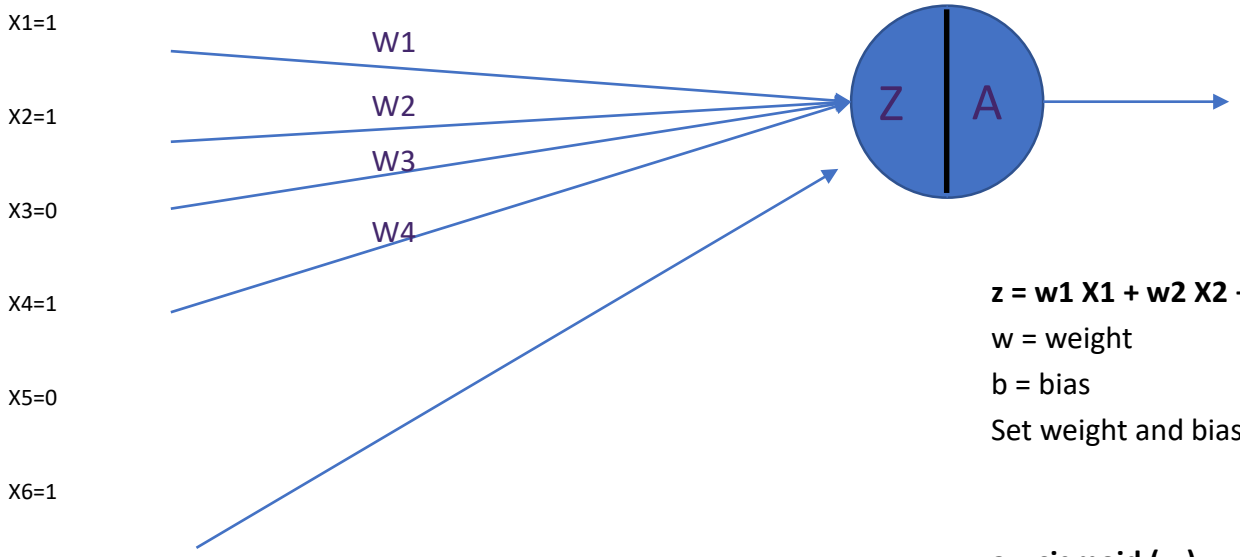
Om de afbeelding van de kat (input) te koppelen aan de beoordeling ‘dit is een kat’ (output), wordt van iedere pixel de RGB waarde genomen.

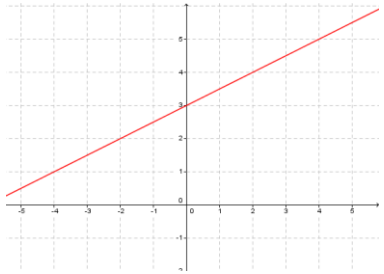


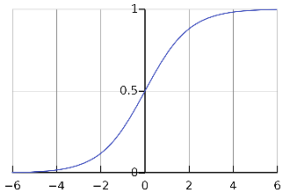
Een afbeelding van 64 x 64 pixels heeft dus 64 x 64 x 3 = 12.288 input-waarden om mee te rekenen!



Deep learning met één node







z = w1 X1 + w2 X2 + w3 X3 + w4 X4 + b

w = weight

b = bias

Set weight and bias to make Y'= Y

a = sigmoid (z)

adds non-linearity

Loss = difference y vs a (Y')

Y = target outcome

Y'= actual outcome

Cost = Average of the Loss of all examples

Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta(x^{(i)}), y^{(i)}\right)$$

$$Cost\left(h_\theta(x), y\right) = \begin{cases} -\log(h_\theta(x)) & if\ y = 1 \\ -\log(1 - h_\theta(x)) & if\ y = 0 \end{cases}$$

- This is the "simplified" formula —>

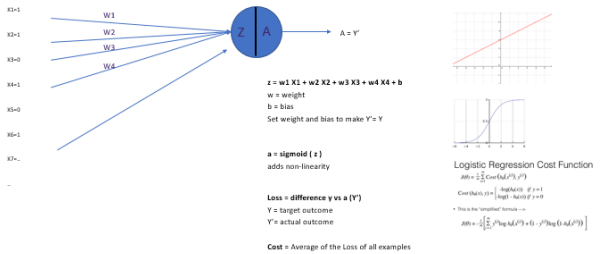
$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)\right]$$

Ondertitel

Deep learning met één node

Neurale netwerken hebben meerdere lagen. Eén laag is de input-laag, één laag is de output laag. Daartussen zitten één of (vrijwel altijd) veel meer ‘verborgen’ lagen. Lagen hebben één of meer nodes. De node in de huidige laag krijgt het volgende door:

- De waarden van alle nodes in de vorige laag, waarbij iedere aanleverende node een weging (‘weight’) krijgt.
- Een bias, ofwel correctiegetal, specifiek horend bij de node in de huidige laag.



