Josh Engelsma

Adam Terwilliger

April 5, 2016

CIS 678 – Machine Learning

Project 4

Abstract

We dive into more advanced supervised learning in CIS 678 – Machine Learning, by implementing a neural network from scratch. Using a sigmoid function, we have implemented forward propagation, and backwards propagation using gradient descent to train a neural network on three distinct datasets, each foundational in machine learning. We looked into classification of Iris (4 features, 3 targets), Cancer (30 features, 1 target), and Wine (10 features, 3 targets). Our NN implementation differentiates itself in three main ways: vectorization, bi-variate parameter design of experiments, and animation of training. We devolved into these distinct sub-areas, while still maintain 95-99% classification rates on both in-class datasets and the three ML datasets.

Implementation

Our program is written in Python 2.7, R 3.1.2, and bash scripting in Unix. These programs were executed locally on each member's respective Macbook Pro (2012), testing on eos23 and okami.

Datasets

Two in-class datasets involved fishing, but testing was too variable with too few observations and game data had four features and looked to predict the action of the game character. Additionally, we used the foundational Iris dataset containing four features, 150 observations, and 3 target flowers. Using the UC-Irvine repository, we uncovered two datasets: Wine (classify 3 types of wine using 10 features and 178 obs.) and Cancer (classify benign or malignant using 30 features and 569 obs.). Each dataset had character features that were discretized and continuous features that were normalized between 0 and 1.

Background

Where do neural networks come from? They are inspired by the neuron from the human brain, as seen in Figure 2.

How do neural networks work?

A neural network, example in Figure 3, maps its inputs to its outputs via multiple non-linear transformations propagated through multiple hidden layers.[1] This process is accomplished by iteratively calculating a weighted sum of the edges to and passed through an activation function. The neural network is able to "learn" through gradient descent and back-propagation of error by updating the edge weights after each iteration.

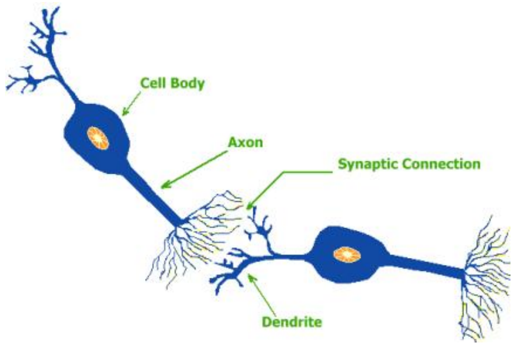


Figure 1. Neuron from Human Brain

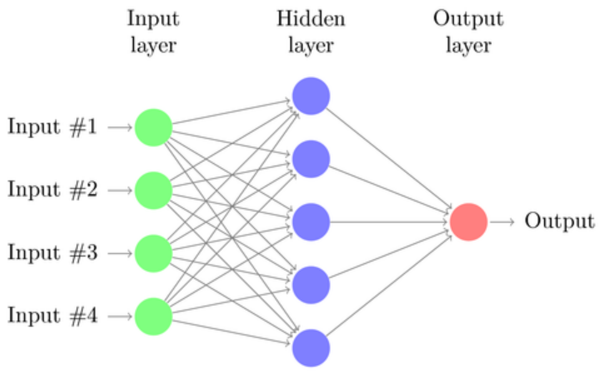


Figure 2. Simple Neural Network

Results

The best relative classification rate vs. training/test split as seen in Figure 3 that our neural network performed was 97.3% for Iris using a 75/25 training/test split, 98.0% for Cancer using a 40/60 training/test split, and 98.9% for Wine using a 50/50 training/test split. We have demonstrated the power of our neural network with high classification rates on reasonable training/test splits. We can infer from the number of observations for Cancer relative to Iris where we only need 40% of the dataset to train, while similarly, the 50/50 with Wine shows the value of adding additional features. Additionally, in Figures 4 and 5, we observe a correlation between classification rate and number of epochs/learning rate. We note a "tipping point" in learning rate around 75% where the classification rate was optimal for each dataset. Similarly, we observe a "tipping point" for number of epochs around log base 2 of 5 ~= 30 epochs.

