Synthesis of the Optimal Quantum Circuit with Deep Reinforcement Learning

---

Sepehr Akbari

2/4/2025

Department of Math & CS, Lake Forest College

Recall: Quantum Computing

- In quantum physics, particles can exist in a superposition of states before getting measured.
- In quantum computing, qubits model this behavior by being in a superposition of binary states.
- In classical computing we use logic gates (AND, OR, NOT, etc.) to manipulate bits.
- In quantum computing we use quantum gates (Hadamard, CNOT, Pauli-X, etc.) to manipulate qubits.
- Quantum gates are just matrices that when applied (multiplied) to a qubit (vector) change its state.
- A set of quantum gates applied in some sequence is called a quantum circuit.

Building a Quantum Circuit

A quantum circuit is essentially a sequence of matrix multiplications applied to a state vector. From linear algebra, each matrix can be viewed as a transformation.

If we have an initial state $|\psi_0\rangle$ and we apply gates $G_1, G_2, \ldots, G_n$ in sequence, we get

$$|\psi_{final}\rangle = G_n \times G_{n-1} \times \cdots \times G_1 \times |\psi_0\rangle$$

Observe. We can multiply all these gate matrices together to form one single unitary matrix U_{total} .

$$U_{total} = G_n \times \cdots \times G_1$$

This means different sequences of gates can result in the *exact same* U_{total}.

Example: Simulating Interactions

In Quantum Machine Learning, Chemistry, etc. we often need to simulate interactions between qubits (e.g. to see how a molecule behaves at the quantum level).

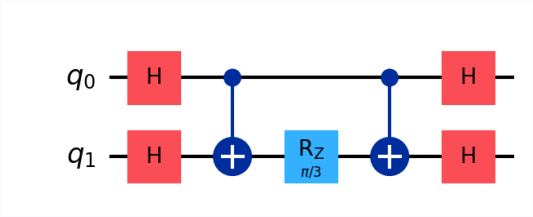
We can use a concept called the Ising coupling

$$\exp(-i\frac{\theta}{2}X \otimes X)$$

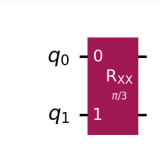
to model these interactions.

Let's build a circuit for this interaction.

Here is a circuit that does this using standard gates



Since this is a common operation, we have a specialized R_{XX} gate that does the same thing more efficiently



Why did we care to make the second circuit?

The NISQ Era Problem

- We are currently in the **N**oisy **I**ntermediate-**S**cale **Q**uantum era.
- Real qubits are noisy. They lose their quantum state (decoherence) very quickly.
- Every extra gate adds time. A deeper circuit means more time for noise to corrupt the result.
- We need to minimize circuit depth to reduce error and achieve higher execution fidelity.
- For complex algorithms, finding the absolute shortest circuit by hand is incredibly difficult (usually NP-hard).

Reinforcement Learning

Overview: Classical RL

Imagine playing a game where you want to maximize your score.

You create a massive lookup table, called a *Q-Table*.

- *Rows:* Every possible situation you can be in.
- *Columns:* Every possible move you can make.
- *Value:* How *good* that move is (Q-Value).

Learning: You try a move, see what happens, and update the number in the table (you could see these Q-values as weights in a neural network). Eventually, you just look up the highest number in your row to win.

State	Action α	Action β	Action δ
X	0.1	**9.5**	-2.0
Y	**5.2**	1.1	0.0

Overview: Deep RL

What happens when the number of states (or actions) is too large... or infinite?

Classical RL fails.

Solution: Use a Neural Network to approximate the Q-Table. The network takes the State as input and predicts the Q-Values for all possible actions. It is called a Q-Network.

Learning: The agent takes an action, receives a reward, and updates the network weights to minimize the difference between predicted Q-Values and target Q-Values (based on the reward and maximum future Q-Value).

