**Perceptron learning**

Victor Wong, z5038168

Tianye Li, z5049945

Diwei Chen, z3494166

COMP9417, Assignment 2 Topic 3

**Introduction**

The perceptron learning algorithm is an example of supervised learning, which was invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt. This learning algorithm takes advantage of a feature vector and its target values to continually update a set of weights used to predict a target attribute. The perceptron can be implemented for both continuous and classification problems, with the main difference being the trasfer function that converts the model output into either a numerical prediction or a classification.

There are two parts in this project. The goal of the first part is to implement the perceptron for a numerical prediction for the autoMpg dataset. The problem will contain both numerical and classification values as inputs, so that will be one of the challenges for the perceptron. There are also missing values observed in the dataset, so those will have to be dealt with as well.

The second part of the project conists of testing the perceptron on different boolean functions with 8 variables, such that the input and output nodes for this problem will all be binary values. The main goal of the second part of the project is to determine the perceptron’s ability to accurately predict linearly separable and non-linearly separable functions, for that we will have to generate different truth tables of boolean functions for the perceptron to train on.

**Part I:**

**Data Description**

There are 8 attributes in the autoMpg dataset, which are cylinders, displacement, horsepower, weight, acceleration, model year, origin and miles per gallon(mpg) respectively. The class to be predicted is the mpg class, which is a continuous attribute. There are several discrete-valued attributes within the dataset, specifically the cylinder, model year and origin attributes. The dataset contains 398 instances, with 6 instances containing missing values for the horsepower attribute.

**Method**

We will first deal with the missing values within the autoMpg dataset. The missing values will cause problems for the perceptron, as each attribute contributes to the final prediction. There are some methods to deal with missing values, we can either ignore the instances with missing values entirely, or attempt to impute values for the missing data.

Imputing values for the missing horsepower attribute is a difficult problem, as simply imputing a fixed value may introduce bias to the dataset, while imputing a value based off of the distribution of the data, however, requires a large amount of work. Given that the entire dataset only contains 6 instances with missing data, and that the data appears to be missing at random, the loss in predictive power of the perceptron if we were to ignore these instances is minimal.

The second issue encountered within the dataset is the discrete valued inputs. For the cylinder node, it seems intuitively clear that the numerical value of the cylinders of a car will have a positive correlation to the mpg rating. As such, we will treat the cylinder attribute identically to the continuous valued inputs. The attributes model year and origin, however, do not follow the same pattern, as their numerical values hold no significance to their relation to the mpg rating. We will split the model year and origin attributes into x nodes each, with x representing the number of distinct values that the node can take. For example, the model year contains 13 possible values, so the perceptron will have 13 corresponding nodes for the model year attribute. When the model year has a value of “71”, for example, the node corresponding to the value 71 will be set to 1, while the other 12 nodes are set to 0. By splitting the discrete-valued attributes into these nodes, the perceptron will be able to assign different weights to each value within the attribute, and correctly represent the information provided by each value. For the autoMpg dataset, we will therefore have 5(continuous) + 13(model year) + 3(origin) + 1(intercept) = 22 input nodes.

The final issue within the dataset is that the input parameters have values that differ greatly in magnitude. For the perceptron, the weights will be updated in proportion to the magnitude of the inputs. If the data is not treated, the inputs with larger magnitude will dominate the perceptron’s learning simply because the weight updates for those nodes are much larger. The solution to this problem is to simply normalize all input nodes into a [0,1] threshold, we will implement this by applying min-max normalization to each column. For the prediction column, we will not apply normalization in order for the errors to be interpretted easily, this will not affect the performance of the perceptron.

Now that the dataset has been treated, the perceptron can commence training on the data. The perceptron initializes all the weights to 0, and proceeds to compute the output of the first instance.

The first step for computing the output is to sum the input node values multiplied by their corresponding weights. The sum is then passed through a transfer function, which in our case is simply the identity function, and the prediction for the mpg rating of the instance is obtained.

After obtaining the prediction, we will subtract it from the actual value of the mpg rating, this is the local error of the instance. The weights of the perceptron are now updated by adding the local error multiplied by the input value and the learning rate. With the weights updated, the perceptron will then move to the next training instance. Once all unique instances have been trained, the perceptron will have completed one epoch, and the iteration error will be computed. The iteration error is computed by simply summing the absolute value of all the local errors of the epoch, and then dividing the sum by the number of instances. As the number of epochs increase, the perceptron should improve by lowering the the iteration error. The perceptron converges either when the iteration error falls below a certain threshold, or after training for a certain number of epochs.

**Results**

For our testing, we firstly train the perceptron with a learning rate of 0.5 for 600 epochs. Figure 1.1.1 shows that the weights are oscillating as the perceptron algorithm learns. This suggests that the learning rate is too high, resulting in the weights not being able to reach their optimal value. This is further suggested by the relatively high value of the iteration error, as observed in Figure 1.1.2.

