Patryk Dziedzic and Mathew Umano
Dr. Cowan
CS440 - Introduction to Artificial Intelligence
14 December 2023

Project 3 Writeup - Data and Analysis

The First Task

Description

- When thinking about the First Task of this project, one must consider the inputs and outputs required. The model created will determine whether a wiring diagram can be considered as the binary representation of dangerous or not dangerous. The machine learning model will interpret this data and make a guess as to whether the output should be dangerous or not dangerous, true or not true. This binary classification led us to use Logistic Regression to build our machine learning model. For the purpose of understanding the intricate specifications of the model, it must be noted that the static variables used were the learning rate alpha, $\boldsymbol{\alpha}$, with value of $\boldsymbol{\alpha}$ = .015, and the initial weight, **w**, with value of **w** = .001. The dynamic variables in Task One, which will be discussed in the analysis portion, includes **numTest**, **numTrain**, and **numTimesTrainAtOnce**. Note that **numTimesTrainAtOnce** has no effect on the accuracy of the model, but instead allows us to reduce the amount of data points produced by the model. In other words, for some numbers **numTest** and **numTrain,** it might be useful to increase the **numTimesTrainAtOnce** so that the model is trained more before calculating the loss and there are less data points that still show the trends of testing and training loss on the graph; **numTest** and **numTrain** are the only variables that have an actual effect on the training and testing loss of this machine learning model.
- Our images are generated by shuffling the order of “R”, “G”, “B”, and “Y” and then iterating in that shuffled order to populate random rows/columns of the image. With a 50% chance, the program will either start by populating a row or by populating a column, as per the project specifications. It is then stored with a value indicating whether or not it is dangerous based on if “R” came before “Y” in the shuffled order.

Input Space

- The input space is a 20x20 wire diagram image of pixels. Each pixel corresponds to a color: {Red, Green, Blue, Yellow} encoded as a 1-hot vector of size 4. Red is [1,0,0,0], Green is [0,1,0,0], Blue is [0,0,1,0], Yellow is [0,0,0,1], and a pixel with no color is [0,0,0,0]. This is flattened into an array of size 1601, with index 0 being the value 1. This image has a corresponding boolean value, 1 representing the image is "dangerous" and 0 representing the image is "not dangerous" or "safe." An image is dangerous if a red wire has been placed under a yellow wire.

Output Space

- The output space is a double value between 0 and 1 indicating the probability that the image is dangerous.

Model Space

- Using Logistic Regression and Soft Classification, the model space is defined to be the sigmoid function:

  $\sigma(w_0 x_0 + w_1 x_1 + \ldots + w_D x_D) = \sigma(\underline{w} \cdot \underline{x}) = \frac{1}{1+e^{-(\underline{w} \cdot \underline{x})}}$

  where the parameter $\underline{w}$ is the weight vector, the parameter $\underline{x}$ is the image vector with $x_0 = 1$, and there are D total pixels in an image.

Loss Function

- The Loss function is the Binary Cross Entropy Loss derived in class:

  $Loss = \frac{1}{N} \sum_{i=1}^{N} [- y_i \ln(f(\underline{x}^i)) - (1 - y_i) \ln(1 - f(\underline{x}^i))]$

  where $f(\underline{x}^i) = \sigma(\underline{w} \cdot \underline{x}^i)$ for a given i^{th} image in N images.

Training Algorithm

- We used Stochastic Gradient Descent for our training algorithm. The derivation for the gradient of the Loss function is as follows:

$$\frac{\partial}{\partial w_j} Loss(f_{\underline{w}}(\underline{x}), y) = -y\frac{\frac{\partial}{\partial w_j} f_{\underline{w}}(\underline{x})}{f_{\underline{w}}(\underline{x})} - (1-y)\frac{\frac{\partial}{\partial w_j}[1-f_{\underline{w}}(\underline{x})]}{1-f_{\underline{w}}(\underline{x})}$$

$$= -y\frac{g'(\underline{w}\cdot\underline{x})x_j}{f_{\underline{w}}(\underline{x})} - (1-y)\frac{-g'(\underline{w}\cdot\underline{x})x_j}{1-f_{\underline{w}}(\underline{x})}$$

$$= -[y\frac{1}{f_{\underline{w}}(\underline{x})} - (1-y)\frac{1}{1-f_{\underline{w}}(\underline{x})}]g'(\underline{w}\cdot\underline{x})x_j$$

$$= -[y(\frac{1}{f_{\underline{w}}(\underline{x})} + \frac{1}{1-f_{\underline{w}}(\underline{x})}) - \frac{1}{1-f_{\underline{w}}(\underline{x})}]g'(\underline{w}\cdot\underline{x})x_j$$

$$= -[y(\frac{1}{f_{\underline{w}}(\underline{x})(1-f_{\underline{w}}(\underline{x}))}) - \frac{1}{1-f_{\underline{w}}(\underline{x})}]g'(\underline{w}\cdot\underline{x})x_j$$

$$= -[y - f_{\underline{w}}(\underline{x})]\frac{g'(\underline{w}\cdot\underline{x})}{f_{\underline{w}}(\underline{x})(1-f_{\underline{w}}(\underline{x}))}x_j$$

Where $g'(z) = g(z)(1-g(z))$ and $g'(\underline{w}\cdot\underline{x}) = f_{\underline{w}}(\underline{x})(1-f_{\underline{w}}(\underline{x}))$. Therefore:

$$= [f_{\underline{w}}(\underline{x}) - y]x_j$$

And the gradient is:

$$\nabla_{\underline{w}} Loss(f_{\underline{w}}(\underline{x}), y) = [f_{\underline{w}}(\underline{x}) - y]\,\underline{x}$$

Using this gradient, our SGD algorithm is as follows:

- Choose a learning rate $\boldsymbol{\alpha} > 0$.
- Initialize the weights of the weight vector $\underline{w}$ to an initial guess, usually near $\underline{0}$. Make $\underline{w}(0)$ be the value 1.
- Choose a random i^{th} image $(\underline{x}, y) \in Data$. Update the parameters ($\underline{w}$) for k iterations based on how poorly the model performs on image i:

