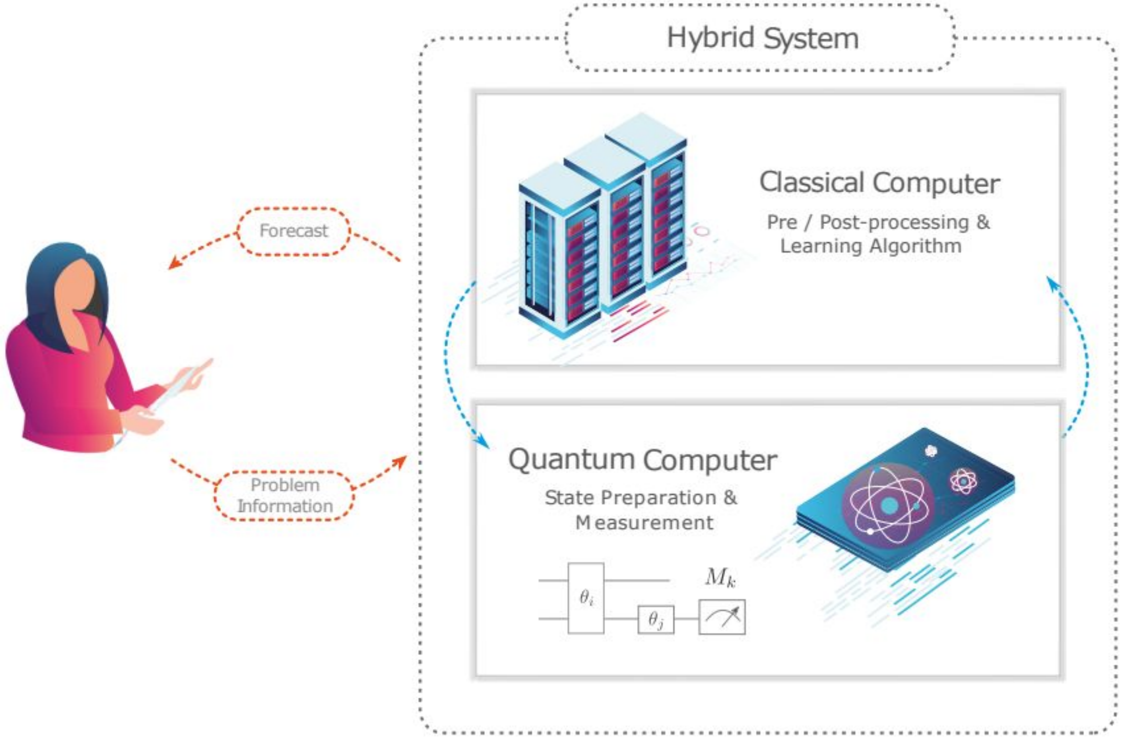
Parametrized Quantum Policies for Reinforcement Learning

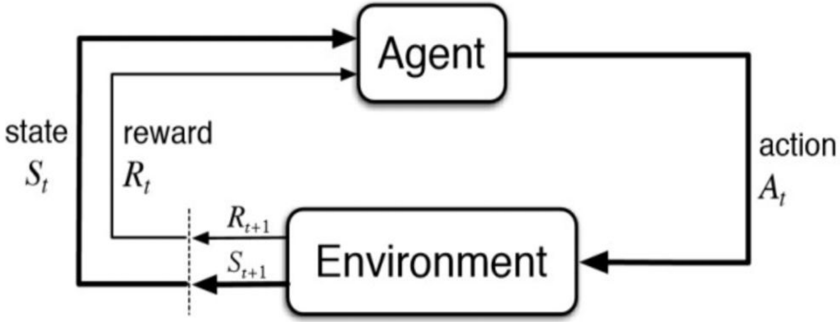
Sofiene Jerbi, Casper Gyurik, Simon C. Marshall, Hans J. Briegel, Vedran Dunjko

