# INF367 Mandatory 2

Christoffer Slettebø Simon Vedaa Khalil Ibrahim

## Data exploration and pre-processing:

- We found out that we have the following dimension: 150,4 150 samples with 4 datapoints each
- We have splitted the data into 70% training data, 15% validation data and 15% test data We then have 105 samples in the training data. We have in the training data samples 38, 34 and 33 of the respective target values 0, 1 and 2. That will say we have a balanced dataset
- The feature value range is from 0.1 to 7.9 so we had to normalize. We did so and scaled it between 0 and pi with sklearn minmaxscaler before we encode it.
- We chose to use angle encoding, and therefore we scaled the data so every data point is larger or equal to zero and smaller or equal to pi.

## Circuits and models

Each model is derived from the `BaseModel` class. The `BaseModel` class implements methods for fitting data, prediction, gradients and measuring circuits. All our circuits are only using 4 qubits. Each model implements its own weight initialization, and its corresponding circuit.

### Circuit/Model 1: Quantum Convolutional Neural Network

The circuit has its inspirations from a traditional neural network. The unitary gates is somewhat simple, we could've used many more gates to make the model more complex, but since we don't have much experience we went with "simple" unitary gates.

Under the project we have tried to change between rz, rx, ry in this circuit, even tried to use u3 gates. The gates we think work, we get an ok result. The changes in gates didn't feel like it impacted that much since the bigger elephant in the room was the parameters.

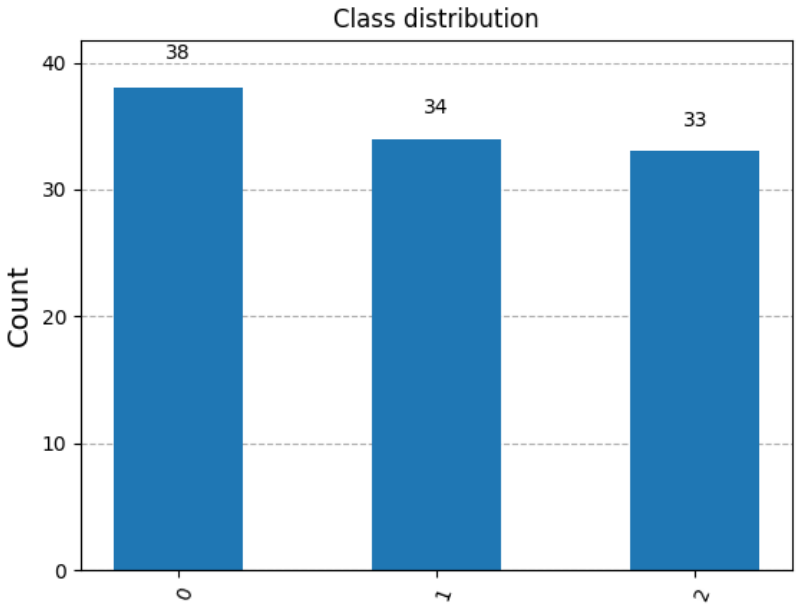


Figure 1: Class distribution of training data

Parameters start with random values from 0 to 2pi. Parameters together with epsilon value and learning rate seem hard to match. It looked like if it had a good start was very important, because the training got always stuck at some point. The model could wary between 20-70% accuracy without changes to the circuit. Later we found out that the learning rate and epsilon need to be in a certain area to be a bit more consistent.

We were not able to get this model to hit consistently 96% test accuracy, but it will often hit 85%, which we think is good.