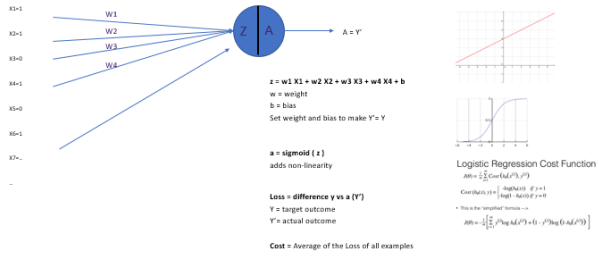
Ondertitel

Deep learning met één node

(vervolg) De waarde van de huidige node wordt waarschijnlijk onwerkbaar groot, daarom wordt deze waarde gecorrigeerd naar bijvoorbeeld een getal tussen -1 en 1. Dit heet ‘activeren’ en kan met functies zoals ‘sigmoid’.

Als het netwerk in de output-node (dat is er meestal slechts één) voorspelt dat een plaatje voor 0,71 (ofwel 71%) zekerheid een kat is, en het is daadwerkelijk een kat (ofwel kat = 1,00), dan is de ‘loss’ voor dit plaatje 0,29.

De ‘cost’ is de gemiddelde loss voor álle plaatjes.



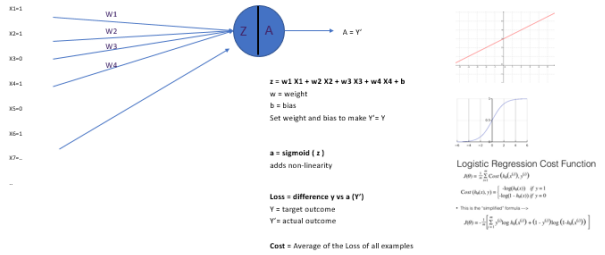
Ondertitel

Deep learning met één node

(vervolg) En dan de kern van neurale netwerken: door voor het gehele netwerk steeds een klein beetje andere waarden te kiezen voor de weights en biases, zal je cost (ofwel ‘hoever zit je er gemiddeld naast’) ook steeds een beetje veranderen.

Als blijkt dat je cost daalt, ben je op de goede weg en pas je de waarden van de weights en biases de volgende keer in dezelfde richting aan (dus nog iets kleiner of juist nog iets groter). Stijgt de cost, dan ga je met de weights en biases de andere richting uit.

En zo leert een neuraal netwerk!



Ondertitel

Deep learning – gradient descent

Minimize the cost (back propagation)

dz = Y’- Y

dw(i) = x(i) dz

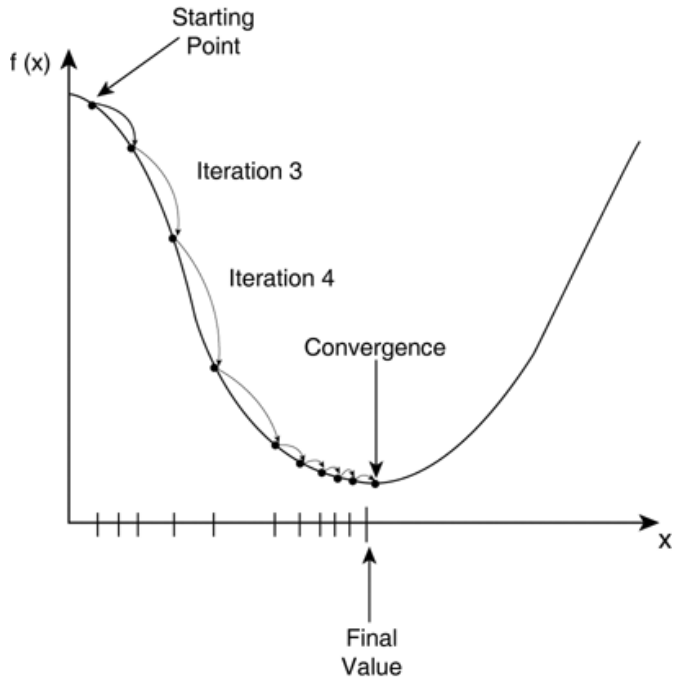
db = dz

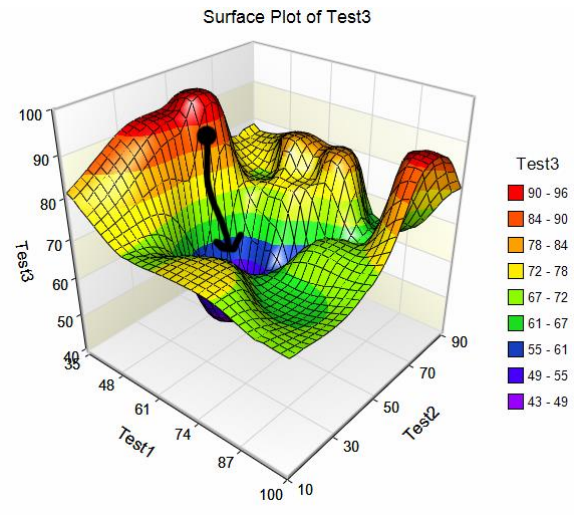
Gradient descent

w(new) = w(old) – alpha dw(i)

b(new) = b(old) – alpha db

alpha = learning rate





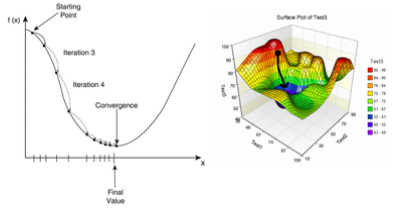
Deep learning – gradient descent

Het berekenen van een voorspelde output-waarde op basis van je input, heet forward propagation. Het berekenen in hoeverre je goed of fout zat (de cost), en welke nodes in je netwerk daar het meest aan hebben bijgedragen, heet back propagation.

