**Artificial Neural Networks**

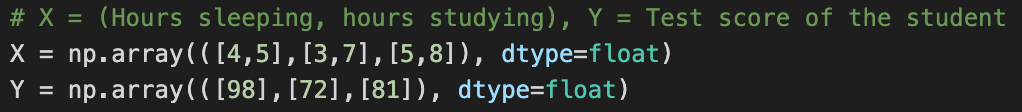
**Jesus Gutierrez**

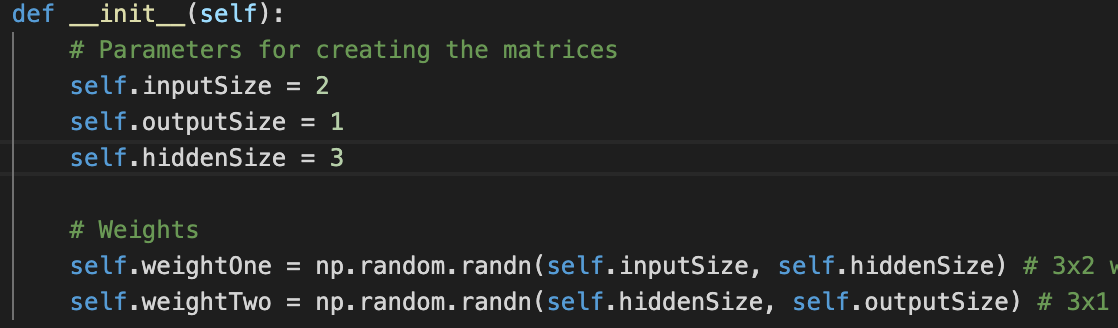
Artificial neural networks are not only useful in building mathematical models of the brain’s activity, but the approach is to solve problems in the same way that a human brain would. These networks or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Over time, attention in this field has moved to performing specific tasks. Artificial neural networks have been used on a variety of tasks such as:

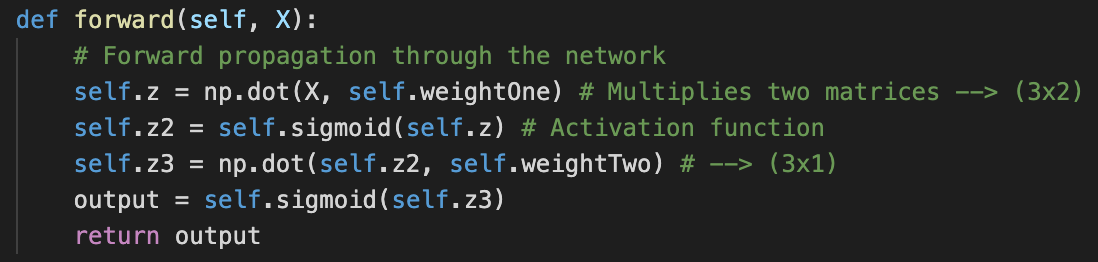
* Image Recognition
* Speech Recognition
* Machine Translation
* Computer Vision
* Medical Diagnosis
* Social Network Filtering
* Playing Board Games/Playing Video Games

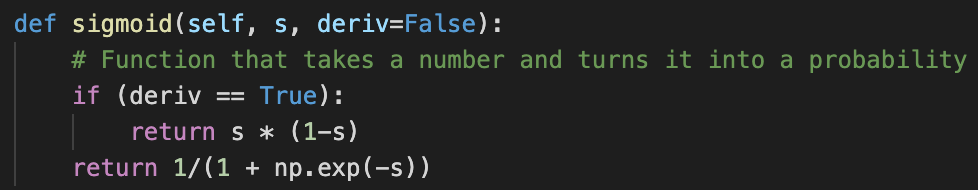
This is one of the main tools used in machine learning. The “neural” part of their name suggests that they are brain-inspired systems which are intended to replicate the way that we humans learn. These are also excellent tools for finding patterns which are too far complex or numerous for a human programmer to extract and teach a machine to recognize. So, how exactly do these neural networks “learn” stuff? Well, in the same way that humans learn. As you see, humans learn from experiences in their lives, neural networks are somewhat the same way, they require data to learn. In most cases, the more data that can be injected at a neural network, the more accurate it will become. A better way to picture this is by thinking of it as any task a human does over and over again. Over time, the human gradually gets more efficient and makes fewer mistakes.

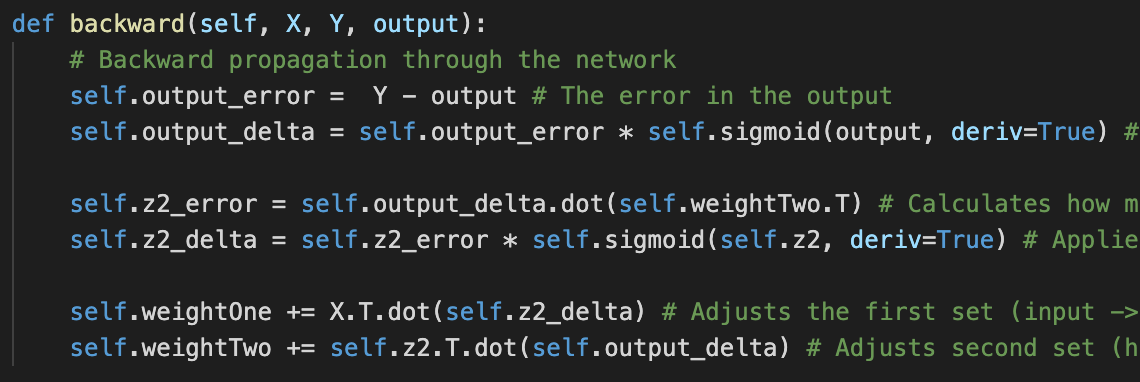
**What has been implemented?**

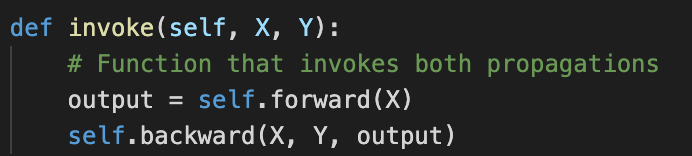
Both the back-propagation and forward-propagation algorithms have been implemented using python. In order to build this neural network, I implemented input data as a pair of values for X and Y. X standing for the number of hours a student is sleeping and studying, and Y standing for the test score of the student as shown in the code below.