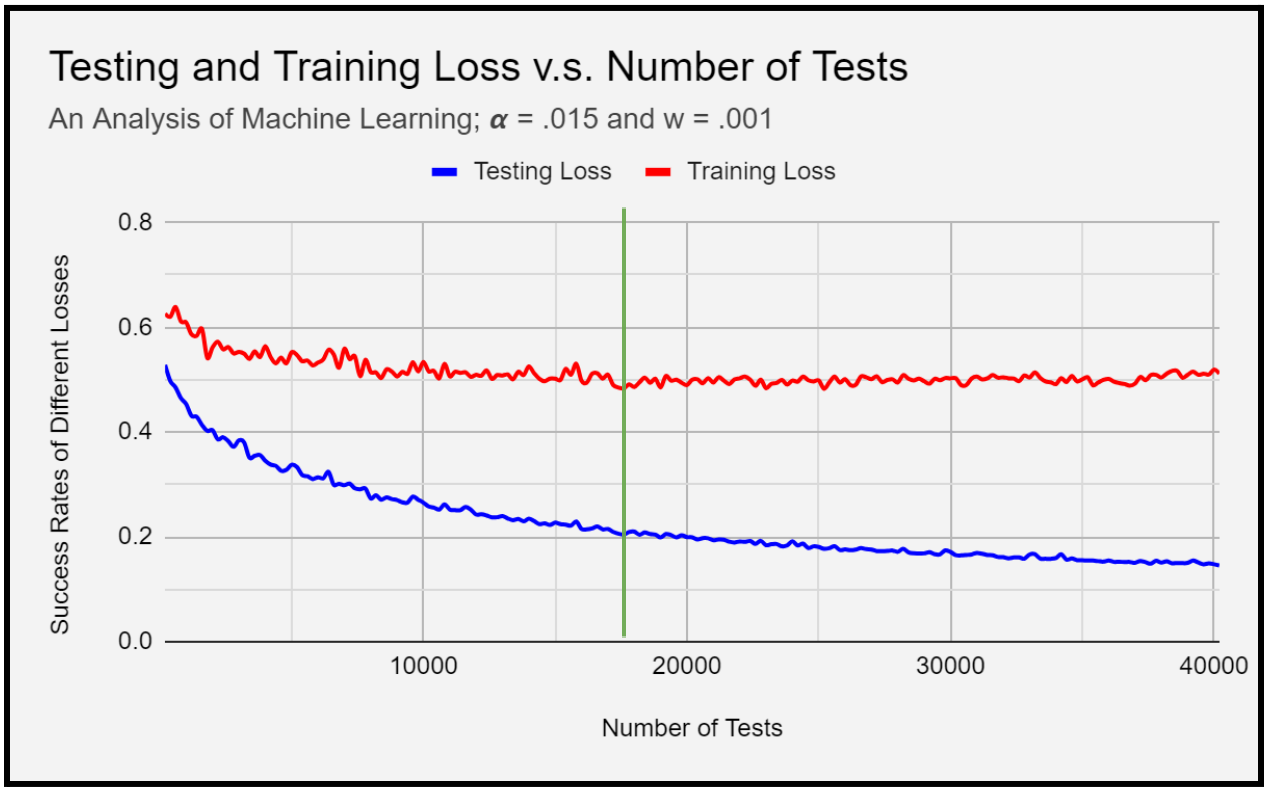
$$\underline{w}(k+1) = \underline{w}(k) - \alpha[f_{\underline{w}}(\underline{x}) - y]\,\underline{x}$$

- Repeat.

Overfitting

- We prevent overfitting by training the model until the training loss and the testing loss begin to diverge. This was found at a unique approximate value of **trainingLoss** for each graph that we stored in the **trainingLossCutoff** variable. Once the model reaches this number, we would stop training the model to prevent divergence. However, we continue until the specified **divergenceCutoff** for the graphs below to display how the training and testing losses diverge. In actual practice, the model would terminate training at the **trainingLossCutoff**, shown by the vertical green line.

Results (Graphs)

