**Classification of Iris Dataset using Single Layer Perceptron (SLP)**

Vidiskiu Fortino Kurniawan

16/395999/PA/17340

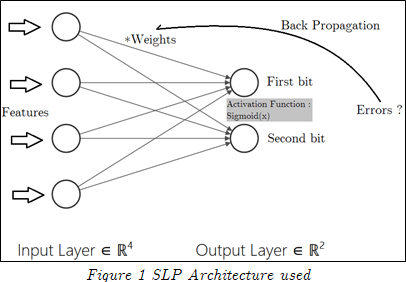
**Introduction**

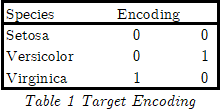
Single layer perceptron (SLP) is a neural network with only 1 ‘active’ layer, the output layer aside from its input layer which has no calculation. Typically, the input nodes are fully connected to the nodes in the output layer in a SLP.

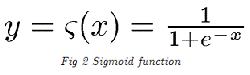
Iris dataset is one of the benchmark dataset researchers of machine learnings use to test their models. It contains 3 classes of 50 records each, which label types of iris plant. It is known that “one class is linearly separable from the other 2; the latter are NOT linearly separable from each other.”[1]

This experiment will attempt to create a Python implementation of SLP that will predict the class of iris plant based on its features.

**Method**

The SLP architecture used is fully-connected SLP with target labels encoded to 2 bits.





No partitioning of dataset is done in this experiment, therefore the accuracy and error taken for each epoch came from the 150 dataset that act as the training data.

100 epoch’s cycles was used in this experiment, with learning rate of 0.1 and 0.8 as a result comparison.

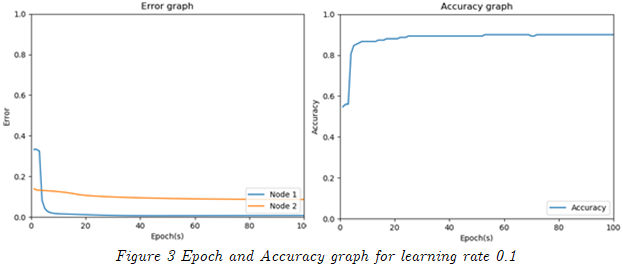
Initial weights for each node’s connection was randomized in excel for task 1 and was used as the initial weights for this experiment. However, the code to generate random values from normal distribution is included in the documentation. The purpose of the same initial weights was just to check the functionality of the program itself, comparing it with the excel results.

No other libraries were used besides the numpy library and the matplotlib.pyplot library.

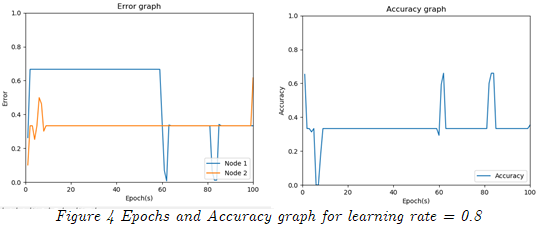
* Numpy : Various mathematical operations, as well as the implementation for array data structure.
* Matplotlib.pyplot : Graph plotting library.

The data structure used is mostly matrix, to ease the mathematical computation in the training process of the SLP.

**Experiment**

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Avg Error1** | **Min Avg Error1** | **Avg Error2** | **Min Avg Error2** | **Avg Acc** | **Max Acc** |
| 0.01945072 | 0.00665032 | 0.09773137 | 0.08661405 | 0.8811333333333343 | 0.9 |

****

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Avg Error1** | **Min Avg Error1** | **Avg Error2** | **Min Avg Error2** | **Avg Acc** | **Max Acc** |
| 0.51138595 | 0.00764463 | 0.10179511 | 0.33566129 | 0.3429999999999998 | 0.66 |

**Documentation**

The code listing is available in the following github link :

Additional processes have been explained in the inline codes. Notable different implementation includes the data structure chosen. Input nodes are recorded as 5 as bias factors must be taken into considerations.

**Conclusion**

Using the fixed initial weights as stated in the methodology, learning rate 0.1 shows better results in comparison to learning rate 0.8. With input data of 0 – 10, learning rate of 0.8 may have exhibit the characteristic of being stuck in the learning curve, unable to converge to the desired weights; hence exhibiting worse results. The errors and accuracy oscillates with similar points, hinting this property. Therefore, there are exist better algorithms to choose learning rate such as scheduled decay learning rate. However, through the check with other lower learning rates, the python algorithm can be concluded to be correct.

To improve results, there exist various other methods that can be implemented in future implementations, such as data pre-processing as well as learning rate decay.

**References**

[1] Marshall, M., & Fisher, R. A. (1988, July 01). Iris Data Set. Retrieved March 5, 2019, from https://archive.ics.uci.edu/ml/datasets/iris