CSDS 391 Project 2 Written Report

By Mehlam Saifudeen

Exercise 1. Clustering

a) Write a program that implements the k-means clustering algorithm on the iris data set. You should use the objective function and learning rule you derived in W4 Q1 (and your implementation will help you uncover any errors, so it is helpful to do this before you turn in W4.) 15 P.

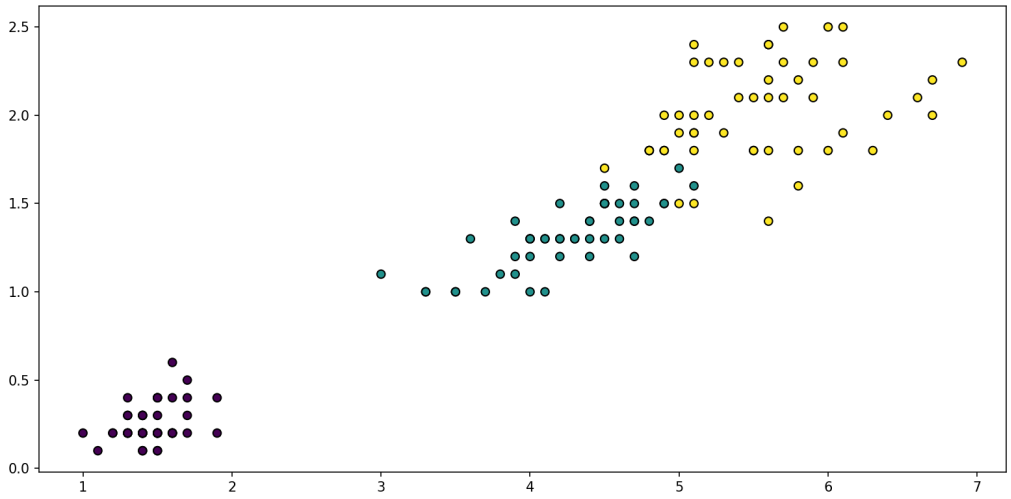
b) Plot the value of the objective function as a function of the iteration 5 P.

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{n,k} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$$

to show that your learning rule and implementation minimizes this expression.

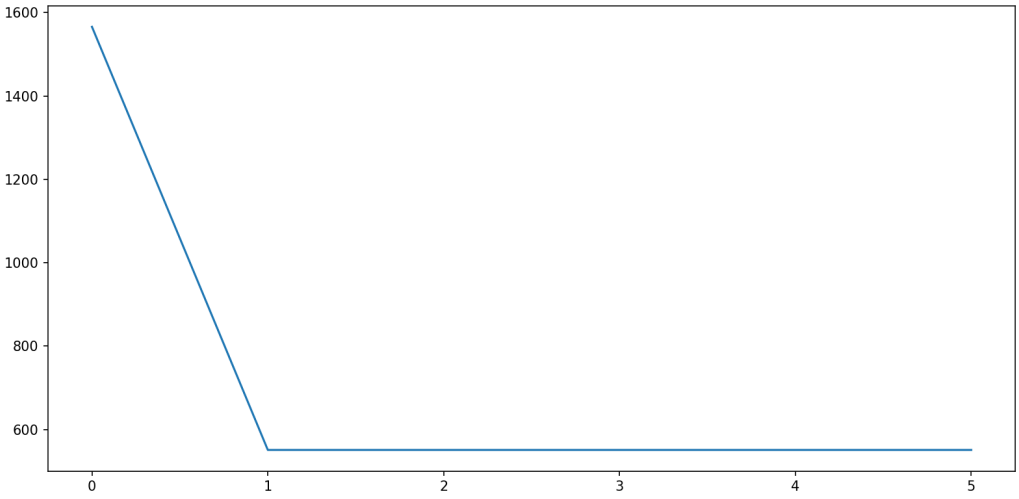
c) Plot the results of the learning process by showing the initial, intermediate, and converged cluster centers overlaid on the data for $k = 2$ and $k = 3$. 5 P.

d) Devise a method to plot the decision boundaries for this dataset using the optimized parameters. Explain your approach and plot your results. 10 P.



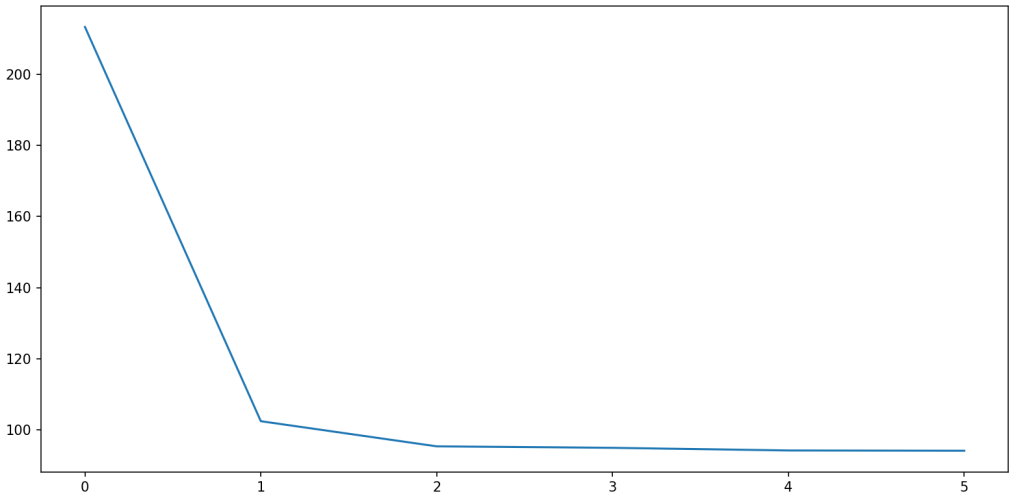
a.

The above graph shows the k-means clustering on the iris data set using the objective function and learning derived in W4 Q1.

