Encoding and Classifier

- The classical data was embedded in two ways in the form of amplitude embedding and angle embedding.
- In the amplitude-embedding technique, data is encoded into the amplitudes of a quantum state. A normalized classical ***N-dimensional*** data point ***x*** is represented by the amplitudes of a ***n-qubit quantum state.***

  $$|\Psi > \ = \sum x_i |i >$$

  Where i is the quantum basis.
- Angle embedding, Encodes N features into the rotation angles of n qubits, where N≤n. The rotations can be chosen as either ***RX, RY or RZ*** gates, as defined by the rotation.
- The Embeddings were done using the Pennylane library in python.

Amplitude Embedding

- Amplitude Embedding needs only two qubits to encode the classical data.
- For the Variation Quantum Classifier, first we implemented one layer of ***RX,RZ and CNOT*** gates with ***3*** layers which gave a ***test accuracy of 0.2*** and ***highest train accuracy of 0.4.***
- We considered batch size of 5 and ***NesterovMomentumOptimizer*** as our optimizer
- Then I changed the ansatz into implementing **RX,RY and RZ and a CNOT** gate which with ***3*** layers gave better training results (Some iterations image):

```
Iter:     1 | Cost: 3.4104577 | Accuracy: 0.4366667
Iter:     2 | Cost: 2.5536059 | Accuracy: 0.5233333
Iter:     3 | Cost: 1.5775721 | Accuracy: 0.4233333
Iter:     4 | Cost: 2.3110371 | Accuracy: 0.5233333
Iter:     5 | Cost: 1.8661167 | Accuracy: 0.4233333
Iter:     6 | Cost: 1.2871659 | Accuracy: 0.4733333
Iter:     7 | Cost: 2.6443048 | Accuracy: 0.4800000
Iter:     8 | Cost: 1.1493262 | Accuracy: 0.5233333
Iter:     9 | Cost: 1.3527157 | Accuracy: 0.6733333
Iter:    10 | Cost: 1.7157764 | Accuracy: 0.3800000
```

- Next Considering the number of layers as ***4*** gave:

```
Iter:     1 | Cost: 1.3446573 | Accuracy: 0.5700000
Iter:     2 | Cost: 1.8204442 | Accuracy: 0.5233333
Iter:     3 | Cost: 1.4120008 | Accuracy: 0.4233333
Iter:     4 | Cost: 1.4381492 | Accuracy: 0.4333333
Iter:     5 | Cost: 2.7283498 | Accuracy: 0.1733333
Iter:     6 | Cost: 1.1202866 | Accuracy: 0.6600000
Iter:     7 | Cost: 0.9492884 | Accuracy: 0.5733333
Iter:     8 | Cost: 1.3184546 | Accuracy: 0.3733333
Iter:     9 | Cost: 1.9484108 | Accuracy: 0.4266667
Iter:    10 | Cost: 0.8060176 | Accuracy: 0.6566667
```

- We also got a testing accuracy of ***0.74***

```
# Compute accuracy
predictions = [np.sign(variational_classifier(weights, bias, x)) for x in X_test]
acc = accuracy(Y_test, predictions)

print(
    " Cost: {:0.7f} | Accuracy: {:0.7f} ".format( cost(weights, bias, X, Y), acc)
)
```

```
Cost: 0.8060176 | Accuracy: 0.7416667
```

- Next ***5*** layers was considered:

```
Iter:     1 | Cost: 1.2976096 | Accuracy: 0.5633333
Iter:     2 | Cost: 1.7779056 | Accuracy: 0.3333333
Iter:     3 | Cost: 0.6407656 | Accuracy: 0.7933333
Iter:     4 | Cost: 1.7630178 | Accuracy: 0.2300000
Iter:     5 | Cost: 1.7932884 | Accuracy: 0.5633333
Iter:     6 | Cost: 1.2295164 | Accuracy: 0.3466667
Iter:     7 | Cost: 1.7967084 | Accuracy: 0.2033333
Iter:     8 | Cost: 1.3215029 | Accuracy: 0.4233333
Iter:     9 | Cost: 1.4793313 | Accuracy: 0.3133333
Iter:    10 | Cost: 0.9544546 | Accuracy: 0.7400000
```

- It gave a test accuracy of ***0.833*** the highest test accuracy.
- Considering number of layers as ***six*** gave us :

```
Iter:      1 | Cost: 1.5990938 | Accuracy: 0.4266667
Iter:      2 | Cost: 1.9411440 | Accuracy: 0.3300000
Iter:      3 | Cost: 1.5537574 | Accuracy: 0.4500000
Iter:      4 | Cost: 2.0847227 | Accuracy: 0.5233333
Iter:      5 | Cost: 1.7075924 | Accuracy: 0.5233333
Iter:      6 | Cost: 1.2553577 | Accuracy: 0.3633333
Iter:      7 | Cost: 1.0514950 | Accuracy: 0.5766667
Iter:      8 | Cost: 1.4040993 | Accuracy: 0.1900000
Iter:      9 | Cost: 1.6749637 | Accuracy: 0.3066667
Iter:     10 | Cost: 2.3157339 | Accuracy: 0.4233333
```

- It gave a test accuracy of ***0.4*** which is significantly lower than that of five layers.
- Also considered ***seven*** layers which gave:

```
Iter:      1 | Cost: 0.8171923 | Accuracy: 0.6666667
Iter:      2 | Cost: 3.2291236 | Accuracy: 0.5233333
Iter:      3 | Cost: 0.9597792 | Accuracy: 0.7466667
Iter:      4 | Cost: 1.5642177 | Accuracy: 0.4766667
Iter:      5 | Cost: 1.7429029 | Accuracy: 0.5766667
Iter:      6 | Cost: 2.2471905 | Accuracy: 0.4366667
Iter:      7 | Cost: 1.2662306 | Accuracy: 0.3866667
Iter:      8 | Cost: 1.9547518 | Accuracy: 0.1733333
Iter:      9 | Cost: 1.5926295 | Accuracy: 0.1966667
Iter:     10 | Cost: 1.3634367 | Accuracy: 0.4766667
```

