AI3011: Lab Assignment Report

Lab Assignment 10

Pratik Rana
U20220108

Declaration

I, Pratik, certify that this project is my own work, based on my personal study and research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this project has not previously been submitted for assessment in any academic capacity and that I have not copied in part or whole or otherwise plagiarised the work of other persons. I confirm that I have identified and declared all possible conflicts that I may have.

Dated: 10 - 04 - 2023

Pratik Rana

Answers

Q1: What are the parallels between Biological NNs and Artificial NNs? List any three.
Ans. 1. BNNs and ANNs both consist of interconnected neurons/units.
2. They involve the transmission of signals (electrical impulses in BNNs, numerical values in ANNs). between neurons.
3. Both can learn and adapt based on experience or training data.

Q2:
What was the McCulloch-Pitts Neuron model? List any two of its limitations.
Ans. The McCulloch-Pitts Neuron model was a simplified mathematical neural model of a biological neuron. Its limitations were:

1. It only supported binary(0,1) outputs, so couldn't represent complex patterns or continuous data.
2. It operated based on fixed rules without adjusting its parameters so it lacked the capability for learning/adaptation.

Q3: How does a perceptron learn?
Ans. The Perceptron learning process continues iteratively until the model converges to a satisfactory solution or reaches a predefined stopping criterion. It adjusts its weights based on the error between the predicted output and the actual output. It uses a learning rate to control the size of weight updates.

Q4: List any three activation functions and their respective output ranges.
Ans. 1. Sigmoid: 0, 1.
2. ReLU - Rectified Linear Unit : 0, ∞
3. Tanh - Hyperbolic Tangent: -1, 1.

Q5: Why was the classical perceptron not able to solve the XOR problem? Can you suggest a way to solve the XOR problem using neural networks but without going into a higher dimensional space?
Ans. The classical perceptron relies on a linear decision boundary, which cannot separate XOR data points effectively. We can use a multilayer perceptron with nonlinear activation functions allowing it to learn and represent nonlinear decision boundaries. A two-layer perceptron with one hidden layer ca be used to solve the XOR problem.

References:

1. **Lecture Slides**

Output and Code

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

In [ ]:
```
np.random.seed(0)
```

In [ ]:
```
mu1 = [2, 2]
sigma1 = [[0.9, -0.0255], [-0.0255, 0.9]]
class1_samples = np.random.multivariate_normal(mu1, sigma1, 250)

mu2 = [5, 5]
sigma2 = [[0.5, 0], [0, 0.3]]
class2_samples = np.random.multivariate_normal(mu2, sigma2, 250)

class1_labels = np.zeros(250)
class2_labels = np.ones(250)

combined_data = np.vstack((class1_samples, class2_samples))
combined_labels = np.hstack((class1_labels, class2_labels))
```

In [ ]:
```
data_with_bias = np.column_stack((np.ones(len(combined_data)), combined_d
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(data_with_bias, combi
```

In [ ]:
```
def step_activation(x):
    return 1 if x >= 0 else 0

def train_perceptron(X, y, learning_rate, max_epochs):
    weights = np.random.rand(X.shape[1])
    for epoch in range(max_epochs):
        for i in range(X.shape[0]):
            prediction = step_activation(np.dot(X[i], weights))
            error = y[i] - prediction
            weights += learning_rate * error * X[i]
    return weights
```

In [ ]:
```
learning_rate = 0.1
max_epochs = 1000
optimal_weights = train_perceptron(X_train, y_train, learning_rate, max_e
```

