# ‘OP HET EERSTE GEZICHT’

*Herkennen van criminelen op basis van gezichtskenmerken met behulp van Artificial Intelligence.*


**Naam:** Gerwin van der Lugt
**School:** Stedelijk Gymnasium Arnhem
**Klas**: 6B
**Vakken**: Wiskunde
Informatica

# Inhoudsopgave

# Inleiding

## Onderwerpkeuze

Computertechnologie ontwikkelt zich in een hoog tempo. Een aantal decennia geleden begonnen wetenschappers voor het eerst te dromen van *Artificial Intelligence* (AI), de mogelijkheid om een echt brein te simuleren met een computer. Om dit te doen is veel processorkracht nodig en die is al jaren onderhevig aan exponentiële groei. Informatici hebben vanaf de jaren '80 hard gewerkt aan deze geheel nieuwe tak binnen de computerwetenschappen. En dat heeft zijn vruchten afgeworpen. Langzaam maar zeker begint de AI onmisbaar te worden in ons dagelijks leven.

Mijn onderwerpskeuze was niet erg moeilijk. AI is een opkomende technologie met vele toepassingen. De wiskunde die erachter steekt is moeilijk, maar niet onmogelijk te begrijpen. In combinatie met het benodigde programmeerwerk leek mij dit een leuke uitdaging.

Om mijn onderwerp wat concreter te maken ga ik dieper in op één van de meestgebruikte AI modellen: *Artificial Neural Networks* (ANN).

## Onderzoek

Ik doe onderzoek naar het toepassen van ANN's om criminelen te kunnen herkennen op basis van hun gezichtskenmerken. Om dit te kunnen bereiken behandel ik in het eerste hoofdstuk de rol van ANN's binnen de wetenschap van de AI. In het hoofdstuk daarna doe ik onderzoek naar de wiskunde die achter ANN's steekt. Die legt de grondslag voor het programmeerwerk, welke wordt behandeld in het derde hoofdstuk.

In het laatste hoofdstuk gebruik ik de opgedane kennis om de hoofdvraag van dit onderzoek te beantwoorden: *Is het mogelijk om met behulp van ANN's criminelen te identificeren op basis van hun gezichtskenmerken?* Ten slotte ga ik de ethische vraagstukken van dit onderzoek bespreken.

## Terminologie en taalgebruik

AI is een onderzoeksgebied waar binnen over het algemeen in het Engels wordt geschreven. Mede doordat het zo nieuw is zijn veel van de vaktermen (nog) niet vertaald in het Nederlands. Alle niet-Nederlandse termen zijn ter herkenning cursief gedrukt. Tevens gebruik ik afkortingen voor veelgebruikte termen zoals *Artificial Intelligence* ( = AI) en *Artificial Neural Networks* ( = ANN), deze zijn niet cursief gedrukt. In de verklarende woordenlijst kunnen al deze termen met uitleg (en eventueel de afkorting) teruggevonden worden.

# 1. Artificial Intelligence en Artificial Neural Networks

## 1.1 Ontstaan en ontwikkeling van AI

AI is de wetenschap van het maken van intelligente systemen. Een systeem is intelligent wanneer het omgevingsinformatie verwerkt en op basis daarvan acties uitvoert die de kans op succes maximaliseren. Het basisprincipe van AI is dat de werking van het menselijk brein zó precies kan worden beschreven dat computers deze kunnen simuleren. Deze opvatting is erg controversieel. Over het algemeen wordt AI gezien als een wetenschap die een lange tijd onderhevig is geweest aan groot optimisme.

Officieel is AI opgericht als wetenschap in 1956. Tot ongeveer 1974 is er veel onderzoek gedaan in het veld. Veel van de mensen die in die tijd werkten aan onderzoeksprojecten met betrekking tot AI geloven dat de problemen van AI binnen afzienbare tijd allemaal worden opgelost. Snel wordt echter duidelijk dat dit flink tegenvalt. In 1974 zet de Amerikaanse en de Britse regering alle subsidie voor AI projecten stop. De jaren die volgen worden de 'AI winter' genoemd. In deze tijd is het ontzettend moeilijk om mensen te vinden die willen investeren in AI onderzoek. Pas in de jaren '80, wanneer enkele particuliere bedrijven enorme commerciële successen boeken met AI technologieën, trekt de interesse weer aan. In de tijd die daarna volgt zijn de grootste successen geboekt in de wetenschapstak van AI.

In 1997 wordt al dit werk beloond met de grootste doorbraak in de AI tot nu toe. De IBM supercomputer 'Deep Blue' verslaat de wereldkampioen schaken in een historische wedstrijd. Deze mijlpaal ontvangt veel aandacht van de media en leidt tot grotere aandacht van de massa voor AI. In 2011 gaat IBM opnieuw een AI uitdaging aan. Deze keer speelt een nieuwe supercomputer Watson tegen de beste Jeopardy! spelers op de wereld. In het Jeopardy! spel krijgen de spelers moeilijke vragen gesteld. De grootste uitdaging voor Deep Blue is het begrijpen van de vragen, die hebben immers vaak een dubbele betekenis of zijn metaforisch bedoeld. In een uitzending die door miljoenen mensen wordt bekeken wint Watson het spel.

## 1.2 De Turing-test

Sinds het ontstaan van AI is er altijd onenigheid geweest over wanneer een computer intelligent genoemd kan worden en of computers kunnen denken. In 1950 publiceerde Alan Turing een artikel waarin hij deze vraag stelde. Volgens Turing lag het probleem in het woord ‘denken’. Het is moeilijk om dit woord te definiëren in de context van een computer. In zijn artikel beschrijft hij een experiment dat volgens hem een bijna perfecte manier is om te testen of een systeem intelligent is.

Het experiment gaat als volgt: er zijn drie kamers met in de eerste kamer een computersysteem, in de tweede een mens en in de laatste een menselijke jury (zie fig. 1). De jury kan door middel van een computerscherm communiceren met

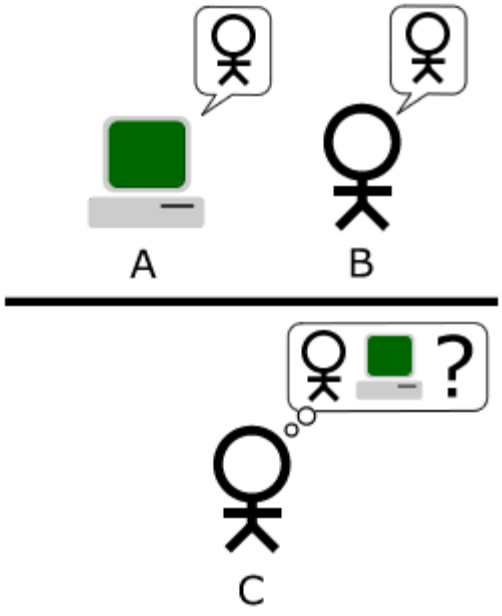



*fig 1. De Turing test: de jury C is gescheiden van het systeem A en het mens B.*

het systeem en de mens. De jury weet niet in welke kamer het systeem zit, en ik welke de mens. Door vragen te stellen moet de jury binnen een redelijke tijd kunnen aangeven welke van de twee het systeem is. Deze test wordt meerdere malen herhaald met telkens een andere jury. Wanneer aan het einde het percentage van foute oordelen even hoog is of hoger dan het percentage goede oordelen kan het systeem als intelligent worden beschouwd. Tot op de dag van vandaag zijn er geen computers die aan deze voorwaarde kunnen voldoen. Alan Turing voorspelde in zijn artikel dat de *Turing-test* rond het jaar 2000 voor het eerst zou worden gewonnen door computers maar die voorspelling is niet uitgekomen. Ray Kurzweil, een futurist werkzaam bij Google, voorspelt dat dit zal gebeuren in 2029.

## 1.3 Methodes voor het maken van intelligente systemen

AI is in essentie de wetenschap die als doel heeft het maken van intelligente systemen. In de loop van de tijd zijn er een aantal manieren ontwikkeld om dat te doen:

- Zoek en optimalisatie-algoritmes. Deze algoritmes zoeken door een set van oplossing en selecteren de beste. Speciale zoek-algoritmes zijn optimalisatie-algoritmes. Ze beginnen met zoeken op een willekeurige plek binnen de oplossingen en verfijnen de gevonden oplossing totdat er geen verfijningen meer mogelijk zijn.
- Logica. Met behulp van logische stappen kunnen bepaalde problemen tot een oplossing komen. Een goed voorbeeld hiervan is *fuzzy logic*. *Fuzzy logic* definieert elke bewering als geheel goed (1) of helemaal fout (0). Een combinatie van ware en foute beweringen kunnen tot een bepaalde oplossing leiden.
- Probabilistische algoritmes voor onzeker redeneren. Hieronder vallen wiskundige algoritmes die oplossingen kunnen vinden voor problemen waarvan niet alle gegevens beschikbaar zijn.
- Classificatie en statistisch leren. Problemen worden opgelost door zogeheten *controllers*. De *controllers* lossen problemen op basis van informatie van *classifiers. Classifiers* gebruiken eerder ingeziene informatie om een gegeven in te delen in een bepaalde groep. *Artificial Neural Networks* vallen onder deze categorie.

## 1.4 ANN's en het menselijk brein

Een ANN is de enige vorm van AI die is geïnspireerd door het menselijk brein. Om een goed beeld te krijgen van het mechanisme achter een ANN ga ik eerst globaal de werking van het brein uitleggen. Het brein bestaat uit veel soorten cellen. De belangrijkste soort cel is het neuron. Al onze gedachten, emoties, acties, herinneringen zijn een product van signalen die door het neuron worden verwerkt. Deze signalen komen binnen bij de dendrieten. Deze uiteindes van het neuron lopen uiteen en splitsen zich in een soort boomstructuur. Aan het andere uiteinde van het neuron zit de axon. De axon geeft het signaal weer door aan andere neuronen of cellen (zoals spiercellen of cellen van een ander orgaan).

Het volgende neuron ontvangt het signaal in de synaps. De overdracht van het signaal gaat door middel van een chemische reactie. De belangrijkste stof die hiervoor wordt gebruikt is de neurotransmitter. Deze wordt vrijgemaakt uit de axon en opgevangen in de synaps door de receptor.

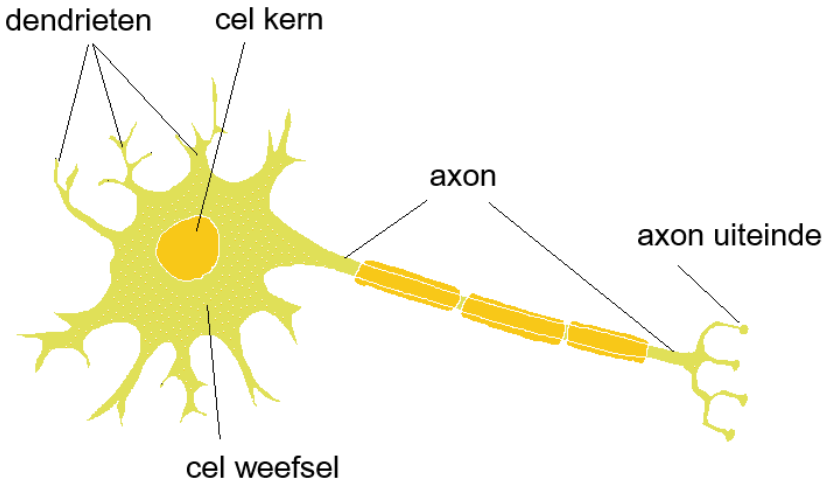



*fig 2. De neuron, met namen van de onderdelen.*

De sterkte van het signaal dat wordt doorgegeven hangt af van de neurotransmitter en receptor. Het veranderen van deze stoffen kan een groot effect hebben op je denkvermogen. Op bepaalde plekken in je brein kan het emoties opwekken of verdoezelen, effect hebben op je coördinatievermogen, gevoel van tijd en plaats, vermogen om beslissingen te nemen en op de manier waarop je visuele informatie verwerkt. Dit is ook de manier waarop bepaalde medicijnen en drugs je brein beïnvloeden.

Een groot netwerk van neuronen, zoals in je brein, kan alle informatie die binnenkomt uit je zintuigen verwerken en op basis daarvan acties ondernemen (door bijvoorbeeld je spieren aan te sturen). Die inkomende informatie is de input, en de acties die je brein onderneemt vormen de output. Om het mechanisme van ons brein te simuleren wordt de input beschouwd als de set van problemen en de output als de daar bijbehorende oplossingen. Wetenschappers hebben de werking van het neurale netwerk in het brein gemodelleerd en geprogrammeerd. De hierdoor ontwikkelde virtuele breinen heten *Artificial Neural Networks* (ANN's).

## 1.5 Structuur van een ANN

Om het neurale netwerk van het brein toe te passen op een computer zijn bepaalde aspecten van de biologische werking van het brein versimpeld of aangepast. Allereerst zijn de neuronen niet, zoals in het brein, willekeurig met elkaar verbonden. Een ANN bestaat uit *layers*, lagen met daarin een aantal neuronen. De eerste laag is altijd de *input layer* en de laatste de *output layer*. Alle lagen daar tussenin bestaan uit neuronen die als tussenstops dienen voor het probleemoplossingsmechanisme maar geen echte input of outputwaarde vertegenwoordigen. Ze worden daarom *hidden layers* genoemd. In fig. 3 zie je een ANN met 1 *hidden layer*. Hoe hoger het aantal *hidden layers*, hoe slimmer het ANN zal zijn.

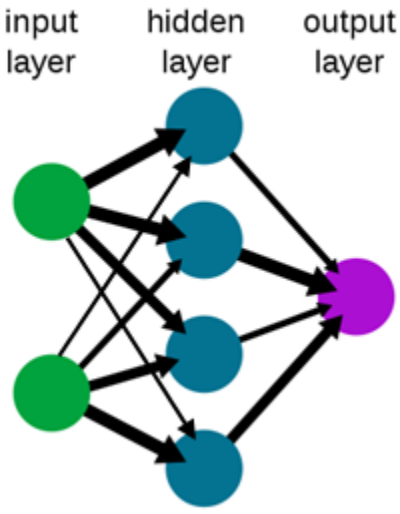



*fig 3. Opbouw van een ANN in lagen.*

De neuronen zijn met elkaar verbonden door de axonen. Een neuron is alleen verbonden met neuronen uit de omliggende lagen. In de meeste gevallen worden alle neuronen in de ene laag verbonden met alle neuronen in

de volgende laag om het maximale aantal axonen te maken. Dit komt omdat (net als met het aantal *layers*) geldt dat hoe meer axonen er zijn, hoe slimmer het netwerk is.

## 1.6 Een ANN gebruiken

Het toepassen van een ANN bestaat uit twee fasen.

1. De training. In de training wordt het netwerk een probleem aangeboden waarvan de oplossing al is ingedeeld in een *classifier.* Het netwerk komt met een uitkomst. Daarna wordt het verschil berekend met het echte antwoord en door middel van een algoritme wordt het netwerk aangepast om het de volgende keer goed te doen.

2. Het oplossen. Nadat het netwerk goed genoeg is getraind kan de gebruiker generieke problemen aanbieden waarop het netwerk is getraind. Als alles goed is gegaan zal het netwerk een groot deel van de problemen correct oplossen.

Om een goed idee te krijgen van wat er mogelijk is met ANN's gebruik ik het volgende voorbeeld: ANN's worden vaak gebruikt voor handschriftherkenning. In dit probleem wordt als input een foto van de geschreven letter gebruikt en als *classifiers* alle letters van het alfabet. De foto's worden tot 24 bij 24 pixels verkleind. Elke pixel wordt gezien als een waarde tussen 0.0 en 1.0 waarin 0.0 een helemaal witte pixel is en 1.0 een geheel zwarte. Alles daar tussen in zijn grijswaarden. Elke pixel krijgt een inputneuron in het netwerk, dat levert dus 576 inputneuronen op. Daartussen worden 3 *hidden layers* met elk 2000 neuronen aangemaakt.