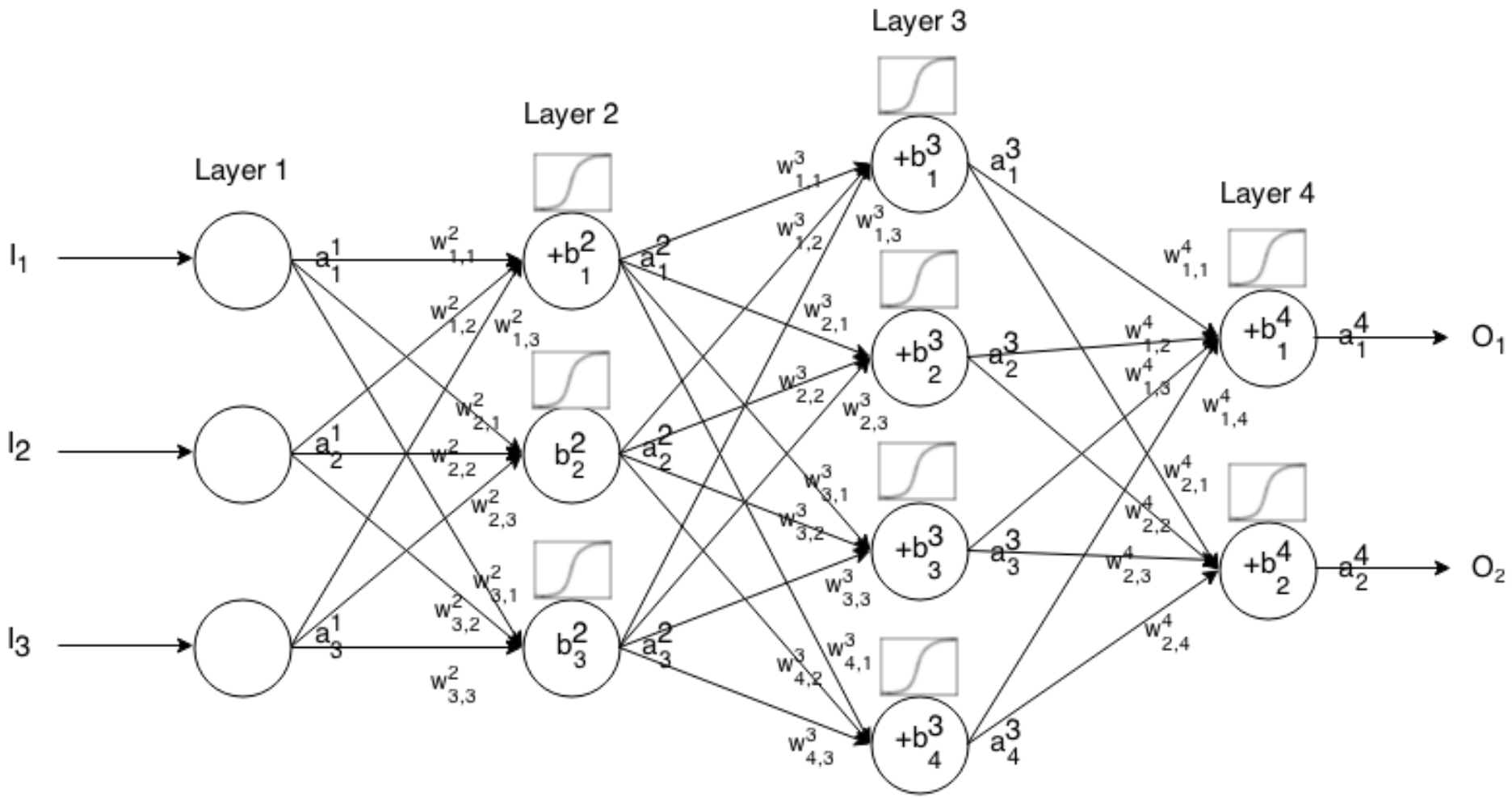
Gradient descent is het geleidelijk laten dalen van je cost functie. De learning rate is de snelheid waarmee je de weights en biases aanpast. Een te kleine learning rate kost je heel veel tijd. Een te grote learning rate geeft de kans dat je over het optimale punt heen springt.