Quantum Machine Learning



Reinforcement Learning

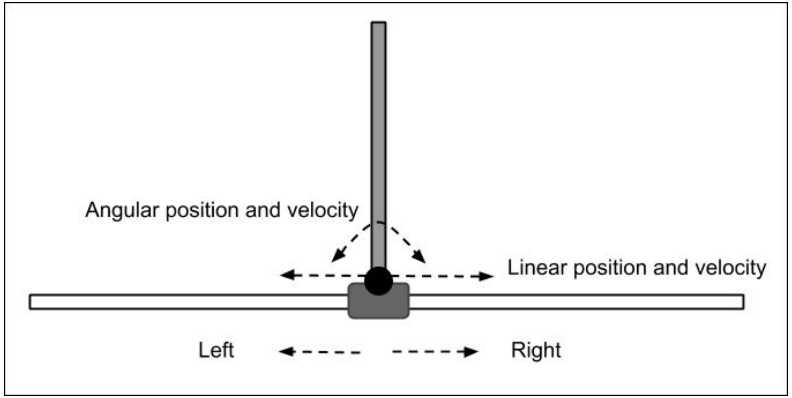
- In RL, an agent does not learn from a fixed data set as in other types of learning, but by making observations on and interacting with an environment.
- An environment consists of a set of possible states S that it can take, and a set of actions A which the agent can perform to alter the environment's state.
- The reward function is designed to evaluate the quality of the agent's actions on the environment based on the learning task at hand



Example: Cartpole Environment

- A baseline MDP environment
- State: [ang from perpendicular, ang velocity, horizontal position, horizontal velocity]
- Action: [left/right] push at base
- Goal: Keep pole upright
- Reward: +1 for each state in which pole has not fallen

(we terminate episode when pole falls/some fixed number of states have happened)



Parameterized Quantum Circuits

Variational or parametrized quantum circuits are quantum algorithms that depend on free parameters. Like standard quantum circuits, they consist of three ingredients:

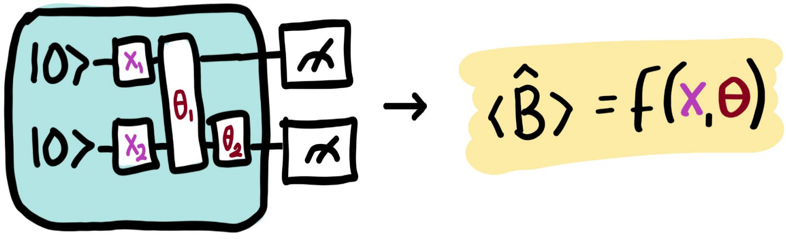
a. Preparation of a fixed initial state (zero state)

b. A quantum circuit $U(\theta)$, parameterized by a set of free parameters θ.

c. Measurement of an observable B^ at the output. This observable may be made up from local observables for each wire in the circuit, or just a subset of wires.

Parameterized Quantum Circuits

1. Input encoding
2. Scaling and parameterized rotations
3. Data Reuploading
4. Entanglement
5. Observation
6. Circuit Learning



Circuit Learning

1. Encode input data $\{\boldsymbol{x}_i\}$ into some quantum state $|\psi_{\text{in}}(\boldsymbol{x}_i)\rangle$ by applying a unitary input gate $U(\boldsymbol{x}_i)$ to initialized qubits $|0\rangle$
2. Apply a $\boldsymbol{\theta}$-parameterized unitary $U(\boldsymbol{\theta})$ to the input state and generate an output state $|\psi_{\text{out}}(\boldsymbol{x}_i, \boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})|\psi_{\text{in}}(\boldsymbol{x}_i)\rangle$
3. Measure the expectation values of some chosen observables. Specifically, we use a subset of Pauli operators $\{B_j\} \subset \{I, X, Y, Z\}^{\otimes N}$. Using some output function F, output $y_i = y(\boldsymbol{x}_i, \boldsymbol{\theta})$ is defined to be $y(\boldsymbol{x}_i, \boldsymbol{\theta}) \equiv F(\{\langle B_j(\boldsymbol{x}_i, \boldsymbol{\theta})\rangle\})$.
4. Minimize the cost function $L(f(\boldsymbol{x}_i), y(\boldsymbol{x}_i, \boldsymbol{\theta}))$ of the teacher $f(\boldsymbol{x}_i)$ and the output y_i, by tuning the circuit parameters $\boldsymbol{\theta}$ iteratively.
5. Evaluate the performance by checking the cost function with respect to a data set that is taken independently from the training one.

