Introduction to Neural Networks Assignment 02

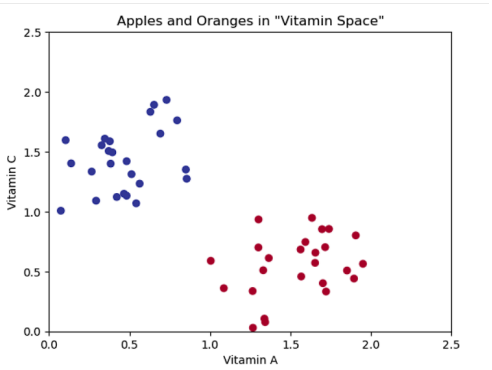
Mangesh Sakordekar

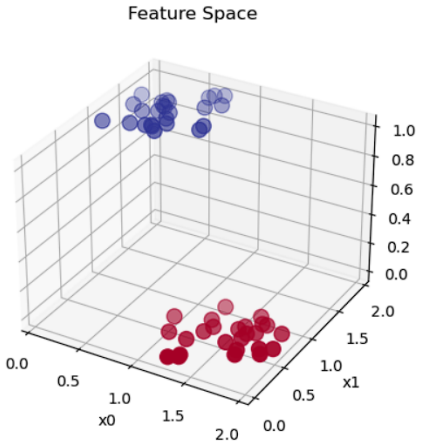
The aim of this project was to perform gradient descent to find the 2 weights of a single neuron (with no bias) that classifies apples and oranges. The steps followed includes:

- Read in the fruit_data.txt file and visualize apple and orange in feature vectors and labels.
- Calculate the Mean Squared Error of the model weight values in the range [-20,20] and plot the surfaces.
- Calculate what the model surface is for initial weight values of [-5,5], and plot in the feature space.
- Implement the gradient descent algorithm, with a variable learning rate, epsilon, to find a sufficiently good solution for the weights
- Plot the trajectory of the weight values during optimization on top of the contour plot of the error surface
- Plot the model surface, and data, for the final weight solution.

Feature Space

Looking at the feature space, we can clearly see the difference between the two clusters formed by the vitamin A and Vitamin C contents of the fruit.



