DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Project No. 02:

PROJECT 2: K-NN AND CONDENSED K-NN

submitted to:

Professor Richard Souvenir
CIS 4526/55256: Fundamentals of Machine Learning
Temple University
College of Science and Technology
1947 North 12th Street
Philadelphia, Pennsylvania 19122

July 23, 18

prepared by:

James Novino
Email: tue73681@temple.edu

1. ABSTRACT

2. ALGORITHM

There were two different variations of the perceptron learning algorithm implemented in this project the first was a standard binary classifier version of the PLA algorithm. The algorithm uses the following steps to update the weight vector w.

1. Generate Linearly Separable Data [X,Y]
2. Calculate the response $h(x) = sign(w^T X)$
3. Calculate the number of errors $h_n \neq Y_n$
4. Pick a misclassified point i and update the weight vector $w \leftarrow w + Y_n X_n$ repeat steps 2-4 until errors=0

The PLA algorithm described above updates the weight vector only when there are errors if h = Y then PLA is complete and w represents the decision boundary. The second algorithm that was implemented in this lab was the regression PLA algorithm known as the "pseudoinverse", the steps for the pseudoinverse is very similar, except instead of using an uninitialized weight vector the pseudo inverse calculates an initial weight vector.

1. Generate Linearly Separable Data [X, Y]
2. Calculate pseudo inverse $X^\dagger = (X^T X)^{-1} X^T$
3. Calculate weight vector $w = X^\dagger y$
4. Calculate response $h(x) = sign(w^T X)$
5. Calculate the number of errors $h_n \neq Y_n$
6. Pick a misclassified point and update the weight vector $w \leftarrow w + Y_n X_n$ repeat steps 4-6 until errors=0

3. PREFORMANCE

The PLA algorithm preforms the steps described in the section above. Using a function generate_data a set of N sized data is generated and guaranteed to be linearly separable data by creating a decision boundary from two randomly selected points and then uses that decision boundary to classify each points as either +1 or -1. Then using these generated points, the PLA algorithm is run, the plots below show the generated points decision boundary is shown on the right and the PLA decision boundary is shown on the right.

