

Practicum verslag voor Neurale Netwerken

Henri van den Esker, 1015990, henri_vd_e@hotmail.com
Sebastiaan, 9121164, sebastia@ch.twi.tudelft.nl

4 juni 2003

1 Inleiding

Neurale netwerken worden vooral toegepast in gebieden waar klassieke simulatie ongewenst is, of gewoon te complex is om een model voor op te bouwen. Voor het bouwen van een model is er veel kennis nodig over het te modelleren. Het probleem kan echter te complex zijn dat een goed model nauwelijks te maken is. Ook voor simpelere modellen die goed voor simulatie gebruikt kunnen worden kan toch de keuze vallen op een simulatie met een neurale netwerk. Een neurale netwerk is, mits goed getraind, veel flexibeler dan een deterministisch model: een goed getraind neurale netwerk 'flipt' namelijk niet zo gauw bij foutieve invoer.

Voor een neurale netwerk hoef je niet heel diep in de stof te duiken om een model te maken. Invoer en gewenste uitvoerdata zijn de 2 essentiële zaken om een neurale netwerk als model te maken voor een situatie. De topologie en grootte van het neurale netwerk zijn uiteraard ook belangrijke zaken, maar redelijk eenvoudig aan te passen en te controleren als de simulatie niet succesvol is. Hier zijn natuurlijk ook vuistregels voor, maar in tegenstelling tot een klassiek model heeft een kleine misgreep vaak geen desastreuze gevolgen.

2 Probleemomschrijving

Voor dit practicum moeten we een systeem voor een trolley ontwerpen. Dit karretje wordt bestuurd door variatie in de spanning V :

- $V > 0$ ($V < 0$) correspondeert met positieve (negatieve) versnelling
- $|V|$ geeft de grootte aan van de versnelling

De parameters die bijgehouden worden zijn de huidige positie en de snelheid. Het probleem is om de trolley op de gewenste positie te krijgen.

De besturing van de trolley gaat met behulp van een tweede orde differentiaalvergelijking in een één-dimensionale ruimte. De differentiaalvergelijking van deze trolley is

$$\frac{d^2 x}{dt^2} + \frac{1}{\tau} \frac{dx}{dt} = \frac{ku}{\tau} \quad (1)$$

waarin u de stroomtoevoer is en hiermee de positie x en snelheid $\frac{dx}{dt}$ regelt.

De parameter k staat voor de maximaal te behalen snelheid van de trolley en τ bepaalt hoe snel de trolley op een nieuwe input moet reageren (de werkelijkheid is

slechts eindig, dus moet er in het model een vertraging ingebouwd worden). Deze parameters hebben wij voor het practicum opgegeven gekregen: $k := 0.4$ en $\tau := 0.045$.

De parameters zijn nauwkeurig gekozen en er moet rekening gehouden worden met de samenhang tussen deze. Een trolley die snel kan en snel reageert op het signaal u schiet z'n doel voorbij en ten gevolge van autocorrectie (feedback) blijft de trolley heen en weer slingeren rond het gewenste punt.

Het neurale netwerk van de trolley bestaat niet uit 1, maar uit 3 invoersignalen en 2 uitvoersignalen. De twee extra invoersignalen, naast het voltage, zijn de positie en de snelheid: als je niet weet waar je bent weet je immers ook niet hoe ver je moet rijden totdat je er bent. Aan de hand van bovenstaande differentiaalvergelijking en de 3 invoerwaarden (voltage, (huidige) positie en (huidige) snelheid) kunnen de nieuwe positie en snelheid berekend worden. Deze data worden gebruikt om het neurale netwerk mee te trainen.

3 Neurale netwerken

In de eerste paragraaf wordt het nut van een neuraal netwerk verklaard, waarna een type neurale controller wordt beschreven.

3.1 Algemeen

Indien men een probleem wil oplossen dan gaat het vaak om de besturing van het systeem. Voor lineaire en tijdsinvariante problemen is de oplossingstrategie goed ontwikkeld, want er is een goede mathematische fundering, bijvoorbeeld Systeemtheorie.

Indien de voorwaarden lineariteit en tijdsinvariantie los worden gelaten dan ontstaat er een situatie die mathematisch gezien niet goed is gefundeerd. Mogelijke oplossing is de introductie van de robuuste controllers, echter deze besturingen zijn afgeleid in het raamwerk van lineaire systemen en biedt ook niet altijd de oplossing

Een oplossing zou zijn een black box die zich aanpast aan de invoer, uitvoer van het systeem en de gewenste uitvoer. Dit systeem past zich aan als er meer kennis (lees toestanden) verworven wordt. Mathematisch is dit niet anders dan een mapping.

Neurale netwerken kan dan een oplossing bieden. Neurale netwerken is een netwerk van knopen die door een bepaalde topologie met elkaar verbonden zijn. Elke knoop reageert op de invoer door een uitvoer te geven en dit signaal aan te bieden aan andere knopen of de uitvoer. Door het trainen van de knopen kunnen de knopen de mapping leren met als doel dat een bepaalde invoer een bepaalde uitvoer levert.

3.2 Model-based predictive controller

Stel dat we een *plant* moeten besturen zodanig dat er geen trainingstest aanwezig is. Simpelweg omdat er geen model is, of dat het probleem niet tijdsinvariant is.