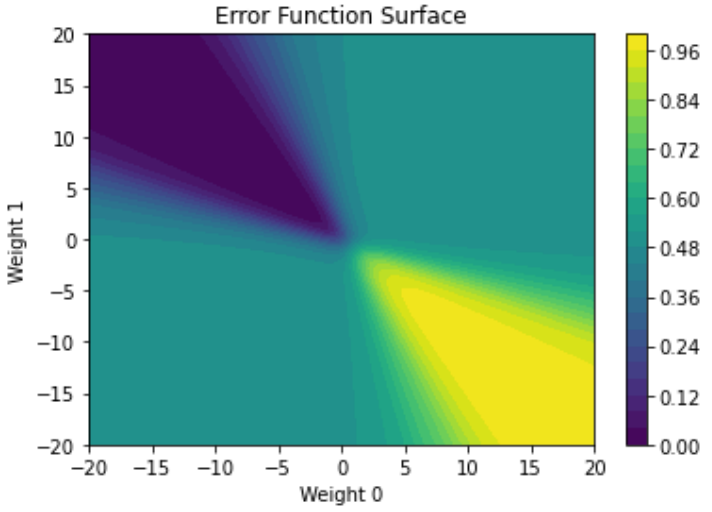


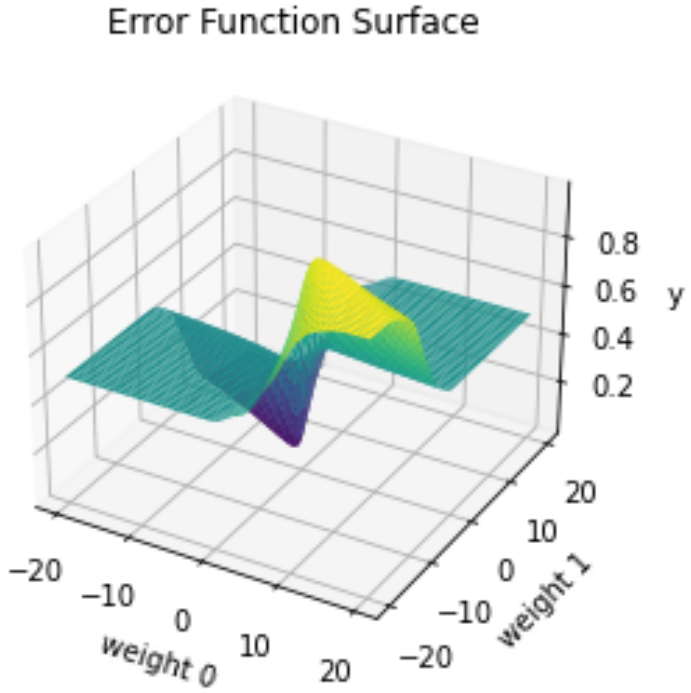
Error Function Surface

To plot the Error Function Surface, a meshgrid was created of the range [-20,20]. Using the meshgrid, mean squared error for the predictions of each weight combination within the range was calculated. Following the neuron diagram provided, sigmoid function was used as the activation function. The code and the plots can be seen below.

```
xx, yy = np.meshgrid(np.linspace(-20, 20, 500), np.linspace(-20, 20, 500), indexing="xy")
xy = np.dstack((xx, yy))
```

```
points = data[:, 0:2]
labels = data[:, 2]
zz = np.ndarray((500 , 500))
for i in range(0, 500):
    for j in range(0, 500):
        curr_sum = 0
        preds = 1/(1 + np.exp(-np.dot(points, xy[i][j])))
        mse = np.sum((preds - labels)**2) / preds.shape[0]
        zz[i][j] = mse
```


Error Function Surface
Weight 1
Weight 0
20
15
10
5
0
−5
−10
−15
−20
0.96
0.84
0.72
0.60
0.48
0.36
0.24
0.12
0.00


Error Function Surface
0.8
0.6
0.4
0.2
y
−20
−10
0
10
20
weight 0
20
10
0
−10
−20
weight 1

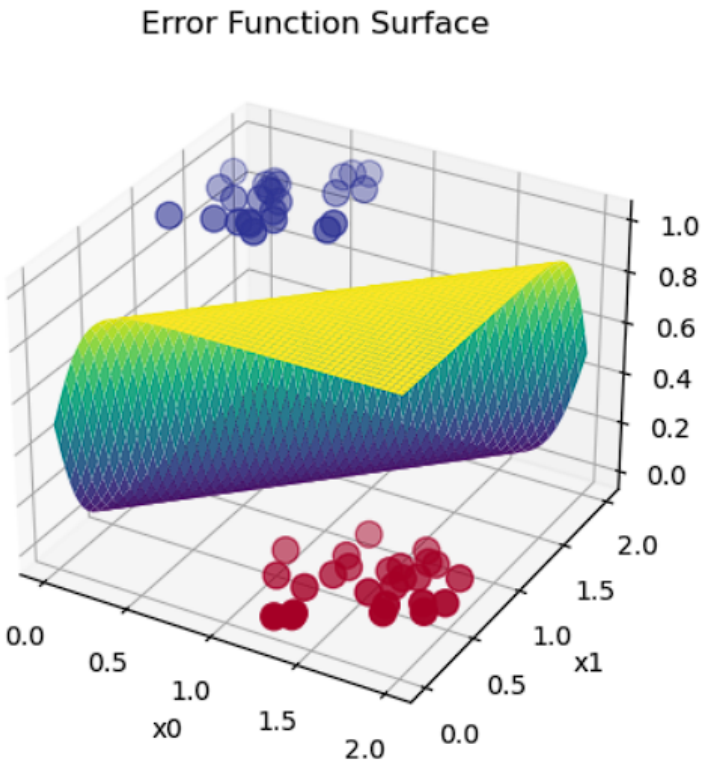
Error Surface for Weights (5,-5)

Again, a meshgrid was used in the range [0,2] to generate the dataset. Predictions were calculated on this dataset following the neuron diagram provided with the wights set to w0 = 5 and w1 = -5.

```
x, y = np.meshgrid(np.linspace(0, 2, 50), np.linspace(0, 2, 50), indexing="xy")
xy1 = np.dstack((xx, yy))

z = np.ndarray((50 , 50))
for i in range(0, 50):
    for j in range(0, 50):
        z[i][j] = 1/(1 + np.exp(-np.dot(xy1[i][j], [5,-5])))
```

These predictions were used the plot the error surface generated by the model in the range [0,2].



Looking at this surface plot we can clearly see that even though this weight combination does the job of separating apples from oranges efficiently, it classifies apples as oranges and vice versa. Thus, this is definitely not a good model and a better model can be achieved.