Minimize the cost (back propagation)
dz = Y'- Y
dw(i) = x(i) dz
db = dz

Gradient descent
w(new) = w(old) – alpha dw(i)
b(new) = b(old) – alpha db
alpha = learning rate

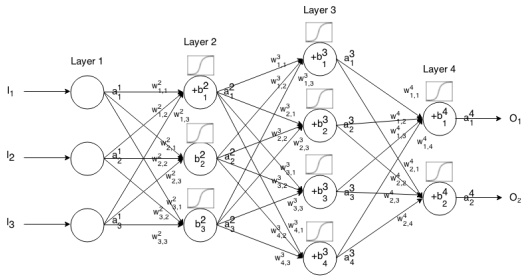


Deep learning – fully connected NN



Deep learning – fully connected NN

Een fully connected neuraal netwerk is een netwerk waarbij alle nodes van een laag zijn verbonden met alle nodes van de vorige laag. Dit wordt veel gebruikt, maar is uiteraard ook het meest zwaar om te trainen. Daarnaast zijn er netwerk-ontwerpen en trainingsmethoden waarbij het handig is om juist NIET alle nodes met elkaar te verbinden.



Deep Learning - Overfitting

High Bias – Underfitting

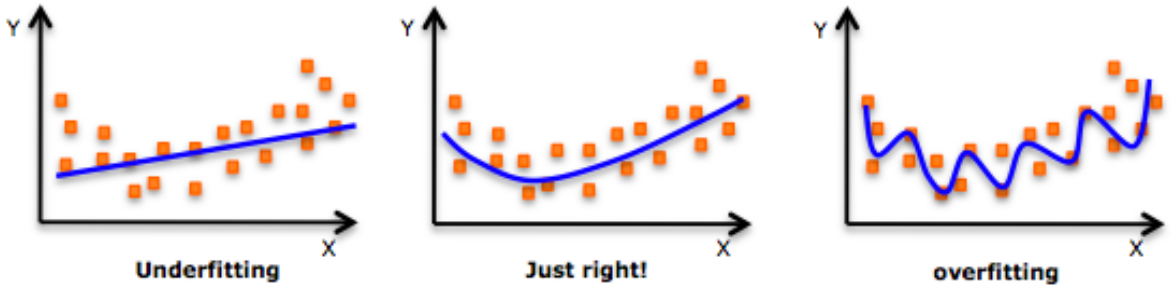
- Training error high
- Test error high

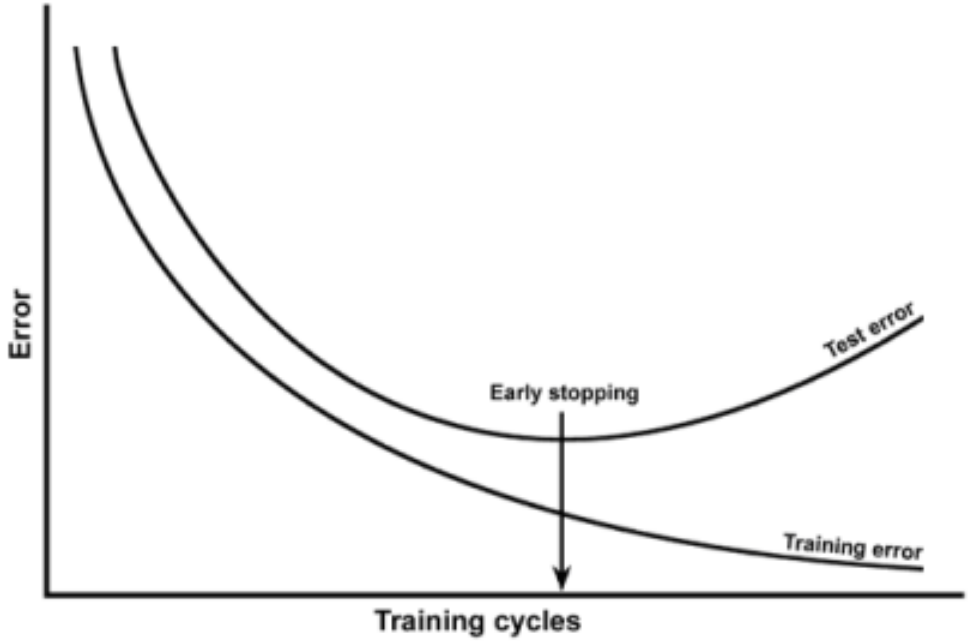
Just Right

- Training error low
- Test error low

High variance – Overfitting

- Training error low
- Test error high





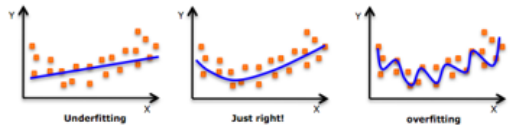
Deep Learning - Overfitting

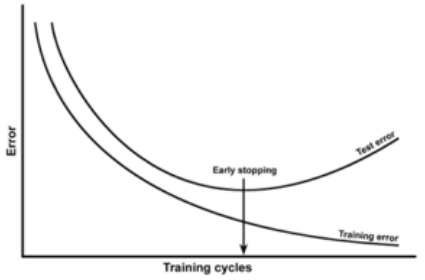
Het model (ofwel het neurale netwerk) trainen zorgt ervoor, dat je weights en biases steeds beter worden en je voorspellingen dus ook. Je kunt echter ook te VEEL trainen. In dat geval krijg je overfitting: het model is helemaal gericht op het voorspellen van de voorbeelden die het heeft gekregen in de training set, maar is niet meer in staat om de algemene trend te herkennen. Denk daarbij aan foto's van katten die hij nog NIET heeft gezien. Dit stel je vast door dergelijke foto's aan te bieden in de vorm van een test set (want uiteraard moet er getest worden! ☺).

