Single-layer Neural Network (Perceptrons)

> Assignment 3

1. **Using the dataset and code provided in the lecture material, modify the code to print the number of wrongly predicted training examples for each of the 5 epochs. What do these numbers suggest about the performance of the perceptron learning with each epoch?**

```python
    def backward(self, x, y):
        """
        Computes the errors
        Used to determine the updates necessary for the weights and bias
        :param x:
        :param y:
        :return:
        """
        predictions = self.forward(x)
        # difference between true class labels and predictions
        errors = y - predictions
        return errors

    def train(self, x, y, epochs):
        for e in range(epochs):
            wrong_predictions = 0
            for i in range(y.shape[0]):
                errors = self.backward(x[i].reshape(1, self.num_features), y[i]).reshape(-1)
                self.weights += (errors * x[i]).reshape(self.num_features, 1)
                self.bias += errors

                wrong_predictions += np.count_nonzero(errors)
            print(f"Epoch {e + 1}: {wrong_predictions} wrong predictions")


    def evaluate(self, x, y):
        predictions = self.forward(x).reshape(-1)
        # total number of matched labels / number of samples
        accuracy = np.sum(predictions == y) / y.shape[0]
        return accuracy
✓ [11] < 10 ms
```

Training the Perceptron

```
ppn = Perceptron(num_features=2)

ppn.train(X_train, y_train, epochs=5)

print('Model parameters:\n\n')
print('  Weights: %s\n' % ppn.weights)
print('  Bias: %s\n' % ppn.bias)
```

✓ [12] < 10 ms

```
Epoch 1: 3 wrong predictions
Epoch 2: 0 wrong predictions
Epoch 3: 0 wrong predictions
Epoch 4: 0 wrong predictions
Epoch 5: 0 wrong predictions
Model parameters:


  Weights: [[1.27340847]
 [1.34642288]]

  Bias: [-1.]
```

At the start, the perceptron is untrained, therefore the no. of wrong predictions is high. As the training progresses, the model adjusts the weights & bias after encountering errors during each training example.

Epoch wise performance improvement shows that the perceptron is effectively learning from the data. Starting from 2nd epoch with 0 wrong predictions, the perceptron shows that the training dataset is linearly separable.

2. **Using the dataset and code provided in the lecture material, modify the code to not shuffle the data. Print the number of wrongly predicted training examples for up to 10 epochs. Do you notice a difference in performance (compared to Question 1 above)? If yes, what may have caused this?**

```
##########################
### DATASET
##########################

# load the dataset to numpy
data = np.genfromtxt('perceptron_toydata.txt', delimiter='\t')
X, y = data[:, :2], data[:, 2]  # X is matrix and y is class labeled array
y = y.astype(int)

print('Class label counts:', np.bincount(y))
print('X.shape:', X.shape)
print('y.shape:', y.shape)

# Shuffling & train/test split
shuffle_idx = np.arange(y.shape[0])  # generate indexes from 0 to 99
# shuffle_rng = np.random.RandomState(123)
# shuffle_rng.shuffle(shuffle_idx)
# X, y = X[shuffle_idx], y[shuffle_idx]   # X and y is shuffled based on random order of shuffle_idx

X_train, X_test = X[shuffle_idx[:70]], X[shuffle_idx[70:]]
y_train, y_test = y[shuffle_idx[:70]], y[shuffle_idx[70:]]

# Normalize (mean zero, unit variance)
# Recommended to improve the performance of the machine learning models
mu, sigma = X_train.mean(axis=0), X_train.std(axis=0)
X_train = (X_train - mu) / sigma
X_test = (X_test - mu) / sigma
✓ [13] < 10 ms
```

```
Class label counts: [50 50]
X.shape: (100, 2)
y.shape: (100,)
```

✓ [16] < 10 ms

Training the Perceptron

```
ppn = Perceptron(num_features=2)

ppn.train(X_train, y_train, epochs=10)

print('Model parameters:\n\n')
print('  Weights: %s\n' % ppn.weights)
print('  Bias: %s\n' % ppn.bias)
```

✓ [17] 10ms

```
Epoch 1: 1 wrong predictions
Epoch 2: 4 wrong predictions
Epoch 3: 1 wrong predictions
Epoch 4: 0 wrong predictions
Epoch 5: 0 wrong predictions
Epoch 6: 0 wrong predictions
Epoch 7: 0 wrong predictions
Epoch 8: 0 wrong predictions
Epoch 9: 0 wrong predictions
Epoch 10: 0 wrong predictions
Model parameters:
```

Yes, there is a significant performance difference.

