

# Voorspellen van Wisselkoersen door Artificiële Neurale Netwerken

Jeroen Jansze  
Student CKI departement Wijsbegeerte  
E-mail: [j.jansze@students.uu.nl](mailto:j.jansze@students.uu.nl)

Onder begeleiding van Vincent van Oostrom en J.E.O. Renaud.  
Met dank aan Joran Bleeker.

**Abstract.** De afgelopen decennia is er veel onderzoek gedaan naar artificiële neurale netwerken (ANN) en applicaties daarvan. Een van deze applicaties is het voorspellen van financiële tijdseries, waaronder ook wisselkoersen. In dit onderzoek worden twee soorten neurale netwerken en een statistische methode voor het voorspellen van wisselkoersen met elkaar vergeleken. De euro/GBP koers en de Canadese dollar/Amerikaanse dollar zijn gebruikt als datasets. Het gebruikte criterium voor de prestaties van de methoden is de Root Mean Squared Error (RMSE). Uit dit onderzoek blijkt dat neurale netwerken beter wisselkoersen kunnen voorspellen dan een statistische methode. Met een goed criterium om een voorspelling te accepteren zouden neurale netwerken meer gebruikt kunnen worden voor het voorspellen van wisselkoersen om effectief te kunnen speculeren op de valutamarkt.

## 1. Introductie

Artificiële neurale netwerken (ANN's) zijn in de afgelopen jaren steeds populairder geworden bij het voorspellen van economische tijdseries [1,2,3]. Een tijdserie bestaat uit een aantal datapunten met ongeveer dezelfde tijd tussen die punten. De waarden in deze series representeren metingen van economische activiteiten. Wisselkoersen zijn voorbeelden van economische tijdseries, maar kunnen ook gemodelleerd worden als continue functies. Een voorbeeld van zo'n model is het Krugman-model [4]. Dit soort modellen bevatten veel variabelen, die in praktijk moeilijk zijn om te bepalen (er zijn hoge kosten verbonden aan het verzamelen van deze informatie). Om deze reden heeft men gezocht naar andere manieren om de wisselkoersen te voorspellen. Omdat wisselkoersen gezien kunnen worden als gecompliceerde functies, is het mogelijk deze te benaderen. Dit kan gedaan worden met een ANN. Er is bewezen dat een ANN met één hidden layer (met voldoende nodes, zie sectie 2.1) elke continue functie optimaal kan benaderen [5] en dus is er veel onderzoek gedaan naar het voorspellen met behulp van ANN's.

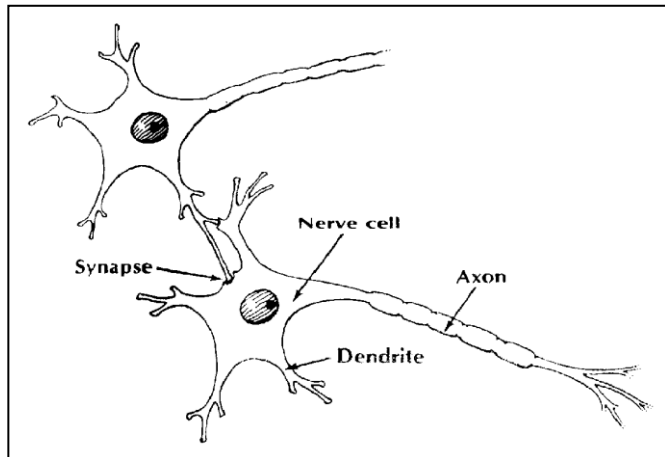
Andere voorbeelden van economische tijdseries zijn de aandelenmarkt en voorraadbeheer (wat nauw samenhangt met het voorspellen van de vraag naar goederen). Het is belangrijk voor partijen om deze tijdseries te kunnen voorspellen, zodat zij kosten kunnen reduceren of omzet kunnen verhogen. Op basis van dat laatste een simpel voorbeeld: ik wil boeken gaan verkopen in de maand september. De boeken worden ingekocht voor 20 euro per stuk en verkocht voor 50 euro per stuk. Elk boek dat niet verkocht wordt kost me dus 20 euro en elk boek dat ik had kunnen verkopen kost me 30 euro ( $50 - 20 =$  de winst die ik misloop per boek). Stel dat ik 30 potentiële klanten heb terwijl mijn voorspelling 32 klanten bedraagt, dan heb ik een omzet van 1500 euro en de kosten bedragen 640 euro. Met een betere voorspelling, bijvoorbeeld 31 klanten, zou ik nog steeds 1500 euro omzet hebben maar nu bedragen de kosten 620.

Statistische methoden zoals de Box-Jenkins-methode (tegenwoordig gebruikt in veel spreadsheetprogramma's) blijken minder goed te werken voor voorspelling in economische tijdseries dan ANN's. Dit komt doordat deze series niet-lineair zijn (d.w.z. de opvolgende waarden zijn niet lineair te mappen) [2]. ANN's kunnen aan de hand van alleen de data, de onderliggende patronen leren en zo een gecompliceerde functie benaderen, waarmee vrij accurate voorspellingen gedaan kunnen worden.

In deze studie wordt de nauwkeurigheid van een Feed Forward Neuraal netwerk (FFN) en een Recurrent Neuraal Netwerk (RNN) vergeleken met een versie van de Box-Jenkins ARIMA methode, genaamd simple exponential smoothing [6]. Vanuit de literatuur is bekend dat elke lineaire statistische methode het niet significant beter zal doen dan een simpel random-walk model [7].

## 1.1. Relevantie voor CKI

Een groot deel van CKI gaat over het modelleren van menselijke eigenschappen. Zo worden in de psychologie, één van de pilaren van CKI, en in de biologie de hersenen bestudeerd. De hersenen bestaan voor een groot deel uit zenuwcellen die ook wel neuronen worden genoemd (zie figuur 1.1). Deze neuronen staan in verbinding met elkaar en geven via deze verbindingen signalen door. Een ANN is een model van dit biologische verschijnsel. De neuronen staan voor de nodes (de knopen) in het netwerk en de synapsen voor de verbindingen. Net als menselijke hersencellen kan een netwerk informatie opslaan en fouten ten opzichte van nieuwe informatie corrigeren. De hersencellen doen dit door op bepaalde snelheden te vuren, wat in het netwerk gerepresenteerd wordt door gewichten (waarden die worden toegekend aan de verbindingen). Hoewel mensen naar schatting in de orde  $10^{27}$  van deze cellen in de hersenen hebben, blijken enkele tientallen nodes ook kleine taken op te kunnen lossen waarbij leren een vereiste is. Men probeert meer te weten te komen over de hersenen door het bestuderen van de netwerkmodellen en er nuttige applicaties mee te maken.



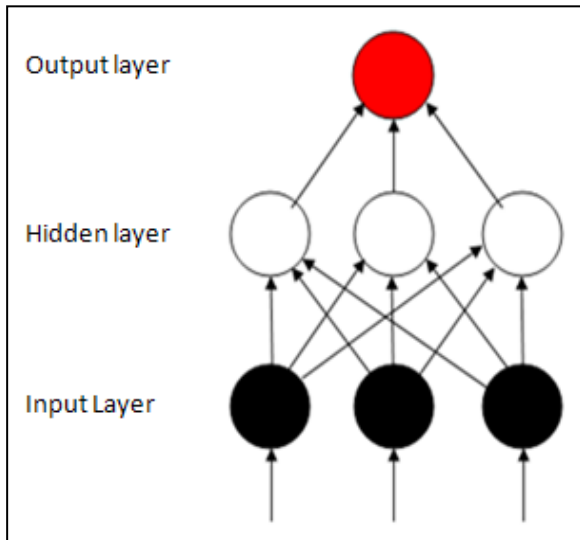
Figuur 1.1 – Twee zenuwcellen verbonden door een synaps.

De indeling van deze scriptie is als volgt: in sectie 2 worden de netwerkmodellen uitgelegd<sup>1</sup>, sectie 3 gaat over de gebruikte inputdata en de experimentopzet en in sectie 4 en 5 worden de resultaten besproken.