High Bias – Underfitting
- Training error high
- Test error high

Just Right
- Training error low
- Test error low

High variance – Overfitting
- Training error low
- Test error high





Deep Learning - Regularization

L1/L2 Regularization

- Add term to cost function to penalize large weights
- Lambda = regularization rate
- Big lambda
  - w is small
  - Z is small
  - More linear part of network
  - Less overfitting
- Better gradient descent

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Deep Learning - Regularization

Regularization is een maatregel om overfitting te voorkomen. Het zorgt ervoor dat nodes die al te zwaar meewegen in het eindresultaat, worden ‘gedempt’. Zo krijg je een regressielijn met minder kronkels (zoals enkele sheets hiervoor te zien was).

L1/L2 Regularization

- Add term to cost function to penalize large weights
- Lambda = regularization rate
- Big lambda
  - w is small
  - Z is small
  - More linear part of network
  - Less overfitting
- Better gradient descent

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

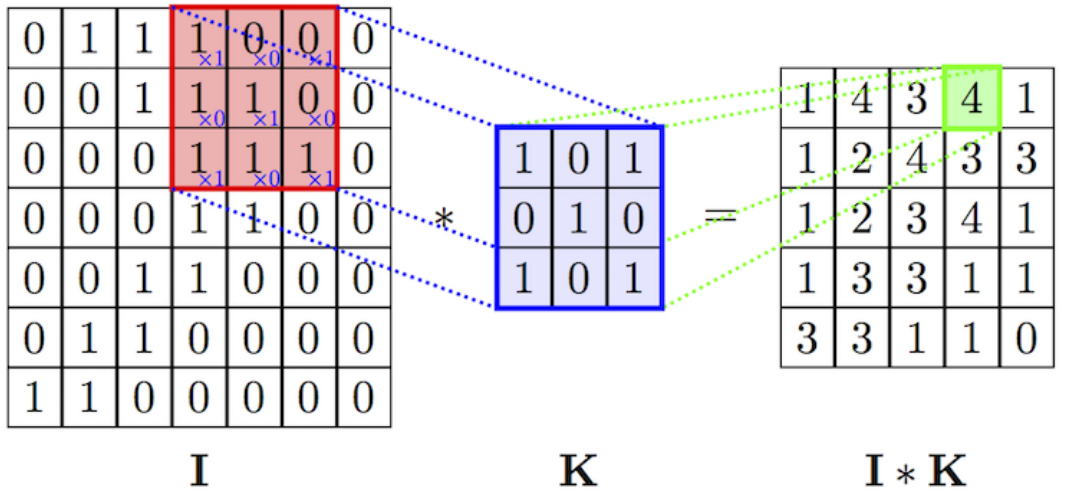
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Deep Learning - CNN

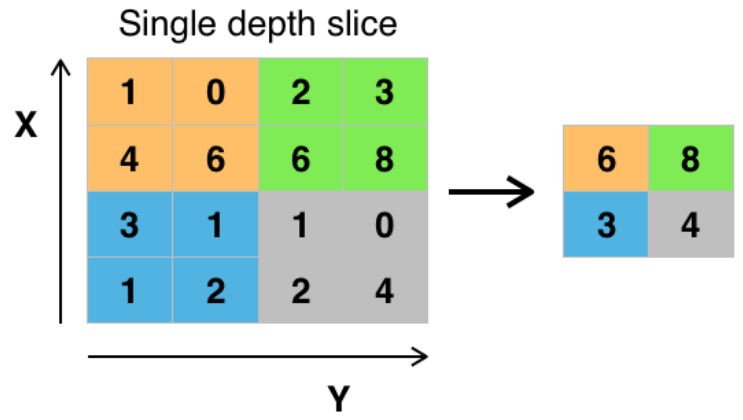
Convolution layer

- Sum of elementwise product
- Using a filter to detect edges



Pooling layer

- Maximum / average
- Reduce height/width
- Reduce sensitivity to location

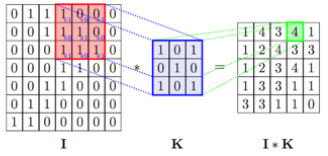


Deep Learning - CNN

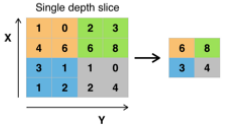
Het beoordelen en wegen van alle pixels in een plaatje, kost héél veel rekenkracht. Een convolutional neural network is een stuk efficiënter, deze gaat op zoek naar voorgedefinieerde vormen (zoals een kruis-vorm of een hoek-vorm). Op die manier krijg je een ‘laag’ over het plaatje die aangeeft waar deze vorm is gevonden. Meerdere van deze lagen geeft een beeld van wat er op het plaatje staat, terwijl je minder afhankelijk bent van WAAR het op het plaatje staat.

