Predicting Quantum Advantage by Quantum Walk with Convolutional Neural Networks

Sajeda Mokbel, Noah Bolger, Haoming Yuan, Jake Malliaros

PHYS 490 - Machine Learning for the Physical Sciences
Taught by Dr. Pooya Ronagh
Department of Physics and Astronomy
University of Waterloo
Waterloo, ON
9th April 2020

Contents

1 Introduction

Predicting Quantum Advantage by Quantum Random Walk with Convolutional Neural Networks by Alexey Melnikov et. al. explores the use of convolutional neural networks in determining whether a quantum random walk on a graph will provide a speed advantage over classical random walk on the same graph. Conventionally, this task involves simulating classical and quantum walk on each graph in order to classify the graph as either "quantum" or "classical". However, this task becomes computationally intensive as graphs increase in size. The simulation method also fails to provide insight into the reason why quantum speedup is occurring on the particular graph. This paper details the Convolutional Neural Network described in the paper, as well as our attempt to re-create the results presented. Through our re-creation, we implemented discrete and continuous quantum and classical walks, and developed a convolutional neural network based on the description from the Methods section of the paper. Our methods will be outlined and contrasted with those from the paper.

2 Quantum and Classical Walk Implementations

2.1 Discrete Quantum and Classical Walks

Initially, we did not perform a continuous classical and quantum walk like in the paper as it was believed discrete walks were more easily implemented. For the discrete walks, the total number of steps for the start node v_s to reach the end node v_t will be the metric in determining which walk is faster. The classical walk was implemented by converting the rows of the input adjacency matrix A to a Markov chain. Each row was normalized in order to create a uniform distribution. At the row of the current index of the walk (initially at the row corresponding to the starting vertex), the row is sampled. The element in the row that was sampled represents the next vertex that the walk transitions to. The number of transitions required to reach the end vertex is output to be compared. For the quantum walk, we used and modified an open repository of a simulation of quantum walk on graphs (https://github.com/ctamon/quantum-walk-on-graphs) to perform our discrete quantum walk. This begins by initializing vertex amplitudes and probabilities for all vertices in the graph. Then performing spectral decomposition on the input adjacency matrix A to acquire the eigenvalues and eigenvectors. Next a unitary U is constructed using e^{-itA}. Given U we can recalculate the amplitudes and probabilities to sample the target vertex v_t. We repeat this iterate this process and track the number of transitions before reaching the target vertex v_t. The steps it takes for both the discrete classical and quantum cases are compared. The graph is labeled classical or quantum when the number of steps in one of the cases is lower than the other.

2.2 Data Generation

Once the discrete quantum and classical walk methods were in place and tested, we began generating the dataset of graphs and performed these walk methods to classify the graphs. We represented the $n \times n$ sized random symmetric adjacency matrices in the form of 2-dimensional numpy arrays. The values of the ith row and jth column of the matrix matches with the jth row and ith column to ensure it was symmetric and the diagonals were 0 as the graphs do not have an edge to itself. To generate the start and end nodes of the walk, the start point was randomly selected from the range of (0, $\frac{n}{2} - 1$) and the end node was randomly selected from the range of ($\frac{n}{2} + 1$,n). This selection method of setting the points further away from each other was to try to generate more quantum cases in our dataset as it was indicated from the paper that distant start and end vertices were more

likely to be quantum than nearby vertices. The walk methods were performed on each generated graph, with the step count acting as the classification parameter that determined whether a graph was a classical graph or a quantum graph. Because of the probability distribution that dictates how a quantum particle traverses a graph, we noticed the quantum step count was not constant each time the quantum walk was performed on an identical graph. To mitigate these fluctuations, we performed both walks a specified number of times (determined by the input parameter "shots") and took the average step count for each walk.

