Programming Neural Networks

Geschreven door: Matthijs Koelewijn

Uitleg code

Call tree

main

…….read_data

…….data_min_max

…….normalise

…….NeuralNetwork__init__

…….NeuralNetwork.set_input_layer

…….InitialNeuron__init__

…….InitialNeuron.set_input_neurons

…….NeuralNetwork.add_layer

…….Neuron__init__

…….print

…….NeuralNetwork.set_input_layer

…….InitialNeuron__init__

…….InitialNeuron.set_input_neurons

…….NeuralNetwork.get_output #

…….get_desired_output

…….NeuralNetwork.forward_propagation

…….Neuron.get_output

…….get_output #

…….Neuron.output_delta_rule

…….transform_array

…….Neuron.delta_rule

…….transform_array

…….NeuralNetwork.backwards_propagation

…….Neuron.update

…….Neuron.get_a

…….NeuralNetwork.update_zeta

…….Neuron.set_zeta

…….validation

…….NeuralNetwork.set_input_layer

…….InitialNeuron__init__

…….InitialNeuron.set_input_neurons

…….get_desired_output

…….NeuralNetwork.get_output

…….plot

…….validation

…….NeuralNetwork.set_input_layer

…….InitialNeuron__init__

…….InitialNeuron.set_input_neurons

…….get_desired_output

…….NeuralNetwork.get_output

…….print

Totaal-uitleg

Het programma "Programming Neural Networks" begint bij de aanroep van "**read_data**" in "main.py". Deze functie geeft de data van het neurale netwerk terug, evenals de trainings- en validatiedata: "NN_data", "training_data" en "validation_data". De dataset "Iris.data" wordt gebruikt en opgesplitst in deze drie verschillende datasets.

Vervolgens wordt "**data_min_max**" aangeroepen om de minimale en maximale waarde van de dataset te bepalen. Zo kunnen de trainings- en validatiedata genormaliseerd worden met behulp van de "**normalise**" functie.

Na het normaliseren wordt het "**NeuralNetwork**" object geïnitialiseerd. Dit object bevat "layers", "output_deltas" en "output_weights". De laag begint altijd met een "input_layer", die wordt aangemaakt met "**Neural.set_input_layer**". In de input layer worden "**InitialNeurons**" objecten geïnitialiseerd. Deze zijn anders dan de "**Neuron**" objecten omdat ze geen input krijgen van neuronen van een vorige laag.

Vervolgens worden lagen toegevoegd met "**NeuralNetwork.add_layer**". Deze functie krijgt het aantal neuronen mee die nodig zijn. Deze "Neuron" objecten worden binnen deze functie geïnitialiseerd.

Nadat dit initialisatieproces is voltooid, wordt het neurale netwerk getraind. Het aantal iteraties om het neurale netwerk te trainen wordt aangegeven in "n_epoch". Om de voortgang bij te houden, wordt de voortgang geprint met "print". Er wordt geïtereerd over het aantal epochs. In elke epoch wordt een training uitgevoerd op de training dataset. Voor deze trainingdata worden nieuwe input layers gemaakt met "**NeuralNetwork.set_input_layer**". Zoals eerder genoemd wordt hier de input layer gemaakt met "**InitialNeuron**" objecten.

Vervolgens wordt de "**get_desired_output**" functie aangeroepen, die een waarde teruggeeft in een formaat dat nodig is. Deze output wordt meegegeven aan de "**NeuralNetwork.forward_propagation**" functie. In deze functie wordt forward propagation toegepast op het neurale netwerk. Deze functie berekent de output van elke "Neuron" object per laag. De forward propagation wordt eerst uitgevoerd op de outputlaag met de functie "**Neuron.output_delta_rule**" voor elke neuron op deze laag. Deze functie geeft de delta en de gewichten terug. Deze worden opgeslagen in de "deltas" en "weights" van het "**NeuralNetwork**" object. De functie "**transform_array**" wordt gebruikt op de gewichten. Deze functie geeft een tweedimensionale array terug.

Daarna wordt voor de middelste lagen de delta rule toegepast en nieuwe waarden berekend. De oude "deltas" en "weights" worden berekend voor elke neuron in elke laag met de functie "**Neuron.delta_rule**" en toegevoegd aan de "deltas" en "weights" lijsten. Wederom wordt de functie "**transform_array**" aangeroepen.

