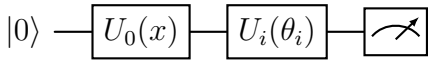
This report supplements the circuit-generating code by deriving a learning algorithm that enables parameterized quantum circuits to serve as policies for reinforcement learning. [1].

Contents

1 Parameter Shift Rule

1.1 Single Parameterized Gate

In classical machine learning, one often computes the gradient of a loss function with respect to network parameters through backpropagation. We follow a similar approach in variational quantum algorithms, except that the output is the expectation value of a measurable observable $\hat{A}$ [2, 3]. Consider a simple circuit with an data-encoding gate $U_0(x)$ and single parameterized gate $U_i(\theta_i)$:



The gate $U_0(x)$ maps classical data x into a quantum state: $|x\rangle = U_0(x)|0\rangle$. We define the circuit function as

$$f(x;\theta_i) = \langle 0|U_0^\dagger(x)U_i^\dagger(\theta_i)\hat{A}U_i(\theta_i)U_0(x)|0\rangle = \langle x|U_i^\dagger(\theta_i)\hat{A}U_i(\theta_i)|x\rangle.$$

Unitary operators apply linear operations that preserve inner products in a Hilbert space, so we can represent them as linear transformations for succinctness. By defining the linear transformation $\mathcal{T}_{\theta_i}(\hat{A}) := U_i^\dagger(\theta_i)\hat{A}U_i(\theta_i)$, we can write the circuit function as

$$f(x;\theta_i) = \left\langle x|\mathcal{T}_{\theta_i}(\hat{A})|x\right\rangle. \tag{1}$$

Since $\mathcal{T}_{\theta_i}(\hat{A})$ depends on smoothly on θ_i, its derivative is well-behaved and given by

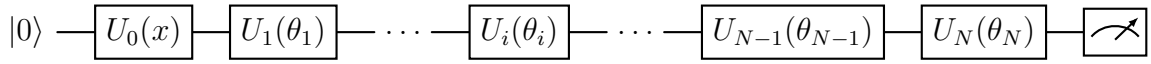
$$\frac{\partial}{\partial\theta_i}f(x;\theta_i) = \left\langle x|\frac{\partial}{\partial\theta_i}\mathcal{T}_{\theta_i}(\hat{A})|x\right\rangle. \tag{2}$$

The gradient of a quantum circuit function can often be written as the linear combination of other quantum functions, so we can apply well-studied gradient-based optimization tools [4]. Here, we express it in terms of the same linear transformation with shifted parameters:

$$\nabla_{\theta_i}\mathcal{T}_{\theta_i}(\hat{A}) = \alpha\left(\mathcal{T}_{\theta_i+\beta}(\hat{A}) - \mathcal{T}_{\theta_i-\beta}(\hat{A})\right), \tag{3}$$

where constants α and β are independent of $\mathcal{T}$, and β is finite (a notable distinction, since the numerical finite-difference method has a similar form but infinitesimal β).

1.2 Multiple Parameterized Gates



In practice, variational circuits have many gates (like the one above) and therefore many tunable parameters $\theta = (\theta_1, \theta_2, \dots)$. The overall unitary transformation is

$$U(x;\theta) = U_N(\theta_N)U_{N-1}(\theta_{N-1})\cdots U_1(\theta_1)U_0(x).$$

The circuit function becomes $f(x;\theta) = \langle 0|U^\dagger(x;\theta)\hat{A}U(x;\theta)|0\rangle$. However, we want to isolate the contribution from the i-th gate, so we construct a partition:

$$f(x;\theta) = \overbrace{\langle x|U_1^\dagger(\theta_1)\cdots}^{\langle\psi_{i-1}|} U_i^\dagger(\theta_i) \overbrace{\cdots U_N^\dagger(\theta_N)\hat{A}U_N(\theta_N)\cdots}^{\hat{A}_{i+1}} U_i(\theta_i) \overbrace{\cdots U_1(\theta_1)|x\rangle}^{|\psi_{i-1}\rangle}.$$