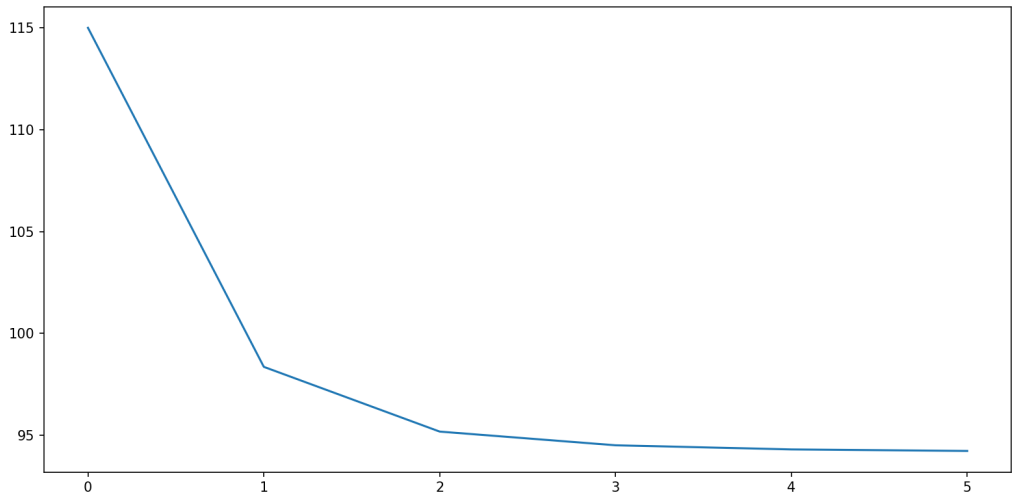


b.

The above graph shows the value of the objective function when k = 1 and hence shows that the learning rule and implementation minimizes this expression

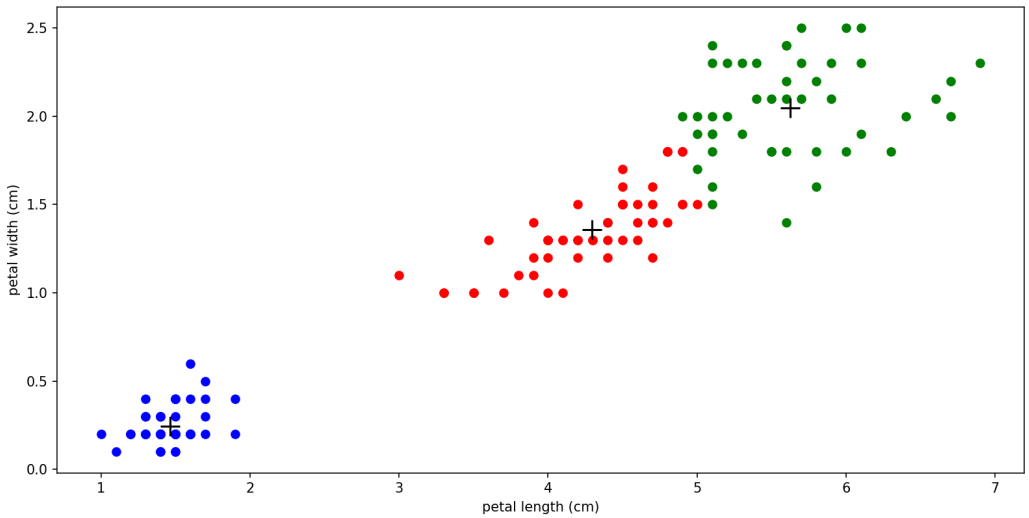


c.



The above graphs show the learning process for when k = 2 and k = 3.

d.



The above graph plots the decision boundaries of the data set using optimized parameters. The centroids are shown for each of the clusters generated.

The above graphs are generated when running the code is present in the file named Project2_Q1.

Exercise 2. Linear decision boundaries

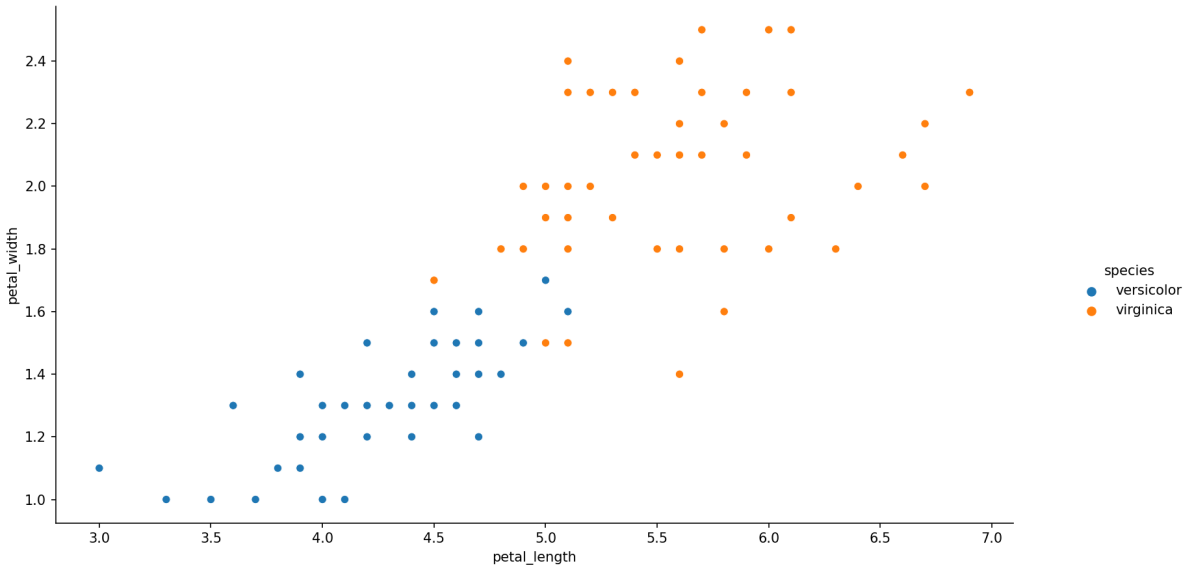
a) Inspect `irisdata.csv` which is provided with the assignment file on Canvas. Write a program (or use a library) that loads the iris data set and plots the 2nd and 3rd iris classes, similar to the plots shown in lecture on neural networks (i.e. using the petal width and length dimensions). 10 P.

b) Define a function that computes the output of simple one-layer neural network using a sigmoid non-linearity (linear classification with sigmoid non-linearity). 5 P.

c) Write a function that plots the decision boundary for the non-linearity above overlaid on the iris data. Choose a boundary (by setting weight parameters by hand) that roughly separates the two classes. Use an output of 0 to indicate the 2nd iris class, and 1 to indicate the 3rd. 5 P.