## 2. Netwerkmodellen

Neurale netwerken kunnen gemodelleerd worden met grafen. Deze bestaan uit knopen, welke de neuronen voorstellen, en verbindingen tussen die knopen, welke kunnen worden gezien als de axonen (zie figuur 1.1). De knopen hebben een topologische ordening. De netwerken die voor dit onderzoek gebruikt worden hebben 3 layers (lagen van knopen): een input layer, waar de data het netwerk binnenkomt, een hidden layer en een output layer die bestaat uit één node, omdat we geïnteresseerd zijn in voorspelling van één stap vooruit.

1. Sommige termen zijn in het Engels, daarmee wordt aangegeven dat deze termen horen bij een netwerk. De gelijke termen in het Nederlands (bijvoorbeeld: value en waarde) horen bij datasets.



Figuur 2.1 – Een volledig verbonden netwerk met, de input nodes zwart, de hidden nodes wit en de output node. rood.

## 2.1. Feed Forward Neuraal Network

De input layer is volledig verbonden met de hidden layer en de hidden layer met de output node (zie figuur 2.1), wat betekent dat elke input node verbonden is met elke hidden node en elke hidden node met de output node. De verbindingen tussen de nodes hebben gewichten gerepresenteerd door een waarde tussen 0 en 1. Deze waarden zijn random geïnitieerd. Het algoritme dat we gebruiken om dit netwerk te trainen heet het backpropagation-algoritme. Dit algoritme bestaat uit twee stappen: de feed-forward-stap en de backpropagation-stap. Tijdens de eerste stap wordt de data aan de input nodes gegeven waarna de values (waarden) voor de hidden nodes berekend worden met de formules:

$$(1a) \quad V_j = \sum_{i=1}^p W_{ij} * I_i$$

$$(1b) \quad A_j(V_j) = \frac{1}{1 + \exp(-V_j)}$$

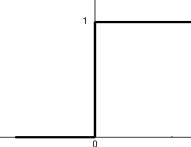
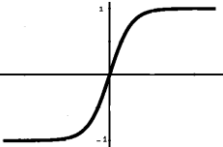
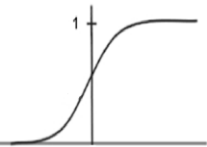
In formule (1a) is  $W_{ij}$  het gewicht van de verbinding tussen node  $i$  en node  $j$ ,  $I_i$  de waarde uit de dataset (in dit onderzoek de hoogte van een wisselkoers) en  $p$  het aantal binnenkomende verbindingen. De som  $V_j$  wordt als argument gebruikt in de activatie-functie  $A_j$ . Deze functie kan gezien worden als regelaar voor de sterkte van het signaal dat via de verbinding bij de volgende node aankomt. Dit correspondeert met de menselijke hersenen, omdat daar ook bepaald moet worden of een neuron vuurt (een signaal afgeeft) of niet. In een netwerk wordt de activatie-functie ook

gebruikt om niet-lineairiteit te introduceren in het netwerk (meer daarover in sectie 2.2.).

## 2.2. Activatie-functies

Formule (1b) is een voorbeeld van een activatie-functie. Deze functie zorgt ervoor dat het netwerk de niet-lineaire mapping kan maken van input naar output. Een functie  $f$  wordt lineair genoemd als deze voldoet aan twee eisen:

- $f(x + y) = f(x) + f(y)$  (Additiviteit-eigenschap)
- $f(\alpha x) = \alpha f(x)$  for all  $\alpha$  (Homogeniteit-eigenschap)

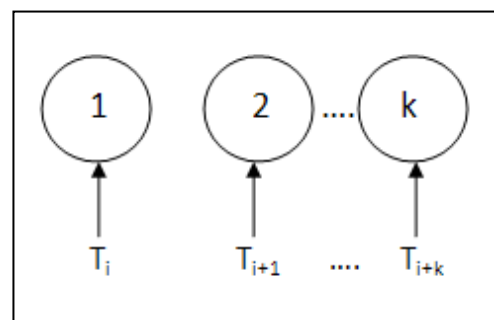
Naam/grafische weergave:	Functie:	Omschrijving:
1. Step-functie 	$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$	- Kan de waarden 0 en 1 krijgen. - Goed bruikbaar voor het kwalificeren van data in twee groepen.
2. Tanh-functie 	$\varphi(v) = \tanh\left(\frac{v}{2}\right) = \frac{1 - \exp(-v)}{1 + \exp(-v)}$	- Kan alle waarden $x$ met $-1 < x < 1$ krijgen. - Goed bruikbaar voor continue data als deze ook negatief is.
3. Sigmoid-functie 	$\varphi(v) = \frac{1}{1 + \exp(-v)}$	- Kan alle waarden $x$ met $0 < x < 1$ krijgen. - Goed bruikbaar voor continue data als deze positief moet zijn.
4. Cloglog-functie	$\varphi(v) = 1 - \exp(-\exp(v))$	Zie 3.
5. Probit-functie	$\varphi(v) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^v \exp\left(-\frac{v^2}{2}\right) dv$	Zie 3.
6. loglog	$\varphi(v) = \exp(-\exp(v))$	Zie 3.

Tabel 2.1 – Voorbeelden van activatie-functies. De  $v$  wordt berekend met formule (1). De eerste 3 zijn veel gebruikte functies. De laatste 3 zijn voorgesteld in recent onderzoek [3] als activatie-functie speciaal voor netwerken die economische tijdseries voorspellen en lijken grafisch gezien sterk op de sigmoid-functie. De reden dat deze functies veel gebruikt worden is vanwege de biologische plausibiliteit. Neuronen vuren bij een bepaalde waarde (wat gemodelleerd kan worden met de step-functie) en om ook continue waarden toe te laten (bijvoorbeeld wisselkoersen) is gekozen voor een continue versie van de step-functie.

Als er een lineaire functie wordt gebruikt is het netwerk equivalent aan een Single Layer Network (SLN) welke niet geschikt is voor het voorspellen van wisselkoersen vanwege het niet-lineaire karakter van de data [8,2]. De reden hiervoor is dat een lineaire functie van lineaire functies ook een lineaire functie is. Meestal hebben activatie-functies uitkomsten tussen 0 en 1, of tussen -1 en 1. Veel niet-lineaire functies kunnen ingezet worden als activatie-functie (zie tabel 2.1). Voor het backpropagation-algoritme is het nodig dat de functie differentieerbaar is. Vanwege de vorm van de data (meer daarover in sectie 3), is er voor dit project gekozen voor de Sigmoid-functie. Deze functie heeft zijn naam te danken aan de S-vormige plot.

### 2.3. Gesuperviseerd leren

Elke input node krijgt een opvolgende waarde uit de dataset, dus met  $k$  input nodes en met  $T$  het aantal datapunten uit de set van input (in ons geval de hoogte van de wisselkoers), krijgt de eerste input node datapunt  $T_i$ , de tweede input node  $T_{i+1}$  enzovoort t/m node  $k$  die het punt  $T_{i+k}$  krijgt (zie figuur 3.1). De waarde die voor supervising wordt gebruikt is  $a_{i+k+1}$  (van het engels, actual value). Het backpropagation-algoritme is een vorm van gesuperviseerd leren en is voor het eerst gepubliceerd door D.E. Rumelhart et al.(1986) [9]. Het netwerk wordt getraind door elke uitkomst van het netwerk (verkregen door het feed-forward-proces, in ons geval de value van de output node,  $o_i$ , te vergelijken met de werkelijke waarde,  $a_{i+k+1}$ . Het doel van het algoritme is om de combinatie van gewichten te vinden die zorgen voor een minimalisatie van de globale error-functie. Deze functie moet aangeven hoeveel de benadering van het netwerk af zit van de werkelijke waarden. Voor het backpropagation-algoritme wordt de “sum of squared errors” gebruikt:



Figuur 3.1 – Elke inputnode krijgt een opvolgend datapunt uit de dataset.

$$(2) \quad SSE = \frac{1}{2} \sum_{i=k+1}^T (a_{i+k+1} - o_i)^2$$

Hierin is  $T$  het aantal datapunten,  $k$  het aantal input nodes van het netwerk,  $o_i$  de output van het netwerk en  $a_i$  de werkelijke waarde. In deze functie wordt de fout gekwadrateerd (om negatieve waarden om te zetten naar positieve waarden en grote fouten extra uit te vergroten). De factor 0,5 is ervoor om een handige afgeleide te krijgen. Deze functie kan geminimaliseerd worden door voor elke gemaakte fout de gewichten aan te passen. Bij een te hoge value  $o_i$  is het de bedoeling de gewichten naar beneden bij te stellen en als de value te laag is dan de gewichten te verhogen,

zodat conform de formules (1a) en (1b) de fout (het verschil tussen de output en de werkelijke waarde) kleiner wordt.

