

Bachelor Project

Neuraal Winkelen

Andrew Li

Universiteit Leiden

Leiden Institute of Advanced Computer Science

Leiden, 2008

Samenvatting

Dit verslag is een bachelorproject op het gebied van kunstmatige intelligentie. Bij het afronden van de bachelorstudie Informatica hoor ik een bachelorproject te maken. Tijdens het college “Kunstmatige Intelligentie” verzorgd door Dr. W.A. Kusters raakte ik geïnteresseerd in dit onderwerp. Ik heb daarom besloten om mijn bachelorproject te doen op het gebied van kunstmatige intelligentie.

Een database met aankooptransacties van klanten kan zeer waardevolle informatie bevatten. Een voorbeeld daarvan is hoe vaak bepaalde producten of combinaties van producten worden gekocht. Maar dat gaat niet altijd even makkelijk. Als er een grote hoeveelheid data verwerkt moet worden of de winkel heeft een groot assortiment producten, dan kan de efficiëntie een groot probleem zijn.

Dit onderzoek probeert de frequentie van elke willekeurige itemset snel uit te rekenen door middel van neurale netwerken. Wanneer het aantal variabelen laag is, dan is het mogelijk om de frequenties van alle itemsets elke keer opnieuw uit te rekenen en/of op te slaan. Maar wanneer er veel variabelen zijn, dan is het gebruik van deze methode niet te doen. Bijvoorbeeld een supermarkt heeft al gauw 1000 producten en dus 1000 variabelen. Dit onderzoek probeert dit probleem op te lossen met behulp van neurale netwerken.

Sleutelwoorden: neurale netwerken, dataset, training, frequente itemsets

Inhoudsopgave

Samenvatting.....	1
Inhoudsopgave	2
Lijst van figuren.....	3
Lijst van tabellen.....	3
1. Introductie.....	4
2. Theorie	6
3. De datagenerator	8
4. Het DF algoritme	10
5. Het Neurale Netwerk	12
6. Evalueer methode Root Mean Squared Error (RMSE).....	15
7. Een experiment uitvoeren	16
8. Experimenten	18
9. Conclusies	20
10. Dankbetuiging.....	23
11. Referenties	24
12. Bijlagen.....	25

Lijst van figuren

Figuur 1 Neuraal Netwerk (Russell & Norvig 2003).....	5
Figuur 2 window-size=10, 14 verborgen knopen, learning-rate=0,2	18
Figuur 3 Test 44 met 14 verborgen knopen en window-size=1	21
Figuur 4 Test 51 met 21 verborgen knopen en window-size=1	21
Figuur 5 Test 52 na 4000 transacties en window-size=1	22
Figuur 6 Test 33 met window-size=5, verborgenknopen=14 en init gewicht=0.0001	22

Lijst van tabellen

Tabel 1 Lijst van uitgevoerde testen.	25
Tabel 2 Gebruikte databases	27

1. Introductie

Wetenschappers hebben allang ontdekt om complexe vraagstukken op te lossen met behulp van kunstmatige intelligentie.

In dit onderzoek willen we met behulp van een neurale netwerk (Russell & Norvig 2003, hoofdstuk 20) berekenen hoe vaak verschillende combinaties van aankopen voorkomen. Een aankoop van een klant noemen we een transactie. De transacties worden allemaal opgeslagen in een bestand(en) dat een dataset wordt genoemd. Het neurale netwerk wordt getraind met deze dataset. Uiteindelijk moet het netwerk kunnen berekenen hoe vaak een combinatie van producten wordt gekocht. Een combinatie van producten (voortaan een itemset genoemd) kan bijvoorbeeld zijn {ei, brood} of {melk, brood, pindakaas}. Zo'n combinatie noemen we dus een itemset. De frequentie van itemset $I = \{ei, brood\}$ is hoe vaak I wordt gekocht (als fractie van alle transacties). De frequentie van I wordt ook wel de *Support* van I genoemd. Snel en accuraat berekenen van de *Support* kan erg nuttig zijn, want vanuit het oogpunt van een manager van een bedrijf is het erg interessant om te weten hoe vaak een bepaalde itemset voorkomt en wat de relaties zijn tussen de verschillende producten. Dit geeft de manager extra mogelijkheden, zoals het verkoop maximaliseren of optimaal bestellen zodat er zo min mogelijk hoeft worden uitgegeven aan magazijnkosten.

Als het aantal variabelen (in dit geval het aantal producten in de supermarkt) laag is, kunnen we alle itemsets tellen met een bruteforce methode. Wanneer er echter veel variabelen zijn is dit niet zo eenvoudig. Stel de supermarkt heeft 3 producten: product A , B en C . De mogelijke itemsets zijn:

X (betekent geen, dus niets gekocht, ofwel de lege verzameling)

A

B

C

AB

AC

BC

ABC

Bij het aantal producten gelijk aan 3 is het aantal mogelijk itemsets gelijk aan 8 ($=2^3$). De combinatie AB en BA is dezelfde, zo ook voor AB en AAB , het gaat namelijk om verzamelingen. Als het aantal producten groter wordt, groeit het aantal mogelijke itemsets ook exponentieel, namelijk 2^n , voor n producten. Een supermarkt met slechts 100 producten heeft dus al 2^{100} mogelijke *itemsets*. Het is niet mogelijk elke keer de hele database door te lezen om de support van een itemset te berekenen. De frequenties van alle itemsets in een bestand opslaan, is ook niet economisch haalbaar voor een supermarkt. Om dit probleem efficiënt op te lossen zijn allerlei algoritmen ontwikkeld, de meeste gebaseerd op APRIORI en op FP-trees. Zie verder FIMI03 en FIMI04 (Frequent Itemset Mining Implementations Repository 2003 en 2004).

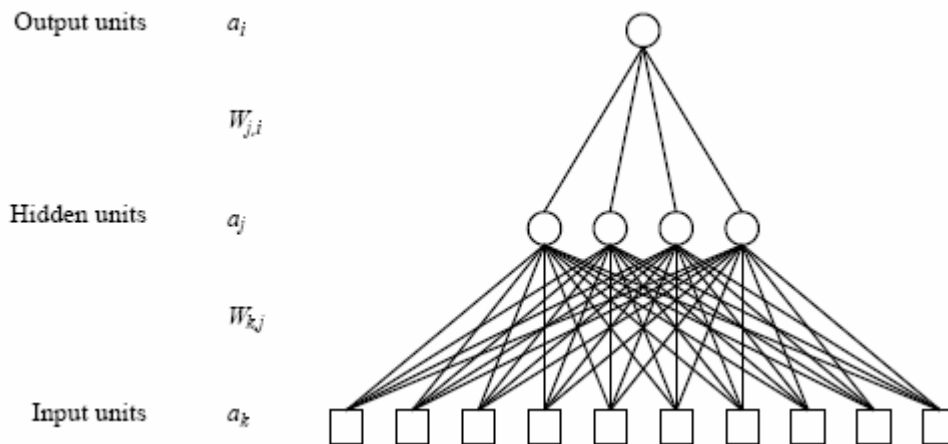
In dit project ga ik onderzoeken of het mogelijk is om met behulp van neurale netwerken snel de *support* van alle itemsets uit te rekenen. Het gebruik van het neurale netwerk om dit probleem op te lossen kent verschillende voordelen:

- Een getraind netwerk kan de “support” van elke willekeurige itemset snel berekenen (bij benadering).
- Het netwerk kan on-the-fly aangepast worden tijdens openingstijden van een winkel zonder dat het netwerk helemaal opnieuw hoeft te rekenen en zonder dat het gebruik van het netwerk wordt onderbroken.
- Het kan gebruikt worden om associatieregels (relaties tussen producten) te ontdekken.
- Het netwerk kan een gedeeltelijke representatie van de originele database weergeven en kan dus gebruikt worden om de originele database beter te begrijpen.