- The test accuracy was ***0.5177*** which is better than six layers but worse than five layers

Angle Embedding

- For angle embedding , we need four qubits to encode the classical data
- We considered batch size of 5 and ***NesterovMomentumOptimizer*** as our optimizer
- The ansatz considered was **RX,RY and RZ and a CNOT** gate which with ***3*** layers gave training results as (Some iterations image):

```
Iter:     1 | Cost: 1.5809759 | Accuracy: 0.4766667
Iter:     2 | Cost: 1.4952856 | Accuracy: 0.5233333
Iter:     3 | Cost: 1.0321662 | Accuracy: 0.4766667
Iter:     4 | Cost: 1.1538944 | Accuracy: 0.5233333
Iter:     5 | Cost: 2.5349856 | Accuracy: 0.4766667
Iter:     6 | Cost: 1.3149044 | Accuracy: 0.4766667
Iter:     7 | Cost: 1.1260962 | Accuracy: 0.5233333
Iter:     8 | Cost: 0.9981576 | Accuracy: 0.5233333
Iter:     9 | Cost: 1.3833043 | Accuracy: 0.5233333
Iter:    10 | Cost: 2.4941570 | Accuracy: 0.4766667
```

- Next Considering the number of layers as ***4*** gave:

```
Iter:     1 | Cost: 1.1282630 | Accuracy: 0.5233333
Iter:     2 | Cost: 1.1214062 | Accuracy: 0.5233333
Iter:     3 | Cost: 1.2046277 | Accuracy: 0.5233333
Iter:     4 | Cost: 1.2467489 | Accuracy: 0.5233333
Iter:     5 | Cost: 1.5495831 | Accuracy: 0.4766667
Iter:     6 | Cost: 1.0998144 | Accuracy: 0.5233333
Iter:     7 | Cost: 1.4051457 | Accuracy: 0.5233333
Iter:     8 | Cost: 1.5609719 | Accuracy: 0.5233333
Iter:     9 | Cost: 1.4117611 | Accuracy: 0.4766667
Iter:    10 | Cost: 1.0215238 | Accuracy: 0.5233333
```

- It gave a test accuracy of ***0.4833.***

- Next Considering the number of layers as ***5*** gave:

```
Iter:     1 | Cost: 1.0045051 | Accuracy: 0.5233333
Iter:     2 | Cost: 1.2125953 | Accuracy: 0.4766667
Iter:     3 | Cost: 2.6445697 | Accuracy: 0.5233333
Iter:     4 | Cost: 1.8856087 | Accuracy: 0.5233333
Iter:     5 | Cost: 1.8997313 | Accuracy: 0.4766667
Iter:     6 | Cost: 1.0875426 | Accuracy: 0.5233333
Iter:     7 | Cost: 1.0829376 | Accuracy: 0.4766667
Iter:     8 | Cost: 1.6559267 | Accuracy: 0.4766667
Iter:     9 | Cost: 1.1080120 | Accuracy: 0.4766667
Iter:    10 | Cost: 1.4331608 | Accuracy: 0.5233333
```

- It gave a test accuracy of ***0.4677.***

- Next Considering the number of layers as ***6*** gave:

```
Iter:     1 | Cost: 1.7537068 | Accuracy: 0.4766667
Iter:     2 | Cost: 1.1955090 | Accuracy: 0.5233333
Iter:     3 | Cost: 2.3393624 | Accuracy: 0.4766667
Iter:     4 | Cost: 1.0095663 | Accuracy: 0.4766667
Iter:     5 | Cost: 1.3053505 | Accuracy: 0.4766667
Iter:     6 | Cost: 1.1883827 | Accuracy: 0.4766667
Iter:     7 | Cost: 1.8977823 | Accuracy: 0.5233333
Iter:     8 | Cost: 1.3684752 | Accuracy: 0.4766667
Iter:     9 | Cost: 1.6650727 | Accuracy: 0.4766667
Iter:    10 | Cost: 1.0005680 | Accuracy: 0.5233333
```

- It gave a test accuracy of ***0.4833.***

ANALYSIS:

- Amplitude embedding needs only two qubits to encode the data as compared to four for angle embedding. Since near-term quantum computers only have a limited number of qubits, amplitude embedding is a better option.
- Other encoding such as kernel, basis were considered and researched on. Since basis encoding needs a lot of qubits (classical data has values in 10000's which would need 14 qubits to embed) while the kernel method was a bit too complex and hence I went with angle and amplitude embedding which were relatively better and simpler.
- Amplitude embedding for the chosen ansatz gave better results and higher accuracy. The ansatz chosen for angle embedding saturated in terms of accuracy and couldn't improve significantly even when layers were increased which shows that the chosen ansatz is not a good one
- For Amplitude embedding, for the chosen ansatz ***five layers gave the best results*** with a ***test accuracy of 0.833*** and also gave the best training accuracy too.
- For Angle embedding, almost all layers gave similar results with training accuracy saturating at ***0.5233***. There were slight variations for the given ansatz with different number of layers and the best result came for four layers which gave a test accuracy of ***0..4833***
- The ansatz chosen was simple and hence the training and test results were not very impressive. A better ansatz involving further more complex gates and entanglements would hopefully give better results.