Modeling Cognitive Development on Balance Scale Phenomena[4] Replicatie van Shultz et al.

Olaf Booij(AI);
9809309
obooij@science.uva.nl

29 mei 2012

1 Inleiding

Ik heb geprobeerd het onderzoek "Modeling Cognitive Development on Balance Scale Phenomena" van Thomas R Shultz, Denis Mareschal en William C Schmidt te repliceren. In dit onderzoek werd geprobeerd de cognitieve ontwikkeling, op het gebied van een balans-taakje, van een kind te simuleren met behulp van een constructief connectionistisch netwerk. Tijdens een balans-taak worden er op een vaste balans met aan beide kanten 5 pinnen, gewichten geplaatst en moet er worden gekozen of de balans aan de rechterkant naar beneden zal gaan, aan de linkerkant of in balans zal zijn. Aan beide kanten wordt telkens maar op één pin 1 tot 5 even zware gewichten geplaatst. Het wordt aangenomen dat een kind deze keuze bepaald volgens een bepaalde regel (zoals: altijd zal de balans aan de rechterkant naar beneden gaan). Tijdens zijn ontwikkeling gebruikt het kind eerst de ene regel en gaat dan over op een andere regel volgens een vast patroon van overgangen. Om dit te simuleren wordt het Cascade-Correlatie netwerk van Fahlman en Lebiere[3] gebruikt.

2 Het Cascade-Correlatie netwerk

Het Cascade-Correlatie netwerk is gebaseerd op het meerlagig feedforward netwerk, dat veel gebruikt wordt in combinatie met de back-propagatie-regel. Het belangrijkste verschil is dat het zelf het aantal hidden-knopen bepaald tijdens het leren. Het begint met het leren van de gewichten van de input-knopen (plus een bias-knoop) naar de output-knopen zonder dat

er een hidden-knoop is (de output-phase). Als blijkt dat de fout van het netwerk niet klein genoeg wordt, dan gaat het over op de input-phase. Hierin wordt een aantal kandidaat-knopen gemaakt die alleen in verbinding staan met de input-knopen (en de al aanwezige hidden-knopen). De gewichten naar deze kandidaten wordt zo geleerd dat de activatie van deze knopen zo veel mogelijk correleert met de fout van de output-knopen. Na een tijdje wordt die kandidaat-knoop uitgekozen die het meest correleert en dit wordt de nieuwe hidden-knoop. De gewichten naar deze hidden-knoop worden bevroren; ze zullen niet meer veranderen. Nu gaat de output-phase weer in en worden de gewichten naar de output-knopen weer geleerd, niet alleen degenen vanuit de input-knopen, maar nu ook die vanuit de hidden-knoop. En zo wisselen output-phase en input-phase elkaar af totdat de fout van het netwerk klein genoeg is.

Voor het leren van de gewichten wordt Quickprop gebuikt, dat afgeleid is van de standaard back-propagatie-regel[2]. Hierbij wordt niet alleen gebruik gemaakt van de vorige gewichts-verandering($\Delta w(t-1)$), maar ook van de vorige afgeleide fout per gewicht, de gradiënt ($\delta E/\delta w(t-1)$). Het foutenlandschap (per gewicht) wordt voorgesteld als dal-parabool en dus kan je de laagste waarde uitrekenen als je genoeg informatie hebt, namelijk de huidige en de vorige gradiënt en de vorige gewichts-verandering:

$$\Delta w(t) = \frac{-\delta E/\delta w(t)}{-\delta E/\delta w(t-1) + \delta E/\delta w(t)} \Delta w(t-1)$$

Een probleem ontstaat als de gradiënt in plaats van afneemt gelijk blijft of toeneemt. Volgens de formule zal de volgende gewichts-verandering dan respectievelijk oneindig groot zijn of de verkeerde kant opgaan (namelijk; tegen de richting van de gradiënt in). Dus wordt de quickprop-regel niet gebruikt als de fout niet aanzienlijk is afgenomen door de vorige gewichtsverandering, maar wordt de gewichtsverandering een zekere hoeveelheid (μ) maal de vorige gewichtsverandering. Daarbij is er ook nog een probleem als de vorige gewichtsverandering gelijk aan 0 was (met als speciaal geval het begin van de training). Om te voorkomen dat er dan nooit meer een gewichtsverandering kan optreden wordt er ook nog is een bepaalde hoeveelheid (ϵ) van de gradiënt bij opgeteld zoals bij de standaard back-propagatie-regel.