Het gebruik van neurale netwerken kent ook nadelen:

- Trainen van het neurale netwerk kan tijdrovend zijn.
- De optimale instelling voor het neurale netwerk is van te voren niet met zekerheid te zeggen. Het kan dus mogelijk zijn dat er meerdere keren getest moet worden om een goede instelling te vinden.
- Het is een black-box systeem, wat wellicht moeilijk te begrijpen is.

In plaats van het wiel opnieuw uit te vinden, maakt dit onderzoek gebruik van een bestaand neurale netwerk. Het neurale netwerk ziet er als volgt uit:



Figuur 1 Neuraal Netwerk (Russell & Norvig 2003)

Een neuraal netwerk bestaat in het algemeen uit drie lagen: de invoerlaag, de verborgen laag en de uitvoerlaag. In de bovenstaande figuur worden knopen in deze lagen respectievelijk weergegeven door a_k , a_j en a_i . De drie lagen zijn onderling meestal volledig verbonden. W_{kj} is het gewicht van de tak tussen invoerknoop k en verborgen knoop j . W_{ji} is het gewicht van de tak tussen verborgen knoop j en uitvoerknoop i .

2. Theorie

De invoerknoppen van de invoerlaag van het neurale netwerk vertegenwoordigen de verschillende producten in een supermarkt. De uitvoer van het neurale netwerk is de frequentie van het voorkomen van een bepaalde itemset. In het voorbeeld van 3 producten, als iemand wil weten hoe vaak product A en C samen worden verkocht, dan geeft hij 101 als invoer aan het neurale netwerk (1 betekent aanwezig, 0 betekent afwezig). Dus de eerste en de derde invoerknoop hebben als invoerwaarde 1. Omdat product B niet aanwezig is in de itemset $\{A, C\}$, heeft de tweede invoerknoop waarde 0. Het neurale netwerk geeft aan wat de frequentie (s) is voor itemset $\{A, C\}$. Als de frequentie van itemset AC (we noteren AC voor $\{A, C\}$) gelijk is aan 0,3 dan bevat 30% van alle transacties tegelijk product A en product C . Omdat AC een subset is van ABC , worden transacties die ABC bevatten ook meegerekend als voorkomen van itemset AC . De frequentie van het voorkomen van itemset AC in een database van transacties wordt ook genoemd de *Support* van AC , zoals eerder gedefinieerd.

$\text{Supp}(\{A, C\}) = 0,3 = \text{aantal transacties met } AC \text{ als fractie van het totale aantal transacties.}$

Hoe benaderen we de support van alle mogelijke combinaties en hoe wordt het netwerk geleerd?

We noteren:

I = een itemset

c = het tot nu toe verwerkte aantal klanten of transacties

s = de support van itemset I tot en met de c -de klant of transactie

s' = de support van I tot en met de $c+1$ -de klant of transactie

De $c+1$ -de klant kan een aankoop doen die itemset I bevat of juist niet. Als de $c+1$ -de klant I wel bevat (de klant heeft dus een verzameling van producten gekocht die subset I bevat, met andere woorden I is een deelverzameling van deze transactie), dan geldt de volgende leerformule om de nieuwe support, s' , voor I te berekenen:

$$s' = \frac{s \cdot c + 1}{c + 1} = \frac{s + 1/c}{1 + 1/c} \geq s$$

Als de transactie niet de itemset I bevat, dan geldt de volgende formule:

$$s' = \frac{s \cdot c}{c + 1} = \frac{s}{1 + 1/c} \leq s$$

Verder kan het netwerk te werk gaan met time-windows. Het is mogelijk om de target waarden (van de twee bovenstaande formules) te veranderen naar $\frac{s+\epsilon}{1+\epsilon}$ en $\frac{s}{1+\epsilon}$ respectievelijk, voor een kleine waarde $\epsilon > 0$. Merk op dat:

$$s \leq \frac{s+\epsilon}{1+\epsilon} \leq 1 \quad \text{en} \quad 0 \leq \frac{s}{1+\epsilon} \leq s$$

Veronderstel dat een *window* ter grootte $w \geq 1$ is gegeven. We nemen aan dat de *support* voor de laatste w klanten s was. En we nemen aan dat de kans dat de eerste van deze klanten (of elk van hen) I bevat, s is.

Dan kunnen we zeggen, als de nieuwe transactie wel I bevat dan kunnen we s' benaderen met:

$$s' = s \cdot s + (1 - s) \cdot \frac{s \cdot w + 1}{w} = s \left(1 - \frac{1}{w}\right) + \frac{1}{w} \geq s$$

Als de nieuwe transactie geen I bevat, dan gebruiken we de volgende formule:

$$s' = s \cdot \frac{s \cdot w - 1}{w} + (1 - s) \cdot s = s \left(1 - \frac{1}{w}\right) \leq s$$

Of gecombineerd:

$$s' = s \left(1 - \frac{1}{w}\right) + \delta \frac{1}{w}$$

Met $\delta = 1$ als de volgende klant I bevat, anders 0. Als we voor $w = 1 + (1/\epsilon)$ nemen, dan krijgen we weer dezelfde formule die we eerder hebben gezien.

3. De datagenerator

Dit onderzoek vereist een grote set aan data om het neurale netwerk goed te laten trainen. Om een grote dataset te genereren, wordt er daarom een datagenerator gemaakt in de vorm van een C++ programma `gen.cc`. De gebruiker kan aangeven hoeveel data gegenereerd moet worden, uit hoeveel items de dataset bestaat en de gemiddelde lengte van de dataset (gemiddeld aantal items per transactie/sample). Met deze generator kan je ook aangeven hoeveel procent van de dataset een bepaalde itemset bevat. Na het runnen van de generator worden er twee datasets (`test.nn` en `test.df`) gegenereerd met de aangegeven parameters. De twee datasets verschillen slechts in formaat en bevatten dezelfde transacties. De reden hiervoor en het gebruik hiervan wordt later uitgelegd.