Overview: Components of RL

- *Agent:* The learner/decision maker (Our Neural Network).
- *Environment:* The world the agent interacts with (The Quantum Simulator).
- *State (s_t):* A snapshot of the current situation (The current Quantum Circuit/State).
- *Action (a_t):* A move the agent can take (Adding a specific Gate).
- *Reward (r_t):* Feedback signal. Positive for success, negative for errors (High Fidelity vs. Circuit Depth).
- *Value Function:* Estimates expected future rewards.

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V^{\pi}(s') \right]$$

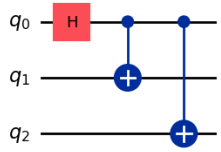
GHZ Circuit

We task the agent with generating the *Greenberger-Horne-Zeilinger (GHZ)* state.

A maximally entangled state crucial for quantum communication and error correction.

For N qubits, the state is defined as

$$|GHZ_N\rangle = \frac{|0\rangle^{\otimes N} + |1\rangle^{\otimes N}}{\sqrt{2}}$$



Can we find this circuit using Reinforcement Learning?

State Representation

We could feed all the full state vector to the Neural Network. Curse of Dimensionality... Dimension is 2^N. For 20 qubits, the input vector size is more than 1 million neurons. This is computationally intractable.

We use *Pauli Expectation Values* instead. We measure the expectation of X, Y, Z operators on each qubit individually.

$$\text{Input Features} = [\langle X_0 \rangle, \langle Y_0 \rangle, \langle Z_0 \rangle, \ldots, \langle Z_{N-1} \rangle]$$

Reduces complexity from *Exponential* (2^N) to *Linear* ($3N$).

Markov Decision Process

We model the environment as a Markov Decision Process (MDP).

We use first construct our list of operators, and compute the expectation values

$$\langle P_i \rangle = \langle \psi | P_i | \psi \rangle$$

and flatten them into our network's input tensor.

Every iteration starts with the ground state $|0...0\rangle$.

We need to have our action space defined as well. An action corresponds to adding a specific gate to the circuit (e.g., `cx(0, 1)`).

Markov Decision Process

Each step in the MDP consists selecting an input action (gate), applying it to the current quantum stat, and receiving a reward based on how close the resulting state is to the target GHZ state.

The reward function takes into account the *fidelity* with the target state. Our goal is to achieve high fidelity with as few gates as possible. Fidelity of two states $|\psi_t\rangle$ and $|\psi_i\rangle$ is defined as

$$F = |\langle\psi_t|\psi_i\rangle|^2$$

Based on the goal, the reward structure is as follows

- Large reward for $F > 0.99$.
- Small penalty for each gate added.
- Small reward for improvements in fidelity $F_{new} - F_{old} > 0$.

Experience Replay

In our circuit, state s_{t+1} is highly correlated with s_t. If a Neural Network learns from sequential data, it overfits to the "current" trajectory and forgets everything else.

It's better to study using shuffled flashcards than to just re-read the textbook in order.

To solve this, we want to give random samples from past experiences to the network during training. We store every step as a tuple

$$e_t = (s_t, a_t, r_t, s_{t+1}, \text{done})$$

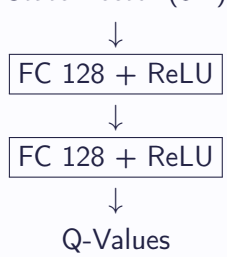
and sample it in batches of 32 memories.

Network Architecture

The Model Structure

- *Input Layer:* Size $3N$ (The Pauli Feature Vector).
- *Hidden Layers:* Two fully connected layers with 128 neurons each.
- *Activation:* ReLU (Rectified Linear Unit) for non-linearity.
- *Output Layer:* Size $N_{actions}$ (One Q-Value for every possible gate operation).

Visualizing the Flow



Double DQN

In standard Q-Learning, we update the network using its own predictions. This is like chasing your own tail (the target changes as you learn).

We can use another network to provide stable targets.