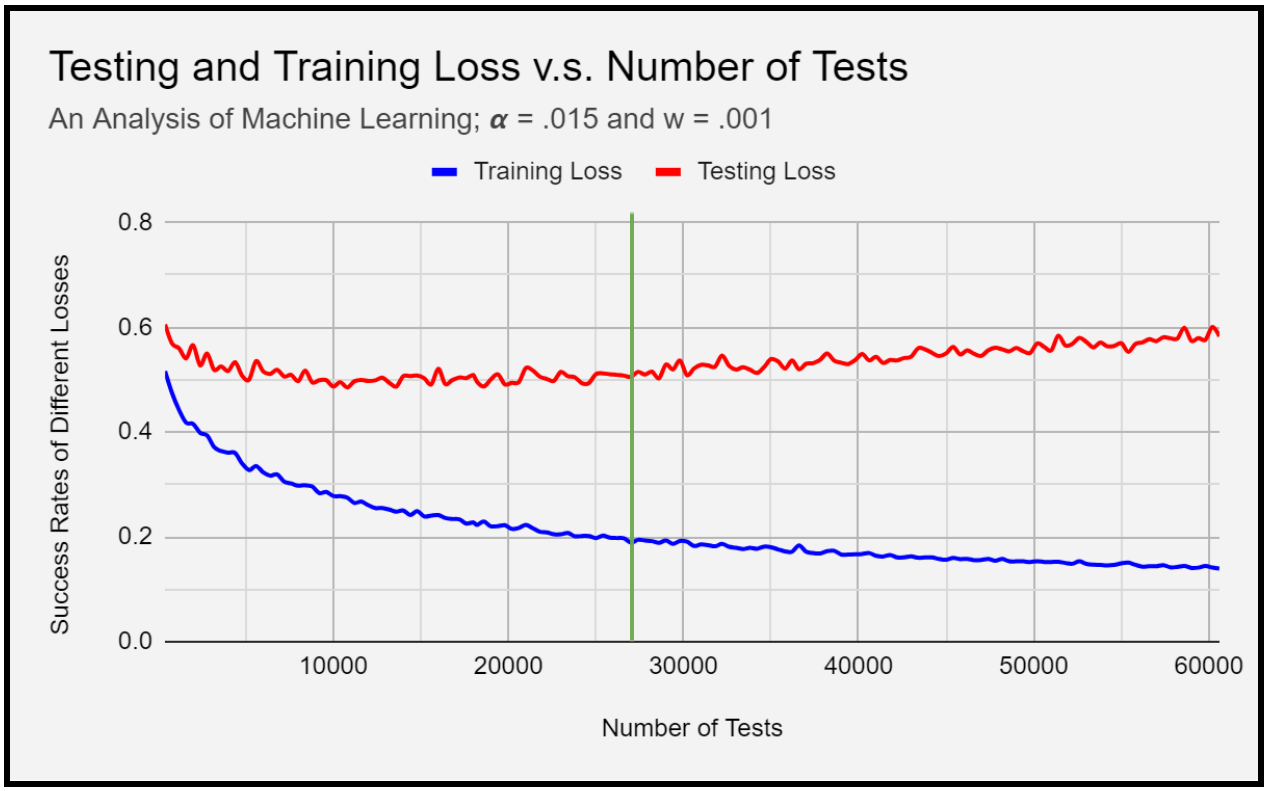


numTest = 2000; **numTrain** = 2000; **numTimesTrainAtOnce** = 2000

Initial Training Loss: 0.6841227364905225

Initial Testing Loss: 0.6850815434266431

trainingLossCutOff = 0.21

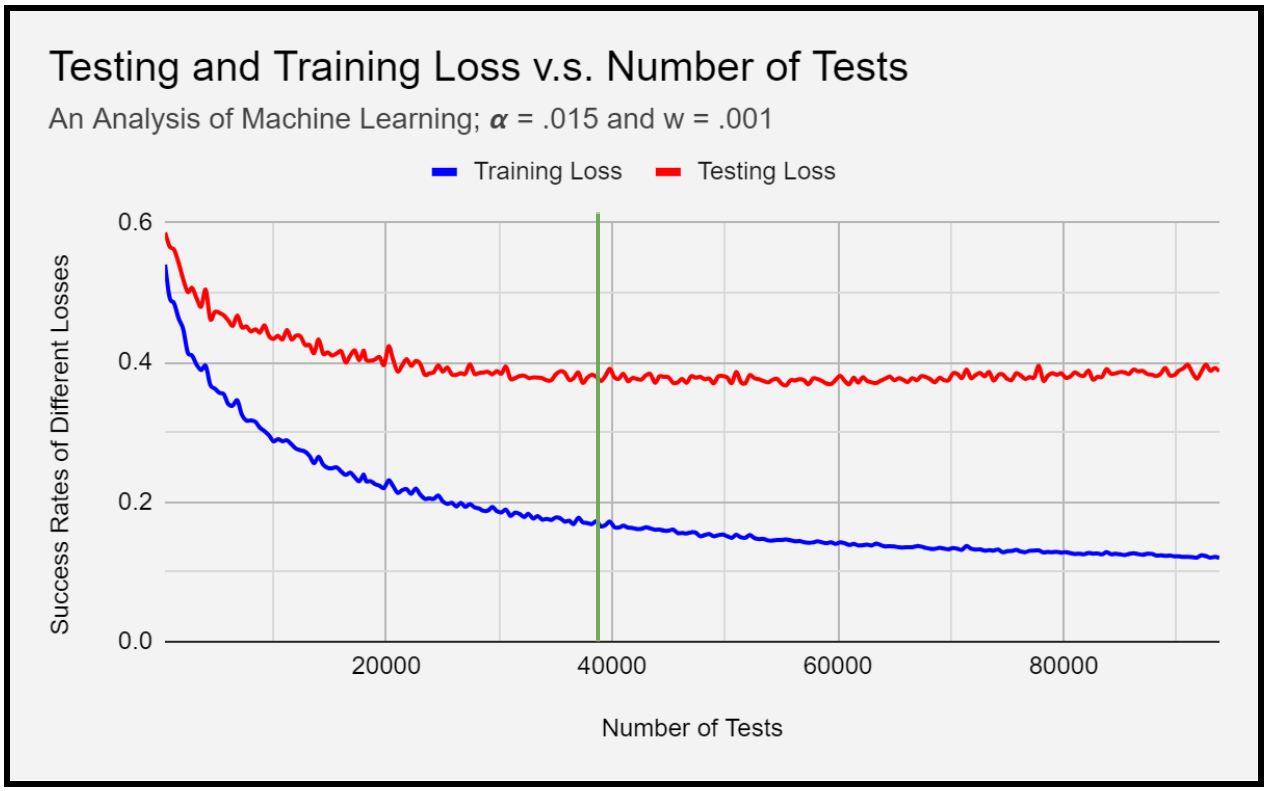


numTest = 2500; **numTrain** = 2500; **numTimesTrainAtOnce** = 4000

Initial Training Loss: 0.7021179979377715

Initial Testing Loss: 0.7049671578071757

trainingLossCutOff = 0.19

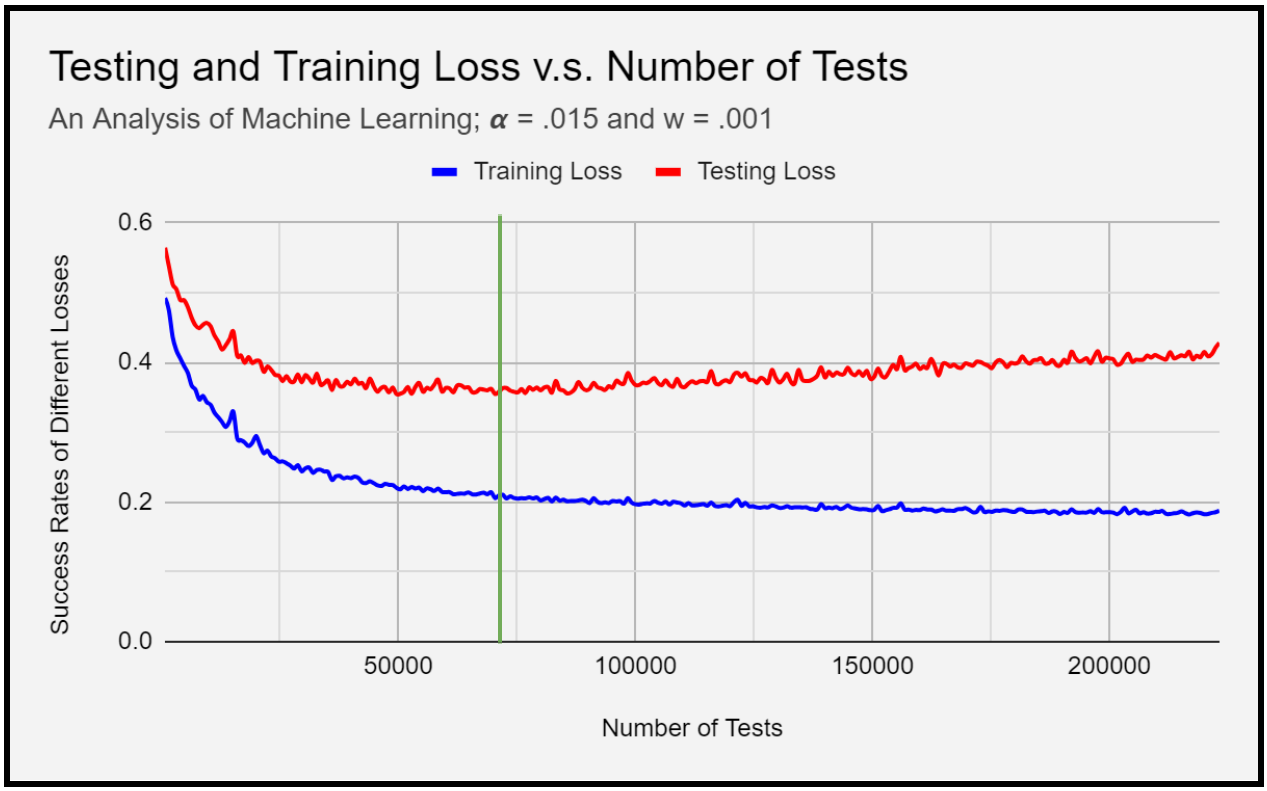


numTest = 3000; **numTrain** = 3000; **numTimesTrainAtOnce** = 4000

Initial Training Loss: 0.6934731110079427

Initial Testing Loss: 0.6943404806266043

trainingLossCutOff = 0.166



numTest = 5000; **numTrain** = 5000; **numTimesTrainAtOnce** = 8000

Initial Training Loss: 0.6975139809329063

Initial Testing Loss: 0.6973114282116145

trainingLossCutOff = 0.205

Analysis of Performance and Results

- For values of 2500 and 5000 for **numTest** and **numTrain**, the magnitude of the divergence is stronger than for values of 2000 and 3000. In all cases, training loss continuously decreases, constantly trying to improve itself, but at some point, the model begins to overfit the data, causing the divergence. Testing loss begins to increase, signifying the beginning of overfitting. Something to note is that the initial testing and training losses start extremely close together, but the training loss decreases much faster. The general trends of the graphs show training loss decreasing rapidly. After a certain point, the magnitude at which the loss decreases slows, but still decreases overall. In contrast, the testing loss decreases moderately quickly, hits a minimum, then begins to increase. The increase in the testing loss, which signifies overfitting, varies from graph to

graph. Vertical green lines on the graph signify when the code was terminated to prevent overfitting, however we let the data continue past this point to show how the model can overfit the data when given too many inputs.

The Second Task

Description

- Our code for the Second Task is very similar to the First with slight differences. The model created will determine which color wire to cut in a given dangerous wiring diagram. This is multiclass classification, with the four classes being Red, Green, Blue, and Yellow. The machine learning model will interpret this data and make an initial guess that will be updated over time, representing the probabilities of each color being the color to cut. This multiclass classification led us to use SoftMax Regression to build our machine learning model. For the purpose of understanding the intricate specifications of the model, it must be noted that the static variables used were the learning rate alpha, $\boldsymbol{\alpha}$, with value of $\boldsymbol{\alpha}$ = .015, and the initial weights, $\mathbf{w_R}$, $\mathbf{w_G}$, $\mathbf{w_B}$, and $\mathbf{w_Y}$, with value of 0.001. We use **numTrain, numTest, numTimesTrainAtOnce, trainingLossCutoff,** and **divergenceCutoff** in the same way as Task 1. To account for the four weights, we introduce four variables called **weightVector_R, weightVector_G, weightVector_B,** and **weightVector_Y** that will be updated as we train the model. We also have **softMaxDenominator** for storing the denominator calculated in the softmax calculation.
- Our images are generated by shuffling the order of "R", "G", "B", and "Y", ensuring this time that "R" comes before "Y" so the image is dangerous. We then iterate in that shuffled order to populate random rows/columns of the image. With a 50% chance, the program will either start by populating a row or by populating a column, as per the project specifications. It is then stored with a value indicating whether or not it is dangerous based on if "R" came before "Y" in the shuffled order—which in this case will always be dangerous. We also store the color to cut which is the 3rd wire laid down, stored as the 3rd color in the shuffled order.

Input Space

- The input space is a 20x20 wire diagram image of pixels, with 1-hot vectors and a dangerous label as in Task 1. However, this time each image is automatically generated to be "dangerous" and carries with it a string value in {"R", "G", "B", "Y"} representing

the color wire that must be cut. The color to be cut is always the 3rd color laid down in sequence.

Output Space

- The output space is a vector of size 4 of double values between 0 and 1, each indicating the probability that its corresponding color is the one to cut, i.e. $y = (p_R, p_G, p_B, p_Y)$ where y is the output vector and p_i is the probability that color i is the color of the wire that must be cut.

Model Space

- We use $\underline{w}_R, \underline{w}_G, \underline{w}_B, \underline{w}_Y$ for the weight vectors associated with each color, indicating the probability that color is the color of the wire that must be cut.
- Our model space uses Soft Max Regression for Multi Class Classification, where the model is defined to be the function:

$$F_{y^i}(\underline{x}^i) = \left(\frac{e^{(\underline{w}_R \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}, \frac{e^{(\underline{w}_G \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}, \frac{e^{(\underline{w}_B \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}, \frac{e^{(\underline{w}_Y \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}\right)$$

where i is the i^{th} image, $\underline{w}$ is the weight vector, $\underline{x}$ is the image vector with $x_0 = 1$, and there are C total classifications of colors. In this case, $C = 4$. This can be simplified to: $F(\underline{x}) = (F_R(\underline{x}), F_G(\underline{x}), F_B(\underline{x}), F_Y(\underline{x}))$.

