University of Minho
School of Engineering

Data Encoding in Variational Q-Learning

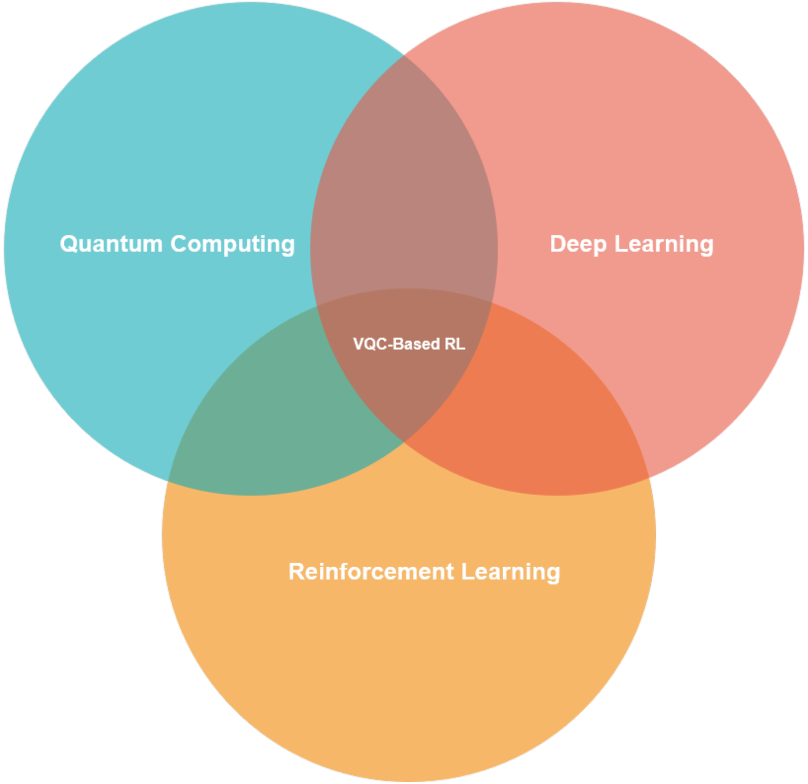
Rodrigo Coelho

Supervisors: Prof. Luís Paulo Santos and André Sequeira

April 20, 2023

Contents

Contextualization



Reinforcement Learning

The Agent-Environment Interaction

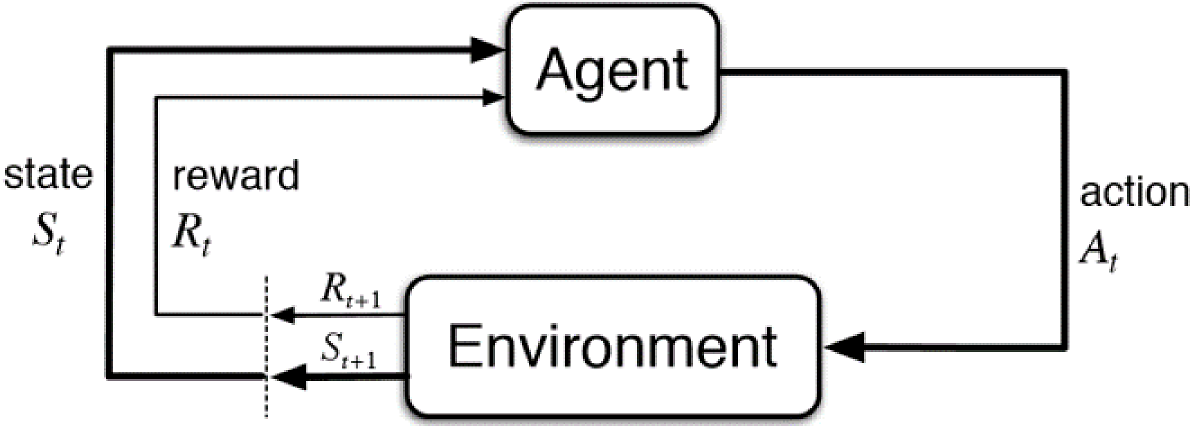


Figure 1: The agent-environment interaction [1]

The agent's goal is to maximize the sum of all the rewards during a sequence of time steps.

Examples of Reinforcement Learning

- Make a humanoid robot walk
- Manage an investment portfolio
- Fly a drone
- Manage a power station
- Defeat the World Champion at Chess
- Play many games better than humans

Markov Decision Process

- A state is considered a Markov state if it captures all relevant information from the past. Once the state is known, the history may be thrown away.
- An MDP is a sequence of Markov states.
- MDPs formally describe an environment for Reinforcement Learning (RL) where the environment is *fully observable*.

Markov Decision Process

A Markov Decision Process is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of Markov states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{P}$ is a state transition probability matrix, $P^a_{ss\prime} = \mathbb{P}[S_{t+1} = s\prime | S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $R^a_s = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$

Return, Policy and Value-Function

The return G_t is the total discounted reward from time-step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

A policy π is a distribution over actions given states

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

The state-value function is the expected return starting from state s and then following policy π

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

Optimality

The optimal state-value function $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_{\pi} v_\pi(s)$$

For any MDP, there exists an optimal policy π_*, that is better than or equal to all other policies $\pi_* \geq \pi, \forall \pi$.