- *Policy Net (θ):* The active network that selects actions and learns.
- *Target Net (θ^-):* A frozen copy used to calculate ground truth values.

We periodically copy weights from Policy to Target to keep updates stable using the rule

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

every 100 episodes.

Learning

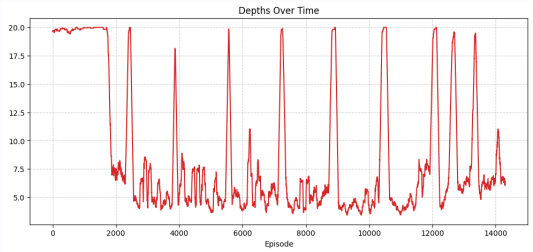
Training the network is as usual.

- *Sample.* Draw a random batch of transitions from the Replay Buffer.
- *Predict.* Calculate current Q-values using the Policy Net: $Q(s, a; \theta)$.
- *Target.* Calculate expected Q-values using the Target Net (Bellman Equation).
- *Loss.* Compute the error between Prediction and Target. In this case, I used Huber Loss because it is less sensitive to outliers.
- *Optimize.* Back-propagate the error to update the weights, with Adam.

The resulting Policy Net is then used to select actions during inference.

Results

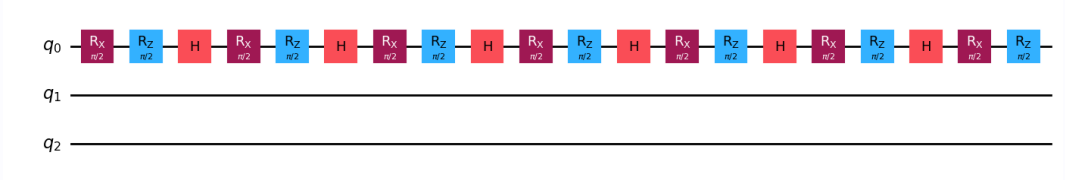
With the goal of generating a 3-qubit GHZ state, our rewards and circuit depth over time



which shows both improvement and correct correlation between reward and circuit depth.

Results

The initial circuit produced by the agent was

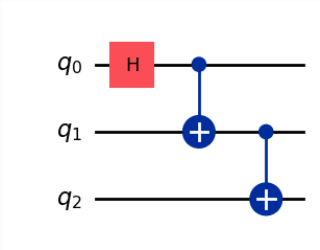


which is essentially just random weights/gates. This has fidelity of 0.25 and depth of 20.

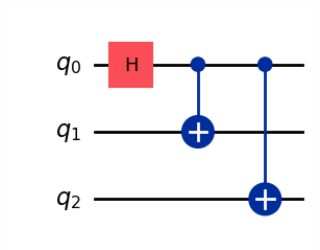
Results

After training, the final circuit produced was

Agent's Circuit



known Optimal Circuit



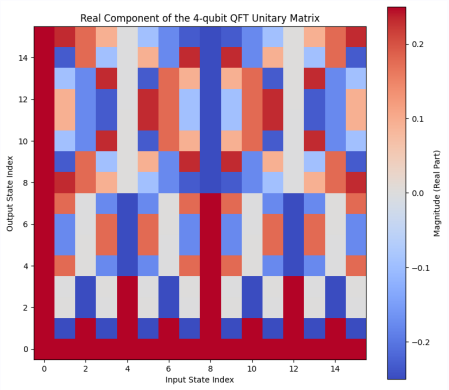
which is, not exactly, but equivalent of the known optimal circuit! It has a fidelity of 1 and depth of 3.

Operator Circuit

Instead of preparing a single state, we want to build a black box circuit that performs the *Quantum Fourier Transform (QFT)*.

QFT is the engine behind *Shor's Algorithm* and Quantum Phase Estimation.

$$\text{QFT}(|j\rangle) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi ijk/N} |k\rangle$$



Operator Circuit

This is called *Unitary Operator Synthesis*, and is a much more challenging task than State Synthesis.