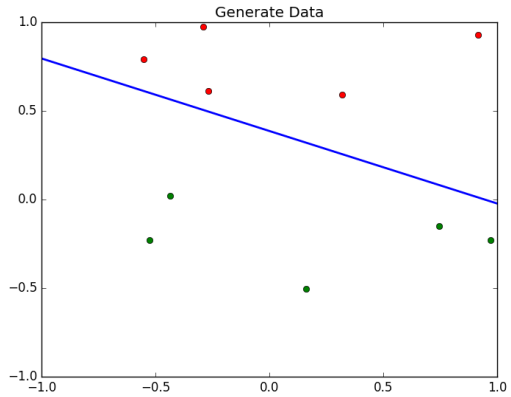

Figure 1:Generate Data N=10

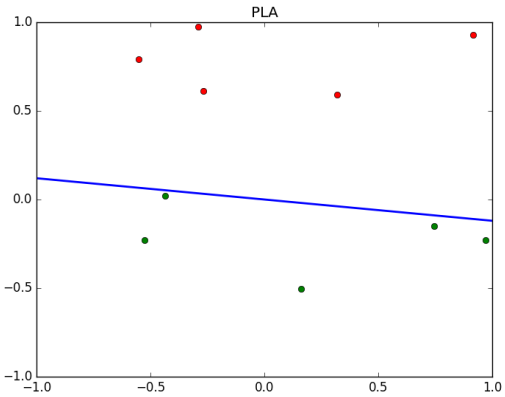

Figure 2: PLA N=10

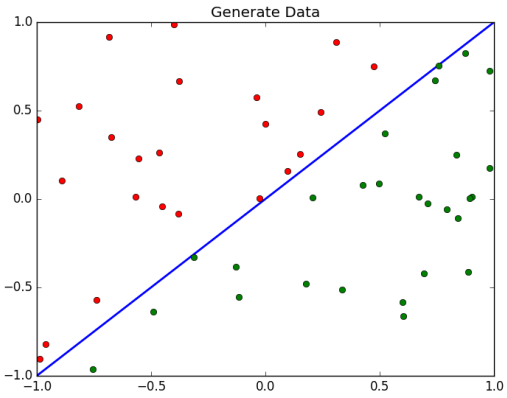

Figure 3: Generate Data N=50

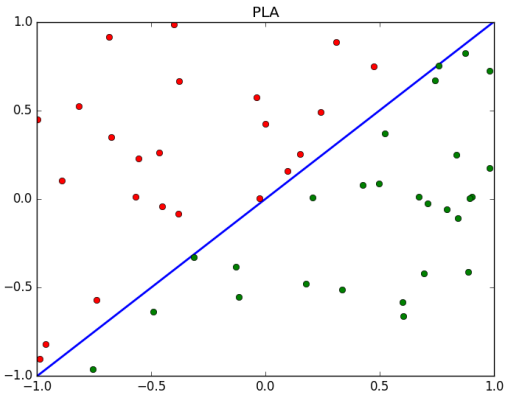

Figure 4: PLA N=50

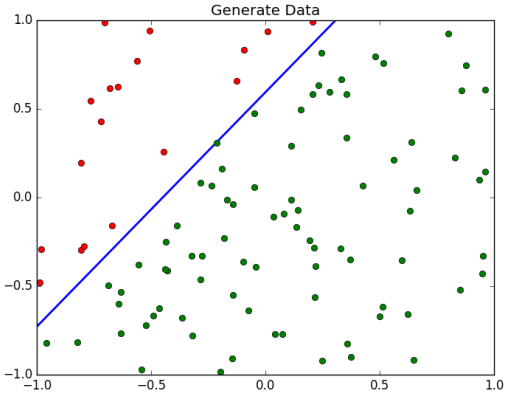

Figure 5: Generate Data N=100

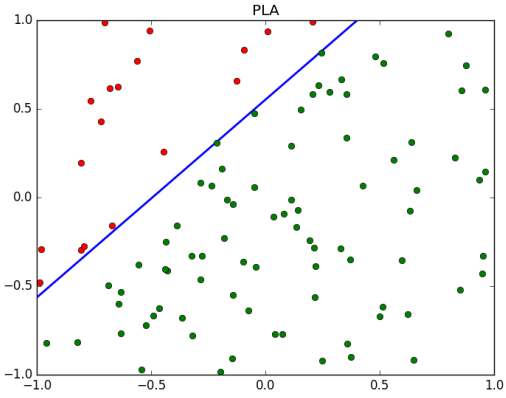

Figure 6: PLA N=100

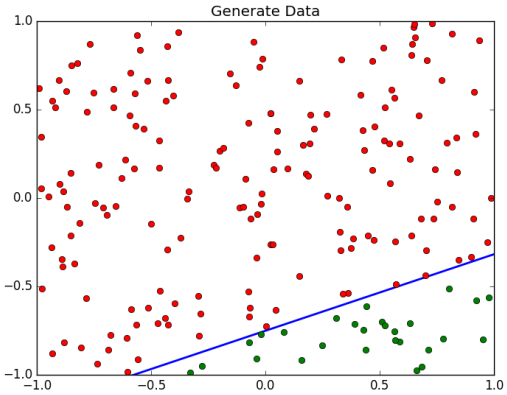

Figure 7: Generate Data N=200

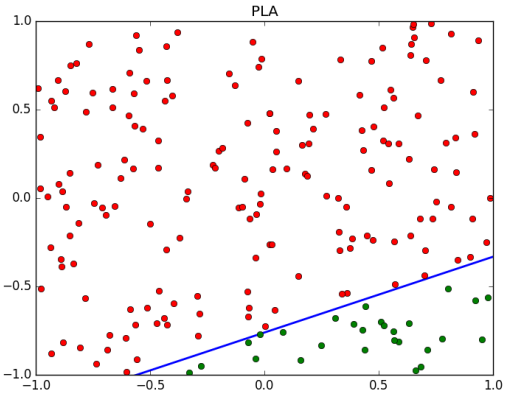

Figure 8: PLA N=200

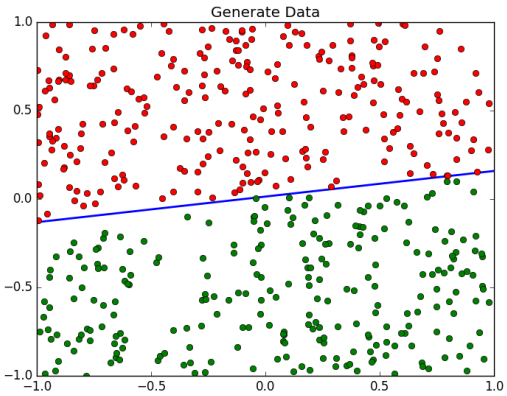

Figure 9: Generate Data N=500