Hoe werkt de generator?

De generator moet eerst gecompileerd worden met

```
make all
```

Met het volgende commando kan de generator gestart worden:

```
./gen [no_items] [samples_count] [filename] [avg_len] [percentage] [item[0]...[n]]
```

Waarbij geldt:

[no_items] : stelt het aantal producten voor.

[samples_count] : het aantal transacties dat door de generator gegenereerd moet worden.

[filename] : bestandsnaam waar naartoe gegenereerde data worden geschreven.

[avg_len] : gemiddeld aantal producten per transactie.

[percentage] : Deze parameter is optioneel en moet samen gebruikt worden met de volgende parameter, *[item[0]...[n]]*. Deze geeft ongeveer aan hoeveel procent van alle transacties de set *[item[0]...[n]]* als subset heeft.

[item[0]...[n]] : definieert een itemset die *[percentage]* procent moet voorkomen als subsets van transacties. Voorbeeld: `[1 3 6]` staat voor $\{item1, item3, item6\}$.

Als er geen parameters worden doorgegeven, dus opstarten met het commando “`./gen`”, worden er default waarden gebruikt. De waarden kunnen aangepast worden in de code (`gen.cc`). De parameters die aangepast kunnen worden zijn:

```
// if no parameter given, these values will be used
#define NO_ITEMS 7
#define SAMPLES_COUNT 40
#define FILENAME "test.nn"
#define FILENAME2 "test.df"
#define AVG_LEN 3
#define PERCENT 50
#define MAX_SET_LEN 100
...
///// make the itemset changes here /////
set[0] = 7;          /////
set[1] = 6;          /////
```

```

set[2] = 3;          //
set[3] = 4;          //
set[4] = 5;          //
set_len = 5;        //
////////////////////

```

De werking van beide manieren van opstarten is dezelfde. De keuze is aan de gebruiker.

Een voorbeeld dataset genereren, gaat als volgt:

```
./gen 4 4 test.nn 2 50 1 3
```

Hiermee wordt een dataset gegenereerd bestaande uit 4 transacties, 4 producten, met een gemiddelde transactielengte 2 en 50% van de transacties (dus 2 transacties) hebben de set {1,3} als subset. De data wordt geschreven naar de bestanden *test.nn* en *test.df*.

Een mogelijke dataset gegenereerd uit het bovenstaande commando :

<i>test.nn</i>		<i>test.df</i>
0 0 1 1	EN	1
0 0 3 1 2 3		1 2 3
0 0 1 3		3
0 0 3 1 3 4		1 3 4

Er worden twee datasets aangemaakt met als enige verschil dat bij de ene dataset alle transacties beginnen met twee nullen gevolgd door de transactielengte. *test.nn* wordt gebruikt door het neurale netwerk en *test.df* wordt gebruikt voor het *DF*-algoritme. Dit heeft met het formaat voor het neurale netwerk te maken en speelt verder geen rol met betrekking tot de werking.

Als we naar de dataset kijken, kunnen we zien dat subset {1,3} inderdaad in 50% van de transacties voorkomt. En de gemiddelde lengte van alle transacties is (ongeveer) gelijk aan 2 (zonder de 2 nullen aan het begin van elke regel).

4. Het DF algoritme

Om de nauwkeurigheid van het neurale netwerk te kunnen verifiëren hebben we de werkelijke frequenties van de itemsets nodig. Hiervoor gebruiken we de depth first implementatie **DF** van APRIORI, *current.cc*, (Kosters en Pijls 2004). De uitvoer van de resultaten staat niet op de gewenste volgorde.

Het DF algoritme is één van de snelste data-mining algoritmen om alle frequente itemsets in een grote database te zoeken. De gegeven database wordt eerst in het geheugen geladen. Er wordt een boom in het geheugen geconstrueerd met alle frequente itemsets. Frequente itemsets zijn alle set met een support groter dan of gelijk aan *minsup*; *minsup* is een van te voren gegeven waarde. De boom wordt opgebouwd door de items één voor één toe te voegen aan de boom. Elk pad in de boom correspondeert met een unieke frequente itemset. Zie verder (Kosters en Pijls, 2004).

Een voorbeeld van uitvoer van het programma is:

```
(800)
2 (297)
2 4 (123)
2 4 1 (63)
2 4 1 5 (42)
2 4 1 5 3 (30)
2 4 1 5 3 7 (26)
2 4 1 5 3 7 6 (25)
2 4 1 5 3 6 (28)
2 4 1 5 7 (32)
.....
```

Per regel staat een itemset gevolgd door een getal tussen haakjes waarin staat aangegeven hoeveel keer die itemset voorkomt in de dataset. Hier kunnen we zien dat itemset {2, 4, 1, 5, 7} 32 keer voorkomt, terwijl itemset \emptyset (de lege verzameling) 800 keer voorkomt: Hieruit kunnen we concluderen dat de dataset uit 800 transacties bestaat. Dat wil zeggen 800 klanten (transacties) hebben ook \emptyset gekocht.

Omdat de uitkomst van de berekeningen niet op de gewenste volgorde wordt afgedrukt, is het vergelijken met de uitkomst van het neurale netwerk erg lastig. Het C++ programma *arrange.cc* kan de resultaten op een gewenste volgorde plaatsen: *arrange.cc* leest de uitvoer van DF in. De niet voorkomende itemsets worden bijgevoegd en krijgen support 0. Dan worden de resultaten op lexicografische volgorde afgedrukt, waarbij als laatste de itemset met alle producten komt. Zo kunnen we makkelijk de uitvoer van het neurale netwerk vergelijken met de werkelijke frequenties.

Hoe gebruik je *current.cc* en *arrange.cc*?

Compileer met:

make all

Wijzig de naam van de dataset in *test.df*. Gebruik hier de dataset waarbij alle transacties in de dataset niet begint met “0 0”.

Run het commando:

```
./current test.df 0 test.out
```

En vervolgens het commando:

```
./arrange [no_items] test.out test.sort
```

Waarbij *[no_items]* het totale aantal verschillende producten.

Het resultaat dat we nodig hebben, staat nu in *test.sort*. In dit bestand staan de berekende frequenties. Die hebben we nodig om te vergelijken met de uitkomst van het neurale netwerk.

5. Het Neurale Netwerk

Voor het neurale netwerk gebruiken we een Fast Artificial Neural Network (fann) (Steffen Nissen 2003 en freegoldbar 2004) implementatie die gebruik maakt van een neuraal netwerk library (Daniel Franklin 1998). Daarna heeft Dr. Barzan Mozafari het neurale netwerk aangepast voor dit onderzoek. Het netwerk gebruikt backpropagation als leermethode om de gewichten van de takken te optimaliseren.