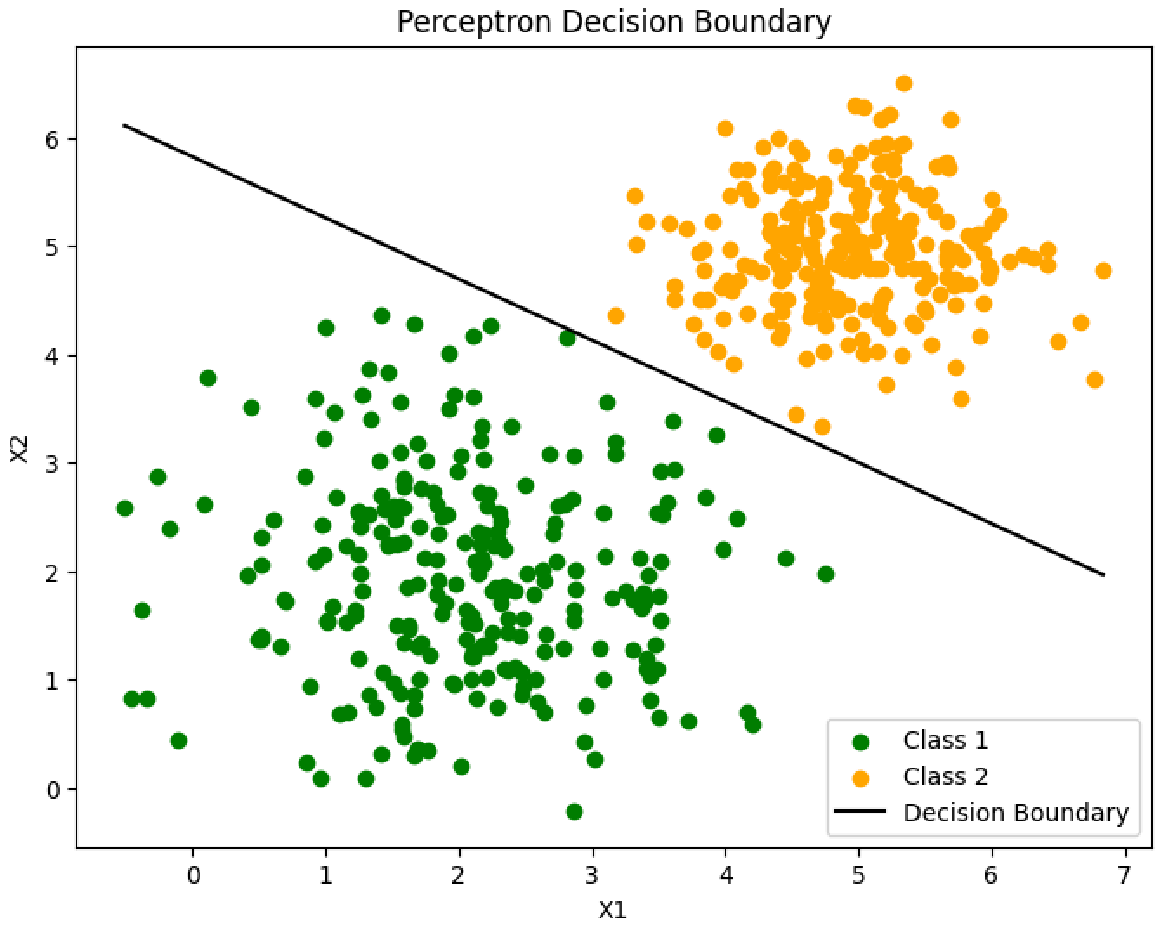
In [ ]:
```
plt.figure(figsize=(8, 6))
plt.scatter(class1_samples[:, 0], class1_samples[:, 1], color='green', la
plt.scatter(class2_samples[:, 0], class2_samples[:, 1], color='orange', l

x_vals = np.array([np.min(combined_data[:, 0]), np.max(combined_data[:, 0
y_vals = -(optimal_weights[0] + optimal_weights[1] * x_vals) / optimal_we
plt.plot(x_vals, y_vals, color='black', label='Decision Boundary')

plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Perceptron Decision Boundary')
plt.legend()
plt.show()

y_pred = [step_activation(np.dot(x, optimal_weights)) for x in X_test]
```

```
ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels=[
```



Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x73d8212f4c40>