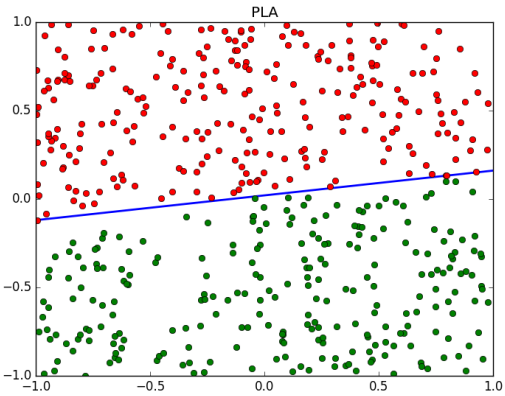

Figure 10: PLA N=500

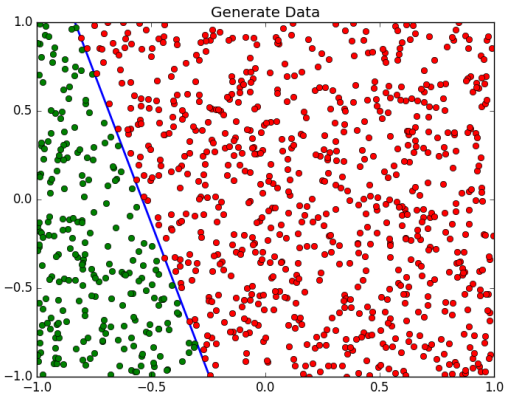

Figure 11: Generate Data N=1000

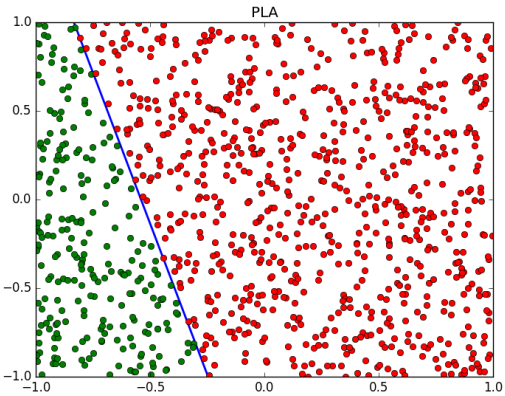

Figure 12: PLA N=1000

As the figures above show the decision boundaries between the generated points and the PLA points differ slightly but still classify the points correctly. The number of iterations that are required for PLA to converge are shown below. The values shown below are average over 100 different iterations to reduce the amount

of deviation attributed to the random generation and random selection process for updating the weights.