Je kunt nog allerlei andere bewerkingen met de pixelwaarden doen, om minder input-waarden te krijgen, zoals inputwaarden samenvoegen (‘pooling’).

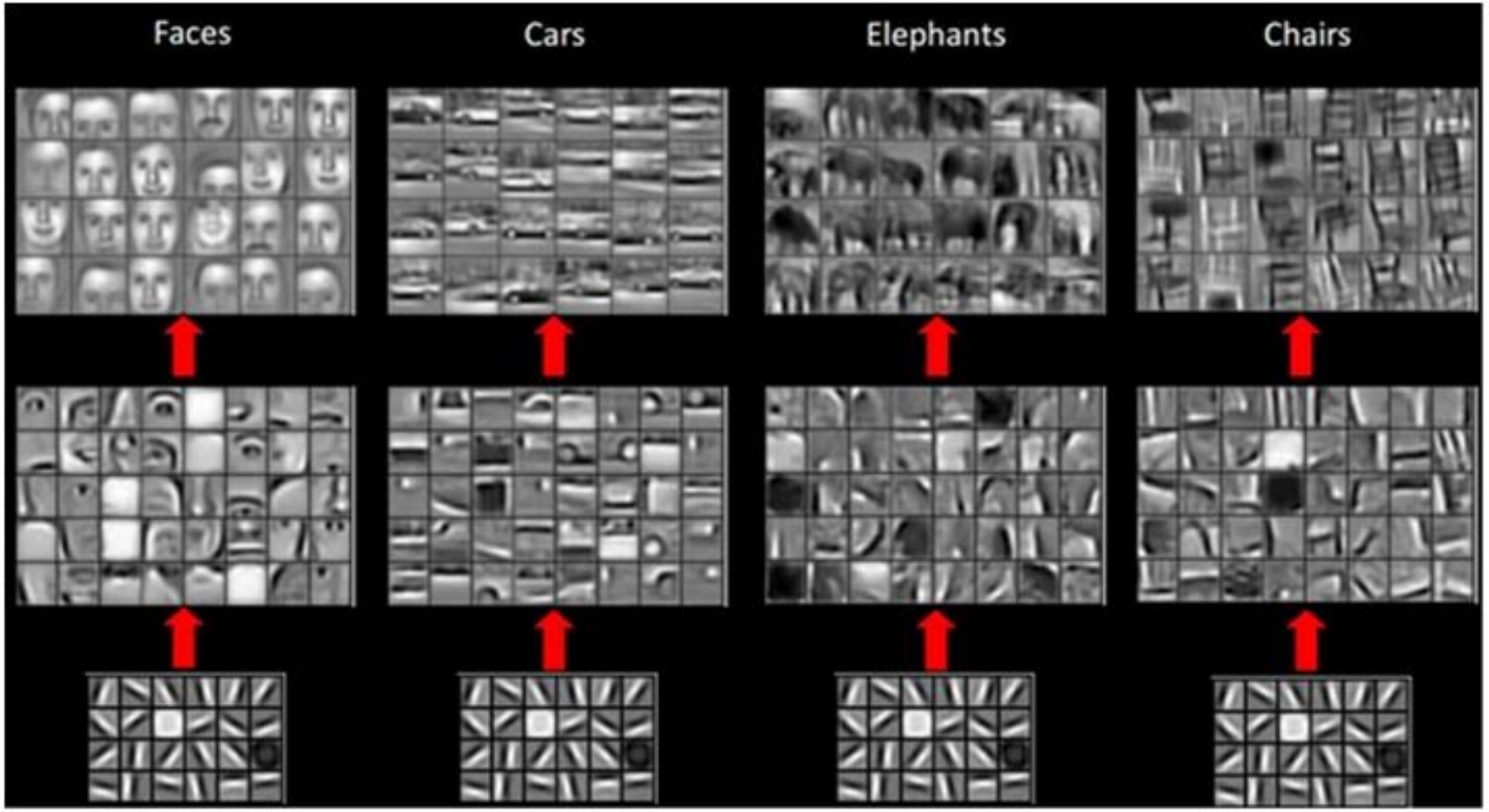
Convolution layer
- Sum of elementwise product
- Using a filter to detect edges



Pooling layer
- Maximum / average
- Reduce height/width
- Reduce sensitivity to location