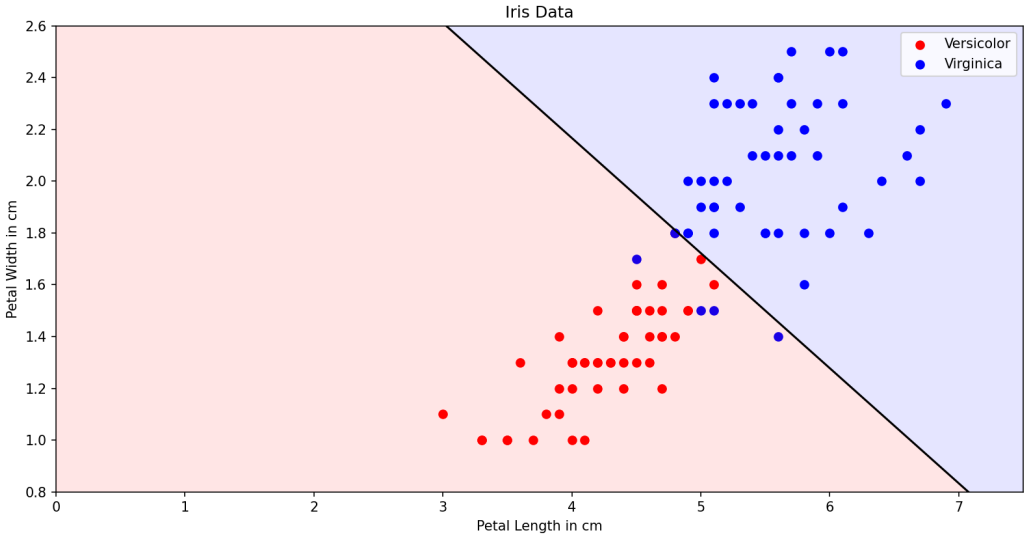
d) Use a surface plot from a 3D plotting libary (e.g. in matlab or using matplotlib in python) to plot the output of your neural network over the input space. This should be similar to learning curve shown in fig 19.17 (18.17 in 3rd ed.) of the textbook. 5 P.

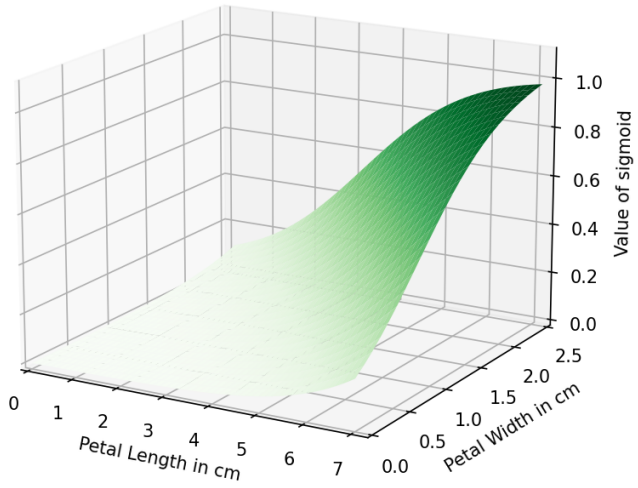
a.



b. We write the function abstractly as 2 parts shown below:
α (y) = 1 / (1 + e ^ (-y)), where y = m^**T** x + b = m1x1 + m2x2 + b.
Here x is the data vector where x1 = petal length and x2 = petal width and y is the output value.

c. Linear decision boundary **y = mx + b = 0**
For the parameters I chose m = [0.6 1.8] and b = -5.73





d.

Above is the surface plot from a 3d plotting library to plot the output of the neural network over the input space.

e. In order to answer this question I wrote a function that prints the petal length, width, species, and the classifier output for each of the data points. The output of the function is shown below:

Data points of virginica petals
Class: [0.81757448]
Petal Length: [5.8] , Petal Width: [2.2] , Class: ['virginica'] , Classifier Output: virginica
Class: [0.86529695]
Petal Length: [6.7] , Petal Width: [2.] , Class: ['virginica'] , Classifier Output: virginica
Class: [0.84553473]
Petal Length: [5.6] , Petal Width: [2.4] , Class: ['virginica'] , Classifier Output: virginica
Data points of versicolor petals
Class: [0.30576366]
Petal Length: [4.7] , Petal Width: [1.4] , Class: ['versicolor'] , Classifier Output: versicolor
Class: [0.25350602]
Petal Length: [4.6] , Petal Width: [1.3] , Class: ['versicolor'] , Classifier Output: versicolor
Class: [0.38225213]
Petal Length: [4.9] , Petal Width: [1.5] , Class: ['versicolor'] , Classifier Output: versicolor
Data points on / around the decision boundaries
Class: [0.39174097]
Petal Length: [4.5] , Petal Width: [1.7] , Class: ['virginica'] , Classifier Output: versicolor
Class: [0.49000133]
Petal Length: [5.0] , Petal Width: [1.7] , Class: ['versicolor'] , Classifier Output: versicolor
Class: [0.46505705]
Petal Length: [5.1] , Petal Width: [1.6] , Class: ['versicolor'] , Classifier Output: versicolor

From the data above we can see that the classifier returns expected outputs closer to 1 and 0 for the individual petal data points but for those which are on / around the decision boundary I ended up choosing a virginica and versicolor, both of which were misclassified by the classifier. Another virginica was chosen which was close and properly classified. The actual value returned by the

classifier for the proper data points is much closer to 0.5. In order to obtain this some data points were taken out of the iris data

The above graphs and values are obtained by running the code in the file named main.py which combines code from other files added in the zip folder.

Exercise 3. Neural networks

a) Write a program that calculates the mean-squared error of the iris data for the neural network defined above. The function should take three arguments: the data vectors, the parameters defining the neural network, and the pattern classes. 10 P.

b) Compute the mean squared error for two different settings of the weights (i.e. two dfiferent decision boundaries). Like above, select these by hand and choose settings that give large and small errors respectively. Plot both boundaries on the dataset as above. 5 P.

c) Give a mathematical derivation the gradient of the objective function above with respect to the neural network weights. You will have to use chain rule and the derivative of the sigmoid function as discussed in class. Use w_0 to represent the bias term. You should show and explain each step. 10 P.

d) Show how the gradient can be written in both scalar and vector form. 5 P.

e) Write code that computes the summed gradient for an ensemble of patterns. Illustrate the gradient by showing (i.e. plotting) how the decision boundary changes for a small step. 10 P.