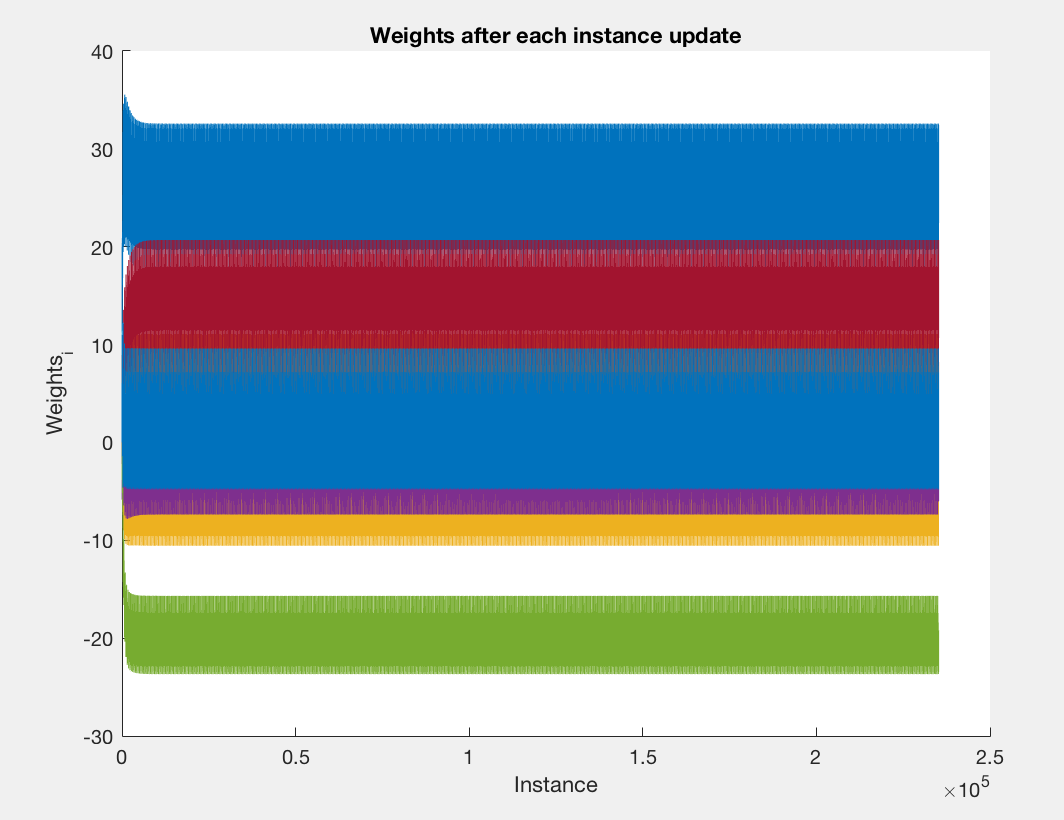


Figure 1.1.1

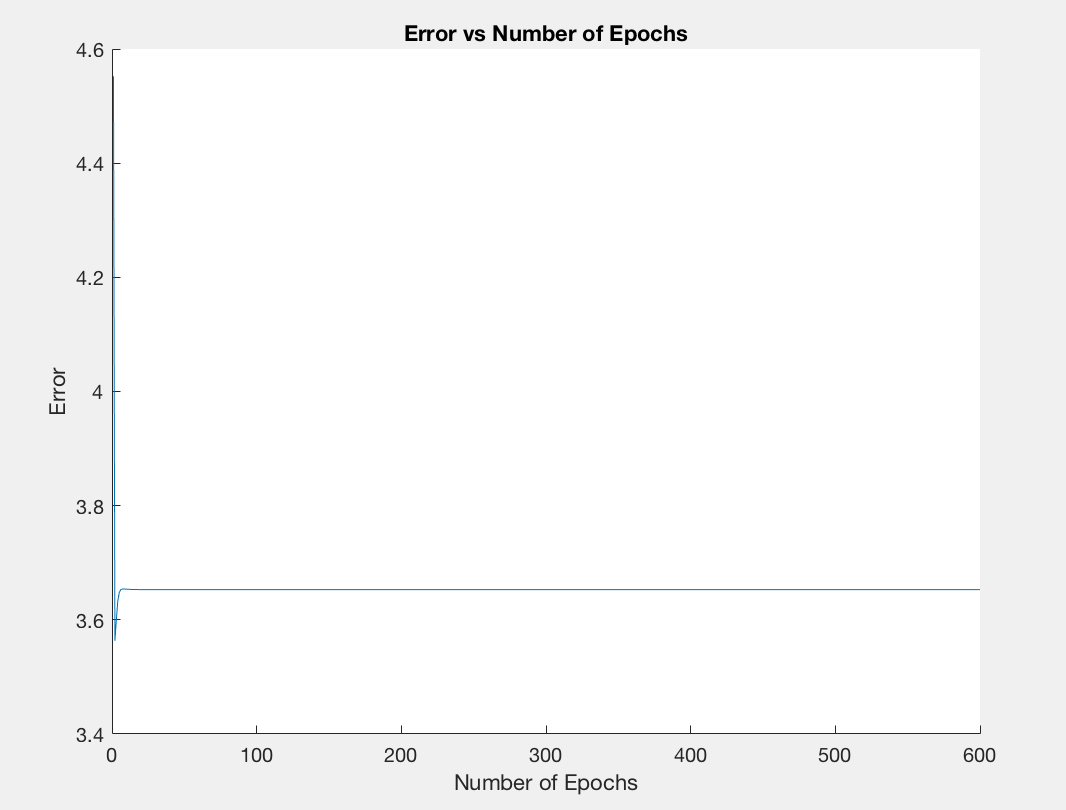


Figure 1.1.2

Our second test trains the perceptron for 600 epochs at a learning rate of 0.01. Figure 1.2.1 shows that the weights are able to converge relatively quickly at around 50,000 training instances, or 100 epochs. The weights can be seen to still oscillate around the converged values, although at a much smaller range than Figure 1.1.1. Figure 1.2.2 provides further evidence that the algorithm converges at around 100 epochs, since the iteration error reaches a lull at that point as well.

