1 Praktijk II: Neural network

1.1 Inleiding

Met de kennis die we hebben opgedaan bij het maken van het theorieverslag gaan we zelf een praktijkopdracht uitvoeren. We dagen onszelf uit tot het zelf maken van een computerprogramma dat de volgende vraag beantwoordt: *welke onderdelen zijn nodig voor een zelflerend computersysteem, in de vorm van een door ons ontworpen computerprogramma, dat in staat is afbeeldingen te classificeren binnen vastgestelde categorieën met een precisie van meer dan 80 procent?*

1.2 Werkwijze

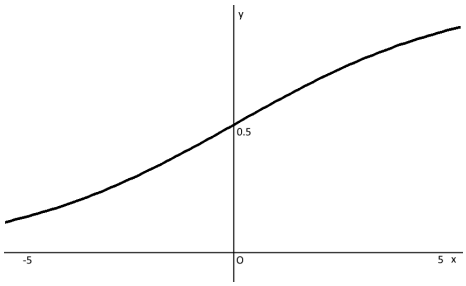
Voordat we kunnen beginnen aan het ontwerp van het systeem moeten we eerst duidelijk maken wat we precies willen bereiken en hoe we dat willen bereiken. We moesten een aantal vragen eerst beantwoorden:
Welke afbeeldingen gaan we proberen te classificeren?
We gaan proberen handgeschreven cijfers te classificeren. We gebruiken hiervoor de MNIST dataset. [TODO] Dit is een dataset van een groot aantal afbeeldingen van handgeschreven cijfers met het bijbehorende label. De afbeeldingen zijn elk 28 x 28 pixels groot. Deze dataset is erg geschikt om Machine Learning op toe te passen.
Welk algoritme gebruiken we voor het classificeren van de afbeeldingen?
Wij gebruiken een Neural Network (TODO) om deze afbeeldingen te classificeren. Het is gebleken dat dit het beste algoritme is voor het classificeren van afbeeldingen. Er zijn 28 * 28 = 784 input neuronen en 10 output neuronen. We kiezen ervoor om één hidden layer te maken.
Welke manier van leren gebruiken we voor het verbeteren voor het algoritme?
We gaan gebruik maken van Gradient Descent (TODO).

1.3 Netwerk

Het aantal input neuronen en het aantal output neuronen staat vast (respectievelijk 784 en 10). Het aantal hidden neuronen kunnen we zelf bepalen. Ook moeten nog aan elke laag een bias toevoegen (TODO). Zoals uitgelegd in (TODO) moeten we ook een *activation function* kiezen. Hiervoor gebruiken we een sigmoid functie (zie figuur 2). De lagen zijn onderling volledig verbonden. Elke verbinding heeft zijn eigen weging die op het begin naar een willekeurige waarde wordt gezet.

$$f(x) = \frac{1}{1 + e^{-u}}$$

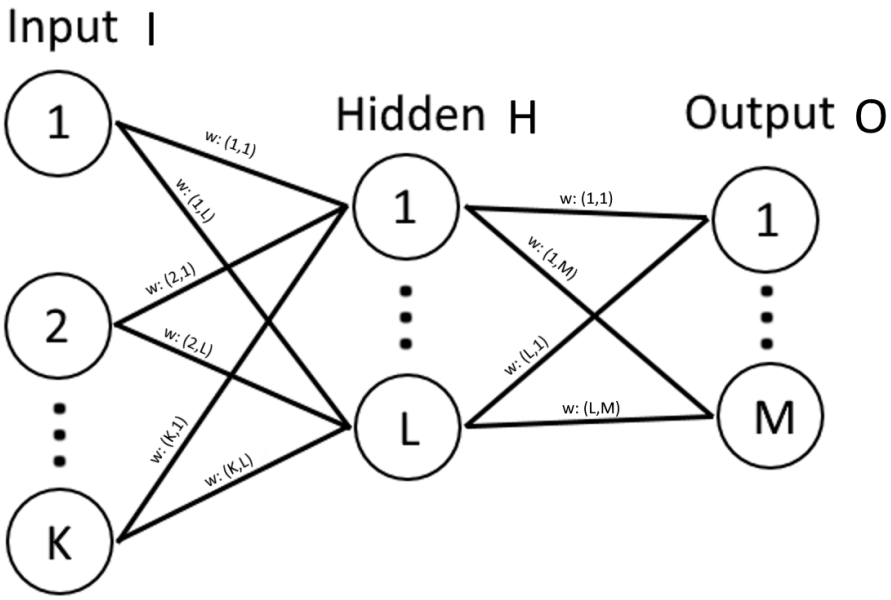
Figuur 1: De sigmoid functie



Figuur 2: De grafiek van de sigmoid functie

1.4 Gradient descent

Nu we een werkend netwerk hebben kunnen we naar het belangrijkste deel kijken: het vinden van de juiste wegingen voor de neuronen. Zoals gezegd gaan we dit doen door middel van gradient descent. Wat we gaan doen is: *het bepalen van de afgeleide van de cost function relatief tot de te veranderen variable.*



