Banana Dataset

K S Ananth

Task 2

1 Overview:

Banana Dataset is a classic binary classification problem. It consists of two features and two classes that form banana-shaped clusters in a two-dimensional space. The Banana Dataset consists of 5300 input values. Each input values contains two features values which marks the x and y axis point in the plot. The output class is one of 1 or -1. In order to train a neural network to be able to predict the class of an input we can use the back propagation algorithm and the Network defined in Task-1 in which we built a whole neural network and implemented the back propagation algorithm on it. The entire walkthrough of the training process is given below:

2 Implementation Before Training:

2.1 Dataset and Dataset Plotting:

Listing 1: Dataset

```
df = pd.read_csv(r"banana.csv")
df
```

	At1	At2	Class
0	1.140	-0.114	-1
1	-1.520	-1.150	1
2	-1.050	0.720	-1
3	-0.916	0.397	1
4	-1.090	0.437	1
...	...	...	...
5295	0.335	1.390	1
5296	-1.700	-0.569	1
5297	2.640	1.140	1
5298	0.769	0.772	-1
5299	-0.255	-0.142	1

5300 rows × 3 columns

Figure 1: Dataset

Listing 2: Dataset Plotting

```
X = df[['At1', 'At2']].values
y = df['Class'].values

# Plot the dataset
plt.figure(figsize=(10, 6))
plt.scatter(X[y == -1][:, 0], X[y == -1][:, 1], color='red', label='Class -1')
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color='blue', label='Class 1')
plt.title('Banana Dataset')
plt.xlabel('At1')
plt.ylabel('At2')
plt.legend()
plt.grid(True)
plt.show()
```

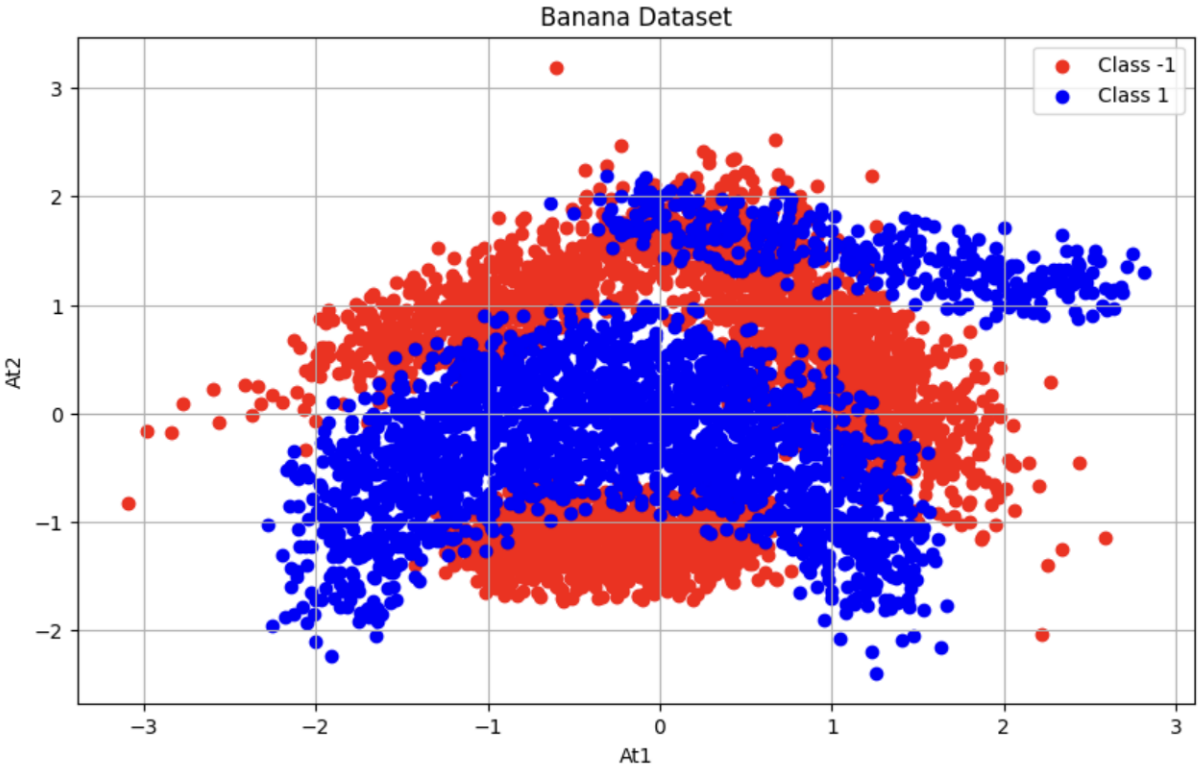

Figure 2: Dataset Plotting

2.2 Dataset Division:

Now, let us divide the dataset into 2 parts one for training the neural network and another to test the trained neural network. Let us divide the total of 5300 inputs into training set with roughly 400 inputs and the testing set having 4900 inputs. We can do this by doing this:

Listing 3: Network Class

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0,
    train_size = .076)
```

By dividing by 0.076 ratio we get 402 training dataset and the rest as test set.
The X_train ans y_train are the train set further divided into input (X_train) and the required output (y_train). Now we use X_train to train the neural network based on the values in y_train.

3 Training:

3.1 Network Declaration:

First off let us declare a neural network for the training purpose. In order to avoid over fitting I have considered a neural network with 2 inputs, 2 hidden layers with 16 neurons each and finally one output.

Listing 4: Network Declaration

```
net = Network(2, [16, 16, 1])
```

3.2 Train:

Let us use the same back propagation algorithm used in Task-1 with MSE loss function and use tanh as the activation function. Let the learning rate be 0.1. Let us run the algorithm for 40 iterations/epochs. Also, print the loss values at each iteration which can help us further analyse.

Listing 5: Back Propagation Algorithm

```
losses = []

for iteration in range(40):
    # forward pass --------------------------------------------------------------
    y_predictions = [net(temp) for temp in X_train]
    loss = sum((y_prediction - y_desired)**2 for y_desired, y_prediction in zip(
        y_train, y_predictions)) / (2*len(X_train))
    # -------------------------------------------------------------------------

    # backward pass -------------------------------------------------------------
    for temp in net.definition():
        temp.grad = 0.0

    loss.backward()
    # -------------------------------------------------------------------------

    # update --------------------------------------------------------------------
    for temp in net.definition():
        temp.value += -0.1 * temp.grad
    # -------------------------------------------------------------------------

    # Store loss for plotting
    losses.append(loss.value)

    print(f"Iteration number {iteration}: {loss.value}")
```

```
Iteration number 0: 0.7386507587059546
Iteration number 1: 0.5760726211254814
Iteration number 2: 0.46412478106760163
Iteration number 3: 0.4125167753617771
Iteration number 4: 0.39424639227409813
Iteration number 5: 0.3848857525139432
Iteration number 6: 0.4368773851614505
Iteration number 7: 0.4424846123255792
Iteration number 8: 0.39273035998351297
Iteration number 9: 0.3785546049605578
Iteration number 10: 0.351713759728523
Iteration number 11: 0.36163619983516176
Iteration number 12: 0.294348111216354
Iteration number 13: 0.3026786999461774
Iteration number 14: 0.29652721846843416
Iteration number 15: 0.2925280571392146
Iteration number 16: 0.28555508065831153
Iteration number 17: 0.2550264542725296
Iteration number 18: 0.264790439608544
Iteration number 19: 0.24814042768804617
Iteration number 20: 0.2599227809897534
Iteration number 21: 0.24199302987871574
Iteration number 22: 0.2459783266595928
Iteration number 23: 0.23509935038672192
Iteration number 24: 0.23483618997945946
Iteration number 25: 0.22755910463644888
Iteration number 26: 0.22956364565883328
Iteration number 27: 0.22089326300060483
Iteration number 28: 0.21668225410007622
Iteration number 29: 0.21540615517073883
Iteration number 30: 0.2160159975049285
Iteration number 31: 0.21592059553678616
Iteration number 32: 0.2157145628706095
Iteration number 33: 0.21351655704121078
Iteration number 34: 0.21246373927629425
Iteration number 35: 0.21012807944558426
Iteration number 36: 0.21245736196137804
Iteration number 37: 0.2092716659301765
Iteration number 38: 0.209285614809032
Iteration number 39: 0.20714552504322128
```

Figure 3: Loss Values

3.3 Plot:

Listing 6: Plot

