QHack 2022

Open hackathon

Quantum Approaches for Dynamic Bayesian Network Structure Learning

Ixchel Meza Chávez

1. Introduction

Dynamic bayesian network structure learning (DBNSL)'s goal is to obtain P(G|D), that means obtaining a direct acyclic graph (DAG) given a dataset consisiting of at least two events or moments in time. This solution can be apply to solve problems in different fields, like finance, biology and ecology to name a few. This kind of problem is super-exponential (O(n! 2^(n!/(2!(n-2)!)))) [1] due to a search over all DAGs possible.

2. Goal

My goal during the Open Hackaton was to explore different approaches than the ones reported on the field, as far as I know, to deal with DBNSL, so I decided to implement three different approaches with the goal in mind to use the minimum number of qubits possible which was $n_{qubits} = 2 * ceil\left(\log_2\left(2 * n_{variables}\right)\right)$. The probabilities measured from the quantum circuit would represent all possible edges, were the first half of the binary number correspond to the vertice where the edge begins and the other half correspond to the vertice where the edge ends i.e. if my DBN has 4 variables, it has 8 nodes (4 of time t and 4 of time t+1), which means 16 possible edges, that can be represented through a binary number of six digits and where for example the edge (2,5) corresponds to 010101

vertice_2 --> vertice_5
010 101

The methods I explored were:

1) Hybrid network = (classic) encoder + (quantum) circuit
2) Variational circuit
3) Quantum circuit structure learning

2. Generated datasets

The dummy datasets I used to train and also to test, were generated through the following steps:

1) Randomly picked between n and n+3 edges, where n is the number of variables
2) Based on the edges, generate the first set of values for time t, choosing the first value randomly, so that for each edge (v_i , v_j), based on the value of v_i , $v_j = v_j + \left(w_i + w_j\right) * v_i$, where w_i is a weight related to vertice i .
3) Generate set of values for time t+1 as in step 2)

The way v_j is obtained in step 2) could change based on possible information of the behavior of the DBN we are interested in.

The ground truth values of the dataset was formed by a vector of zeros except the ground truth edges of the DBN, which had the value 1/n_gt_edges

3. Approaches

3.1. Hybrid network = (classic) encoder + (quantum) circuit.

I implemented a hybrid network composed by an encoder with an input of the shape (1,n_nodes) with one hidden layer and one output layer with shape (1,n_qubits), and a quantum circuit with a set of layers ls formed by three layers of parameterized RX, RY and RZ gates followed by rings of CNOT gates in all wires. I changed the number of ls layers changing the depth of the circuit.
The output of the network was the probabilities vector of the quantum circuit.

3.2. Variational circuit

This quantum circuit consisted of layers of parameterized Rot gates with a embedding layer consisting of Rot gates distributed in the qubits whose angles corresponded to $\phi = v_t^i, \theta = v_{t+1}^i, \omega = v_{t+1}^i - v_t^i$.
If the circuit had more number of variables than number of qubits, there were more embedding layers. Here the output was also the probabilities vector of the quantum circuit

3.3. Quantum circuit structure learning

This quantum circuit was a variant of the variational circuit approach, except that the parameterized layers weren't fixed, following the rotoselect algorithm [2]

4. Implementation

I used pennylane and pytorch to implement the algorithms.

For the first round, I trained the models with tiny datasets consisting of 4 variables. It was done locally monitoring the loss, and two scores, one simple and the other weighted, based on the position or ranking r_i of the predicted edges arranging them based on their probability in descending order,

$$score = 1 - \frac{\sum r' - \sum_{n=1}^{n_{gtedges}} n}{\sum_{n=n_{edges}-n_{gtedges}}^{n_{edges}} n - \sum_{n=1}^{n_{gtedges}} n}$$

where $r_i' = r_i * P_i * n_{gtedges}$ if weighted, otherwise $r_i' = r_i$

5. Results

The models with the tiny datasets learned! Although I needed more time to explore more combinations of hyperparameters and I got interesting graphs. However when I started using braket to implement the other bigger datasets I had prepared, I got the following error

ValidationException: An error occurred (ValidationException) when calling the CreateQuantumTask operation: Shots 0 for result type probability are only supported for range 1 to 100000, inclusive, for backend ARN arn:aws:braket:::device/quantum-simulator/amazon/sv1

which was due to my outputs measures being the probabilites, so It was recommended to me not to implement them on the simulator, just to use the local device, also because my approaches were thought for circuits with fewer qubits. However I did experiment on braket. I didn't attempt to use the QPU, I needed more time!

Finally I want to thank you all deeply. In two weeks I learned a lot and I am eager to continue!

5. References:

[1] Prashant S. Emani1, et al, Quantum Computing at the Frontiers of Biological Sciences
[2] https://pennylane.ai/qml/demos/tutorial_rotoselect.html