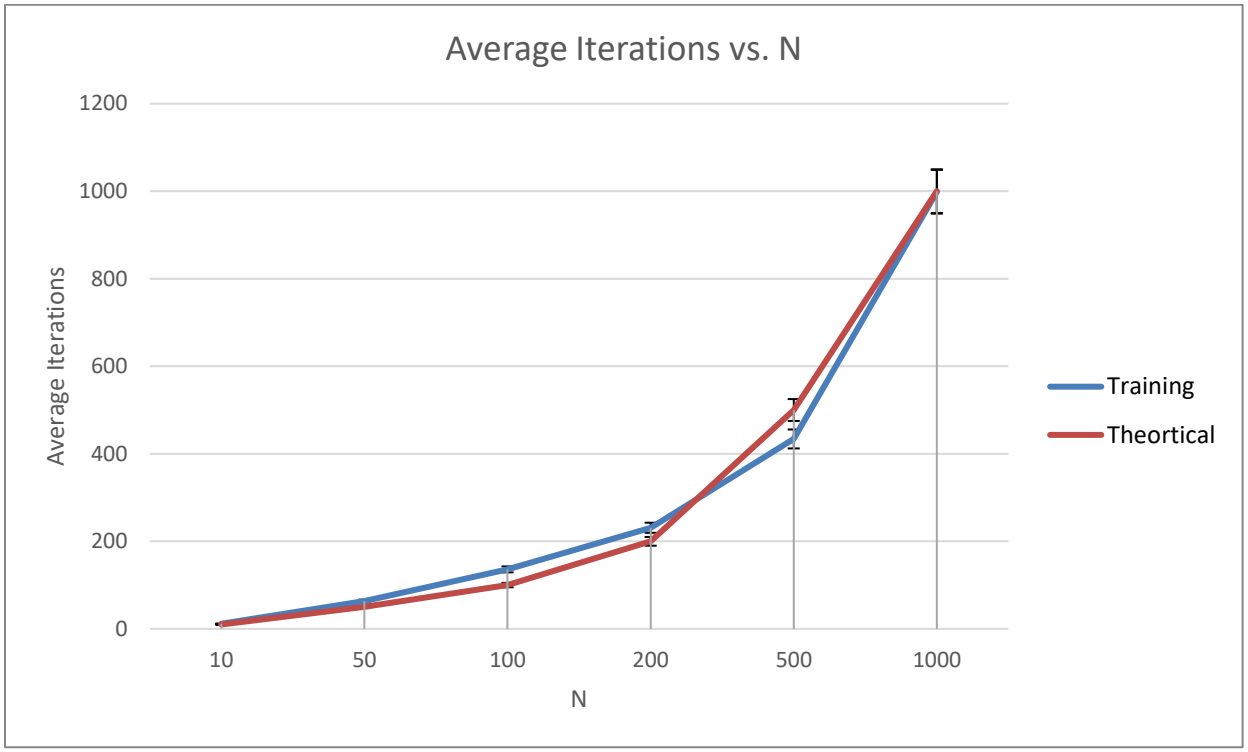

Figure 13

As shown above the number of iterations required are roughly around N, which is also plotted in red above. However, there is still some errors between N and the number of iterations required so in order to further verify this hypothesis the number of times the results where averaged over was increased to 1000.