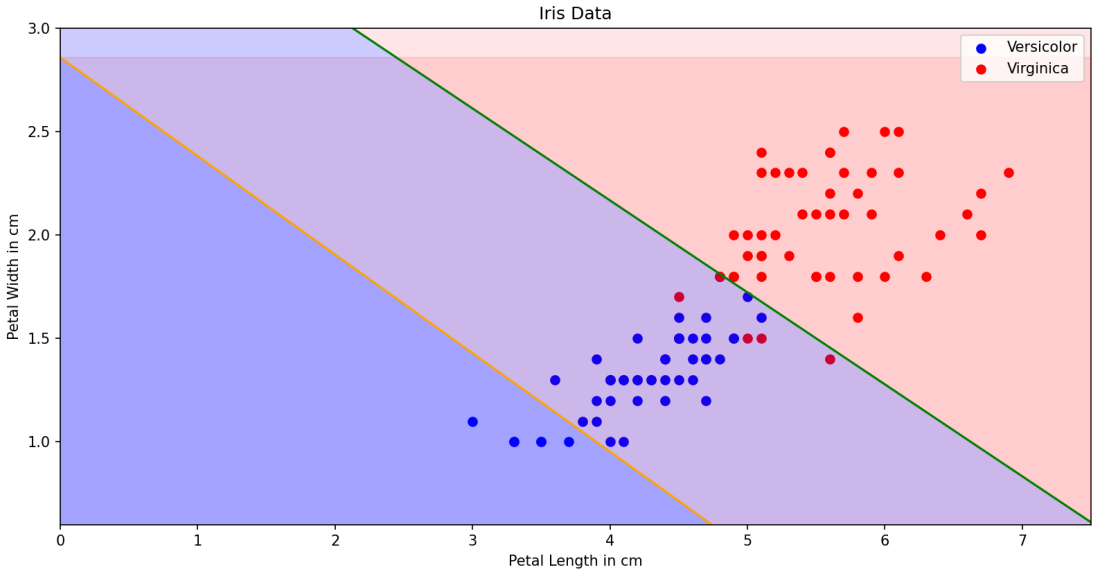
3a.

The code written calculates the root mean square error value by using the below equation:

$$\frac{1}{n}\sum_{i=1}^{n}(\text{class}(\mathbf{x_i}) - \text{prediction}(\mathbf{x_i}))^2$$

The function `def mse(X, m, b, species)in the file Project2_Q3` will calculate the root mean-squared error values of the iris data for the neural network defined above. The function takes in parameters as given in the question.

3b. To select a good choice of parameters, I chose what I used in part 2c which is m [0.6 1.8] and b = -5.73. Similarly for a bad choice of parameter I chose m = [1.1 2.2] and b = -5. Both the decision boundaries are plotted in green and orange for the good and bad choice parameters respectively. The good choice which is the green boundary splits the data well.



3c.

3c) When we consider one input data vector $x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$,

the objective function which we want to decrease is the MSE

$$MSE(x) = (\text{class}(x) - \text{prediction}(x))^2$$

$y(x)$ = the class, $w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$ = the weights and

$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$ which is the input data

Prediction function $(\sigma(w^T x)) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$

We therefore write the function as:-

$$(y(x) - \sigma(w^T x))^2$$

Next we take the partial derivative with respect to w_i.

$$\frac{\partial}{\partial w_i} (y(x) - \sigma(w^T x))^2 = 2(y(x) - \sigma(w^T x)) \frac{\partial}{\partial w_i} (y(x) - \sigma(w^T x))$$

$$-2(y(x) - \sigma(w^T x))(-\sigma'(w^T x)) \frac{\partial}{\partial w_i} (w^T x)$$

$$= 2(y(x) - \sigma(w^T x))(-\sigma'(w^T x)) x_i$$

$$= -2\left(y(x) - \frac{1}{1+e^{-\omega^T x}}\right)\left(\frac{1}{1+e^{-\omega^T x}}\left(1 - \frac{1}{1+e^{-\omega^T x}}\right)\right)x_i$$

$$= -2\left(y(x) - \frac{1}{1+e^{-\omega^T x}}\right)\left(\left(\frac{1}{1+e^{-\omega^T x}} - \left(\frac{1}{1+e^{-\omega^T x}}\right)^2\right)\right)x_i$$

$$= -2\left(y(x) - \frac{1}{1+e^{-\omega^T x}}\right)\left(\left(\frac{1+e^{-\omega^T x}}{1+e^{-\omega^T x}}\right)^2 - \left(\frac{1}{1+e^{-\omega^T x}}\right)^2\right)\right)x_i$$

$$= -2\left(y(x) - \frac{1}{1+e^{-\omega^T x}}\right)\left(\left(\frac{e^{-\omega^T x}}{1+e^{-\omega^T x}}\right)^2\right)\right)x_i$$