Wat kunnen we nog? We kunnen met behulp van de uitvoer van de *plant* bepalen met behulp van metingen. Zeg dat we meten y_l met $l \in \{1, \dots, k\}$ met l de l -de meting. Verder is de invoer u_l op het l -de moment.

De vraag is of we de reeks $\{y_l\}_{l=k+1}^{\infty}$ kunnen voorspellen op basis van de keuze $\{u_k\}_{l=k+1}^{\infty}$ en de geschiedenis.

We willen een kostenfunctie J introduceren en deze functie willen we minimaliseren op basis van de invoer $\{u_l\}_{l=k+1}^{\infty}$. De gevonden $u(k)$ wordt dan gebruikt als invoer voor de *plant*. Hierna wordt \tilde{y}_{k+1} bepaald. Merk op het verschil in notatie: y_{k+1} is de verwachte uitkomst en \tilde{y}_{k+1} is de echte uitkomst van de *plant*. Na gaan we u_{k+1} bepalen op basis van de geschiedenis waarbij \tilde{y}_{k+1} behoort.

Laten we het één en ander formaliseren:

- De kostenfunctie

$$J = \sum_{l=1}^H \left\{ (\tilde{y}_{k+l} - y_{k+l})^2 + \Delta u_{k-1+i}^2 \right\} \text{ met } \Delta u_l = u_{l+1} - u_l$$

Met $\{\tilde{y}_l\}_{l=k}^H$ de gewenste uitvoer, $\{y_l\}_{l=k}^H$ de verwachte basis op de rij $\{u_l\}_{l=k-1}^H$. Wellicht is de introductie van H opmerkelijk aangezien we formeel H gelijk aan ∞ moeten nemen. Echter dit zou een ondoenlijk minimaliseringprobleem opleveren. Met H bepalen we de complexiteit van het minimaliseringprobleem, aangezien we eisen dat de optimale realtime bepaald kan worden.

- De geschiedenis. Het bepalen van de $\{y_l\}_{l=k}^H$ in combinatie van $\{u_l\}_{l=k}^H$ wordt gedaan met behulp van een neurale netwerk. Dit netwerk zal de geschiedenis opslaan door middel van training. Dit neurale netwerk wordt gebruikt om op basis van de rij $\{u_l\}_{l=k}^H$ de rij $\{y_l\}_{l=k}^H$ te bepalen. Onderwijl wordt dit netwerk getraind met de invoer u_k en de uitvoer \tilde{y}_{k+1} .

Het minimaliseren van J kan op verschillende manieren worden gedaan. Met behulp van een gradiëntachtige methode of juist andere methoden die niet gebaseerd zijn op gradiëntachtige methoden. Indien een gradiëntachtige methode wordt gebruikt, dan is het mogelijk dat de partiële afgeleiden uitgedrukt worden in bijvoorbeeld *backpropagated network errors*, zie [D.A. Hoskins, 1992]. In gradiënt vrije methoden is het noodzakelijk dat er afgeleiden worden bepaald, dit is fijn als het bepalen van de afgeleide moeilijk is.

4 Opzet

In deze paragraaf vertellen we hoe we ons onderzoek hebben gedaan.

4.1 Theorie

Voor het trainen van een netwerk heb je een trainingset nodig. Dit is een verzameling met invoerdata en gewenste uitvoer. In het geval van de trolley is de invoer de huidige snelheid en positie en het voltage. De uitvoer zijn alleen de snelheid en positie. Deze data worden gegenereerd met behulp van de differentiaalvergelijking (1).

De axonen van het neurale netwerk hebben in het begin random waarden (gewichten). Wanneer je bepaalde waarden uit de trainingset op de invoer van het netwerk zet krijg je, zeer waarschijnlijk, een ongewenste uitvoer. Het verschil tussen de gewenste en ongewenste uitvoer is de fout die gecorrigeerd moet worden door de gewichten van de axonen in het netwerk aan te passen.

Voor een netwerk met 1 laag kan dit vrij direct gedaan worden door elk gewicht naar proportie te verminderen of vermeerderen met de fout die zich voordoet. Als het netwerk uit meerdere lagen bestaat dan wordt de fout voor elke laag geschaald.

Vervolgens wordt dan per laag de gewichten naar proportie aangepast (aan de hand van de geschaalde fout). Dit algoritme heet 'backpropagation'.

Wanneer dit vaak genoeg gebeurt zullen de gewichten zich stabiliseren (en daarmee ook het netwerk). In dat geval is het netwerk getraind en geeft elke invoer bij benadering de gewenste uitvoer. De benadering wordt vaak gerepresenteerd als de kleinste kwadratenfout van de oplossingen. Als \tilde{y} de gewenste uitvoer is en y de uitvoer van het netwerk, dan is de kleinste kwadratenfout (root mean squared error, RMS):

$$\sqrt{\frac{1}{N} \sum_{i=1}^N |\tilde{y}_i - y_i|^2} \quad (2)$$

Deze fout kan gebruikt worden als indicatie voor de getraindheid van het netwerk.