Encoding Classical States Quantumly

- For discrete state-action spaces, sometimes it may be possible to **exactly** encode binary representations of states to qubits
- In the continuous case, we can treat initial qubit rotations also as parameters to learn
- We learn scales over state dimension and appropriately apply rotations

In particular for this paper: s[i] - > sum(s[i]*lambda[j])

After this we apply an activation function. Here lambda is a learnable parameter

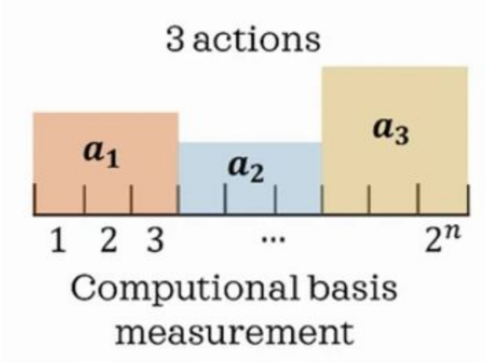
Observation and Quantum Policy

- For policy gradient methods policy is expressed as the **probability of taking an action in a state**

$$\pi_{\boldsymbol{\theta}}(a|s) = \langle P_a \rangle_{s,\boldsymbol{\theta}}$$

- By partitioning our Hilbert space into subspaces, each corresponding to an action we can use a projective measurement on our final state(after unitaries) to get our action

$$\sum_a P_a = I \textit{ and } P_a P_{a'} = \delta_{a,a'}$$



Parameter Shift Rules

- We need gradient of policy for optimisation of parameters in REINFORCE
- After we get our gradients, we can perform gradient descent

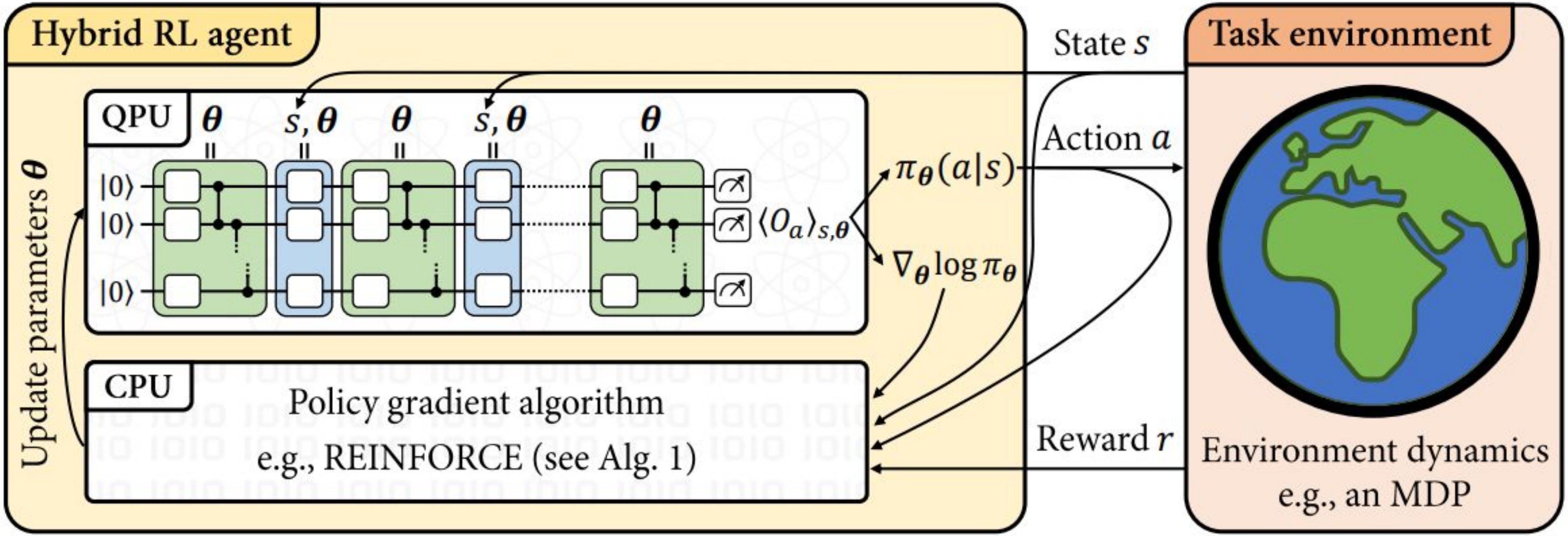
$$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) = \beta\Big(\nabla_{\boldsymbol{\theta}} \langle O_a \rangle_{s,\boldsymbol{\theta}} - \sum\nolimits_{a'} \pi_{\boldsymbol{\theta}}(a'|s) \nabla_{\boldsymbol{\theta}} \langle O_{a'} \rangle_{s,\boldsymbol{\theta}}\Big)$$

