Neural network: Report TP#3

Due on Thu, November 12, 2022

Master 2 GSI

Tarek Berkane

1 Purpose of project

- Produce multilayer perceptron neural network
- Understand different method of activation function
- Implement forward propagation and backward propagation
- Understand how neural network learn/train

2 Expected goal

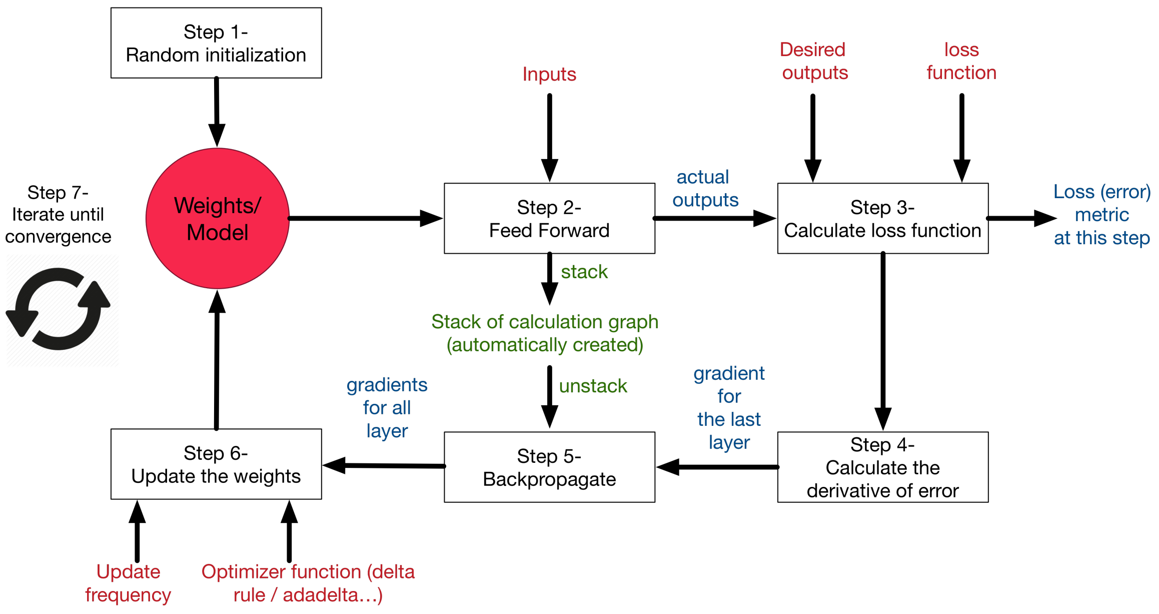
Realization of multilayer perceptron neural network with the ability to classify between two items `Velo` and `Moto`.

2.1 Understading Neural networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network

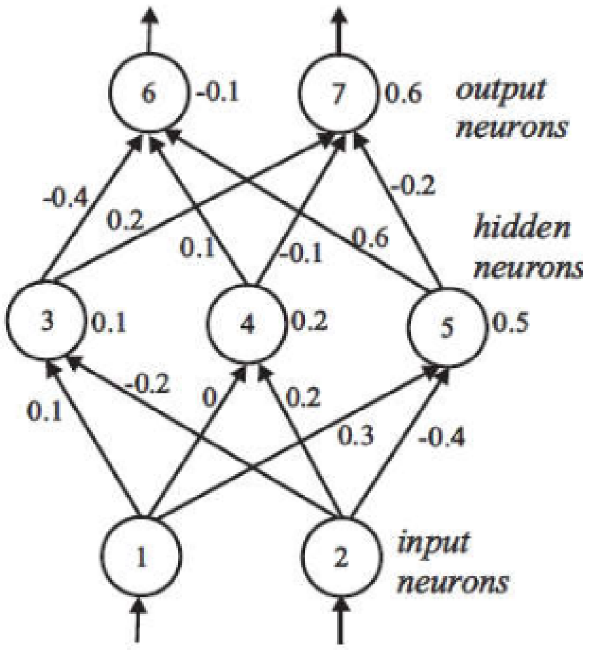
2.2 Steps of Neural networks



3 Implementation of homework

3.1 Explanation of architecture used

The architecture used in our model is consist of **3 Layers** `(input, hidden, output)`
We use **sigmoid** function in hidden layers, cause the sigmoid have a smooth curve and always return a positive number
We use a **step function** in the output layer to make the output node produce one of the two values, 1 or 0.