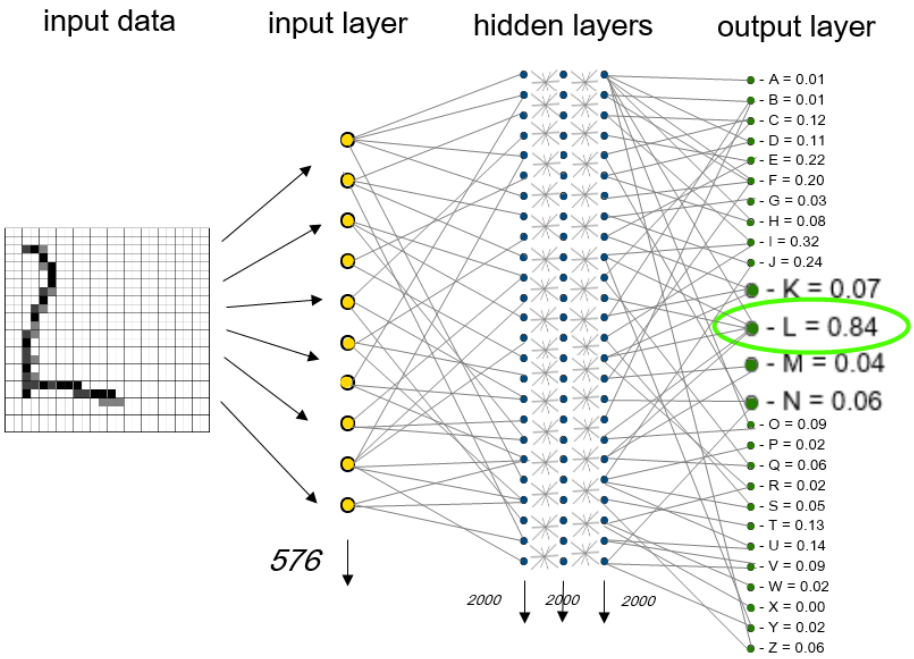



*fig 4. Opbouw van een ANN die gebruikt kan worden voor handschriftherkenning. Alle neuronen zijn verbonden met alle neuronen in de naastgelegen layer.*

Als laatste komt de *output layer* met 26 outputneuronen die elk een letter van het alfabet vertegenwoordigen. De waarde van het outputneuron bepaalt de kans dat de desbetreffende letter op de foto te zien was. Als bijvoorbeeld outputneuron 12 uitkomt op 0.84 is er een grote kans dat de letter L op de foto was te zien (zie voorbeeld ANN in fig. 4).

Zolang het netwerk in de leerfase is, worden er problemen ingevoerd die al met de hand zijn toegekend aan de correcte *classifier*. In het geval van dit netwerk zou als trainingsdata bijvoorbeeld een geschreven brief kunnen worden gebruikt die wordt overgetypt op de computer. Op die manier is van elke letter in de brief bekend welke het is. De foto's van de letters worden ingevoerd in het netwerk en de output vergeleken met de *target output*. De *target outputs* zijn de waardes van de *output layer* die de trainingsdata in de correcte *classifier* indelen. Als de eerste letter van de brief bijvoorbeeld een H is, dan zijn de *target outputs* voor elke outputneuron 0.00 behalve die van de H, die is 1.00. Een speciaal algoritme vergelijkt de *target outputs* met de echte outputs en verandert het netwerk naar gelang.

Nadat het netwerk genoeg trainingsdata heeft verwerkt kan het worden gebruikt voor herkenning. Nu kunnen handgeschreven letters worden ingevoerd zonder dat bekend is wat voor een letter het eigenlijk is. Het netwerk kan dan met redelijke zekerheid zeggen welke letter op de foto te zien is.

## 1.7 Samenvatting

AI is één van de nieuwste wetenschapstakken binnen de informatica en ontvangt veel aandacht. In de laatste jaren is er veel voortgang geboekt door wetenschappers. In die jaren is er een groot aantal manieren ontdekt om AI toe te passen. De meest gebruikte vorm van AI is classifiseren met behulp van ANN's. Een ANN is bijzonder aangezien zijn opbouw en werking zijn gebaseerd op de werking van het menselijk brein. Hierdoor is het mogelijk dat een ANN zonder speciaal gemaakte algoritmes toch moeilijke problemen kan oplossen die voorheen met de hand moesten worden opgelost.

# 2. De wiskunde achter ANN's

## 2.1 Feed-forward ANN's

Er zijn een aantal manieren om de input om te zetten in output met behulp van een ANN. De meest gebruikte ANN-vorm heet *feed-forward ANN*. Feed-forward slaat hier op de weg die de input data neemt om bij de output te komen. In *feed-forward ANN's* worden de inputwaardes vooruit gestuwd, door alle axonen, naar de neuronen in de volgende *layer*.

De waardes zijn onderweg in het netwerk onderhevig aan algoritmes en bewerkingen. In een *feed-forward ANN* gebeurt dit op twee plekken:

- In de axonen. Elke axon, de verbinding tussen twee neuronen, heeft een *weight*. De *weight* van de axon is simpelweg een variabele waarmee de waarde die door axon stroomt wordt vermenigvuldigd. In het neuron waar de waarde uitkomt, worden alle binnenkomende gewogen waardes bij elkaar opgeteld.

- In het neuron zelf. De inkomende *weights* worden bij elkaar opgeteld en met behulp van een *activation function* wordt de output van de neuronen berekend. Deze waarde wordt op zijn beurt doorgegeven aan de volgende laag.

## 2.2 Weighted sum

De functie die de inkomende waardes bij elkaar optelt heet de *weighted sum*. Vanuit het perspectief van het rechter neuron worden de waardes van alle *left-connected* neuronen gebruikt. Dat zijn de neuronen in de *layer* links van het neuron waarvoor de output moet worden berekend. De formule voor de *activation function* is dan als volgt:

$$A(\bar{x}, \bar{w}) = \sum_{i=0}^{n} x_i w_{ji}$$

*formule 1. Activation function met weighted sum.*

Hierin is $x$ de set van outputwaardes uit de *left-connected* neuronen. En $w$ is de set van de bijbehorende *weights*. Deze somfunctie neemt de loper $i$ met 0 als beginwaarde en loopt $n$ rondes, $n$ is dus het aantal *left-connected* neuronen. Voor elk van de *left-connected* neuronen berekent de functie de gewogen waarde door $x_i$ (waarde van het *left-connected* neuron) te vermenigvuldigen met $w_{ji}$ (de bijbehorende *weight*). De somfunctie telt al de gewogen waardes bij elkaar op en geeft de *weighted sum*.

## 2.3 Sigmoid function

De volgende stap in het berekenen van de output van het neuron is de *activation function* (ook wel *output function* genoemd). Het doel van de *activation function* is om de binnnekomende waardes zo te veranderen dat de kwaliteit van het netwerk omhoog gaat. De simpelste *activation function* is:

$$O(\bar{x}, \bar{w}) = \begin{cases} 0 \text{ if } A(\bar{x},\bar{w}) < 0 \\ 1 \text{ if } A(\bar{x},\bar{w}) > 0 \end{cases}$$

*formule 2. Simpel voorbeeld van een activation function: de step function.*

Wanneer de *weighted sum* lager is dan 0 wordt de output 0, anders 1. Deze functie wordt ook wel de *step function* genoemd. Deze *activation function* is echter niet geschikt voor complexe ANN's. Er wordt daarom gekozen voor een geavanceerdere *activation function* die vaak gebruikt wordt in ANN's: de *sigmoid function* (formule 3). Die heeft een mooiere verdelingsgraad (fig. 5).

$$O(\bar{x}, \bar{w}) = \frac{1}{1 + e^{-A(\bar{x},\bar{w})}}$$

*formule 3. Sigmoid function.*

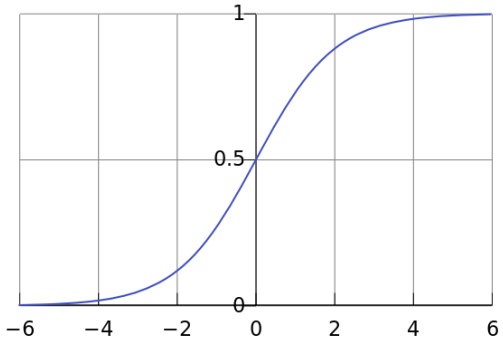



*fig 5. Grafiek van de sigmoid function.*

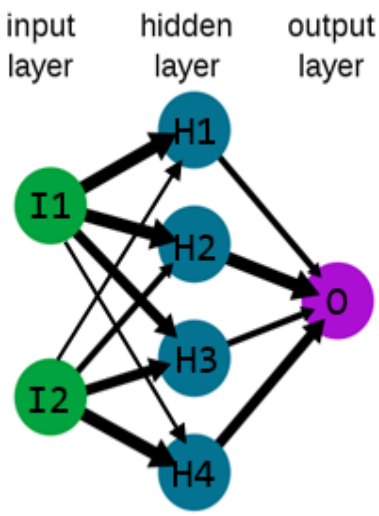
De *sigmoid function* is een exponentiële functie die gebruikt maakt van Eulers constante (*e*). Later in dit hoofdstuk wordt toegelicht waarom het gebruik van *e* hier zo ideaal is. Je ziet in de grafiek dat voor hoge positieve getallen de uitkomst dicht bij 1 ligt en voor lage negatieve getallen dicht bij 0. Dit zorgt ervoor dat de functie ongeveer hetzelfde werkt als de simpele *step function* maar met iets meer nuance. Zo draagt deze functie bij aan de effectiviteit van het ANN.

## 2.4 Rekenen aan de probleemoplossingsfase

De genoemde formules zijn de enige formules die nodig zijn om informatie door een ANN te stuwen. Ik pas de formules toe in rekenvoorbeeld 1. Hiervoor wordt een ANN met willekeurig gekozen *weights* gebruikt.

Het voorbeeldnetwerk bestaat uit drie *layers* met respectievelijk 2, 4 en 1 neuronen (zie fig. 3). In dit rekenvoorbeeld wordt de output berekend van het tweede neuron in de enige *hidden layer*. De inputneuronen hebben als beginwaardes respectievelijk 0.64 en 0.40 (zie fig. 6). De *weight* van de axon tussen het eerste inputneuron en het te berekenen neuron is 0.25. De *weight* tussen het tweede inputneuron en het te berekenen neuron is 0.50.

Eerst wordt de *weighted sum* berekend. Daarvoor worden de gewogen waardes van de *left-connected* neuronen opgeteld. Voor inputneuron 1 is dat 0.64 keer 0.25, dus 0.16. Voor inputneuron 2 is dat 0.40 keer 0.50, dus 0.20. Samen is dat 0.36 (zie rekenvoorbeeld 1 hieronder).




Gegevens:

- $x_{I1}$ = 0.64 (waarde van I1)
- $x_{I2}$ = 0.40 (waarde van I2)
- $w_{I1,H2}$ = 0.25 (*weight* tussen I1 en H2)
- $w_{I2,H2}$ = 0.50 (*weight* tussen I2 en H2)

Te berekenen: $x_{H2}$

1. Vul de *weighted sum* in.

$$A_{H2} = \sum x_{I_n} w_{H2,I_n} = (0.64 \times 0.25) + (0.40 \times 0.50) = 0.36$$

2. Vul de *sigmoid function* (*activation function*) in.

$$O_{H2} = \frac{1}{1 + e^{-A}} = 1 \div (1 + e^{-0.36}) = 0.59$$

Nieuwe waarde: $x_{H2}$ = **0.59**

*rekenvoorbeeld 1. Rekenen aan de probleemoplossingsfase van een ANN. De gebruikte waardes zijn verzonnen maar geven een goed beeld van een reële berekening.*

Dan wordt de output berekend met de *sigmoid function*. O(0.36) = 0.59. Dit is de uiteindelijke waarde van het neuron. In een volgende stap zou deze output weer worden gewogen in het outputneuron samen met de andere output uit de *hidden layer* enzovoorts.

## 2.5 Error function tijdens de trainingsfase

In het vorige voorbeeld werd een netwerk gebruikt met willekeurige *weights*. In het echt wordt een netwerk getraind om de optimale *weights* te vinden om het netwerk zo effectief mogelijk te maken. Dit gebeurt met behulp van de trainingsdata. De trainingsdata bestaat uit *patterns*. Een *pattern* is een set van inputs en *target outputs* die met de hand zijn vastgesteld en dus altijd goed zijn. Het netwerk wordt gevoed met de inputs van een *pattern* en de output wordt vergeleken met de *target output*. Op basis daarvan wordt het netwerk aangepast om beter te preseteren. Dit proces wordt wiskundig beschreven in het *backpropagation* algoritme.

Allereerst is het belangrijk om de output *error* te berekenen. De output *error* is een indicatie van hoe ver de output en *target output* uit elkaar liggen. De output *error* wordt berekend met de *error function*. Omdat alleen de *error* van de outputneuronen van belang is kan de *error function* alleen worden gebruikt op outputneuronen. De meest gebruikte *error function* is de *sum-of-squares error function*:

$$E(\bar{x}, \bar{w}, t) = \frac{1}{2}(t - O(\bar{x}, \bar{w}))^2$$

*formule 4. Sum-of-squares error function.*

Het verschil tussen de *target output* (t) en de echte output (O(x, w)) wordt gekwadrateerd zodat de uitkomst altijd positief is. En omdat de formule dan een hogere uitkomst geeft wanneer de *error* groot is. De factor van een ½ zorgt ervoor dat deze functie makkelijker af te leiden is. Later zal de afgeleide worden gebruikt.

Om de fout van het hele netwerk te verkrijgen wordt simpelweg de som van alle *errors* op de outputneuronen genomen. Deze waarde wordt vaak gebruikt om een beeld te krijgen van de effectiviteit van het netwerk op een *pattern.* Hoe hoger de foutwaarde, hoe slechter het netwerk presteert.

## 2.6 Gradient descent met partiële afgeleide

Nu kan worden berekend wat de *error* van een outputneuron is kan het netwerk worden aangepast door een set van *weights* te vinden die de *error* zo laag mogelijk houdt. Wanneer een grafiek wordt opgesteld met op de z-as de *error* en op de x en y as twee *weights* waar de *error* van afhankelijk is wordt een grafiek verkregen zoals te zien is in fig. 6. Het doel van het *backpropagation* algoritme is om de twee *weights* te vinden die de minimale *error* leveren. In fig. 6 ligt het nulpunt op (0, 0, 0).

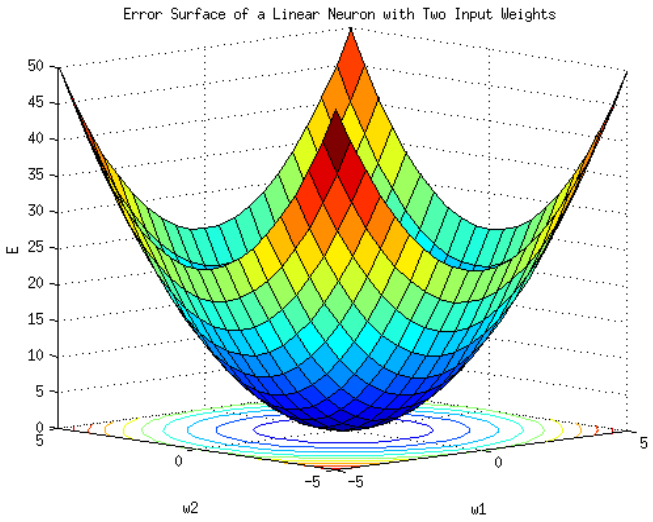



*fig 6. Grafiek van een error function met twee weights.*

Maar omdat de functie uit twee of meer variabelen bestaat kan het minimum niet gevonden worden op dezelfde manier als dat kan met parabolische functies. Wetenschappers hebben een andere methode gevonden om hetzelfde te bereiken. Deze methode heet de *gradient descent* en maakt gebruik van de partiële afgeleide om een benadering van het minimum te vinden. De *gradient descent* zoekt de optimale waarde van een variabele voor het bereiken van het minimum in multidimensionale functies.