Policy-Based RL

- A policy-based algorithm seeks to learn the optimal policy directly
- The policy is parametrized $\pi(a|s, \theta)$ and the goal is to find parameters θ such that the resulting policy is optimal
- This is done by maximizing a performance measure $J(\theta)$

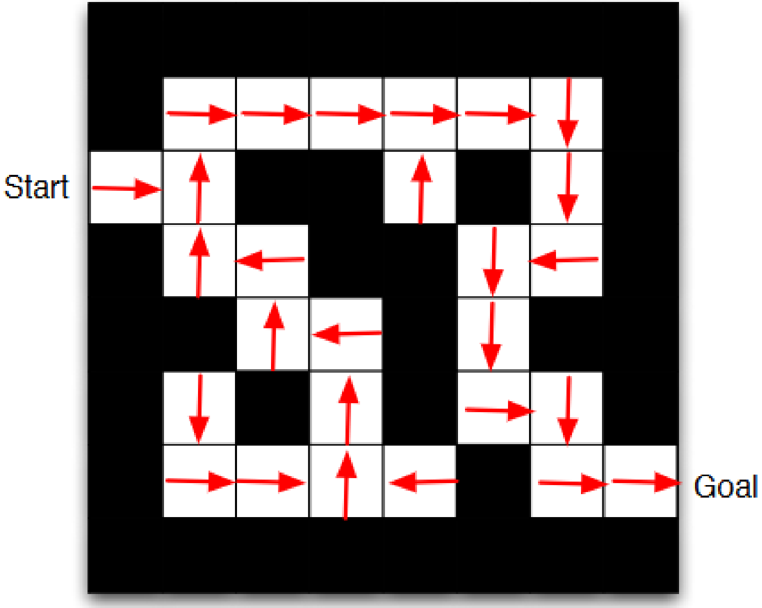


Figure 2: Image from [2]

Value-based RL

- In a value-based algorithm, a value-function is learned and the policy is then implicitly given by this function
- The agent will always pick the action which yields the highest expected return according to the value-function

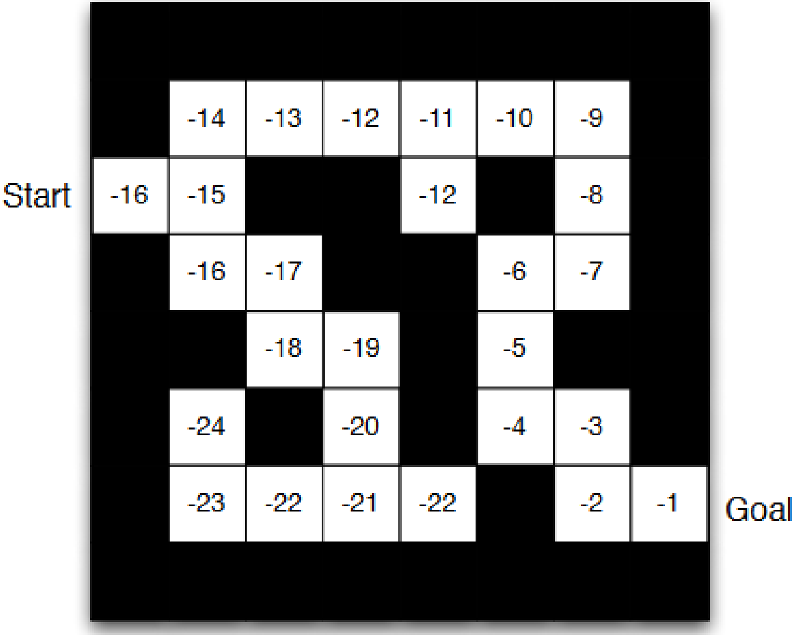


Figure 3: Image from [2]

Action-Value Function

The action value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

It can be decomposed into immediate reward plus discounted reward of successor state-action pair

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \end{aligned}$$

Q-Learning

The idea behind Q-Learning is to learn the optimal action-value function and, consequently, derive the optimal policy by maximizing over $q_*(s, a)$

$$\pi_*(a, s) = \text{argmax}_a q_*(s, a)$$

To ensure sufficient exploration, a ϵ-greedy policy is used

$$a_t = \begin{cases} \text{argmax}_a q(s_t, a), & \text{with probability } 1 - \epsilon \\ \text{a random action}, & \text{with probability } \epsilon \end{cases}$$

The Q-values are updated by the following rule,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

Tabular Reinforcement Learning

- So far, we have assumed that the value-functions are represented by lookup tables
- Problem with large MDPs (complex environments with large state and/or action spaces)
- Go $\rightarrow 10^{170}$ states
- Agents need to generalize and come up with intelligent decisions!

Deep Reinforcement Learning

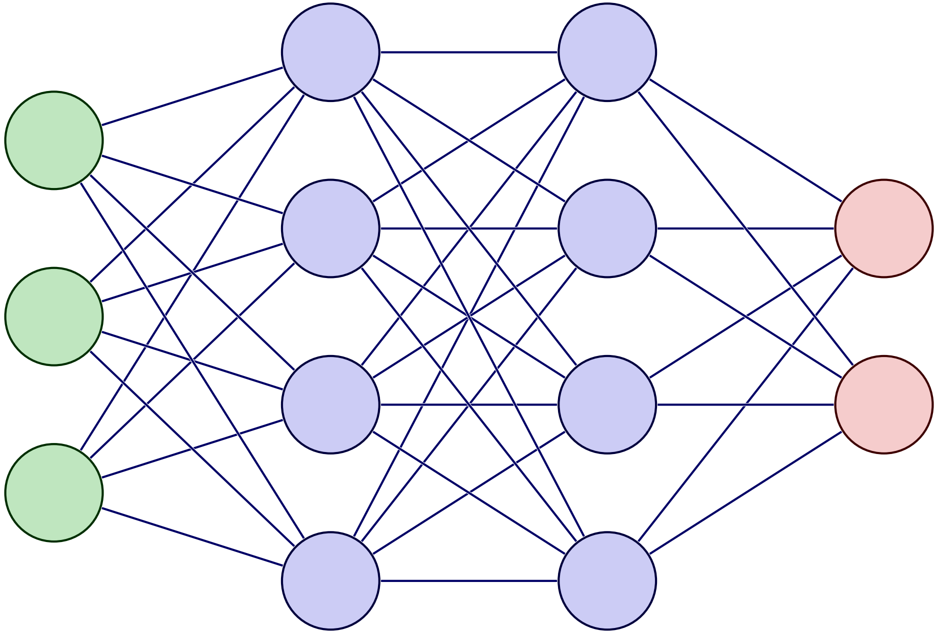
Function Approximators

- Solution for large MDP's:
  - Estimate value function with function approximation:

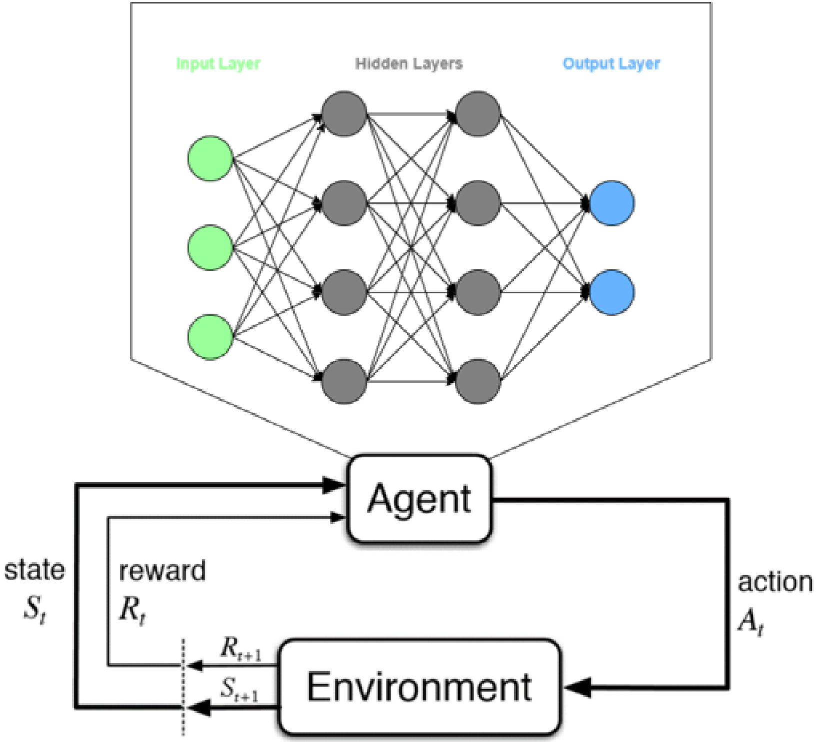
$$\hat{q}(s, a, w) \approx q_{\pi}(s, a)$$

- Non-linear Function Approximators → Neural Networks
- But there are others...

Deep Neural Networks



Deep Reinforcement Learning



Deep Q-Network (DQN)

DQN uses an **experience replay** and a **target network**

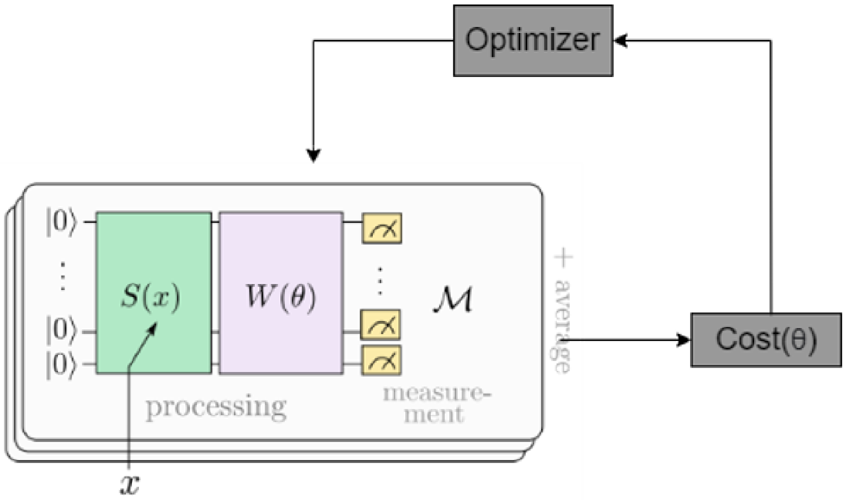
- Take action a_t according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions $(s, a, r, s\prime)$ from $\mathcal{D}$
- Compute Q-learning targets w.r.t old, fixed parameters w^-
- Optimize MSE (or some other cost function) between Q-network and Q-learning targets

$$\mathcal{L}_i(w_i) = \mathbb{E}_{s,a,r,s\prime \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a\prime} Q(s\prime, a\prime; w_i^-) - Q(s, a; w_i)\right)^2\right]$$