Het aanpassen van de gewichten per laag wordt van achter naar voren gedaan. De gewichten in de laatste laag worden als eerste aangepast. Daarna wordt de laag die daarvoor zit aangepast, enzovoorts. In het Engels wordt dit backpropagation genoemd. Het algoritme van de backpropagation is kort als volgt weer te geven:

pre: Geef elk gewicht in het netwerk een willekeurige waarde.

fp: (*forward pass*) Kies een trainingspaar, dat wil zeggen een uitvoer en een invoer, uit de trainingset. Bereken de uitvoer van het neurale netwerk op basis van de invoer van het trainingspaar.

bp: (*backward pass*)

1. De waarde van een uitvoerneuron, zeg y , komt in het algemeen niet overeen met de gewenste uitvoer, zeg \tilde{y} , volgens de uitvoer bijbehorende bij de trainingset. De gemaakte fout is dan $\epsilon = y - \tilde{y}$. De errorgradiënt, zeg δ , wordt dan gecorrigeerd door de afgeleide van de activatiefunctie: $\delta = f'(y - \tilde{y})$. Hiermee worden de gewichten van de uitvoerneuronen aangepast.
2. Het aanpassen van de gewichten in de eerst verborgen laag (ten opzichte van de uitvoerlaag) gebeurt met behulp van de δ 's van de uitvoerneuronen en de reeds aangepaste gewichten. Hiermee kan men de inkomende gewichten van de eerste verborgen laag worden aangepast.
3. Dit kan ook toegepast worden op achtereenvolgens de 2^{de} , 3^{de} , 4^{de} , ... verborgen laag worden toegepast.

post: Ga naar stap [fp]

4.2 Onderzoek

Veel parameters beïnvloeden de werking van een neurale netwerk. Niet alleen het aantal lagen en het aantal elementen in een laag, maar ook de leerratio, momentum, grootte van de trainingset, aantal trainingen, begingewichten van het netwerk, enzovoorts. Om niet van het pad te raken in het bos hebben we maar één punt beschouwt, namelijk de topologie van het netwerk. We hebben de factoren die niets met de topologie te maken hebben ongewijzigd gelaten. Onze parameters zijn gegeven in tabel 1.

Het eerste netwerk dat we hebben geprobeerd is het netwerk waar de practicumhandleiding van uit gaat. De andere netwerken zijn experimenteel: meer lagen, minder knopen. De keuze van een netwerk is afhankelijk geweest van de prestatie van het vorige netwerk: een groter netwerk met meer lagen presteerde slechter dan het eerste netwerk. Het volgende netwerk werd dan een kleiner netwerk dan het eerste. De gebruikte netwerkconfiguraties zijn te vinden in tabel 2.

Model

| | |
|-------|--------|
| 0.4 | k |
| 0.045 | τ |

Identifier

| | |
|---------|----------------------------------|
| 0.5 | leerratio |
| 0.5 | moment |
| 0.01 | stopcriteria |
| 0.31415 | randomizer |
| 123 | omvang trainingset |
| 61500 | aantal iteraties voor 500 epochs |
| 50 | omvang controleset |
| 1230 | aantal iteraties voor contrôle |

Controler

| | |
|-------|--------------|
| 0.5 | leerratio |
| 0.5 | moment |
| 0.01 | stopcriteria |
| 0.618 | randomizer |

Setpoints

| | |
|-----|--------------------------|
| #1 | vierkant signaal, random |
| 0.2 | amplitude |
| 2 | periode |
| #2 | foutproportionaliteit |
| 5 | factor |

Tabel 1: Gebruikte parameters

| netwerk | op pagina |
|-----------|-----------|
| 3-30-2 | 9 |
| 3-13-21-2 | 10 |
| 3-5-3-8-2 | 11 |
| 3-5-8-2 | 12 |

Tabel 2: Gebruikte netwerktopologieën

5 Programatuur

Het programma dat we hebben gebruikt voor het verkrijgen van resultaten is de *Neuro Control Workbench*. De verwerking van de resultaten in een verslag is gedaan met behulp van: *TeXnicCenter*, *MikTeX*, *Linux L^AT_EX* versie, *PCXdump* en *Adobe Photoshop 7.0*. Omdat de practicumhandleiding suggereerde dat we een *486DX33* moesten gebruiken hebben we op een stoffige zolder nog naar zo'n bak gezocht. Dit is niet gelukt.

5.1 Simulatie

De simulatie is gedaan met behulp van *Neuro Control Workbench 1.00 β* , verder afgekort als *NCW*. Dit programma is geschreven door *Jacek Jarmulak*. De programatuur is pre-windoze, maar voldoet echter voor het doen van het practicum.

NCW stelt ons in staat om enkele voor enkele problemen een besturing te maken. De besturing wordt geïmplementeerd door een neurale netwerk. Het programma stelt ons in staat om een Identifier te construeren, welke bestuurd/gecorrigeerd wordt door een Controller. Beide zijn geïmplementeerd als neurale netwerken.

Met NCW is het mogelijk om één van de volgende problemen te bestuderen:

1. een lineaire continue *plant*
2. een lineaire discrete *plant*
3. een trolley
4. bioreactor
5. een water kolom
6. een *plant* behandeld in [Krijgsman, 1993], niet-lineaire discreet SISO systeem of een versimpeld titratie proces
7. geïnverteerde pendulum

5.2 Verwerking

De verwerking van de resultaten moest enigszins omslachtig gebeuren, aangezien NCW geen goede uitvoer kon geven. NCW bevat een functie om PCX te genereren, echter deze bestanden zijn niet in te lezen met Adobe Photoshop.