Figuur 3: Een neural network met aangegeven weights

De cost functie is:

$$C(x) = \frac{1}{2} \sum_{m=1}^{M} (y_m - t_m)^2$$

Let op dat $\frac{1}{2}$ gewoon wordt toegevoegd om later makkelijker de afgeleide te kunnen bepalen. We moeten ook de afgeleide van de sigmoid functie weten, deze is gelukkig erg simpel, namelijk:

$$f'(x) = f(u)(1 - f(u)))$$

Wat we nu doen is het bepalen van de afgeleide van de cost functie relatief tot de weging van de hidden neuronen naar de output neuronen (w_{ho}). Om tot dit resultaat moeten we een aantal keer de ketting regel toepassen:

$$\begin{aligned} c'(x) &= (O_m - t_m) * O' \\ O' &= O_m(1 - O_m) * H_l \\ c'(x) &= (O_m - t_m) * O_m(1 - O_m) * H_l \end{aligned}$$

Nu we de afgeleide hebben berekend kunnen we de weight een beetje in de goede richting verplaatsen. Hierin staat η voor de learning rate (TODO ref).

$$w_{lm} = w_{lm} - \eta * (O_m - t_m) * O_m(1 - O_m) * H_l$$

Om de afgeleide van de cost functie te bepalen relatief tot de weging van de de input neuronen naar de hidden neuronen moeten we nog een aantal keer de ketting regel toepassen, maar uiteindelijk krijgen we de volgende afgeleide:

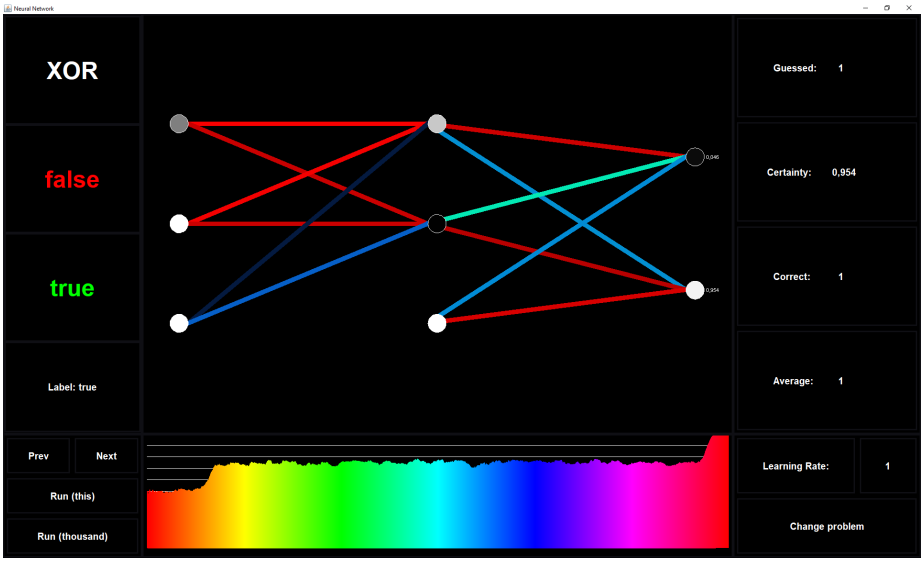
$$\begin{aligned} C'(x) &= \sum_{m=1}^{M} [(O_m - t_m) * O'] \\ O' &= O_m(1 - O_m) * H_l * H' \\ H' &= H_l(1 - H_l) * I_k \\ C'(x) &= \sum_{m=1}^{M} [(O_m - t_m) * O_m(1 - O_m) * w_{ho}] * H_l(1 - H_l) * I_k \end{aligned}$$

De weging w_{lm} wordt dan:

$$w_{lm} = w_{lm} - \eta * \sum_{m=1}^{M} [(O_m - t_m) * O_m(1 - O_m) * w_{ij}] * H_l(1 - H_l) * I_k$$

1.5 Programma

Bij het opstarten van het programma verschijnt een scherm waarin de beginwaarden van het neural network bepaald kunnen worden en het probleem gekozen kan worden. Als je met de muis op een onderdeel staat verschijnt er meer informatie over het desbetreffende onderdeel. Om uit te testen of het programma goed functioneert hebben we het eerst een simpel probleem uitgetest: XOR. Het doel van een XOR port is als volgt: Als één van beide input waarden "1" is, moet het antwoord "1" worden, anders "0". Na even trainen lukt leert het neural network dit probleem op te lossen! Dit betekent dat het programma goed functioneert, dus kunnen we naar het *echte* probleem.