Figure 3. Neural Network Classification Rate vs. Training/Test Split

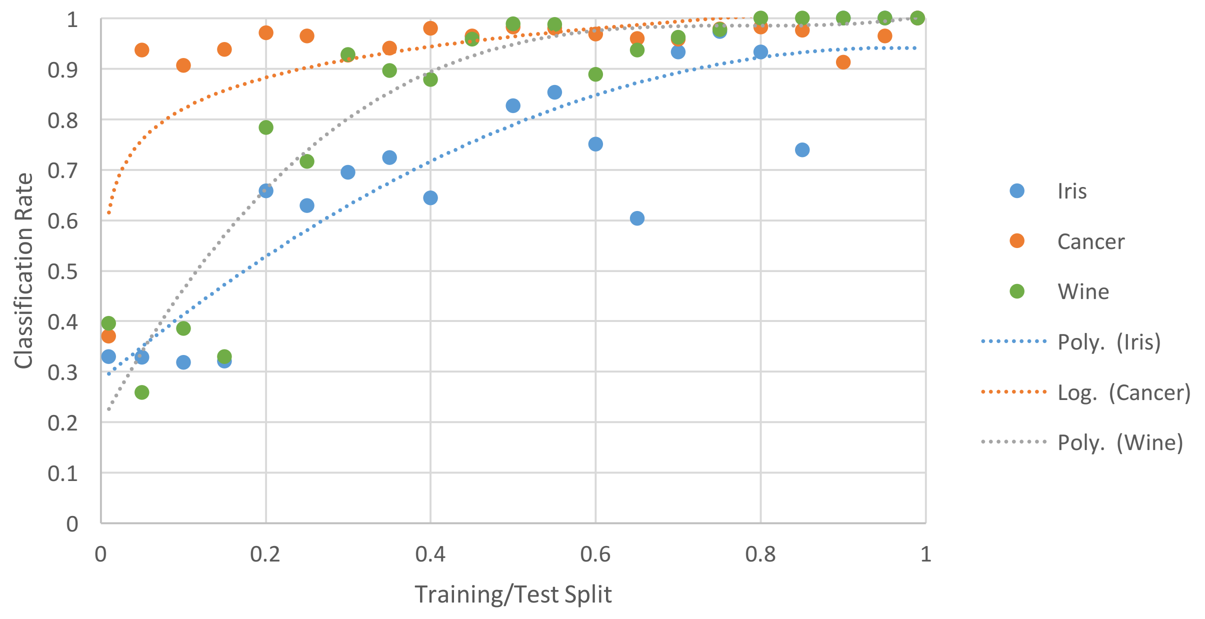


Figure 4. Neural Network Classification Rate vs. Learning Rate

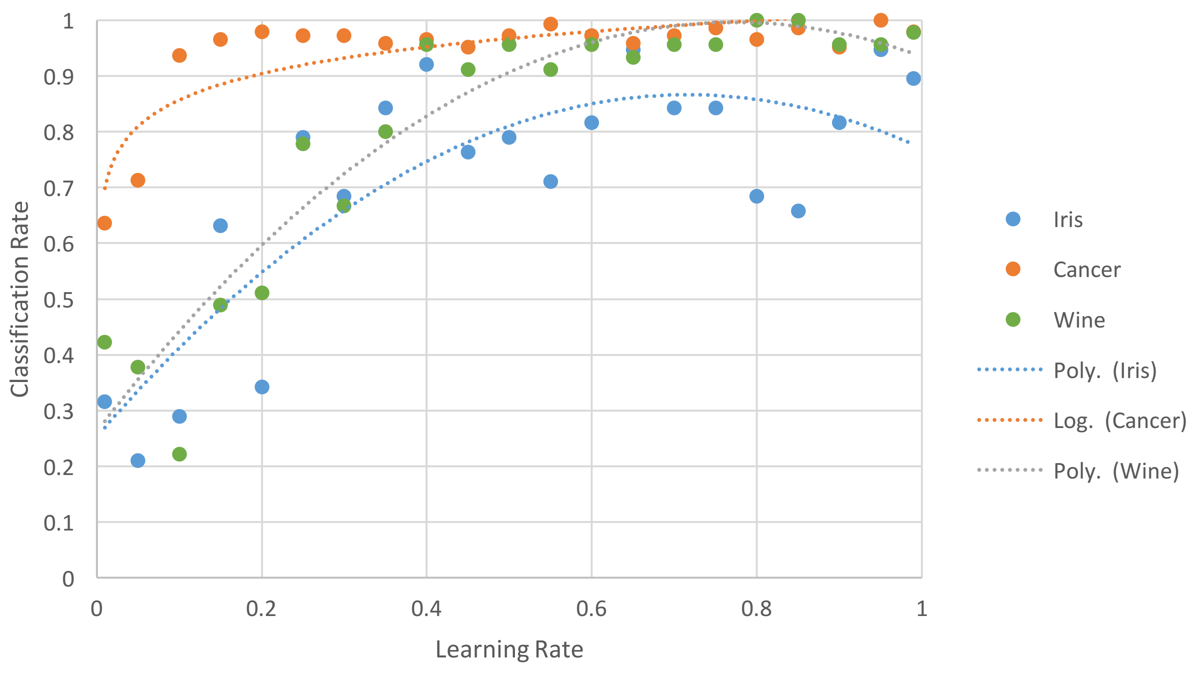
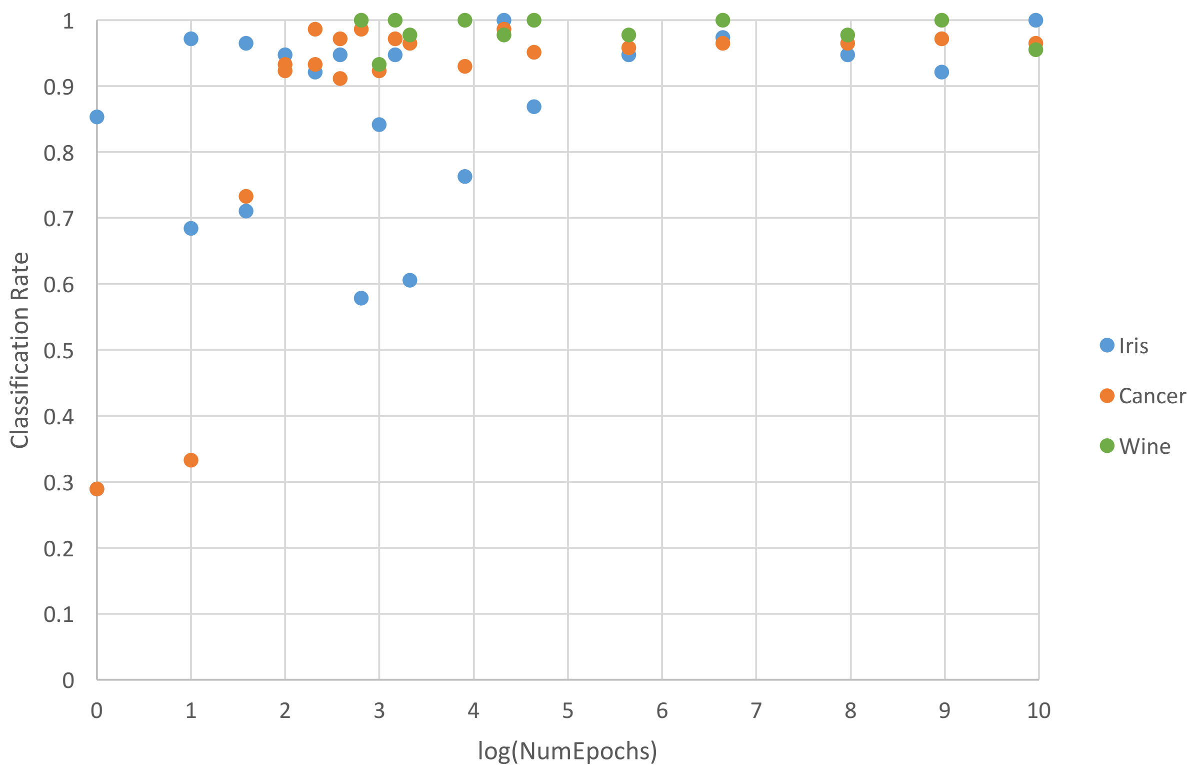


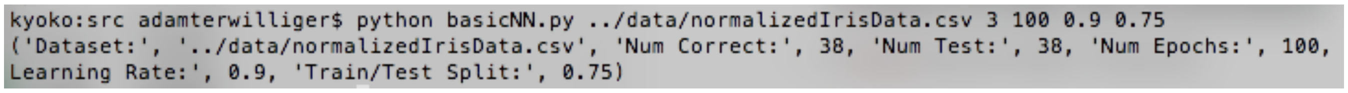
Figure 5. Neural Network Classification Rate vs. log(NumEpochs)



Discussion

As previously noted, our implementation differentiates itself in three main ways: vectorization, bi-variate parameter design of experiments, and animation of training.