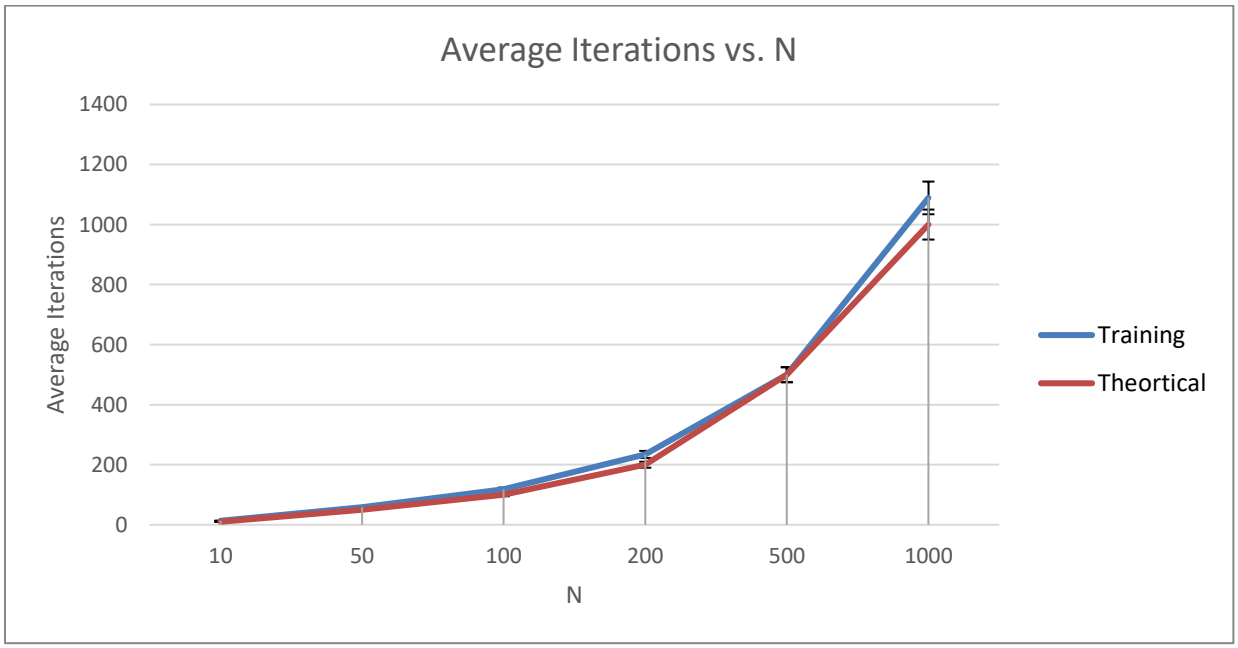

Figure 14

As the graph above the average number of iterations does trend towards N. There were some minor deviations with the large values of 200, 1000 which would be further reduced by averaging over a large number of runs. The execution performance was also interesting, similar to the number of iterations required to converge, the execution time of the PLA algorithm was also plotted.