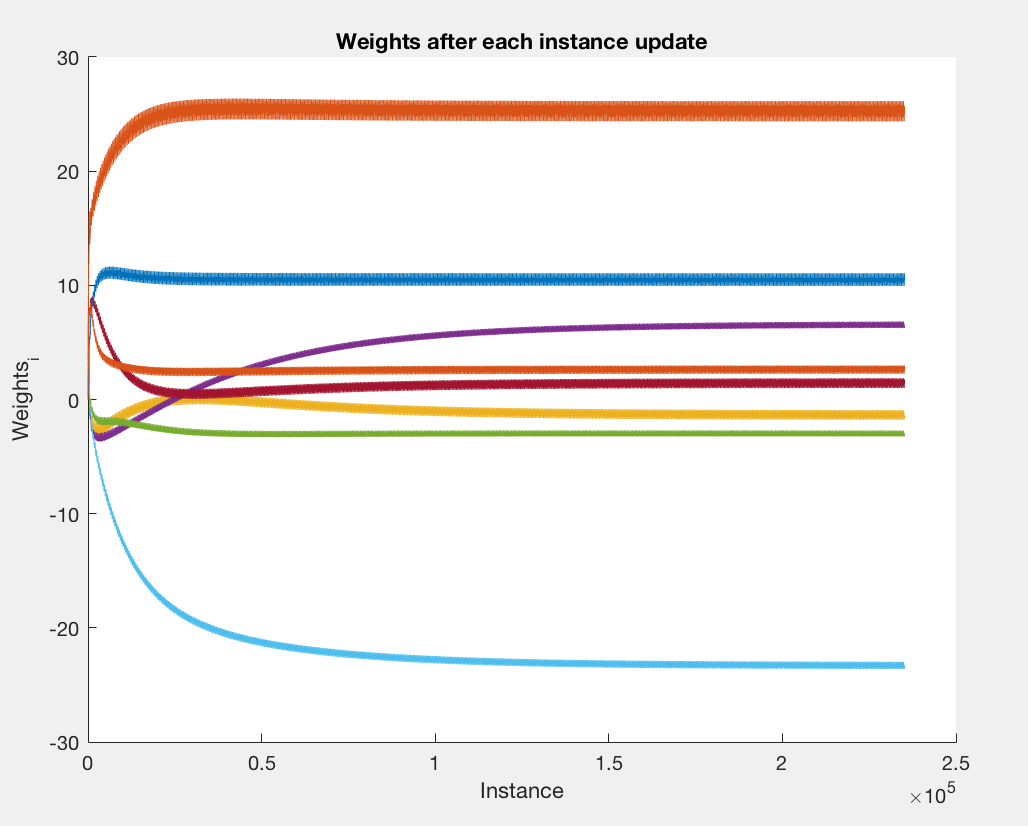


Figure 1.2.1

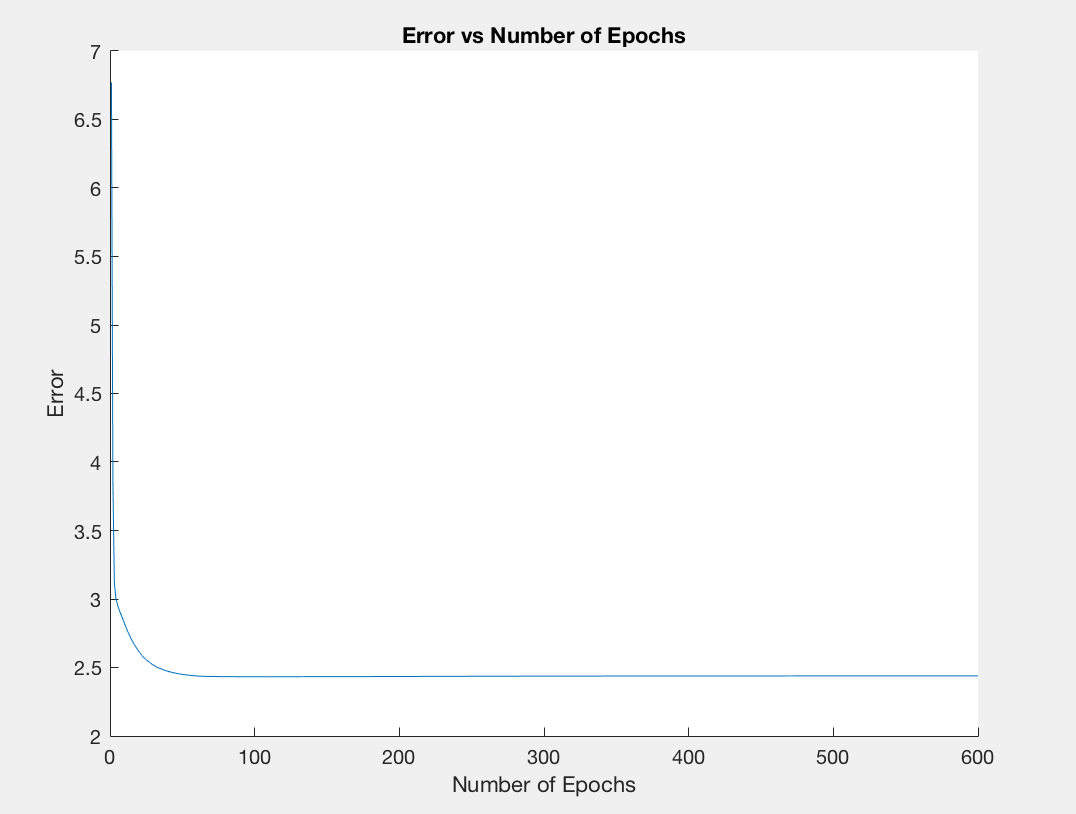


Figure 1.2.2

Our final test trains the perceptron for 600 epochs with a learning rate of 0.001. Figure 1.3.1 shows a much slower convergence for the weights, with several weights still being updated after 392\*600=235,200 iterations. The affect of the decreased learning rate can be seen quite clearly here, as the time taken to reach convergence is clearly slower. The effects of a lower learning rate can also be seen in Figure 1.3.2, where the iteration error decreases at a slower rate. The advantages to having a slower learning rate can also be seen in both figures. The oscilliations for the weights have been significantly reduced, and the iteration error is also lower as the perceptron is able to generate more accurate results.

For this problem, it appears that a learning rate of around 0.01 is ideal, as higher values may cause the algorithm to have convergence issues, while lower numbers are much slower and only provide marginally better predictions.