Wellicht ligt er nog een oude WP 5.1 op een $5^{1/4}$ floppy ergens op diezelfde stoffige zolder waar we zochten naar een 486 of dat er toch nog ergens een goede uitleg ligt over de werking van *graphics.com* die ook netjes naar een PDF-printer of één of andere virtuele printer kan afdrucken. Derhalve hebben we het internet gebruikt als de vergaarbak van alle mogelijke oplossingen. Na wat zoeken bleek PCXdump de ideale tool. PCXdump 9.32 is een tool om schermen te dumpen. Deze schermen worden dan omgezet in PCX bestanden.

De volgende stap was het bewerken van de beelden en enigszins aanpassen van de kleuren. Dit is gedaan met behulp van Adobe Photoshop 7.0 en een script binnen Photoshop. Waarna de afbeeldingen in het verslag zijn opgenomen.

Dit verslag is geschreven in L^AT_EX. Als interface is gebruikt TeXnicCenter 1 beta 6.01 met compiler MiKTeX 2.2. Een handige omgeving om schoon L^AT_EX te schrijven, met icons om eventuele vergeten commando's weer op te frissen.

6 Resultaten

We hebben de topologie van 4 netwerken bekeken en van elk netwerk de training van het netwerk, de controle hiervan en het gedrag van de simulatie. Dit laatste houdt in het opstarten en het verloop op lange termijn.

We hebben elk netwerk met een trainingset van 123 elementen 500 keer getraind. Dit is te zien in de figuren die linksboven van de pagina's 9, 10, 11 en 12 staan. In tabel 3 zijn de fouten van elk netwerk weergegeven na 500 trainingsepochs. Rechtsboven is het resultaat gegeven van de training met een nieuwe trainingset van 50 elementen. Dit om te onderzoeken in hoeverre de training geslaagd is.

Daarna hebben we het netwerk in de simulatie gestopt. Het figuur linksonder zijn de eerste stappen van de simulatie. Je ziet hier hoe de controller zich aanpast. Het figuur rechtsonder laat de simulatie op langere termijn zien.

| type: | na training | | na controle | |
|-----------|-------------|-----------|-------------|-----------|
| | MAX fout | RMS | MAX fout | RMS |
| 3-30-2 | 0.0580991 | 0.0154106 | 0.0337406 | 0.0138434 |
| 3-13-21-2 | 0.0826265 | 0.0218149 | 0.0674651 | 0.0220899 |
| 3-5-3-8-2 | 0.1346915 | 0.0401390 | 0.0780359 | 0.0280475 |
| 3-5-8-2 | 0.0942103 | 0.0315300 | 0.0790453 | 0.0345111 |

Tabel 3: Informatie over de MAXimumim fout en de kleinste kwadraten fout (RMS) na training en na controle

Ons eerste netwerk (zie pagina 9) is volgens de beschrijving van de practicum-handleiding opgezet en getraind. We zien dat de RMS beneden de 0.02 ligt. Ook de controle van de training laat zien dat dit netwerk voldoende getraind is. We zien dit aan het korte niet stijgende lijntje.

Voor de simulatie van de trolley moeten we een controller kiezen. Als aanwijzing in de practicumhandleiding staat dat deze dezelfde soort topologie moet hebben als de identifier. We hebben ervoor gekozen om deze exact hetzelfde te kiezen als de identifier. De controller is echter nog niet getraind. Dit wordt in de simulatie gedaan. Dit zien we terug in figuur 3 als slingeren (blauwe lijn). In het verdere verloop van de simulatie zien we dat de trolley netjes inspeelt op de gewenste uitvoer (met vertraging, dit is uiteraard wat wenselijker voor de passagiers dan een directe verandering waarbij de G -kracht $\rightarrow \infty$ gaat).

Het tweede netwerk bestaat uit 2 lagen en we zien aan de RMS dat deze niet significant beter, maar zelfs slechter is dan ons eerste netwerk van slechts 1 laag en 30 knopen tegenover 34 knopen in dit netwerk. De controle van deze training geeft weer dat het netwerk voldoende getraind is. Ook in de simulatie zijn verder geen bijzondere dingen te zien. Ook hier heeft de controller hetzelfde netwerk als de identifier.

Een groter netwerk gaf ons geen betere resultaten, dus hebben we ons derde netwerk klein gekozen in het aantal knopen, maar wel een extra laag toegevoegd. De RMS is twee keer zo groot als onze doelfout (want daar gingen we voor). Ook bij de controle van de training zien we dat het netwerk doorgetraind wordt. Dit is het gevolg van het feit dat het netwerk te klein is en dus te weinig vrijheidsgraden heeft.

Echter, in de simulatie is te zien dat het allemaal wel meevalt. We hebben in dit geval de controller gekozen als 3-8-3-5-2 (netwerk is 3-5-3-8-2). We zien in het opstarten van de simulatie dat de trolley zich in even korte tijd aanpast als de vorige gevallen. Ook op de langere termijn rijdt dit karretje prima.

We hebben nu gezien dat een netwerk met 3 lagen het niet beter doet dan 1 laag en dat een 3-5-3-8-2 netwerk een te grote RMS oplevert. Dit willen we combineren in een nieuwe keuze als klein 2 lagig netwerk 3-5-8-2. Helaas blijkt de RMS nog steeds redelijk groot te zijn. Maar bij de controle van de training zien we dat het netwerk wederom goed reageert.