Om het *backpropagation* algoritme te vinden moet de partiële afgeleide worden gevonden van de *error function* naar de *weight* $w_{ji}$. De partiële afgeleide naar $w_{ji}$ kan worden berekend door alle variabelen behalve $w_{ji}$ als constanten te beschouwen. Omdat de *error function* eigenlijk een kettingfunctie is met de *activation function* en de *weighted sum* kan de kettingregel worden gebruikt om de afgeleide te berekenen:

$$E'_{w_{ji}}(\bar{x}, \bar{w}, t) = (t - O) \cdot O' \cdot A'_{w_{ji}}$$

*formule 5. Tussenstap in het zoeken naar de partiële afgeleide van de error function naar $w_{ji}$.*

De afgeleide van *E(x, w)* is de eerste term. De factor van ½ is opgeheven door de factor 2 van de afgeleide. De afgeleide van *O(x, w)* is *O(1 – O)*. Hier is te zien waarom de *sigmoid function* zo'n goede keuze is voor *activation function*. De afgeleide van deze functie is namelijk een functie van zichzelf. Als laatste wordt de afgeleide van *A(x, w)* berekend. Dit is de *weighted sum* en maakt gebruik van *x* en *w*. Omdat hier de partiële afgeleide wordt berekend kan $x_i$ worden beschouwd als constante. De afgeleide van de 'constante' $x_i$ keer $w_{ji}$ is de constante $x_i$ zelf. Daarbij wordt nog een variabele $\eta$ toegevoegd, die fungeert als leerconstante (*learning rate*). Wanneer $\eta$ wordt verghoogd, versnelt die het leerproces. Als die echter te hoog is, schiet de uitkomst steeds voorbij het optimale minimum. De uiteindelijke formule voor het berekenen van de verandering in $w_{ji}$ is als volgt:

$$\Delta w_{ji} = \eta(t - O)O(1 - O)x_i$$

*formule 6. De partiële afgeleide van de error function naar $w_{ji}$ bepaalt de verandering van de weights in de output layer.*

## 2.7 Backpropagation algoritme samenstellen

De functie in formule 6 is alleen te gebruiken voor de *output layer* van het netwerk. Voor alle andere lagen is immers niet bekend wat de *target output* is. Om een formule te vinden voor de verandering van *weights* in *hidden layers* moet een benadering worden gevonden van *t – O* die met een zekere redelijkheid de werkelijke waarde nadert. Dit is mogelijk met behulp van de deltawaardes van de volgende laag. De deltawaarde wordt, vanuit het perspectief van een neuron in de volgende laag, bepaalt met de functie *δ*.

$$\delta = (\sum_{k=0}^{n} \delta_k w_{jk})O(1 - O) \quad \text{voor input neuronen}$$

$$\delta = (O - t)O(1 - O) \quad \text{voor output neuronen}$$

*formule 7. Bepaal de deltawaarde voor een neuron.*

Zoals te zien is, is de delta voor *output layers* hetzelfde als een gedeelte uit formule 6. De deltawaarde voor neuronen uit *hidden layers* hangt af van de gewogen som van de deltawaardes uit de *right-connected* neuronen. *Right-connected* neuronen zijn de neuronen waarmee dit neuron is verbonden in de volgende laag. Op deze manier stromen de deltawaardes vanuit de *output layer* naar links door het netwerk. Het is niet nodig om de deltawaardes van inputneuronen te bepalen aangezien er geen *left-connected* neuronen zijn om die te gebruiken.

Nu de deltawaarde is bepaald kan een functie worden gemaakt die de verandering van een *weight* kan berekenen voor axonen:

$$\Delta w_{ji}(n) = \eta \delta_j x_i + \alpha \Delta w_{ji}(n-1) \quad \text{met}$$

$$\delta = (\sum_{k=0}^{n} \delta_k w_{jk}) O(1-O) \quad \text{voor input neuronen}$$

$$\delta = (O-t) O(1-O) \quad \text{voor output neuronen}$$

*formule 8. Het backpropagation algoritme: de weight verandering voor hidden én outputneurons.*

In deze formule is $\delta_j$ de deltawaarde van het neuron. Verder is *n* een teller die omhoog gaat bij elk *pattern*. Achter de term die er als was, is nu een term toegevoegd. Deze term heeft een constante *α*. Deze constante is het momentum en wordt vermenigvuldigd met de verandering in *weight* van het vorige *pattern*. Zo beïnvloedt elke verandering in *weight* de volgende en weet de formule nog in welke richting het de vorige keer ging. Dit maakt de afdaling naar het minimum stabieler (zie fig. 7).

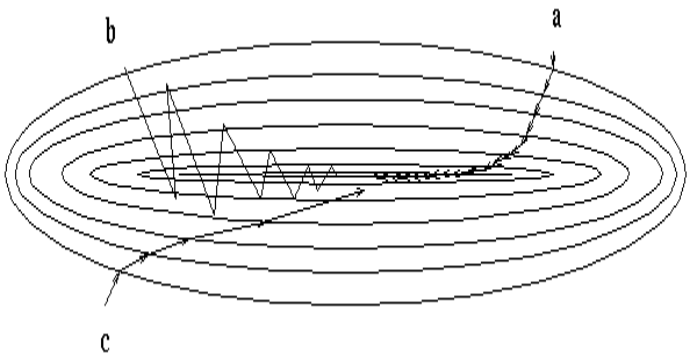


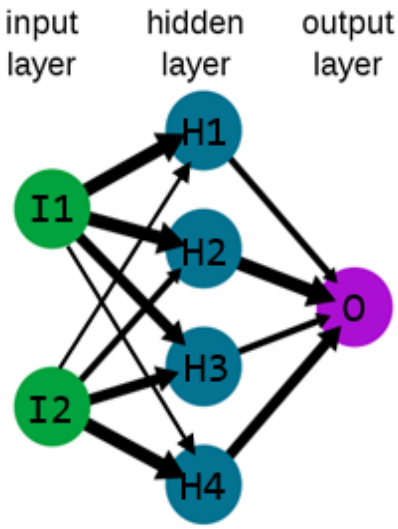

*fig 7. Afdaling in de multidimensionale grafiek van de error function naar het minimum. Route a wordt afgelegd bij een te kleine leerconstante η, en route b bij een te grote η. Route c wordt afgelegd bij een normale η en met goed afgesteld momentum α.*

## 2.8 Rekenen aan de leerfase

Om de theorie in de praktijk te brengen wordt hetzelfde netwerk als in fig. 6 genomen en getraind met een *pattern.* Omdat het netwerk uit één outputneuron bestaat is er maar één *target output*. De waarde hiervan wordt gesteld op 0.80. Veder zijn er in het rekenvoorbeeld enkele constanten opgenomen die nodig zijn voor de berekening.

Er wordt gezocht naar de nieuwe *weight* voor *hidden neuron* 2 (H2). Het rekenvoorbeeld bestaat uit vier stappen. In stap één wordt de deltawaarde van het outputneuron berekend. Deze is nodig in stap twee, daar wordt immers de deltawaarde voor H2 zelf berekend. Omdat er maar één neuron zit in de laatste *layer* is de som van de gewogen delta's gelijk aan de gewogen delta van het outputneuron (O). Met de leerconstante, momentum en de deltawaarde van H2 zijn alle gegevens benodigd voor het invullen van het *backpropagation* algoritme compleet. In de vierde en laatste stap wordt de uitkomst van het algoritme opgeteld bij de oude *weight*. (zie rekenvoorbeeld 2)

De uiteindelijke *weight* is veranderd van 0.2500 naar ongeveer 0.2502. Zoals je hier ziet zijn de veranderingen in het netwerk meestal maar erg klein. In een echt netwerk worden er velen duizenden *patterns* ingevuld voordat het netwerk voldoende getraind is.




*rekenvoorbeeld 2. Rekenen aan de leerfase van een ANN. De gebruikte waardes zijn verzonnen maar geven een goed beeld van een reële berekening.*

Gegevens:

- $\eta = 0.24$ (netwerk leerconstante)
- $\alpha = 0.04$ (netwerk *momentum*)
- $x_{I1} = 0.64$ (waarde van I1)
- $x_{I2} = 0.40$ (waarde van I2)
- $w_{I1,H2}(n-1) = 0.25$ (vorige *weight* tussen I1 en H2)
- $\Delta w_{I1,H2}(n-1) = 0.0001$ (onthouden *weight* verschil)
- $w_{H2,O}(n) = 0.30$ (*weight* tussen H2 en O)
- $x_O = 0.70$ (output van O)
- $t_O = 0.80$ (*target output* van O)

Te berekenen: $w_{I1,H2}(n)$

1. Bereken *delta* van O.

$$\delta_O = (t_O - x_O)x_O(1 - x_O) = (0.80 - 0.70) \times 0.70 \times (1 - 0.70) = 0{,}021$$

2. Daarmee wordt de *delta* van H2 berekend.

$$\delta_{H2} = \sum \delta_{O_n} w_{H2,O_n}(n)\, x_O(1 - x_O) = \delta_O w_{H2,O}(n) x_O(1 - x_O)$$
$$= 0{,}021 \times 0.30 \times 0.70 \times (1 - 0.70) = 0{,}001323$$

3. Nu wordt alles ingevuld in het *backpropagation* algoritme.

$$\Delta w_{I1,H2}(n) = \eta \delta_{H2} x_{I1} + \alpha \Delta w_{I1,H2}(n-1)$$
$$= 0.24 \times 0{,}001323 \times 0.64 + 0.04 \times 0.0001 = 0.0002072128$$

4. Tel de uitkomst op bij de oude *weight*.

$$w_{I1,H2}(n) = w_{I1,H2}(n-1) + \Delta w_{I1,H2}(n) = 0.25 + 0.00100768 = 0.2502072128$$

Nieuwe *weight*: $w_{I1.H2}(n)$ = **0.2502072128**

# 3. Het programmeren van ANN's

## 3.1 Programmeertaal, ontwikkelomgeving en methodiek

Voor het programmeren van ANN's zijn een aantal hulpmiddelen van belang. De taal die in dit onderzoek wordt gebruikt is ANSI C. C is een snelle programmeertaal en daarom erg geschikt voor ANN's. Deze zijn immers complex en kennen een lange rekentijd. Tevens is C een makkelijk te begrijpen taal, ook voor mensen zonder programmeerkennis.

De getoonde broncode is ontwikkeld en getest in Visual Studio 2013 op Windows 8.1 maar is geschreven in ANSI C en zal dus ook werken op andere *compilers* en platformen. Dit werk valt onder de GPL v3 licentie.

Delen van de broncode zijn niet relevant voor dit onderzoek en worden niet behandeld in dit hoofdstuk. Om de volledige werking van het programma in te zien kan men de broncode in bijlage 1 raadplegen. Verwijzingen naar elementen in codeblokken zijn `gefixeerd` gedrukt.

## 3.2 Structs definiëren

Computers slaan informatie op met behulp van datatypes. Datatypes zijn een vorm van abstractie die aangeeft op welke manier de onderliggende informatie moet worden geïnterpreteerd. Een datatype kan bijvoorbeeld aangeven dat de onderliggende data een getal is. Een ander datatype geeft aan dat het om een stuk tekst gaat, enzovoorts. Om de data in het programma te rangschikken worden *structs* opgebouwd die als datatypes fungeren. Een *struct* is eigenlijk een combinatie van bestaande datatypes die samen iets zeggen over een specifiek onderdeel van het programma. In het programma worden in ieder geval de volgende *structs* gedefinieerd.

1. Axon: met daarin *weight* en verwijzingen naar de twee verbonden neuronen.
2. Neuron: met daarin linker en rechter axonen, de interne waarde en de delta.
3. *Layer*: een verzameling neuronen.
4. Net: een verzameling *layers*.

De axon *struct* ziet er als volgt uit.

```
struct axon {
     struct neuron* left;
     struct neuron* right;
     double weight;
};
```

Zoals je ziet heeft de *struct* drie 'onderdelen', deze worden *members* genoemd. De eerste heet `left` en refereert naar het *left-connected* neuron. In die broncode geeft het sterretje (*) aan dat het om een verwijzing gaat naar een bestaand neuron. Voor `right` geldt hetzelfde, deze refereert aan het *right-connected* neuron. Op de laatste lijn in de *struct* wordt de `weight` met het datatype `double` toegevoegd. `Double` is het datatype dat wordt gebruikt voor decimale getallen met hoge precisie. Omdat de verandering in de *weight* vaak maar erg subtiel is (zie rekenvoorbeeld 2) is deze hoge precisie vereist.

Vervolgens wordt de neuron *struct* gedefinieerd:

```
struct neuron {
     llist_t laxons;
     llist_t raxons;
     double x;
};
```

De eerste twee *members* zijn *linked lists*. *Linked lists* kunnen een onbepaald aantal verwijzingen vasthouden naar andere objecten. In dit geval zullen allebei de *linked lists* gevuld worden met verwijzingen naar axonen. De `laxons` *member* slaat alle verwijzingen op naar axonen die het neuron verbindt met *left-connected* neuronen en `raxons` doet hetzelfde voor de *right-connected* neuronen. De `double x` slaat de waarde van het neuron op nadat deze is verwerkt door de *activation function*.

De *layer struct*:

```
struct layer {
     int num_neurons;
     struct neuron* neurons;
};
```

Hierin wordt het aantal neuronen in de *layer* opgeslagen in `num_neurons`. Deze *member* is van het datatype `int`, wat staat voor *integer*. Een *integer* slaat hele getallen op. De neuronen zelf worden opgeslagen in `neurons`. Hier betekent het sterretje een verwijzing naar meerdere neuronen, in tegenstelling tot de `left` en `right` *members* in de axon *struct*, die maar naar één neuron verwijzen.

De *struct* die gebruikt wordt voor het ANN zelf wordt `net` genoemd:

```
typedef struct net {
     int num_layers;
     struct layer* layers;
     int num_axons;
     struct axon* axons;
     struct net_mode* mode;
} net_t;
```

Zoals te zien is, is deze *struct* omgeven door `typedef` en `net_t`. Deze combinatie zorgt ervoor dat dit datatype kan worden aangeroepen met `net_t` zonder er `struct` voor te schrijven. Net zoals in de *layer struct* is hier een collectie `layers` met aantal `num_layers` en eveneens een collectie `axons` met aantal `num_axons`. Als laatste is hier ook nog een *member* `mode` toegevoegd.

De *struct* `net_mode` is als volgt gedefinieerd:

```
struct net_mode {
     activ_func_t activ_func;
     activ_deriv_func_t activ_deriv_func;
};
```

De twee *members* zijn verwijzingen naar functies. `activ_func` verwijst naar de *activation function* die moet worden gebruikt in het netwerk en `activ_deriv_func` naar de bijbehorende afgeleide.

### 3.3 Sigmoid function implementeren

De implementaties voor de *sigmoid function*, die eerder in hoofdstuk 2 is besproken, zien er als volgt uit:

```
double sigmoid_func (double x)
{
     return ( (double) 1.0 / (1.0 + exp (-x) ) );
}

double sigmoid_deriv_func (double y)
{
     return (y * (1.0 - y) );
}
```

De functie `sigmoid_func` heeft één argument `x`. Deze waarde wordt doorgegeven aan de functie `exp`. Dit is een standaardfunctie die de $x^{de}$ macht van *e* teruggeeft. Deze kan worden ingevuld in de *sigmoid function* (zie formule 3).

De afgeleide van de *sigmoid function* is een functie van zichzelf. De variabele `y` is de output van een neuron. De functie geeft vervolgens de uitkomst terug van de afgeleide van de *sigmoid function* (te vinden in formule 6).

Met deze lijn wordt een variabele `net_mode_sigmoid` van het datatype `struct net_mode` gedefinieerd die kan worden gebruikt om het netwerk in te stellen.

```
static struct net_mode net_mode_sigmoid =
     { &sigmoid_func, &sigmoid_deriv_func };
```

### 3.4 Random number generator

Een belangrijk onderdeel van de broncode is de *random number generator*. Deze term wordt gebruikt om het systeem aan te duiden dat willekeurige getallen levert. Deze getallen worden gebruikt bij het initialiseren van het netwerk voor de *weights* van de axonen.

Om de *random number generator* te initialiseren wordt de functie `net_init_rnd` aangeroepen met de minimale en maximale waarde van de te genereren willekeurige getallen. Om een goede balans

te vinden tussen de getallen wordt als minimale waarde 0 en als maximale waarde positief 1 gebruikt. Zodoende kan `net_init_rnd` op de volgende manier worden aangeroepen:

```
net_init_rnd (0.0, 1.0);
```

De implementatie van deze functie is onbelangrijk. De interne functies van het ANN kunnen met behulp van de functie `net__rnd` een willekeurig getal verkrijgen. De *weight* van een axon wordt dus op de volgende manier toegewezen:

```
axon->weight = net__rnd ();
```

### 3.5 Initialiseren van het netwerk

Een nieuw netwerk wordt gecreëerd met de functie `net_create`. Deze geeft een referentie terug naar een variabele van het type `net_t`. Deze variabele wordt in het vervolg meegeven aan functies die aanpassingen doen aan het netwerk. `net_create` heeft een onbepaald aantal argumenten gelijk aan *n + 1* waarin *n* het aantal *layers* is. Het eerste argument is het aantal *layers*, de daaropvolgende argumenten zijn het aantal neuronen in de betreffende *layer*.

Voor het initialiseren van een netwerk zoals in fig. 3 (met respectievelijk 2, 4 en 1 neuronen) wordt `net_create` op de volgende manier aangeroepen:

```
net_t* net = net_create (3, 2, 4, 1);
```

`net_create` heeft als taken:

1. Geheugenruimte reserveren voor alle *layers*, neuronen en axonen en het netwerk zelf.
2. Het verbinden van alle neuronen en daarmee dus het vullen van de `left` en `right` *members* in `struct axon` en de *members* `raxons` en `laxons` in `struct neuron`.
3. Het toewijzen van *weights* voor alle axonen. Dus het vullen van de *member* `weight` in `struct axon` met een willekeurige waarde.

De broncode die wordt gebruikt voor stap 1 is te vinden in de implementatie van `net_create`. Stap 2 en 3 worden uitgevoerd door de interne functie `net__connect_all` (welke wordt aangeroepen door `net_create`).