Since the data is not shuffled, the perceptron's weight update has become biased early during the training for `class 0` (since they are the ones encountered first)

That's why we see a spike in **epoch 2**. The biased weights from epoch 1 continue to misclassify

`class 1` samples cause class 0 samples dominate the initial portion of the training data.

The perceptron may now try to resolve the decision boundary by adjusting weights for `class 1` , which may now over-correct, leading to mis-classification for previously correctly classified `class 0`. That's why we see `1` error in **epoch 3.**

The errors finally reduces to 0 post initial fluctuation and the perceptron is able to linearly separate the data.

3. **Consider dataset `Data` generated using the python code below**

```
import numpy as np

a1 = np.random.uniform(4, 6, [200, 2])
a2 = np.random.uniform(0, 10, [200, 2])

Data_X = np.vstack((a1, a2))
Data_Y = np.hstack((np.ones(200).T, np.zeros(200).T)).astype(int
```

Answer the following

a. **Visualise (using matplotlib) the above data and comment on what accuracy you expect the perceptron learning algorithm to result in. Explain your reason.**

b. **Using the first 150 samples from class 1 and the first 150 samples from class 0 for training, and the rest of the samples for testing, train the perceptron for 10 epochs. What accuracy do you get on training data? What accuracy do you get on testing data?**

c. **Do you expect a model to perform well on testing data, if it does not perform well on training data? Explain your reason.**

---

a.

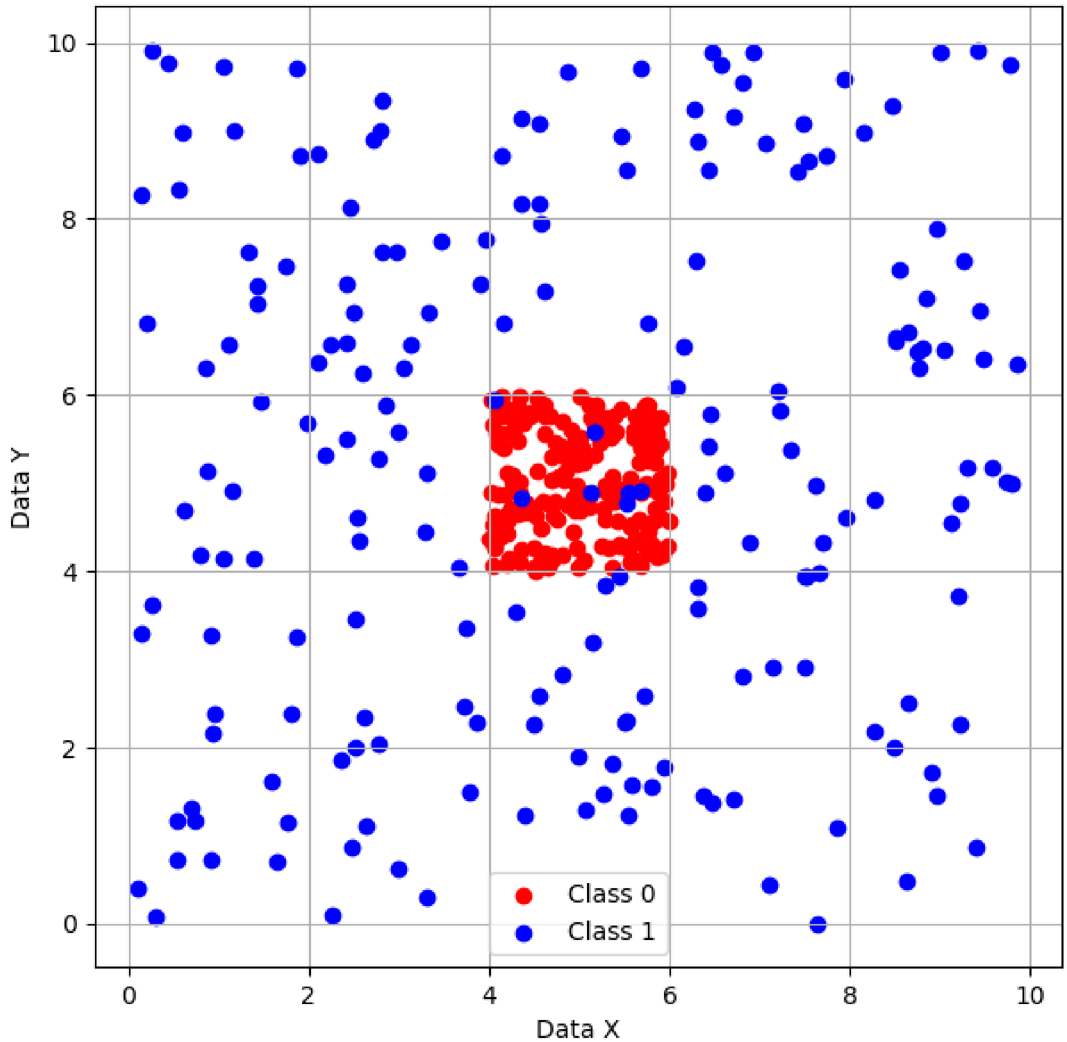
```
import numpy as np
import matplotlib.pyplot as plt

a1 = np.random.uniform(4, 6, [200, 2])
a2 = np.random.uniform(0, 10, [200, 2])

Data_X = np.vstack((a1, a2))
Data_Y = np.hstack((np.ones(200).T, np.zeros(200).T)).astype(int

# Q1
plt.figure(figsize=(7, 7))
plt.scatter(a1[:, 0], a1[:, 1], c='red', label='Class 0')
plt.scatter(a2[:, 0], a2[:, 1], c='blue', label='Class 1')

plt.xlabel('Data X')
plt.ylabel('Data Y')
plt.legend()
plt.grid(True)
plt.show()
```