Het is nodig dat fout verdeeld wordt over de gewichten, omdat niet elk gewicht evenveel "schuld" heeft aan de gemaakte fout. Voor elke node wordt hiertoe eerst de blame berekend. De blame is een maat voor de verdeling van de gemaakte fout over de gewichten. Rumelhart et al. hebben hiervoor op een intuïtieve manier een formule gevonden. De formule bestaat uit 2 delen:

- De verandering van de gemaakte fout, geschreven als functie van een verandering in  $V_j$  (zie (1)), voor de output node is dit:  $(a_{j+k+1} - o_j)$ , de afgeleide van de SSE.
- Het effect van de verandering van één gewicht op  $V_j$  welke de afgeleide is van de activatie-functie. De afgeleide van de Sigmoid-functie is:  $S(x)(1-S(x))$

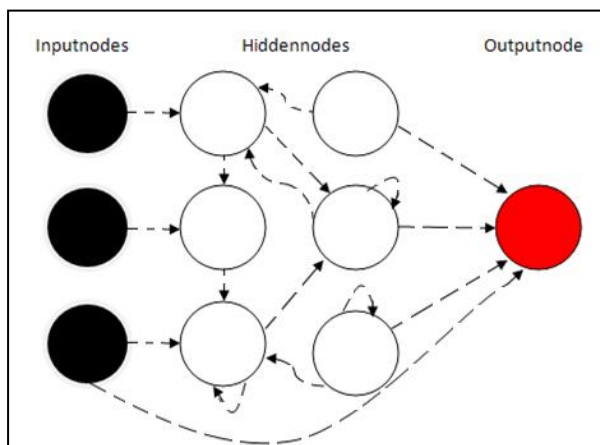
De hele vergelijking voor het berekenen van de fout voor de output node wordt dus:

$$(3) \quad B_j = (a_{j+k+1} - o_j) * A_j(V_j) * (1 - A_j(V_j))$$

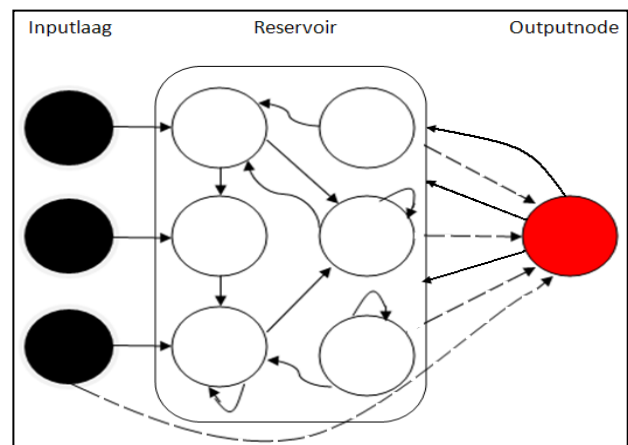
Waarin  $B_j$  de blame voor de output node is. Deze fout wordt teruggepropageerd door het netwerk (van de output node naar de input nodes) door de fout van node  $j$  te vermenigvuldigen met  $W_{ij}$ . Zoals eerder gezegd moeten de gewichten worden aangepast met een gedeelte van fout, omdat alle input nodes hebben bijgedragen aan deze fout. De value die verandert door het aanpassen van de gewichten is de value van de opvolgende node in het netwerk en die is berekend door de activatie-functie. De formule voor het updaten van de gewichten wordt:

$$(4) \quad \Delta W_{ij} = \eta * B_j * A_i(V_i)$$

Hierin is  $\eta$  een learning rate, welke er voor zorgt dat het leren in kleinere stappen gebeurt. Dit is nodig voor een betere convergentie van het netwerk. In appendix A staat dit uitgelegd met een grafisch voorbeeld.



Figuur 2.2 – Een voorbeeld van een RNN. Tussen de hidden nodes kan de data teruggevoerd worden.



Figuur 2.3 – Voorbeeld van een RNN die getraind wordt met reservoir computing. De hele hidden layer wordt het reservoir genoemd. Alleen de doorbroken lijnen worden getraind, de rest van de lijnen worden random geïnitieerd en blijft daarna constant.

De procedure, feedforward en backpropagation, kan een paar keer herhaald worden over de dataset, het aantal training runs genoemd (meer daarover in sectie 3.1).

## 2.4. Recurrent Neuraal Netwerk

Een meer recent netwerk dat gebruikt wordt voor het voorspellen van wisselkoersen is een RNN [10,11]. Bij een RNN wordt de data op een bepaald punt in het netwerk teruggevoerd (zie figuur 2.2.). Dit in tegenstelling tot een FFN (zie sectie 2.1.). Sommige wetenschappers denken dat economische tijdseries autocorrelatie bevatten [11]. Dit is de mate waarin de onderlinge punten van een tijdserie met elkaar correleren. Of dit ook voor wisselkoersen geldt is nog maar de vraag. RNN's hebben, in tegenstelling tot FFN's, korte termijn geheugen [12], wat nodig is om de correlaties tussen onderlinge punten te leren. Bij een FFN worden de gewichten alleen geüpdatet bij het invoeren van externe input (nieuwe datapunten uit de dataset). Bij een RNN worden de gewichten ook geüpdatet met activatie uit de vorige feedforward-iteratie. Het RNN presteert niet op elke punt beter dan het FFN en er kan dus niet met zekerheid gezegd worden dat de wisselkoersen ook echt autocorrelatie bevatten. Een andere reden voor het gebruik van een RNN ten opzichte van een FFN is dat het RNN model meer lijkt op de organisatie van menselijke neuronen. Om bij mensen korte termijn geheugen mogelijk te maken is het nodig dat informatie dezelfde neuronen kan blijven bereiken. Uit dit onderzoek blijkt dat niet in elk geval het RNN beter wisselkoersen voorspelt dan een FFN.

## 2.5. Reservoir Computing

Een groot nadeel van een RNN is dat het veel tijd kost om deze te trainen. Voor dit onderzoek zijn er relatief kleine datasets gebruikt. Het is ook mogelijk om van de afgelopen 10 jaar de wisselkoersen van elke uur als dataset te gebruiken. Wat zal resulteren in 87600 datapunten. Met deze hoeveelheden data gaan rekenkosten een rol spelen. In tegenstelling tot een FFN gaat de data meerdere keren door het netwerk wat per iteratie meer tijd gaat kosten. Daar komt bij dat het langer kan duren voordat het RNN convergeert, wat te wijten is aan het extra geheugen. Om deze reden is er vrij recent een manier gevonden om RNN's efficiënt te trainen, genaamd reservoir computing. In deze methode worden de hidden nodes gezien als een reservoir (zie figuur 2.3). Een globale descriptie van het model wordt gegeven door:

$$(5a) \quad x[t + 1] = f(W_{res,res}x[t] + W_{inp,res}u[t])$$

$$(5b) \quad y[t + 1] = W_{res,out}x[t + 1] + W_{inp,out}u[t] + W_{bias,out}$$



Hierin is  $u[t]$  de input,  $x[t+1]$  de state (een matrix die een configuratie representeert) van het netwerk, en  $y[t+1]$  de outputs van het netwerk. De functie  $f$  is een activatie-functie (zie tabel 2.1). Alle  $W_{inp,res}$  en  $W_{res,res}$  (matrices van gewichten) worden random geïntialiseerd en blijven constant voor het trainen en testen. De onderliggende gedachte is dat het reservoir werkt als een complexe, niet-lineaire en dynamische transformatie van de inputs (zie (5a)), welke met een simpele lineaire mapping vertaald kan worden naar de output (zie (5b)). Alleen  $W_{res,out}$  en  $W_{inp,out}$  worden getraind (de stippellijnen in figuur 2.3), wat de rekenkosten laag houdt.