All gates preceding $U_i(\theta_i)$ are absorbed into updated state $|\psi_{i-1}\rangle = U_{i-1}(\theta_{i-1})\cdots U_1(\theta_1)|x\rangle$ which represents the state on the wire just before the i-th gate. All gates following $U_i(\theta_i)$ are absorbed into an updated observable $\hat{A}_{i+1} = U_{i+1}^\dagger(\theta_{i+1})\cdots U_N^\dagger(\theta_N)\hat{A}U_N(\theta_N)\cdots U_{i+1}(\theta_{i+1})$. Armed with new notation, we rewrite the circuit function as

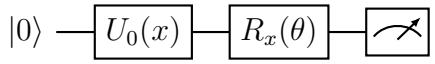
$$f(x;\theta) = \langle\psi_{i-1}|U_i^\dagger(\theta_i)\hat{A}_{i+1}U_i(\theta_i)|\psi_{i-1}\rangle = \langle\psi_{i-1}|\mathcal{T}_{\theta_i}(\hat{A}_{i+1})|\psi_{i-1}\rangle.$$

Having arrived at a function identical to the single-gate case in Eq. 1, we know that the derivative will be analogous to Eq. 2, so we can write its gradient in terms of Eq. 3:

$$\nabla_{\theta_i} f(x;\theta) = \langle\psi_{i-1}|\nabla_{\theta_i}\mathcal{T}_{\theta_i}(\hat{A}_{i+1})|\psi_{i-1}\rangle. \tag{4}$$

With this, we can update one gate at a time, enabling exact gradient-based optimization of the circuit parameterization vector θ. We implement this in PennyLane, a framework for quantum differentiable programming. When instantiating a QNode, specify the differentiation method with decorator `@qml.qnode(dev, diff_method="parameter-shift")`.

1.3 Example: Rotation Operator Gates



For the data-encoding and processing circuit, let the parameterized gate be an R_z rotation operator defined as

$$R_z(\theta) = e^{-i\theta\sigma_z/2} = \sum_{n=0}^{\infty}\frac{1}{n!}\left(-\frac{i\theta}{2}\sigma_z\right)^n.$$

Here, θ is the angle of rotation about the z-axis and σ_z is a Pauli matrix. We define linear transformation $\mathcal{T}_\theta(\hat{A}) := R_z^\dagger(\theta)\hat{A}R_z(\theta)$. To compute Eq. 2, we must differentiate the R_z operator by first applying the chain rule:

$$\frac{\partial}{\partial\theta}R_z(\theta) = \sum_{n=1}^{\infty}\frac{n}{n!}\left(-\frac{i}{2}\sigma_z\right)\left(-\frac{i\theta}{2}\sigma_z\right)^{n-1} = -\frac{i}{2}\sigma_z\sum_{n=1}^{\infty}\frac{1}{(n-1)!}\left(-\frac{i\theta}{2}\sigma_z\right)^{n-1}.$$

Reindexing the summation by letting $m = n-1$, we recognize the original series:

$$\frac{\partial}{\partial\theta}R_z(\theta) = -\frac{i}{2}\sigma_z\sum_{m=0}^{\infty}\frac{1}{m!}\left(-\frac{i\theta}{2}\sigma_z\right)^{m} = -\frac{i}{2}\sigma_z R_z(\theta).$$

We follow a similar process for the adjoint:

$$\frac{\partial}{\partial\theta}R_z^\dagger(\theta) = \frac{i}{2}R_z^\dagger(\theta)\sigma_z.$$

Now, we can differentiate the linear transformation $\mathcal{T}_\theta(\hat{A})$:

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{T}_\theta(\hat{A}) &= \left(\frac{\partial}{\partial\theta}R_z^\dagger(\theta)\right)\hat{A}R_z(\theta) + R_z^\dagger(\theta)\hat{A}\left(\frac{\partial}{\partial\theta}R_z(\theta)\right)\\ &= \frac{i}{2}R_z^\dagger(\theta)\sigma_z\hat{A}R_z(\theta) - \frac{i}{2}R_z^\dagger(\theta)\hat{A}\sigma_z R_z(\theta)\\ &= \frac{i}{2}R_z^\dagger(\theta)\left[\sigma_z,\hat{A}\right]R_z(\theta).\end{aligned} \tag{5}$$

We use the commutator $[\sigma_z,\hat{A}] = \sigma_z\hat{A} - \hat{A}\sigma_z$. The commutator cannot be evaluated directly, but we can apply an identity involving Pauli operator P_j, rotation operator $U_j = \exp(-i\theta_j P_j/2)$, and arbitrary operator ρ [2]. Since $R_z^\dagger(\frac{\pi}{2}) = R_z(-\frac{\pi}{2})$ we obtain the commutator by taking the difference of circuits prepared with shifted parameters:

$$\begin{aligned}[P_j,\rho] &= i\left(U_j\left(\frac{\pi}{2}\right)\rho U_j^\dagger\left(\frac{\pi}{2}\right) - U_j\left(-\frac{\pi}{2}\right)\rho U_j^\dagger\left(-\frac{\pi}{2}\right)\right),\\ [\sigma_z,\hat{A}] &= -i\left(R_z^\dagger\left(\frac{\pi}{2}\right)\hat{A}R_z\left(\frac{\pi}{2}\right) - R_z^\dagger\left(-\frac{\pi}{2}\right)\hat{A}R_z\left(-\frac{\pi}{2}\right)\right).\end{aligned}$$

Substituting this into our derivative from Eq. 5:

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{T}_\theta(\hat{A}) &= \frac{i}{2}R_z^\dagger(\theta)\left[-i\left(R_z^\dagger\left(\frac{\pi}{2}\right)\hat{A}R_z\left(\frac{\pi}{2}\right) - R_z^\dagger\left(-\frac{\pi}{2}\right)\hat{A}R_z\left(-\frac{\pi}{2}\right)\right)\right]R_z(\theta)\\ &= \frac{1}{2}R_z^\dagger(\theta)\left[R_z^\dagger\left(\frac{\pi}{2}\right)\hat{A}R_z\left(\frac{\pi}{2}\right) - R_z^\dagger\left(-\frac{\pi}{2}\right)\hat{A}R_z\left(-\frac{\pi}{2}\right)\right]R_z(\theta).\end{aligned} \tag{6}$$

When one composes two unitary operators (applying one after the other) the resulting transformation is also unitary. In our case,

$$R_z^\dagger(\theta)R_z^\dagger\left(\pm\frac{\pi}{2}\right) = R_z^\dagger\left(\theta\pm\frac{\pi}{2}\right) \quad \text{and} \quad R_z\left(\pm\frac{\pi}{2}\right)R_z(\theta) = R_z\left(\theta\pm\frac{\pi}{2}\right).$$

Expanding Eq. 6 and applying these properties, we write the derivative as

$$\frac{\partial}{\partial\theta}\mathcal{T}_\theta(\hat{A}) = \frac{1}{2}\left(\mathcal{T}_{\theta+\pi/2}(\hat{A}) - \mathcal{T}_{\theta-\pi/2}(\hat{A})\right).$$

Returning to the quantum circuit function, recall from Eq. 2 that $\frac{\partial}{\partial\theta}f(x;\theta) = \langle x|\frac{\partial}{\partial\theta}\mathcal{T}_\theta(\hat{A})|x\rangle$:

$$\frac{\partial}{\partial\theta}f(x;\theta) = \left\langle x|\frac{1}{2}\left(\mathcal{T}_{\theta+\pi/2}(\hat{A}) - \mathcal{T}_{\theta-\pi/2}(\hat{A})\right)|x\right\rangle = \frac{1}{2}\left(\langle x|\mathcal{T}_{\theta+\pi/2}(\hat{A})|x\rangle - \langle x|\mathcal{T}_{\theta-\pi/2}(\hat{A})|x\rangle\right).$$