Voor deze simulatie hebben we weer een controller gekozen identiek aan de identifier. Wat opvalt is dat de trolley deze keer behoorlijk op gang moet komen. De trolley schiet steeds door met als gevolg van slingeren. Op de lange termijn gaat het echter weer goed.

| | |
|---------|-----------------------------|
| zwart | gewenste uitvoer |
| rood | uitvoer van de <i>plant</i> |
| blauw | uitvoer van de identifier |
| cyaan | kleinste kwadraten fout |
| magenta | maximum fout |

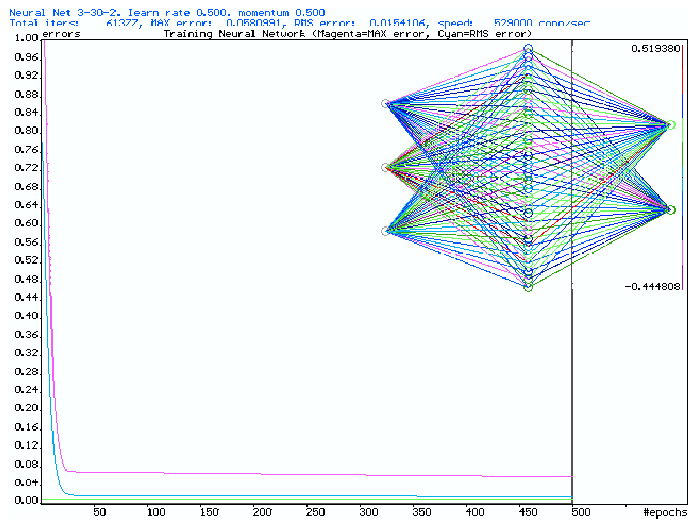
Tabel 4: Gebruikte kleuren in de plaatjes

7 Discussie

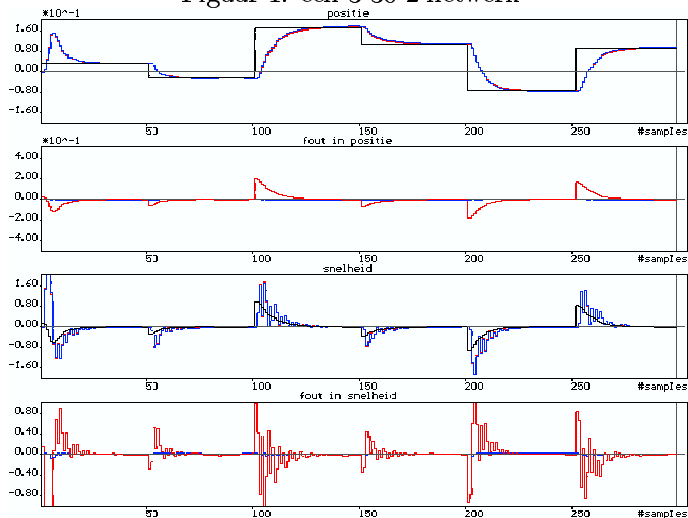
We zien dat een te groot netwerk geen significante bijdrage levert aan de training van het netwerk met oog op de RMS. Een te klein netwerk geeft wel grotere RMS. In alle gevallen is de uiteindelijke besturing van de trolley wat we wensen. De vraag rijst in hoeverre je je voor dit probleem op de RMS moet focussen. Als richtlijn van het practicum was om de RMS beneden de 0.02 te houden. Ook al is deze twee keer zo groot, de passagiers komen op hun bestemming aan.

Referenties

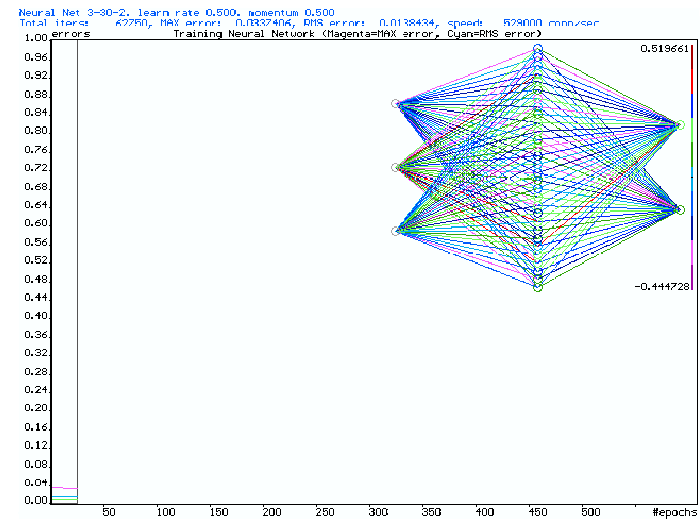
- [D.A. Hoskins, 1992] D.A. Hoskins, J.N. Hwang, J. V. (1992). *Iterative inversion of neural networks and its applications to adaptive control*. IEEE Trans. Neural Networks.
- [Jarmulak, 1994] Jarmulak, J. (1994). Neurocontrol workbench ver. 1.00 - user manual.
- [José C. Principe, 2000] José C. Principe, Neil R. Euliano, W. C. L. (2000). *Neural and Adaptive Systems*. John Wiley & Sons, Inc.
- [Krijgsman, 1993] Krijgsman, A. (1993). *Artificial Intelligence in Real-Time Control*. PhD thesis, TUDelft.