Net als bij een FFN wordt de data (al dan niet genormaliseerd) gevoerd aan de input layer en zo aan het reservoir. Hierna worden de states verzameld van het netwerk en die vertaald naar outputs (zie 5a en 5b). De vertaling is een lineaire mapping en het doel is om een optimale mapping te vinden. Dit wordt gedaan door lineaire regressie of ridgeregressie (zie sectie 8.1.1 van [13]).

### 3. Data en experiment

Voor dit experiment is gekozen voor de bilaterale wisselkoersen (d.w.z. de wisselkoersen waarbij één munt op 1,00 wordt gezet en de andere een uitdrukking is van waarde ten opzichte van de eerste) tussen de Britse pond (GBP) en de Euro. De data komt uit de database van de European Central Bank (ECB) en bestaat uit de wekelijkse koers van woensdag 6 januari 1999 t/m 11 mei 2011. De data is dagelijks vastgesteld op 2:15 pm Central Eastern Time. Als een woensdag geen wisselkoers is vastgesteld dan is de dinsdag ervoor genomen en als die ook niet beschikbaar is, de donderdag in dezelfde week. De dataset bestaat uit 641 datapunten.

De Euro is als basis gezet (op 1,00), omdat de GBP meer waard is dan de Euro en de wisselkoers zo tussen 0 en 1 uitkomt. Dit is wenselijk, omdat de output van het netwerk ook tussen 0 en 1 is geschaald (door het gebruik van de Sigmoid-functie, zie sectie 2.1.). Voor het vergelijken van de werkelijke waarde,  $a_{i+k+1}$ , met de output value,  $o_i$ , is het node dat deze hetzelfde geschaald zijn om iets te kunnen zeggen over de gemaakte fout. Er zijn onderzoeken die suggereren dat behalve het verkrijgen van vergelijkbare waarden, normalisatie ook de voorspelling verbeterd. Een uitgebreid onderzoek hierover is gedaan door M. Shanker et al. [14]. In dit onderzoek is aangenomen dat normalisatie de voorspelling niet verbeterd, omdat er in de literatuur nog geen consensus is bereikt op dit punt.

Een tweede onafhankelijke dataset bestaat uit de bilaterale wisselkoersen tussen de Amerikaanse dollar en de Canadese dollar. Deze data komt uit de database van Bank of Canada en bestaat uit dagelijkse wisselkoersen van woensdag 20 juni 2001 t/m dinsdag 18 september 2007. De data is vastgesteld op 0:00 pm Central Eastern Time. De dataset bestaat uit 1567 datapunten.

Alle datasets zijn bij deze scriptie gevoegd.

### 3.1. Experimentopzet FFN

Twee parameters voor het netwerk die gekozen moeten worden zijn het aantal input nodes en het aantal hidden nodes. Voor tijdseries met een seizoensfactor hebben de netwerken gewoonlijk net zoveel input nodes als datapunten in één cyclus. Voor wisselkoersen vallen onderzoekers terug op experimenteren [1,2]. In dit project bleek er weinig verschil te zijn bij 6 of meer input nodes. Het is vanwege rekenkosten beter om zo weinig mogelijk nodes te hebben met een optimaal resultaat. Voor de hidden nodes is het een stuk moeilijker gebleken. 13 t/m 17 hidden nodes maakte weinig verschil. Bij minder dan 13 waren de resultaten (de SSE) slechter en bij meer dan 17 kwam het vaak voor dat het netwerk slecht convergeerde. Er is gekozen om verder te werken met 16, omdat de hoeveelheid hidden nodes voor de hoeveelheid geheugen van het netwerk staat (meer hidden nodes is meer gewichten is meer geheugen). Voor de grote dataset is er gebruikt gemaakt van 20 hidden nodes.

Een andere parameter die bepaald moet worden is het aantal trainingsrondes. Als een netwerk te weinig getraind wordt en om deze reden nog niet sterk genoeg geconvergeerd is, zal dit leiden tot een slechte nauwkeurigheid bij het voorspellen (ook wel under-fitting). Aan de andere kant, als een netwerk te veel getraind wordt, kan dit leiden tot over-fitting [15,16]. Dit is een staat van het netwerk waarbij alle datapunten heel goed onthouden worden door het netwerk, maar waarbij onderliggende functie slecht benaderd wordt. Over-fitting komt vaker voor in grote netwerken, omdat deze meer geheugen hebben. In de literatuur zijn verscheidene methoden geïntroduceerd die over-fitting tegengaan zoals: een restrictie op de gewichten [15], het robuuster maken van het BP-algoritme [16] en/of het vroegtijdig afbreken van de training [17]. Voor dit project is er gekeken na welk aantal trainingsrondes de uitkomsten nauwelijks (in de orde van  $10^{-5}$ ) meer veranderden. Dit was bij meer dan 5 trainingrondes. Toch is er gekozen voor 10 rondes, omdat met te weinig trainingsrondes het netwerk te vaak slecht convergeert (door het random initialiseren van de gewichten).

Voor dit onderzoek is een FFN geschreven in java. Dit programma is afhankelijk van net.datastructures<sup>1</sup>, vanwege de gebruikte datastructuren, en JFreeChart<sup>2</sup> voor het maken van grafieken. Deze zijn allemaal bij deze scriptie gevoegd.

### 3.2. Experimentopzet exponential smoothing.

We vergelijken de prestaties van het FFN met de statistische methode exponential smoothing. Bij deze methode krijgen datapunten een gewicht. Deze gewichten worden met elke iteratie verminderd met een exponentiële factor (oudere datapunten krijgen steeds minder gewicht). Deze methode gaat niet uit van cyclussen

1. <http://net3.datastructures.net/>

2. <http://www.jfree.org/jfreechart/>

of trends in de data. Een voorspelling door deze methode wordt gegeven met de formule:

$$(6) \quad o_t = \alpha * a_{t-1} + (1 - \alpha) * o_{t-1}$$

$\alpha$  is hierin een smoothing-constante ( $0 < \alpha < 1$ ) en bepaalt hoe flexibel dit model omgaat met fluctuaties in de data. Het bepalen van de beste smoothing-constante moet experimenteel bepaald worden. Een  $\alpha$  van 0.03 gaf de beste resultaten (zie appendix C) dus deze waarde zal voor verdere experimenten gebruikt worden.

### 3.3. Experimentopzet RNN.

Alle experimenten met RNN's zijn gedaan met de oger toolbox<sup>1</sup>. De toolbox is afhankelijk van andere packages. Voor dit onderzoek is gebruik gemaakt van pythonxy<sup>2</sup>. Dit is software die zorgt voor alle benodigdheden voor de oger toolbox. Voor werkende versie van een programma dat wisselkoersen voorspelt en de "grid search" (daarover hieronder meer) kan contact opgenomen worden met Joran Bleeker (j.bleeker@students.uu.nl).

