

# TI2736-A

## Assignment 1: Artificial Neural Network

David Akkerman - 4220390  
Jan Pieter Waagmeester - 1222848

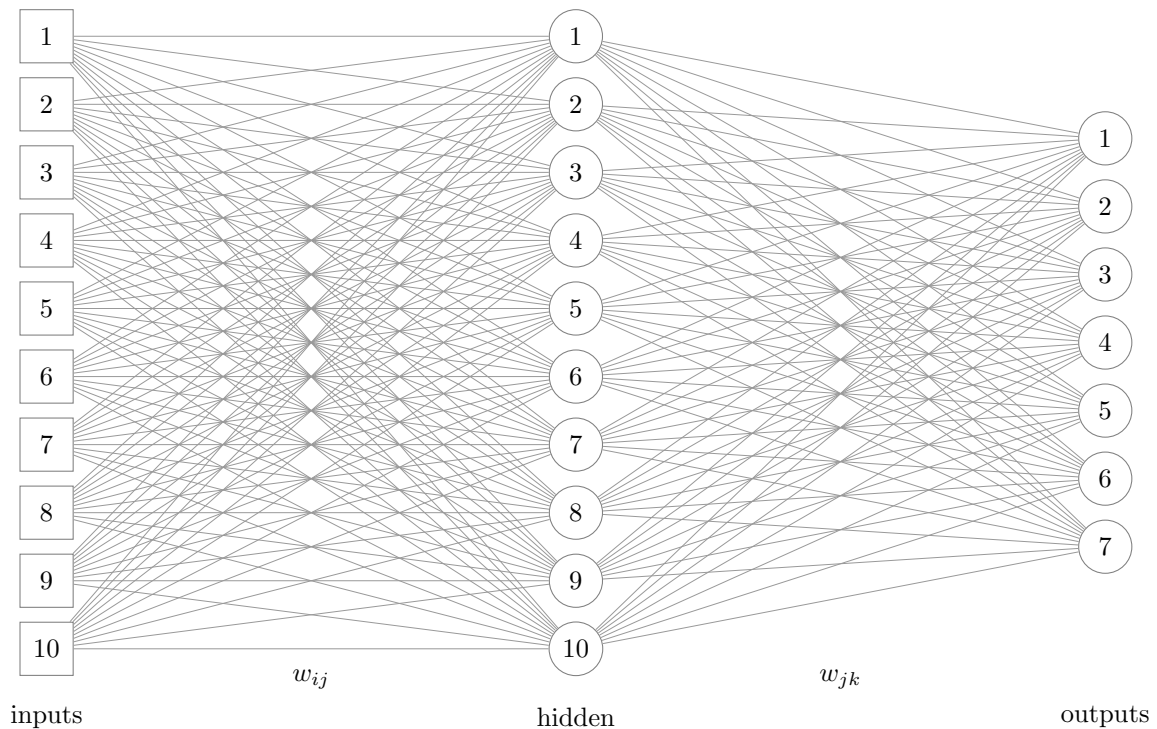
19 september 2014

Onze groepsgenoot is om persoonlijke redenen gestopt met het vak. Deze opdracht hebben we met ons tweeën gemaakt.