- Vectorization: by utilized numpy and linear algebra techniques, we avoid any extraneous loops by implementing forward and back propagation using vector and matrix operators. This made the code cleaner, faster, and more expandable (passing parameters to program via command line)



```
kyoko:src adamterwilliger$ python basicNN.py ../data/normalizedIrisData.csv 3 100 0.9 0.75
('Dataset:', '../data/normalizedIrisData.csv', 'Num Correct:', 38, 'Num Test:', 38, 'Num Epochs:', 100,
Learning Rate:', 0.9, 'Train/Test Split:', 0.75)
```

Figure 6.

- Bi-variate parameter design of experiments
  - We ran a couple thousands trials of possible combinations of learning rate, number of epochs, and train/test splits, with each iteration controlling for one of the three variables. This allowed us to uncover any inherent relationships among these parameters, as seen in Figures 7-9. Additionally, we are able to narrow down the best set of parameters to utilize for the neural network given a dataset.

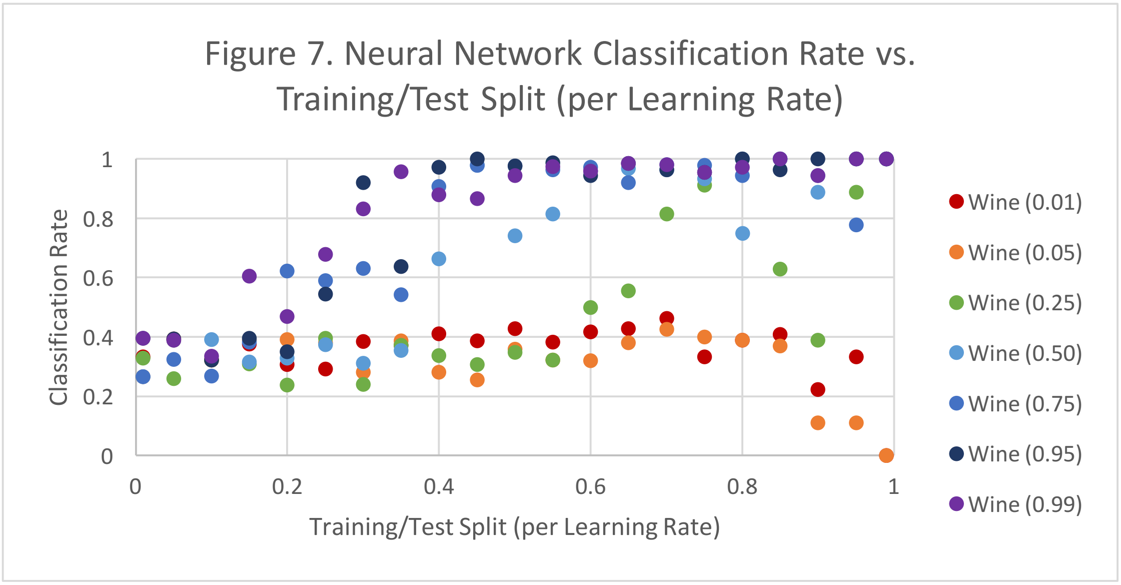


Figure 8. Neural Network Classification Rate vs. Training/Test Split (per Num Epochs)