In dit onderzoek wordt telkens één variabele veranderd en de rest van de parameters constant gehouden. In het neurale netwerk programma kunnen we de volgende variabelen variëren:

- De “connectionsrate”
- Window size
- Learning rate (leer snelheid)
- Aantal verborgen knopen

Connectionrate :

Hiermee kunnen we bepalen hoeveelste deel van de takken tussen verschillende neuronen een netwerk bestaat ten opzichte van een volledig verbonden neurale netwerk. Een volledig verbonden neurale netwerk heeft een *connectionrate* gelijk aan 1. En een netwerk met *connectionrate* = 0,5 heeft half zoveel verbindingen tussen de knopen.

Learning rate:

Hiermee kunnen we bepalen hoe snel een netwerk leert richting de optimale waarde. Een grotere leersnelheid kan het aantal berekeningen reduceren, maar kan misschien ook de balans van de gewichten van het netwerk verstoren. Voor *learningrate* = 1, probeert het neurale netwerk zijn gewichten zodanig aan te passen zodat het netwerk een uitvoerwaarde heeft die gelijk is aan de doelwaarde.

Aantal verborgen knopen:

Meer verborgen knopen in het neurale netwerk geeft de mogelijkheid om meer complexe problemen op te lossen. Meer verborgen knopen betekent ook dat het neurale netwerk meer berekeningen moet uitvoeren.

Hier volgt de pseudocode om een beeld te geven hoe het netwerk werkt.

Main :

```
lees_in_programma_parameters();
initialiseren_neuraal_netwerk();
while dataset niet leeg
    window_nummer = 1;
    for window 1 to windows_size
        lees_een_transactie_uit_dataset(transactie);
        train_netwerk_v1(transactie, window_nummer, window_size);
```

```

        window_nummer++;
    end for
end while
print_voorspelling_alle_itemset();

```

Bij elke iteratie van de while-loop wordt een transactie uit de database gelezen. Met *window_nummer* en *window_size* wordt de nieuwe support van alle itemsets berekend. Het netwerk past zijn gewichten aan, door middel van backpropagation, voor elke itemset. Dit wordt bereikt door voor elke itemset het netwerk te trainen zolang nodig is totdat het neurale netwerk een support uitvoert dat ongeveer gelijk is aan de berekende support voor die itemset. Als het netwerk genoeg heeft getraind voor deze itemset wordt het proces herhaald voor de volgende itemset totdat het netwerk voor alle itemset heeft getraind.

Een voorbeeld van een transactie die ingelezen wordt :

0 0 2 4 6 7 8

Dit is een transactie waarbij product nummer 2, 4, 6, 7 en 8 gekocht wordt. De twee nullen aan het begin van de transactie worden door het netwerk genegeerd en hebben verder geen invloed op de werking van het neurale netwerk.

Train_netwerk_v1(transactie,window_nummer,window_size) :

```

for itemset_num = 1 to 2^no_items // doe voor alle mogelijke itemsets het volgende
    if itemset_num is subset_van_transactie then
        update_gewichten (itemset_num, (sup*(sofar-1)+1)/sofar);
    else
        update_gewichten (itemset_num, (sup*(sofar-1))/sofar);
    fi
end for

```

N.B. : *sofar* is het totale aantal tot nu toe gelezen transacties. *sup* is de oude support van de huidige itemset.

update_gewichten (itemset_num, nieuw_sup) :

```

while (abs(support_netwerk - nieuw_sup_berekend) > desired_error)
    back_propagation();
end while

```

Zolang de door het netwerk berekende waarde en de werkelijke waarde meer dan *desired_error* verschil blijft het netwerk zijn gewichten aanpassen door middel van de backpropagation methode.

Dit algoritme traint voor alle transacties de verandering van de support van alle itemsets. Dit kan leiden tot een lange trainingstijd van het neurale netwerk. Om de trainingstijd te beperken en de efficiëntie te verhogen is er een verstandig algoritme nodig om het werk van het neurale netwerk te verminderen. Een mogelijkheid hiervoor is bijvoorbeeld zoveel mogelijk onnodige of minder belangrijke trainingen te vermijden. Met hier boven besproken trainingmethode proberen we elke transactie die het neurale netwerk inleest te

vergelijken met alle mogelijke itemsets en daarna voor elke itemsets de nieuwe support te trainen. Als de itemset een subset is van de ingelezen transactie, dan gebruiken we de formule die de support verhoogt (de positieve formule), anders gebruiken we de andere formule (de negatieve formule). Zie Hoofdstuk 2 voor de formules.

We passen de conditie aan wanneer het netwerk moet trainen en met welke formule. De volgende aanpassingen worden aangebracht voor de nieuwe trainingsmethode. Voor elke transactie die het neurale netwerk inleest, wordt vergeleken met alle mogelijke itemsets. Als een itemset een subset is van de huidige transactie dan wordt het neurale netwerk getraind met de positieve formule. Als een itemset geen subset is van de ingelezen transactie, maar wel gemeenschappelijke items bevat met de transactie dan kan er zowel een positief als een negatief effect ontstaan voor de support. In dit geval wordt het neurale netwerk niet getraind. Voor de rest van de itemsets wordt het neurale netwerk getraind met de negatieve formule.

Stel er zijn in totaal m producten in voorraad. Gemiddeld koop een klant n producten. Dan moet er per transactie 2^n keren getraind worden met de positieve formule en 2^{m-n} keren met de negatieve formule. Het aantal itemsets die niet worden getraind is $2^m - 2^n - 2^{m-n}$. Als gemiddeld elke klant maar slechts een klein aantal producten koopt uit een grote assortiment van producten dan kan er een ongebalanceerde netwerktraining voorkomen. Omdat de verzameling van itemset die getraind moet worden met positieve formule veel kleiner is dan de verzameling van itemsets die getraind moet worden met negatieve formule. En erger nog is de verzameling van itemset die niet getraind wordt veel groter dan de twee andere verzameling te samen. Er is dus een compromis tussen nauwkeurigheid en snelheid.

Het algoritme met gereduceerde training ziet er als volgt uit.

Train_netwerk_v2(transactie,window_nummer,window_size):

```
for itemset_num = 1 to 2^no_items // doe voor alle mogelijke itemsets het volgende
  if itemset_num = subset_van_transactie then
    update_gewichten (itemset_num, (sup*(sofar-1)+1)/sofar);
  else if doorsnede_itemset_en_transactie is leeg
    update_gewichten (itemset_num, (sup*(sofar-1))/sofar);
  else
    /* doe niets, bespaar werk */
  fi
end for
```

Om dit algoritme te gebruiken moet in *main.cc* het volgende aangepast worden:

```
#define TRAININGVERSIE 1
veranderen in:
#define TRAININGVERSIE 2
```

6. Evalueer methode Root Mean Squared Error (RMSE)

De voorspelling van het neurale netwerk gaan we evalueren en vergelijken met de werkelijke (berekende) waarden. Hiervoor gebruiken we de RMSE (Root Mean Squared Error), ook bekend als RMSD (Root Mean Square Deviation). Dit is een vaak gebruikte meetmethode om het verschil van een voorspelde of benaderde waarde van een model met de waarde die werkelijk wordt geobserveerd te vergelijken.