Het RNN heeft verschillende parameters die bepaald moeten worden. Ten eerste de grootte van het reservoir. In tegenstelling tot een FFN heeft een RNN geen last van over-fitting. Dus de enige reden waarom er niet teveel nodes in het reservoir zouden moeten zijn is dat de rekentijd langer wordt met meer nodes. Voor dit project is gekozen voor 300 nodes, wat veel meer is dan voor een FFN. We hebben ook met 400 nodes getest om te zien of dat beter zou werken dan het gebruik van 300 nodes. Dit maakte geen verschil er is dus voor 300 gekozen. Waarschijnlijk geeft deze grootte van het reservoir de optimale voorspellingen voor een RNN, alleen zou het kunnen dat de rekentijd korter kan, als een kleiner reservoir gekozen wordt.

Verder zijn er nog vier parameters: de input scaling, de leak rate, de spectral radius en de parameter voor ridgeregressie. Al deze parameters zijn experimenteel bepaald door een "grid search" (alle combinaties van parameters worden afgegaan) en worden hieronder verder toegelicht.

De data kan niet rechtstreeks aan het netwerk gevoerd worden, omdat er gebruikt gemaakt wordt van de tanh-functie (zie tabel 2.1). De inputs moeten dus geschaald worden. Voor dit project was deze parameter 0,5. Dit wil zeggen dat elke waarde uit de inputmatrix vermenigvuldigd is met 0,5.

De leak rate is de maat waarmee de activiteit gedempt wordt voor elke node en ligt tussen 0 en 1 (waarbij 1 ook mogelijk is) [18]. Als de leak rate 1 is zijn alle nodes gelijk aan de nodes van een FFN en hebben dus geen geheugen (de waarde hangen af van de input). Deze nodes zijn beter voor sterk fluctuerende data (geeft een hoge

1. <http://reservoir-computing.org/oger>

2 <http://www.pythonxy.com/>

dynamiek in de output). Bij lage waarden reageert het netwerk minder sterk op “pieken” in de input. In plaats van formule (5a) wordt dan gebruikt gemaakt van:

$$(7) \quad x[t + 1] = C(-Lx[t]) f(W_{res,res}x[t] + W_{inp,res}u[t])$$

Hierin is C een tijdconstante en L de leake rate. Uit de grid search kwam een waarde van 0,6 en deze is dus gebruikt voor de experimenten.

De derde parameter is de spectral radius. Dit is de waarde waarmee de gewichten geschaald worden. De formule is:

$$(8) \quad \rho = \max(\text{eig}(W_{res,res}))$$

Voor het netwerk bepaalt deze parameter, net als de leak rate, hoe snel activiteit in het netwerk gedempt wordt. Bij een hoge instelling zal er minder gedempt worden en zal een input signaal langer interacteren met het netwerk. Als deze te laag wordt ingesteld zal het signaal na het “vuren” van 1 node verdwijnen. Uit de search kwam een waarde van 1.3, wat vrij hoog is. Met deze instelling geeft het netwerk soms een chaotische output (zie appendix C).

De parameter voor ridge regression kwam uit op  $10^{-8}$ .

Voor dit project testen we alle gebruikte methoden 1 t/m 60 stappen vooruit. De datapunten worden hiervoor in twee groepen verdeeld. Eén groep is voor het trainen en die heeft een grootte van 641 – het aantal stappen vooruit. De andere groep is net zo groot als het aantal stappen dat vooruit wordt voorspeld en dient voor het testen van het netwerk (de ouputs van het netwerk wordt vergeleken met de waarden uit deze groep).

Als laatste moet er gespecificeerd worden wat de criteria zijn voor de prestaties van het netwerk. In tabel 3.1 staan een paar veel voorkomende. Dit zijn allemaal metrieken. Een metriek is een functie die de afstand tussen elk tweetal elementen van een verzameling definieert. Metrieken voldoen aan de volgende eigenschappen:

- $d(x,y) \geq 0$  (alle afstanden zijn positief).
- Als  $d(x,y) = 0$  dan en slechts dan als  $x = y$  (als de afstand 0 is zijn x en y dezelfde punten en als x en y dezelfde punten zijn is de afstand 0).
- $d(x,y) = d(y,x)$  (de afstand tussen x en y is altijd gelijk aan de afstand tussen y en x).
- $d(x,z) \leq d(x,y) + d(y,z)$  (de driehoeksongelijkheid, de afstand tussen x en z is altijd kleiner of gelijk aan de afstand via y).

De meest bekende metriek is de Euclidische metriek welke in een 2D ruimte wordt gedefinieerd door:

Naam:	Functie:	Omschrijving:
1. Root Mean Squared Error	$RMSE = \sqrt{\frac{\sum(a_t - o_t)^2}{N}}$	- Geeft een gemiddelde van alle errors. - Geeft meer gewicht aan relatief grote fouten.
2. Mean Absolute Error	$MAE = \frac{\sum  a_t - o_t }{N}$	- Geeft een gemiddelde van alle errors. - Alle foutmetingen zijn hierbij gelijk.
3. Mean Absolute Percentage Error	$MAPE = \frac{100}{N} \sum \left  \frac{a_t - o_t}{a_t} \right $	- Geeft de nauwkeurigheid als percentage. - Is goed bruikbaar voor het vergelijken van verschillende series (vanwege de uitdrukking in een percentage).
4. Normalized Mean Squared Error	$NMSE = \frac{1}{N} \frac{\sum(a_t - o_t)^2}{\sigma^2}$  (Waarbij $\sigma^2$ de variantie is van de hele dataset)	- Is goed bruikbaar voor het vergelijken van verschillende series (vanwege het normaliseren door de variantie)
5. Symmetric Mean Absolute Percentage Error	$SMAPE = \frac{100}{N} \sum \frac{ a_t - o_t }{(a_t + o_t)/2}$	- Een variatie van 3. - Heeft een bovengrens en ondergrens (is daardoor minder gevoelig voor uitschieters in een serie in vergelijking met een andere serie) - Overschatting en onderschatting worden niet gelijk behandeld.

Tabel 3.1 – Veel gebruikte maatstaven voor de fouten van voorspellingen, waarbij N een subset is van T. De NMSE en de SMAPE zijn recenter ontwikkeld dan de eerste 3.

$$(9) \quad d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Hierin is  $d(x, y)$  de afstand tussen  $x$  en  $y$ , welke zich bevinden op de coördinaten  $(x_1, x_2)$  en  $(y_1, y_2)$  respectievelijk. Hiervan is de RMSE afgeleid. Een andere bekende metriek is de Manhattan metriek. In een 2D ruimte is deze als volgt gedefinieerd:

$$(10) \quad d(x, y) = |x_1 - y_1| + |x_2 - y_2|$$

Deze ligt ten grondslag aan onder andere de MAE, MAPE en SMAPE. MAPE en SMAPE zijn uitdrukkingen in percentages. Welke beter zijn geschikt als criteria bij het vergelijken van verschillende dataschalen. Ter illustratie een voorbeeld: als een werkelijke waarde 100 is voor een punt uit een dataserie terwijl hiervoor 300 voorspeld is en 1000 voor een punt uit een andere dataserie waarvoor 1200 voorspeld is zal de RMSE en MAE gelijk zijn. Beide voorspellingen worden dus even goed gevonden, terwijl de eerste 200% van de werkelijke waarde af zit en de tweede

20%. Dus om te bepalen welke voorspelling beter is, moet er gekozen worden voor een uitdrukking in percentage.