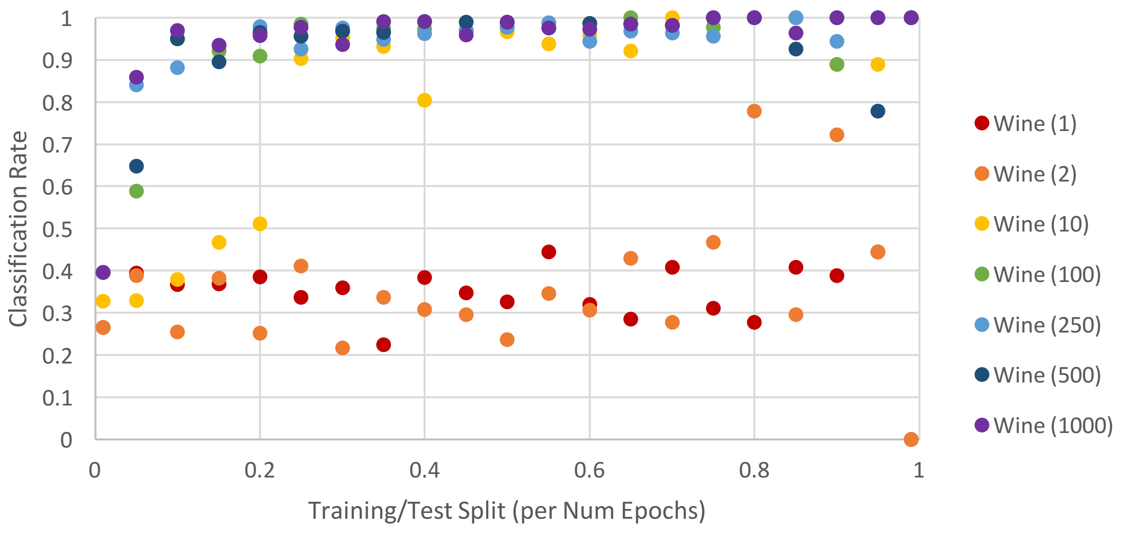
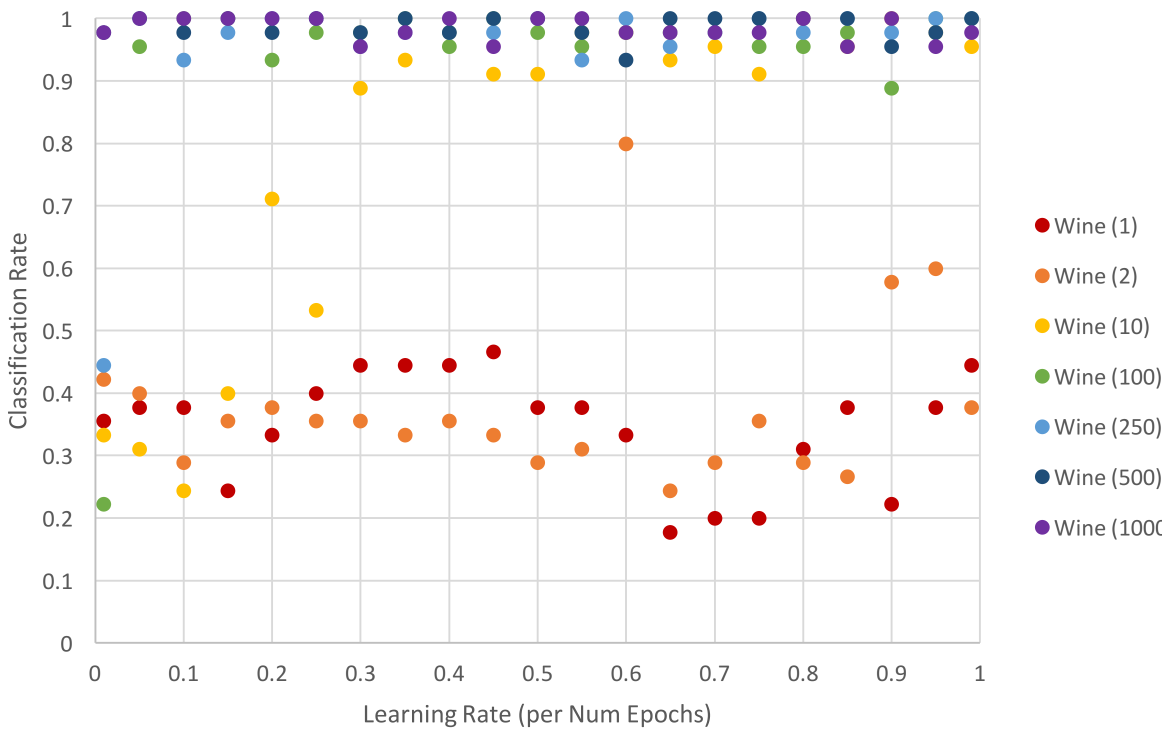