Stel Θ_1 is de geobserveerde waarde en Θ_2 is de voorspelwaarde.

$$\Theta_1 = \begin{bmatrix} X_{1,1} \\ X_{1,2} \\ \vdots \\ X_{1,n} \end{bmatrix} \quad \text{en} \quad \Theta_2 = \begin{bmatrix} X_{2,1} \\ X_{2,2} \\ \vdots \\ X_{2,n} \end{bmatrix}$$

Dan kunnen we de RMSE van Θ_1 en Θ_2 berekenen met de volgende formule:

$$RMSE(\Theta_1, \Theta_2) = \sqrt{MSE(\Theta_1, \Theta_2)} = \sqrt{E((\Theta_1 - \Theta_2)^2)} = \sqrt{\frac{\sum_{i=1}^n (X_{1,i} - X_{2,i})^2}{n}}$$

Een RMSE van nul, betekent dat de voorspelling een perfecte voorspelling is. RMSE-waarden anders dan nul hebben op zich zelf weinig betekenis. Het wordt voornamelijk gebruikt om de nauwkeurigheid tussen verschillende modellen te vergelijken. Een model met kleinere RMSE-waarde is beter dan een model met grotere RMSE-waarde.

7. Een experiment uitvoeren

Nu we alle benodigdheden hebben, kunnen we de experimenten uitvoeren. Hoe kunnen we een experiment uitvoeren? Stel we willen een experiment uitvoeren met een dataset van 800 transacties, 7 items (producten), gemiddelde lengte van de transacties is 3. En het neurale netwerk heeft één verborgen laag, 7 verborgen knopen, *connectionrate* = 1, en een window-size van 5.

Hoe moeten we nu te werk gaan? Laten we eerst de dataset creëren. Gebruik het commando.

```
./gen 7 800 test.nn 3
```

We hebben nu twee datasets met zelfde transacties: *test.nn* is voor het neurale netwerk en *test.df* is voor DF om de werkelijke support te berekenen.

Executeer vervolgens de volgende commando's.

```
./current test.df 0 test.out  
./arrange 7 test.out test.sort
```

Nu hebben we de werkelijke waarden voor de support van alle itemsets in *test.sort*. Dit bestand gebruiken we later om te vergelijken met de uitkomst van het neurale netwerk.

Voordat we het neurale netwerk kunnen runnen moeten we eerst de parameters voor het neurale netwerk aanpassen in de code.

Pas de code aan in *neuralfreq.cpp*:

```
...  
const float NeuralFreq::connection_rate = 1.0f; //fully connected NN  
const float NeuralFreq::learning_rate = 0.2f;  
const float NeuralFreq::desired_error = 0.0001f;  
...  
NeuralFreq::NeuralFreq(int no_single_items)  
{  
    ...  
    num_hidden = no_items; // pas hier de aantal verborgen knopen aan  
    ...  
}
```

Compileer en run nu het netwerk met de volgende commando's :

```
make clean  
make all  
./main 3 7 5 test.nn > test.netout
```

De tussenresultaten van de netwerktraining en de uiteindelijke voorspelling van het neurale netwerk worden geschreven naar *test.netout*. Het eindresultaat is te vinden aan het eind van dit bestand. Kopieer nu de voorspelling van het netwerk en de berekende uitkomst (in *test.sort*) naar een spreadsheet om de data uit te zetten in een grafiek en de totale error te berekenen met de RMSE-techniek.

We veranderen telkens één van deze parameters en houden de overige parameters constant. Voor elke parameter wordt er een aantal testen uitgevoerd met variërende waarden. Zo kunnen we voor elke parameter één voor één onderzoeken wat voor invloed het heeft op de uitkomst op het netwerk.

8. Experimenten

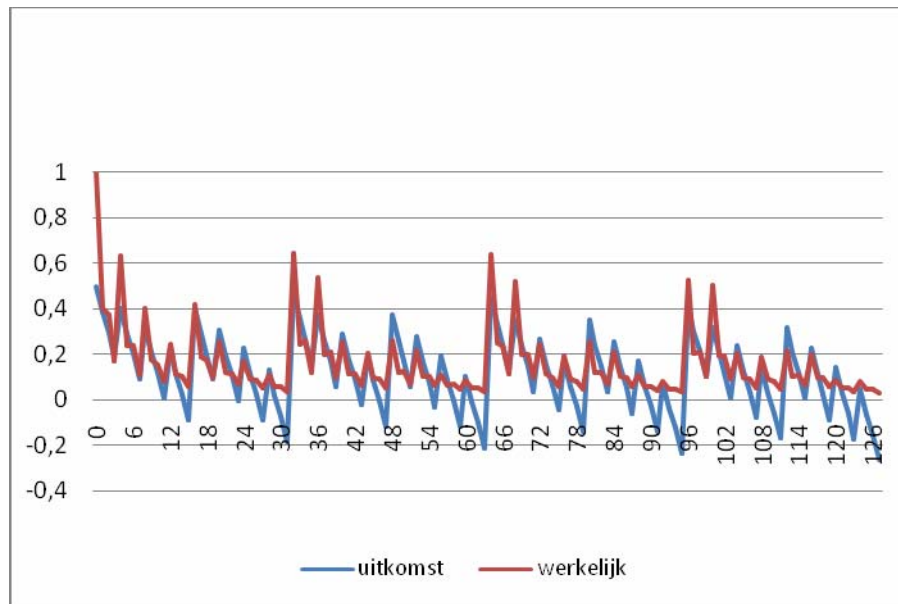
We geven een tabel van de uitgevoerde testen op het netwerk, zie bijlagen. Er worden in totaal vier verschillende datasets gebruikt voor dit onderzoek die random zijn gegenereerd met de generator. Zie bijlage Tabel 2 voor de eigenschappen van de verschillende datasets.

Testen 1-52 zijn uitgevoerd op een dataset van 7 producten, met gemiddelde transactielengte van 4, bestaande uit 800 transacties. Test 53, 54, 54b gebruiken een dataset van 9 producten, gemiddelde transactielengte van 4 en 10.000 transacties.

We bekijken de testgroep met testnummers $\{18, 19, 8, 9, 20, 10, 21, 34\}$: Het netwerk bevat 7 verborgen knopen en heeft learningrate van 0,2. De window-size varieert van maximaal 800 (gelijk aan het totale aantal samples) tot minimum 1. Hieruit zien we dat bij maximale window-size (size = 800) het resultaat het meest afwijkt van de realiteit. Voor kleinere window-size (size = 1 of size = 5) krijgen we betere resultaten, maar nog geen geweldige.

Op dezelfde manier kunnen we natrekken voor welk aantal verborgen knopen en window-size het neurale netwerk het beste is.

