Coding Two: Advanced Frameworks
Assignment Element 1: Lab Work
Week 5 Exercise - Neural Networks by hand

In week 5 we created a simple toy neuron by hand

This should have left you with enough information to create a single later of neurons

If you managed to do this, you may submit this as one of your in-class assignments.

In this simple neural network Python tutorial, we will use the Sigmoid activation function.

There are many types of neural networks. In this project, we will create a feedforward or perceptual neural network. This type of ANN directly transfers data from front to back.

The training of feedforward neurons often requires backpropagation, which provides the corresponding input and output sets for the network. When the input data is sent to the neuron, after processing, an output is produced.

Create a NeuralNetwork class

We will use Python to create a NeuralNetwork class to train neurons in order to give accurate predictions. This class will also have other helper functions.

Even if we will not use the neural network library in this simple neural network example, we will import the numpy library to assist in the calculation.

We will use the Sigmoid function to draw a characteristic "S" curve as the activation function of the neural network.

This function can map any value to a value between 0 and 1, and it will help us to weight and normalize the input.

After that, we will create the derivative of the Sigmoid function to help calculate the adjustment parameters of the weight.

The output of the Sigmoid function can be used to generate its derivative. For example, if the output variable is "x", its derivative is x*(1-x).

Training model

This is the stage where we teach neural networks to make accurate predictions. Each input has a weight-it can be positive or negative. This means that inputs with larger positive or negative weights will have a greater impact on the output of the result. Remember, we initially started by assigning a weight to each random number.

Here is the training process of this neural network example:

The first step: extract the input from the training data set, adjust it according to the weight of the training data set, and filter it through a method of calculating the output of the neural network.

Step 2: Calculate the backpropagation error rate. In this case, it is the difference between the predicted output of the neuron and the expected output of the training data set.

The third step: using the error weighted derivative formula, according to the obtained error range, some smaller weight adjustments are made.

Step 4: Perform 15,000 iterations of this process. In each iteration, the entire training set is processed simultaneously.

We use the ".T" function to convert the matrix from a horizontal position to a vertical position.

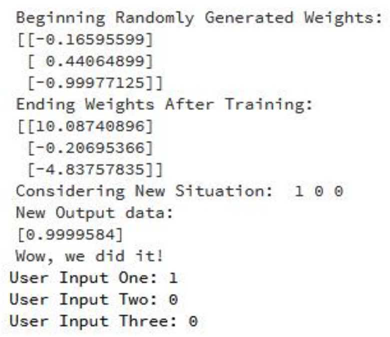
Ultimately, the weights of neurons will be optimized based on the provided training data. Later, if the neuron is asked to consider a new state, which is the same as the previous state, it can make an accurate prediction. This is the way of backpropagation.

Pack and run

Finally, after the NeuralNetwork class is successfully initialized, the code can be run.

Here is the complete code of how to create a neural network in a Python project:

```
import numpy as np

class NeuralNetwork():


    def __init__(self):


        # seeding for random number generation


        np.random.seed(1)


        #converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of 0


        self.synaptic_weights = 2 * np.random.random((3, 1)) - 1



    def sigmoid(self, x):


        #applying the sigmoid function


        return 1 / (1 + np.exp(-x))


    def sigmoid_derivative(self, x):


        #computing derivative to the Sigmoid function


        return x * (1 - x)


    def train(self, training_inputs, training_outputs, training_iterations):
        #training the model to make accurate predictions while adjusting weights continually

        for iteration in range(training_iterations):

            #siphon the training data via  the neuron

            output = self.think(training_inputs)

            #computing error rate for back-propagation

            error = training_outputs - output

            #performing weight adjustments

            adjustments = np.dot(training_inputs.T, error * self.sigmoid_derivative(output))

            self.synaptic_weights += adjustments

    def think(self, inputs):

        #passing the inputs via the neuron to get output

        #converting values to floats

        inputs = inputs.astype(float)

        output = self.sigmoid(np.dot(inputs, self.synaptic_weights))

        return output
if __name__ == "__main__":

    #initializing the neuron class
```
The output after running the code:



```
 Beginning Randomly Generated Weights:
 [[-0.16595599]
  [ 0.44064899]
  [-0.99977125]]
 Ending Weights After Training:
 [[10.08740896]
  [-0.20695366]
  [-4.83757835]]
 Considering New Situation:  1 0 0
 New Output data:
 [0.9999584]
 Wow, we did it!
User Input One: 1
User Input Two: 0
User Input Three: 0
```

In this way, we successfully created a simple neural network.

Neurons first assign themselves some random weights, and then use training examples to train themselves.

After that, if a new state [1,0,0] appears, it will get a value of 0.9999584.

This value is very close, and the output value of the Sigmoid function is between 0 and 1.