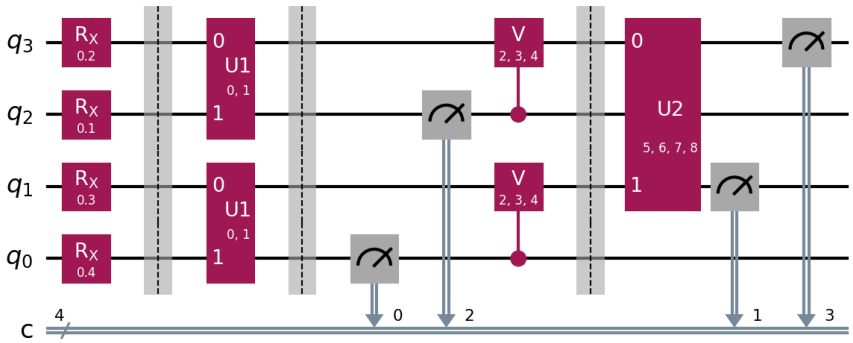


Figure 2: Circuit 1

## Circuit/Model 2: Real Amplitudes

The second circuit is the implementation of IBM's Real Amplitudes. This uses a rotational encoding by Rx gates, and each subsequent layer contains Ry gates and entanglements(cx gates), followed up by full measurement. We thought it would be interesting to see how this circuit compared to our own circuits and use it as some sort of baseline.

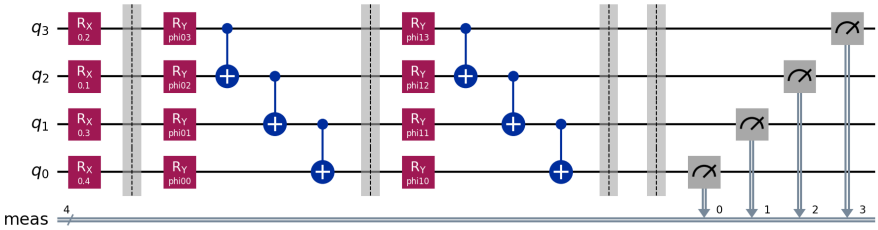


Figure 3: Circuit 2: Real Amplitudes

## Circuit/Model 3:

In this circuit we have created a variation of the "Real Amplitudes" ansatz with several customizations to enhance data representation and entanglement. We use Hadamard gates and $R_z$ rotations for feature encoding on each qubit, ensuring that input data is represented in the quantum states. In each layer, We have added both $R_x$ and $R_y$ rotations with adjustable parameters on each qubit, providing increased flexibility. Additionally, we expanded the entanglement pattern by including multiple CX gates between adjacent qubits, as well as an extra CX gate between qubits 1 and 2 in each layer. These modifications make the circuit more expressive and better suited to capture complex patterns in the data.

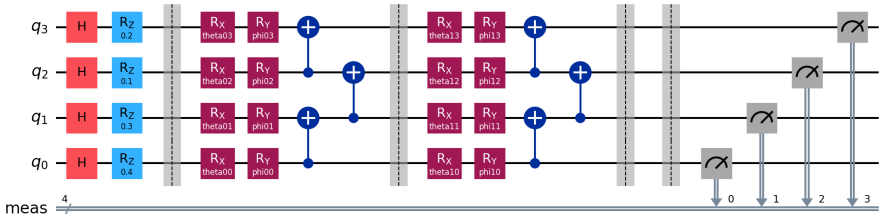


Figure 4: Circuit 3

## Measurement and Circuit output

When measuring, we apply the modulus operator on the number of classes. Then sum up for each class and divide by the number of shots to get the probability for each class. That becomes the output of the model.

# Training

## Loss function

We are using cross-entropy loss(log_loss) as we are solving a classification problem.

## Gradient Descent

Our gradient descent function is just a standard stochastic gradient descent using the finite difference method. For each gradient calculation we use 1000 shots.

## Early stopping

We implemented early stopping to prevent overfitting and save computation time by stopping training when validation loss improvement stagnates. The mechanism uses a patience counter to allow minor fluctuations, incrementing if the validation loss doesn't improve by min_delta within a set number of epochs. If the loss worsens by more than max_delta, training stops immediately.

## Model training and selection

The training process is rather simple. We define some model and hyperparameter combinations, and train each combination for a maximum of 20 epochs. Early stopping was set with a patience of 2. For selection, we choose the model with the highest validation accuracy.