Classified datasets of random graphs were created for n = 4, 5, 6, 7, 8, 9, 10, 12, 15 nodes. Initially, 15 000 graphs were classified for each graph size. After removal of duplicate graphs and graphs that could not output a step count, the datasets were reduced to 1000 graphs each. Early on, we noticed that the number of classical graphs greatly outweighed the number of quantum graphs. To mitigate this issue, we only removed classical graphs when downsizing the dataset, and stratified the datasets during the creation of training and test datasets. These steps maximized the ratio of quantum to classical graphs and ensured this ratio was represented in the training and test sets.

However, after using these datasets to train and test our implemented convolutional neural network, we noticed unusually high accuracy's. After brief investigation, we found that the neural network's precision on quantum labelled graphs in the test set was zero percent. This meant our high accuracy was solely due to the correctly labelled classical graphs and indicated overfitting on the classical graphs. This lead us to explore continuous quantum and classical walks, with the hope that these implementations would yield more quantum graphs. Additionally, the paper by Melnikov et. al. outlined a process for implementing coninuous quantum and classical walks that we followed for our implementation.

2.3 Continuous Quantum and Classical Walks

For each continuous-time random walk (classical and quantum) we begin by defining the threshold probability

$$p_{th} = \frac{1}{logn} \tag{1}$$

where n is the dimension of the adjacency matrix. For classical walk, the walk is complete when

$$p(t) > p_{th} \tag{2}$$

In the paper, this inequality is reversed. However, classical probabilities increase with time making that inequality invalid. Our code reflected the corrected inequality.

and for quantum walk, the walk is complete when

$$p(t) < p_{th} \tag{3}$$

The time at which the walk crosses this threshold is called the hitting time and is the parameter used for classifying continuous-time walks.

The classical walk probability is obtained by solving the differential equation

$$\frac{dp(t)}{dt} = (T - I)p(t) \tag{4}$$

where p(t) represents a vector of probabilities, T is the transition matrix of probabilities and I is the identity matrix [1]. The transition matrix is obtained by replacing each element of the nth

column of the adjacency matrix with 0, except for $A_{nn} = 1$. Then, all entries in each column are divided by the in-degree of the respective column. These modifications effectively make the graph directed so that the particle will stay in place one it has reached the final node [1].

From the paper, the solution to the differential equation is

$$p(t) = e^{-t}e^{Tt}p(0) \tag{5}$$

where $p(0) = (1, 0, ..., 0)^T$ representing the probability vector corresponding to a classical particle at the beginning node [1]. A simple loop implementation over a set time range and logical check against the threshold probability determines the time when the particle has traversed the graph. The quantum walk probability can be obtained by solving the Gorini-Kossakowski-Sudarshan-Lindblad (GKSL) equation

$$\frac{d\rho(t)}{dt} = -\frac{i}{\hbar}[H, \rho(t)] + \gamma(L\rho(t)L^\dagger - \frac{1}{2}\{L^\dagger L, \rho(t)\}) \tag{6}$$

where the $\rho(t)$ is a matrix of probabilities, with the Hamiltonian $H = hA^q$, with operator $L = |n+1\rangle\langle n|$, and $\gamma = 1$ [1]. A^q is based on the input adjacency matrix, adding a column and row of zeros to change the dimension from $n \times n$ to $(n+1) \times (n+1)$ [1]. This is done to create a 'sink' vertex which can only be reached if it comes from the target vertex v_t. γ has a large influence of the dynamics and determining if the sink vertex is reached [1]. In order to maintain consistency in the experiment the authors set $\gamma = 1$ for all graphs [1]. The GKSL equation is solved numerically using a wrapper for the SciPy integration library called odeintw. It is given the initial condition of $\rho(0) = |1\rangle\langle 1|$ [1]. The output time is found by tracking the $(n+1) \times (n+1)$ indexed element of ρ, and comparing to the threshold probability p_{th} [1].