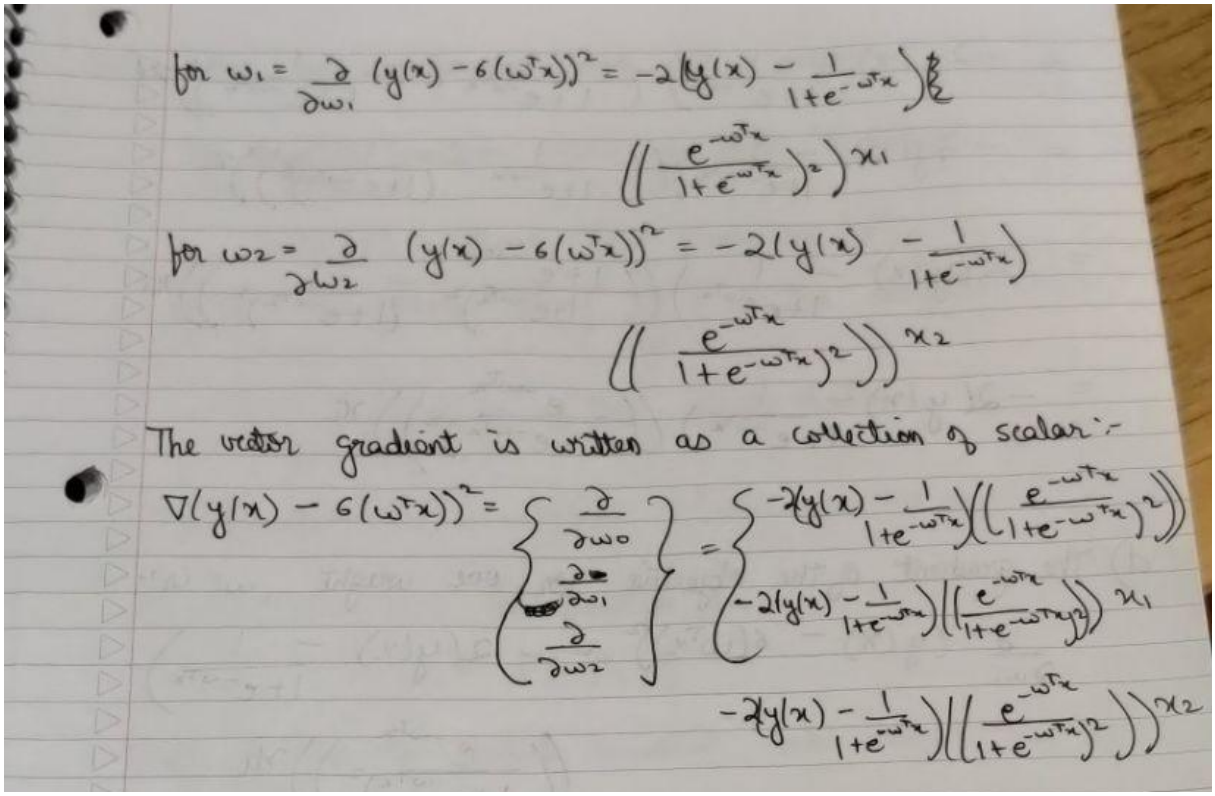
3d.

d) The gradient of the objective for one weight, w_i is:-

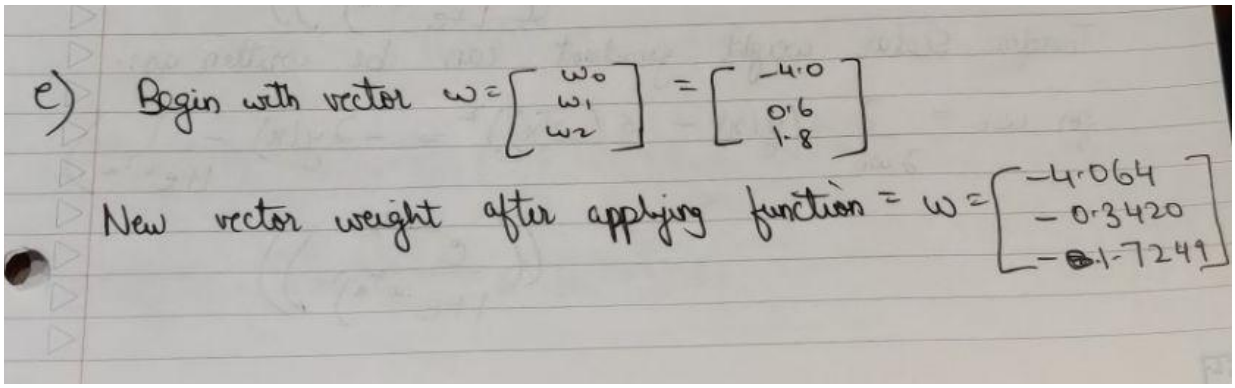
$$\frac{\partial}{\partial w_i}\left(y(x) - \sigma(\omega^T x)\right)^2 = -2\left(y(x) - \frac{1}{1+e^{-\omega^T x}}\right)\left(\left(\frac{e^{-\omega^T x}}{1+e^{-\omega^T x}}\right)^2\right)\right)x_i$$

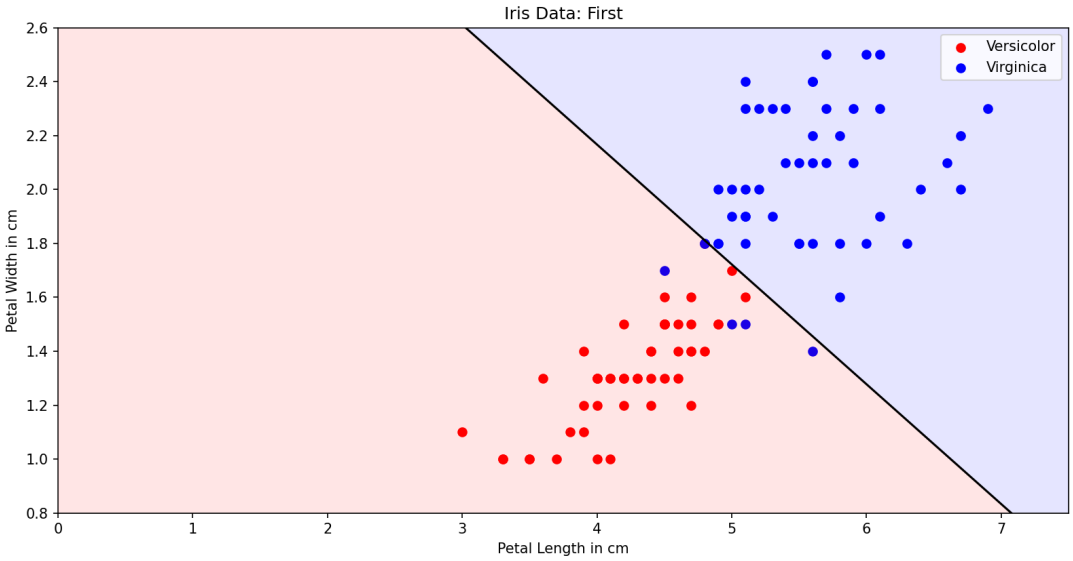
Therefore scalar weight gradient can be written as:-