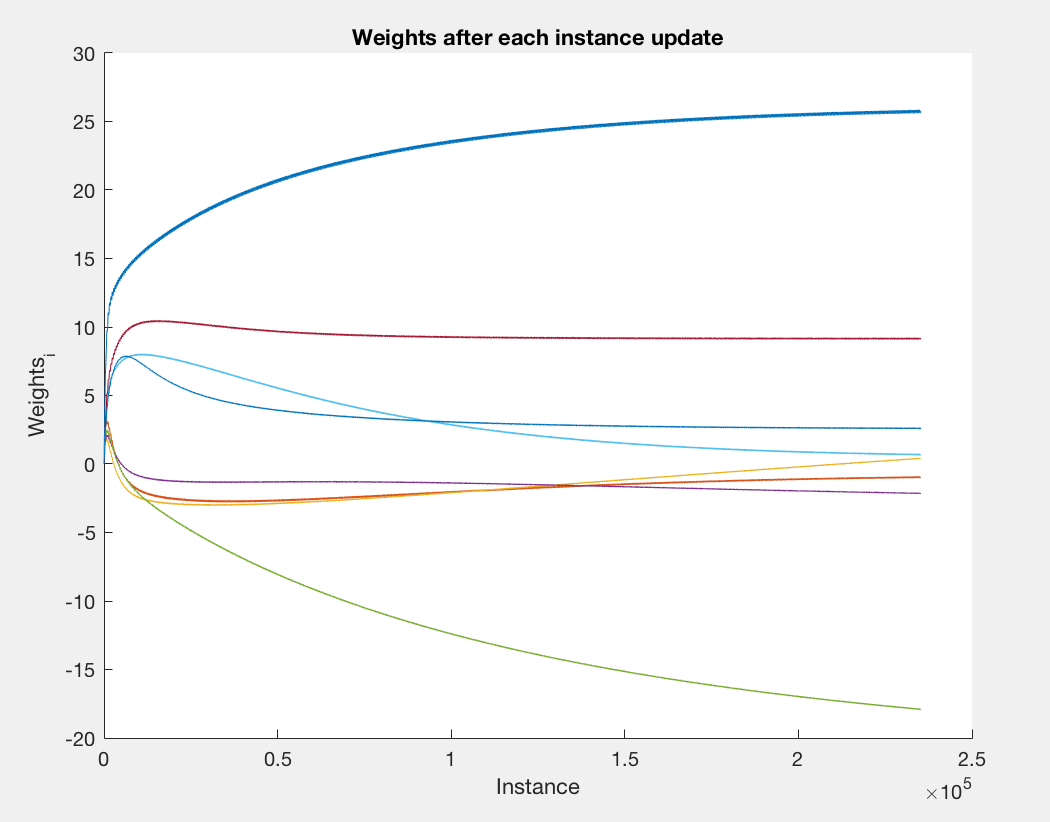


Figure 1.3.1

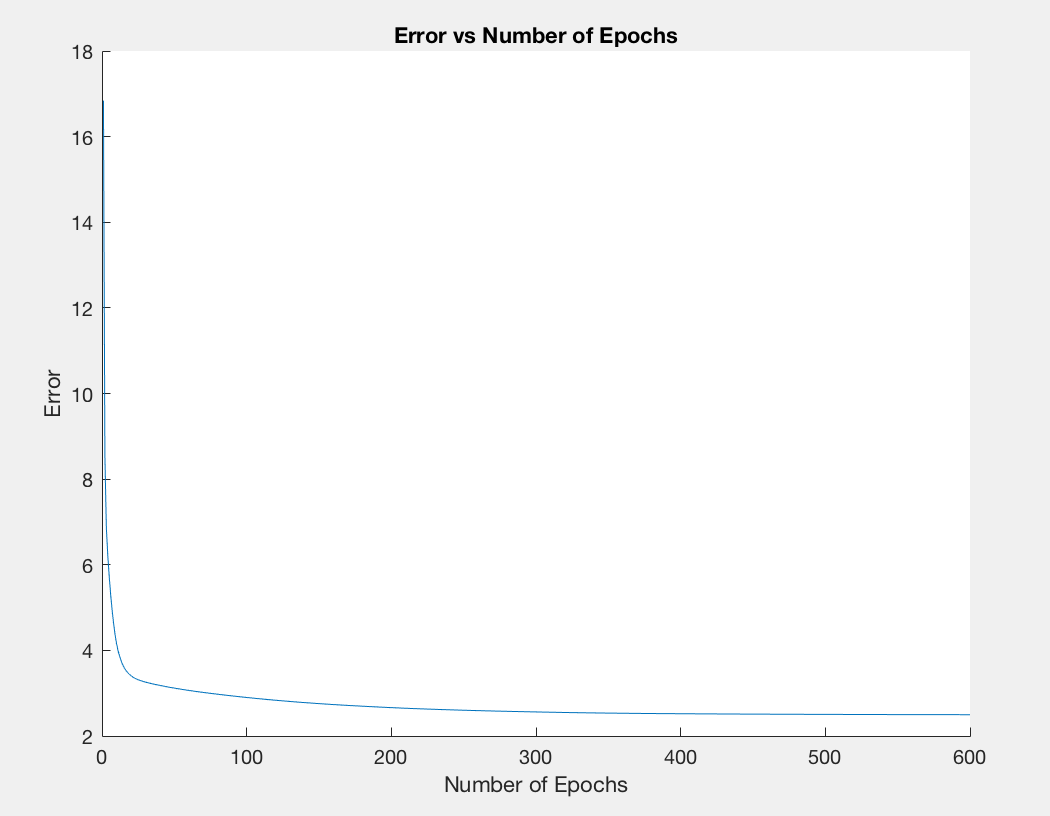


Figure 1.3.2

**Part II:**

**Data Description**

The data used will include 8 input Boolean attributes(x1, x2, …, x8) and 1 target Boolean variable x9. The datasets used for training will contain the entire truth tables for certain boolean functions.

**Method**

The problem here is much better defined in comparison to the autoMpg dataset. All of the inputs/outputs are binary 0/1 values, so there is no need to deal with issues of missing data and normalization. The only difference between the perceptron training method here compared to the perceptron in part I is the transfer function for the output. The transfer function used for predicting a binary classifier is typically a step function. Since this particular problem deals with a binary valued output, we simply use a one-step function with the threshold set at 0.5.

**Results**

Our first test is for the simple boolean function (x1 AND -x2 AND x3 AND -x4 AND x5 AND -x6 AND x7 AND -x8), which only returns true for one set of parameters. Figure 2.1.2 shows that the perceptron is able to correctly estimate all cases of the function. Figure 2.1.1 shows the final weights of the nodes, which are (0.1, 0.1, -0.05, 0.1, -0.05, 0.1, -0.05, 0.1, -0.05), with w0 as the intercept weight.