Variational Q-Learning

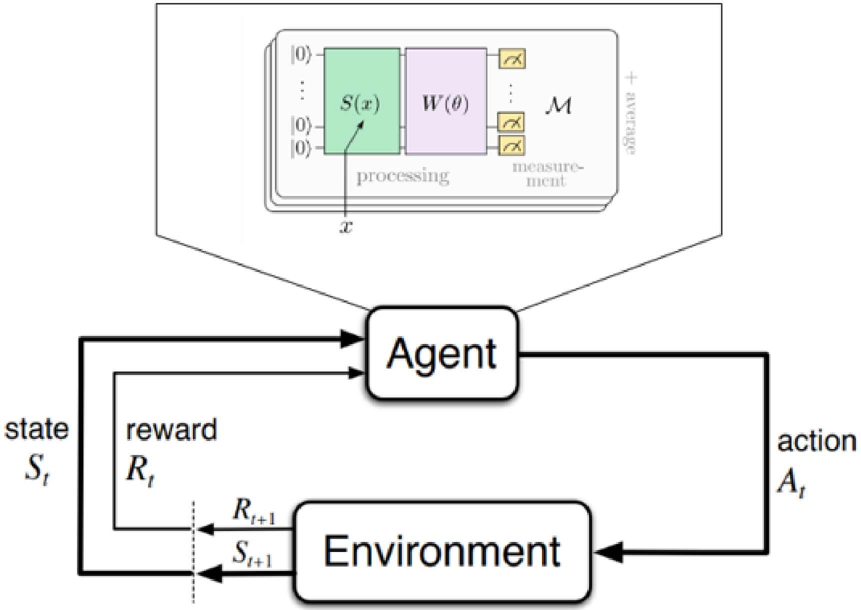
Variational Quantum Circuits (VQCs)

- VQC's are quantum circuits that depend on free parameters. They consist of three ingredients:
  - Preparation of an initial state (data-encoding)
  - A quantum circuit $W(\theta)$
  - Measurement of an observable at the output
- They are trained by a classical optimizer
- They are suitable for NISQ devices



VQC-based RL

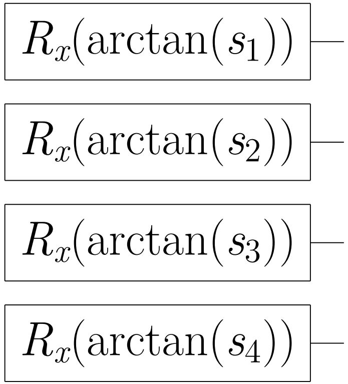
- The same way Neural Networks can be used as function approximators in RL, so can VQCs
- The result is a hybrid quantum-classical algorithm
- It can and has been used for both policy-based [3] [4] and value-based [5] [6] algorithms successfully



Data Encoding

- **Continuous encoding**: Each component/feature x of an input state vector $\mathbf{x}$ is scaled to $x\prime = \arctan(x) \in [-\pi/2, \pi/2]$ and then rotated in the X direction by the angles $x\prime$
- Number of qubits = number of components

Assuming a state $s = [s_1, s_2, s_3, s_4]$



Q-Values and Output Scaling

- The Q-values of our quantum agent are computed as the expectation values of a VQC that is fed a state s as

$$Q(s,a) = \left\langle 0^{\otimes n} \right| U_\theta^\dagger(s) O_a U_\theta(s) \left| 0^{\otimes n} \right\rangle$$

- The model outputs a vector including Q-values for every possible action (O_a)
- Problem: Q-values can have any arbitrary range but expectation values are bounded.
  - Solution: Multiply the expectation values by a classical trainable weight such that the Q-values become

$$Q(s,a) = \left\langle 0^{\otimes n} \right| U_\theta^\dagger(s) O_a U_\theta(s) \left| 0^{\otimes n} \right\rangle \cdot \omega_{O_a}$$

Data Re-uploading

- The output of a VQC can be written as a Partial Fourier Series in the data where the frequencies are given by the data encoding gates and the coefficients by the rest of the circuit

$$f(x) = \sum_{\omega \in \Omega} c_\omega e^{i\omega}$$

- By repeating simple data encoding gates multiple times, we can reach a higher frequency spectra.

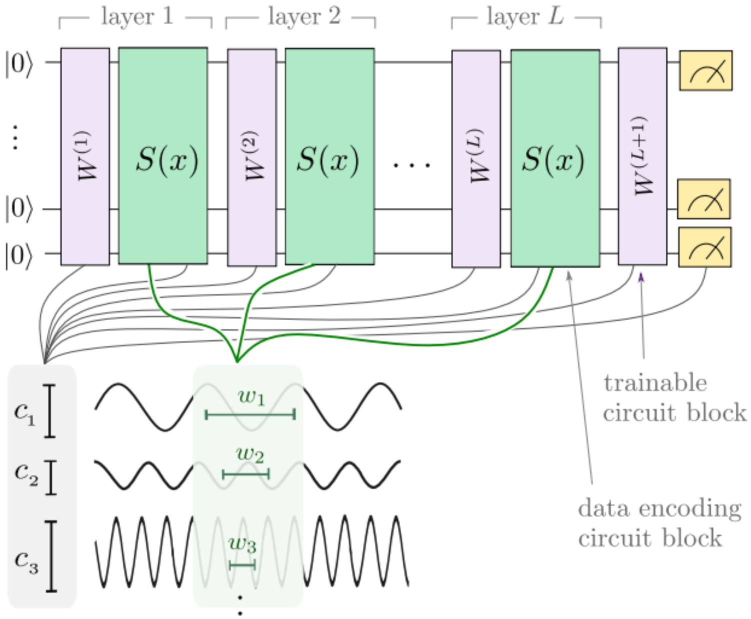


Figure 4: Image from [7]

Input Scaling

Multiplying the inputs by trainable weights allows for:

- Frequency matching between the output of the quantum model and the target function
- A frequency spectrum with access to more frequencies → increased expressivity of the quantum model