3 Interpretation of CQCNN Structure

In the methods of the paper, the convolutional neural network architecture consists of 4-5 convolutional layers followed by fully connected layers. The authors of the paper provide the equations for two most important filters, the edge-to-edge and edge-to-vertex filters (seen below). We recreated the functions based on these matrix equations, and added them as filters in our CNN architecture. Their CQCNN architecture demonstrated six convolutional filters, indicating more depth associated with these filters, however, no explanation was provided in the paper on how this depth was actually implemented. Thus, our second layer that follows the filter was a layer that deletes the symmetric part of the matrices to eliminate redundant information (still following the methods of the paper). Our third layer applies a 3×3 sized convolutional filter to find the relation between the edges. The last layer summarizes the information of the edges in the vertices, using the ETV filter, and reduces the size of the matrix.

$$F_{ij}^{ETE}(A) = [\sum_{k=1}^{n}(A_{ik} + A_{kj}) - 2A_{ij}]A_{ij} \tag{7}$$

$$F_{i}^{ETV}(A) = \sum_{k=1}^{n}(A_{ik} + A_{kj}) - 2A_{ii} \tag{8}$$

For our interpretation of the neural network, we set the first layer with the edge-to-edge filter. Because the paper does not explicitly indicate where the two known filters were used, we assumed that it would be the first layer and assumed it was a generalization of the 6 filters in the paper's first

layer. The second layer of our network deleted the symmetric part of the matrix which corresponded with the paper. The third layer was set as a conv2d layer with an output of 10 channels with a Relu activation to process the matrix transformed by the first two layers. The fourth layer is another conv2d with an output channel and a filter size of 3 which we thought corresponded to their third layer. The final layer applied was the edge to vertex filter as it matches the paper's description of the layer of summarizing the edges in the vertices and reduces the data size to $n \times 1$. For the fully connected layers, we took the implementation into our own hands and had 2 layers with a relu in between and a softmax activation at the end. Generally, our architecture matched that of the paper very closely considering the information provided to us.

4 Reflection on author's CQCNN structure

As mentioned above, one of the main reasons our results differed from the expected was due to our CQCNN structure. Unfortunately, our paper did not describe how the depth of the convolutional layers were implemented, but rather just provided matrix equations to apply the mentioned edge-to-edge and edge-to-vertex filtering. This lead us to explore further sources, until we stumbled upon a similar paper by the same author, *Machine Learning Transfer Efficiencies for Noisy Quantum Walks* [2]. This was only recently discovered by our group, much after we designed our own CQCNN architecture, which is why it was not implemented before the final project deadline, but was still of great interest to us. This new paper also uses the same CQCNN they constructed before.

The CQCNN has specifically designed learnable *cross* filters. The first layer of cross filters extracts features corresponding to a function of weighted numbers of neighbouring edges. The second layer of cross filters extracts information about neighboring edges of neighboring edges. The third layer continues to learn deeper about neighbors of neighbors, and extracts information about the overall connectivity of the graph. The features of graphs are found in the cross rows and columns of graphs, which is why the specifically designed convolution works with cross filters.

Our implementation includes the mentioned edge-to-edge cross filter required for the model to extract graph feature information, however, the missing piece of the puzzle may be that we did not add *enough* filters. If we apply numerous cross filters after another, we may see better results. This is where the depth of the CNN layers should come from.

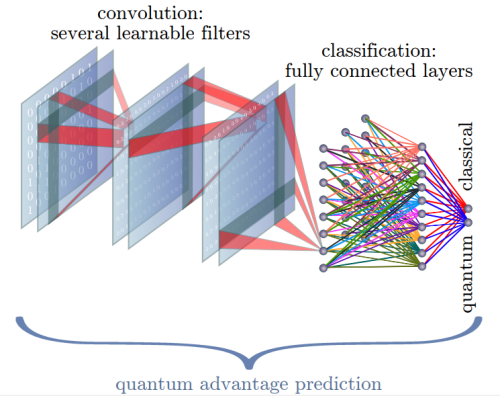

Figure 1: CQCNN structure, with several nested cross filters, used to extract information on total connectivity of the graph [1]

5 Comparison of results

Figure 2, from the paper shows the CQCNN's performance on each label (classical and quantum) by comparing the recall and precision on a test set of 1000 graphs. This plot is important in detecting overfitting on either of the labels and is a good indicator of the network's overall performance. Once we had implemented the continuous random walks, we intended to re-create figure 2 to indicate if the same overfitting was occurring, and if our network's accuracy was representative of it's performance.