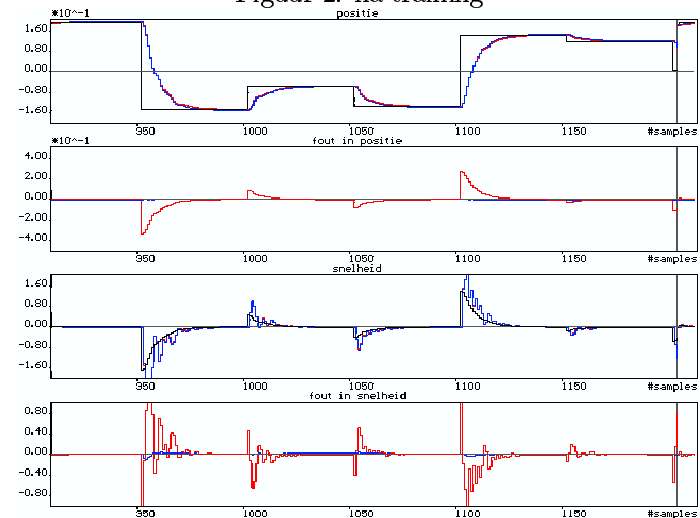
Figuur 1: een 3-30-2 netwerk



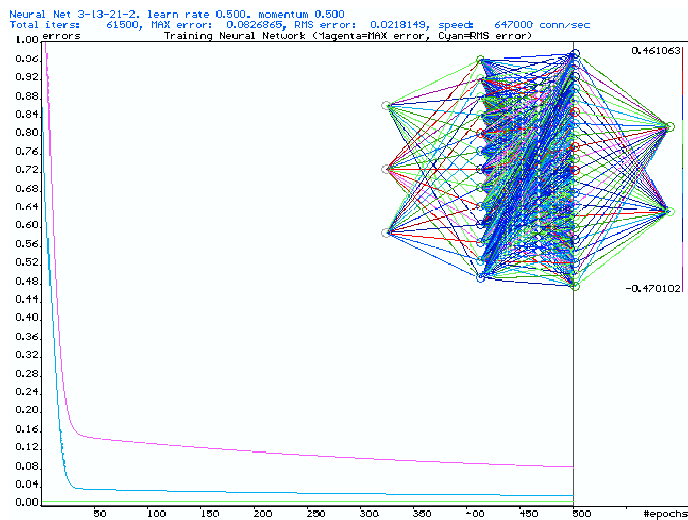
Figuur 3: begin simulatie



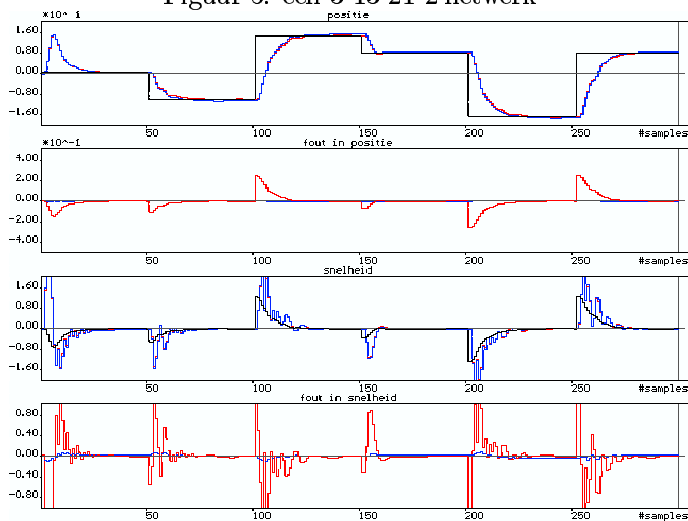
Figuur 2: na training



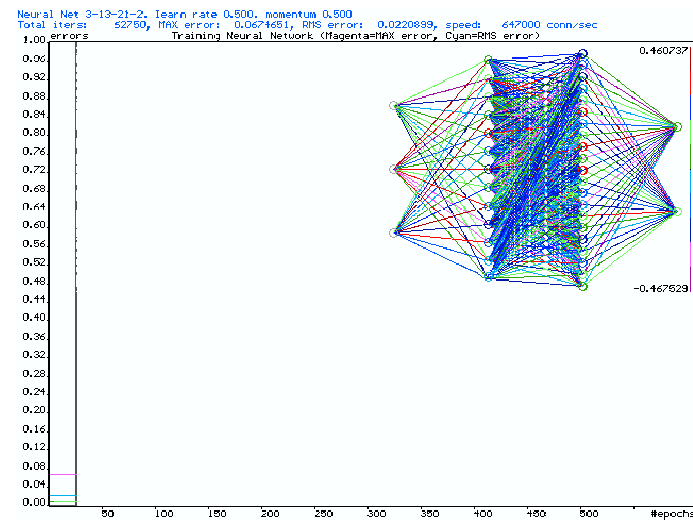
Figuur 4: verder verloop



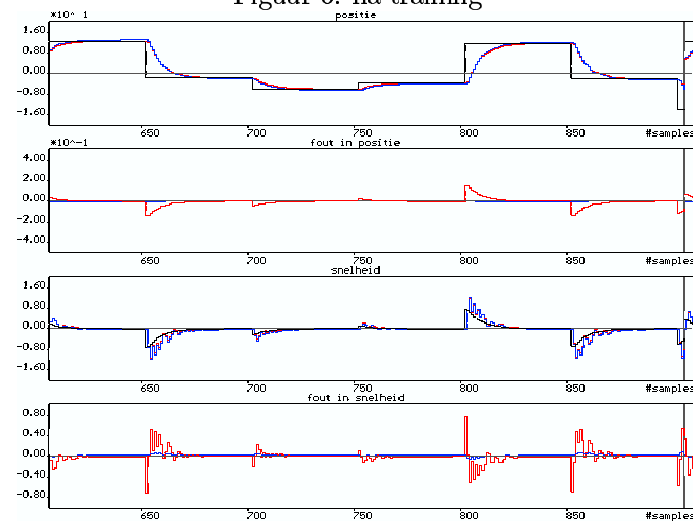