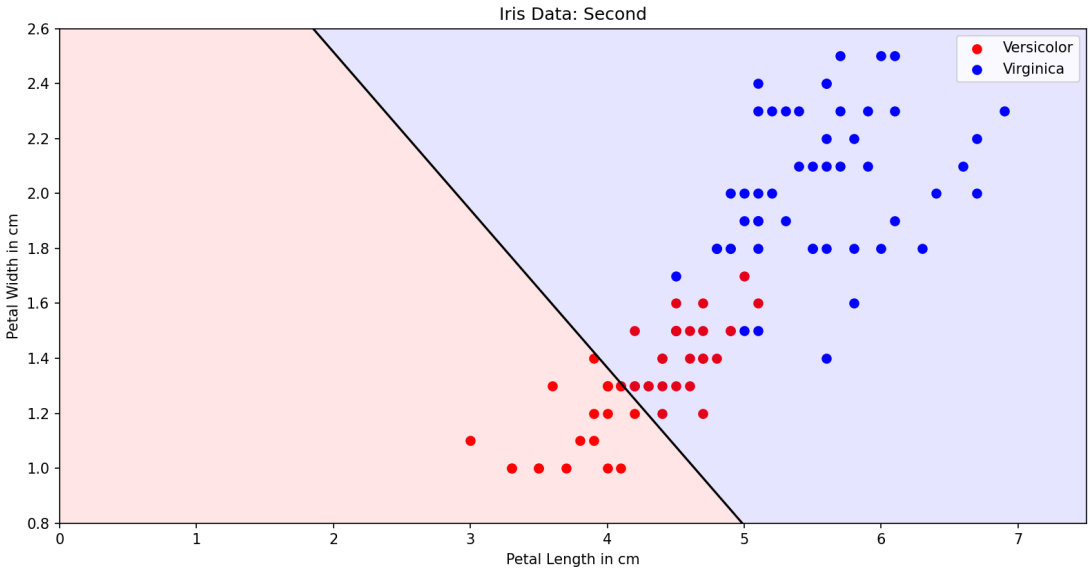
$$\text{for } \omega_0 = \frac{\partial}{\partial \omega_0}\left(y(x) - \sigma(\omega^T x)\right)^2 = -2y(x) - \frac{1}{1+e^{\omega^T x}}\left(\left(\frac{e^{-\omega^T x}}{1+e^{-\omega^T x}}\right)^2\right)\right)$$


for $w_1 = \frac{\partial}{\partial w_1}(y(x) - \sigma(w^T x))^2 = -2\left(y(x) - \frac{1}{1+e^{-w^T x}}\right)\left(\left(\frac{e^{-w^T x}}{1+e^{-w^T x}}\right)^2\right) x_1$
for $w_2 = \frac{\partial}{\partial w_2}(y(x) - \sigma(w^T x))^2 = -2\left(y(x) - \frac{1}{1+e^{-w^T x}}\right)\left(\left(\frac{e^{-w^T x}}{1+e^{-w^T x}}\right)^2\right) x_2$
The vector gradient is written as a collection of scalar:-
$\nabla(y(x) - \sigma(w^T x))^2 = \begin{Bmatrix} \frac{\partial}{\partial w_0} \\ \frac{\partial}{\partial w_1} \\ \frac{\partial}{\partial w_2} \end{Bmatrix} = \begin{Bmatrix} -2\left(y(x) - \frac{1}{1+e^{-w^T x}}\right)\left(\left(\frac{e^{-w^T x}}{1+e^{-w^T x}}\right)^2\right) \\ -2\left(y(x) - \frac{1}{1+e^{-w^T x}}\right)\left(\left(\frac{e^{-w^T x}}{1+e^{-w^T x}}\right)^2\right) x_1 \\ -2\left(y(x) - \frac{1}{1+e^{-w^T x}}\right)\left(\left(\frac{e^{-w^T x}}{1+e^{-w^T x}}\right)^2\right) x_2 \end{Bmatrix}$

3e.


e) Begin with vector $w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -4.0 \\ 0.6 \\ 1.8 \end{bmatrix}$
New vector weight after applying function = $w = \begin{bmatrix} -4.064 \\ -0.3420 \\ -1.7249 \end{bmatrix}$





MSE of First Decision Boundary: 0.08710012804256104
MSE of Second Decision Boundary: 0.2688489937921696
I have graphed the first and second weights' decision boundaries as seen above.

The above graphs and values are obtained by running the code in the file named main.py which combines code from other files added in the zip folder.

Exercise 4. Learning a decision boundary through optimization

a) Using your code above, write a program that implements gradient descent to optimize the decision boundary for the iris dataset. 5 P.

b) In your program, include code that shows the progress in two plots: the first should show the current decision boundary location overlaid on the data; the second should show the *learning curve*, i.e. a plot of the objective function as a function of the iteration. 5 P.

c) Run your code on the iris data set starting from a random setting of the weights. Note: you might need to restrict the degree of randomness so that the initial decision boundary is visible somewhere in the plot. In your writeup, show the two output plots at the initial, middle, and final locations of the decision boundary. 10 P.

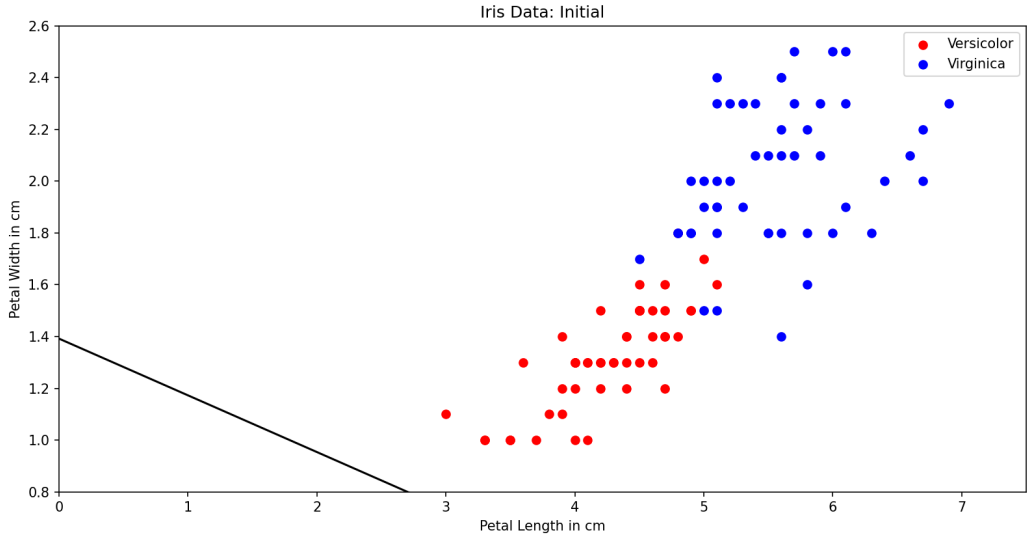
d) Explain how you chose the gradient step size. 5 P.

e) Explain how you chose a stopping criterion. 5 P.