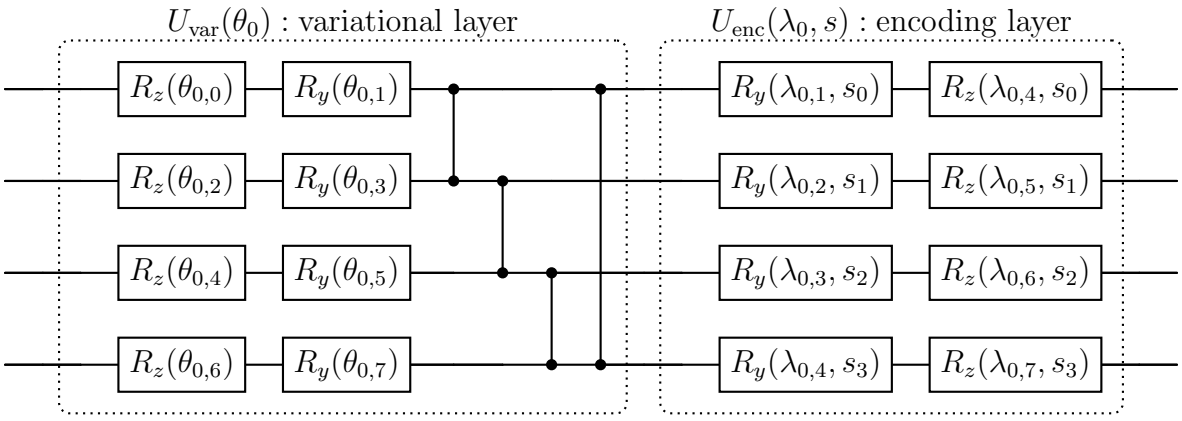
Finally, recall that by definition, $f(x;\theta = \pm\frac{\pi}{2}) = \langle x|\mathcal{T}_{\theta\pm\pi/2}(\hat{A})|x\rangle$. Replacing expectation values with function evaluations at shifted parameters:

$$\nabla_\theta f(x;\theta) = \frac{1}{2}\left[f\left(x;\theta+\frac{\pi}{2}\right) - f\left(x;\theta-\frac{\pi}{2}\right)\right]. \tag{7}$$

Therefore, to compute the gradient of the expectation value of our quantum circuit with respect to parameter θ, one only needs to run the quantum circuit twice and take the difference, with $\alpha = \frac{1}{2}$ and $\beta = \frac{\pi}{2}$. Note that these two measurements yield an exact gradient, and that this process can be applied to $R_x(\theta)$ and $R_y(\theta)$.

2 Parametrized Quantum Policies

2.1 Quantum Circuit Ansatz



Parameterized quantum policies rely on parameterized quantum circuits (PQCs) to map classical input data to measurable outputs. The search space of possible gate arrangements grows exponentially with the number of qubits and circuit depth, so exhaustively searching for an optimal architecture is computationally unfeasible. Instead, we make an educated guess, our "circuit ansatz". Similar to the architecture of a neural network, this only defines the base structure, and the free parameters are later optimized through variation (Eq. 4). The circuit depth is the number of repetitions of these layers and must also be tuned, as increasing depth increases expressivity but also susceptibility to noise and decoherence.

A common ansatz is the Hardware-Efficient Ansatz (HEA), which uses repeated quantum blocks that alternate between variational and data-encoding layers, as illustrated above [5].

Each variational layer $U_{\text{var}}(\theta)$ consists of single-qubit rotations about the z- and y-axes, followed by entangling controlled-Z rates arranged in a circular topology. Data-encoding layers $U_{\text{enc}}(\lambda, s)$ use rotation gates parameterized by both trainable parameters λ and the classical data x. By alternating encoding and variational layers, the circuit "re-uploads" data to circumvent the no-cloning theorem; quantum computers cannot copy data, but classical devices can [6]. The number of times a given layer is repeated is a hyperparameter.