## Results

Chosen parameters:

| Model | Learning Rate | epsilon | layers |
|---|---|---|---|
| Model2 | 1.2 | 0.7 | 2 |

Performance(accuracy):

| Training | Validation | Test |
|---|---|---|
| 23.81% | 59.09% | 43.48% |

The chosen model does not perform very well. It seems that we may have been too strict with our early stopping, and that the model could have trained for longer.

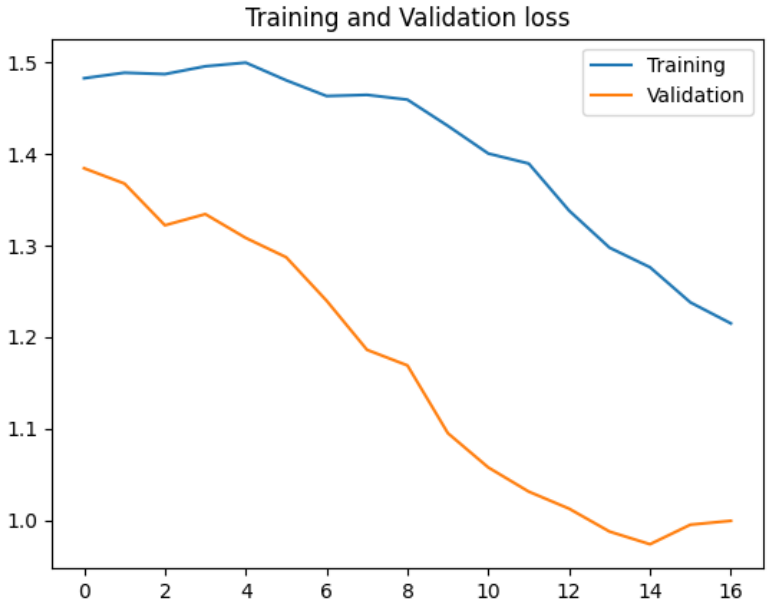


Figure 5: Loss

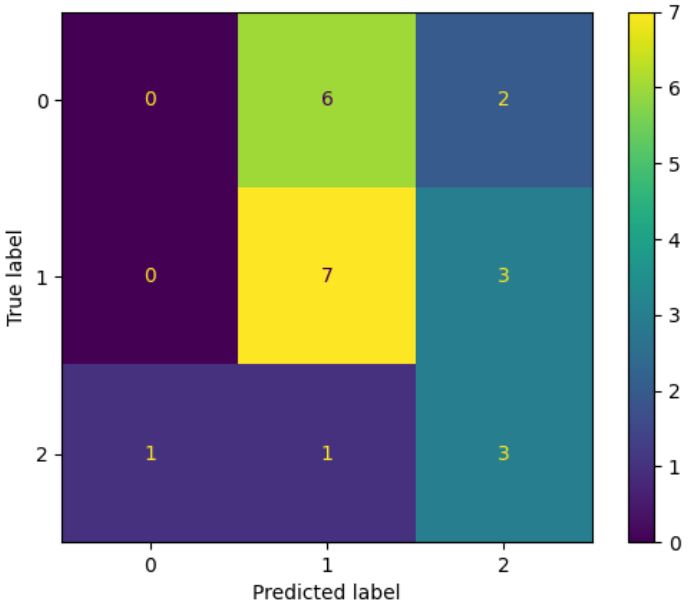


Figure 6: Confusion matrix

## Observations and Conclusion

There are a few observations which affect our results:

- The dataset is small, so it is easy to overfit to training data. Additionally, if the model underfits it sometimes gets good validation results as the sample size is small.
- Long training times due to gradient calculations, as we need to calculate the loss $2n$ times the number of parameters a models has.
- The combination of learning rate and epsilon had a big impact on model performance. Do small changes in either of them had widely different results.
- Due to the nature of quantum circuits, we sometimes got very different results even when using the same hyperparameters.