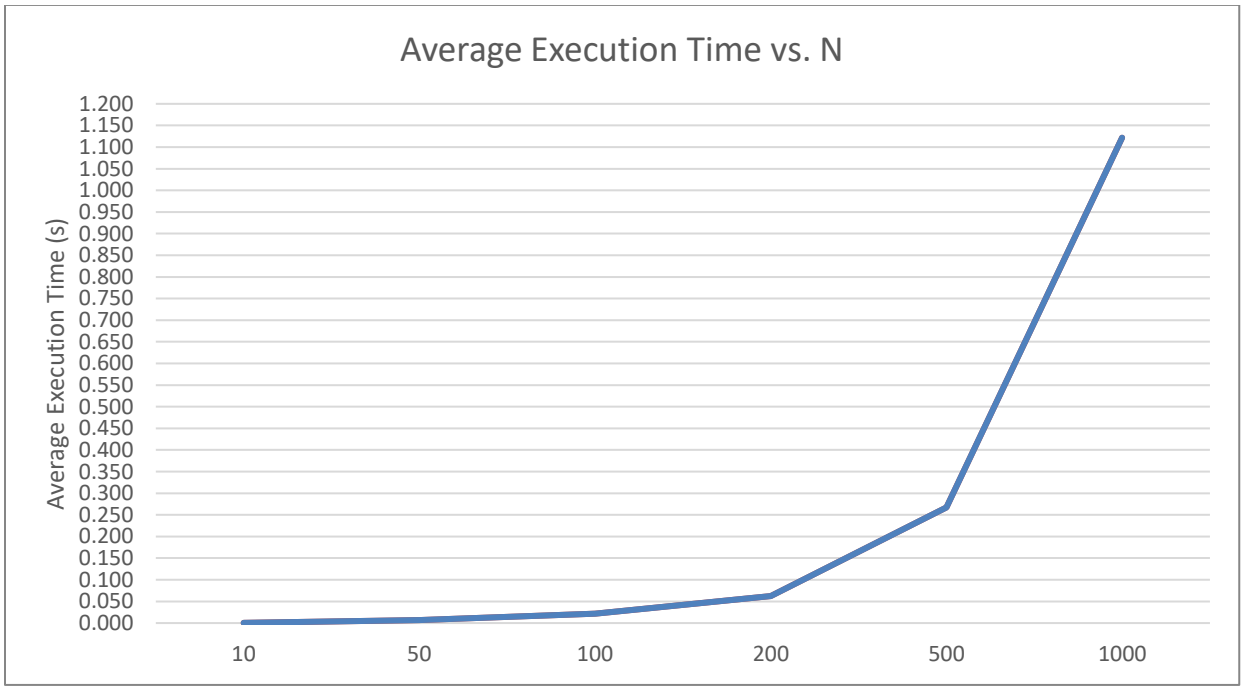

Figure 15

The average execution time increases exponentially with increases in N. The calculated big-O for the algorithm was $O(n^2)$. The initialized version of PLA or "pseudo inverse" had a very similar performance for interactions the chart below shows the iterations vs. N.

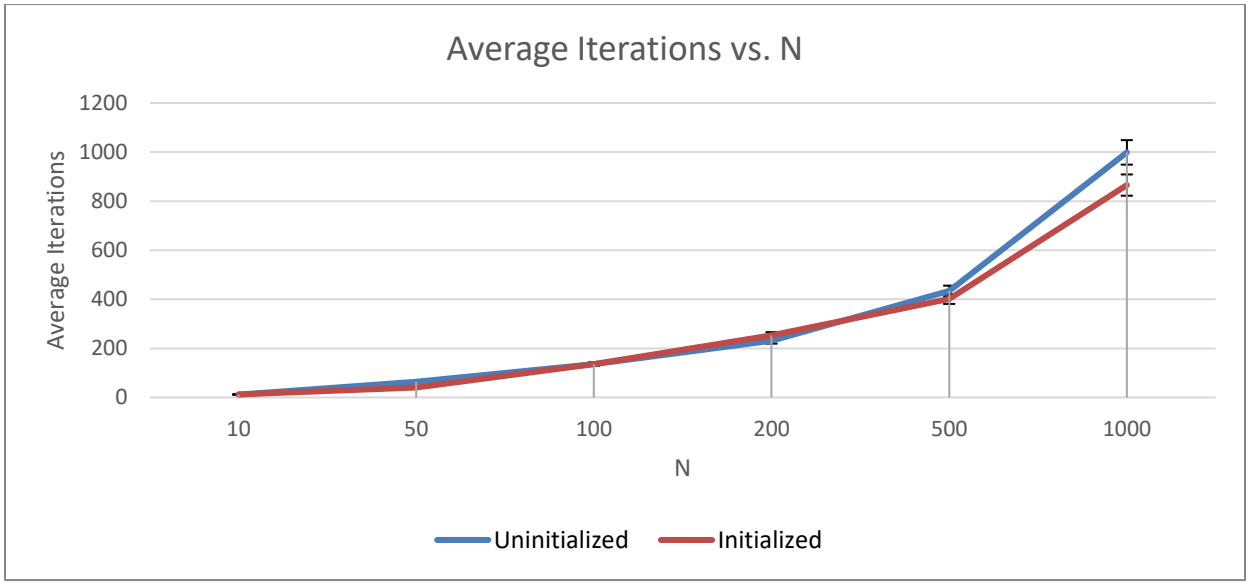

Figure 16

As the figure above shows the performance between the uninitialized and the initialized versions of the PLA algorithm are very similar but the initialized version performed better for the number of iterations to converge on higher values of N.

