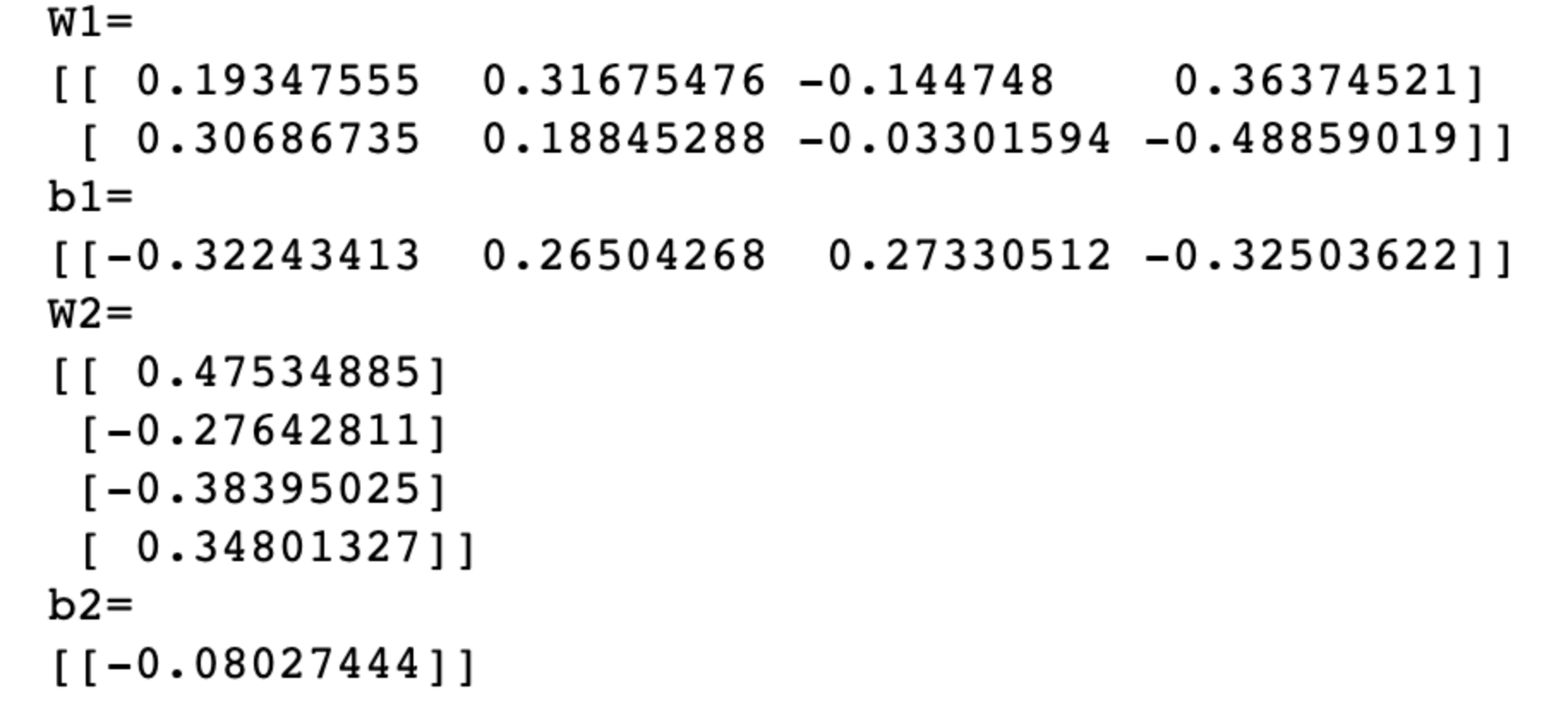
**Neural Network Computing**

**Final Project**

*Muhammad Osama Khan (mok232)*

**Part 1: XOR Weights Validation**

Using the initial weights and biases provided, the following weights and biases were obtained after 1 epoch of training when using a 2-4-1 neural network architecture with the following hyperparameters:



These values were validated with the GA before moving on to the rest of the experiments.

**Part 2a: Varying (Quadratic Cost Function)**

In this part of the project, the quadratic cost function was used and the following hyperparameter combinations were tested:

Hence, a total of 27 hyperparameter combinations were tried. Note that for this part, the NN architecture was kept fixed at 2-4-1 in order to observe the effect of the different learning rates, weights and bias initializations and slope of the transfer function. The results are summarized below:

*Note: Throughout all the experiments, tolerance was kept fixed at 0.05 and max number of epochs was set to 700 after empirically trying several different combinations.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Final Epoch Error | Convergence | # Training Epochs |
| 0.1 | 0.5 | 0.5 | 0.0496 | Yes | 172 |
| 0.1 | 0.5 | 1 | 0.0499 | Yes | 661 |
| 0.1 | 0.5 | 1.5 | 4.0497 | No | 700 |
| 0.1 | 1 | 0.5 | 0.0495 | Yes | 120 |
| 0.1 | 1 | 1 | 0.0499 | Yes | 535 |
| 0.1 | 1 | 1.5 | 1.3302 | No | 700 |
| 0.1 | 1.5 | 0.5 | 0.0499 | Yes | 109 |
| 0.1 | 1.5 | 1 | 0.0499 | Yes | 478 |
| 0.1 | 1.5 | 1.5 | 0.0852 | No | 700 |
| 0.2 | 0.5 | 0.5 | 0.0496 | Yes | 70 |
| 0.2 | 0.5 | 1 | 4.205 | No | 700 |
| 0.2 | 0.5 | 1.5 | 4.0934 | No | 700 |
| 0.2 | 1 | 0.5 | 0.0493 | Yes | 48 |
| 0.2 | 1 | 1 | 0.05 | Yes | 316 |
| 0.2 | 1 | 1.5 | 0.0515 | No | 700 |
| 0.2 | 1.5 | 0.5 | 0.0498 | Yes | 45 |
| 0.2 | 1.5 | 1 | 0.0499 | Yes | 428 |
| 0.2 | 1.5 | 1.5 | 0.05 | Yes | 517 |
| 0.3 | 0.5 | 0.5 | 0.0487 | Yes | 38 |
| 0.3 | 0.5 | 1 | 4.3105 | No | 700 |
| 0.3 | 0.5 | 1.5 | 4.1365 | No | 700 |
| 0.3 | 1 | 0.5 | 0.0492 | Yes | 28 |
| 0.3 | 1 | 1 | 0.0496 | Yes | 219 |
| 0.3 | 1 | 1.5 | 0.0499 | Yes | 491 |
| 0.3 | 1.5 | 0.5 | 0.0482 | Yes | 32 |
| 0.3 | 1.5 | 1 | 0.0499 | Yes | 129 |
| 0.3 | 1.5 | 1.5 | 0.0499 | Yes | 431 |

We see that 19 out of the 27 hyperparameter combinations tested lead to convergence. With max number of epochs set to 700, the learning rate **α** does not seem to affect how many hyperparameter combinations converge (6-7 convergence results in each set of 9 rows with **α**=0.1, **α**=0.2 and **α**=0.3). However, the learning rate **α** does affect the *rate of convergence* if the experiments converge*.* For instance, consider the hyperparameter combinations indicated by the rows highlighted in yellow. With the same settings for **,** increasing the learning rate decreases the number of training epochs required for convergence. Note however that increasing the learning rate leads to greater risk of skipping minima during the gradient descent process, which may lead to divergence. As a further note, even though have the same values in the rows considered above, they might not necessarily yield the same results every time since the weights and biases are randomly initialized.

Furthermore, it is observed that using relatively low initialization values of the weights and biases (e.g. = 0.5) decreases the chance of convergence. This can be seen from the rows highlighted in green where increasing decreases the number of epochs needed for convergence.

Lastly, it was observed that increasing (slope argument of the transfer function) decreased the rate of convergence. This can be seen, for example, from the rows highlighted in orange. The following graphs for the bipolar sigmoid transfer function help explain why this is the case. The red, blue and green curves represent =0.5**,** =1 and =1.5 respectively. Since the slope of the red curve is greater, this means that lower values of =0.5 lead to faster rates of convergence as was confirmed empirically by the results above.