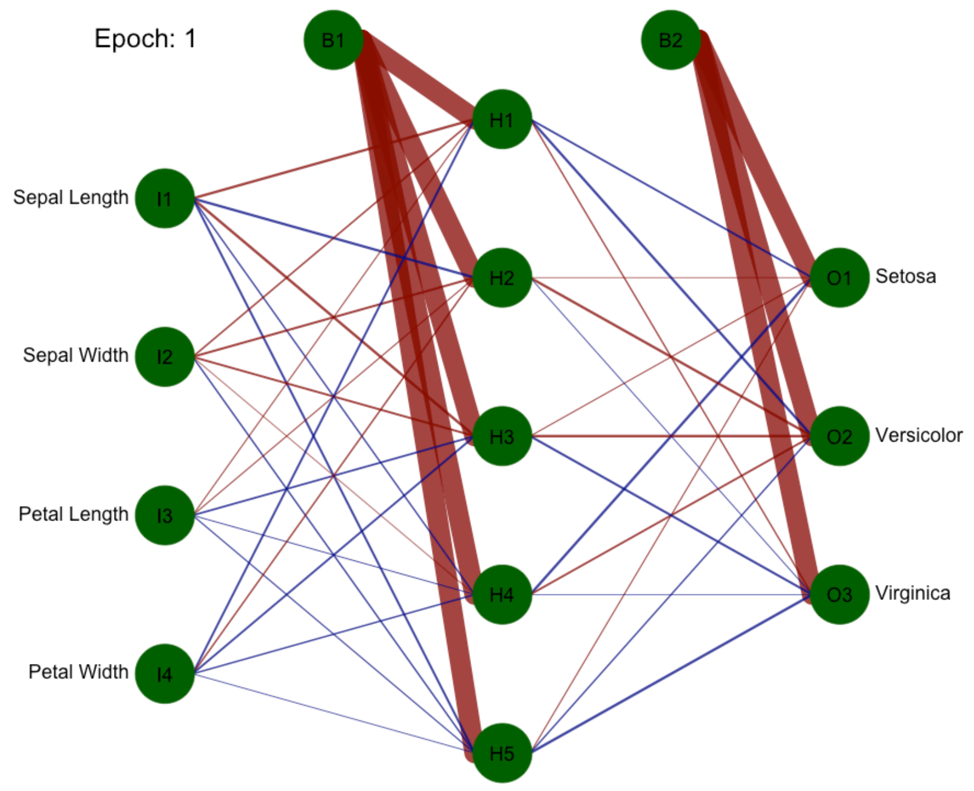
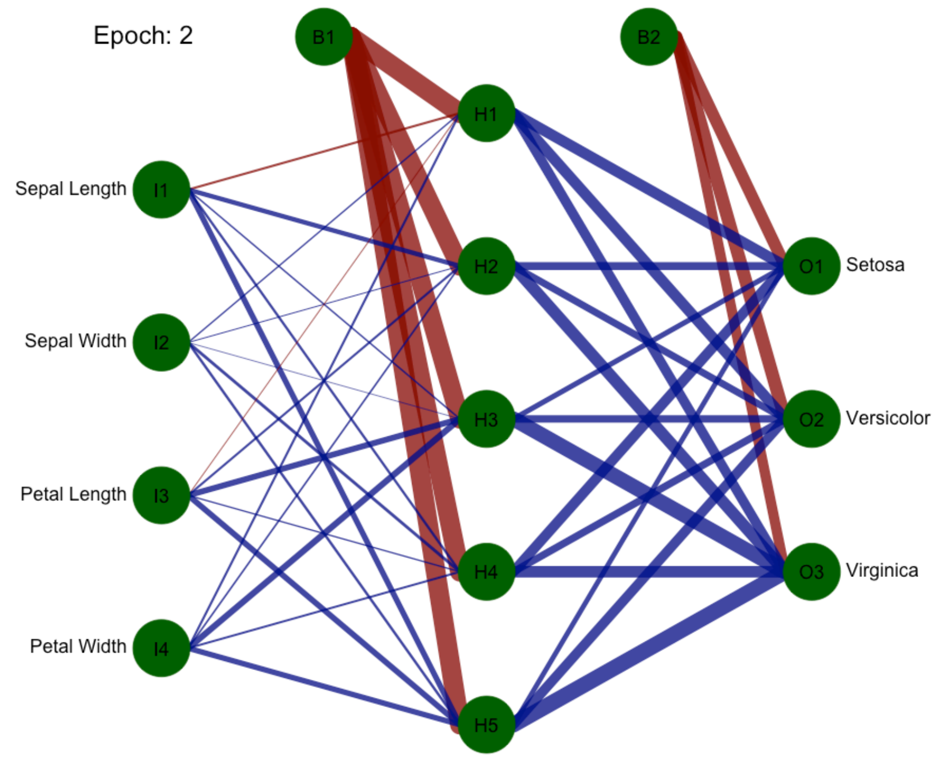