Een voorbeeld van een testresultaat ziet er als volgt uit. Dit is een test met 14 verborgen knopen en window-size = 10. Zie Figuur 2.



Figuur 2 window-size=10, 14 verborgen knopen, learning-rate=0,2

Op de verticale as staan de frequenties (als fractie van alle transacties). Op de horizontale as vinden we het itemsetnummer. Itemsetnummer 35 (binair 100011) staat voor itemset {item1, item2, item6}. Zo kun je uit de grafiek lezen dat de support van $I = \{item1, item2, item6\}$ is (11,75%) als $I = \{item1, item2, item6\}$.

De experimenten worden uitgevoerd op een 1.8 MHz PC met Linux operating system. Voor 9 items en 10.000 transacties is slechts enkele minuten nodig voor de netwerk training, en voor 16 items en 10.000 transacties is een uur vereist. Voor 20 items neemt de runtijd drastisch toe tot 11 uren.

Test 65 gebruikt het algoritme dat alles berekent (trainingversie 1) en test 66 gebruik het gereduceerde algoritme (trainingversie 2). Als we test 65 en test 66 met elkaar vergelijken kunnen we zien dat de trainingstijd van de tweede methode een groot deel is afgenomen, namelijk 3374 seconden en 2128 seconden respectievelijk. De afgenomen runtijd gaat ten koste van de nauwkeurigheid van de netwerkvoorspelling. De RMSE van test 65 is 0,0171, terwijl voor test 66 een veel grotere RMSE geldt namelijk RMSE = 0,0786.

Eveneens zien we het zelfde gedrag voor de *learningrate*, vergelijk test 58 met 59 en test 61 met 62. Een grotere *learningrate* bespaart trainingstijd maar het netwerk voorspelt support met grotere fout (RMSE). Zo zien we voor test 58 en 59 een RMSE van 0,0299 en 0,0591 respectievelijk, en voor test 61 en 62 een RMSE van 0,0215 en 0,0356 respectievelijk.

Er zijn enkele opmerkingen voor test 64. Dit is een test op een dataset met groot aantal producten (20) en 10.000 transactie. Wat opvallend hiervan is, is dat er geen negatieve waarden voorkomen, in tegenstelling tot andere testen die wel negatieve waarden hebben. De voorspelde waarden door het neurale netwerk liggen in dezelfde orde als de werkelijke berekende waarden. Itemsets met lage frequenties worden door het neurale netwerk iets hoger voorspeld en itemset met zeer hoge frequenties worden juist of iets lager voorspeld door het neurale netwerk. Alle voorspellingen liggen nauwkeurig tussen 0 en het totaal aantal transacties (10.000). Enkele voorbeeld ervan zijn:

Itemsets met werkelijke frequenties tussen 0 en 10 worden door het neurale netwerk voorspeld met waarden tussen 0 en 90.

Als de werkelijke frequentie ligt tussen 10 – 100, dan geeft het netwerk een waarde tussen 90 – 200.

Als de werkelijke frequentie ligt tussen 100 – 500, dan geeft het netwerk een waarde tussen 90 – 1000.

Als de werkelijke frequentie ligt tussen 500 – 1000, dan geeft het netwerk een waarde tussen 300 – 1500.

Als de werkelijke frequentie hoger is dan 2000, dan geeft het netwerk een waarde dat minder dan 2000 verschil met de werkelijke waarde.

Maar het kan ook voorkomen dat bepaalde voorspellingen helemaal fout zijn. Het uitzetten van de resultaten in een grafiek is helaas niet praktisch, er zijn teveel waarden waardoor het grafiek zeer onleesbaar maakt.

9. Conclusies

Als we kijken naar test 51, test 44 en test 52 na 4.000 transacties of test 33, kunnen we met redelijke zekerheid zeggen dat het gebruik van neurale netwerken voor het voorspellen of berekenen van de frequenties van alle mogelijke itemsets wel mogelijk is. Alleen is het kiezen en instellen van de geschikte of juiste parameters erg van belang.

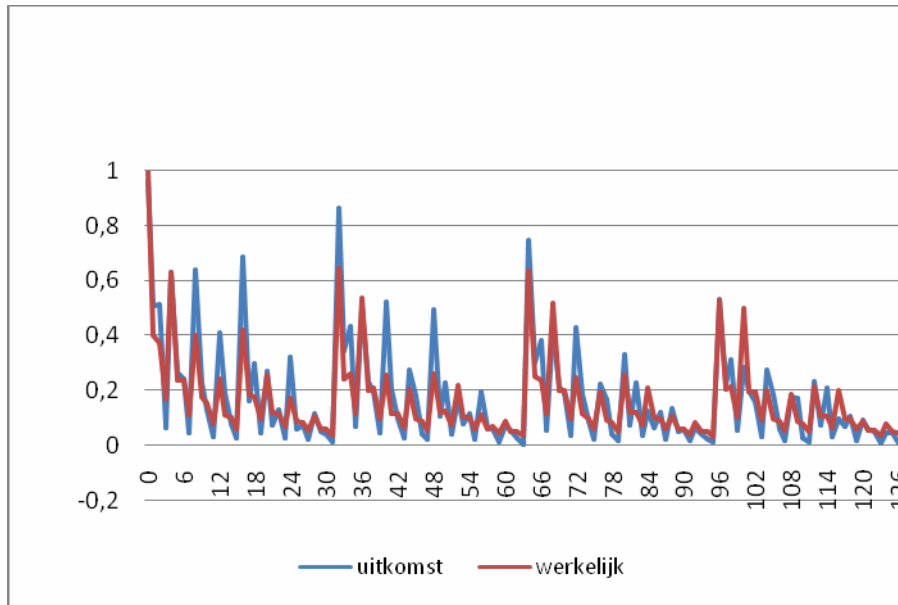
Meer verborgen knopen kunnen leiden tot meer complicaties. Onnodige extra knopen en relaties geven meer kans op problemen. Vergelijk bijvoorbeeld test 44 (14 verborgen knopen) met test 51 (21 verborgen knopen). Hoewel beide testen heel goede resultaten leveren, zijn er in de grafiek van test 51 iets meer kleine afwijkingen. Stel dat de resultaten even goed zijn, is het nog steeds beter om te kiezen voor de simpelste oplossing. Een geschikte keuze voor het aantal verborgen knopen voor een klein aantal inputs (7 inputs) is 1 tot 2 maal het aantal inputs.

Het belangrijkste is de window-size. Alle acceptabele tot uitstekende resultaten hebben een kleine window-size gemeenschappelijk. Hoe kleiner de window-size, hoe beter. window-size = 1 en window-size = 5 geeft zelfs de beste resultaten in meeste gevallen.