Gradient Descent

To find the ideal weights, gradient descent was performed on the error surface generated for the [-20,20] range. Basic idea of gradient descent is we move along the gradient of the surface untill we find the lowest point, in this case the lowest error. As we discussed in the class, we know that:

$$\overrightarrow{V_\Theta} E(\Theta) = \begin{bmatrix} \delta E/\delta w_0 \\ \delta E/\delta w_1 \end{bmatrix} \tag{1}$$

$$\frac{\delta E}{\delta w_j} = \frac{2}{N} \times \sum_{i=0}^{N-1} (Y_{pred}^{(i)} - Y^{(i)}) \cdot Y_{pred}^{(i)} \cdot (1 - Y_{pred}^{(i)}) \cdot X_j^{(i)} \tag{2}$$

In this algorithm we calculate the gradient vector ($\overrightarrow{V_\Theta} E(\Theta)$) for the current point (starting at (5,-5)) we are on and move in the direction.

```
w0 = 5
w1 = -5

xpath = [w0]
ypath = [w1]

mse = 1

learning_rate = 2.5

while mse > 0.00001:
    preds = 1/(1 + np.exp(-np.dot(points, [w0, w1])))

    mse = np.sum((preds - labels)**2) / preds.shape[0]
    if mse < 0.00001:
        break

    dw0 = 2 * np.sum( (preds - labels) * preds * (1-preds) * points[:, 0]) / 50.0
    dw1 = 2 * np.sum( (preds - labels) * preds * (1-preds) * points[:, 1]) / 50.0

    w0 -= learning_rate * dw0
    w1 -= learning_rate * dw1

    xpath.append(w0)
    ypath.append(w1)

print(w0, w1)
```

```
-1.1022803748031288e+01 10.739700326782241
```

For the current point in weights space, we calculate the mean squared error and break out of the loop if the error is less than the threshold. In this implementation a threshold of 0.00001 was used.Once the gradient vector is calculated, we move in that direction by updating w_0 and w_1. The magnitude of the movement is determined by the learning rate. For this algorithm a learning rate of 2.5 was used. Coordinates of each point in the weight space we moved to is added to a list to plot the path of the descent. This path can be seen in the plot below.

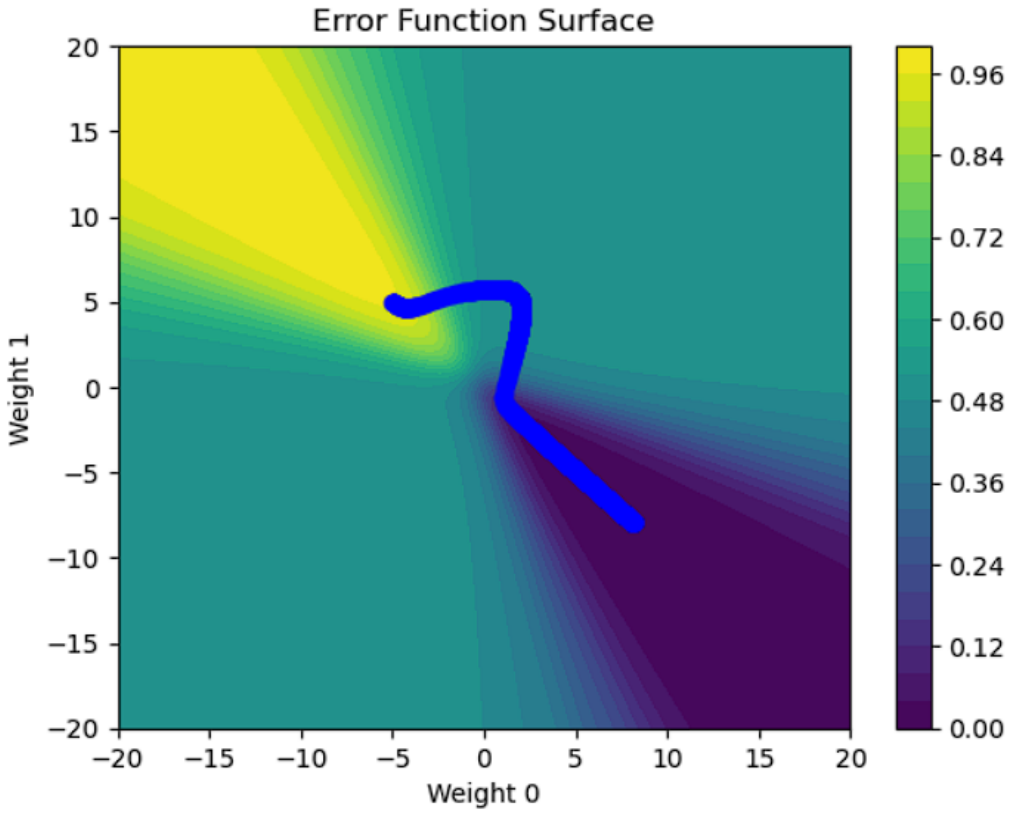


Figure 1: Path of the Gradient Descent

The final weights achieved by this model were :
w_0 = -11.022803748031288
w_1 = 10.739700326782241