De “**NeuralNetwork.backwards_propagation**” functie wordt vervolgens aangeroepen. Deze functie werkt van achteren naar voren. Voor elke neuron in elke laag wordt de “**Neuron.update**” functie aangeroepen. Deze functie wijzigt de gewichten van elke neuron door de backwards propagation formule toe te passen. De “**Neuron.get_a**” functie helpt bij het verkrijgen van de output.

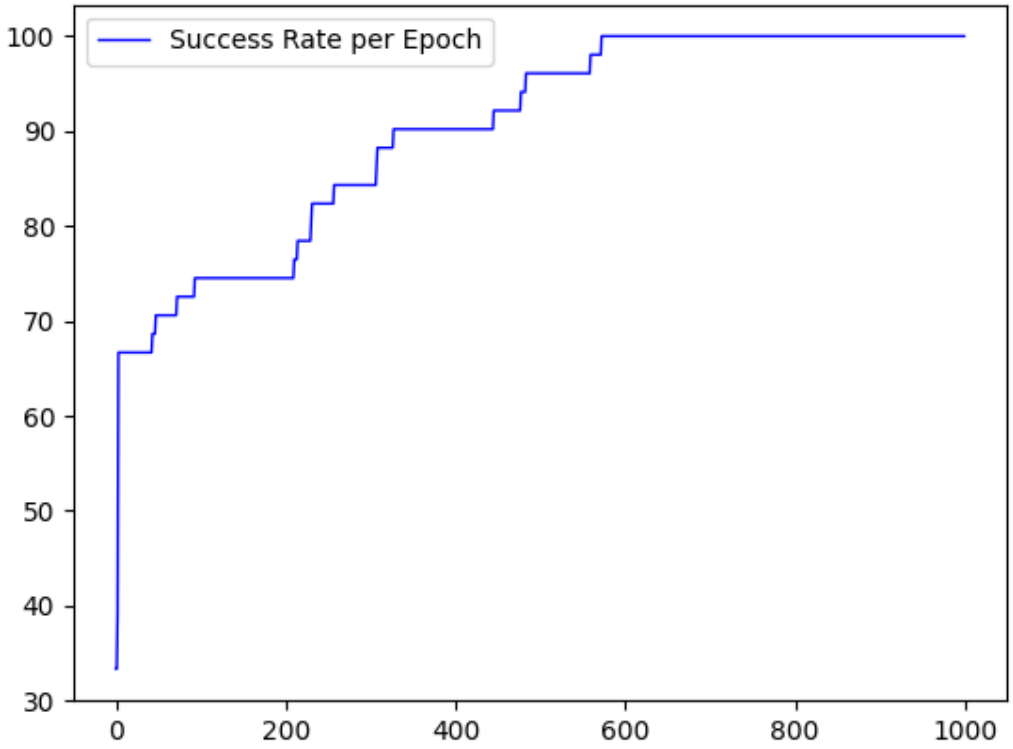
Na het toepassen van forward propagation en backward propagation voor alle trainingsdata, wordt de “Zeta” aangepast voor elk “Neuron” object in de “**NeuralNetwork**”. De “**NeuralNetwork.update_zeta**” functie vernieuwt de “Zeta” door “**Neuron.set_zeta**” aan te roepen.

Om de succesratio en het aantal juiste voorspellingen te bepalen, wordt de “**validation**” functie gebruikt. Deze functie krijgt het neurale netwerk en de validatiedata als parameters. Binnen deze functie wordt voor elke neuron de gewenste output vergeleken met de verkregen output, en wordt het percentage correcte voorspellingen berekend.

Nadat alle epochs zijn uitgevoerd, worden deze resultaten geplot met de “**plot**” functie van “matplotlib”. Deze plot geeft de succesratio weer voor het aantal epochs.

De succesratio en het aantal correcte voorspellingen voor de laatste epoch worden ten slotte nog geprint in de terminal.

Resultaten




```
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project> & c
The final successrate is 94.95%
The network has correctly guessed 94 out of 99 datapoints
(venv) PS C:\Users\MatthijsKoelewijnDen\Documents\Github\AAI-Project>
```

De hoogste succesrate is 94.95%, 94 van de 99 punten zijn correct.