$$\partial_i \langle O_a \rangle_{s,\boldsymbol{\theta}} = \frac{1}{2}\Big(\langle O_a \rangle_{s,\boldsymbol{\theta}+\frac{\pi}{2}\boldsymbol{e_i}} - \langle O_a \rangle_{s,\boldsymbol{\theta}-\frac{\pi}{2}\boldsymbol{e_i}} \Big)$$

$$\text{Compute } \Delta\boldsymbol{\theta} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^{(i)}|s_t^{(i)}) \left(G_{i,t} - \widetilde{V}_{\boldsymbol{\omega}}(s_t^{(i)})\right)$$

$$\text{Update } \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\Delta\boldsymbol{\theta};$$

Pipeline



Hybrid RL agent
QPU
$\boldsymbol{\theta}$
$s, \boldsymbol{\theta}$
$\boldsymbol{\theta}$
$s, \boldsymbol{\theta}$
$\boldsymbol{\theta}$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$\langle O_a \rangle_{s,\boldsymbol{\theta}}$
$\pi_{\boldsymbol{\theta}}(a|s)$
$\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}$
Update parameters $\boldsymbol{\theta}$
CPU
Policy gradient algorithm
e.g., REINFORCE (see Alg. 1)
State s
Action a
Reward r
Task environment
Environment dynamics
e.g., an MDP

REINFORCE

Algorithm 1: REINFORCE with PQC policies and value-function baselines

Input: a PQC policy $\pi_{\boldsymbol{\theta}}$ from Def. 1; a value-function approximator $\widetilde{V}_{\boldsymbol{\omega}}$

Initialize parameters $\boldsymbol{\theta}$ and $\boldsymbol{\omega}$;

while *True* **do**

  Generate N episodes $\{(s_0, a_0, r_1, \ldots, s_{H-1}, a_{H-1}, r_H)\}_i$ following $\pi_{\boldsymbol{\theta}}$;

  for *episode i in batch* **do**

    Compute the returns $G_{i,t} \leftarrow \sum_{t'=1}^{H-t} \gamma^{t'} r_{t+t'}^{(i)}$;

    Compute the gradients $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^{(i)} | s_t^{(i)})$ using Lemma 1;

  Fit $\{\widetilde{V}_{\boldsymbol{\omega}}(s_t^{(i)})\}_{i,t}$ to the returns $\{G_{i,t}\}_{i,t}$;