Zoals waarschijnlijk al duidelijk is zitten er in Cascade-Correlatie en Quickprop een hoop parameters, veel meer nog dan hierboven aangegeven. Shultz zegt hierover dat hij over het algemeen de default-waardes heeft gebruikt zoals aangegeven door Fahlman en Lebiere[5][4]. Deze zijn echter helemaal niet duidelijk over de waardes van de parameters en zelfs niet over de details van Cascade-Correlatie en Quickprop (bijvoorbeeld over de stop-criteria van de out- en input-phase). Dus ik heb grotendeels de parameters zelf moeten schatten, zie tabel 1. Ik heb ook veel gebruik gemaakt van een c-implementatie van Cascade-Correlatie van R. Scott Crowder uit 1992 [1] dat een port is van de originele Common Lisp implementatie van

parameter	waarde
candidates	8
max-epochs-output	200
max-epochs-input	200
patience-output	8
patience-input	8
changethreshold-output	0.015
changethreshold-input	0.03
μ-input	2.0
μ-output	2.0
ϵ-output	0.175
ϵ-input	1.0
weight-decay-output	0.0001
weight-decay-input	0.0
weight-init-range	-1.0 - 1.0
sigmoid-prime-offset	0.1

Tabel 1: Parameters voor Cascade-Correlatie zoals gebruikt voor de balans-taak.

Scott E. Fahlman, die volgens mij gebruikt wordt door Shultz (gezien noot 2 van [4]). Als benchmark heb ik het twee-spiralen-probleem gebruikt (zie figuur 1), omdat deze ook werd gebruikt door Fahlman en Liebere, zodat ik mijn uitkomst kon vergelijken met die van hen. Bij de c-implementatie stond een programmaatje om data voor het twee-spiralen-probleem te generen [1].

2.1 Stop-criteria

Shultz zegt niets expliciet over de stop-criteria, behalve dat er gestopt werd na 300 output-epochs[5][4]. Fahlman en Liebere zeggen dat als de error (of de correlatie in de input-phase) niet meer significant (changethreshold) verandert gedurende een zeker aantal epochs (patience) er gestopt wordt of als er een zeker maximaal aantal epochs (max-epochs) is bereikt[3]. Maar ze zeggen dus niks over de waardes van deze parameters zelf. Dus heb ik de waardes overgenomen uit de c-implementatie en sommigen ervan met de hand verandert tot ik goede resultaten kreeg. Sommige parameters heb ik anders ingesteld bij het twee-spiralen-probleem dan bij de balans-taak (de parameters voor de balans-taak staan in tabel 1). De changethreshold-output had ik bij het twee-spiralen-probleem op 0.005 staan en de patience-input op 30. Door het aanpassen van de parameters voor de balans-taak had het netwerk ongeveer evenveel output-epochs nodig als bij Shultz. Dit is nodig omdat het anders niet meer goed te vergelijken valt, omdat er per output-epoch een nieuw patroon bij de leerset wordt toegevoegd.

2.2 Leer-parameters

De leer-parameters zijn de parameters die gebruikt worden door Quickprop. Ten eerste heb je hier de μ, die er op één of andere manier voor moet zorgen dat de gewichts-veranderingen niet te groot kunnen worden. Volgens Fahlman zou de gewichts-verandering als het groter is dan μ keer de vorige gewichts-verandering, moeten veranderen in μ keer de vorige gewichts-verandering[2]. In de c-implementatie hebben ze dit anders gedaan: daar is het zo dat de gewichts-verandering niet groter mag zijn dan $\frac{\mu}{1+\mu}$ keer de vorige gewichts-verandering, om door (wederom) μ keer de vorige gewichts-verandering te worden vervangen. Deze laatste manier heb ik overgenomen.

Over de leer-parameter ϵ zegt Fahlman erg veel, maar hij heeft het niet over een default waarde[2]. Wel zegt hij dat hij experimenten heeft gedaan met het delen van ϵ door de "fan-in", i.e. het aantal knopen dat verbonden is met de knoop waar het gewicht heen gaat. In de c-implementatie wordt dit gebruikt bij de input-phase, maar niet bij de output-phase en dit heb ik ook zo gedaan. Ook wordt ϵ (in de c-implementatie en bij mij) gedeeld door het aantal patronen in de training-set. Dit is waarschijnlijk omdat er batch-leren wordt gebruikt en hiervoor gecompenseerd moet worden. Shultz zegt over deze parameter dat hij ze met een 1/2 heeft vermindert,[5][4] maar naar welke waarde zegt hij niet; dus daar had ik niet zoveel aan. Ik heb met de hand verandert totdat ik goede resultaten kreeg: voor het twee-spiralen-probleem een ϵ-output van 0.1 en een ϵ-input van 2.0. (Voor de balans-taak zie wederom tabel 1.)