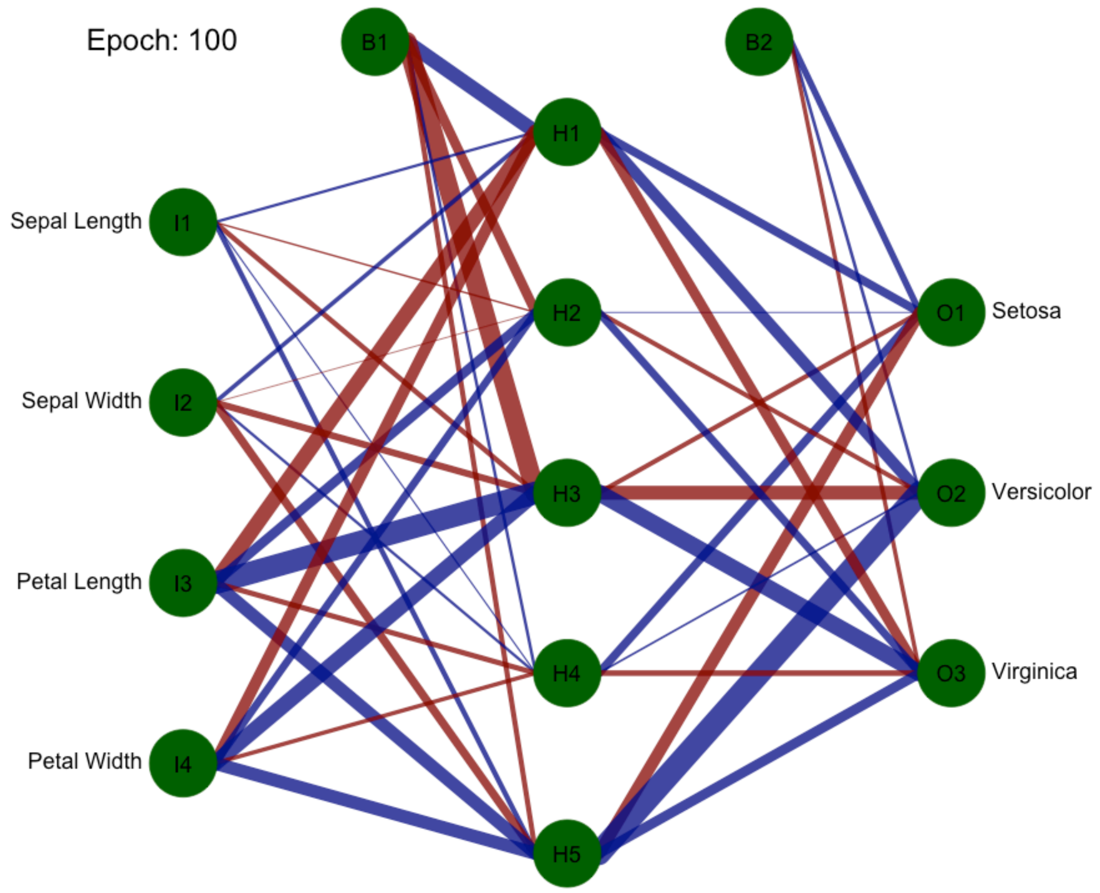
Figure 9. Neural Network Classification Rate vs. Learning Rate (per Num Epochs)



Figures 10 – 12. States of Neural Network during Training







Figures 10-12 represent states of the neural network during training with red as positive and blue as negative weights with width showing a larger magnitude. We can observe after the first epoch the effects of randomized weights and heavy bias weights. After the second epoch, the neural looks to overcompensate for initial weight error and biases. In Epoch 100, we can visualize a converged state of the network. These images are combined as a gif using R and prove its value as a tool for debugging neural network implementations for others.

Future Work

We would explore ideas of using different activation functions, visualizing more networks using animations, deep learning (adding hidden layers), and more extensive design of experiments with thousands to millions of trials to validate with statistical confidence these relationships.