Given,

- `a1` and `a2` are generated as **random uniform distributions** over certain ranges:
  - `a1` clusters around `[4, 6]` (Class 0).
  - `a2` spreads from `[0, 10]` (Class 1).

From the ranges, we can infer that data overlaps with each other making it non-linearly separable. Therefore the perceptron will never achieve 100% accuracy (since it's a linear classifier).

The accuracy of the perceptron highly depends on the characteristic of the dataset as it is a linear classifier. That being said theoretically we could expect the accuracy around 50 % with some amount of deviation (as we dealing with random samples).

This is because the mid point of the ranges is `5` and we could say the number of points between is evenly distributed for each class and perceptron being a linear classifier, would classify 50 % of samples correctly while mis-classifying the other half. (assume a line divides a box into equal halfs.)

---

b.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score

a1 = np.random.uniform(4, 6, [200, 2])
a2 = np.random.uniform(0, 10, [200, 2])

# Q2

# use 150 training samples
train_X = np.vstack((a1[:150], a2[:150]))
train_Y = np.hstack((np.ones(150).T, np.zeros(150).T)).astype(in

# use 50 samples for testing
text_X = np.vstack((a1[150:], a2[150:]))
text_Y = np.hstack((np.ones(50).T, np.zeros(50).T)).astype(int)

perceptron = Perceptron(max_iter=10)  # 10 epochs
perceptron.fit(train_X, train_Y)

train_predictions = perceptron.predict(train_X)
train_accuracy = accuracy_score(train_Y, train_predictions)
```

```
print("train_accuracy: ", train_accuracy)

test_predictions = perceptron.predict(text_X)
test_accuracy = accuracy_score(text_Y, test_predictions)
print("test_accuracy: ", test_accuracy)
```

```
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+ python L03_perceptron/homework/q3.py
train_accuracy:  0.5
test_accuracy:  0.5
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+ python L03_perceptron/homework/q3.py
/Users/neilharia7/PycharmProjects/stat479-deep-learning-ss19/venv/lib/python3.12/site-packages/sklearn/linear_model/_stochastic_gradient.py:738: ConvergenceWarning: Maximum num
ber of iteration reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
train_accuracy:  0.5
test_accuracy:  0.5
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+ python L03_perceptron/homework/q3.py
train_accuracy:  0.37333333333333335
test_accuracy:  0.35
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+ python L03_perceptron/homework/q3.py
train_accuracy:  0.5533333333333333
test_accuracy:  0.58
(venv) neilharia7 ~/PycharmProjects/stat479-deep-learning-ss19 master ±+
```

As expected the accuracy is around 50 % with some deviation for both training and test dataset.

---

c.

No, the model is not expected to perform well on the testing data if it does not perform well on the training data. This is because, when the model is trained on the training data, it tries to learn patterns, decision boundaries, etc. from this data. If it fails to perform well on the training data, it suggests that the model is unable to properly learn the underlying patterns in the training data.

As the test data is unseen by the model during training, (assuming we have a balanced distribution of training and testing samples, i.e. similar ratio of both classes in this case) a poorly performing model on the training data will likely perform as bad on the testing data (as can be seen from the results above).