De weight-decay is ervoor om te zorgen dat de gewichten niet heel erg groot worden (dat waarschijnlijk nogal is gebeurd bij Quickprop). Ik heb de waardes overgenomen van de c-implementatie, wat inhoudt dat er alleen weight-decay is in de output-phase.

2.3 Initialisatie

Voor het initialiseren van de gewichten heb ik het bereik (weight-init-range) genomen zoals in de c-implementatie.

2.4 Aantal kandidaten

Het aantal kandidaten dat per input-phase worden getraind en waar de beste uit wordt gekozen is volgens mij één van de belangrijkste parameters die er zijn, maar Shultz zegt hier niks over. Ik heb de waarde overgenomen van de c-implementatie.

2.5 Activatie-functie

Als activatie-functie wordt de sigmoid-functie gebruikt tussen -0.5 en 0.5:

$$\frac{1.0}{1.0 + e^{-inputsum}} - 0.5$$

De sigmoid-prime-offset is om ervoor te zorgen dat de afgeleide van de fout van een output-knoop, die wordt gebruikt voor de gewichts-verandering, niet te dicht bij nul kan komen. Dit is het geval als de output in de buurt van 0.0 of 1.0 komt, want de afgeleide van de sigmoid is $a_o(1 - a_o)$ (waarin a_o de activatie van een output-knoop is). Ik heb de waarde (0.1) overgenomen van Fahlman [2](eindelijk een waarde die wel genoemd wordt); deze werd ook gebruikt in de c-implementatie. De formule voor de afgeleide van de sigmoid wordt nu dus:

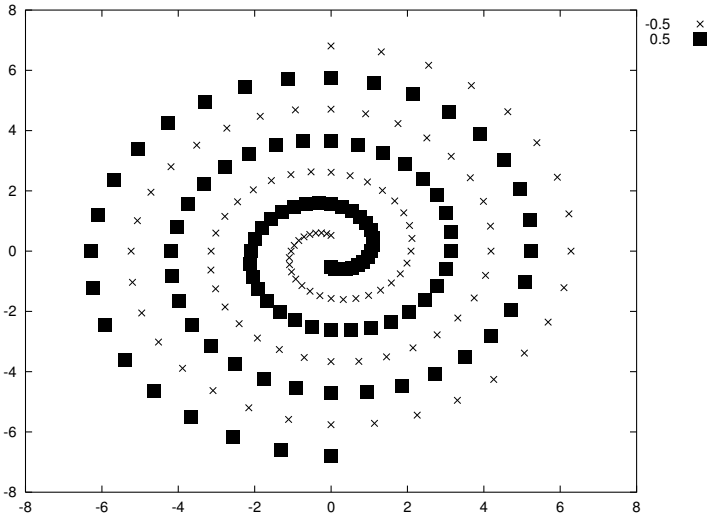
$$0.25 - activatie^2 + 0.1$$

2.6 De fout

In de output-phase is het duidelijk wat er verkleind moet worden, namelijk de "sum squared error", net als in back-propagation vaak gedaan wordt. Maar in de input-phase is dit erg onduidelijk. Shultz geeft zelf de formule voor de "correlatie"die hij in de Lips-code van Fahlman heeft gevonden[4]:

$$C = \frac{\sum_o |\sum_p (a_p - \bar{a})(e_{op} - \bar{e})|}{\sum_o \sum_p (e_{op} - \bar{e_o})^2}$$