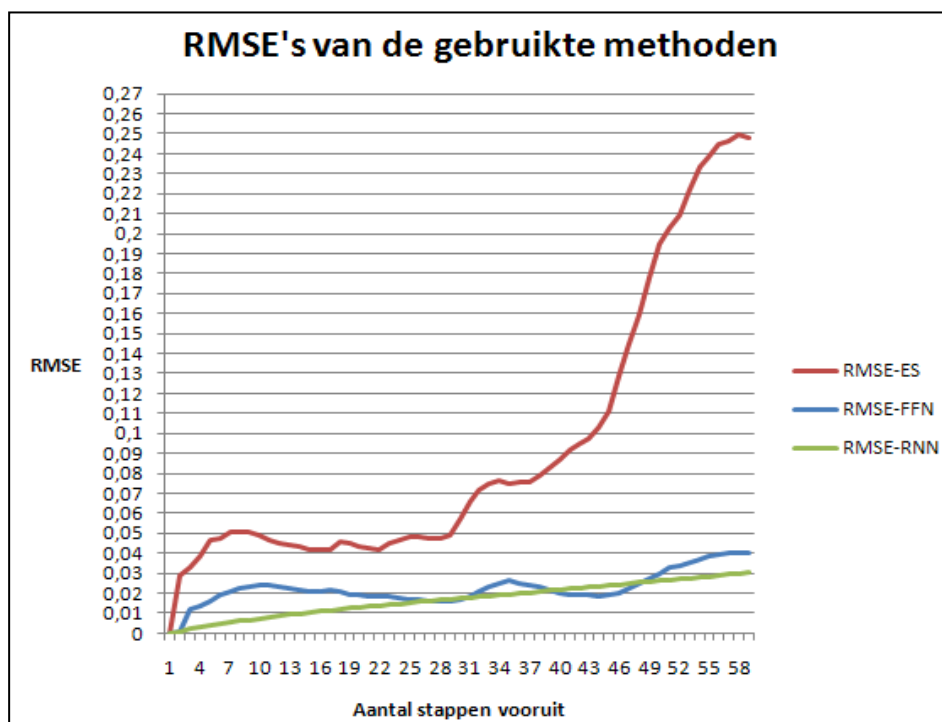
Voor dit onderzoek zal de RMSE worden gebruikt. Alle methoden worden met dezelfde datasets getest, het is dus niet nodig om een criterium te kiezen dat ontwikkeld is voor het vergelijken van verschillende series.

#### 4. Resultaten

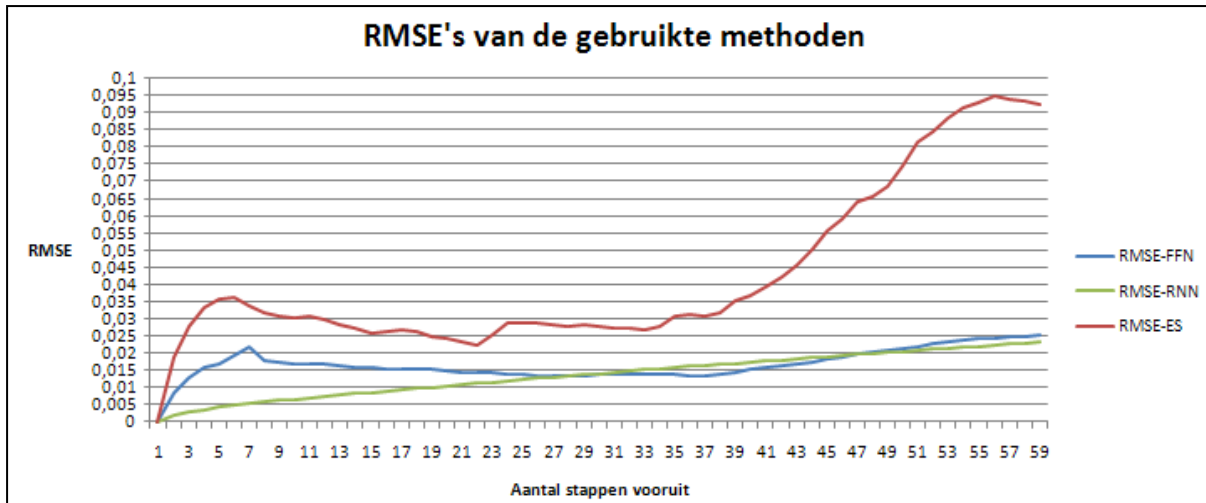
Van de resultaten van de FFN zijn alleen de uitkomsten waarbij het netwerk goed geconvergeerd is meegeteld. Goed geconvergeerd betekent dat met twee trainingruns het inputsignaal gevolgd gaat worden door het predictie-sigtaal (zie appendix c). Voor het RNN zijn alle uitkomsten meegeteld die geen chaos-output gaven (zie appendix c). Een chaos-output kan komen door een te hoge instelling van de spectral radius parameter (zie sectie 3.2.). Het gebruikte criterium hiervoor is dat alle resultaten zijn meegeteld, waarbij elke uitkomst van het RNN (op elk aantal stappen vooruit) tussen de laagste en hoogste waarde van de dataset is gebleven.

Exponential smoothing heeft met dezelfde input altijd dezelfde output.

De resultaten van de kleinere set (van 641 datapunten) staan in figuur 4.1 en de resultaten van de grotere set (van 1567 datapunten) staan in figuur 4.2. Voor beide netwerken zijn het gemiddelden van 20 resultaten.



Figuur 4.1 – Resultaten van het testen van alle methoden op de kleine dataset.



Figuur 4.2 – Resultaten van het testen van alle methoden op de grote dataset.

Er is duidelijk te zien dat op elk van de geteste stappen vooruit/criteria de netwerken beter voorspellen dan de exponential smooting. Ook is te zien dat exponential smooting en het FFN 12 stappen vooruit slechter voorspelt dan 24 stappen. Dit kan komen doordat de laatste 12 datapunten zich anders gedragen dan de 12 daarvoor. De RMSE is op een gemiddelde gebaseerd (er wordt gedeeld door het aantal test-datapunten). Dus als de datapunten 617 t/m 629 veel makkelijker zijn voor de smooting-methode en het benaderen door het FFN dalen de RMSE's. Alle methoden voorspellen slechter naarmate er verder vooruit voorspeld wordt, (na ongeveer 36 stappen) doordat er bij het testen niet meer gesuperviseerd wordt (hetgeen leidt tot een opstapeling van fouten).

Bij gemiddeld (over meerdere tests genomen) 28 van de 60 stappen convergeerde het FFN slecht op de grote dataset en op de kleine dataset waren dat er 18 op de 60. Het RNN geeft gemiddeld 16 op de 60 een chaos-output voor beide sets. Het blijkt dat hoe meer de parametrisering van de netwerken ervoor zorgt dat het netwerk slecht convergeert ofwel chaos-output geeft, des de beter voorspelt het netwerk als het wel goed gaat. Dit betekent dat het belangrijk is een goed criterium te hebben om te bepalen of een netwerk een goede voorspelling heeft gedaan of niet.

## 5. Conclusie en discussie

De reden dat neurale netwerken nog weinig worden gebruikt in de praktijk (bijvoorbeeld voor het voorspellen van wisselkoersen) is de oncontroleerbaarheid van de methode. Neurale netwerken worden op dit moment zelden in kritische applicaties gebruikt. Uit dit onderzoek blijkt dat met goede criteria voor het accepteren van voorspellingen, neurale netwerken goed gebruikt kunnen worden voor het voorspellen van wisselkoersen. Mensen die speculeren op de valuta-markt met geld dat niet een direct doel heeft, zouden moeten overwegen om af te gaan op de voorspellingen van neurale netwerken in plaats van statistische methoden. Tegenwoordig staan de spaarrente onder de 3 procent. Met iets meer risico kan

kapitaal dan beter belegd worden in valuta die in waarde gaat stijgen. Hoewel de dagelijkse fluctuaties in orde duizendste gaan, kan een goede voorspelling zorgen voor een beter rendement dan de 3 procent op lange termijn. Voor particulieren, die een minder groot kapitaal hebben is dit wellicht niet rendabel, maar voor grote banken en bedrijven zeker wel.