Verder moet de `net_mode` worden toegewezen. Hiervoor wordt de `net_mode_sigmoid` waaraan eerder de implementaties van de *sigmoid function* en de bijbehorende afgeleide zijn toegewezen gebruikt:

```
net->mode = &net_mode_sigmoid;
```

De ruimte die door `net_create` is gereserveerd moet aan het einde van het programma weer worden vrijgegeven. Dat wordt gedaan met `net_free`:

```
net_free (net);
```

## 3.6 Functies voor probleemoplossingsfase

Voor het invullen van de *patterns* is de functie `net_set_input` gegeven. Deze functie verwacht een referentie naar een blok met *n* aantal *doubles* waarin *n* gelijk is aan het aantal neuronen in de *input layer*. Deze waardes worden gekopieerd naar de *member* `x` in de *structs* van de inputneuronen.

Om een netwerk zoals te zien in fig. 3 te initialiseren met dezelfde *patterns* als in rekenvoorbeeld 1 (respectievelijk 0.64 en 0.40) wordt `net_set_input` als volgt aangeroepen:

```
double in[2] = { 0.64, 0.40 };
net_set_input (net, in);
```

De eerste lijn maakt een blok met 2 *doubles* en geeft ze de waardes 0.64 en 0.40. Op de lijn daarna wordt `net_set_input` aangeroepen met als eerste argument `net`, omdat er een aanpassing op het dat netwerk wordt gedaan. Het tweede argument is het blok met de twee *patterns*.

Nu kan `net_run` worden aanroepen door de gegevens in het netwerk te verwerken en de benodigde berekeningen uit te voeren om tot de outputwaardes te komen:

```
net_run (net);
```

De implementatie van `net_run` berekent de nieuwe `x` van elk afzonderlijk neuron op de volgende manier:

```
double activation = 0;
ll_node_t* n;

/* Hieronder begint de for-constructie. */
for (n = neuron->laxons.head; n != NULL; n = n->next) {
     struct axon* axon = (struct axon*) n->v;

     activation += axon->left->x * axon->weight;
}

neuron->x = net->mode->activ_func (activation);
```

Op de eerste lijn wordt een variabele `activation` aangemaakt die als waarde nul heeft. De lijn daarna dient als voorbereiding voor de `for`-constructie. Het begin van de `for`-constructie wordt

aangegeven met een *comment* (in het groen). *Comments* dienen altijd als toelichting op de code en tellen niet mee voor het programma.

De `for`-constructie voert de lijnen tussen de twee accolades uit voor elke referentie naar een `struct axon` in de *linked list* `laxons` van het huidige neuron. Binnen de accolades wordt op de eerste plaats een referentie verkregen naar de huidige axon. Verder wordt de waarde van het linker neuron vermenigvuldigd met de *weight* van de axon (`axon->left->x * axon->weight`). De uitkomst hiervan wordt opgeteld bij de `activation` met behulp van het `+=` teken.

Dit geheel is de implementatie van de *activation function* en levert de *weight sum* zoals te zien is in formule 1.

De implementatie van de *activation function* is afhankelijk van de `net_mode` die is toegewezen bij het initialiseren van het netwerk. Het aanroepen van `activ_func` gebeurt op de laatste lijn (`net->mode->activ_func (activation)`). De uitkomst hiervan wordt meteen toegewezen aan de `x` *member* van het neuron.

Dit proces wordt aangeroepen voor elk neuron afzonderlijk. Wanneer dit gebeurd is kunnen de outputwaardes worden gevonden in de outputneuronen. Met behulp van `net_get_output` kunnen deze waardes als volgt worden verkregen:

```
double out[1];
net_get_output (net, out);
```

De eerste lijn maakt een blok met één waarde, omdat het netwerk maar één outputneuron heeft. `Net_get_output` kopieert de outputwaardes naar het blok `out`.

### 3.7 Voorbeeldprogramma voor probleemoplossingsfase

Met behulp van de hiervoor besproken functies en *structs* kan een voorbeeldprogramma worden gemaakt op basis van rekenvoorbeeld 1.

Het programma bestaat uit 5 bestanden:

1. `net.c`: Implementatie van alle interne en externe functies met betrekking tot het ANN.
2. `net.h`: *Header file*: Lijst van functies uit `net.c` die openbaar zijn voor andere broncode bestanden.
3. `sigmoid.c`: Implementatie van sigmoid function en afgeleide.
4. `sigmoid.h`: Header bestand.
5. `main.c`: Code van het voorbeeldprogramma.

Het bestand `main.c` bestaat uit een functie `main` die dient als aanvangsfunctie. Deze wordt uitgevoerd wanneer het programma wordt gestart. De code hierin wendt alle functies uit `net.h` en `sigmoid.h` aan om een ANN te maken.

```
1  #include "net.h"
   #include "sigmoid.h"

   int main (int argc, char* argv[])
   {
        net_t* net;

2       double in[2] = { 0.64, 0.40 };
        double out[1];

3       net_init_rnd (-1.0, 1.0);

4       net = net_create (3, 2, 4, 1);
        net->mode = &net_mode_sigmoid;

5       net->axons[1].weight = 0.25;
        net->axons[5].weight = 0.50;

6       net_set_in (net, in);
        net_run (net);
        net_get_out (net, out);

7       printf ("%f\n", net->layers[1].neurons[1].x);
        getchar ();
        net_free (net);

        return 0;
   }
```

1. De bovenste twee lijnen geven aan dat gebruik wordt gemaakt van de functies uit de bestanden `net.h` en `sigmoid.h` van welke de implementaties zijn te vinden in de bijbehorende *source files* `net.c` en `sigmoid.c`.
2. De `in` en `out` blokken behoren bovenaan in de `main` functie. De inputwaardes zijn dezelfde als die uit rekenvoorbeeld 1.
3. De *random number generator* wordt geïnitialiseerd met minimaal -1 en maximaal 1.
4. Het netwerk met drie *layers* van respectievelijk twee, vier en één neuron(en) wordt gecreëerd en het netwerk wordt ingesteld om de *sigmoid function* te gebruiken.
5. De *weights* worden veranderd zodat ze kloppen met die uit rekenvoorbeeld 1.
6. De inputwaardes worden ingevoerd, het netwerk wordt gedraaid en de outputwaardes worden verkregen.
7. De `x` waarde van het tweede neuron uit de tweede *layer* wordt op het scherm geprint. Hiervoor wordt de functie `printf` gebruikt. De functie `getchar` dient hier om het programma te stoppen totdat er op een toets wordt gedrukt. Tenslotte wordt het netwerk verwijderd.

Wanneer dit programma wordt uitgevoerd print het de waarde van het tweede neuron uit de tweede *layer* van het netwerk op het scherm. Deze is in rekenvoorbeeld 1 al berekend als 0.59. De output van het programma is 0.589040 (zie fig. 8). Dit klopt dus met het rekenvoorbeeld. De uitkomst van het programma is preciezer en bovendien sneller dan handmatig rekenen.

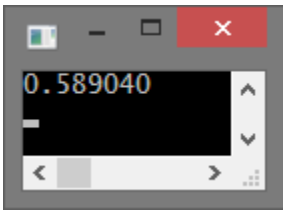



*fig 8. Output scherm van het programma.*

De uitkomsten die zijn berekend met behulp van dit programma zeggen nog niets. Ze zijn immers uit een netwerk voortgekomen dat is geïnitialiseerd met willekeurige waardes. Om tot zinnige uitkomsten te komen moet het programma worden uitgebreid met de leerfase.

## 3.8 Broncode aanpassen voor de leerfase

Voor de leerfase moet er allereerst een nieuw *member* worden toegevoegd aan `struct neuron`. Hierin wordt de deltawaarde opgeslagen. Het datatype van de nieuwe *member* is `double`. De aangepast neuron *struct* ziet er zo uit:

```
struct neuron {
     llist_t laxons;
     llist_t raxons;
     double x;
     double delta;
};
```

Ook is het nodig dat er een *member* wordt toegevoegd aan `struct axon` waarin het *weight* verschil kan worden opgeslagen om te gebruiken voor het momentum. De axon *struct* verandert als volgt:

```
struct axon {
     struct neuron* left;
     struct neuron* right;
     double weight;
     double weight_d;
};
```

## 3.9 Functies voor de leerfase

Het is belangrijk dat de voortgang van het netwerk kan worden bepaald tijdens de training. Hiervoor wordt de *sum-of-squares error function* (formule 4) gebruikt. Deze wordt berekend door voor elk neuron de *error* te berekenen en deze bij elkaar op te tellen. De *error* wordt zo berekend:

```
error += 0.5 * pow (target[i] - outlayer->neurons[i].x, 2);
```

De functie `pow` wordt gebruikt voor het kwadrateren. De eerste variabele die wordt meegegeven is de target output min de echte output van het huidige neuron. Het getal `2`, in de tweede variabele, geeft aan dat de waarde tot de macht twee wordt verheven. De uitkomst hiervan wordt vermenigvuldigd met een half en opgeteld bij het totaal. De functie die dit alles implementeert is

`net_error`. Deze functie zal veelvuldig worden gebruikt voor het meten van de effectiviteit van het netwerk.

Ten tweede is een functie nodig die het *backpropagation* algoritme implementeert en het netwerk aanpast aan de hand van gegeven *target outputs*. Deze functie is `net_adjust` genoemd. Behalve een referentie naar de netwerk *struct* neemt deze functie eveneens een referentie naar de *target outputs*, de *learning rate* en het momentum. De functienaam met zijn argumenten, ook wel het prototype van de functie genoemd, ziet er dus als volgt uit:

```
int net_adjust (net_t* net, double* target, double lr, double mom);
```

Allereerst loopt deze functie door alle neuronen in de *output layer* en berekent de deltawaardes zoals in formule 7:

```
neuron->delta = (target[j] - neuron->x) *
                net->mode->activ_deriv_func (neuron->x);
```

Zoals te zien is, wordt het verschil tussen de *target output* en de echte output vermenigvuldigd met de afgeleide van de *activation function*. De implementatie van deze functie is eerder in het hoofdstuk ingesteld op de *sigmoid function*. Voor neuronen in de *hidden layers* wordt de delta zo berekend:

```
double deltas = 0;
ll_node_t* n;

for (n = neuron->raxons.head; n != NULL; n = n->next) {
     struct axon* axon = (struct axon*) n->v;

     deltas += (axon->right->delta * axon->weight);
}

neuron->delta = deltas * net->mode->activ_deriv_func (neuron->x);
```

Opnieuw wordt hier de for-constructie gebruikt. Deze keer worden alle gewogen delta's van de *right-connected* neuronen bij elkaar opgeteld in `deltas`. De uitkomst wordt vermenigvuldigd met de uitkomst van de afgeleide van de *activation function* zoals in form 7 en opgeslagen in het neuron. Deze waardes worden gebruikt bij het uitvoeren van het *backpropagation* algoritme.

De laatste stap is het aanpassen van de *weights*. Alle axonen in het netwerk worden doorlopen en aangepast met de volgende code:

```
double weight_d = (lr * axon->right->delta * axon->left->x) +
                  (mom * axon->weight_d);

axon->weight_d = weight_d;
axon->weight += axon->weight_d;
```

Een tijdelijke variabele `weight_d` wordt aangemaakt om de uitkomst van het algoritme erin op te slaan. De *learning rate* wordt vermenigvuldigd met de delta van het rechter neuron, en dat wordt weer vermigvuldigd met de output van het linker neuron. Hierbij wordt de vorige verandering in deze axon, vermenigvuldigd met het momentum, opgeteld. Dit alles zoals in formule 8.

De verandering in de *weight* wordt opgeslagen in het axon voor de volgende berekening. Tenslotte wordt de verandering doorgevoerd in het axon.

## 3.10 Trainen van het netwerk

Met de aanpassingen die zijn gedaan aan de broncode kan het netwerk nu tevens getraind worden. In dit programma wordt een simpel netwerk getraind om de sinus te berekenen van een waarde. Om het netwerk te trainen zijn *patterns* nodig. Omdat dit netwerk één input- en één outputneuron heeft bestaat een *pattern* simpelweg uit twee waardes. De *struct* wordt gedefinieerd als volgt:

```
struct pattern {
     double x;
     double y;
};
```

Een aparte functie `create_patterns` wordt gemaakt dat een blok met `n` aantal *patterns* maakt. In deze functie worden de individuele *patterns* als volgt geïnitialiseerd:

```
patterns[i].x = ( (double) rand () / RAND_MAX);
patterns[i].y = sin (patterns[i].x);
```

De input `x` wordt gezet op een willekeurige waarde tussen nul en één. De outputwaarde wordt gezet op de sinus van de zonet toegewezen input. Op deze manier is de outputwaarde altijd de perfecte uitkomst van de inputwaarde. Wanneer het netwerk zal worden getraind, zal het netwerk op den duur het verband herkennen tussen `x` en `y` en gelijke resultaten leveren.

Na het trainen zal de functie `evaluate` tien tests uitvoeren op het netwerk. De resultaten hiervan worden op het scherm geprint. Met deze gegevens kan een conclusie worden getrokken over de kwaliteit van het netwerk.

De functie `evaluate` wordt als volgt geïmplementeerd:

```
void evaluate (net_t* net)
{
     int i;

1    struct pattern tests[10] = {
           { 0.20       , sin (tests[0].x) },
           { 0.7        , sin (tests[1].x) },
           { 0.123      , sin (tests[2].x) },
           { 0.5050     , sin (tests[3].x) },
           { 0.32       , sin (tests[4].x) },
           { 0.23423    , sin (tests[5].x) },
           { 0.234      , sin (tests[6].x) },
           { 0.98       , sin (tests[7].x) },
           { 0.235345   , sin (tests[8].x) },
           { 0.34534645 , sin (tests[9].x) },
     };

2    for (i = 0; i < 10; i++) {
           double out;

3          net_set_in (net, &tests[i].x);
           net_run (net);
           net_get_out (net, &out);

           printf (" [ Evaluatie ] >> Input            : %f\n", tests[i].x);
4          printf ("               >> Output           : %f\n", out);
           printf ("               >> Verwachte output : %f\n", tests[i].y);
           printf ("               >> Verschil         : %f (lager is
beter)\n", fabs (tests[i].y - out) );
           printf ("\n");
     }
}
```

1. Tien *patterns* worden gemaakt. Links staat telkens de input en rechts de verwachte output.
2. De *patterns* worden alle tien doorlopen met een `for`-constructie.
3. De inputwaarde van het huidige *pattern* wordt in het netwerk ingevoerd. Het netwerk wordt gedraaid en de output opgeslagen in `out`.
4. De resultaten worden op het scherm geprint. Daaronder vallen de inputwaarde, de outputwaarde, de verwachte outputwaarde en het verschil tussen de echte en de verwachte outputwaarde. De `fabs` functie die wordt gebruikt bij het printen van het verschil zorgt ervoor dat de waarde altijd positief is.

De benodigde functies voor het programma zijn gereed. Hier volgt de code van het gehele programma:

```
1  #include <math.h>

   #include "net.h"
   #include "sigmoid.h"

2  #define NUM_PATTERNS   100000
   #define NUM_EPOCHS     10

   struct pattern {
        double x;
        double y;
   };

   /* create_patterns () en evaluate () behoren hier maar zijn uitgevoegd. */

   int main (int argc, char* argv[])
   {
        int i, j;
3       struct pattern* ptrns = create_patterns (NUM_PATTERNS);

        net_t* net;
4       net_init_rnd (0, 0.01);
        net = net_create (3, 1, 100, 1);
        net->mode = &net_mode_sigmoid;

5       for (i = 0; i < NUM_EPOCHS; i++) {
              double net_err = 0;

              for (j = 0; j < NUM_PATTERNS; j++) {
                    net_set_in (net, &ptrns[j].x);
6                   net_run (net);
                    net_err += net_error (net, &ptrns[j].y);

7                   net_adjust (net, &ptrns[j].y, 0.28, 0.04);
              }

8             printf ("  [ Training ] >> Gemiddelde sum-of-squares error :
   %f\n", (net_err / NUM_PATTERNS) );
        }

9       evaluate (net);

10      net_free (net);
        free (ptrns);

        getchar ();
        return 0;
   }
```

1. De *header file* `math.h` is nodig voor de wiskundige functies die worden gebruikt in het programma zoals `fabs` en `sin`.
2. Hier worden twee constanten in het leven geroepen. Op elke plek in de code wordt de constante vervangen voor zijn bijbehorende waarde. `NUM_PATTERNS` is het aantal *patterns* dat zal worden gebruikt voor de training. `NUM_EPOCHS` staat voor het aantal *epochs* die zullen worden gedraaid. In een *epoch* worden alle *patterns* getraind. De *patterns* worden meerdere keren getraind omdat dat de kwaliteit van het netwerk doet verbeteren.
3. De *patterns* worden gemaakt met `create_patterns`.
4. Het netwerk wordt geïnitialiseerd. De *random number generator* waardes worden op 0 en 0.01 gezet. De verlaagde waarde van het maximum houdt verband met de grootte van het netwerk. Wanneer een netwerk veel neuronen in een laag heeft moeten de beginwaardes van de *weights* met dezelfde factor worden verlaagd. Anders zal de *weighted sum* (formule 1) erg hoog uitvallen en wordt het netwerk ontrainbaar.
5. Met behulp van een for-constructie worden de *epochs* gedraaid. Het wordt bepaald door de waarde toegewezen aan `NUM_EPOCHS`.
6. Elk *pattern* wordt doorlopen en getraind. Eerst wordt de inputwaarde van het *pattern* in het netwerk gekopieerd met `net_set_in`, vervolgens wordt het netwerk gedraaid. De huidige *error* wordt opgeteld bij het totaal voor later.
7. Dan wordt `net_adjust` gebruikt om de training uit te voeren. De *learning rate* is 0.28 en het momentum 0.04. Deze waardes zijn experimenteel als beste uit de bus gekomen.
8. De totale *error* wordt gedeeld door het aantal *patterns* om de gemiddelde *error* van deze *epoch* te verkrijgen. De uitkomst wordt geprint op het scherm. Hieraan kan worden afgelezen of het netwerk beter wordt.
9. `evaluate` wordt aangeroepen om het netwerk te testen.
10. Geeft geheugenruimte van het netwerk en de *patterns* vrij.