De covarianties tussen de activatie van de kandidaat-knoop (a) en de fout bij de output-knoop (e_o) worden dus gesommeerd over alle output-knopen en gedeeld door de variantie van de fout. (Dit is niet echt een correlatie, maar goed: normaliseren is waarschijnlijk ook niet nodig.) Fahlman en Lebiere gebruiken alleen de covariantie (en delen dus door niks)[3]. In de c-implementatie gebruiken ze geloof ik wel de echte correlatie, maar ik weet dit niet zeker (de code is erg moeilijk te lezen). In elk geval gebruiken ze niet de "correlatie"van de huidige epoch maar die van de vorige epoch, om het programma sneller te laten lopen. Dit heb ik niet gedaan, maar ben gewoon van de huidige epoch uit gegaan, zoals Fahlman omschrijft[2]. Ik heb de covariantie gebruikt, omdat hiervan in [3] ook de afgeleide staat (voor de gewichtsverandering) en ik geen zin om deze zelf te bepalen. En daarbij lijkt het me ook niet zo relevant om de fout op een bepaalde manier te schalen. Wel is het zo dat dit invloed heeft op alle parameters en ik deze om die reden niet zomaar kon overnemen van de c-implementatie.



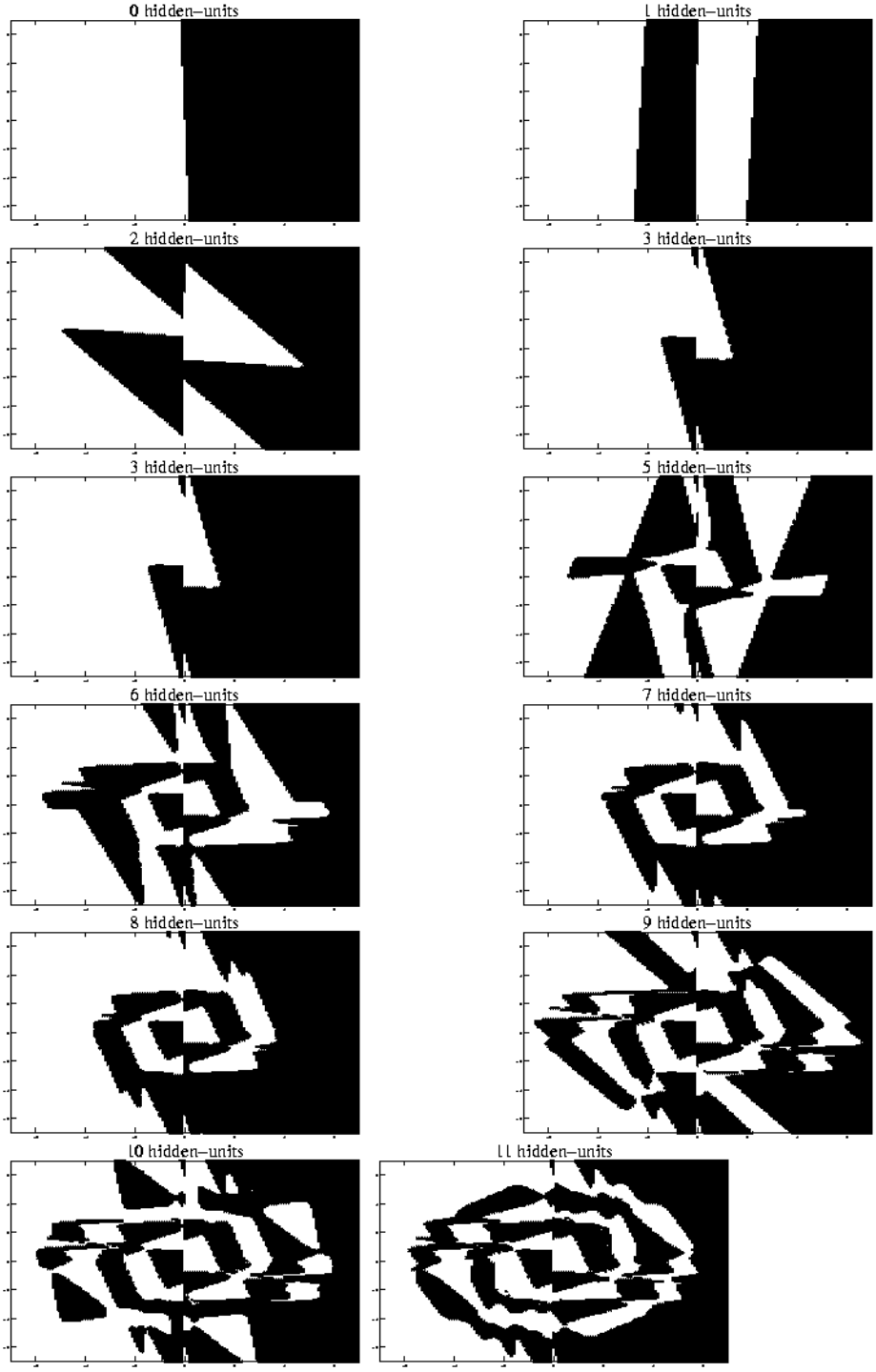
Figuur 1: Het twee spiralen probleem

3 Twee-spiralen-probleem

Zoals eerder vermeld heb ik het netwerk getest op het twee-spiralen-probleem, zie figuur 1. Deze bestaat uit twee spiralen van elk 96 punten in een plat vlak, waarvan de ene spiraal telkens de waarde 0.5 heeft en de andere een waarde van -0.5. Het is de bedoeling dat het netwerk net zolang leert totdat alle punten goed worden geclassificeerd. De resultaten zijn goed; dat wil zeggen: ze verschillen niet veel van de resultaten van Fahlman en Lebiere. Ik heb het 100 keer laten lopen en allemaal konden ze de punten classificeren. Gemiddeld gebruikten ze 2789 