Data Engineering 414 ~ Practical 4: Multilayer Perceptron

Johan Neethling 24739286

2024 - 04 - 29

Introduction

We were provided with skeleton code and then asked to implement a multilayer perceptron.

Report Overview

Per question, we began by plotting the data to identify how we should design the separation of the data.

We then finish implementing the perceptron by implementing a forward pass, a backward pass.

Report

Question 1

Part a

```
#************************ YOUR CODE HERE ***********************#

colours = ['r' if x == 1 else 'g' for x in train_Y]
plt.scatter(train_X[:, 0], train_X[:, 1], c = colours)
plt.title("Scatterplot of Training Data")
plt.xlabel("x1")
plt.ylabel("x2")
plt.show()

#***************************************************************#
```

Figure 1: Code for 1a

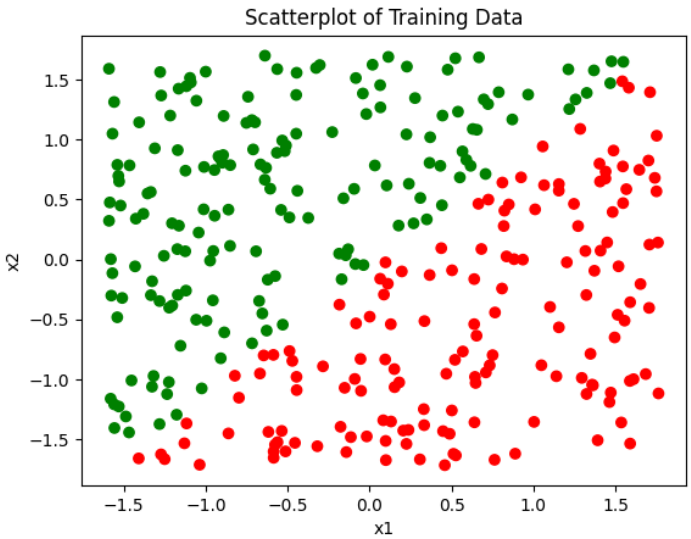

Figure 2: Output of code in 1a

We define how we want our datapoints to be coloured then plot using the matplotlib library.

We can also see that the data is clearly linearly separable.

Part b

```
#************************** YOUR CODE HERE **************************#

z_i = np.matmul(W.T, u_i)
v_i  = sigmoid(z_i)

#********************************************************************#
```

Figure 3: Code for 1b

We dot product the input vector (u[i]) and the correlating weights (W[i]), this is stored as the linear output (z[i]).

We also get the activation function output by applying the provided sigmoid function to the linear output.

$$v[i] = \frac{1}{1 + e^{-z[i]}}$$

Figure 4: Formula for the sigmoid function

Part c

```
#************************** YOUR CODE HERE **************************#

dL_dy_hat = (y - y_hat) / (y_hat*(1 - y_hat) + eps)
return dL_dy_hat

#********************************************************************#
```

Figure 5: Code for 1c

We implement calculation of the derivative of the binary cross-entropy, formula as seen above.

Part d

```
#************************ YOUR CODE HERE ************************#
arr = v_i*(1 - v_i)

dvi_dzi = np.diag(arr.flatten())
dzi_dvi_prev = W_i[1:, :]

Lambda = np.reshape(Lambda,[-1, 1])

for index, col in enumerate(W_i.T):
    dzi_dwi = np.zeros([u_i.shape[0], v_i.shape[0]])
    dzi_dwi[:, index] = u_i.flatten()

    dJ_dwi_column = np.matmul(np.matmul(dzi_dwi, dvi_dzi), Lambda)
    if dJ_dWi is None:
        dJ_dWi = dJ_dwi_column
    else:
        dJ_dWi = np.hstack((dJ_dWi, dJ_dwi_column))
Lambda = np.matmul(np.matmul(dzi_dvi_prev, dvi_dzi), Lambda).flatten()
#****************************************************************#
```

Figure 6: Code for 1d

Credit to Alex Petika, I had to use his code here, I could not get it to work on my own.

This section of code calculates the gradients of the weights for a single layer of our multilayer perceptron. $\frac{dv[i]}{dz[i]}$ = diagonal array of $v_q[i](1- v_q[i])$, this is a derivative of the activation layer. We slice a section of the weight matrix out and store it in dzi_dvi_prev for later use. We then reshape lambda into a column vector to align with our other matrices' shapes. In the for loop we create our get the derivative of our linear layer which is just a matrix of zeros with a column of $u_p[i]$ at column q. Finally, we update our lambda by doing the dot product of our previous lambda and the dot product of our current and previous $\frac{dv[i]}{dz[i]}$'s.

Part e

Epoch 0 loss: -0.2729 test_loss: -0.1529
Epoch 99 loss: -0.0351 test_loss: -0.0205

As shown above, our loss drops dramatically over the 100 epochs. This is a positive indication that our training worked well.