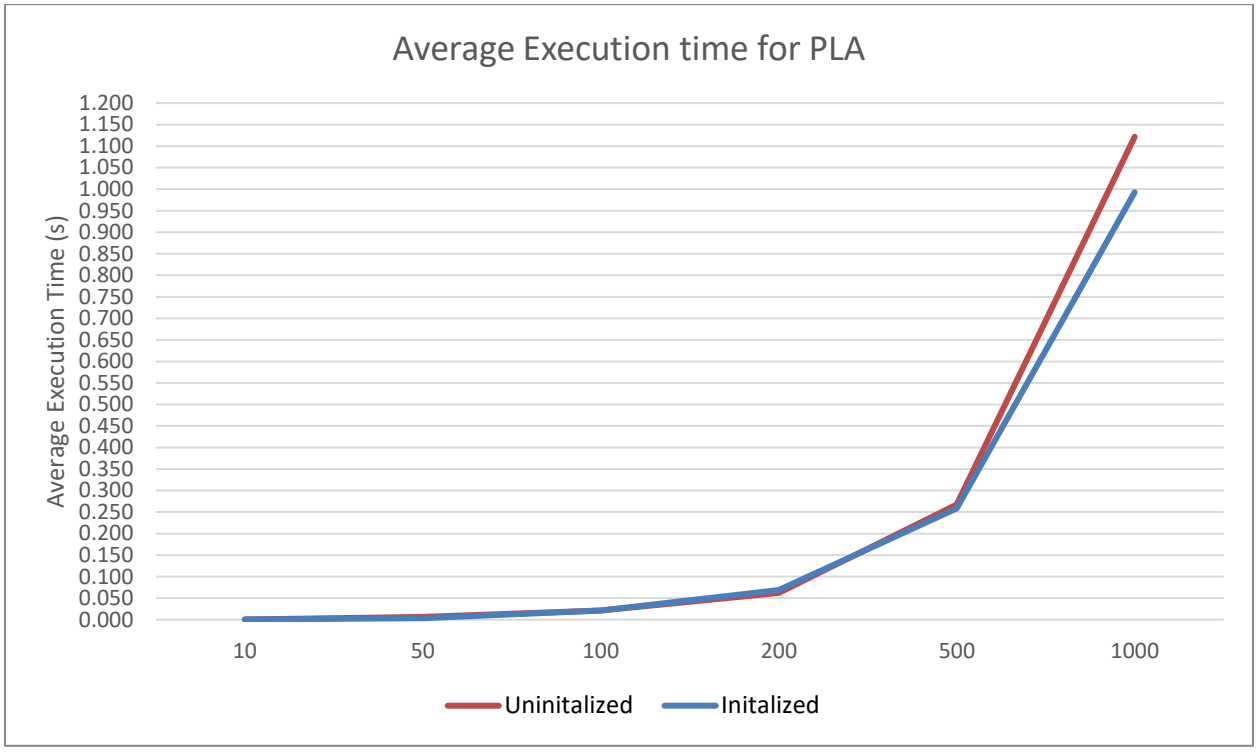

Figure 17

Similarly, the execution time for the initialized version of the PLA also had a slight improvement, which can be directly attributed to the reduction in the number of iterations required to converge.

4. CONCLUSION

Based on the results described above the "pseudo inverse" version of the PLA has minor performance increases when it comes to number of iterations required for convergence, this directly translates into a minor performance increase. This gap between the uninitialized and initialized version would likely increase as N approached infinity. Some additional things that could have been done to improve performance and learning would have been to use a gradient decent algorithm with an exponential or squared loss function to improve learning performance. The gradient decent with a regularize would have likely improved the uninitialized PLA more than the initialized PLA.