epochs (bij Fahlman en Lebiere was dit 1700) (het aantal epochs is totaal aantal van output- en input-phase). En ze gebruikten gemiddeld 14.46 hidden-knopen (bij Fahlman en Lebiere waren dit er 15.2). Mooi om te laten zien is hoe het leren verloopt (zoals ook in het verslag van Fahlman en Lebiere) te zien in figuur 2. In het figuur zie je wat het netwerk na elke output-phase representeert.

4 Balans-taak

Ik de taak precies hetzelfde uitgevoerd als Shultz beschrijft (hoop ik). Voor de training-set heb ik uit de 625 mogelijke configuraties de (100) afstand-problemen eruit gehaald. Ik heb 100 problemen gekozen met telkens 90% kans op een afstand-probleem. En na elke output-epoch heb er weer 1 probleem bij gedaan, dit keer met terugleggen, met dezelfde kansverdeling.



Figuur 2: Cascade-Correlatie losgelaten op de spiralen.

Voor de testset heb ik de 625 problemen opgesplitst in de 6 verschillende probleem-groepen (sommige vallen onder geen van deze groepen, dus die gooi ik weg). En de probleem-groepen heb ik weer opgesplitst in de 4 verschillende torque-difference-groepen. Leuk is hierbij dat de balans en balans-conflict problemen natuurlijk helemaal geen torque-difference hebben. Shultz heeft dit geloof ik een beetje over het hoofd gezien, want hier staat niks over is z'n verslag. Uit deze 24 groepen kies ik dan telkens 1 probleem.

Met deze testset test ik na elke output-epoch welke regel het netwerk gebruikt volgens dezelfde methode als Shultz. Ik heb het experiment 16 gedaan en telkens gestopt na 300 output-epochs (of als de hele trainingsset werd geclassificeerd). De resultaten zijn te zien in figuur 3.

Hierin is te zien dat er 5 keer 2 hidden-knopen nodig waren en 11 keer 1 . Dit is vergelijkbaar met de resultaten van Shultz[5][4].