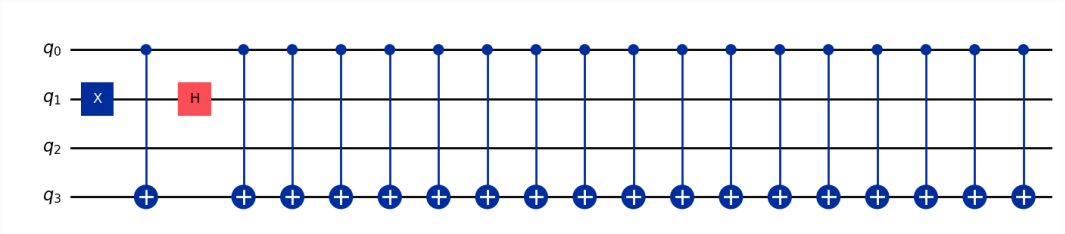
- *In GHZ_3.* The circuit assumes input $|000\rangle$.
- *In QFT.* The circuit must work for any input state $|\psi\rangle$.

There are some minor differences in the process, but mainly we change the reward metric to *Unitary Fidelity* (Trace Distance) which is defined as

$$F(U, V) = \frac{1}{d^2}|\mathrm{Tr}(U^\dagger V)|^2$$

Results

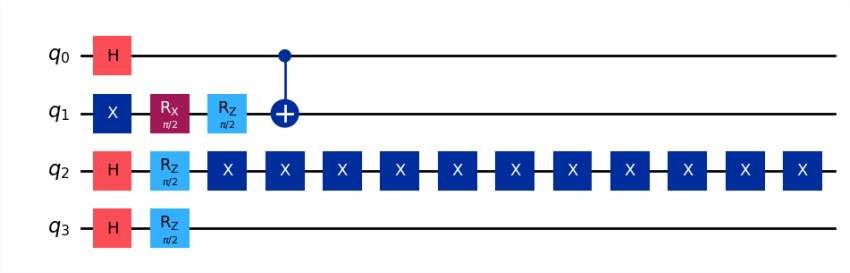
The initial circuit produced by the agent was



Very bad. This has unitary fidelity of 0.0026 and depth of 18.

Results

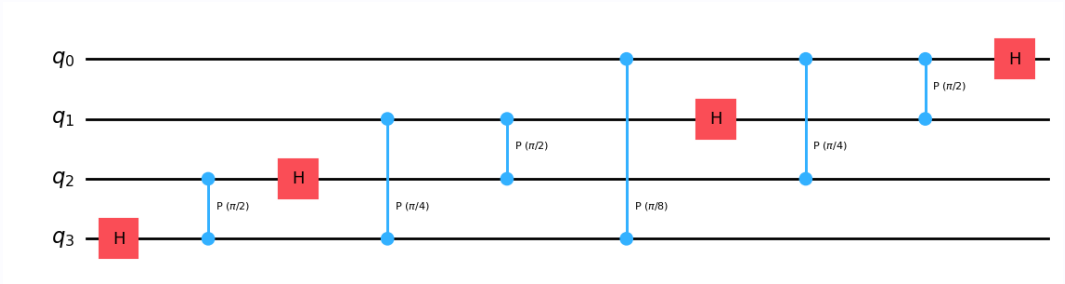
Something interesting that happened here, was this was our circuit after about 5000 episodes of training:



Pretty bad as well. But this has unitary fidelity of 0.0006 and depth of 13. Which shows that the agent was more focused on minimizing depth rather than maximizing fidelity (correct behavior).

Results

The final circuit after training was



this does not have a known optimal, but the fidelity is about 0.99 and depth of 7. This is still not an optimal behavior, since the known best depth is 4 or 5.

Thank You!

You can view more details and the full source code at

`https://github.com/SepehrAkbari/qgate-dqn`

Thank You!