Loss Function

- The Loss function is the Cross Entropy Loss derived in class:

$$Loss = \frac{1}{N}\sum_{i=1}^{N}[-\ \ln(F_{y^i}(\underline{x}^i))]$$

with $F_{y^i}(\underline{x}^i)$ defined above for a given i^{th} image in N images.

Training Algorithm

- We used Stochastic Gradient Descent for our training algorithm. When deriving softmax, there are two cases of derivatives—one when the actual output color is the same as the index we are looking at, and one when they are not the same.
- When deriving a component of $F(\underline{x})$ that corresponds to the current color a, we do:

$$\frac{\partial}{\partial w_a}(F_a(\underline{x})) = \frac{\partial}{\partial w_a}\left(\frac{e^{(\underline{w}_a \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}\right)$$

When taking the quotient rule, we only derive $e^{(\underline{w}_a \cdot \underline{x})}$ in the sum because it is w.r.t. a:

$$= \frac{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right) \cdot \frac{\partial}{\partial w_a}\left(e^{(\underline{w}_a \cdot \underline{x})}\right) - \left(e^{(\underline{w}_a \cdot \underline{x})}\right) \cdot \frac{\partial}{\partial w_a}\left(e^{(\underline{w}_a \cdot \underline{x})}\right)}{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right)^2}$$

$$= \frac{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right) \cdot \left(e^{(\underline{w}_a \cdot \underline{x})}\right)}{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right)^2} - \frac{\left(e^{(\underline{w}_a \cdot \underline{x})}\right) \cdot \left(e^{(\underline{w}_a \cdot \underline{x})}\right)}{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right)^2}$$

$$= \left(\frac{e^{(\underline{w}_a \cdot \underline{x})}}{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right)}\right) - \left(\frac{\left(e^{(\underline{w}_a \cdot \underline{x})}\right)}{\left(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}\right)}\right)^2$$

Substitute the softmax function $F_a(\underline{x})$:

$$= F_a(\underline{x}) - (F_a(\underline{x}))^2$$

$$= F_a(\underline{x})(1 - F_a(\underline{x}))$$

- When deriving a component $F_b(\underline{x})$ that does NOT correspond to the current color a:

$$\frac{\partial}{\partial w_a}(F_b(\underline{x})) = \frac{\partial}{\partial w_a}\left(\frac{e^{(\underline{w}_b \cdot \underline{x})}}{\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}}\right)$$

When taking the quotient rule, we only derive $e^{(\underline{w}_a \cdot \underline{x})}$ in the sum because it is w.r.t. a:

$$= \frac{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}) \cdot \frac{\partial}{\partial w_a}(e^{(\underline{w}_b \cdot \underline{x})}) - (e^{(\underline{w}_b \cdot \underline{x})}) \cdot \frac{\partial}{\partial w_a}(e^{(\underline{w}_a \cdot \underline{x})})}{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})})^2}$$

$$= \frac{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})}) \cdot (0) - (e^{(\underline{w}_b \cdot \underline{x})}) \cdot (e^{(\underline{w}_a \cdot \underline{x})})}{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})})^2}$$

$$= - \frac{e^{(\underline{w}_b \cdot \underline{x})} \cdot e^{(\underline{w}_a \cdot \underline{x})}}{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})})^2}$$

$$= - \left(\frac{e^{(\underline{w}_b \cdot \underline{x})}}{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})})}\right) \cdot \left(\frac{e^{(\underline{w}_a \cdot \underline{x})}}{(\sum_{k=1}^{C} e^{(\underline{w}_k \cdot \underline{x})})}\right)$$

Substitute the softmax functions $F_b(\underline{x})$ and $F_a(\underline{x})$:

$$= - F_b(\underline{x}) \cdot F_a(\underline{x})$$

- Therefore, we have the derivative to be $F_a(\underline{x})(1 - F_a(\underline{x}))$ when the color is the same as the term's color, and $- F_b(\underline{x}) \cdot F_a(\underline{x})$ when the color is different. We can compile this into a 4x4 matrix with colors R, G, B, Y where:

$$\frac{\partial F(\underline{x})}{\partial w} =$$

$$\begin{bmatrix} F_R(\underline{x})(1 - F_R(\underline{x})) & -F_R(\underline{x}) \cdot F_G(\underline{x}) & -F_R(\underline{x}) \cdot F_B(\underline{x}) & -F_R(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_G(\underline{x}) \cdot F_R(\underline{x}) & F_G(\underline{x})(1 - F_G(\underline{x})) & -F_G(\underline{x}) \cdot F_B(\underline{x}) & -F_G(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_B(\underline{x}) \cdot F_R(\underline{x}) & -F_B(\underline{x}) \cdot F_G(\underline{x}) & F_B(\underline{x})(1 - F_B(\underline{x})) & -F_B(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_Y(\underline{x}) \cdot F_R(\underline{x}) & -F_Y(\underline{x}) \cdot F_G(\underline{x}) & -F_Y(\underline{x}) \cdot F_B(\underline{x}) & F_Y(\underline{x})(1 - F_Y(\underline{x})) \end{bmatrix}$$

- When calculating the gradient of cross entropy loss, we do:

$$\frac{\partial Loss}{\partial F_{y^i}(\underline{x})} = [\frac{\partial}{\partial F_R(\underline{x})}(- \ln(F_R(\underline{x})), (\frac{\partial}{\partial F_G(\underline{x})}(- \ln(F_G(\underline{x})), \frac{\partial}{\partial F_B(\underline{x})}(- \ln(F_B(\underline{x})), \frac{\partial}{\partial F_Y(\underline{x})}(- \ln(F_Y(\underline{x}))]$$

$$= [- \frac{y^i_R}{F_R(\underline{x})}, - \frac{y^i_G}{F_G(\underline{x})}, - \frac{y^i_B}{F_B(\underline{x})}, - \frac{y^i_Y}{F_Y(\underline{x})}]$$

Where y_a^i is either 1 or 0 for a given color a depending on if that is the output color.