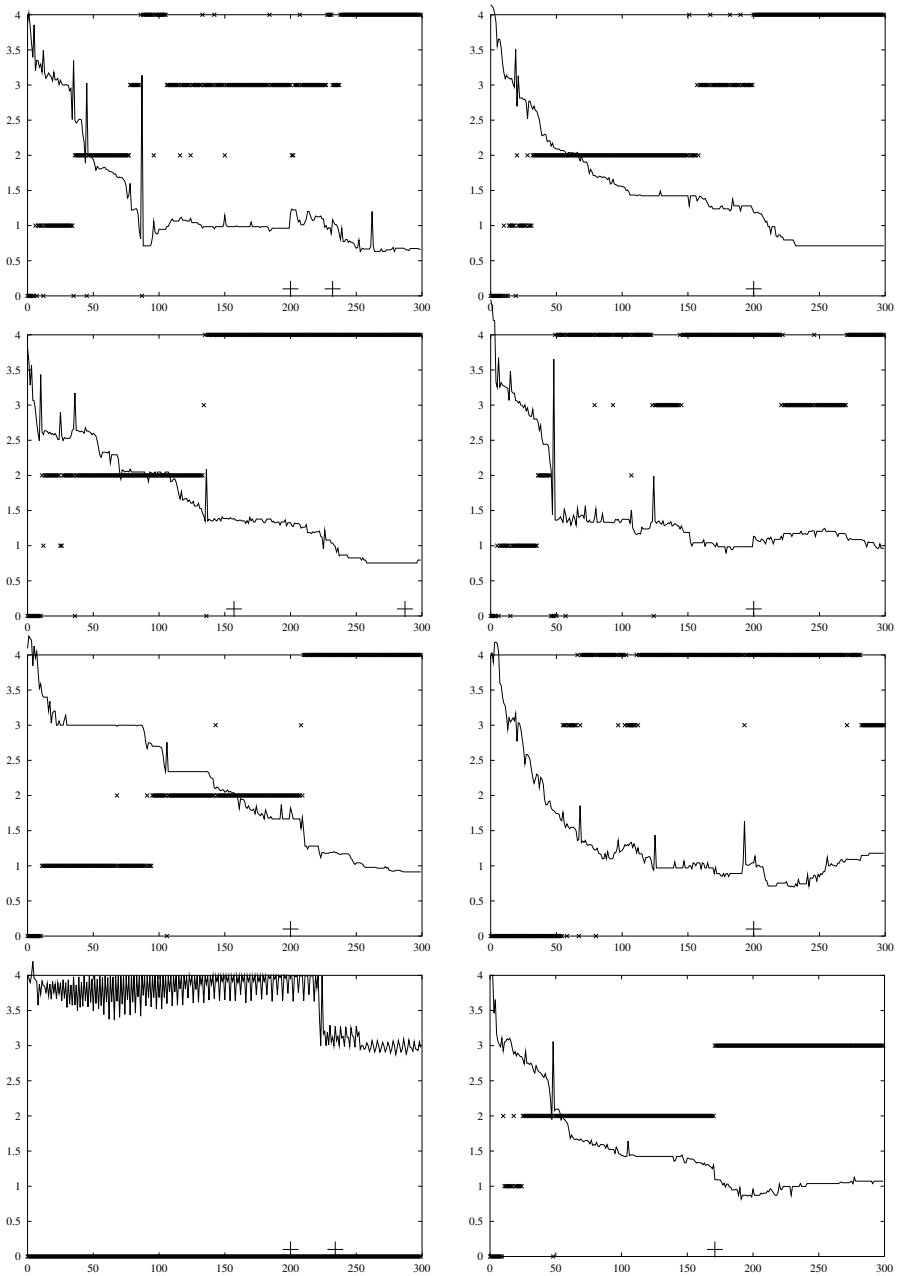
De volgorde waarin de regels worden gebruikt is niet echt duidelijk af te leiden, want vaak komt het voor dat er een kleine fluctuatie is (bijvoorbeeld dat 200 opeenvolgende epochs regel 4 wordt gebruikte behalve ergens middenin wanneer plotseling regel 3 wordt gebruikt). Maar over het algemeen is wel een bepaalde volgorde af te leiden, al hangt dit wel erg af van interpretatie.

