Using a Generative Modeling Approach to a Variational Circuit Problem

A brief introduction to the task given to me was to train a variational circuit to give a 4-qubit output state from a random 4-qubit input state. I completed this task using Qiskit's Circuit Library to create and test the variational circuit as a quantum neural network. Through executing the job on a QASM Simulator, it allowed me to get results that should be expected on NISQ devices.

Model Testing

There were many possible approaches I could have taken this problem. After iterative testing with many quantum neural networks, I came to the conclusion that my selection performed the best.

OpflowQNN from Qiskit

My several circuits were trained on Qiskit's already created quantum neural networks with classification. It would be disrespectful for me to say that any of the classifiers created by the Qiskit Team and the Qiskit community aren't effective, but for my problem, they did not seem to produce the results I was hoping for. One realization I came to when training this classifier is that I realized that it could not solve classification problems with multiple classes. I came to this after help from a newly started community: *EntangledQuery.com*. This led me to my next set of tests.

CircuitQNN from Qiskit

The added benefit from this quantum neural network circuit was that I could specify the number of classes through the additional parameter: *output_shape*. I was very happy to see that I could train a variational circuit to have an output supporting multiple classes because that was the requirement of the task. However, after several days of testing, I noticed that the training time was taking too long, several hours in fact. Even though I tuned the hyperparameters to as low as I thought should be allowed, this was still causing a major problem. Some tests were done through switching between different feature maps like RealAmplitudes and TwoLocal. Other tuning was through changing the maxiter and other parameters of the feature map and the ansatz function. I had to find a solution to decrease training time and have an acceptable overall accuracy.

Custom Quantum Neural Network

My next testing process was through creating a custom quantum neural network. It allowed for much faster training times because I could see all the different parts of the QNN: feature map, variational circuit, loss/cost function, gradient, etc. More details are in the Jupyter Notebook. Through more testing of creating a custom quantum neural network, I believe a variational circuit could be trained the have significantly better accuracy than the one I created.

Final Approach

Before I continue to my last approach and the conclusions I had drawn from it, briefly, I would like to mention that I never came to a satisfactory loss value or accuracy value. However, there are several reasons for why that happened which I will go over later.

To begin with, while understanding *A generative modeling approach for benchmark and training shallow quantum circuits*, I came across an approach that I wanted to take to train and test my model for. After applying X gates for the desired input state, the first step was GHZ state preparation, which entangles all the qubits in a superposition state. This is easily done with a Hadamard gate and CNOT gates. Next there were two types of layers used, the first was ion trapping and the second was the cost layer. Ion trapping brings full connectivity among the qubits, allowing for many different circuit layouts. While I didn't test on different circuit layouts, this is an added benefit of using ion trapping. The basic premise of the ion trap setting was to perform single qubit rotations, namely, R_z, R_x, then R_z again on each qubit for each layer. It was also discussed that the rotations could be done in the reverse, namely, R_x, R_z, R_x, however, this was advised against because it did not result in an effective reduction of number of parameters, and it is more convenient to use more R_z instead of R_x rotations. The other layer used was a cost layer chosen from the set of cost functions used in the *Supplementary Material*: *clipped negative log-likelihood*, *earth mover's distance*, and *moment matching*. I will continue to specify each of these cost functions as A, B, and C, respectively. Lastly, after the number of layers specified was performed, measurement to classical bits was done to get an output state.

In one attempt of training this variational circuit, I used the B cost function out of random decision, since it was mentioned that all the cost functions performed equally as well. Training the circuit was attempted using a cross-entropy loss function with gradient descent to update θ.

Due to my circumstances and time constraints, I could not train the model on the amount of training data, layer repetitions, and epochs I would have liked to get optimal results. The trained model used 40 training datapoints, 20 epochs, and 2 layer repetitions. This was done to complete training by the deadline, otherwise, more epochs and training points would have been used. Without these constraints, I would assume better accuracy and lower loss would be achieved.

References

1. Benedetti, M., Garcia-Pintos, D., Perdomo, O. *et al*. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Inf* 5, 45 (2019). https://doi.org/10.1038/s41534-019-0157-8