3.2 Activation function

We use two type of activation function **Sigmoid** in `hidden layer` and **Step function** in `output layer`

3.2.1 Simgoid

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

Derivation of sigmoid

$$sig(x)' = sig(x) * (1 - sig(x))$$

3.2.2 Step function

$$step(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

3.3 Forward propagation

$$\vec{Hidden} = sig(\vec{W_{input}} \cdot \vec{Input})$$
$$\vec{Out} = step(\vec{W_{hidden}} \cdot \vec{Hidden})$$

3.4 gradient descent

$$\frac{\partial Error}{\partial Weight} = \frac{\partial Error}{\partial Out} \times \frac{\partial Out}{\partial In} \times \frac{\partial In}{\partial Weight}$$
$$\Delta_{output} = MEA'(step'(\hat{y} - y)')$$
$$\Delta_{output} = \pm 1 \times (\hat{y} - y)$$
$$\Delta_{hidden} = \vec{Weight}_{hidden} \cdot \Delta_{output} \times sig(x) \times (1 - sig(x))$$

3.5 Backward propagation

$$\vec{Out}_{error} = \vec{label} - \vec{prediction}$$
$$\vec{Hidden}_{error} = (\vec{W}^{T}_{hidden} \cdot \vec{Out}_{error}) * (sig(x) * (1 - sig(x))$$

3.6 Update the weights

$$\vec{Weight}_{hidden}(t+1) = \vec{Weight}_{hidden}(t) * \eta * (\vec{Out}_{error} \cdot \vec{Hidden}^{T})$$
$$\vec{Weight}_{input}(t+1) = \vec{Weight}_{input}(t) * \eta * (\vec{Hidden}_{error} \cdot \vec{Input}^{T})$$

3.7 Error function (Mean Absolute Error)

$$error = |\vec{Label} - \vec{Prediction}|$$

4 Implemntation in python

4.1 Functions

4.1.1 Simgoid

```python
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
# OR
import scipy.special
scipy.special.expit(x)
```

4.1.2 Derivation of sigmoid

```python
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))

dsigmoid = lambda x:sigmoid(x) * ( 1 - sigmoid(x))
```

4.1.3 Step function

```python
# Activation function for output layer
def step_function(x):
    return (x >= 0) * 1
```

4.2 Forward propagation

```python
samples = selected_samples[0]
label = selected_samples[1]

inputs = numpy.array(samples, ndmin=2).T
targets = numpy.array(label, ndmin=2).T

hidden_inputs = numpy.dot(self.weight_input_hidden, inputs)
hidden_outputs = self.activation_function_sigmoid(hidden_inputs)

final_inputs = numpy.dot(self.weight_hidden_output, hidden_outputs)
final_outputs = self.activation_function_step(final_inputs)
```

4.3 Backward propagation

```python
output_errors = targets - final_outputs
hidden_errors = numpy.dot(self.weight_hidden_output.T, output_errors) *
↪   hidden_outputs * (1.0 - hidden_outputs)
```

4.4 Update weight

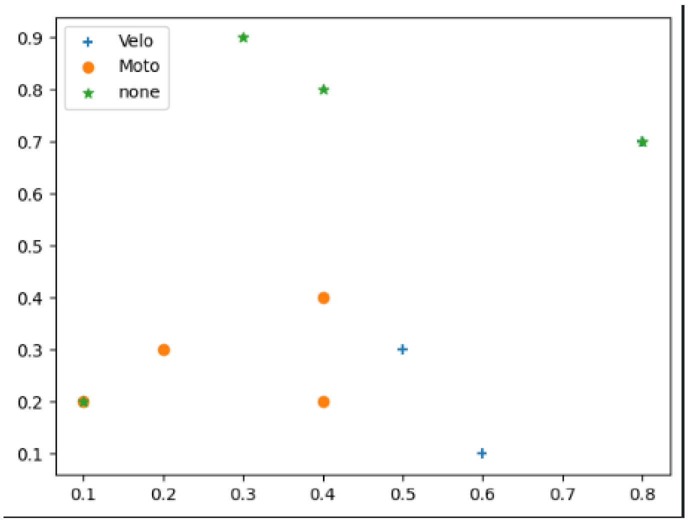
```python
self.weight_hidden_output += self.lr * numpy.dot(
        (output_errors),numpy.transpose(hidden_outputs),
)

self.weight_input_hidden += self.lr * numpy.dot(
        (hidden_errors),numpy.transpose(inputs),
)
```