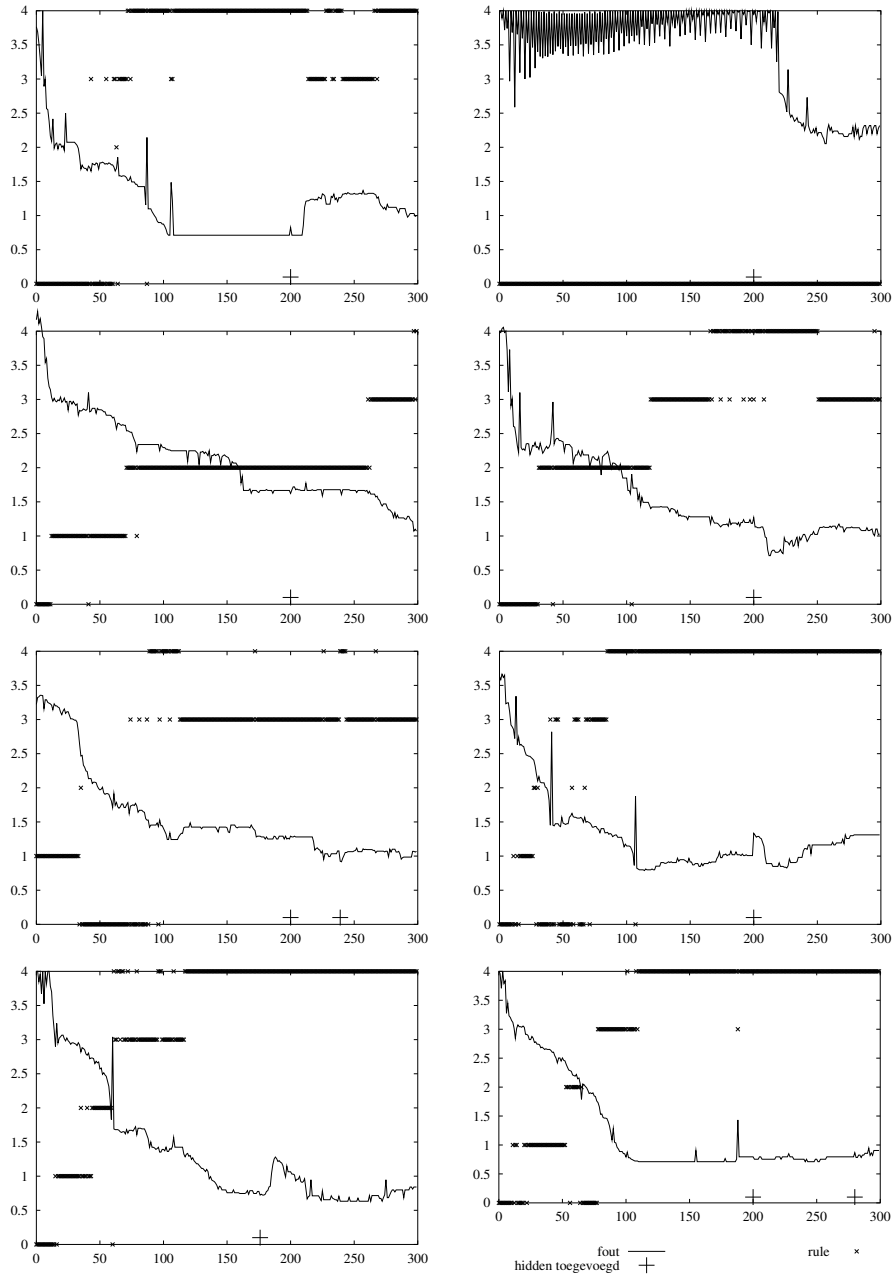
Mijn interpretatie is: 8 met de volgorde 1-2-3-4, waarvan 1 weer terug ging naar 3, 2 met volgorde 1-2-4, 1 met volgorde 1-3-4, 1 met 3-4 en 1 met 1-2-3. Ook bleven er 2 steken op 0 (geen regel), wat waarschijnlijk komt door een te grote ϵ gezien de fluctuatie in de leercurve (de fout). Wel is het verrassend om te zien dat regel 4 vaak al gebruikt wordt voordat er een hidden-knopen is toegevoegd.

Deze leercurve stelt het aantal fout geclassificeerde problemen voor. Ik heb hiervoor niet de 24 problemen uit de testset genomen waarmee de gebruikte regel wordt afgeleid, maar ik heb gewoon alle voorbeelden uit de probleem-groepen genomen, genormaliseerd naar het aantal problemen per probleemgroep (in totaal 425). Ik begrijp ook niet zo goed waarom Shultz maar 24 test-problemen gebruikt en niet gewoon allemaal, wat volgens mij een stuk betrouwbaarder is.