  Compute $\Delta\boldsymbol{\theta} = \dfrac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t^{(i)} | s_t^{(i)}) \left(G_{i,t} - \widetilde{V}_{\boldsymbol{\omega}}(s_t^{(i)}) \right)$;

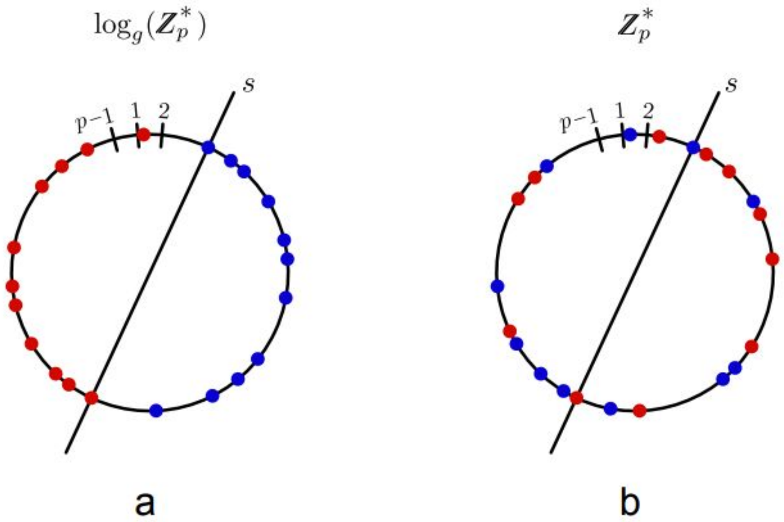
  Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta\boldsymbol{\theta}$;

Quantum Advantage

By embedding classical-hard problems to our environments we can get provable quantum advantage

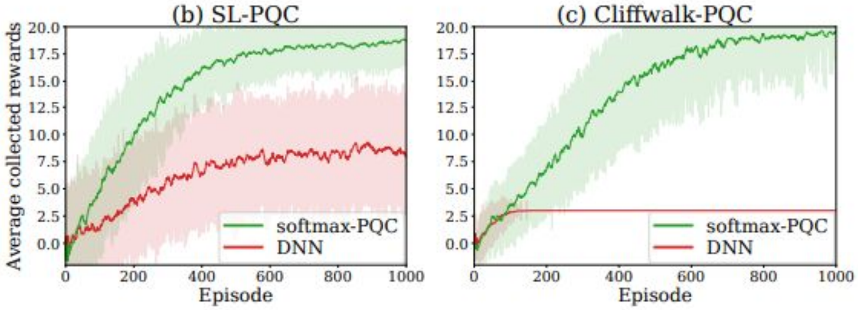
For example, following function creates a separation in our group which can be easily determined if we can find discrete log

$$f_s(x) = \begin{cases} +1, & \text{if } \log_g x \in [s, s + \frac{p-3}{2}] \\ -1, & \text{otherwise.} \end{cases}$$



Extra: Results on Quantum Advantage

- A PQC is used as labeling function
- Aim to use a ‘quantum function’ approximated by a quantum circuit
- A classical algorithm cannot build an easy separation



Extra: Results on Quantum Advantage

- SL-PQC: this degenerate RL environment encodes a classification task in an episodic RL environment: at each interaction step of a 20-step episode, a sample state s is uniformly sampled from the dataset S, the agent assigns a label $a = \pm 1$ to it and receives a reward $\delta_{f(s),a} = \pm 1$.

- Cliffwalk-PQC: this environment essentially adds a temporal structure to SL-PQC: each episode starts from a fixed state $s_0 \in S$, and if an agent assigns the correct label to a state s_i, $0 \leq i \leq 19$, it moves to a fixed state s_{i+1} and receives a +1 reward, otherwise the episode is instantly terminated and the agent gets a −1 reward. Reaching s_{20} also causes termination.