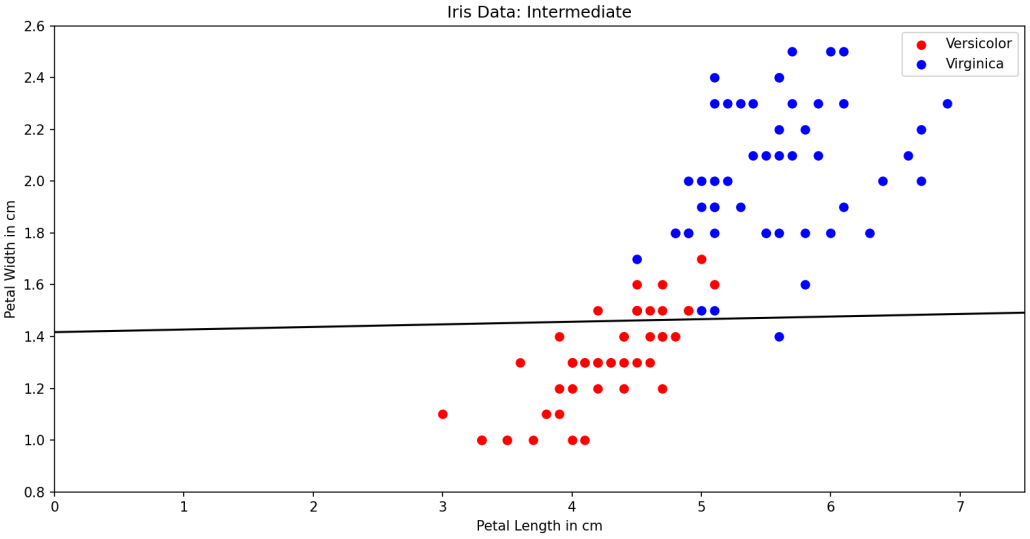
a. The function `def fit(classifier, step_size, X_augmented, petal_length, petal_width, species, progress_output=False)` `in the file Project2_Q4.py` does the work of implementing the gradient descent. When I passed a matrix containing 100 3 x 1 data vectors, the step size, and a simple classifier model object with weights, it generated the graphs and plots needed for part b of this question. If the value of the last parameter which is progress_output is changed to TRUE we can print the value of the final weights vector.

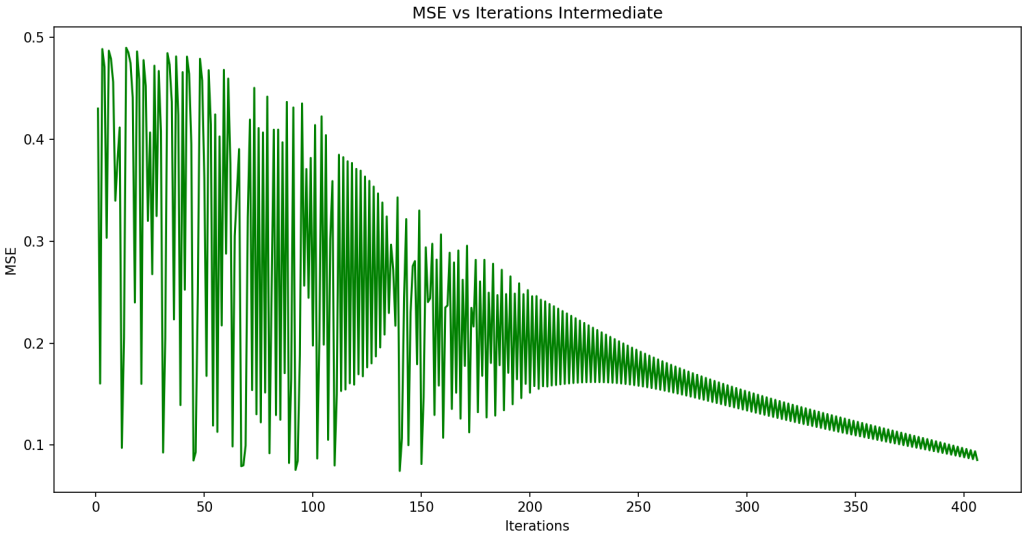
b. When we call the function `def fit(classifier, step_size, X_augmented, petal_length, petal_width, species, progress_output=True)`, there are 2 things that happen
   - It creates 3 decision boundary plots for the first, middle, and last iterations by calling the function `def plot_loss_over_iterations(mse_store, num_iterations, subtitle = None)` `function 3 times from the same python file as in 4a.`
   - Plots the MSE (loss) over the number of iterations plot.