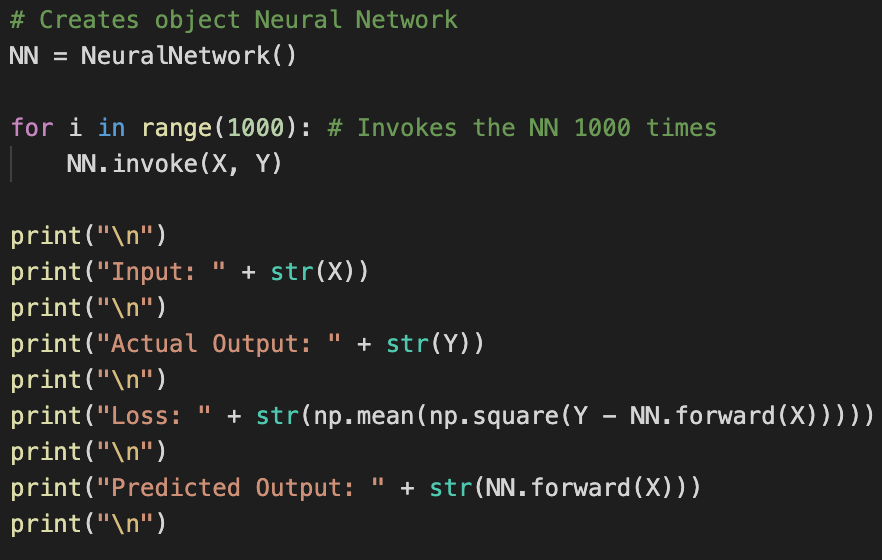
Parameters and weights are also created in order to create the appropriate matrices needed for the forward and backward propagation. Code is shown below:

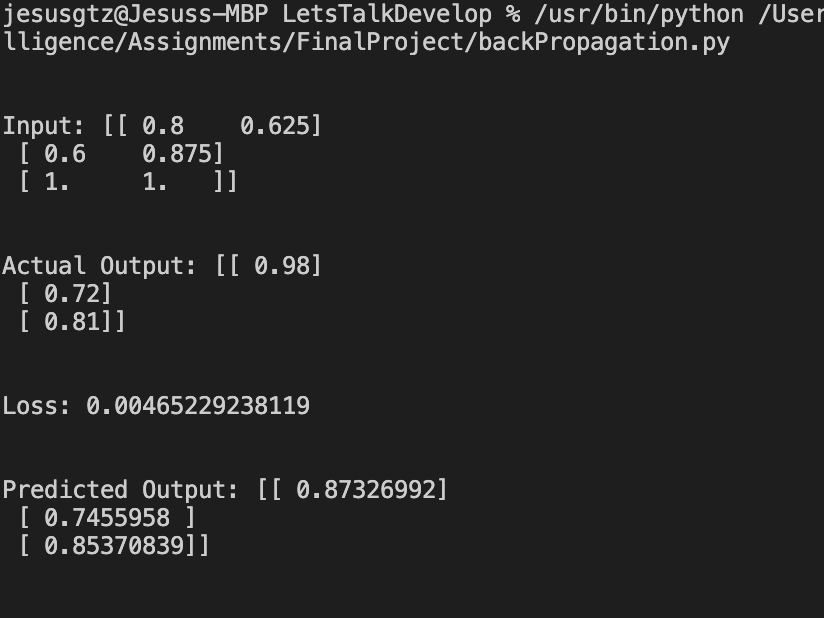
The forward method takes the matrix as a parameter as well as the default in order to conduct the forward propagation through the network that we have created. 

Another function used is sigmoid, this function takes one number as input and a derivative goal parameter as well. All this function does is it takes a number and turns it into a probability or a sigmoid value. Code is shown below:

Another defined function is the backward method. It takes the input matrix, the desired output, and the actual output matrix in order to backward propagate through the network. The main intention for this function is to calculate the errors and to minimize those errors. The errors obtained is nothing but the difference between the desired output that is required and the actual output produced by the network. Code is shown below:

Now a method to invoke the forward and backward propagations is shown below:

Now lastly, create an object of the Neural Network () method and invoke this object 1000 times, and print the results of the neural network’s input, actual output, loss, and predicted output.

In the image shown below is the terminal for the printed data.

**Conclusion**

The challenges that rose with these algorithms was more of learning about how to implement them correctly since I was getting a lot of errors trying to run this. For me, this was the perfect topic because neural networks are being powerfully built and used today, and that interests me. The future someday will be built in neural networks. The code provided is invoked by 1000 iterations, but there is no limit, just like with neural networks.

**References**

*Chapter 18.7 “Artificial Neural Networks” in book Artificial Intelligence, A Modern Approach Third Edition by Stuart J. Russell & Peter Norvig*