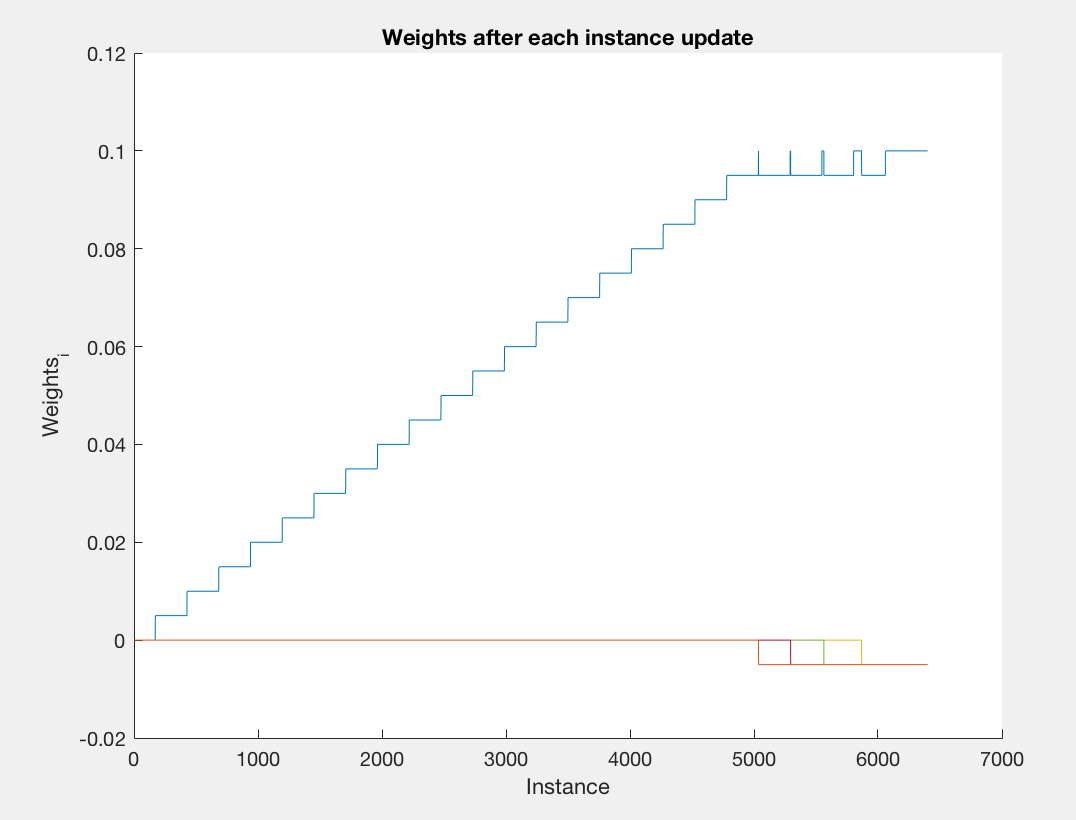


Figure 2.1.1

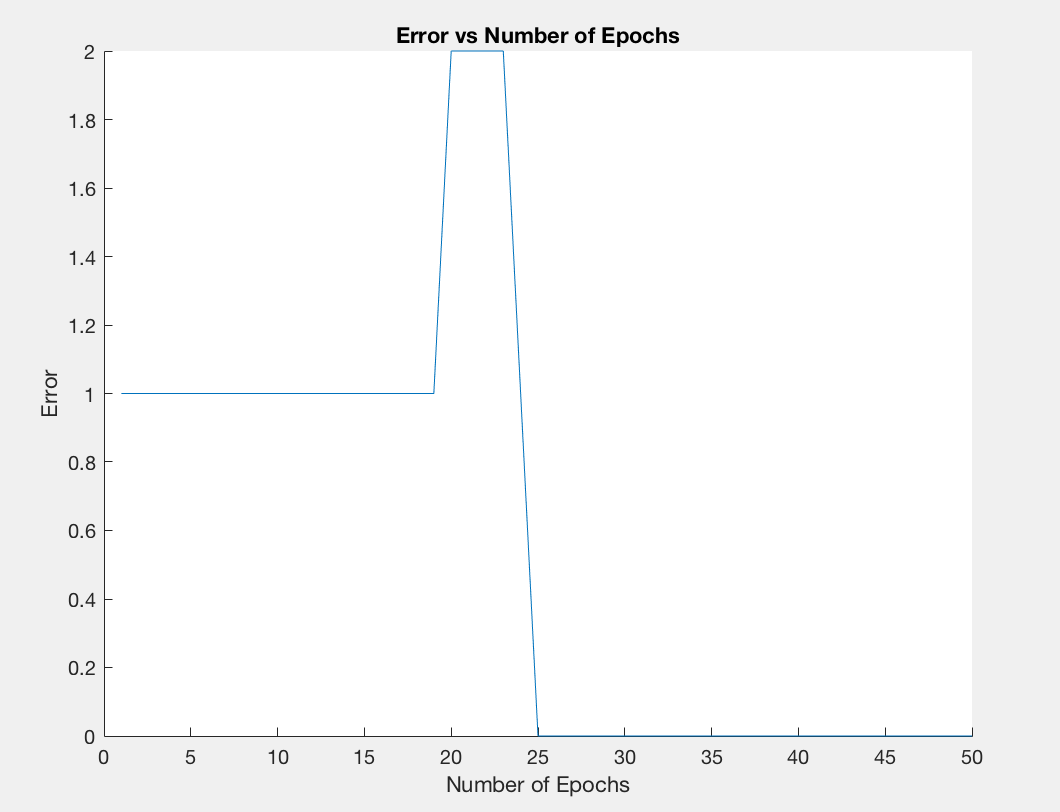


Figure 2.1.2

The second test is also a linearly separable function, which is simply the boolean function (x1). This boolean function has exactly half the inputs resulting in true statements. The final results are (0.2300 0.2700 0.0250 0.0250 0.0450 0.0350 0.0400 0.0450 0.0450), which only returns true if x1 is also true.