2.2 Policy Selection

In reinforcement learning (RL), agent behavior is guided by a policy $\pi(a|s)$ which maps the observed state s into probabilities for selecting action a. Implementing a parametrized quantum circuit (PQC) involves deciding how quantum states translate into classical policies. For an n-qubit PQC, the circuit is parameterized by a set of angles θ (controlling qubit rotations) and scaling parameters λ (adjusting input encoding). Given a classical input state s, the PQC applies the unitary transformation $|\psi_{s,\theta,\lambda}\rangle = U(s,\theta,\lambda)|0\rangle^{\otimes n}$. Measuring this quantum state provides expectation values needed to define a policy.

Given a set of $|A|$ possible actions in state s, we partition the Hilbert space $\mathcal{H}$ into $|A|$ disjoint subspaces, each corresponding uniquely to an action a. Each action a is associated with a projection operator P_a. By definition, $\sum_a P_a = I$, covering the entire Hilbert space, and $P_a P_{a'} = \delta_{a,a'} P_a$, ensuring subspaces do not overlap unless $a = a'$. We define a simple quantum policy, where the probability of selecting action a is the probability of measuring the quantum state within the subspace corresponding to a [1]:

$$\pi_{\theta,\lambda}(a|s) = \langle P_a \rangle_{s,\theta,\lambda} = \langle \psi_{s,\theta,\lambda} | P_a | \psi_{s,\theta,\lambda} \rangle.$$

Since P_a is a projection, we have $\pi_{\theta,\lambda}(a \mid s) \geq 0$ and $\sum_a \pi_{\theta,\lambda}(a \mid s) = 1$, ensuring the policy is a valid probability distribution. If an action typically yields higher rewards in state s, training adjusts parameters θ and λ so that state $|\psi_{s,\theta,\lambda}\rangle$ increasingly overlaps with the corresponding subspace of $\mathcal{H}$. This makes the policy more likely to choose rewarding actions. However, the above method lacks explicit control over how "greedy" the policy is. Ideally, policies should gradually shift from a near-uniform distribution (promoting exploration) to a sharply peaked distribution (favoring exploitation).

To address this, we first replace the fixed projection operator P_a with a general Hermitian operator O_a, defined as a weighted sum of fixed Hermitian operators:

$$O_a = \sum_i w_{a,i} H_{a,i}, \tag{8}$$

where $w_{a,i}$ are learnable weights, and $H_{a,i}$ are fixed Hermitian operators (such as Pauli operators like σ_x). Let $\phi = \theta, \lambda, w$ summarize all tunable parameters. The expectation value is then:

$$\langle O_a \rangle_{s,\phi} = \langle \psi_{s,\phi} | O_a | \psi_{s,\phi} \rangle = \Big\langle \psi_{s,\phi} \Big| \sum_i w_{a,i} H_{a,i} | \psi_{s,\phi} \Big\rangle.$$

Applying a softmax function to these expectation values defines our second policy [1]:

$$\pi_\phi(a|s) = \frac{e^{\beta \langle O_a \rangle_{s,\phi}}}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}}}. \tag{9}$$

When β is small, differences across exponents are also small, resulting in a nearly uniform action probability distribution. Conversely, large β amplifies expectation value differences, making the policy sharply peak around actions with the highest $\langle O_a \rangle_{s,\phi}$. That is, varying β allows explicit control over the exploration-exploitation balance.

2.3 Estimating Policy Gradients

Having defined how our PQC maps states to action probabilities, we next need a principled way to tweak the rotation and encoding parameters so the policy actually improves. The policy gradient framework provides exactly this: an expression for the gradient of expected return with respect to θ. We begin with an RL setup: an agent interacts with the environment in discrete time steps $t = 0, 1, 2, \ldots$. At each step the agent observes state s_t, selects action $a_t \sim \pi_\theta(\cdot \mid s_t)$, receives reward r_{t+1}, and transitions to s_{t+1}. Our goal is to adjust θ to maximize the expected cumulative reward (return) from the start state, defined as