Assuming a state $s = [s_1, s_2, s_3, s_4]$

$$R_x(\arctan(s_1 * \lambda_1))$$

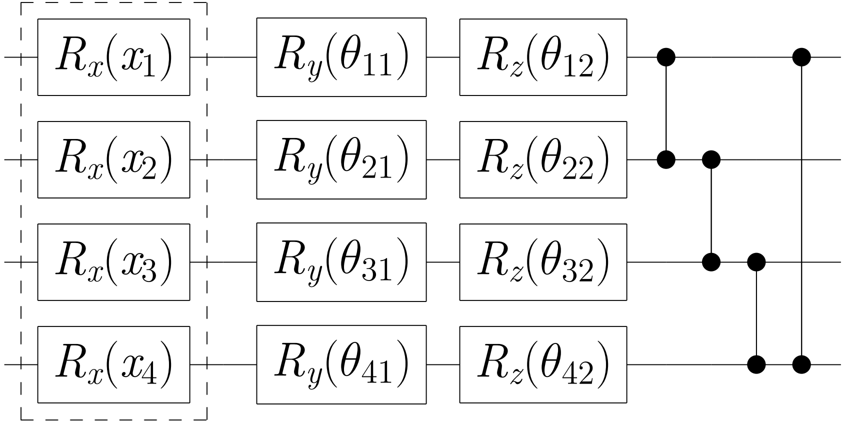
$$R_x(\arctan(s_2 * \lambda_2))$$

$$R_x(\arctan(s_3 * \lambda_3))$$

$$R_x(\arctan(s_4 * \lambda_4))$$

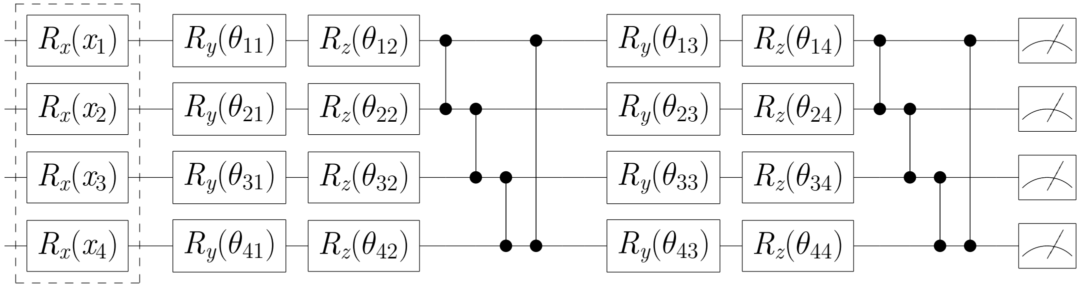
Circuit Architecture

- If Data Re-uploading is being used, the whole circuit on the right is repeated in each layer. Otherwise, just the part that is not surrounded by the dashes is repeated.

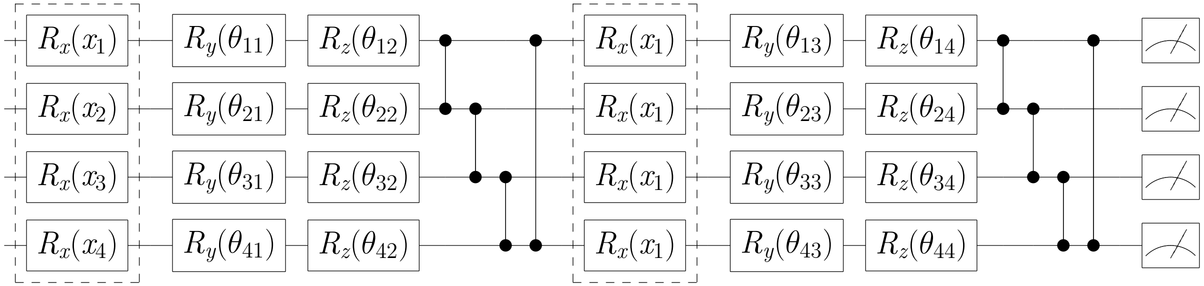


Circuit Architecture

Circuit with two layers and no data re-uploading:

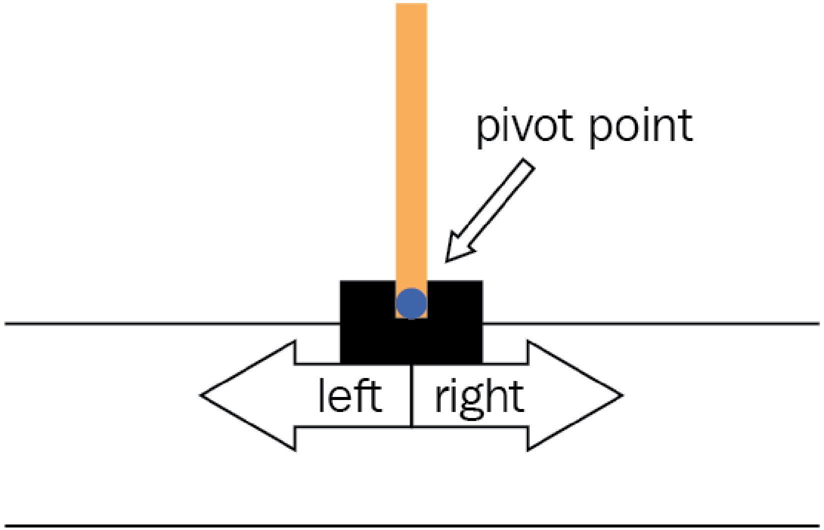


Circuit with two layers and data re-uploading:



Environment - CartPole-v0

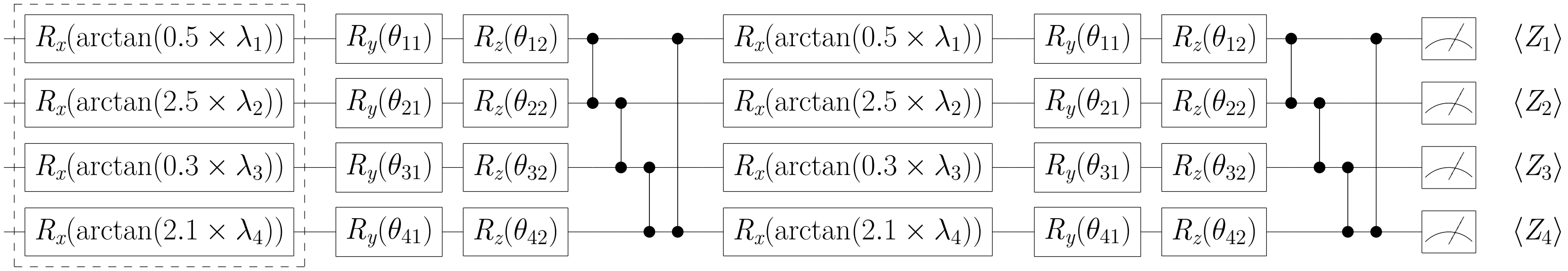
- Observation Space:
  - 1 - Cart Position
  - 2 - Cart Velocity
  - 3 - Pole Angle
  - 4 - Pole Angular Velocity
- Action Space:
  - Push cart to the left
  - Push cart to the right



The model in action

Let's imagine the model, which is a VQC with Data Re-uploading and two layers, interacts with the environment and observes state s:

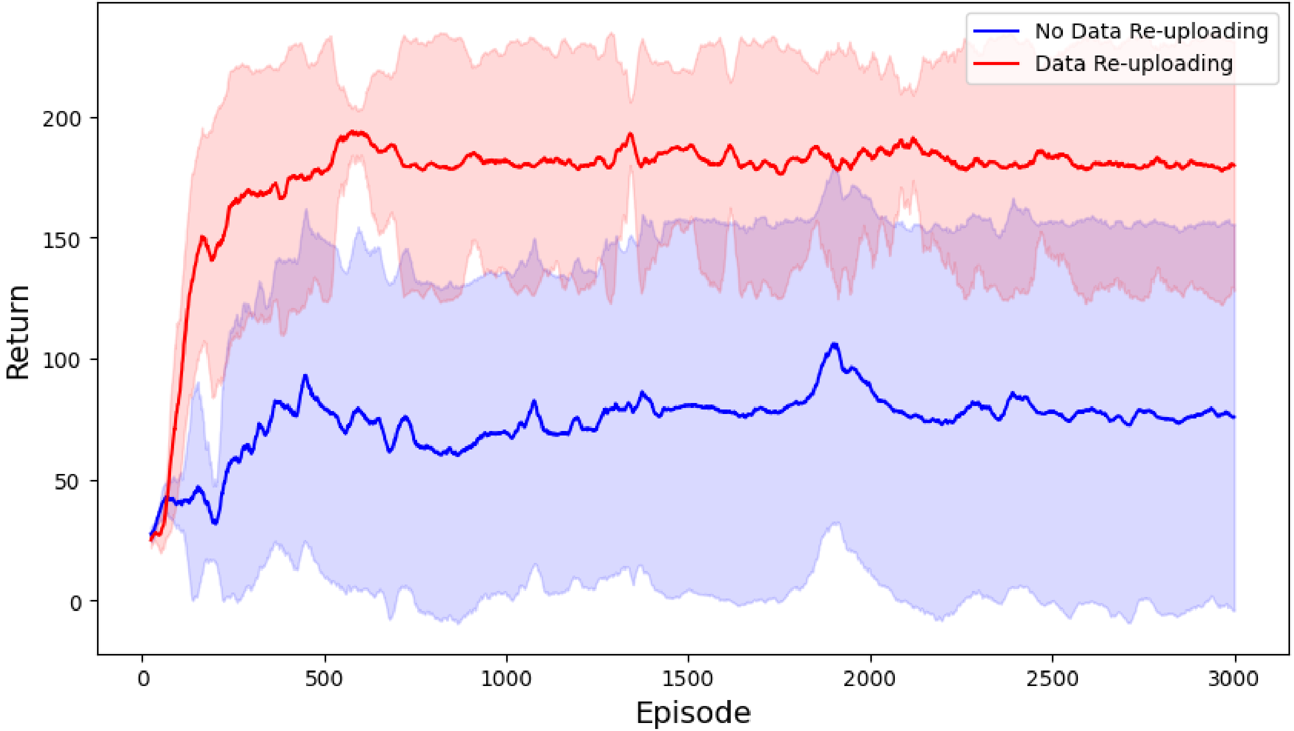
$$s = [0.5, 2.5, 0.3, 2.1]$$



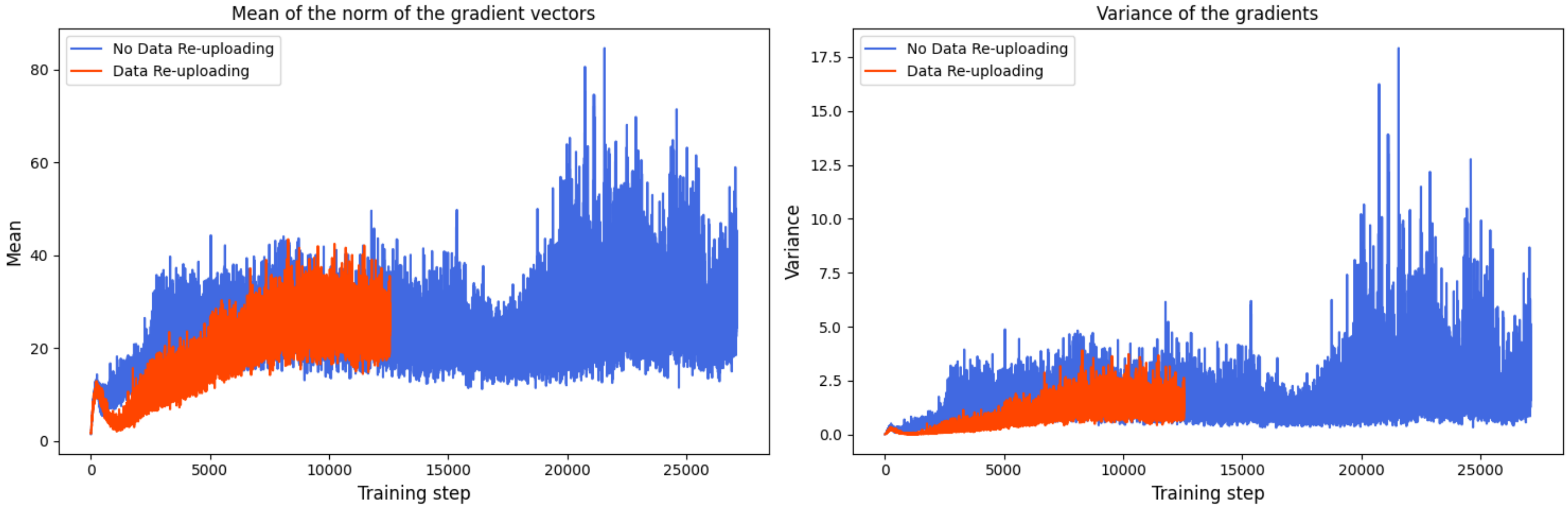
$$Q(s, \text{left}) = \langle Z_1 Z_2 \rangle \times \omega_1 = 70$$
$$Q(s, \text{right}) = \langle Z_3 Z_4 \rangle \times \omega_2 = 100$$

Results

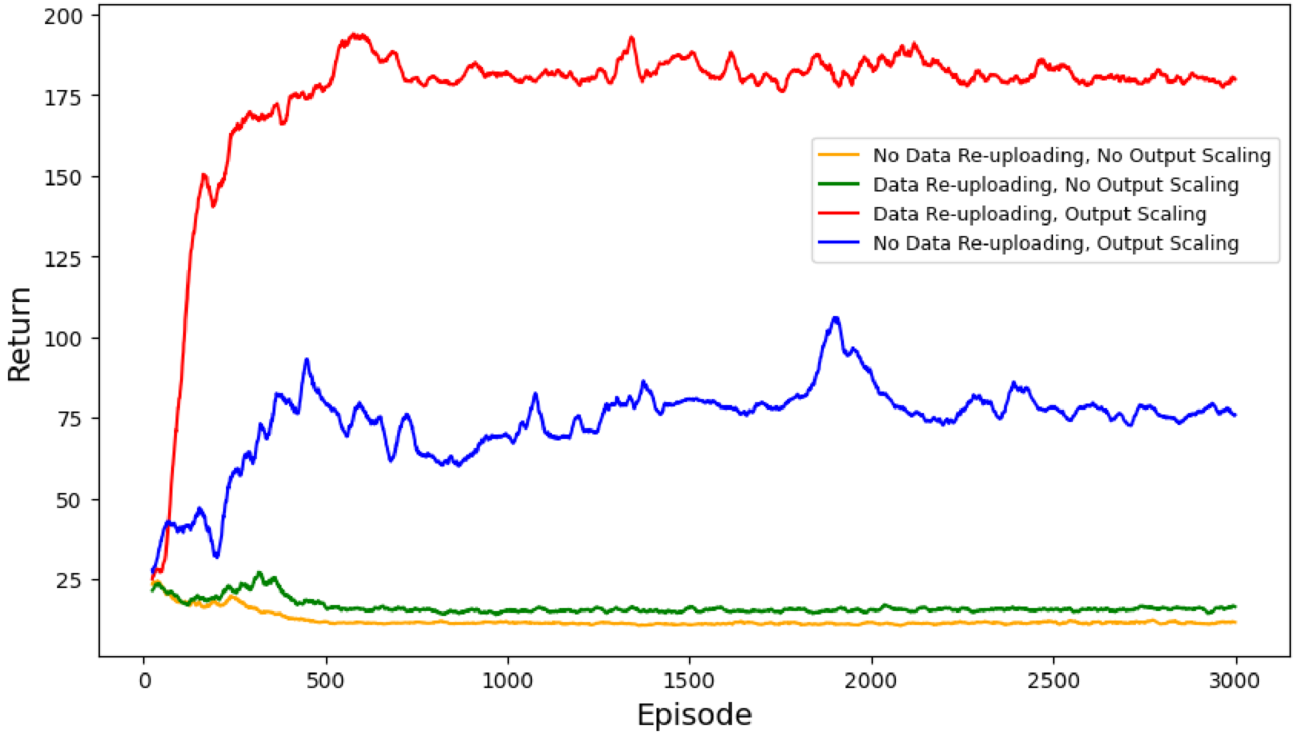
The effect of Data Re-uploading



Gradients


Mean of the norm of the gradient vectors
No Data Re-uploading
Data Re-uploading
Mean
0
20
40
60
80
0
5000
10000
15000
20000
25000
Training step
Variance of the gradients
No Data Re-uploading
Data Re-uploading
Variance
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
0
5000
10000
15000
20000
25000
Training step

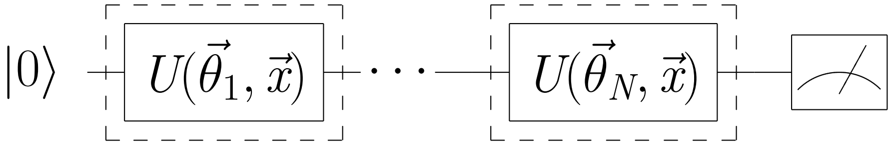
The effect of Output Scaling



Universal Quantum Classifier (UQC)