Figuur 5: een 3-13-21-2 netwerk



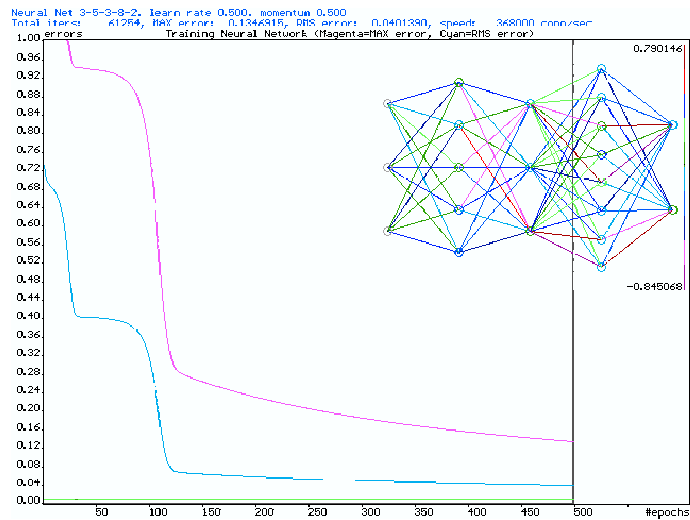
Figuur 7: begin simulatie



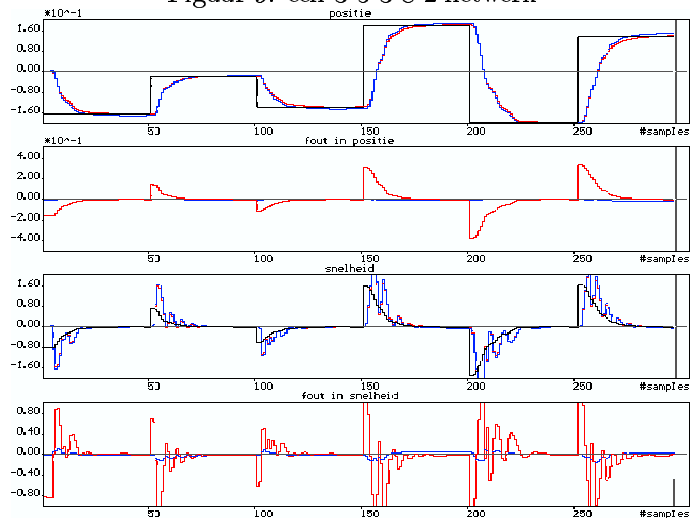
Figuur 6: na training



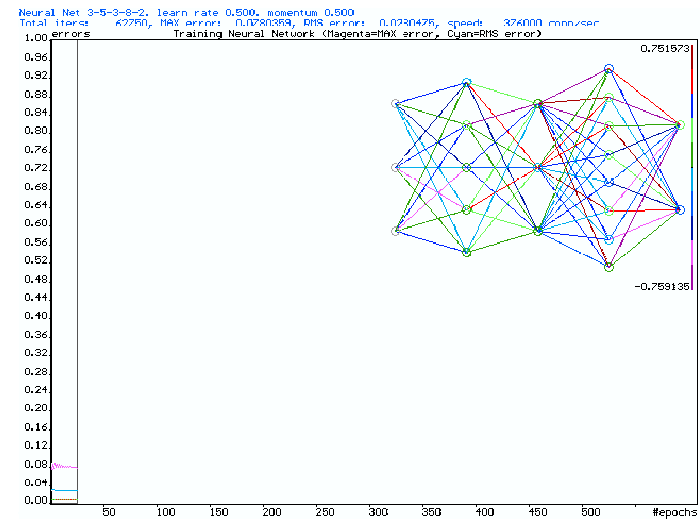
Figuur 8: verder verloop



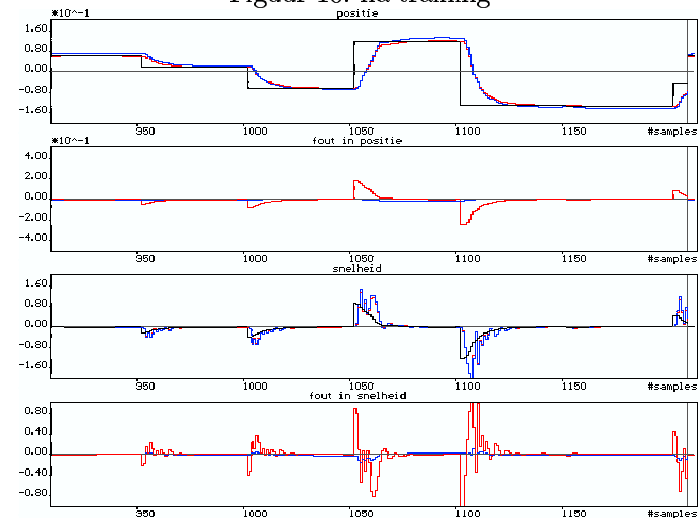
Figuur 9: een 3-5-3-8-2 netwerk



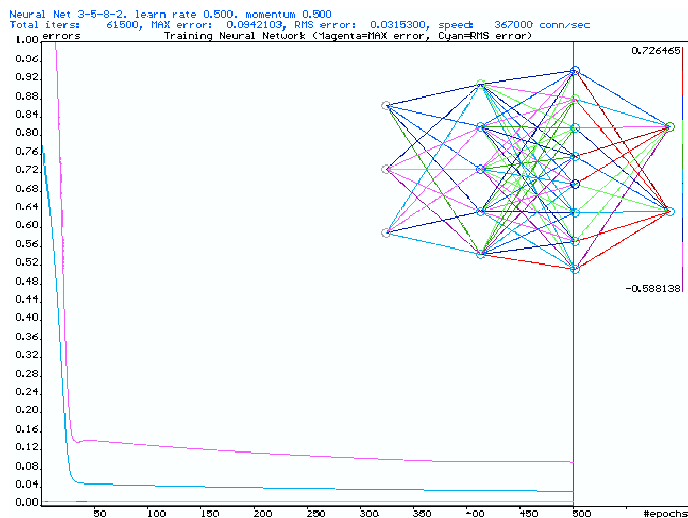