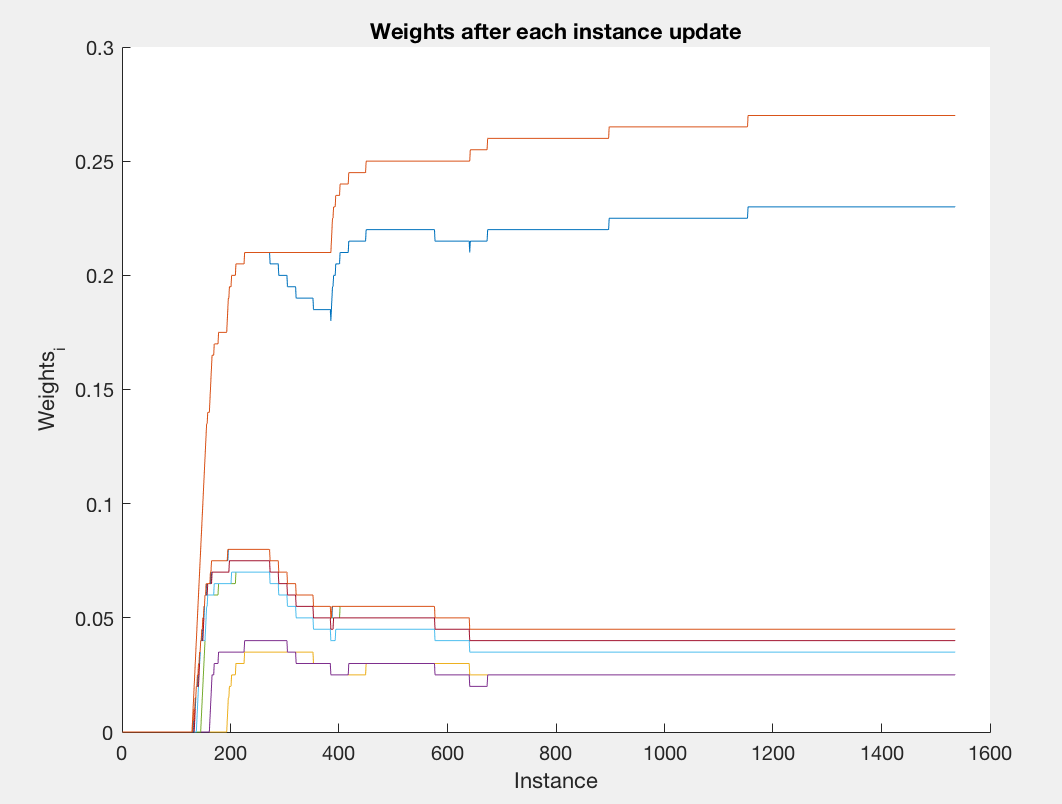


Figure 2.2.1

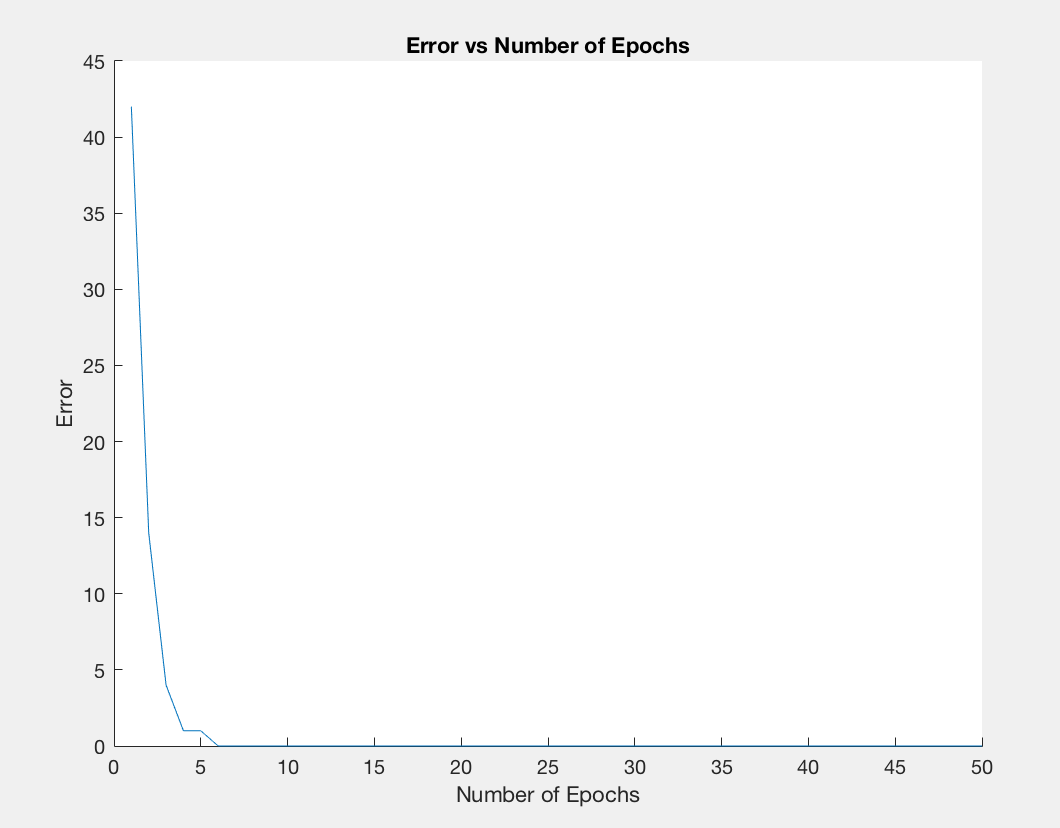


Figure 2.2.2

The last test for our perceptron is a non-linearly separable boolean function, which corresponds to the boolean function ((x1 AND x2) OR (-x1 AND -x2)). As we can see from Figure 2.3.2, the perceptron does not converge to a solution that can adequately explain the boolean function.