De exacte output van het programma is te vinden in bijlage 2.

## 3.11 Trainingsresultaten interpreteren

De gemiddelde *error* wordt berekend over elke *epoch*. In fig. 9 is het verloop van de *error* te zien. De daling van de *error* is erg snel in het begin en wordt steeds langzamer. De reden hiervoor is dat er in de beginfase het leren erg gemakkelijk is door de grote hoeveelheid aan nieuwe informatie. Wanneer het netwerk *patterns* tegenkomt die vergelijkbaar of hetzelfde zijn kan het hier minder van leren. Hierdoor wordt het steeds moeilijker om de *error* te verlagen.

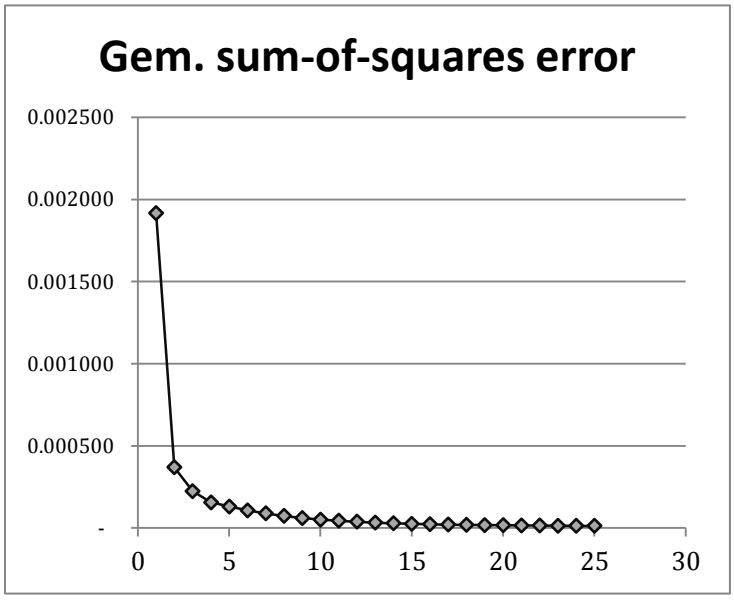



*fig 9. Verloop van gemiddelde sum-of-squares error tijdens leren.*

| Inputwaarde | Outputwaarde | Verwachte outputwaarde | Verschil (lager is beter) |
|---|---|---|---|
| 0.200000 | 0.198232 | 0.198669 | 0.000437 |
| 0.700000 | 0.650722 | 0.644218 | 0.006505 |
| 0.123000 | 0.117329 | 0.122690 | 0.005361 |
| 0.505000 | 0.487443 | 0.483807 | 0.003635 |
| 0.320000 | 0.318880 | 0.314567 | 0.004313 |
| 0.234230 | 0.234092 | 0.232094 | 0.001998 |
| 0.234000 | 0.233855 | 0.231870 | 0.001984 |
| 0.980000 | 0.825675 | 0.830497 | 0.004822 |
| 0.235345 | 0.235242 | 0.233178 | 0.002063 |
| 0.345346 | 0.342756 | 0.338523 | 0.004233 |

*fig 10. Resultaten van evaluatie in de* `evaluate` *functie van het voorbeeldprogramma.*

In fig. 10 zijn de resultaten te zien van de evaluatie functie (`evaluate`) in het voorbeeldprogramma. De outputwaarde van het netwerk en de verwachte outputwaarde worden vergeleken. Hiermee is te zien hoe dicht het netwerk bij het goede antwoord zat. Alle testen behalen een verschil van 0.01 of lager. De test met inputwaarde 0.2 behaalt zelfs een verschil lager dan 0.001. Bij elkaar duurde het trainen en evalueren in dit programma 43.04 seconde.

ANN's zijn niet geschikt voor het berekenen van wiskundige functies. Het netwerk zal in de hidden layers de inputwaardes in een bepaald bereik indelen en op basis daarvan een schatting van de outputwaarde maken. Dit geldt voor alle problemen die met behulp van een beperkte set regels kunnen worden opgelost. De resultaten die het ANN behaalt op de sinusfunctie zijn beperkt en komen niet in de buurt van de precisie die de ingebouwde functie zelf behaalt.

## 3.12 Bias

Bovenop de traditionele neuronen kan een ANN worden uitgebreid met *biases*. Een *bias* is een neuron dat als outputwaarde altijd 1.0 heeft. De *bias* is *right-connected* met alle neuronen uit een *layer*. Een *layer* heeft maar één *bias* nodig en zal dan betere prestaties leveren.

*Biases* werken omdat ze een horizontale translatie uitvoeren op de *activation function* zoals in fig. 11. De *weights* tussen de *bias* en zijn neuronen worden meegetraind en de translatie bereikt een optimum die het leerproces stimuleert.

De `net_create` functie is zo aangepast dat aan de eerste variabele `NET_BIASED` als optie kan worden meegeven. Wanneer dit wordt gedaan wordt voor elke *hidden layer* een *bias* worden aangemaakt. *Biases* hebben geen effect op input en output *layers*.

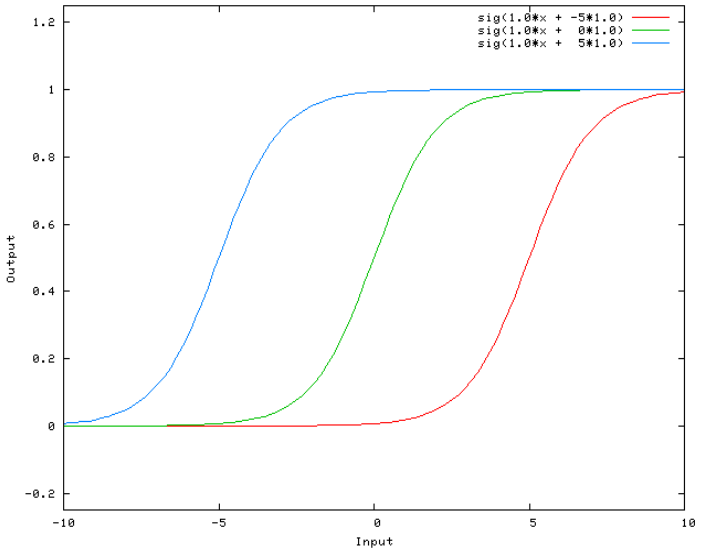



*fig 11. Effect van een bias op de activation function.*

## 3.13 Incremental-learning vs. batch-learning

Het netwerk dat is geïmplementeerd in dit hoofdstuk past *incremental-learning* toe. Bij deze methode worden de *weights* direct veranderd nadat het netwerk een enkele *pattern* heeft verwerkt. Een alternatief is *batch-learning*. Hierbij wordt eerst een hele *epoch* gedraaid. De veranderingen in de *weights* worden bij elkaar opgeteld en aan het eind gemiddeld. Dan pas worden ze toegepast op het netwerk.

*Batch-learning* heeft als voordeel dat het soms lokale minima overslaat. Lokale minima kunnen ontstaan wanneer er meerdere minima in de grafiek van een *error function* (zoals in fig. 12) zijn te vinden. Het netwerk kan dan afdalen in één van de hogere minima en minder accurate resultaten produceren na de training. Niet alle *incremental-learning* netwerken vervallen in een lokaal minima en net zo slaan niet alle *batch-learning* netwerken de lokale minima over. De kans dat ze dit doen is echter wel groter.

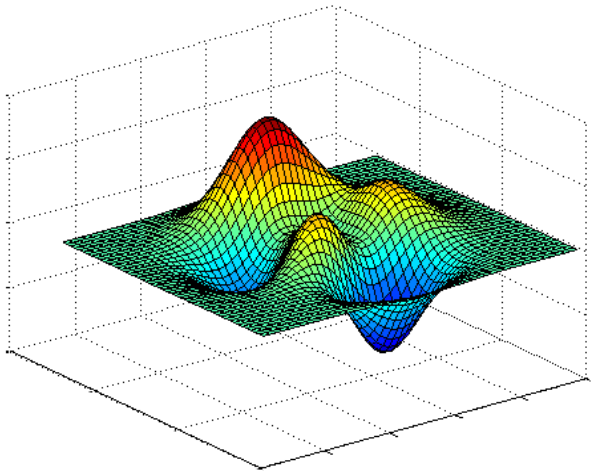


*fig 11. Grafiek van error function met lokale minima.*

*Batch-learning* heeft het nadeel dat het erg langzaam werkt. Ook kan het verkeerd kiezen van het aantal *patterns* in een *batch* nadelige gevolgen hebben voor het netwerk. Om die redenen is in dit onderzoek gekozen voor de toepassing van *incremental-learning*.

# 4. Criminelen herkennen met ANN's

## 4.1 Plan van aanpak

Het doel van dit hoofdstuk is om te onderzoeken of het mogelijk is om met behulp van een ANN op basis van gezichtskenmerken vast te stellen of iemand een crimineel is. Om dit te bewerkstelligen zijn een aantal dingen nodig:

- Een groot aantal foto's van gezichten. Van elk van deze gezichten moet bekend zijn of de persoon een crimineel is of niet.
- Het fotovoorbereidingsprogramma: een programma dat gezichtskenmerken uit foto's kan extraheren. Deze informatie wordt gebruikt als input voor het ANN.
- Het trainingsprogramma: dit programma traint het ANN om criminelen te herkennen op basis van gezichtskenmerken.
- Het testprogramma: wanneer het ANN voldoende getraind is zal dit programma de mogelijkheid bieden om gezichtsfoto's te testen op criminaliteit.

## 4.2 Het verzamelen van fotomateriaal

Op het moment dat gevangenen in de Verenigde Staten worden verwelkomt in hun gevangenis worden ze gefotografeerd. Deze foto's worden publiekelijk opgeslagen op het internet (sheriff.org) en zijn vrij toegankelijk voor iedereen. Omdat alle foto's frontaal worden gemaakt zijn ze erg geschikt om te gebruiken voor het netwerk. Met behulp van een script worden deze foto's van het internet opgehaald en opgeslagen. Dit levert 28807 frontale gezichtsfoto's van criminelen op.

Een kanttekening bij deze foto's is dat ze worden gemaakt van elke persoon die de gevangenis in of uit gaat. Hieronder vallen dus ook mensen die in voorarrest of voorlopige hechtenis zijn. Deze mensen zijn nog niet veroordeeld en worden misschien onschuldig bevonden. ANN's hebben de eigenschap dat ze kunnen omgaan met een kleine foutmarge in de input en het zal dus geen onoverkomelijk obstakel vormen voor de leerfase.

Foto's van niet-criminelen worden op dezelfde manier opgehaald van een andere website. Deze website (thatsmyface.com) biedt mensen de mogelijkheid om hun gezicht op een actiefiguurtje te laten drukken. De foto's die mensen insturen moeten verplicht frontaal worden genomen om het figuurtje te laten lukken. Vervolgens worden deze foto's publiekelijk op het internet geplaatst. Deze bron levert 22511 foto's op.

Bij deze gezichten kan echter niet met zekerheid worden gezegd of de persoon een crimineel is of niet. Er moet dus (net als in de criminele fotoset) genoegen worden genomen met een kleine foutmarge in het leerproces van het netwerk.

De foto's worden in mappen opgeslagen met de volgende structuur:

```
Mappen --.
         |
         |--- dataset faces       : Bevat 22511 JPEG bestanden van 00001.jpg tot
         |                          22511.jpg. De gezichtsfoto's van thatsmyface
         |                          .com.
         |
         `--- dataset inmates     : Bevat 28807 JPEG bestanden van t0001000000.jpg
                                    tot t0001029999.jpg. Deze gezichtsfoto's van
                                    sheriff.org met gevangenen.
```

## 4.3 Gezichtskenmerken extraheren

In hoofdstuk 1 worden (in het voorbeeld van handschriftherkenning) de grijswaardes van pixels als input voor het netwerk gebruikt. Maar omdat de te analyseren foto's groter zijn van formaat wordt het netwerk te langzaam met deze methode, dan moeten immers alle pixels worden verwerkt. Het ANN wordt in plaats daarvan getraind met *facial landmarks*. *Facial landmarks* zijn belangrijke punten op het gezicht die bij iedereen verschillen.

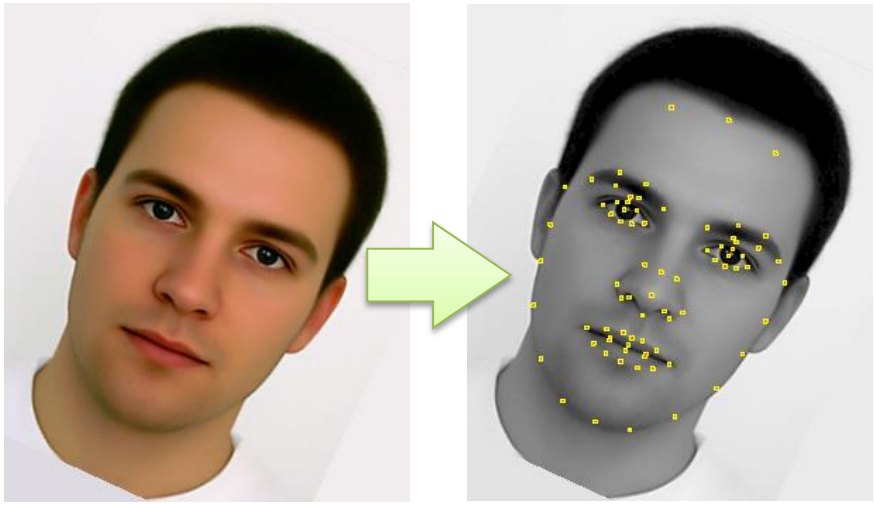
Het fotovoorbereidingsprogramma dat wordt ontwikkeld heeft de taak om deze *facial landmarks* te vinden. Ik heb de code daarvoor niet zelf geschreven. Er wordt gebruik gemaakt van een extern programma met de naam Stasm dat 77 *facial landmarks* kan vinden op een gezichtsfoto.

Het fotovoorbereidingsprogramma gaat als volgt te werk:

1. Het lokaliseert de 77 *facial landmarks* met behulp van Stasm.
2. Het normaliseert de verkregen coördinaten.
3. Het slaat de resulterende coördinaten op.

In de eerste stap moeten de *facial landmarks* worden gevonden. Voordat een gezichtsfoto aan Stasm wordt overgedragen wordt die zwart-wit gemaakt. De resulterende foto bevat geen kleurinformatie meer en is daarom makkelijker te verwerken voor Stasm. In fig. 12 is te zien waar Stasm 77 *facial landmarks* lokaliseert op dit voorbeeldgezicht.

*fig 12. Stasm heeft alle 77 facial landmarks gevonden in dit voorbeeldgezicht.*



Het voorbeeldgezicht in fig. 12 is gedraaid. Sommige gezichten in de verworven fotosets zijn dat ook. Hierdoor zijn de locaties van de gevonden *landmarks* niet geheel correct. Als deze gegevens zouden worden ingevoerd in het netwerk zal het meer moeite hebben met leren. Het moet immers eerst leren dat de draaiing van het gezicht niet belangrijk is voor het resultaat. Om dit te voorkomen worden de *landmarks* eerst genormaliseerd zoals in fig. 13. De *landmarks* worden naar links gedraaid over hoek $\alpha$.