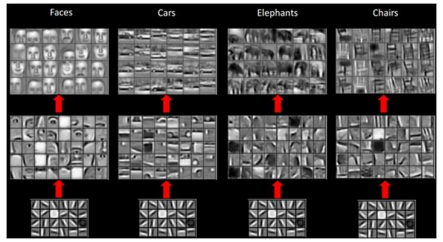


Deep Learning - CNN

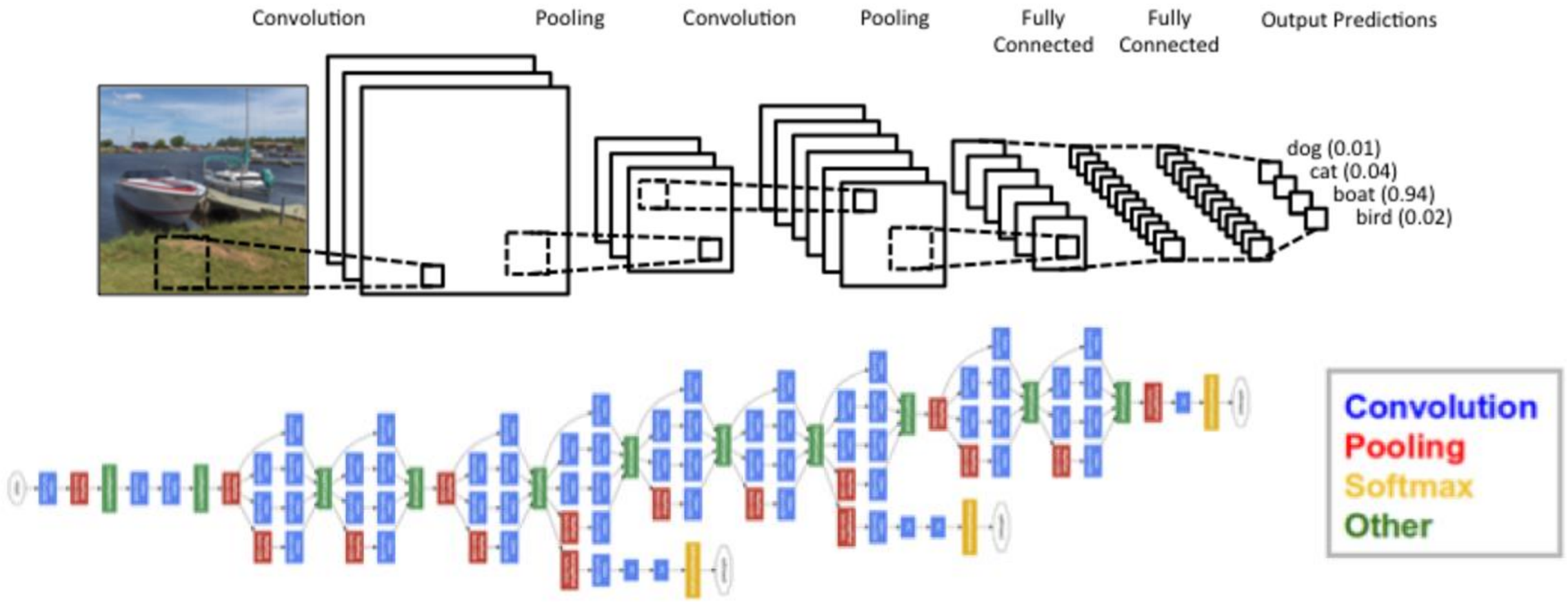


Deep Learning - CNN

De basisvormen van afbeeldingen zijn vaak dezelfde, dit zie je in onderin het plaatje. Als je deze basisvormen al hebt getraind in het eerste deel van het neurale netwerk, kun je later een stuk sneller trainen op het herkennen van gezichten, auto’s, olifanten, stoelen, enzovoort.



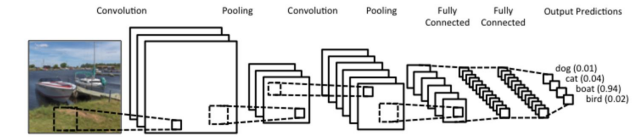
Deep Learning - CNN

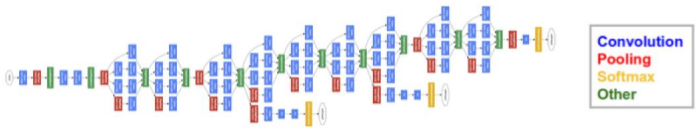


Deep Learning - CNN

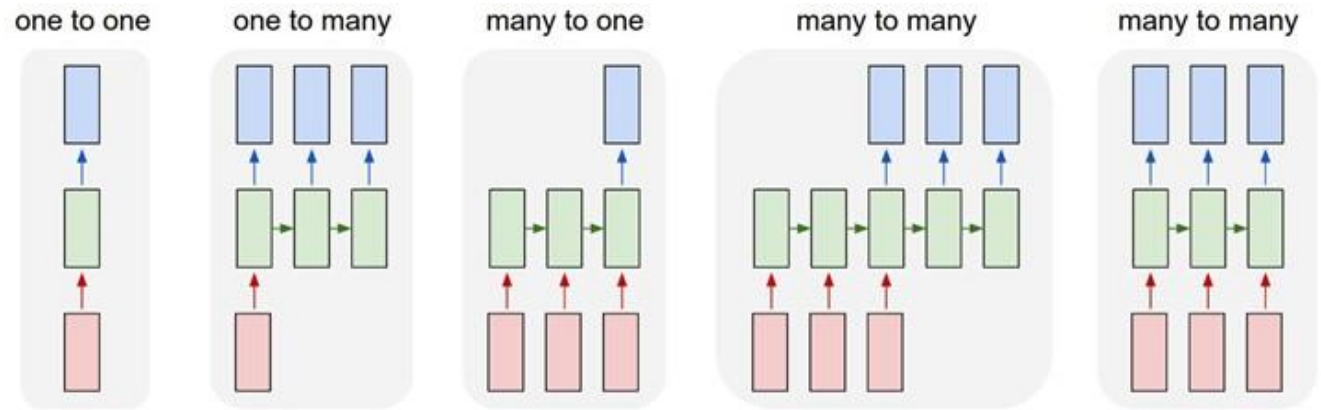
Neurale netwerken, ook CNN’s, hebben vaak een groot aantal lagen waar goed over moet worden nagedacht. Dit is nog allemaal mensenwerk.