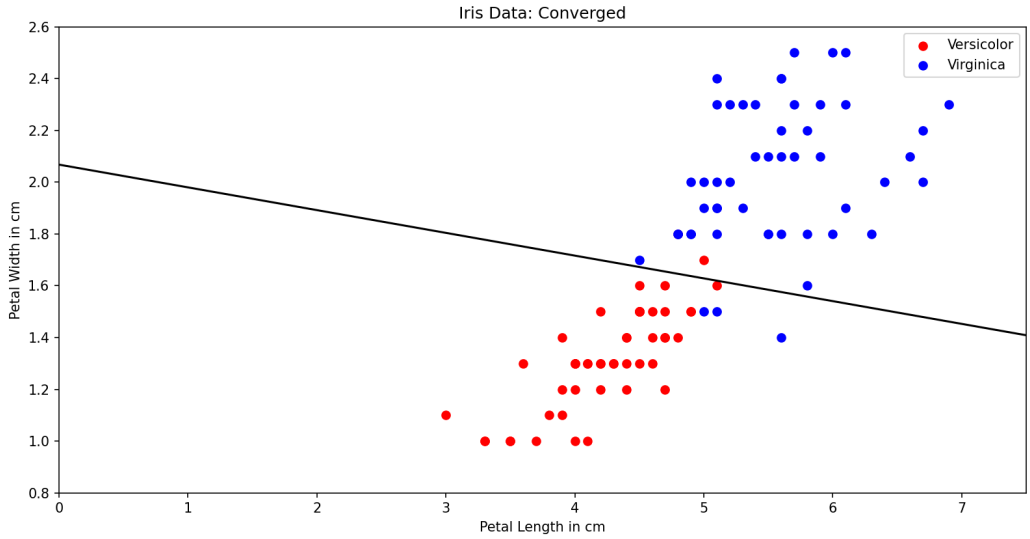
c. **PartC**

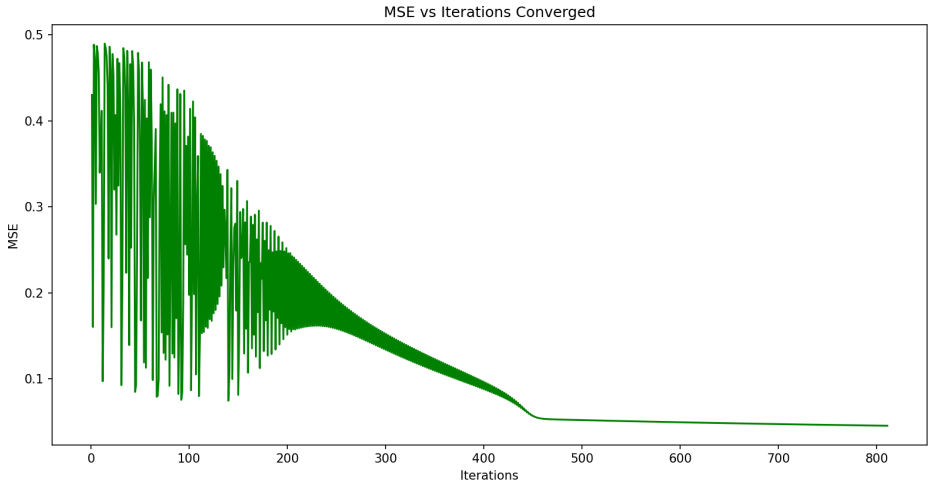


There would be no loss graph here because it would be empty at the beginning.




MSE vs Iterations Intermediate
MSE
Iterations
0.5
0.4
0.3
0.2
0.1
0
50
100
150
200
250
300
350
400


Iris Data: Converged
Petal Width in cm
Petal Length in cm
Versicolor
Virginica
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
1
2
3
4
5
6
7



d. Gradient descent identifies the optimal value by taking big steps when we are far away from the optimal sum of the squared residual, and start to make many steps when it is close to the best solution. Then we can calculate the derivative d of each point of the function created by the points. In other words, we are taking the derivative of the loss function.
Gradient Descent uses derivatives in order to find where the sum of the squared residuals is lowest. The closer we get to the optimal value for the intercept, the closer the slope of the curve gets to zero.
Step size = slope * learning rate.
To summarize the steps of choosing the gradient step size we have:
    - Take the derivative of the Loss Function for each parameter in it, and pick random values for parameters.
    - Plug the parameter values into the derivatives (Gradient).
    - Calculate the Step Sizes: Step Size = Slope * Learning Rate.

    The step size that I found = alpha = 0.0025

e. When the function is when ∇MSE = 0. Although considering how small the set size is this seems very unrealistic due to some errors and miscalculations creeping in. Hence, upon testing the values for a threshold such that the stopping criteria would satisfy ∇MSE is less than a particular value. When comparing the loss vs number of iterations in the graph, the change in loss plateaued and the threshold value obtained was **0.25** which could eliminate redundant values.

Exercise 5. Extra credit: Using a machine learning toolbox

Use a machine learning toolbox such as keras or scikit-learn to generalize the network to perform classification on the full iris dataset. Your network should have one or more hidden layers and classify all three iris classes using all four data dimensions. Show your results and explain your architecture. +27 P.

In addition to implementing the neural network classifier, also think of an issue to explore. Write your own mini-tutorial to report your results. Here are some examples of topics you can consider:

- Illustrate how a multi-layer network makes a decision.
- Explore different non-linearities.
- Explore different network architectures and their classication and generalization performance.
- Contrast two different machine learning algorithms (e.g. a neural network vs k-means clustering) and how they perform on the full iris data set.
- Explore algorithms for adjusting the learning rate.

#Extra Credit: The main goal that I am trying to address through this extra credit problem is to classify the 3 iris classes using keras as a learning toolbox. The code *extra_credit_Q5.py* contains a simple neural network written in keras to classify the iris data set.

In order to achieve this I first started with importing the iris data set and a couple other libraries through sklearn dataset followed by using the model_selection and the preprocessing libraries from the API. The model_selection allowed me to set a blueprint of the data and then measure and analyze new data. Selecting a proper model allows us to generate accurate results when using machine learning to make predictions. To implement this library I had to train the data set using the train_test_split which allows splitting of random matrices into random train and test subsets. The preprocessing package helped in other utility functions related to vectors.

Next I built the network using the keras library. In particular I used the functions Adam, Sequential and Dense. Adam allows for optimized learning rate of the neural network for first and second order moments. Sequential is a way of creating deep learning models and densely allows me to create instances of deep learning models. The Adam function takes a learning rate and I tried using this for several values starting from 1.0, 0.5, 0.1, to 0.01, 0.001, 0.0001, and 0.00001. The accuracy generated for each of these is in order of 0.33, 0.27, 0.9, 0.91, 0.96, 0.98, and 0.99 respectively. Hence the more we decrease the learning rate, the more accurate the model gets. In addition to this as the value of the learning rate goes down, the test loss rate increases as well. To verify this the Adam(lr) function is present on line number 31 of the *extra_credit_Q5.py* file.

Hence it shows that the higher the learning rate, the less accurate the classification model gets. The learning rate plays an important role in the classification and network building of data sets.