In het normalisatie proces worden de *landmarks* tevens in een 200 bij 200 pixel plaatje opgeslagen zodat de afstand van het gezicht tot de rand van de foto geen effect meer heeft op de gegevens. De verhouding tussen de breedte en hoogte van het gezicht wordt echter nooit aangepast. Dit zou er toe leiden dat het netwerk geen beslissingen meer kan nemen op basis van de breedte of hoogte van het gezicht omdat die informatie is aangepast. Het resultaat van het normalisatie proces is te zien in fig 13.

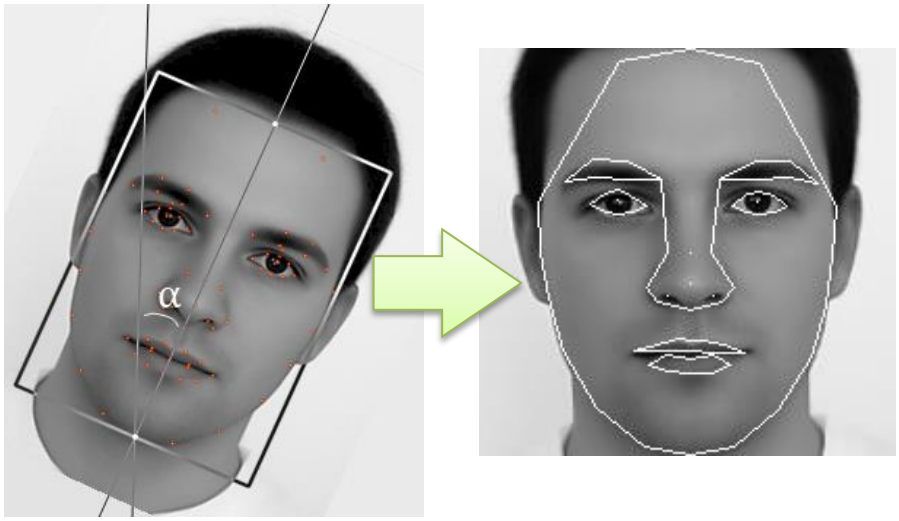



*fig 13. De facial landmarks worden genormaliseerd om het netwerk moeite te besparen.*

In de laatste stap worden alle punten opgeslagen in een bestand. Dit bestand bevat de x en y coördinaat van elk *landmark* in het gezicht. Als gevolg van de normalisatie zullen al deze waardes tussen de 0 en 200 liggen. Het programma plaatst alle bestanden in een aparte map met bijna dezelfde naam als de foto waarmee het begon. Alleen de extensie .jpg (JPEG bestand, een foto) wordt vervangen door .lms (LandMarkS-bestand).

In sommige foto's wordt geen gezicht gevonden door het programma. Deze worden simpelweg overgeslagen en niet gebruikt in de leerfase. Van de niet-criminele foto's worden 97,96% van de gezichten herkend. In de criminele set 99,16%.

Het fotovoorbereidingsprogramma levert uiteindelijk de volgende bestandsstructuur op:

```
Mappen --.
         |
         |--- dataset faces       : Bevat 22511 JPEG bestanden van 00001.jpg tot
         |                          22511.jpg. De gezichtsfoto's van thatsmyface
         |                          .com.
         |
         |--- dataset faces lms   : Bevat 22052 landmarks-bestanden van 00001.lms
         |                          tot 22511.lms (sommige missen).
         |
         |--- dataset inmates     : Bevat 28807 JPEG bestanden van t0001000000.jpg
         |                          tot t0001029999.jpg. Deze gezichtsfoto's van
         |                          sheriff.org met gevangenen.
         |
         `--- dataset inmates lms : Bevat 28565 landmarks-bestanden van t0001000000
                                    .lms tot t0001029999.lms (sommige missen).
```

## 4.4 Voorbereiden van de leerfase

In de volgende stap worden de *facial landmarks* van verworven gezichtsfoto's gebruikt voor de leerfase van het ANN. Het trainingsprogramma zal deze taak uitvoeren. Dit programma heeft als basis de code uit hoofdstuk 3. Een aantal functies worden daaraan toegevoegd.

- `f_lms_2_ps_file`: gebruikt `f_lms_read_seq_0` en `f_lms_read_seq_1` om alle *landmark* bestanden van de niet-criminelen en criminelen in het geheugen te laden. Vervolgens wordt deze informatie met behulp van `f_ps_write` opgeslagen in twee bestanden `ps0.dat` en `ps1.dat`. Nu alle *patterns* zijn verdeeld over maar twee bestanden gaat het inladen sneller.
- `f_net_save`: slaat de huidige staat van het netwerk op in `net.dat`. Zo kan het trainen op elk gewenst moment worden gestopt en later weer hervat.
- `f_net_open`: hiermee kan het netwerk in `net.dat` weer in het geheugen worden geladen.

Het trainingingsprogramma doorloopt de volgende stappen.

1. Inlezen van *patterns* uit `ps0.dat` en `ps1.dat`.
2. Proberen `net.dat` te openen. Als het bestand niet bestaat wordt een nieuw netwerk aangemaakt.
3. Het netwerk trainen.
4. Het netwerk evalueren.
5. Het netwerk opslaan in `net.dat`.

De inverse functie van `f_ps_write` (gebruikt voor het wegschrijven van de *patterns*) is `f_ps_read`. Hiermee worden de *patterns* in het geheugen geladen. Wanneer alle *landmarks* uit de afzonderlijke bestanden zouden worden uitgelezen, kost het inladen enkele minuten. Dankzij `f_lms_2_ps_file` duurt dit maar enkele seconden.

Hierna kan het netwerk worden aangemaakt. Als er al een bestand bestaat met de naam `net.dat`, dan wordt dit bestand gebruikt als netwerk. Op deze manier kan het programma tijdens het trainen om de zoveel tijd een backup maken van het netwerk.

Het aantal *epochs* die het netwerk draait kan worden aangepast in `NET_NUM_EPOCHS`. Dit bepaalt het aantal keer dat alle *patterns* door het netwerk worden bekeken. Het aantal *patterns* dat wordt gebruikt voor de training wordt bepaald door een factor `NET_TRAINING_SHARE`. Het aantal *patterns* gereserveerd voor de training is gelijk aan `NET_TRAINING_SHARE` keer het totaal aantal *patterns*. Tijdens het trainen wordt telkens om en om een niet-crimineel en een crimineel gebruikt.

De overgebleven *patterns* worden gebruikt voor de evaluatie. Er worden altijd evenveel evaluatie *patterns* gedraaid voor beide *classifiers*. Omdat er minder niet-criminele *landmark*-bestanden zijn dan criminele betekent dit dat een gedeelte van de criminelen ongebruikt blijft. Voordat het programma eindigt wordt het netwerk in zijn huidige staat opgeslagen in `net.dat`.

## 4.5 Netwerk optimalisatie

In het trainingsprogramma zijn een groot aantal variabelen aan te passen. Om de beste combinatie variabelen te vinden worden verschillende situaties vergeleken. Dit zijn de variabelen die kunnen worden aangepast:

- `NET_INIT_WEIGHTS_MIN`: de minimale waarde van de willekeurige getallen die worden gebruikt voor de *weights*. Deze waarde blijft nul en wordt niet aangepast.
- `NET_INIT_WEIGHTS_MAX`: de maximale waarde van de willekeurige getallen die worden gebruikt voor de beginwaardes van de *weights*. Het ideale getal hiervoor kan als volgt worden gevonden: $\frac{1}{N_{H1} \times N_{H2} \times \ldots}$. De uitkomst is afhankelijk van het aantal *hidden layers* en de hoeveelheid neuronen die erin te vinden zijn.
- `NET_NUM_LAYERS`: meerdere *layers* maken een netwerk slimmer, maar maken het ook exponentieel moeilijker voor het netwerk om te leren. Omdat dit een relatief simpel probleem is wordt maar één *hidden layer* gebruikt. Plus de *input layer* en *output layer* wordt dit dus drie *layers*.
- `NET_BIASED`: of de *hidden layer* een *bias* heeft of niet. Omdat een bias altijd een toegevoegde waarde heeft aan het netwerk
- `NET_LH1_NUM_NEURONS`: het aantal neuronen in de *hidden layer*.
- `NET_LEARNING_RATE`: De *learning rate* voor het *backpropagation* algoritme.
- `NET_MOMENTUM`: Het momentum voor het *backpropagation* algoritme.

De optimale waardes van de laatste drie variabelen zijn dus alleen te bepalen door experimenteren. Het trainingingsprogramma wordt geïnstrueerd om voor elke variabelen de resultaten te bekijken voor een domein aan waardes. Zo kan worden afgeleid wat de optimale waardes zijn. In de optimalisatie test worden vier *epochs* gedraaid. De gemiddelde *sum-of-squares error* van de laatste *epoch* en het foutpercentage van de evaluatie worden daarna opgeslagen.

De volgende variabelen worden geoptimaliseerd:

- Het *momentum*. De test wordt gedraaid voor alle waardes tussen 0.05 en 0.90 met tussenstappen van 0.05. De resultaten zijn geplot in fig. 14. Te zien is dat het foutpercentage en de *error* ongeveer gelijk oplopen. Het laagste punt op de grafiek is voor beiden 0.80. Dit is de optimale waarde van het momentum.

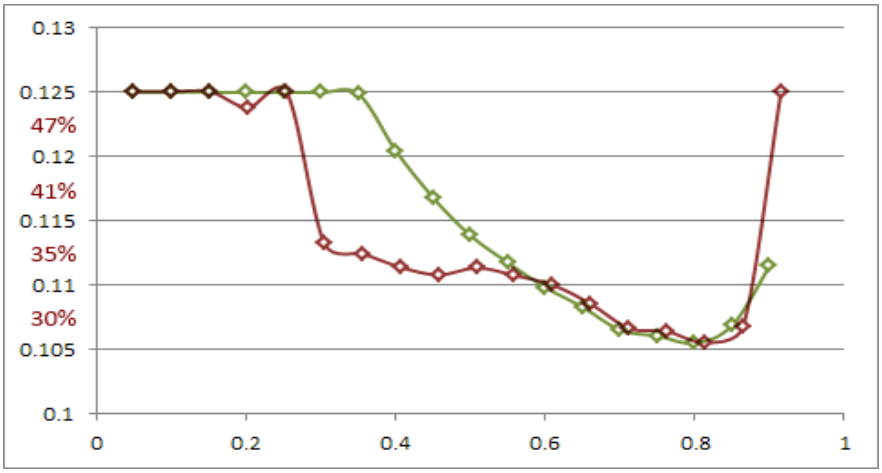



*fig 14. Foutpercentage voor momentum x / Sum-of-squares error voor momentum x.*

- De *learning rate*. Hier wordt een domein van 0.05 tot 0.50 gebruikt. Boven een *learning rate* van 0.50 kan het netwerk niet meer leren en blijft die haken op een *error* van $^1/_8$ste. Het netwerk zal dan namelijk telkens voorbij het pad naar het minimum schieten zonder ooit dichterbij te komen. Bij deze test is het zonet gevonden optimum van het momentum gebruikt. De resultaten zijn geplot in fig 15. De optimale *learning rate* is moeilijker af te lezen dan het momentum. Een lagere *learning rate* lijkt de *error* doen dalen, maar het foutpercentage stijgt dan juist. De compromis ligt op een waarde van 0.20. Daar zijn de *error* en het foutpercentage beiden relatief laag.

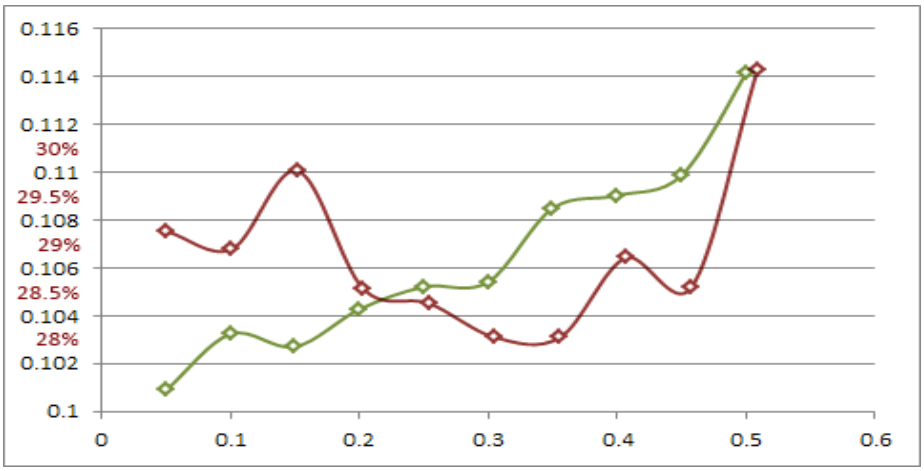



*fig 15. Foutpercentage voor learning rate x / Sum-of-squares error voor learning rate x.*

- De grootte van de *hidden layer*. De zonet gevonden optima worden gebruikt in het netwerk. De resultaten (fig. 16) zijn niet eenduidig, er is geen patroon te vinden in de grafiek. Het is daarom de vraag of de grootte van de *hidden layer* echt effect heeft op het netwerk. Toch wordt een geschikt punt van de grafiek genomen om een optimum te verkrijgen. De laagste *error* en foutpercentage zijn te vinden op 100.

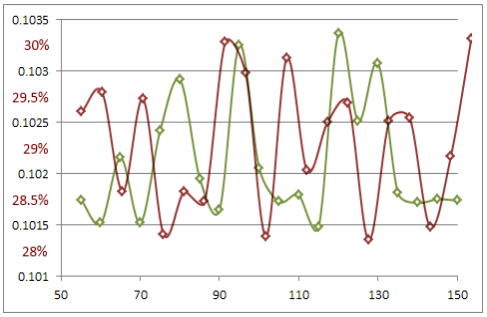



*fig 16. Foutpercentage voor hidden layer grootte x / Sum-of-squares error voor hidden layer grootte x. Er is geen bruikbaar patroon te vinden in deze grafiek.*

Samengevat, de gevonden optima zijn 0.80, 0.20 en 100 voor respectievelijk het momentum, de *learning rate* en de grootte van de *hidden layer*. Dit betekent dat het maximum van de beginwaardes van de *weights* gelijk is $^1/_{100}$.

## 4.6 Uitvoering en resultaten van de leerfase

Nadat een uitgebreide training van het netwerk is uitgevoerd, kunnen met behulp van de resultaten conclusies worden getrokken over het onderzoek. Om het netwerk zo slim mogelijk te maken moet het zo lang mogelijk worden getraind.

Uit tests blijkt dat de gemiddelde tijd voor één epoch ongeveer 20 seconde is. Om het netwerk goed te trainen worden er 200 epochs uitgevoerd. Dit duurt ongeveer een uur. Hieronder is te zien hoe de error en het foutpercentage zich ontwikkelen met het aantal epochs. (fig. 17).