Hoewel er al vrij goede voorspellingen gedaan kunnen worden met neurale netwerken (fouten in orde van  $10^{-3}$ ) is het onderzoek naar neurale netwerken is nog steeds in volle gang. De meest recente onderzoeken op dit gebied gaan over spiking neurale netwerken [19,20]. De nodes in deze netwerken kunnen beter gecontroleerd worden doordat voor elke node apart het vuren bepaald kan worden. Dit zou kunnen zorgen voor preciezere en efficiëntere berekeningen voor het hele netwerk. Zoals hierboven uitgelegd is meer controle over de berekening van de voorspelling wenselijk. Dit netwerk zou nog meer lijken op biologische neurale netwerken, omdat dit model rekening houdt met een membraanpotentialiaal. Dit houdt in dat neuronen niet elke iteratie een signaal afgeven, maar alleen als het membraanpotentialiaal (gemodelleerd als differentiaalvergelijking) een bepaalde waarde heeft bereikt. Het gegeneerde signaal gaat naar de volgende neuron en verlaagt of verhoogt daar het membraanpotentialiaal. Voor meer informatie over de biologische plausibiliteit en basiskennis van dit model zie [21] .



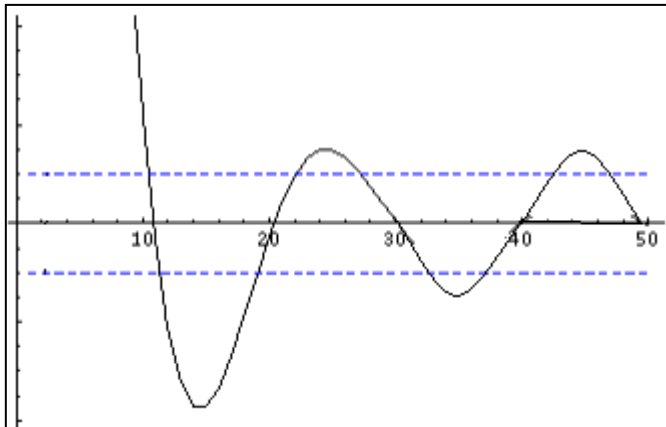
## Referenties

- [1] G. Zhang en M. Y. Hu (1998). "Neural Network Forecasting of the British Pound/US Dollar Exchange Rate". *Elvisier*: Volume 26, Issue 4, 24 August 1998, Pages 495-506.  
[doi:10.1016/S0305-0483\(98\)00003-6](https://doi.org/10.1016/S0305-0483(98)00003-6)
- [2] I. Kaastra en M. Boyd (1996). "Designing a neural network for forecasting Financial and economic time series". *Elvisier*: Volume 10, Issue 3, April 1996, Pages 215-236.  
[doi:10.1016/0925-2312\(95\)00039-9](https://doi.org/10.1016/0925-2312(95)00039-9)
- [3] G. S. da S. Gomes, T. B. Ludermir, L. M. M. R. Lima (2010) "Comparison of new activation functions in neural network for forecasting financial time series". *Neural Computing & Applications*: Volume 20, Number 3, 417-439. [DOI: 10.1007/s00521-010-0407-3](https://doi.org/10.1007/s00521-010-0407-3)
- [4] P. R. Krugman, "Target Zones and Exchange Rate Dynamics", *The Quarterly Journal of Economics* Vol. 106, No. 3 (Aug., 1991), pp. 669-682. <http://www.jstor.org/stable/2937922>
- [5] K. Funahashi (1989), "On the approximate realization of continuous mappings by neural networks", *Neural Networks*, Volume 2, Issue 3, 1989, Pages 183-192.  
[doi:10.1016/0893-6080\(89\)90003-8](https://doi.org/10.1016/0893-6080(89)90003-8)
- [6] G. E. P. Box en G. M. Jenkins (1968). "Some Recent Advances in Forecasting and Control". *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, Vol. 17, No. 2, pp. 91-109.  
<http://www.jstor.org/stable/2985674>
- [7] R. A. Meese and K. Rogoff (1983), "Empirical exchange rate models of the seventies: Do they fit out of sample?". *Journal of International Economics* 14 (1983), pp. 3-24.  
[doi:10.1016/0022-1996\(83\)90017-X](https://doi.org/10.1016/0022-1996(83)90017-X)
- [8] S. Narayan (1997), "The generalized sigmoid activation function: Competitive supervised learning", *Information Sciences*, Volume 99, Issues 1-2, June 1997, Pages 69-82.  
[doi:10.1016/S0020-0255\(96\)00200-9](https://doi.org/10.1016/S0020-0255(96)00200-9)
- [9] D.E. Rumelhart, G.E. Hinton en R.J. Williams (1986). "Learning internal Representations by error propagation". [http://www-psych.stanford.edu/~jlm/papers/PDP/Volume%201/Chap8\\_PDP86.pdf](http://www-psych.stanford.edu/~jlm/papers/PDP/Volume%201/Chap8_PDP86.pdf)
- [10] F. Wyffels, B. Schrauwen (2010), "A comparative study of Reservoir Computing strategies for monthly time series prediction", *Neurocomputing*, Volume 73, Issues 10-12, Subspace Learning / Selected papers from the European Symposium on Time Series Prediction, June 2010, Pages 1958-1964. [doi:10.1016/j.neucom.2010.01.016](https://doi.org/10.1016/j.neucom.2010.01.016)
- [11] G. Dematos, M. S. Boyd, B. Kermanshahi, N. Kohzadi, I. Kaastra (1996), "Feedforward versus recurrent neural networks for forecasting monthly japanese yen exchange rates". *Asia-Pacific Financial Markets* 1996-02-01, Volume 3, Issue 1, Pages 59-75  
[Doi: 10.1007/BF00868008](https://doi.org/10.1007/BF00868008)
- [12] M. Bodén (2001), "A guide to recurrent neural networks and backpropagation".  
[http://itee.uq.edu.au/~mikael/papers/rn\\_dallas.pdf](http://itee.uq.edu.au/~mikael/papers/rn_dallas.pdf)

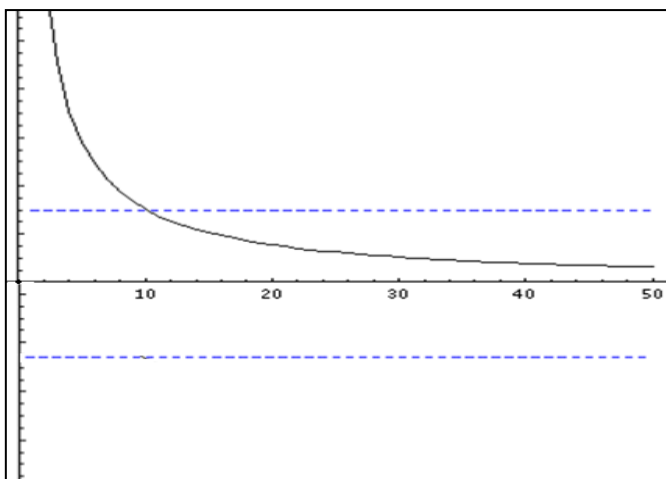
- [13] M. Lukoševičius, H. Jaeger (2009), "Reservoir computing approaches to recurrent neural network training", *Computer Science Review*, Volume 3, Issue 3, August 2009, Pages 127-149. [doi:10.1016/j.cosrev.2009.03.005](https://doi.org/10.1016/j.cosrev.2009.03.005)
- [14] M. Shanker, M. Y. Hu, M. S. Hung, Effect of data standardization on neural network training, *Omega*, Volume 24, Issue 4, August 1996, Pages 385-397. [doi:10.1016/0305-0483\(96\)00010-2](https://doi.org/10.1016/0305-0483(96)00010-2)
- [15] K. Hagiwara, K. Fukumizu, "Relation between weight size and degree of over-fitting in neural network regression", *Neural Networks*, Volume 21, Issue 1, January 2008, Pages 48-58. [doi:10.1016/j.neunet.2007.11.001](https://doi.org/10.1016/j.neunet.2007.11.001)
- [16] Ji-Hong Wang, Jian-Hui Jiang, Ru-Qin Yu, "Robust back propagation algorithm as a chemometric tool to prevent the overfitting to outliers". *Chemometrics and Intelligent Laboratory Systems*, Volume 34, Issue 1, August 1996, Pages 109-115. [doi:10.1016/0169-7439\(96\)00005-6](https://doi.org/10.1016/0169-7439(96)00005-6)
- [17] R. Caruana, S. Lawrence, L. Giles, "Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping". In *Proceedings of Neural Information Processing Systems Conference*, Denver, CO, USA, November 28-30, 2000; pp. 402-408.
- [18] Jaeger, H. (2001). "The "echo state" approach to analysing and training recurrent neural networks". GMD report no. 148. GMD—German National Research Institute for Computer Science. <http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf>
- [19] J. Touboul, O. Faugeras, "A Markovian event-based framework for stochastic spiking neural networks". *NeuroMathComp Laboratory*, INRIA, Sophia Antipolis, France February 21, 2011. [arXiv:0911.3462v2](https://arxiv.org/abs/0911.3462v2)
- [20] A. K. Vidybida, "Testing of information condensation in a model reverberating spiking neural network". *International Journal of Neural Systems (IJNS)*, Volume: 21, Issue: 3 (June 2011), Page: 187-198. Doi: 10.1142/S0129065711002742
- [21] W. Gerstner, W. M. Kistler (2005), "Spiking Neuron Models Single Neurons, Populations, Plasticity". ISBN: 9780511075063. <http://icwww.epfl.ch/~gerstner/SPNM/SPNM.html>