$$J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)], \quad R(\tau) = \sum_{t=0}^{H-1} \gamma^t r_{t+1}, \quad 0 \leq \gamma \leq 1,$$

where γ is the discount factor and H is the horizon (maximum trajectory length). $J(\theta)$ only depends on θ through the policy π_θ, and we can make its dependence explicit by writing expectations over a complete trajectory $\tau = (s_0, a_0, r_1, \ldots, s_{H-1}, a_{H-1}, r_H)$. First, we define trajectory probability under our policy as $p_\theta(\tau) = \prod_{t=0}^{H-1} \pi_\theta(a_t|s_t)$. When taking the gradient of $J(\theta)$, we pull it inside the expectation to apply the log-derivative trick (chain rule):

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) R(\tau)\, d\tau = \int \nabla_\theta p_\theta(\tau) R(\tau)\, d\tau = \int \nabla_\theta \ln p_\theta(\tau) p_\theta(\tau) R(\tau)\, d\tau.$$

The environment transition probabilities are independent of θ, so from our definition of trajectory probability, we have

$$\ln p_\theta(\tau) = \ln \left(\prod_{t=0}^{H-1} \pi_\theta(a_t|s_t) \right) = \sum_{t=0}^{H-1} \ln \pi_\theta(a_t|s_t),$$
$$\nabla_\theta \ln p_\theta(\tau) = \sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t|s_t).$$

Substituting this back into our integral, we recognize an expectation over trajectories $\tau \sim \pi_\theta$, yielding:

$$\nabla_\theta J(\theta) = \int \left(\sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t|s_t) \right) p_\theta(\tau) R(\tau)\, d\tau = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t|s_t) R(\tau) \right]. \quad (10)$$

We arrive at the policy gradient theorem, expressing the gradient of the expected return purely in terms of quantities we can sample (specifically, the cumulative rewards from a trajectory and the gradients of the policy probabilities at each step). As with any expectation, it

can be estimated with a sample mean. By collecting a set of trajectories $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_N\}$ for policy π_θ, an unbiased estimator for the gradient is:

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} \sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t|s_t) R(\tau) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t^{(i)}|s_t^{(i)}) R(\tau^{(i)}).$$

Note that one could simply stop here, set some learning rate α, and implement stochastic gradient ascent with $\theta_{i+1} \leftarrow \theta_i + \alpha \widehat{\nabla_\theta J(\theta)}$, but the high variance of gradient estimates will make the process unreliable [7]. We mitigate this by introducing a baseline term to reduce variance. Since

$$\mathbb{E}_{\pi_\theta}\big[b(s_t)\, \nabla_\theta \ln \pi_\theta(a_t|s_t)\big] = 0,$$

subtracting any state-dependent function $b(s_t)$ will not bias our gradient estimate [8]. A common choice for this baseline is the on-policy value function $V_{\pi_\theta}(s_t)$, defined as the expected return for an agent starting in state s_t that acts according to policy π_θ thereafter. Also, our previous policy-gradient expression uses the full-trajectory return $R(\tau)$, granting credit even for rewards received before an action. To guarantee that each action is reinforced only by its consequences, we replace $R(\tau)$ with the reward-to-go at time t, i.e. the remaining cumulative reward:

$$G_t = \sum_{k=0}^{H-1-t} \gamma^k r_{t+k+1} = \sum_{j=1}^{H-t} \gamma^{j-1} r_{t+j}. \tag{11}$$

Incorporating these variance-reducing updates into Eq. 10, the gradient becomes:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t|s_t) \overbrace{\Big(G_t - V_{\pi_\theta}(s_t)\Big)}^{\text{previously R}(\tau)}\right].$$

Subtracting the value function turns each raw reward-to-go into a measure of advantage: how much better (or worse) the actual future return is compared to the policy's own estimate. Our baseline-improved Monte Carlo estimator over many trajectories takes the final form:

$$\widehat{\nabla_\theta J(\theta)} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \nabla_\theta \ln \pi_\theta(a_t^{(i)}|s_t^{(i)}) \Big(G_t^{(i)} - V_{\pi_\theta}(s_t^{(i)})\Big). \tag{12}$$