Figuur 11: begin simulatie



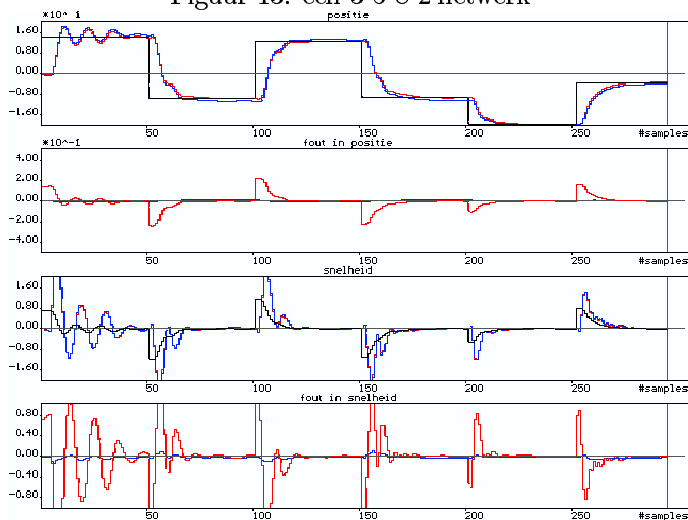
Figuur 10: na training



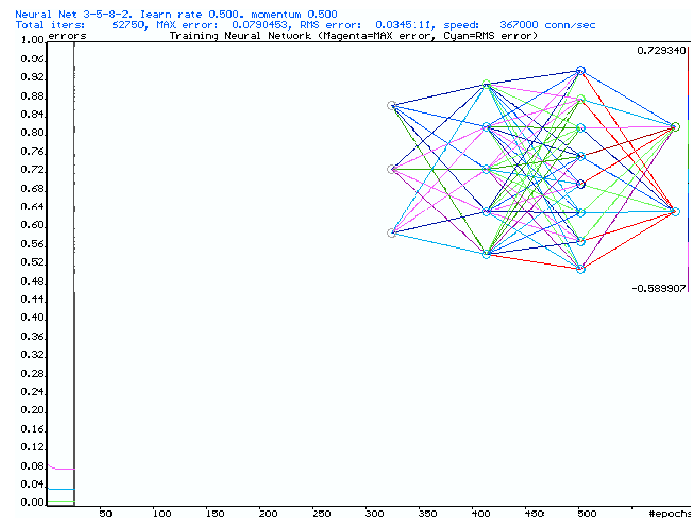
Figuur 12: verder verloop



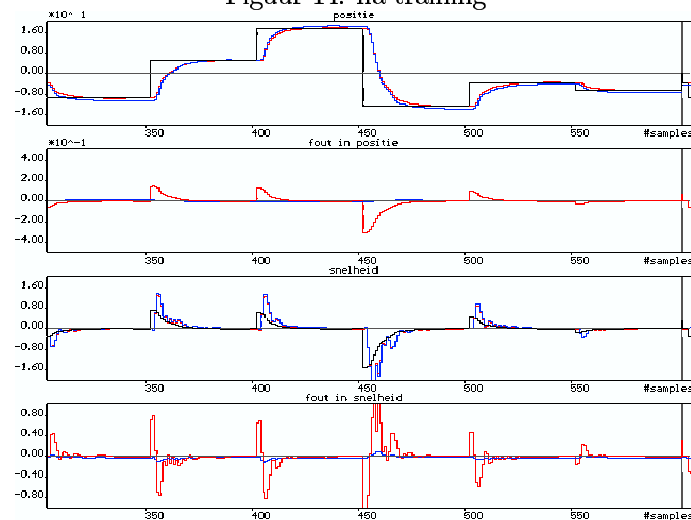
Figuur 13: een 3-5-8-2 netwerk



Figuur 15: begin simulatie



Figuur 14: na training



Figuur 16: verder verloop