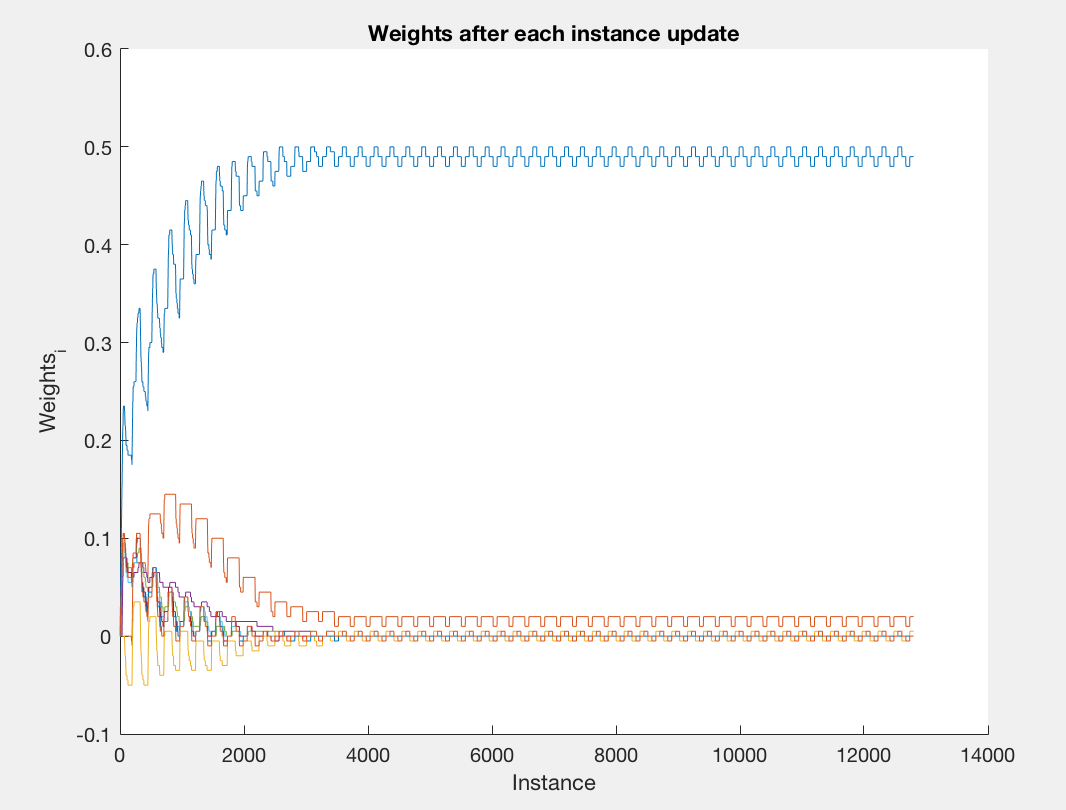


Figure 2.3.1

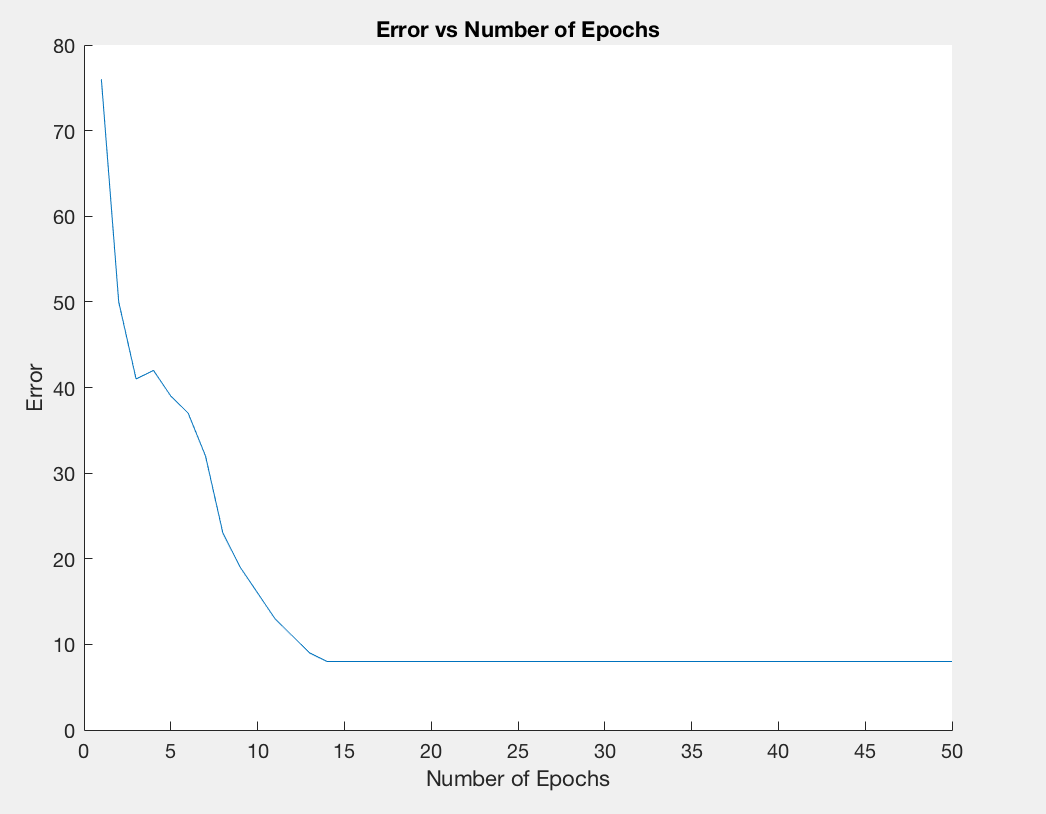


Figure 2.3.2

**Conclusion**

The perceptron as a learning algorithm has shown to be a great tool for learning linearly separable problems. From the training of the perceptron for the autoMpg dataset, we can see that perceptrons work adequately well for problems that are not neccessarily linearly separable either, as the predictions are only roughly 10% off overall.