- Therefore the gradient is:

$$\frac{\partial Loss}{\partial \underline{w}} = \frac{\partial Loss}{\partial F_{y^i}(\underline{x})} \frac{\partial F_{y^i}(\underline{x})}{\partial \underline{w}} =$$

$$\begin{bmatrix} F_R(\underline{x})(1 - F_R(\underline{x})) & -F_R(\underline{x}) \cdot F_G(\underline{x}) & -F_R(\underline{x}) \cdot F_B(\underline{x}) & -F_R(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_G(\underline{x}) \cdot F_R(\underline{x}) & F_G(\underline{x})(1 - F_G(\underline{x})) & -F_G(\underline{x}) \cdot F_B(\underline{x}) & -F_G(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_B(\underline{x}) \cdot F_R(\underline{x}) & -F_B(\underline{x}) \cdot F_G(\underline{x}) & F_B(\underline{x})(1 - F_B(\underline{x})) & -F_B(\underline{x}) \cdot F_Y(\underline{x}) \\ -F_Y(\underline{x}) \cdot F_R(\underline{x}) & -F_Y(\underline{x}) \cdot F_G(\underline{x}) & -F_Y(\underline{x}) \cdot F_B(\underline{x}) & F_Y(\underline{x})(1 - F_Y(\underline{x})) \end{bmatrix}$$

$$\cdot$$

$$\left[-\frac{y_R^i}{F_R(\underline{x})}, \ -\frac{y_G^i}{F_G(\underline{x})}, \ -\frac{y_B^i}{F_B(\underline{x})}, \ -\frac{y_Y^i}{F_Y(\underline{x})} \right]$$

$$=$$

$$-\begin{bmatrix} y_R^i - F_R(\underline{x})\left(y_R^i + y_G^i + y_B^i + y_Y^i\right) \\ y_G^i - F_G(\underline{x})\left(y_R^i + y_G^i + y_B^i + y_Y^i\right) \\ y_B^i - F_B(\underline{x})\left(y_R^i + y_G^i + y_B^i + y_Y^i\right) \\ y_Y^i - F_Y(\underline{x})\left(y_R^i + y_G^i + y_B^i + y_Y^i\right) \end{bmatrix}$$

Because the y vectors are 1-hot vectors, the sum of the y's will be 1. This simplifies to:

$$F_{y^i}(\underline{x}^i) - y^i$$

Using this gradient, our SGD algorithm is as follows:

- Choose a learning rate $\boldsymbol{\alpha} > 0$.
- For each color $c \in \{R, G, B, Y\}$
  - Initialize the weights of the weight vector $\underline{w}_c$ to an initial guess, usually near $\underline{0}$.

    Make $\underline{w}_c(0)$ be the value 1.
  - Choose a random i^{th} image $(\underline{x}, y) \in Data$. Update the parameters ($\underline{w}_c$) for k iterations based on how poorly the model performs on image i:

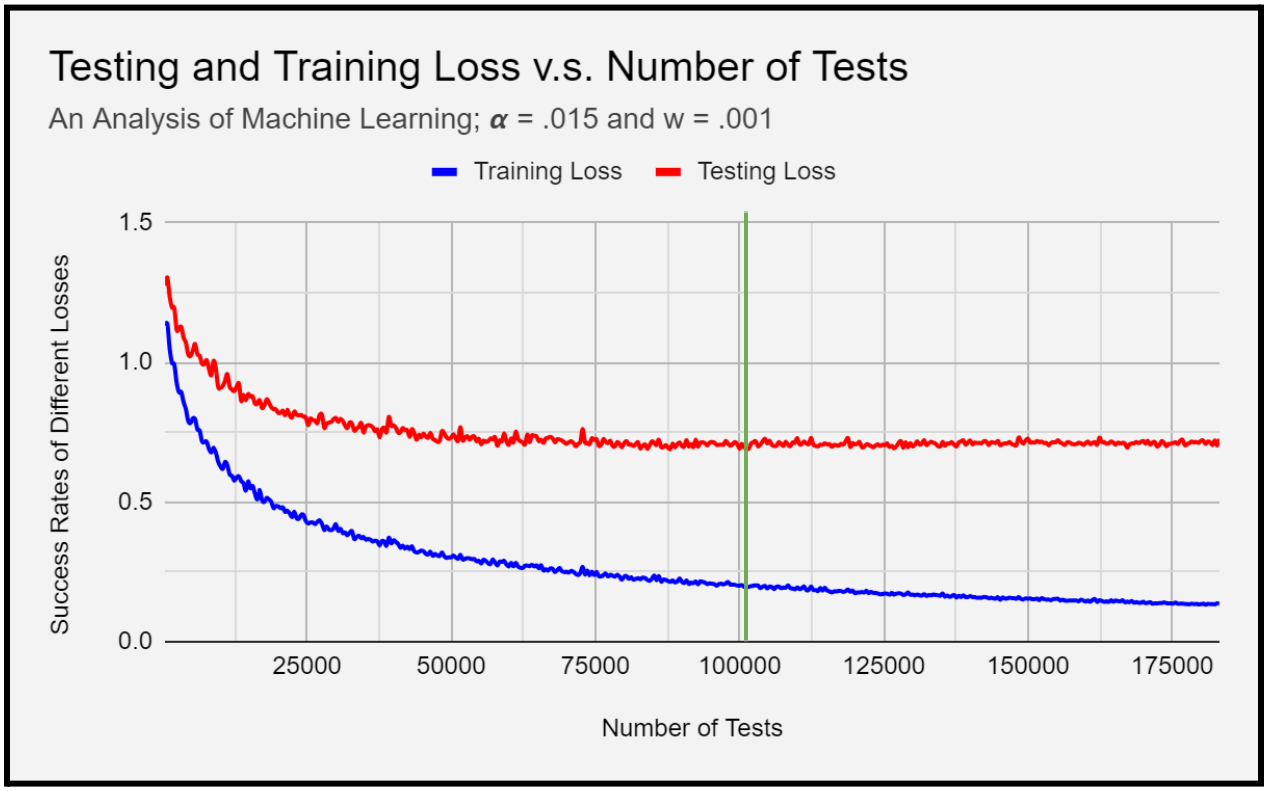
$$\underline{w}_c(k + 1) = \underline{w}_c(k) - \alpha[F_{y^i}(\underline{x}^i) - y^i]\underline{x}^i$$

- Repeat.