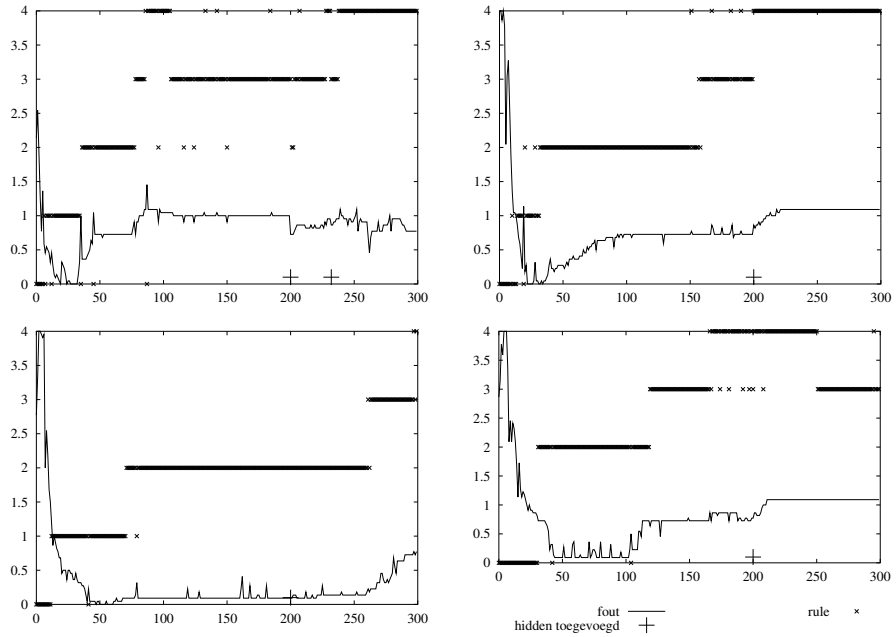
In de figuren kan je nu kijken of de verandering van regel samenhangt met een sprong in de leercurve. Dit is soms wel het geval, maar over het algemeen is dit niet echt duidelijk. Meestal gaat de leercurve gestaag omlaag, met wat uitschieters naar boven.

Om het de ontwikkeling, zoals Shultz beweert dat aanwezig is, verder te testen, heb ik extra gekeken naar de conflict-gewicht problemen. De classificatie van deze problemen hoort een vreemd verloop te hebben: met regel 1 en 2 is de classificatie 100% correct, met regel 3 is het maar 33% correct en met regel 4 wederom 100%. Dit moet terug te zien zijn in de leercurve van deze problemen, dus heb ik het aantal fout geclassificeerde





Figuur 3: De gebruikte regel uitgezet tegen de leercurve.



Figuur 4: De gebruikte regel uitgezet tegen het aantal fout geclassificeerde conflict-gewicht problemen.

conflict-gewicht problemen afgezet tegen de regel die gebruikt wordt, zie figuur 4.

Zoals te zien gaat de het aantal fout geclassificeerde conflict-gewicht problemen eerst sterk naar beneden, gaat daarna weer omhoog en blijft hoog. Dit klopt dus inderdaad met de voorspelling van regel 1, 2 en 3, maar niet met regel 4. Maar als je beter kijkt zie je dat het aantal fouten vaak al sterk toeneemt tijdens het gebruik van regel 2.

5 Conclusie

Ik vind Cascade-Correlatie niet fijn als netwerk. Het is leuk dat er tijdens het leren nieuwe knopen bijkomen, maar dit gebeurt wel met veel rekenwerk en niet echt op een elegante manier. Ook zijn er heel veel parameters en onduidelijkheden. Dit is op zichzelf niet erg, maar wel als de gebruiker ervan niet duidelijk aangeeft welke parameter-instellingen hij/zij heeft gebruikt. Ook is het zo dat de parameter-instellingen tijdens het onderzoek veranderd kunnen worden, om bepaalde gewenste resultaten te bereiken.

Ook vind ik de keuze van Shultz voor dit netwerk niet gegrond. Hij vind dat alleen een constructief netwerk de kwalitatieve cognitieve ontwikkeling

goed kan simuleren, door middel van het toevoegen van hidden-knopen. Maar zoals hij zelf heeft ontdekt, voegt Cascade-Correlatie vaak maar één knoop toe tijdens het leren, ook al doorloopt het netwerk 4 stadia. En daarbij gebruikt het netwerk meestal al regel 4 wanneer deze eerste knoop wordt toegevoegd.

6 Code

Ik heb het geheel in c++ geschreven, dat te vinden is op `http://gene.wins.uva.nl/õbooij/casco/`.

Referenties

[1] R. S. Crowder. C-implementatie van Cascade-Correlatie, port van Lips-implementatie van Scott E. Fahlman, 1992.

[2] Scott E. Fahlman. An empirical study of learning speed in back-propagation networks. Technical Report Computer Science Technical Report, 1988.

[3] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, February 1990.

[4] Thomas R. Schultz, Denis Mareschal, and William C. Schmidt. Modeling cognitive development on balance scale phenomena. *Machine Learning*, 16(1/2):57–86, 1994.

[5] T. R. Shultz and W. C. Schmidt. A cascade-correlation model of balance scale phenomena. In *13. Annual Conference of the Cognitive Science Society*, pages 635–640, Hillsdale, NJ, 1991. Cognitive Science Society.