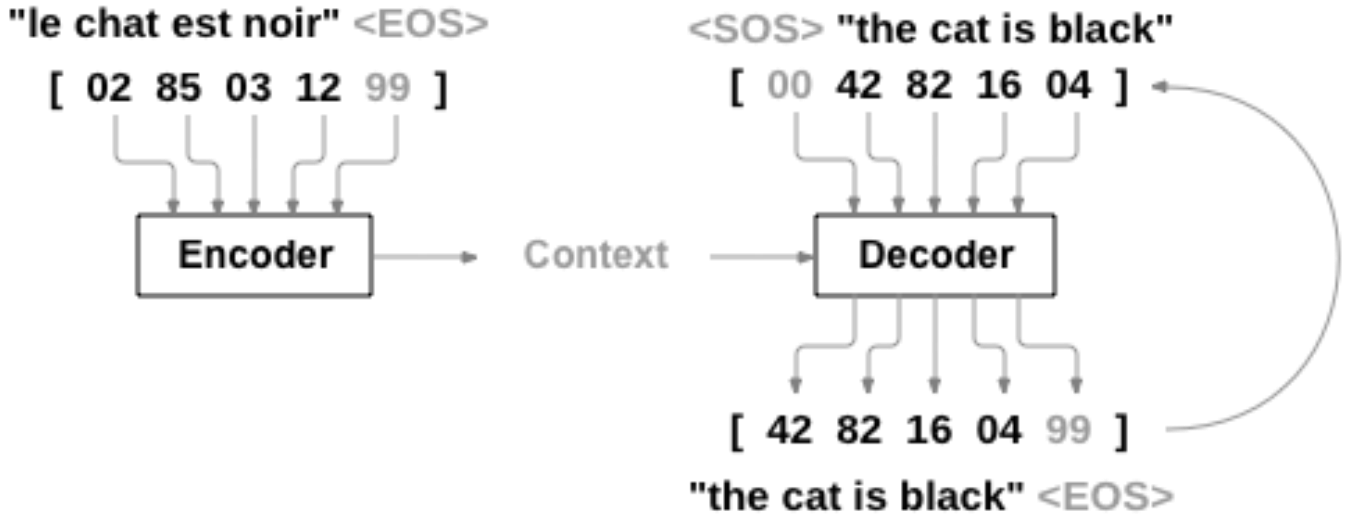
- Convolutie: vormen herkennen
- Pooling: meerdere pixelwaarden samenvoegen tot één waarde, bijvoorbeeld door van die waarden de hoogste te kiezen (maxpooling).
- Softmax: het omzetten van absolute waarden naar relatieve waarden (ten opzichte van elkaar, zodat de som van de waarden 1 is)





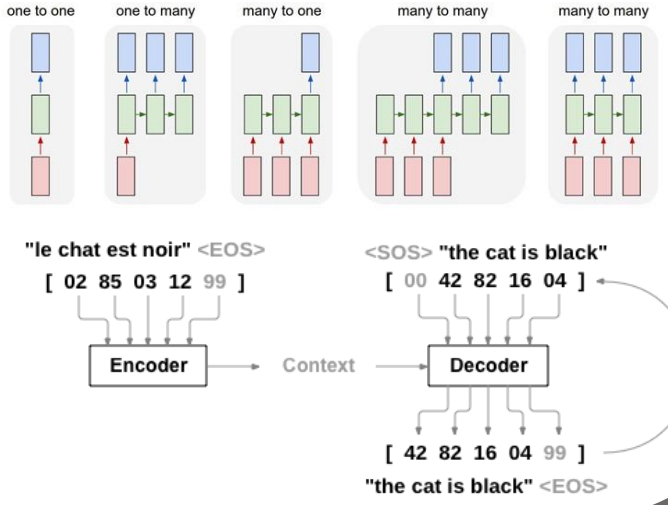
Deep Learning - RNN





Deep Learning - RNN

Een Recurrent Neural Network heeft de mogelijkheid om de output van het netwerk te hergebruiken als input voor zichzelf. En niet alleen de meest recente output, álle resultaten tot dat moment kunnen worden meegenomen. Het netwerk kan doorlopend inputs tot zich nemen (rode blokken) of outputs geven (blauwe blokken). Door grote hoeveelheden tekst te trainen, kan het voorspellen wat de volgende letter of het volgende woord waarschijnlijk is.



Dit is bijvoorbeeld hoe Google je zoekopdrachten aanvult en je zinnen vertaalt.

Vragen?

Neem contact op met de werkgroep Testen met AI.
Kijk op testnet.org of mail naar werkgroepen@testnet.org.