Overfitting

- We prevent overfitting the same way we did for Part 1. We train the model until the training loss and the testing loss seem to diverge. The point in which we terminate the program is shown by the vertical green line. Any data after that is just to display the hypothetical divergence that would occur if the model would continue to train.

Results (Graphs)

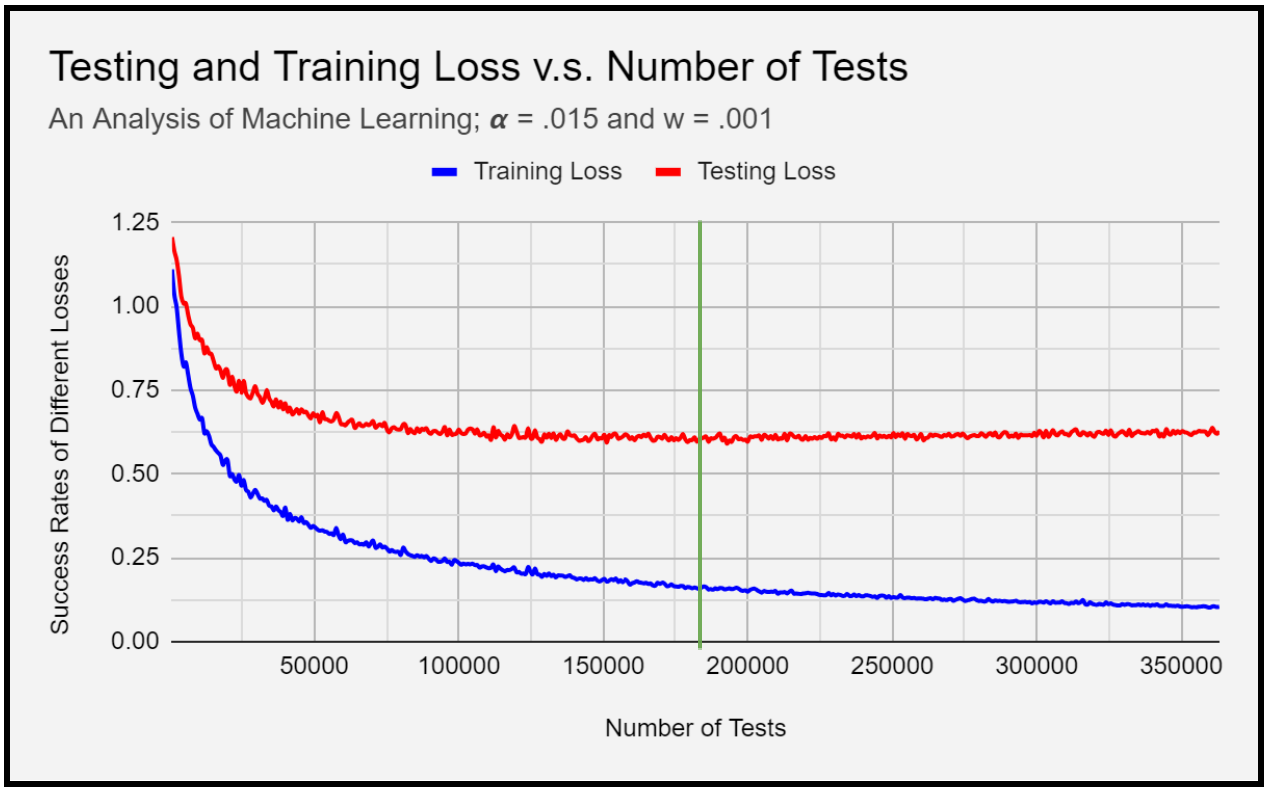


numTest = 5000; **numTrain** = 5000; **numTimesTrainAtOnce** = 4000

Initial Training Loss: 1.380041947015595

Initial Testing Loss: 1.3790159180597938

trainingLossCutOff = 0.2

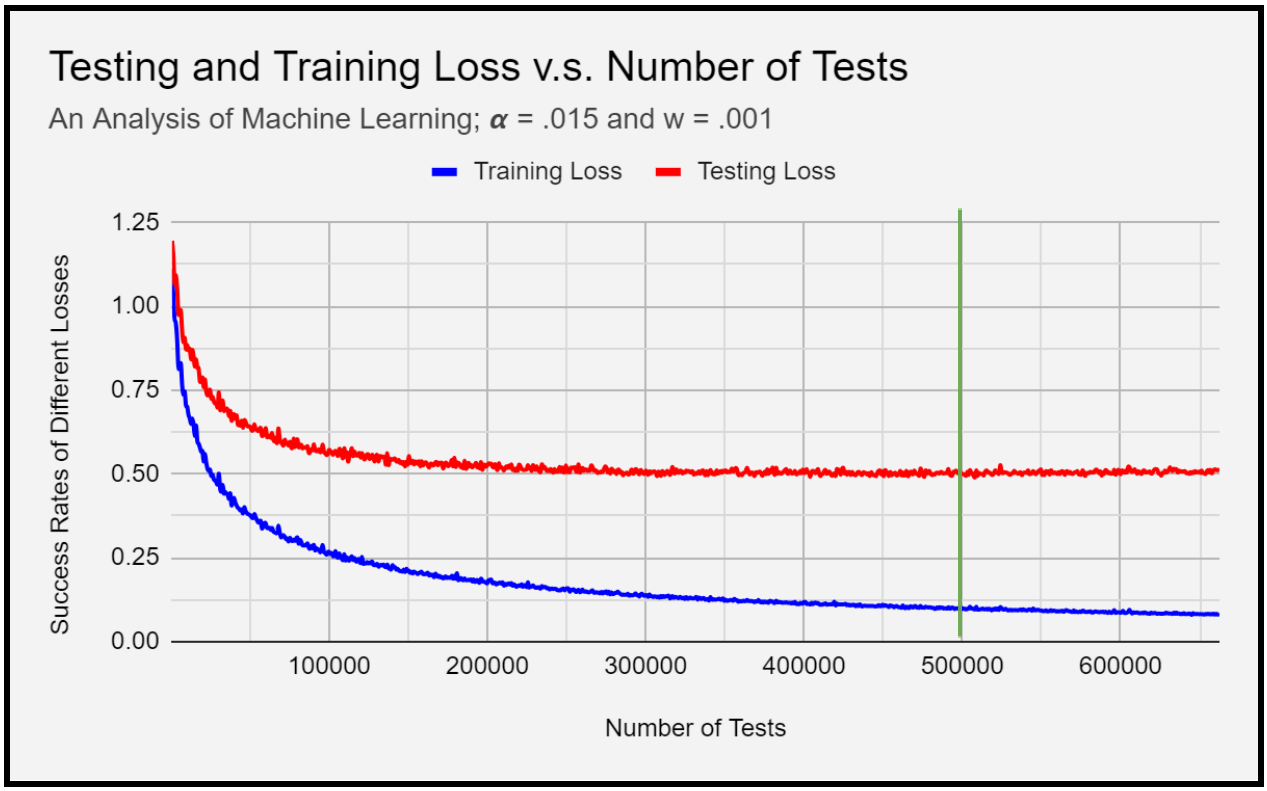


numTest = 6000; **numTrain** = 6000; **numTimesTrainAtOnce** = 8000

Initial Training Loss: 1.3728666165920642

Initial Testing Loss: 1.372157962166296

trainingLossCutOff = 0.16

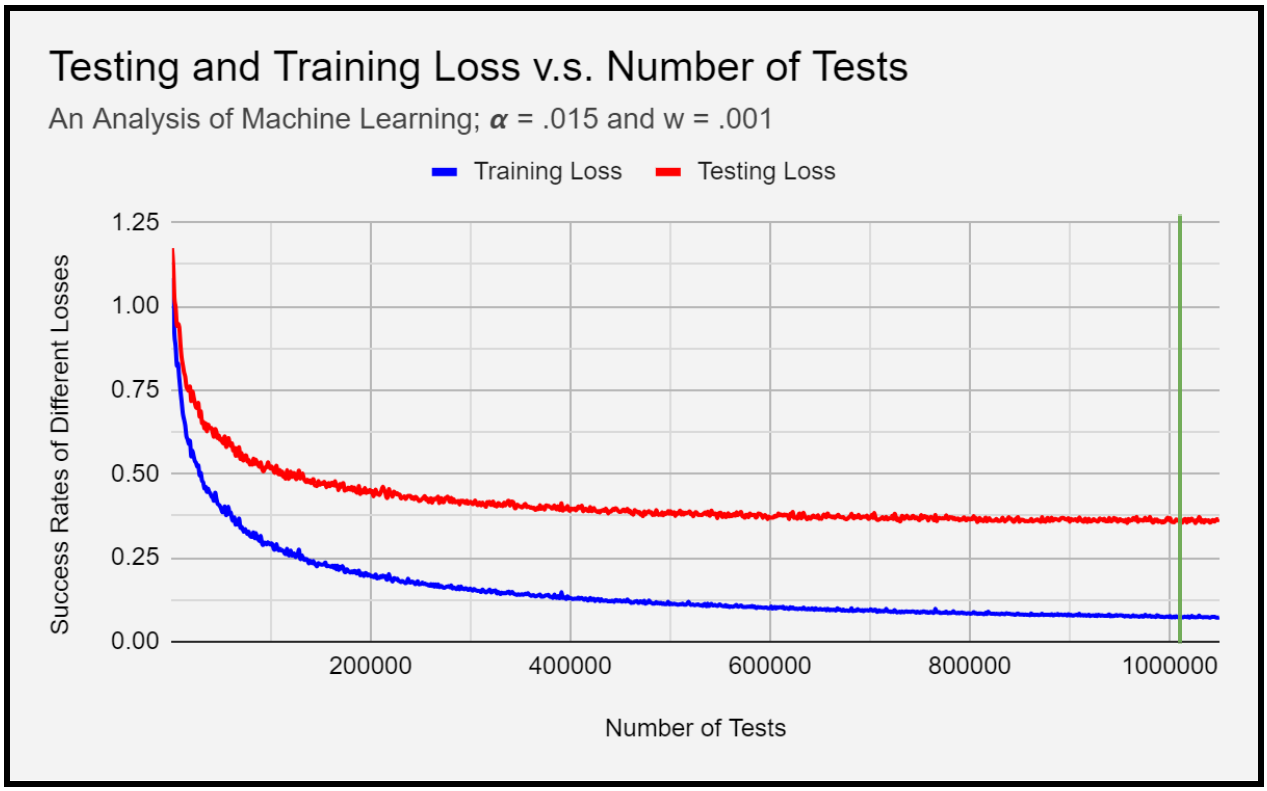


numTest = 7500; **numTrain** = 7500; **numTimesTrainAtOnce** = 8000

Initial Training Loss: 1.3761069867992717

Initial Testing Loss: 1.3772897044663985

trainingLossCutOff = 0.1



numTest = 10000; **numTrain** = 10000; **numTimesTrainAtOnce** = 12000

Initial Training Loss: 1.3796139763143904

Initial Testing Loss: 1.3801823571943643

trainingLossCutOff = 0.075

Analysis of Performance and Results

- For all graphs, the initial training and testing losses were always within a tenth or less of each other, however, it is easy to understand that both the training and testing loss decreases extremely fast, but that the testing loss begins to level out a lot sooner than the training loss. The final training loss for all graphs when the executing code was terminated varies, but they all follow a decreasing trend, approaching some minimum value greater than 0. Though, it would not be ideal to have the training loss get this low due to the fact that overfitting occurs after the training loss reaches a specific value, denoted by the **trainingLossCutOff.** Similarly, the testing loss also decreases before reaching a minimum. The testing loss then begins to increase slightly or generally oscillate between within a few tenths of decimal, plus or minus, of some value. Though it

may be difficult to tell when overfitting is occurring, the denotation of the vertical green line signifies when the training was stopped and overfitting began to occur. For some graphs, especially when **numTest** and **numTrain** have values of 6000 and 7500, a slight upward trend is noticeable, again, being indicative of overfitting occurring after the training was terminated. Again, vertical green lines on the graph signify when the code was terminated to prevent overfitting, however we let the data continue past this point to show how the model can overfit the data when given too many inputs.