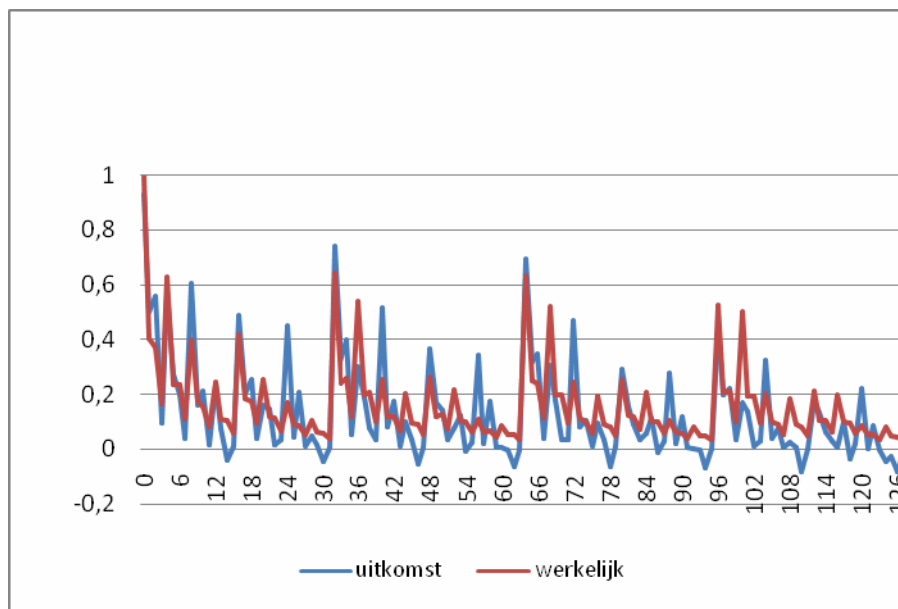
Het komt af en toe voor dat de frequentie voor een bepaalde itemset een negatief percentage als waarde heeft. Maar dit gebeurt meestal wanneer de werkelijke frequentie van die itemset dichtbij 0 (nul) procent is. Dit heeft dus weinig invloed op het globale idee van het onderzoek. Maar als we weten dat negatieve frequenties niet bestaan, kunnen we deze observatie gebruiken om het trainingproces van het netwerk te optimaliseren.

Als laatste valt te concluderen dat het wel mogelijk is met neurale netwerken de frequentie van alle voorkomende itemsets te berekenen. Een aan te raden instelling is als volgt: Aantal verborgen knopen is één of twee keer zoveel als het aantal inputknopen. De gewichten van het neurale netwerk moeten random geïnitieerd worden met getallen tussen -1 en 1. Als laatste moet de window-size ingesteld worden op een waarde tussen 1 en 5.

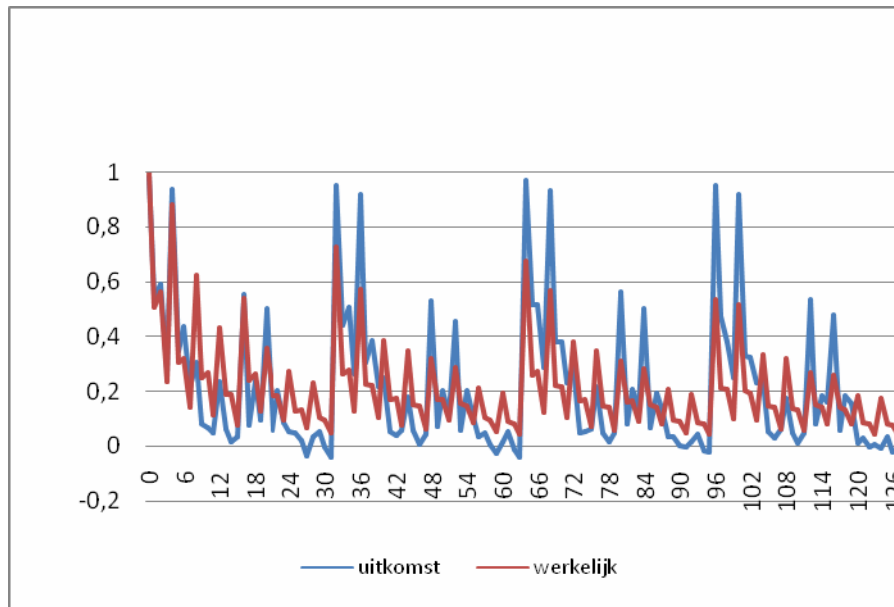
Voor 16 en 20 items levert het neurale netwerk (voor alle uitgevoerde instellingen) geen ideale voorspellingen. Maar de extreme waarden voor de supports komen in meer of mindere mate overeen met de werkelijke waarden. Uit de resultaten van het netwerk voor meerdere items (16 of 20 items) zien we een opmerkelijke gedrag. Itemsets die in bijna zelfde frequentie voorkomen in de originele database worden allemaal door het neurale netwerk voorspeld met één supportwaarde. Dit kan misschien een teken zijn van een tekort aan verborgen knopen.



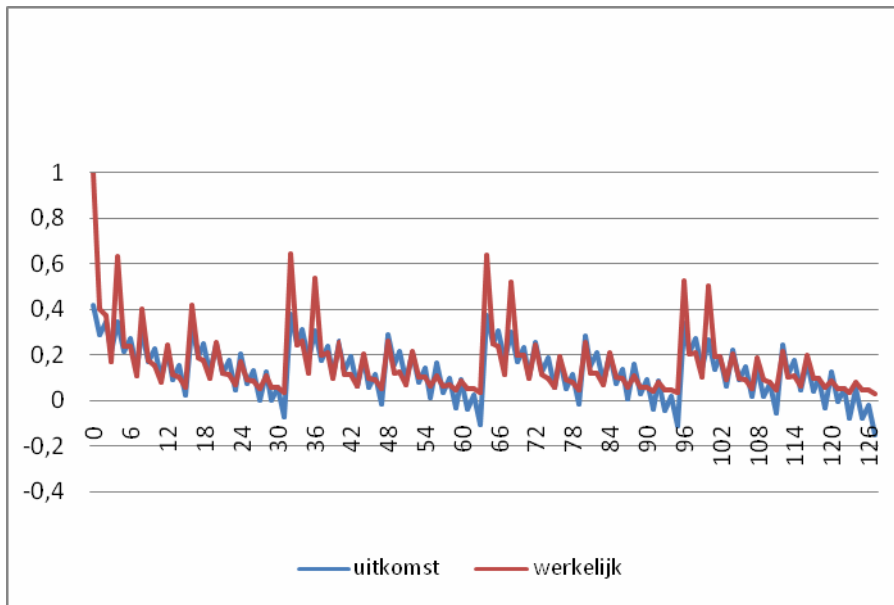
Figuur 3 Test 44 met 14 verborgen knopen en window-size=1



Figuur 4 Test 51 met 21 verborgen knopen en window-size=1



Figuur 5 Test 52 na 4000 transacties en window-size=1



Figuur 6 Test 33 met window-size=5, verborgenknopen=14 en init gewicht=0.0001

10. Dankbetuiging

Ik zou graag mijn dankbaarheid willen uiten aan Dr. Walter Kusters en Dr. Barzan Mozafari voor hun begeleiding, supervisie en geduld gedurende de voortgang van mijn scriptie. Ook wil ik dank zeggen aan Dr. Barzan Mozafari voor het opstellen van het neurale netwerk framework voor dit onderzoek.