- The Universal Quantum Classifier (UQC) allows for an arbitrary number of qubits to encode the input
- Even one qubit is enough

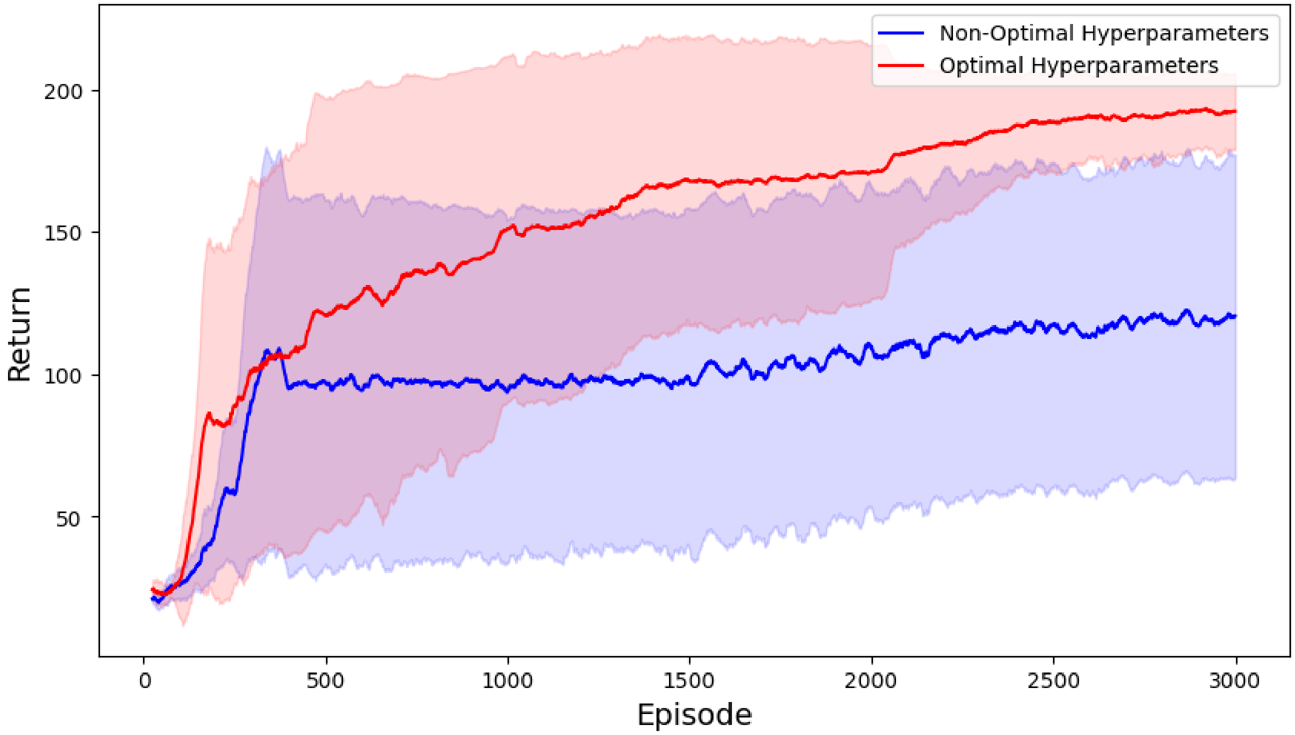
A UQC with one qubit and N layers:



Where each processing gate U is given by:

$$U^{UAT}(\vec{x}; \vec{\omega}, \alpha, \varphi) = R_y(2\varphi) R_z(2\vec{\omega} \cdot \vec{x} + 2\alpha)$$

UQC on CartPole



Conclusions

Conclusions

- Data Re-uploading is extremely important as it increases the expressivity of the quantum circuit
  - However, it seems like it leads to smaller gradients...
- Output scaling is also essential since it scales the expectation values to match the Q-values of the environment
- One qubit with data re-uploading is enough to solve CartPole

Future Work

- Finding an optimal set of hyperparameters for the UQC (model seems highly unstable)
- Studying the Hessian Matrix to further confirm the claim that data re-uploading decreases the trainability of the models
- Experimenting the UQC with more qubits
- Testing on different environments

Discussion

INESCTEC

References I

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] David Silver (Deepmind), *Introduction to reinforcement learning with david silver*, URL: https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver, 2015.

[3] S. Jerbi, C. Gyurik, S. Marshall, H. J. Briegel, and V. Dunjko, "Variational quantum policies for reinforcement learning," *arXiv preprint arXiv:2103.05577*, 2021.

References II

[4] A. Sequeira, L. P. Santos, and L. S. Barbosa, "Variational quantum policy gradients with an application to quantum control," *arXiv preprint arXiv:2203.10591*, 2022.

[5] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, "Variational quantum circuits for deep reinforcement learning," *IEEE Access*, vol. 8, pp. 141 007–141 024, 2020.

[6] A. Skolik, S. Jerbi, and V. Dunjko, "Quantum agents in the gym: A variational quantum algorithm for deep q-learning," *Quantum*, vol. 6, p. 720, 2022.

References III

[7] M. Schuld, R. Sweke, and J. J. Meyer, "Effect of data encoding on the expressive power of variational quantum-machine-learning models," *Physical Review A*, vol. 103, no. 3, p. 032 430, 2021.

[8] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[9] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, "Data re-uploading for a universal quantum classifier," *Quantum*, vol. 4, p. 226, 2020.