## Appendix A:

Figuur A.1 laat een functie zien die convergeert naar de nullijn. Zonder learning rate kan het zo zijn dat de nullijn bij elke iteratie te veel wordt overschat, wat correspondeert in een grafiek met golven met een grote amplitude (hoogte). Het kan zelfs voorkomen dat de benadering niet meer dichterbij de nullijn kan komen, dan we eigenlijk zouden willen (bijvoorbeeld de blauwe stippellijn). In figuur A.2 is een functie weergegeven die correspondeert met een benadering met een (lage) leerconstante. De nullijn wordt meer gelijkelijk benaderd en wordt niet overschat.

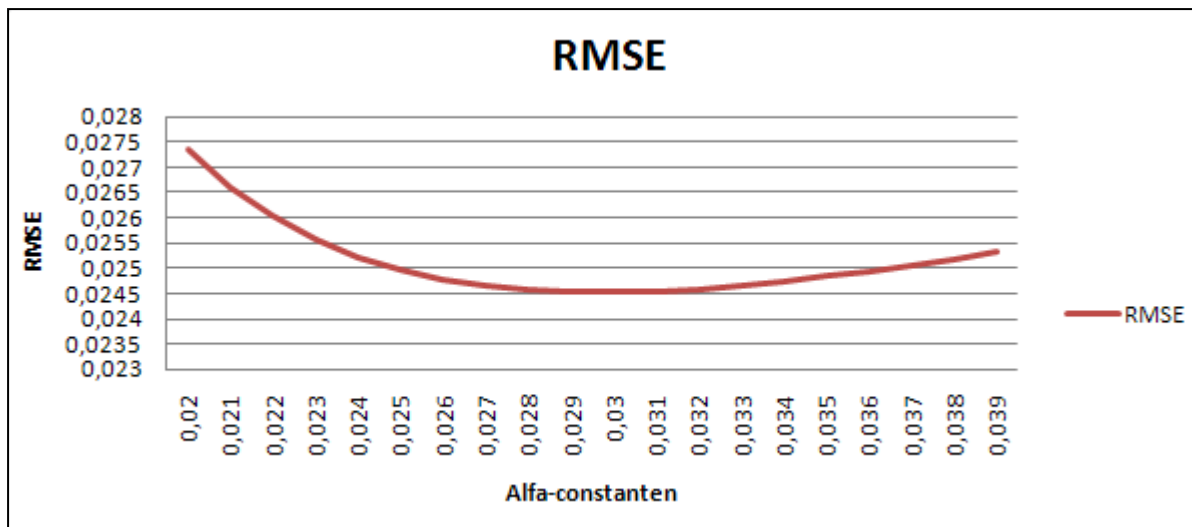


Figuur A.1 – Een voorbeeld van convergentie (naar de nullijn), welke op gegeven moment nauwelijks meer dichterbij de nullijn komt.

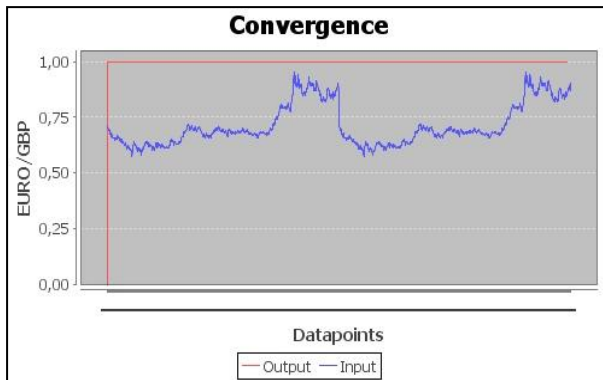


Figuur A.2 – Een voorbeeld van een benadering met leerconstante.

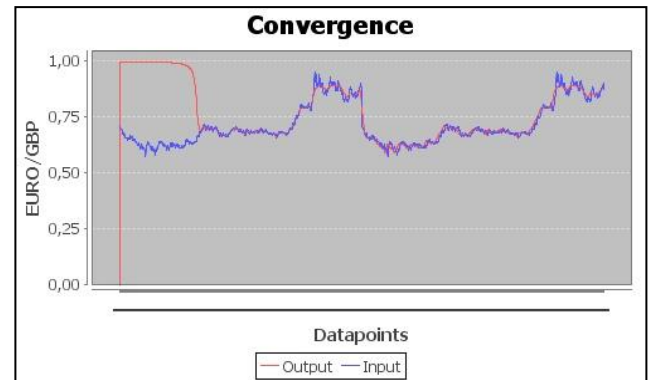
## Appendix B:



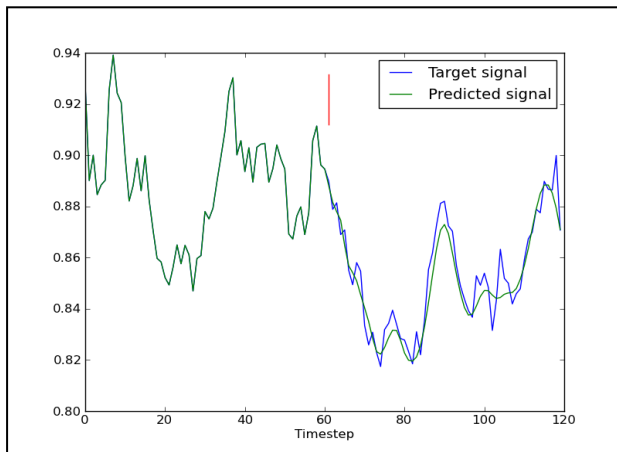
## Appendix C:



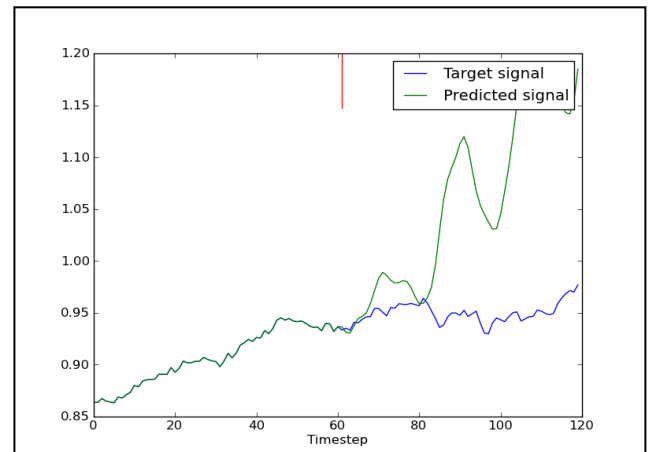
Figuur C.1 – Niet geconvergeerd FFN. De rode lijn is de output van het netwerk en blauw de input.



Figuur C.2 – Goed geconvergeerd FFN.



Figuur C.3 – RNN zonder chaos-output.



Figuur C.4 – RNN met chaos-output.