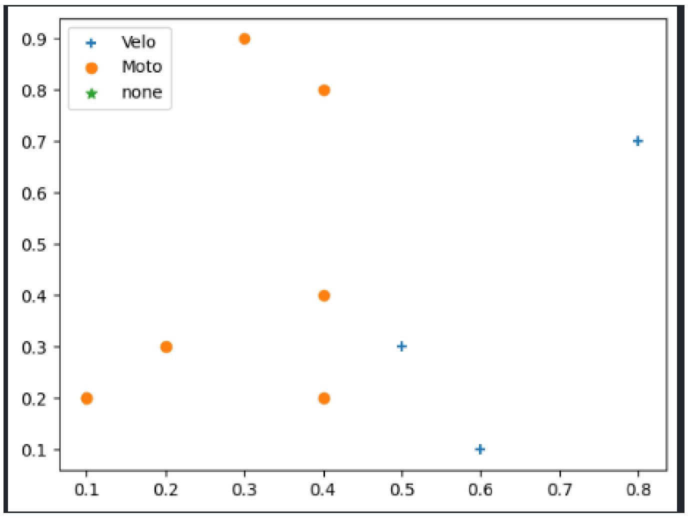
5 Prediction Table 2

0.4	0.8	Moto
0.8	0.7	Velo
0.3	0.9	Moto
0,1	0.2	Moto

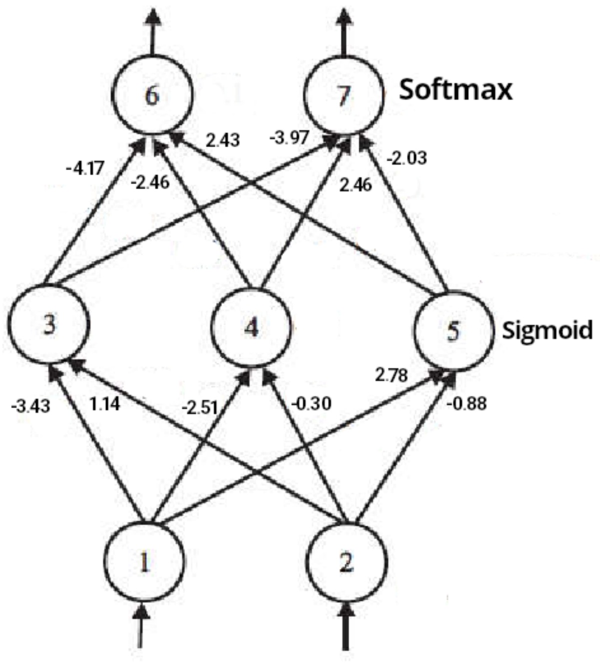
5.1 Without prediction



5.2 With prediction



6 New weight


6
7
Softmax
-4.17
-2.46
2.43
-3.97
2.46
-2.03
3
4
5
Sigmoid
-3.43
1.14
-2.51
-0.30
2.78
-0.88
1
2

References

[1] W. Trask, *grokking Deep Learning*, packets (2019).

[2] , Tariq *Make Your Own Neural Network*, packets (2016).

[3] Neural network, `https://en.wikipedia.org/wiki/Neural_network`

[4] Backpropagation, `https://en.wikipedia.org/wiki/Backpropagation`

[5] What is backpropagation really doing? , `https://www.youtube.com/watch?v=Ilg3gGewQ5U`

[6] Backpropagation Details , `https://www.youtube.com/watch?v=iyn2zdALii8`