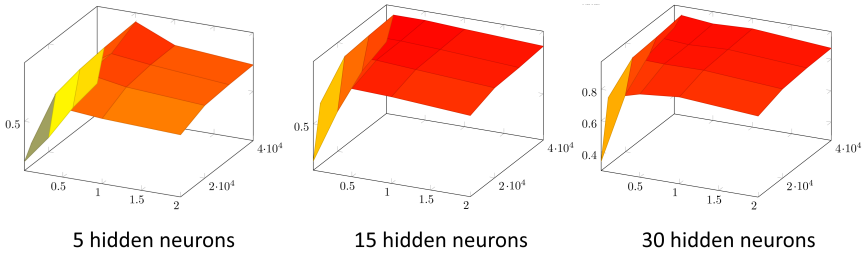
Figuur 4: Het door ons geschreven Neural Network programma dat zelf heeft geleerd XOR op te lossen

1.6 Resultaten

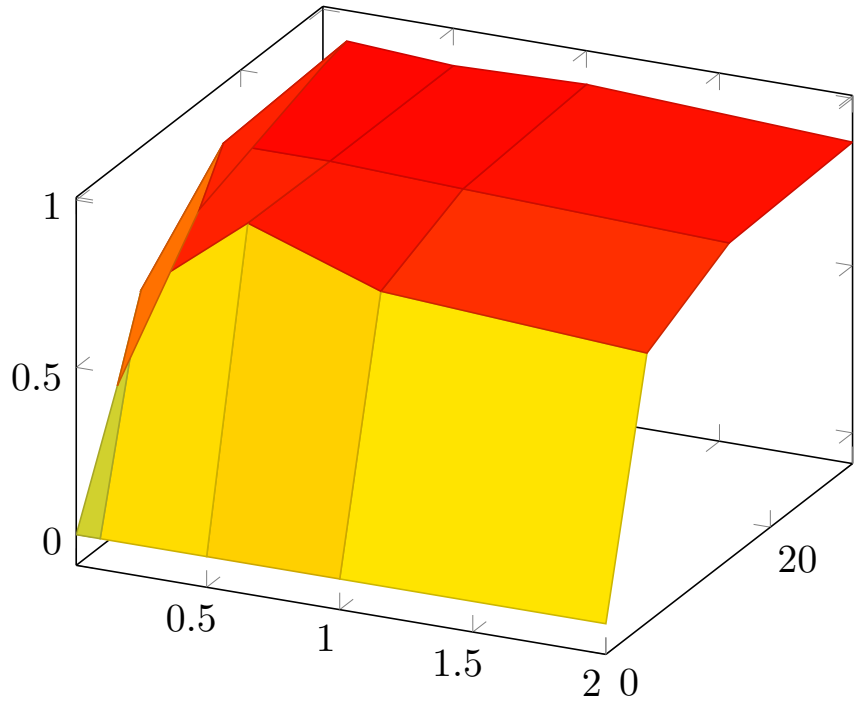
De prestaties van het neural network worden door een aantal variable beëinvloed:

- Het aantal hidden neurons
- De learnig rate
- Het aantal training plaatjes

In figuur 5 is te zien welk effect het veranderen van de variable heeft op de prestaties van het netwerk.



Figuur 5: Resultaten van ons neural network. De y-as duidt de prestatie van het netwerk aan. De x-as duidt de de learning rate aan. De z-as duidt het aantal plaatjes aan.



Figuur 6: Resultaten van ons neural network getrained op **40000** plaatjes. De y-as duidt de prestatie van het netwerk aan. De x-as duidt de de learning rate aan. De z-as duidt het aantal hidden neurons aan.

1.7 Conclusie

Eën manier voor het maken van een computerprogramma dat in staat is afbeeldingen te categoriseren is door gebruik te maken van een neural network dat verbeterd wordt doormiddel van gradiënt descent.