Part f

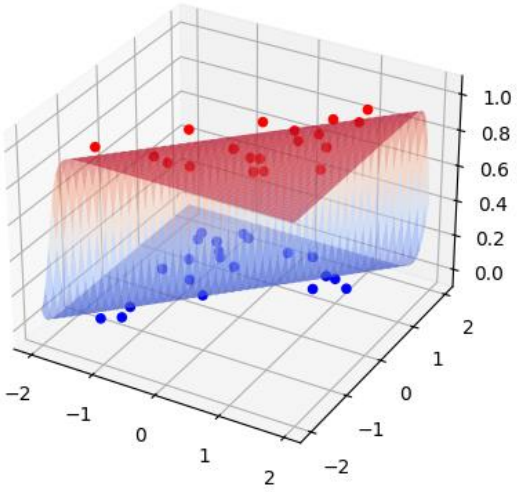

Figure 7: Result of plot_surface function

As you can see above, our model's decision surface is linear over the line we would expect from figure 2.

Question 2

Part a

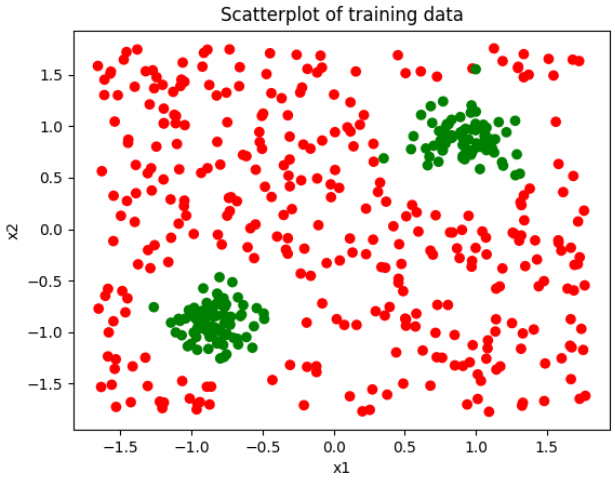

Figure 8: Result from code in 2a

We use the same code as in 1a and the data is clearly not linearly separable. There are, however, two obvious clusters.

Part b

Part b is simply a copy and paste implementation of the relevant questions from section 1. We do not need to hash it out.

Part c

```
#********************** YOUR CODE HERE **********************#

W_1 = np.random.uniform(-1 / np.sqrt(Q_1), 1 / np.sqrt(Q_1), size=(P_1 + 1, Q_1)) # initialise weights for layer W_1
W_2 = np.random.uniform(-1 / np.sqrt(Q_2), 1 / np.sqrt(Q_2), size=(P_2 + 1, Q_2))
W_3 = np.random.uniform(-1 / np.sqrt(Q_3), 1 / np.sqrt(Q_3), size=(P_3 + 1, Q_3))

#************************************************************#
```

Figure 9: Code for 2c

Here we simply added more layers following the defined method.

Part d

```
#************************ YOUR CODE HERE ************************#

z_2, v_2 = forward(np.vstack([np.array([1]), v_1]), W_2)
z_3, v_3 = forward(np.vstack([np.array([1]), v_2]), W_3)
y_hat = v_3

#****************************************************************#
```

Figure 10: Code for 2d

Here we feed our input arrays and relevant weights through the previously implemented forward function, per layer.

With this adjustment our training lop can move forward through the additional layers.

Part e

```
#************************ YOUR CODE HERE ************************#

dJ_dw3,Lambda = backward(np.vstack([np.array([1]),v_2]),v_3,W_3,Lambda)
dJ_dw2,Lambda = backward(np.vstack([np.array([1]),v_1]),v_2,W_2,Lambda)

#****************************************************************#
```

Figure 11: Code for 2e

Here we use the previously defined backward functions, we just stack them to account for our additional layers.

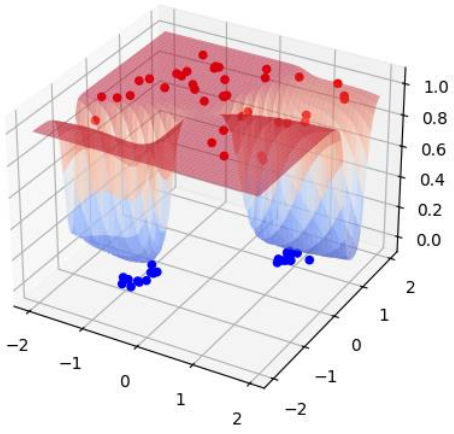

Figure 12: Result from plotting two_blob with plot_surface function

As shown above, we have successfully modelled the decision surface of the two blobs data set. You can see how it is shaped to divide the dataset at the two *blobs*.

Conclusion

The practical took us through the steps of defining and implementing a multilayer perceptron. We started with a simple linearly separable dataset and then moved on to a more complicated dataset. With the second we had to use more layers in our model to accurately define the decision surface.

This is a powerful tool that can be utilised in a multitude of areas, such as: image and speech recognition; signal processing and control systems, to name a few.