11. Referenties

Kosters, W.A. en Pijls, W. (2004), *Apriori, a depth-first implementation*
Website : <http://www.liacs.nl/home/kosters/df/> [2008]

Frequent Itemset Mining Implementations Repository (2003, 2004), *FIMI03* en *FIMI04*
Website : <http://fimi.cs.helsinki.fi/> [2008]

Russell, S. J. en Norvig, P. (2003), *Artificial Intelligence: A modern approach*, second edition, Prentice Hall, New Jersey, USA

Daniel Franklin (1998), *Neural Network Library*, onderzoeks materiaal van Dr. Barzan Mozafari

Steffen Nissen (2003) en freegoldbar (at) yahoo dot com (2004), *Fast Artificial Neural Network Library (fann)* , onderzoeks materiaal van Dr. Barzan Mozafari

12. Bijlagen

Tabel 1 Lijst van uitgevoerde testen.

test #	Leersnelheid	knopen	transacties	maximale fout	verbinding	database	trainingsmethode	window size	RMSE	tijd verbruik (sec)
6	0.2	7	800	0.0001	1	a	0	100	0,244411	
7	0.2	7	800	0.0001	1	a	0	10	0,111289	
18	0.2	7	800	0.0001	1	a	1	800	0,290337	
19	0.2	7	800	0.0001	1	a	1	200	0,240409	
8	0.2	7	800	0.0001	1	a	1	100	0,243611	
9	0.2	7	800	0.0001	1	a	1	100	0,243373	
20	0.2	7	800	0.0001	1	a	1	20	0,131982	
10	0.2	7	800	0.0001	1	a	1	10	0,206063	
21	0.2	7	800	0.0001	1	a	1	5	0,147542	
34	0.2	7	800	0.0001	1	a	1	1	0,144444	
11	0.2	7	800	0.0001	1	a	1	800	0,303665	
12	0.2	7	800	0.0001	1	a	1	200	0,313034	
13	0.2	7	800	0.0001	1	a	1	100	0,21801	
14	0.2	7	800	0.0001	1	a	1	20	0,295934	
15	0.2	7	800	0.0001	1	a	1	10	0,10359	
16	0.2	7	800	0.0001	1	a	1	5	0,10359	
17	0.2	7	800	0.0001	1	a	1	5	0,137929	
35	0.2	7	800	0.0001	1	a	1	1	0,11169	
28	0.2	14	800	0.0001	1	a	1	800	0,296817	
29	0.2	14	800	0.0001	1	a	1	200	0,342213	
30	0.2	14	800	0.0001	1	a	1	100	0,31502	
31	0.2	14	800	0.0001	1	a	1	20	0,135829	
32	0.2	14	800	0.0001	1	a	1	10	0,109839	
33	0.2	14	800	0.0001	1	a	1	5	0,093166	
36	0.2	14	800	0.0001	1	a	1	1	0,334694	
22	0.2	21	800	0.0001	1	a	1	800	0,28794	

23	0.2	21	800	0.0001	1	a	1	200	0,329093	
26	0.2	21	800	0.0001	1	a	1	100	0,316049	
24	0.2	21	800	0.0001	1	a	1	20	0,142715	
27	0.2	21	800	0.0001	1	a	1	10	0,133474	
25	0.2	21	800	0.0001	1	a	1	5	0,287471	
37	0.2	21	800	0.0001	1	a	1	1	0,217072	
38	0.2	14	800	0.0001	1	a	1	800	0,311268	
39	0.2	14	800	0.0001	1	a	1	200	0,345626	
40	0.2	14	800	0.0001	1	a	1	100	0,267211	
41	0.2	14	800	0.0001	1	a	1	20	0,110861	
42	0.2	14	800	0.0001	1	a	1	10	0,159343	
43	0.2	14	800	0.0001	1	a	1	5	0,170244	
44	0.2	14	800	0.0001	1	a	1	1	0,078703	
45	0.2	21	800	0.0001	1	a	1	800	0,332115	
46	0.2	21	800	0.0001	1	a	1	200	0,345891	
47	0.2	21	800	0.0001	1	a	1	100	0,3317	
48	0.2	21	800	0.0001	1	a	1	20	0,123431	
49	0.2	21	800	0.0001	1	a	1	10	0,126451	
50	0.2	21	800	0.0001	1	a	1	5	0,169831	
51	0.2	21	800	0.0001	1	a	1	1	0,101483	
52	0.2	14	500-5000	0.0001	1	a	1	1		
53	0.05	9	10000	0.0001	1	b	1	1	0,22758672	336
54	0.15	9	10000	0.0001	1	b	1	1	0,2642648	123
54b	0.25	9	10000	0.0001	1	b	1	1	0,28834697	237
55	0.2	16	10000	0.0001	1	c	1	1	0,02622851	3479
56	0.8	16	10000	0.0001	1	c	1	1	0,04589469	2457
57	0.2	16	10000	0.0001	0.7	c	1	1	0,01465453	3067
58	0.2	16	10000	0.0001	1	c	1	5	0,02999695	2496
59	0.8	16	10000	0.0001	1	c	1	5	0,0591784	2166
60	0.2	16	10000	0.0001	0.7	c	1	5	0,02913762	2250
61	0.2	16	10000	0.0001	1	c	3	1	0,02154944	3384
62	0.8	16	10000	0.0001	1	c	3	1	0,03569892	2453
63	0.8	16	10000	0.0001	0.7	c	3	5	0,05800126	2289
64	0.1	20	10000	0.0001	1	d	3	1		39825
65	0.05	16	10000	0.0001	1	c	1	1	0,01719677	3374

66	0.05	16	10000	0.0001	1	c	3	1	0,078682	2128
----	------	----	-------	--------	---	---	---	---	----------	------

Dit is een tabel van de uitgevoerde experimenten. Uit de tabel kunnen we aflezen welke instelling en database wordt gebruik voor verschillende testen. Uit deze tabel kunnen we ook de resultaat zien in vorm van RMSE-waarde en de tijdsduur om het netwerk te laten trainen.

Tabel 2 Gebruikte databases

Database	Items	Gemiddelde transactie lengte	#transacties
a	7	3	800
b	9	4	10000
c	16	5	10000
d	20	5	10000

Dit is een tabel van de gebruikte database voor dit onderzoek. Bijvoorbeeld database *d* heeft 20 items en bestaat uit 10.000 transacties waarbij de gemiddelde transactie lengte gelijk is aan 5. Uit Tabel 1 kunnen we lezen dat database *d* wordt gebruik voor test 64.