```
# Plot epoch graph after all iterations
plt.plot(range(40), losses, color='blue')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.title('Loss per Iteration')
plt.show()
```

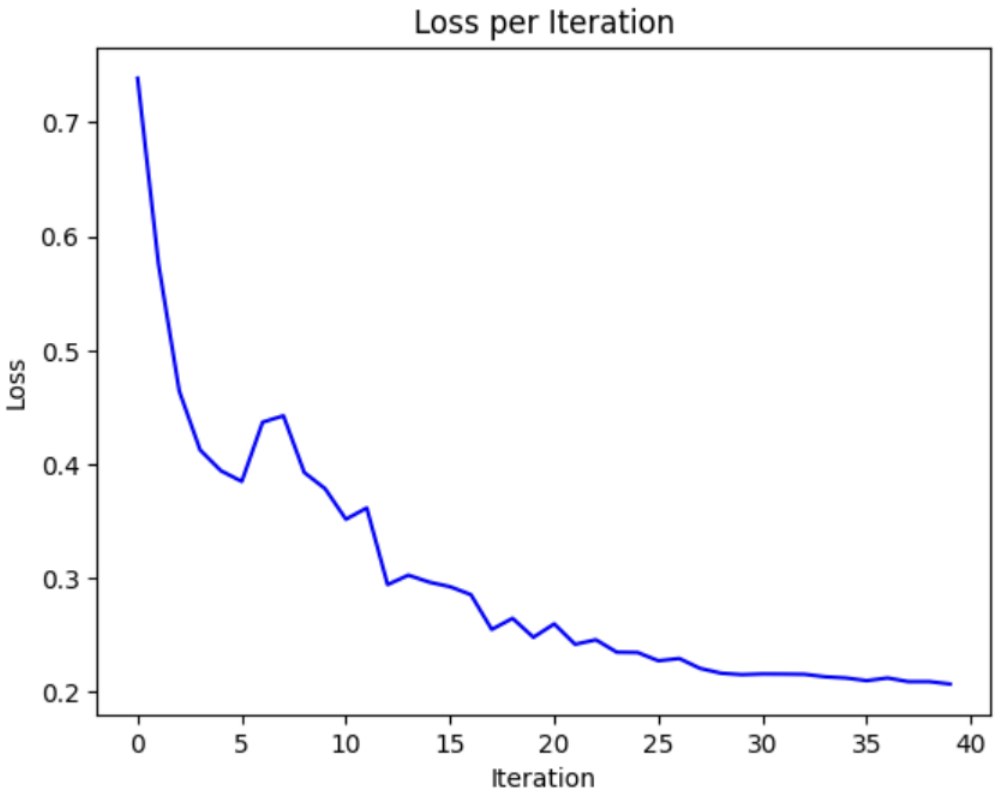

Figure 4: Loss function vs Iterations/Epochs

4 Test:

4.1 Run X_test in the trained network:

Listing 7: Run X_test

```
y_test_pred = [net(temp).value for temp in X_test]
```

4.2 Check accuracy:

Now with the obtained predictions y_test_pred we check how good is our network by checking with y_test and see how plausible our network is by calculating the accuracy.

Listing 8: Run X_test

```
def compute_accuracy(y_pred, y_true):
    # Convert the predicted probabilities to 1 or -1 based on a threshold of 0
    predictions = np.where(y_pred > 0.0, 1, -1)
    # Calculate the accuracy as the mean of correct predictions
    accuracy = np.mean(predictions == y_true)
    return accuracy

y_test_pred = np.array(y_test_pred)
test_accuracy = compute_accuracy(y_test_pred, y_test)

print(f"Test set accuracy: {test_accuracy}")
```

Output: Test set accuracy: 0.8713760718660678

Now Let us take a step further and check if our model is decent by changing the threshold to 0.3 and -0.3. If a predicted value is above 0.3 then it will be revalued as 1, if less than -0.3 then -1 and if in between then it will be revalued to 0. Then we check the accuracy for this:

Listing 9: Run X_test

```
def compute_accuracy(y_pred, y_true):
    # Convert the predicted probabilities to 1 or -1 based on a threshold of 0.3
        and -0.3
    predictions = np.where(y_pred > 0.3, 1, np.where(y_pred < -0.3, -1, 0))
    # Calculate the accuracy as the mean of correct predictions
    accuracy = np.mean(predictions == y_true)
    return accuracy

y_test_pred = np.array(y_test_pred)
test_accuracy = compute_accuracy(y_test_pred, y_test)

print(f"Test set accuracy: {test_accuracy}")
```

Output: Test set accuracy: 0.7580645161290323

5 Conclusion:

The training and testing have been successful for given 40 number of iterations/epochs yielding an accuracy of $\tilde{8}7\%$ considering 0.0 as a threshold. If we increase the number of epochs to a larger number for example 100 and more, the accuracy will be higher which has been tried and tested with the same implementation.