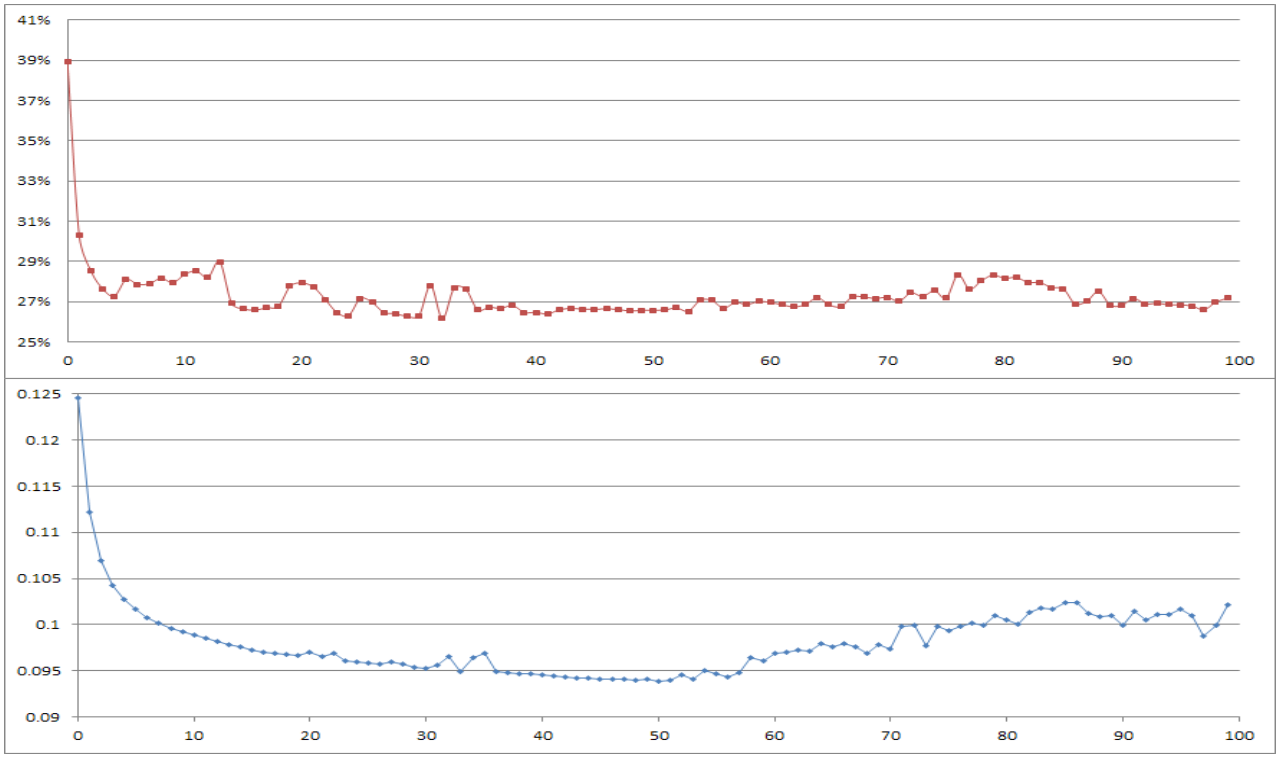



*fig 17a. Verloop van de leerfase met het aantal epochs. Te zien is de sum-of-squares error van het netwerk.*
*fig 17b. Verloop van de leerfase met het aantal epochs. Hier is het foutpercentage van de evaluatie te zien.*

Het volgende kan gezegd worden over de grafiek:

- De grootste vooruitgang wordt geboekt in de eerste 5 à 10 *epochs*. Wanneer ANN's dezelfde data heel erg vaak zien wordt het steeds moeilijker om er iets nieuws uit te leren. Daarom wordt de daling van beiden grafieken in fig. 17 steeds kleiner.
- Vanaf *epoch* 52 worden beide statistieken minder stabiel. Dit fenomeen wordt *overfitting* genoemd en treedt op bij een te lange training. Het netwerk begint dan met het uit het hoofd leren van bijzonderheden van de individuele *patterns*. Hierdoor leert het netwerk niet meer het concept van een crimineel, maar probeert te onthouden hoe alle criminelen in de trainingsfoto's eruitzagen. Omdat er te veel *patterns* zijn lukt dit niet, en gaat de kwaliteit van het netwerk omlaag.
- Precies bij *epoch* 50 is het netwerk op zijn optimum. De *error* is het laagst (fig. 17a) en *overfitting* is nog niet opgetreden. Tevens is het foutpercentage op dit punt laag en stabiel (fig. 17b). Dit netwerk wordt gekozen als het resultaat.

Het netwerk dat is gekozen als resultaat heeft de volgende exacte resultaten:

- Gemiddelde *sum-of-squares error* van *epoch*: **0.093868**
- Percentage *patterns* uit test set goed geraden: **73.423268%**

## 4.7 Resultaatnetwerk testen

Om de betrouwbaarheid van het resultaatnetwerk te testen worden enkele gezichtsfoto's van één enkele persoon gebruikt. De betrouwbaarheid wordt bepaald aan de hand van de output van het netwerk. Als er geen grootte verschillen zijn tussen de outputs van de verschillende foto's kan met zekerheid gezegd worden dat het netwerk beslissingen neemt op basis van de persoon, niet van de eigenschappen van de individuele foto.

In deze test worden 7 foto's gebruikt van voormalig minister president van de Verenigde Staten George W. Bush.

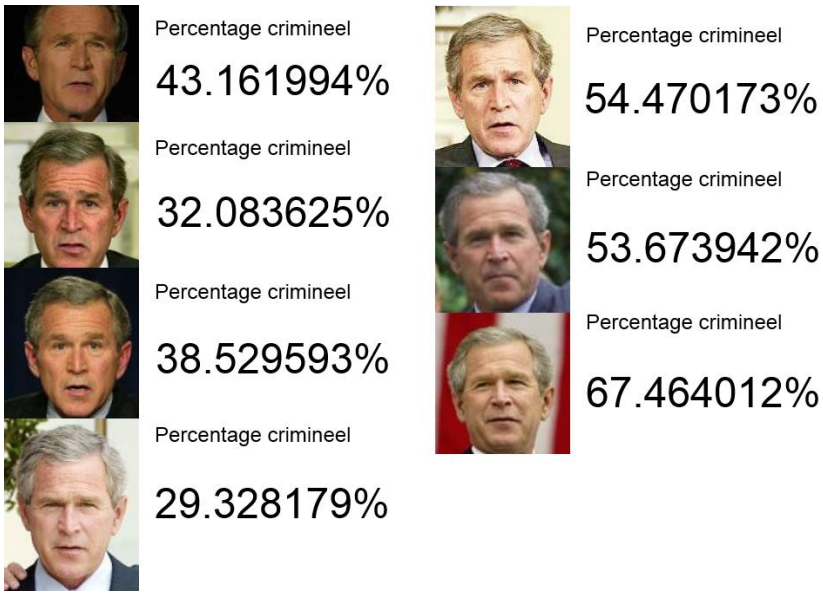



*fig 18. Zeven foto's van George W. Bush en de bijbehorende outputs van het netwerk.*

De resultaten op elk van de foto's zijn te zien in fig. 18. De waardes zijn redelijk consistent. Er is één uitzondering te vinden in de laatste foto (67,46%), deze is hoger dan de anderen. Al met al komt de heer Bush op een gemiddelde van 45,53%, nog net geen crimineel dus. Al de waardes slingeren rond dit gemiddelde.

De conclusie die hieruit getrokken kan worden is dat het netwerk niet tot willekeurige waardes komt. Wanneer eenzelfde gezicht wordt gebruikt zal het netwerk een vergelijkbare output geven. Het resultaatnetwerk voldoet aan de verwachtingen.

# Conclusie

## Interpretatie van de resultaten

In hoofdstuk 4 is een netwerk gecreëerd dat met 73% nauwkeurigheid kan zeggen (op basis van iemands gezicht) of diegene een crimineel is of niet. Dit cijfer is bepaald met behulp van 8673 test *patterns*. De kans dat deze uitkomst het product is van puur gokken is dus nihil. Het is met zekerheid te zeggen dat het netwerk in zekere mate onderscheid kan maken tussen criminelen en niet criminelen. Dit wordt ondersteund door de resultaten van de betrouwbaarheidstest (4.7).

Er is reden voor voorzichtigheid in deze conclusie, immers:

- Het netwerk kan zich baseren op de emoties van de gezichten. Iemand die op de foto moet voordat hij naar zijn cel moet heeft een andere emotie dan iemand die een foto maakt voor een actiefiguurtje.
- Het netwerk kan zich tevens hebben gebaseerd op andere eigenschappen van de foto's in de fotosets. Bijvoorbeeld: hogere differentiatie van de verticale hoek van het hoofd in de niet-criminele gezichtsfoto's, eventueel hogere percentage van verkeerd herkende gezichten in de niet-criminele foto's of effecten van de constante lichtintensiteit in de criminele foto's.
- Een fout of onregelmatigheid in het normalisatie proces van het programma dat de voorbereiding van de foto's op zich nam (zie 4.3) kan een basis vormen voor het neurale netwerk om te bepalen uit welke dataset de foto komt.

Met deze kanttekeningen in het achterhoofd, is de conclusie dat het **in zeker mate** mogelijk is om met behulp van ANN's een crimineel te herkennen op basis van een foto van zijn/haar gezicht. Hiermee is de onderzoeksvraag positief beantwoord.

## Eventueel vervolgonderzoek

Dit onderzoek kan een begin vormen voor een uitgebreider onderzoek naar het herkennen van criminelen met behulp van ANN's. Wetenschappers kunnen het netwerk beter optimaliseren en hebben toegang tot computers met meer rekenkracht.

Daarbij kunnen op een aantal punten verbeteringen worden aangebracht die niet binnen het bereik van dit onderzoek vielen:

- Onderzoeken of een ander type ANN betere resultaten oplevert (zoals *convolutional neural networks*).
- Het zoeken naar een *activation function* die voor dit probleem betere resultaten levert.
- De input van het netwerk op een andere manier aanleveren (bijv. als plaatje of met meerdere *facial landmarks*, met kleurinformatie, etc.).
- Datasets van academische kwaliteit gebruiken om de onregelmatigheden in de huidige foto's weg te nemen.

## Epiloog

De uitkomst van dit onderzoek brengt een ethisch vraagstuk met zich mee. Het resultaatnetwerk doet immers een uitspraak over de mate waarin iemand crimineel is op basis van zijn uitelijk.

Deze technologie kan op een aantal manier worden gebruikt:

- Overheden of andere instanties kunnen (preventieve) actie ondernemen tegen personen die op basis van deze test crimineel bevonden worden.
- Overheden of andere instanties kunnen bij misdaads- (of andere veiligheid gerelateerde) onderzoeken verdachten beoordelen of zelfs veroordelen op basis van deze test.
- Werkgevers kunnen sollicitanten testen met het netwerk en hun keuze daar (deels) van laten afhangen.
- In het dagelijks leven kunnen mensen hun familie, vrienden of kennissen testen met het netwerk en een (voor)oordeel vormen.

De vraag is of deze mogelijkheden wel toegestaan zijn volgens de wet. Discriminatie wordt gedefinieerd als '*het maken van ongeoorloofd onderscheid*'. Waarbij ongeoorloofd meestal wordt beschreven als: op basis van ras, uiterlijk, afkomst, seksuele geaardheid of een andere eigenschap die in de gegeven situatie niet relevant is.

In dit onderzoek wordt onderscheid gemaakt tussen criminelen en niet-criminelen op basis van uiterlijk. Uiterlijk is bij het beoordelen van criminaliteit vaak geen relevant gegeven. In essentie is het gebruik van deze methode dus een vorm van discriminatie. En dat is in Nederland verboden.

Soms wordt echter onderscheid gemaakt waarbij uiterlijk wel een relevant gegeven is. Bijvoorbeeld bij het aannemen van vliegtuigpiloten. Wanneer de sollicitant niet lang genoeg is, kan hij/zij geen piloot worden. Wanneer het ANN in de toekomst wordt verbeterd en de betrouwbaarheidsgraad stijgt zal er een moment komen wanneer uiterlijk wél als relevant gegeven kan worden beschouwd bij het herkennen van criminelen, net als bij het aannemen van piloten.

Uiteindelijk is het een belangenafweging. Kiezen we voor deze methode, dan moet de enkeling die een criminele neus heeft, maar geen crimineel is, de lasten dragen. Hij zal constant onder surveillance staan en altijd uit de rij worden gehaald bij het vliegtuig. De veiligheid in Nederland zal echter toenemen. Anderzijds kunnen we het belang van Jan met de criminele neus zwaarder wegen dan een beter veiligheidsapparaat.

Zijn we bereid de mogelijkheid van een veiliger Nederland te laten liggen voor Jan met de criminele neus?

## Dankwoord

Graag wil ik mijn begeleider, meneer Karnebeek, bedanken voor zijn ondersteuning bij het maken van dit onderzoek. Zijn expertise en begeleiding hebben mij erg geholpen.

Tevens wil ik mijn ouders bedanken. Zonder hun onmetelijke interesse in mijn vooruitgang, vele tips, het controleren van het hele verslag, en het onophoudelijke meedenken was dit onderzoek nooit een succes geweest.

## Verklarende woordenlijst

| | |
|---|---|
| *Activation function (A)* | Functie waarom de inkomende waardes in de neuronen worden omgezet naar uitgaande waardes. |
| *ANSI C* | Manier van programmeren in C waardoor de broncode gegarandeerd is te werken in andere omgevingen. |
| *Artificial Intelligence (AI)* | De wetenschap die als doel heeft dat computers intelligent gedrag vertonen. |
| *Artificial Neural Network (ANN)* | Een toepassing van AI die intelligentie kan opwekken met een manier die is afgekeken van het menselijk brein. |
| *Axon* | De verbinding tussen twee neuronen. In ANN's gaat deze verbinding gepaard met een waarde, de *weight*. |
| *Backpropagation algoritme* | Het algoritme dat een ANN op basis van zijn resultaten zo aanpast dat het de volgende keer beter presteert en zo het netwerk in staat stelt te 'leren'. |
| *Batch-learning* | Een methode waarbij het *backpropagation* algoritme pas wordt uitgevoerd nadat meerdere *patterns* zijn verwerkt. |
| *Bias* | Een neuron die als output altijd 1 heeft en doorgaans verbonden is met alle neuronen uit een andere laag. |
| *C* | De meest geschikte programmeertaal voor ANN's om zijn snelheid, naamsbekendheid en relatief makkelijk gebruik. |
| *Classifier* | Een bepaalde groep waarin de input van het ANN kan worden ingedeeld. Bijvoorbeeld: criminelen en niet-criminelen. |
| *Comment* | Gedeelte in de broncode dat de code toelicht maar verder geen effect heeft op het programma. |
| *Compiler* | Software die de broncode omzet in een programma. |
| *Controller* | De actie die wordt ondernomen wanneer input in een *classifier* is ingedeeld. |
| *Convolutional Neural Networks* | Een ander type ANN dat meer rekening houdt met lokale gegevens en beter geschikt is voor de verwerking van bijvoorbeeld foto's. |

| | |
|---|---|
| *Dataset* | Een verzameling gegevens. In dit onderzoek wordt vaak verwezen naar ofwel de niet-criminele gezichtsfoto's, ofwel de criminelen gezichtsfoto's. |
| *Datatype* | Geeft aan van welk type een variabelen in het programma is. Sommige datatypes zijn ontworpen om tekst in op te slaan, anderen getallen, enzovoorts. |
| *Delta (δ)* | De uitkomst van een tussenstap in *backpropagation* algoritme die (een benadering van) de benodigde verandering van de output van een neuron aangeeft tijdens de leerfase. |
| *Dendriet* | Uiteindes van het neuron in het menselijk brein. |
| *Double (`double`)* | Datatype dat decimale getallen met hoge precisie opslaat. |
| *Epoch* | Gedeelte van de leerfase waarin alle beschikbare *patterns* één keer door het netwerk zijn verwerkt. |
| *Error function (E)* | Functie waarvan de uitkomst een indicatie geeft van de effectiviteit van het netwerk op het verwerkte *pattern*. |
| *Error* | Uitkomst van een *error function*. |
| *Eulers constante (e)* | Constante waarvan de afgeleide gelijk is aan zichzelf. Deze wordt gebruikt in de *activation function* en komt later goed van pas bij de afleiding van het *backpropagation* algoritme. |
| *Facial landmarks* | Punten die op een gezicht worden gezet om een goed beeld te krijgen van de maten en verhoudingen. Bijvoorbeeld de punt van de neus, hoeken van de mond, de ogen, enzovoorts. |
| *Feed-forward ANN* | Een type ANN waarin de inputwaardes in het netwerk door de axonen en neuronen naar voren worden gestuwd. |
| *For-constructie (`for`)* | Constructie in programmeercode die hetzelfde stuk code meerdere keren uitvoert, elke keer met toepassing op een andere variabele. |
| *Fuzzy logic* | Logica gebaseerd op kansen. Wordt soms gebruikt in de AI. |
| *GPL v3 licentie* | Licentie die het distribueren, aansprakelijkheid, enzovoorts van de broncode bij dit onderzoek regelt. |
| *Gradient descent* | Wiskundige techniek waarmee het minimum in een multidimensionale functie kan worden gevonden. Deze wordt gebruikt bij het afleiden van het *backpropagation-algoritme*. |