2.4 Learning Algorithm

To compute policy updates according to Eq. 12, we need analytical expressions for the log-policy $\nabla_\theta \ln \pi_\theta(a|s)$, where policy $\pi_\theta(a|s)$ is defined in Eq. 9:

$$\begin{aligned}
\nabla_\theta \ln \pi_\theta &= \nabla_\theta \ln \left(\frac{e^{\beta \langle O_a \rangle_{s,\phi}}}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}}} \right) \\
&= \beta \, \nabla_\theta \langle O_a \rangle_{s,\phi} - \nabla_\theta \ln \sum_{a'} e^{\beta \langle O_a \rangle_{s,\phi}} \\
&= \beta \, \nabla_\theta \langle O_a \rangle_{s,\phi} - \frac{1}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}}} \sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}} \beta \, \nabla_\theta \langle O_{a'} \rangle_{s,\phi} \\
&= \beta \Bigg(\nabla_\theta \langle O_a \rangle_{s,\phi} - \sum_{a'} \underbrace{\frac{e^{\beta \langle O_a \rangle_{s,\phi}}}{\sum_b e^{\beta \langle O_b \rangle_{s,\phi}}}}_{\pi_\theta(a'|s)} \nabla_\theta \langle O_{a'} \rangle_{s,\phi} \Bigg).
\end{aligned}$$

In the last line, we recognize that $\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}}$ is constant for action a', so we pull it into the summation to recover the policy-weighted sum. The final softmax policy gradient is

$$\nabla_\theta \ln \pi_\theta = \beta \left(\nabla_\theta \langle O_a \rangle_{s,\phi} - \sum_{a'} \pi_\theta(a'|s) \nabla_\theta \langle O_{a'} \rangle_{s,\phi} \right). \tag{13}$$

Recall that $\phi = \theta, \lambda, w$ summarizes all tunable parameters, and that from Eq. 8, $O_a = \sum_i w_{a,i} H_{a,i}$. By the linearity of expectation, partial derivatives with respect to observable weights are given by

$$\frac{\partial}{\partial w_{a,i}} \langle O_a \rangle = \frac{\partial}{\partial w_{a,i}} \Big\langle \sum_j w_{a,j} H_{a,j} \Big\rangle = \frac{\partial}{\partial w_{a,i}} \sum_j w_{a,j} \langle H_{a,j} \rangle = \langle H_{a,i} \rangle = \big\langle \psi_{s,\phi} | H_{a,i} | \psi_{s,\phi} \big\rangle.$$

Partial derivatives with respect to the tunable rotation angles θ and λ can be efficiently computed with the parameter shift rule (Eq. 7). We first define the circuit function $f(x; \theta, \lambda) \mapsto \langle \psi_{s,\theta,\lambda} | O_a | \psi_{s,\theta,\lambda} \rangle = \langle O_a \rangle_{s,\theta,\lambda}$. Then for each variational parameter θ_i,

$$\frac{\partial}{\partial \theta_i} \langle O_a \rangle_{s,\theta,\lambda} = \frac{1}{2} \left[f\left(x; \theta + \frac{\pi}{2} e_i, \lambda\right) - f\left(x; \theta - \frac{\pi}{2} e_i, \lambda\right) \right],$$

where e_i is the unit vector with 1 in the i-th position, shifting only θ_i. Note that in the encoding layers, each gate's actual rotation angle is $s_j \lambda_{i,j}$, where s_j is the jth feature of the classical input. The parameter shift rule applies to this product, not directly to $\lambda_{i,j}$. So, for each encoding parameter $\lambda_{i,j}$, we apply the chain rule $\frac{\partial}{\partial \lambda_{i,j}} = \frac{\partial s_j \lambda_{i,j}}{\partial \lambda_{i,j}} \frac{\partial}{\partial s_j \lambda_{i,j}}$:

$$\frac{\partial}{\partial \lambda_{i,j}} \langle O_a \rangle_{s,\theta,\lambda} = \frac{s_j}{2} \left[f\left(x; \theta, \lambda + \frac{\pi}{2 s_j} e_{i,j}\right) - f\left(x; \theta, \lambda - \frac{\pi}{2 s_j} e_{i,j}\right) \right],$$

where $e_{i,j}$ is the unit vector with 1 in the (i,j)-th slot, shifting only $\lambda_{i,j}$. Each $\pm\pi/2$ shift is applied to the gate in the direction of its defined rotation axis.

Having defined a softmax PQC policy to select actions (Eq. 9) to select actions, a reward-to-go function to correct cumulative rewards (Eq. 11), a log-policy gradient that evaluates the gradient of the expected returns with respect to tunable circuit parameters (Eq. 13), and Monte Carlo estimator that uses samples from trajectories to estimate the policy gradient (Eq. 12), we can finally assemble our learning algorithm, an extension of the classical REINFORCE algorithm for PQCs [1]. We outline the steps involved in Algorithm 1. Note that in previous sections, we explored a policy π_θ parameterized by vector θ, but since $\phi = (\theta, \lambda, w)$ now represents the complete set of trainable parameters, the policy is denoted as π_ϕ.

Algorithm 1: REINFORCE for Re-Uploading PQC

Input: PQC policy $\pi_\phi(a|s) = \frac{e^{\beta \langle O_a \rangle_{s,\phi}}}{\sum_{a'} e^{\beta \langle O_{a'} \rangle_{s,\phi}}}$; learning rate $\alpha > 0$; discount factor $\gamma \in [0, 1]$; number of trajectories N; horizon H.

Randomly initialize $\phi = (\theta, \lambda, w)$;
while *not converged* **do**
  Collect N trajectories $\{\tau^{(i)} \sim \pi_\phi\}_{i=1}^N$, $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_1^{(i)}, \ldots, s_{H-1}^{(i)}, a_{H-1}^{(i)}, r_H^{(i)})$;
  for $i = 1$ **to** N **do**
    for $t = 0$ **to** $H - 1$ **do**
      Compute reward-to-go $G_t^{(i)} = \sum_{k=0}^{H-1-t} \gamma^k \, r_{t+k+1}^{(i)}$;
      Compute log-policy gradient $\nabla_\phi \ln \pi_\phi(a_t^{(i)} \,|\, s_t^{(i)})$;
    end
  end
  Compute gradient estimate $\widehat{\nabla_\phi J(\phi)} = \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{H-1} \nabla_\phi \ln \pi_\phi(a_t^{(i)} \,|\, s_t^{(i)}) \; G_t^{(i)}$;
  Update parameters $\phi \leftarrow \phi + \alpha \, \widehat{\nabla_\phi J(\phi)}$;
end

The above REINFORCE-based learning algorithm is implemented in the accompanying repository using PennyLane and PyTorch, and is ready for application to various Gymnasium environments.

References

[1] Sofiene Jerbi, Casper Gyurik, Simon C. Marshall, Hans J. Briegel, and Vedran Dunjko. Parametrized quantum policies for reinforcement learning, 2021.

[2] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii. Quantum circuit learning. *Physical Review A*, 98(3), September 2018.

[3] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), March 2019.

[4] PennyLane Team. Glossary: Parameter shift. `https://pennylane.ai/qml/glossary/parameter_shift`, 2025. Accessed: April 09, 2025.

[5] Xiaoyu Guo, Takahiro Muta, and Jianjun Zhao. Quantum circuit ansatz: Patterns of abstraction and reuse of quantum algorithm design, 2024.

[6] Adrián Pérez-Salinas, Alba Cervera-Lierta, Elies Gil-Fuster, and José I. Latorre. Data re-uploading for a universal quantum classifier. *Quantum*, 4:226, February 2020.

[7] Evan Greensmith, Peter Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2020.