However, after investigation of the network's performance on the new dataset, we found that the network was still not "learning" the quantum graphs, and that the reported accuracy falsely reported good network performance.

We think this our lack of results in this case is due to a couple factors. The most significant issue we faced was an under-representation of quantum graphs in our dataset. Over all graph sizes, we could not achieve more than 6% quantum graphs, with many datasets composed of fewer than 5% quantum graphs. An over-representation of classical graphs means that classical graph features are being learned more consistently than quantum graph features and will have more influence during testing. We were also unable to train our network as extensively as the CQCNN in the paper. Our training was performed with 10 epochs, whereas the paper trained on 2000 epochs. However, without training our network with the same parameters, we are unable to confirm with certainty that this discrepancy is a cause of the lack of results.

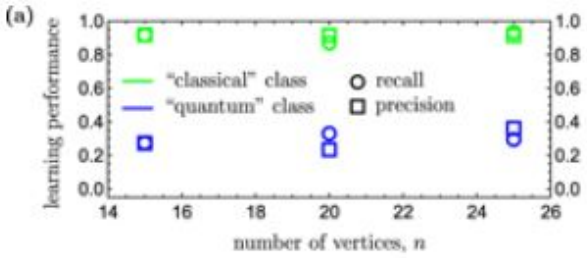

Figure 2: Learning performance (recall and precision) of the CQCNN on graphs of size n = 15, 20, 25.

Additionally, the paper reported the accuracy of the CQCNN with a plot of test accuracy's obtained for graphs of size n= 6, 7, 8, 9, 10 and training loss on a network trained specifically on n = 4, 5 sized graphs. We were not able to re-create this specific plot as our n = 4, 5 datasets did not contain any quantum graphs on which to train. As can be seen in our presentation slides, we did report a similar plot of accuracy's and losses, however our plot represents the training accuracy and training loss for individual networks trained on n = 6, 7, 8, 9, 10, 12, 15 sized graphs.

6 Conclusion

As a group we investigated an implementation of the ideas proposed in the paper by Alexy Melinkov et. al. We discovered different results as our model began to overfit. This may have been caused by the variations in the CQCNN architecture, the training time, and hyperparameter tuning. Additionally, the data generated can vary drastically in graph structure. Thus, it is possible that we were unable to produced similar results due to the fixed CQCNN architecture in the original paper. If we were to implement this project again, we would remodel the CQCNN closer to the authors code (which we only had access to on April 1st) and allow for additional hyperparameters such as number of layers, filters and layer order of the CQCNN. This would allow a greater likely hood for the feature detection layers to capture the structure of the varying graph data sets. Next, we would increase the amount of training data, and number of epochs to attempt to reduce the overfitting. The paper was good at describing the fundamental implementations of classical and quantum random walks. It would have been valuable for the reader if the paper went more in depth into the specifics of the implementation of the CQCNN architecture. Furthermore, the randomly generated data became difficult to replicate exactly as there is a large variation of graph structure as the number of vertex increases. By the nature of this implementation, this makes it difficult to reproduce results without overfitting or receiving different accuracies. In the future it would be useful for researches to have a standard data set similar to the purpose of MNIST, CIFAR-10 or ImageNet which could benchmark various quantum and classical algorithms on graphs. This would allow machine learning implementations in physics to be compared more consistently.

References

[1] Alexey A Melnikov *et al 2019 New J. Phys.* **21** 125002

[2] A. A. Melnikov, L. E. Fedichkin, R.-K. Lee, and A. Alodjants, Adv. Quantum Technol. 3, 1900115 (2020)