| | |
|---|---|
| *Grijswaarde* | De donkerheid van een pixel, het gemiddelde van de rood-, blauw-, en groenheid van een pixel. |
| *Header file (`.h`)* | Bestand waarin de functies in de bijbehorende *source file* worden aangekondigd. |
| *Incremental-learning* | Methode waarbij het *backpropagation* algoritme wordt toegepast na elk *pattern*, wordt gebruikt in dit onderzoek. |
| *Input* | Gegevens die het netwerk in gaan bij de *input layer*. Kan ook slaan op een andere vorm van informatie dat een systeem in gaat. |
| *Inputneuron* | Neuron in een *input layer*. |
| *Integer (`int`)* | Datatype dat wordt gebruikt voor het opslaan van gehele getallen. |
| *JPEG-bestand (`.jpg`)* | Bestandstype voor foto's. |
| *Landmarks* | *zie facial landmarks* |
| *LandMarkS-bestand (`.lms`)* | Bestand dat de coördinaten bevat van de *facial landmarks* van een gezicht. |
| *Layers, Input-*<br>*Hidden-*<br>*Output-* | Onderdeel van een ANN dat neuronen bevat, die neuronen zijn altijd alleen verbonden met de neuronen uit de volgende laag. De eerste laag is de *input layer* en de laatste de *output layer*. Alle tussenliggende *layers* zijn *hidden layers*. |
| *Learning rate (η)* | Constante in het *backpropagation* algoritme dat de leersnelheid bepaalt, een te hoge waarde zorgt ervoor dat het netwerk telkens voorbij het optimum schiet. |
| *Leerconstante* | *zie learning rate* |
| *Leerfase* | Fase waarin het netwerk *patterns* wordt gevoegd en op basis daarvan het *backpropagation* algoritme toepast. |
| *Left-connected neuronen* | Verbonden neuronen die zich in de *layer* aan de linkerkant bevinden vanuit het standpunt van een neuron of axon. |
| *Linked list (`llist_t`)* | Datatype dat een onbepaald aantal verwijzingen naar andere variabelen opslaat in een programma. |
| *Member* | Variabele die onderdeel is van een *struct*. |

| | |
|---|---|
| *Momentum (α)* | Constante in het *backpropagation* algoritme dat bepaalt in welke mate de verandering in de *weight* van de vorige in acht wordt genomen bij de berekening. |
| *Netwerk* | *zie Artificial Neural Network* |
| *Neuraal netwerk* | *zie Artificial Neural Network* |
| *Neuron* | Onderdeel van een ANN dat input omzet in output met behulp van de *activation function*. |
| *Neurotransmitter* | Stofje in het brein dat de overdracht van signalen regelt tussen neuronen. |
| *Ontwikkelomgeving* | Software die wordt gebruikt voor het ontwikkelen van broncode. |
| *Output function* | *zie activation function* |
| *Output (O)* | Gegevens die het netwerk uitkomen bij de *output layer*. Kan ook slaan op de gegevens die het programma op het scherm print of elke andere soort informatie die uit een systeem komt. |
| *Outputneuron* | Neuron in een *output layer*. |
| *Overfitting* | Treedt op in een ANN na een te lange leerfase. Het netwerk probeert alle *patterns* die zijn gebruikt te onthouden voor het goede antwoord en dat lukt niet. De effectiviteit van het netwerk neemt af. |
| *Partiële afgeleide* | Afgeleide waarbij alle variabelen, behalve één, als constante worden behandeld. Deze wordt gebruikt bij het afleiden van het *backpropagation* algoritme. |
| *Pattern* | Een inputwaarde met bijbehorende *target output* die wordt ingevoerd in het netwerk tijdens de leerfase. |
| *Probleemoplossingsfase* | Fase waarin het netwerk al getraind is en gebruikt kan worden voor zijn taak. |
| *Random number generator* | Onderdeel van het programma die willekeurige getallen genereert. |
| *Receptor* | Onderdeel van het brein dat bindt aan de neurotransmitter. |
| *Right-connected neuronen* | Verbonden neuronen die zich in de *layer* aan de rechterkant bevinden vanuit het standpunt van een neuron of axon. |

| | |
|---|---|
| *Sigmoid function* | Veel gebruikt type *activation function* die ook wordt gebruikt in dit onderzoek. |
| *Source file (`.c`)* | Bestand dat de functies in de *header file* implementeert en dus de broncode bevat. |
| *Stasm* | Stuk software (ontwikkeld op basis van een wetenschappelijke studie) dat de *facial landmarks* op een gezicht vindt. |
| *Step function* | Type *activation function* dat ofwel een 0, ofwel een 1 als uitkomst heeft. |
| *Struct (`struct`)* | Datatype dat wordt gevormd door een verzameling van variabelen met verschillende datatypes. |
| *Sum-of-squares error function* | *Error function* die vaak wordt gebruikt voor ANN's. Ook in dit onderzoek wordt deze *error function* gebruikt. |
| *Synaps* | Onderdeel van het neuron in het menselijk brein dat signalen ontvangt. |
| *Target output (t)* | De verwachte output. Deze is gebundeld met de bijbehordende input in een *pattern*. |
| *Trainingsdata* | *zie patterns* |
| *Trainingsfase* | *zie leerfase* |
| *Turing-test* | Een test, verzonnen door Alan Turing, die bepaalt of een systeem intelligent is. |
| *Visual Studio 2013* | Softwarepakket waarmee de broncode in dit onderzoek is ontwikkeld, bevat o.a. een *compiler*. |

## Literatuurlijst


(sd). Opgeroepen op 1 16, 2014, van Broward Sheriff's Office: http://www.sherrif.org/

(sd). Opgeroepen op 1 16, 2014, van ThatsMyFace.com: http://thatsmyface.com/

(sd). Opgeroepen op 1 26, 2014, van VanDale: http://www.vandale.nl

*Artificial Intelligence*. (n.d.). Retrieved 12 2, 2013, from Wikipedia: http://en.wikipedia.org/wiki/Artificial_intelligence

*Artificial Neural Network.* (sd). Opgeroepen op 12 12, 2013, van Wikipedia: http://en.wikipedia.org/wiki/Artificial_neural_network

*Backpropagation.* (sd). Opgeroepen op 12 12, 2013, van Wikipedia: http://en.wikipedia.org/wiki/Backpropagation

Beeman, D. (2001). *Multi-layer perceptrons (feed-forward nets), gradient descent, and back propagation.* Opgeroepen op 12 12, 2013, van University of Colorado: http://ecee.colorado.edu/~ecen4831/lectures/NNet3.html

Çeliktutan, O., Ulukaya, S., & Sankur, B. (2013). *A comparative study of face landmarking.* EURASIP Journal on Image and Video Processing.

Cootes, T., & Taylor, C. (2004). *Statistical Models of Appearance.* Manchester: University of Manchester.

Gershenson, C. (n.d.). Artificial Neural Networks for Beginners.

Golda, A. (2005). *Principles of training multi-layer neural network using backpropagation.* Retrieved 12 12, 2013, from http://galaxy.agh.edu.pl/~vlsi/AI/backp_t_en/backprop.html

Huang, G., Ramesh, M., Berg, T., & Learned-Miller, E. (2007). *Labeled Faces in the Wild: A Database for Studying.* Amherst: University of Massachusetts.

IBM. (n.d.). *History of Deep Blue*. Retrieved 12 2, 2013, from IBM.com: http://www-03.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/

Jones, E. R. (2004). *An Introduction to Neural Networks.* Retrieved 12 12, 2013, from roguewave.com: http://www.roguewave.com/DesktopModules/Bring2mind/DMX/Download.aspx?entryid=756&command=core_download&PortalId=0&TabId=607

Leverington, D. (2009). *A Basic Introduction to Feedforward Backpropagation Neural Networks.* Opgeroepen op 12 12, 2013, van Texas Tech University: http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html

Milborrow, S., & Nicolls, F. (2014). *Active Shape Models with SIFT Descriptors and MARS.* University of Cape Town, South Africa.

National Institute of Neurological Disorders and Stroke. (sd). *Know Your Brain.* Opgeroepen op 12 12, 2013, van nih.gov: http://www.ninds.nih.gov/disorders/brain_basics/know_your_brain.htm

*Partiële afgeleiden.* (sd). Opgeroepen op 12 12, 2012, van uvt.nl: http://cdata3.uvt.nl/wiskundevoorbedrijfseconomen/node/456

Reingold, E., & Nightingale, J. (n.d.). *The Turing Test.* Retrieved 12 12, 2013, from utoronto.ca: http://psych.utoronto.ca/users/reingold/courses/ai/turing.html

Rhode, C. (sd). *Into Neural Networks.* Opgeroepen op 12 12, 2013, van Lower Columbia University: http://www.lowercolumbia.edu/classes/faculty-pages/rhodeCary/intro-neural-net.aspx

Rojas, R. (1996). The Backpropagation Algorithm. In R. Rojas, *Neural Networks.* Springer-Verlag.

*Role of Bias in Neural Networks.* (sd). Opgeroepen op 1 11, 2014, van Stackoverflow: http://stackoverflow.com/questions/2480650/role-of-bias-in-neural-networks

Sarle, W. S. (2002). *FAQ: Neural Networks.* Opgeroepen op 12 12, 2013, van sas.com: ftp://ftp.sas.com/pub/neural/FAQ.html

Shaunacy, F. (2013, 6 18). Stanford's Artificial Neural Network Is The Biggest Ever. *Popular Science*.

# Bijlagen

## 1. Volledige broncode

In verband met de grootte van alle broncode bestanden worden deze niet bijgevoegd in het schriftelijk verslag. De volledige broncode is echter publiek toegankelijk via GitHub onder de volgende link: https://github.com/gerwin3/pws-spot-criminals.

De bestanden kunnen worden teruggevonden in de bestandsstructuur die te vinden is op de webpagina. De volledige uitwerking van deze structuur is hieronder te zien:

```
pws --.
      |- LICENSE                          Licentie van de gehele broncode.
      |
      |- pws-img-prep ---.
      |                  |- data/         Map bevat benodigde gegevens voor Stasm.
      |                  |- stasm/        Map met broncode van Stasm.
      |                  |- debug.cpp     Code voor analyseren werking programma.
      |                  |- face.cpp      Inladen van foto en landmarks vinden.
      |                  |- face.h        Header-file van face.cpp.
      |                  |- main.cpp      Instructies voor het programma.
      |                  |- transform.cpp Landmarks normaliseren.
      |                  `- transform.h   Header-file van face.h.
      |
      `- pws-neural-net -.
                         |- list.c        Code voor linked list.
                         |- list.h        Header file van list.c.
                         |- main-example-1.c  Bevat broncode uit 3.10.
                         |- main.c        Instructies van het programma; zoals
                         |                testen, leren, evalueren en optimaliseren.
                         |- net.c         ANN code uit hoofdstuk 3.
                         |- net.h         Header file van net.c.
                         |- sigmoid.c     Sigmoid function implementatie.
                         `- sigmoid.h     Header file van sigmoid.h.
```

## 2. Resultaten trainingsfase

*Output tijdens training*

```
[ Training ] >> Gemiddelde sum-of-squares error : 0.001916
[ Training ] >> Gemiddelde sum-of-squares error : 0.000369
[ Training ] >> Gemiddelde sum-of-squares error : 0.000224
[ Training ] >> Gemiddelde sum-of-squares error : 0.000156
[ Training ] >> Gemiddelde sum-of-squares error : 0.000131
[ Training ] >> Gemiddelde sum-of-squares error : 0.000108
[ Training ] >> Gemiddelde sum-of-squares error : 0.000089
[ Training ] >> Gemiddelde sum-of-squares error : 0.000073
[ Training ] >> Gemiddelde sum-of-squares error : 0.000061
[ Training ] >> Gemiddelde sum-of-squares error : 0.000051
[ Training ] >> Gemiddelde sum-of-squares error : 0.000044
[ Training ] >> Gemiddelde sum-of-squares error : 0.000038
[ Training ] >> Gemiddelde sum-of-squares error : 0.000033
[ Training ] >> Gemiddelde sum-of-squares error : 0.000029
[ Training ] >> Gemiddelde sum-of-squares error : 0.000026
[ Training ] >> Gemiddelde sum-of-squares error : 0.000023
[ Training ] >> Gemiddelde sum-of-squares error : 0.000021
[ Training ] >> Gemiddelde sum-of-squares error : 0.000019
[ Training ] >> Gemiddelde sum-of-squares error : 0.000018
[ Training ] >> Gemiddelde sum-of-squares error : 0.000017
[ Training ] >> Gemiddelde sum-of-squares error : 0.000016
[ Training ] >> Gemiddelde sum-of-squares error : 0.000015
[ Training ] >> Gemiddelde sum-of-squares error : 0.000014
[ Training ] >> Gemiddelde sum-of-squares error : 0.000013
[ Training ] >> Gemiddelde sum-of-squares error : 0.000013
```

*Output van evaluatie*

```
[ Evaluatie ] >> Input           : 0.200000
              >> Output          : 0.198232
              >> Verwachte output : 0.198669
              >> Verschil        : 0.000437 (lager is beter)

[ Evaluatie ] >> Input           : 0.700000
              >> Output          : 0.650722
              >> Verwachte output : 0.644218
              >> Verschil        : 0.006505 (lager is beter)

[ Evaluatie ] >> Input           : 0.123000
              >> Output          : 0.117329
              >> Verwachte output : 0.122690
              >> Verschil        : 0.005361 (lager is beter)

[ Evaluatie ] >> Input           : 0.505000
              >> Output          : 0.487443
              >> Verwachte output : 0.483807
              >> Verschil        : 0.003635 (lager is beter)

[ Evaluatie ] >> Input           : 0.320000
              >> Output          : 0.318880
              >> Verwachte output : 0.314567
              >> Verschil        : 0.004313 (lager is beter)

[ Evaluatie ] >> Input           : 0.234230
              >> Output          : 0.234092
              >> Verwachte output : 0.232094
              >> Verschil        : 0.001998 (lager is beter)

[ Evaluatie ] >> Input           : 0.234000
              >> Output          : 0.233855
              >> Verwachte output : 0.231870
              >> Verschil        : 0.001984 (lager is beter)

[ Evaluatie ] >> Input           : 0.980000
              >> Output          : 0.825675
              >> Verwachte output : 0.830497
              >> Verschil        : 0.004822 (lager is beter)

[ Evaluatie ] >> Input           : 0.235345
              >> Output          : 0.235242
              >> Verwachte output : 0.233178
              >> Verschil        : 0.002063 (lager is beter)

[ Evaluatie ] >> Input           : 0.345346
              >> Output          : 0.342756
              >> Verwachte output : 0.338523
              >> Verschil        : 0.004233 (lager is beter)

[ Evaluatie ] >> Benodigde tijd : 43.040000 seconde
```

## 3. Logboek

**Leerling:** Gerwin van der Lugt | **Begeleider:** dhr. R.J.M. Karnebeek | **Klas:** 6B | **Sbu:** 80
**Titel:** 'Op het eerste gezicht'

| Datum | Tijd | Plaats | Verrichte werkzaamheden |
|---|---|---|---|
| 12maa13 | 1 uur | SGA | Introductie |
| 19maa13 | 2 uur | SGA | Presentaties vorig jaar |
| 3jun13 | 1 uur | Thuis | Hoofd- en deelvragen |
| 25sep13 | 3 uur | Thuis | Mediatheekopdracht |
| 4nov13 | $^1/_2$ uur | SGA | Bespreken mediatheekopdracht |
| 27nov13 | 3 uur | Thuis | Plan van Aanpak |
| 30nov13 | 2 uur | Thuis | Orienteren op AI |
| 2dec13 | 3 uur | Thuis | Begin H1 |
| 10dec13 | 5 uur | Thuis | Hoofdstuk 1 |
| 12dec13 | 8 uur | Thuis | Orienteren op ANN Wiskunde |
| 14dec13 | 4 uur | Thuis | Nog meer wiskunde |
| 17dec13 | 5 uur | Thuis | Wiskunde + stukje schrijven |
| 18dec13 | 4 uur | Thuis | Hoofdstuk 2 |
| 21dec13 | 8 uur | Thuis | Hoofdstuk 2 + Programmeren |
| 23dec13 | 2 uur | Thuis | Programmeren |
| 29dec13 | 6 uur | Thuis | Programmeren |
| 30dec13 | 4 uur | Thuis | Programmeren + Begin H3 |
| 2jan14 | 6 uur | Thuis | Hoofdstuk 3 |
| 3jan14 | 5 uur | Thuis | Hoofdstuk 3 |
| 4jan14 | $3^1/_2$ uur | Thuis | Hoofdstuk 3 |
| 6jan14 | 4 uur | Thuis | H3 Af + Orientatie op onderzoek |
| 10jan14 | 3 uur | Thuis | Fotoset (criminelen) |
| 11jan14 | 6 uur | Thuis | Fotosets (ook niet-criminelen) + imgprep |
| 15jan14 | 3 uur | Thuis | Programmeren imgprep |
| 16jan14 | 3 uur | Thuis | Netwerk in leerfase |
| 18jan14 | 8 uur | Thuis | Resultaten + H4 + programmeren |
| 22jan14 | 2 uur | Thuis | Conclusie + ethiek |
| 23jan14 | 2 uur | Thuis | Checkronde #1 |
| 24jan14 | $3^1/_2$ uur | Thuis | Opmaak + verklarende woordenlijst |
| 25jan14 | 2 uur | Thuis | Checkronde #2 |
| 27jan14 | 3 uur | Thuis | Verbeteren + spelfouten + opmaak |
| 29jan14 | 8 uur | Thuis | Bijlages klaarmaken + broncode + laatste verbeteringen |
| **TOTAAL** | **$118^1/_2$ uur** | | |