Model Surface for the Final Weight Solution

Steps followed in this section were the same as plotting the error surface for weights (-5,5), but the weights used were the ones calculated by the model. Code and the model can be seen below

```
x, y = np.meshgrid(np.linspace(0, 2, 50), np.linspace(0, 2, 50), indexing="xy")
xy1 = np.dstack((xx, yy))

z = np.ndarray((50 , 50))
for i in range(0, 50):
    for j in range(0, 50):
        z[i][j] = 1/(1 + np.exp(-np.dot(xy1[i][j], [w0,w1])))
```

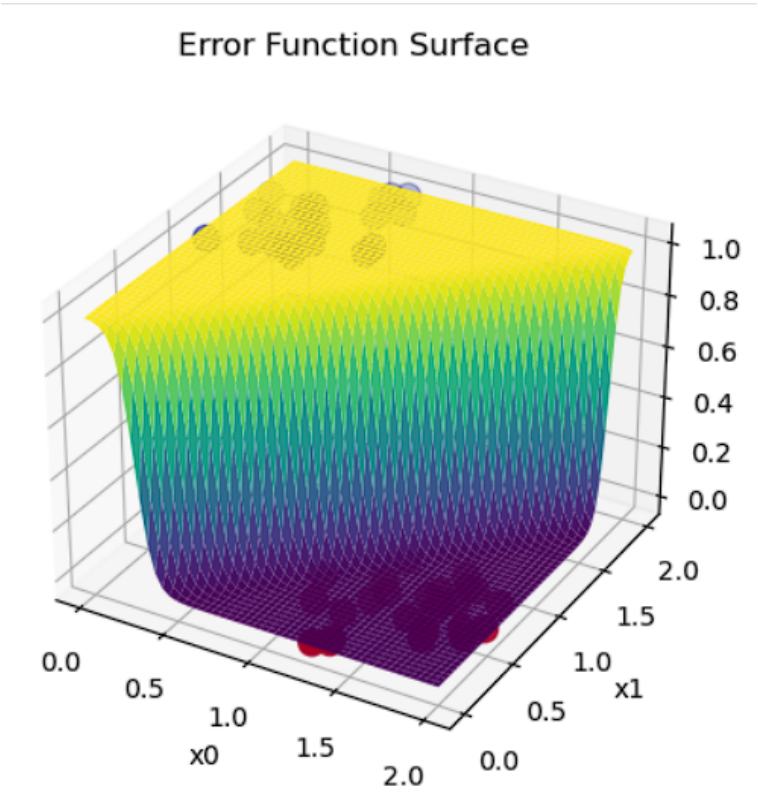


Figure 2: Model Surface for the Final Weight Solution

Helper Code

Following functions were used to plot the different graphs.

```
def plot3dgraph(plot_surface, surface, plot_data, data):
    fig = plt.figure()
    ax = plt.axes(projection='3d')
    if plot_data == True:
        ax.scatter3D(data[:, 0], data[:, 1], data[:, 2], c=data[:, 2], cmap='RdYlBu', s=100)
        ax.set_title('Feature Space')
    if plot_surface == True:
        ax.plot_surface(surface[0], surface[1], surface[2], cmap='viridis')
        ax.set_title('Error Function Surface')
    ax.set_xlabel('x0')
    ax.set_ylabel('x1')
    ax.set_zlabel('y')

    if plot_surface == True and plot_data == False:
        ax.set_xlabel('weight 0')
        ax.set_ylabel('weight 1')

    plt.show()
```

Figure 3: Function for 3D plots

```
def plot2dgraph(plot_surface, surface, plot_path, path):
    if plot_surface == True:
        plt.contourf(surface[0], surface[1], surface[2], levels=28)
    if plot_path == True:
        plt.plot(path[0], path[1], 'bo', linestyle="--")
    plt.xlabel('Weight 0')
    plt.ylabel('Weight 1')
    plt.title('Error Function Surface')
    plt.colorbar()
    plt.show()
```

Figure 4: Function for 2D plots