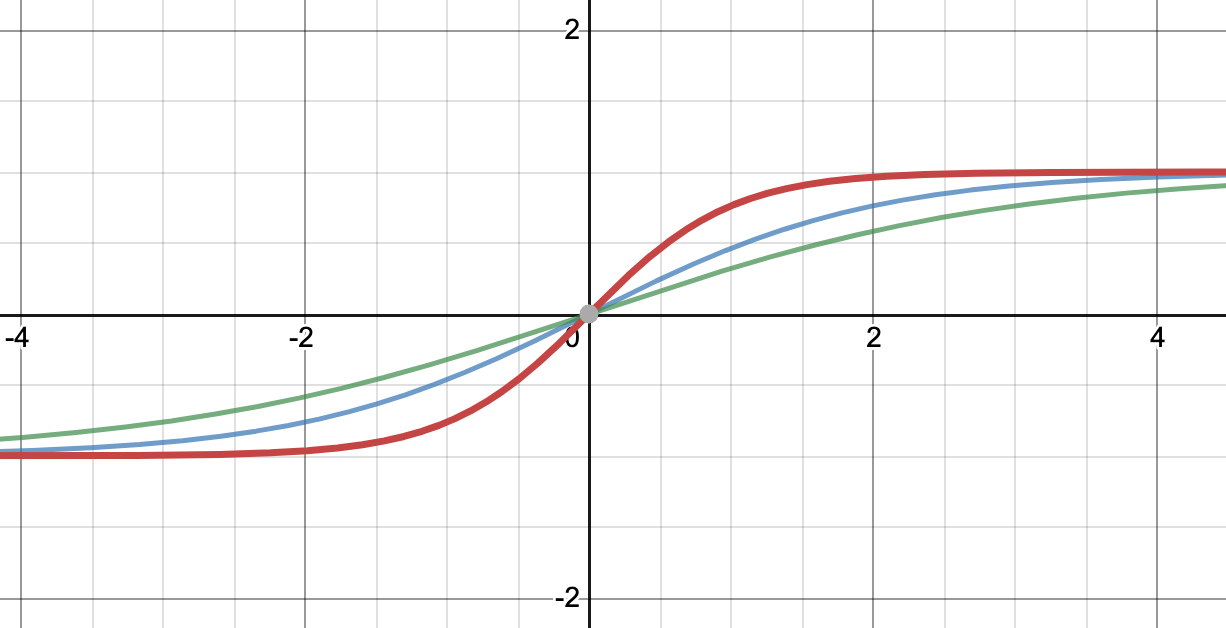


Figure 1 Bipolar sigmoid transfer functions with different values of . The red, blue and green curves represent =0.5**,** =1 and =1.5 respectively.

**Part 2b: Varying (hidden layer neurons) (Quadratic Cost Function)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | Final Epoch Error | Convergence | # Training Epochs |
| 0.1 | 0.5 | 0.5 | 0.0478 | Yes | 75 |
| 0.1 | 0.5 | 1 | 4.205 | No | 700 |
| 0.1 | 0.5 | 1.5 | 4.1366 | No | 700 |
| 0.1 | 1 | 0.5 | 0.0486 | Yes | 51 |
| 0.1 | 1 | 1 | 0.0493 | Yes | 153 |
| 0.1 | 1 | 1.5 | 0.0493 | Yes | 275 |
| 0.1 | 1.5 | 0.5 | 0.0483 | Yes | 25 |
| 0.1 | 1.5 | 1 | 0.0497 | Yes | 114 |
| 0.1 | 1.5 | 1.5 | 0.0497 | Yes | 181 |
| 0.2 | 0.5 | 0.5 | 0.0495 | Yes | 52 |
| 0.2 | 0.5 | 1 | 4.4182 | No | 700 |
| 0.2 | 0.5 | 1.5 | 4.2751 | No | 700 |
| 0.2 | 1 | 0.5 | 0.0472 | Yes | 26 |
| 0.2 | 1 | 1 | 0.0481 | Yes | 93 |
| 0.2 | 1 | 1.5 | 0.0484 | Yes | 187 |
| 0.2 | 1.5 | 0.5 | 0.0427 | Yes | 12 |
| 0.2 | 1.5 | 1 | 0.0484 | Yes | 34 |
| 0.2 | 1.5 | 1.5 | 0.0497 | Yes | 177 |
| 0.3 | 0.5 | 0.5 | 5.3024 | No | 700 |
| 0.3 | 0.5 | 1 | 4.638 | No | 700 |
| 0.3 | 0.5 | 1.5 | 4.4182 | No | 700 |
| 0.3 | 1 | 0.5 | 0.047 | Yes | 9 |
| 0.3 | 1 | 1 | 0.05 | Yes | 58 |
| 0.3 | 1 | 1.5 | 0.0478 | Yes | 114 |
| 0.3 | 1.5 | 0.5 | 0.0381 | Yes | 9 |
| 0.3 | 1.5 | 1 | 0.0476 | Yes | 42 |
| 0.3 | 1.5 | 1.5 | 0.0492 | Yes | 96 |

In summary, the directory structure looks as follows:

* Project\_1 (main folder)
  + *Cannys\_Edge\_Detector.py (python file containing source code)*
  + *Cannys\_Edge\_Detector.ipynb (Jupyter Notebook file containing source code and image visualizations)*
  + *Cannys\_Edge\_Detector.html* (html file containing Jupyter Notebook results)
  + Zebra-crossing-1.bmp (test image)
  + Houses-225.bmp (test image)
  + Houses-225 (directory for saving canny edge detector results for this image)
  + Zebra-crossing-1 (directory for saving canny edge detector results for this image)

Note: The above 2 directories will be automatically created by the code if they do not exist. In order to save new image results in these folders, delete these 2 folders and rerun the Python or Jupyter Notebook files.

1. Running the program:
   1. The Python file can be run by running “python Cannys\_Edge\_Detector.py” from the terminal.
   2. The Jupyter notebook file can be run by opening the Jupyter Notebook locally (make sure you have Jupyter Notebook installed). Click within the cell and press “Shift+Enter” to run the corresponding cell. The results have also been displayed in the Jupyter Notebook file for convenience.

Libraries required: numpy, matplotlib, cv2.

* + pip install numpy
  + pip install matplotlib
  + pip install opencv

Note: Python 3 is used.

1. Source Code and Edge Detection Results for the 2 test images:

*Note: T1 = 5, T2 =10 for the houses image; T1 = 8, T2 = 16 for the zebra image.*