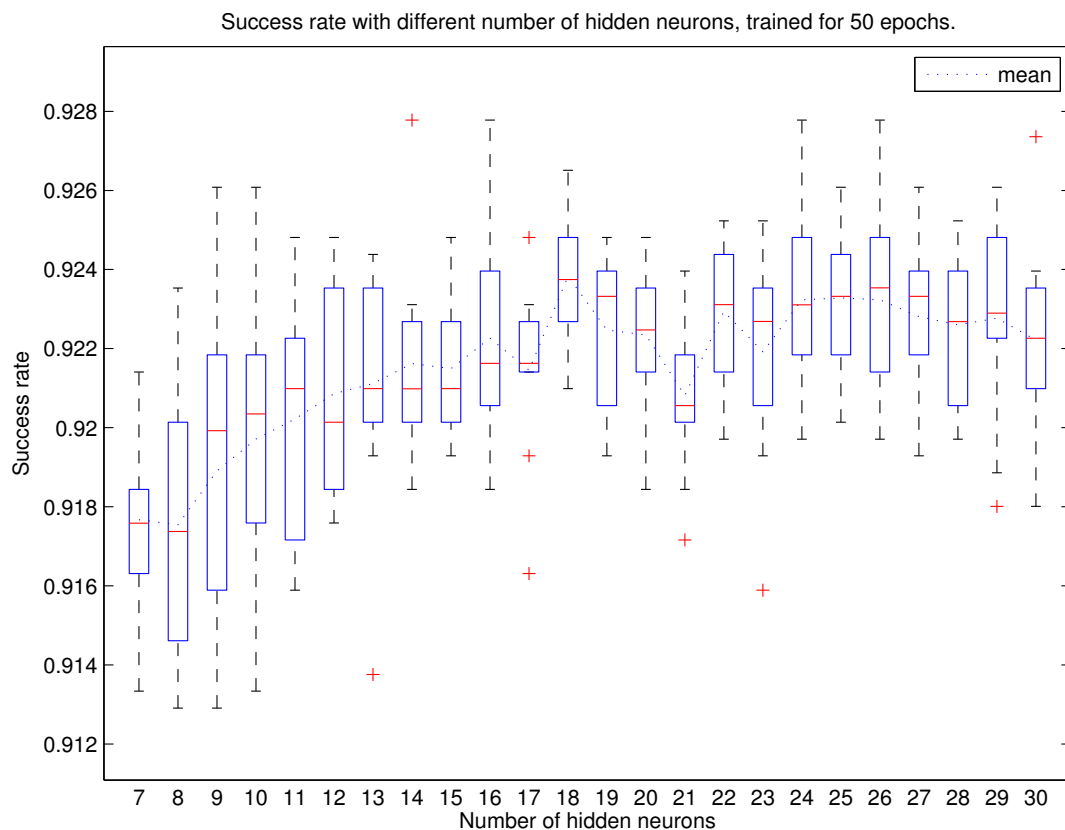
1. Omdat er 10 eigenschappen zijn hebben we 10 inputneurons nodig.
2. Voor de 7 verschillende classes hebben we 7 outputneurons nodig.
3. We beginnen met 10 hidden neurons.
4. We zullen de sigmoid-functie gebruiken:

$$Y^{sigmoid} = \left( \frac{1}{1 + e^{-X}} \right)$$

5. Ons netwerk kan voorgesteld worden als in het volgende plaatje. Daarbij kan het aantal hidden neurons uiteraard gevarieerd worden.



6. We gebruiken ongeveer 50% van de data voor training, 25% voor validatie en 25% als testset. De data lijkt aardig random gesorteerd te zijn, het is dus niet nodig om te zorgen dat er ongeveer even veel van elke class in elke set aanwezig is.
7. Door te kijken naar de *Mean square error* (MSE) op de trainingsset, en op de validatieset. Zolang ze beide dalen, wordt het netwerk beter. Als de MSE op de trainingsset daalt, maar die op de validatieset gaat stijgen is er sprake van overfitting: het netwerk wordt beter in het herkennen van ruis in de trainingsset, maar niet beter in het herkennen van de algemene patronen.
8. Op het moment dat het netwerk overtrained dreigt te raken, dus wanneer de MSE niet langer daalt en en begint te stijgen, wordt de training gestopt, zodat het nog wel geschikt is voor andere data dan de trainingsset. Daarnaast is het handig voor de snelheid van het netwerk te stellen dat wanneer de MSE van het netwerk over tien epochs niet meer dan 0.1% verschilt, ook het trainingsproces te beëindigen. Gebeurt dit niet, dan kan het nog erg lang doorgaan met optimaliseren, zonder dat er in het uiteindelijke resultaat verschil merkbaar is.
- 9.
10. We hebben 23 verschillende netwerken, met 7 - 30 *hidden neurons* 10 keer getraind. De succes-ratio van die netwerken hebben we verwerkt in de volgende boxplots: Hieruit blijkt ondermeer dat de



performance ook bij een klein aantal *hidden neurons* nog best goed is. Naast dit plaatje hebben we nog gekeken naar kleinere aantallen en zelfs met 4 *hidden neurons* wordt in gemiddeld 85% van de gevallen nog een correcte indeling in klassen gemaakt.

Verder is duidelijk dat het op een gegeven moment geen zin meer heeft om nog meer *hidden neurons* toe te voegen, het resulterende netwerk heeft er geen significant betere performance door.

11.

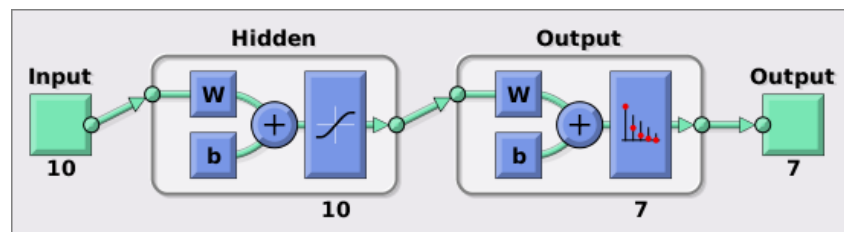
12.

13. Een confusionplot gemaakt met `plotconfusion`. Op de  $y$ -as staan de output-classes zoals ons netwerk produceert. Op de  $x$ -as staat de verwachtingswaarde. Een juist voorspelde waarde staat dus op de diagonaal, in de groene cellen.
- Verder is er ook uit af te lezen dat in dit geval het netwerk 20 keer een 4 voorspelde, terwijl het een 7 had moeten zijn. Dat is een duidelijke uitschieter waar verder voornamelijk fouten onder de 10 staan.

Confusion matrix (testset) 400 epochs, training set: 4000, hidden neurons: 40

Output Class	1	2	3	4	5	6	7	
	313 13.3%	2 0.1%	4 0.2%	8 0.3%	4 0.2%	2 0.1%	5 0.2%	92.6% 7.4%
	0 0.0%	307 13.0%	3 0.1%	0 0.0%	2 0.1%	1 0.0%	3 0.1%	97.2% 2.8%
	7 0.3%	7 0.3%	295 12.5%	3 0.1%	3 0.1%	1 0.0%	4 0.2%	92.2% 7.8%
	3 0.1%	0 0.0%	1 0.0%	280 11.9%	6 0.3%	3 0.1%	20 0.8%	89.5% 10.5%
	7 0.3%	2 0.1%	15 0.6%	0 0.0%	327 13.9%	8 0.3%	0 0.0%	91.1% 8.9%
	3 0.1%	7 0.3%	3 0.1%	1 0.0%	9 0.4%	350 14.9%	5 0.2%	92.6% 7.4%
	3 0.1%	9 0.4%	9 0.4%	9 0.4%	0 0.0%	3 0.1%	297 12.6%	90.0% 10.0%
Target Class	1	2	3	4	5	6	7	
	93.2% 6.8%	91.9% 8.1%	89.4% 10.6%	93.0% 7.0%	93.2% 6.8%	95.1% 4.9%	88.9% 11.1%	92.1% 7.9%

14. Geüpload naar blackboard als `5_classes.txt`
15. Met `nprtool` is het binnen een paar minuten mogelijk om een netwerk te maken wat er zo uit ziet:



16. De performance van dit netwerk is erg goed, en de training is ook erg snel klaar. Ook met 10 hidden neurons worden succes-percentages van boven de 90% gehaald.
- Tijdens het maken van het eigen neurale netwerk had ik het idee dat het gebruiken van de sigmoid-activatiefunctie wellicht voor de output-layer niet handig was. Dit netwerk gebruikt een softmax activatiefunctie in de output-layer, wellicht ook een reden van betere prestaties.

De veel hogere trainingsefficiëntie is waarschijnlijk verklaarbaar door bijvoorbeeld implementatie in correcte matrix-algebra en het toevoegen van het aanpassen van de *learning rate*, het toevoegen van een momentum-term in de delta-rule (Negnevitsky, equation 6.17).