# Prospects of quantum computing approach to reinforcement learning

**University of Turin and NTTDATA**
**Matteo Conterno**
July 5, 2022


**Relator:** **prof.Carlini Alberto**
**Counter-rapporteur :** **prof. Castellani Leonardo**

# Abstract

Reinforcement learning is one of three main techniques that allows a model to learn, notably focusing on creating an optimal agent able to reach an objective by interacting with an environment. This thesis tries to analyze the possible potentials and advantages that would derive from using quantum circuits with neural networks. To examine and explain how it is possible to create hybrid algorithms that exploit the improvements of the classical and quantum algorithms.
The Cartpole environment tested uses the Deep Q-Network algorithm and is compared with the quantum version to see what kind of advantage is present. After demonstrating the quantum advantage on the Cartpole, a more complex environment with an industrial application, the robotic arm, is tested using another kind of algorithm called Soft-Actor Critic. Differently from the DQN version, it requires multiple components with different purposes increasing the possible configurations that needs to be tested.
For this reason, multiple configurations were run, such as one where only a single component has the quantum variation and one with all components quantum variated.Finally, confronting the quantum variation and all "classical" models, a clear advantage can be extrapolated, showing possible future applications in the industry and other fields.

# Contents

# Acronyms

**A2C** Actor-Critic. 14

**AI** Artificial Intelligence. 5

**DDPG** Deep Deterministic Policy Gradient. 49

**DQN** Deep Q-Network. 13, 15, 36, 48

**DRL** Deep Reinforcement Learning. 5, 12, 14, 48, 49

**i.i.d.** indipendent identical distributed. 12

**MDP** Markov Decision Process. 6, 7, 12

**NN** Neural Networks. 5, 10–12, 23, 29, 32, 36, 38, 39, 44, 48

**PPO** Proximal Policy Optimization. 14, 49

**QC** Quantum Computing. 5

**QRL** Quantum Reinforcement Learning. 5

**RL** Reinforcement Learning. 23

**SAC** Soft-Actor Critic. 14, 36, 48

**VQA** Variational Quantum Algorithm. 5, 6, 25–27, 29–32, 34, 36, 48, 49

# 1 Introduction to Deep Reinforcement Learning and Quantum Computing

This section of the thesis is created so that it can give an introduction to the fields of Artificial Intelligence(AI) specifically to Deep Reinforcement Learning(DRL) and the basis of Quantum Computing(QC) in order to understand afterward the fusion of these two different fields on Quantum Reinforcement Learning(QRL). If skilled in these fields feel free to skip to the next section, many concepts are taken from the books [1] and [2].

## 1.1 Approaches of learning

Currently any kind of AI requires the following components to learn:

- Data
- Model
- Approach of learning

Data is necessary for every AI model, quantity and quality can heavily influence the ability to correctly and efficiently reach his goal. The model is an algorithm that, given some data and a predefined objective, tries to complete its task using some learning approach. To comprehend if the model has learned correctly, it will be later tested on unseen data and evaluated to understand if it can replicate the performances given a similar dataset.
The approach of learning specifies how the model can learn to complete his task. There are three major ways:

- **Supervised learning**: the data is labelled, meaning the possibility to estimate when the model is incorrect or correct.
- **Unspervised learning**: the data is unlabelled with the model that must uncover some pattern, meaning that it can't be easily estimated when the model is correct or incorrect.
- **Reinforcement learning**: an agent interacts with the environment to become the most optimal agent to complete the task.

This thesis will focalize mainly on the reinforcement learning approach and especially on the Deep Reinforcement Learning (DRL) which uses Neural Networks (NN) to define the most optimal agent. Studying what advantage is obtained via a quantum algorithm such as Variational Quantum Algorithm (VQA) instead of Neural Networks (NN).
Furthermore, it was demonstrated that VQA and NN share some similarities

and properties. The main drawback is that when a VQA training is conducted on a classical device, there is a major overhead of time due to the simulation of the quantum circuit, also the number of qubits that can be simulated is limited.

## 1.2 Reinforcement Learning

As said earlier reinforcement learning approach consists of creating the most optimal agent capable to execute a predefined task by interacting with the environment and taking some action. Questions arise: how do determine if an action is good or bad? How to model a dynamic environment? The answer is to use a statistical model called Markov Decision Process (MDP).

### 1.2.1 Markov Decision Process

To model a dynamic environment where action can influence the system it is necessary to use the Markov Decision Process (MDP) which is an extension of Markov chains. A Markov chain is a stochastic model that can describe a sequence of possible events that satisfy the Markov property, which is each event depends only on the previous case.
It is necessary to note that an MDP is based on Markov Chains, which model the states and time. Time can be defined as continuous or discretized for this model. MDP is an extension of Markov Chains because the agent can influence the state of the environment and outcome, so a framework is necessary to define his decisions and their consequences. An MDP is defined as a 4-tuple containing the following elements:

- $S$ : set of states
- $A$ : set of actions
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s)$ : is the transition probability of going from state $s$ to $s'$ by taking an action $a$
- $R_a(s, s')$ is the immediate reward obtained by transitioning from state $s$ to $s'$ by action $a$

The difference with a Markov chains is the presence of $P_a(s, s')$ and $R_a(s, s')$ which are necessary for the decision process, to see an example graph of and MDP see Figure 1.
Interaction of agent and environment is classified by time, discrete-time steps imply viewing it as separate points in time uniquely defined and with a single state value that can be associated. Continous time-steps involve viewing every step as continued points having a single state value associated. The sequence of observation over time forms a chain of states called **history**,

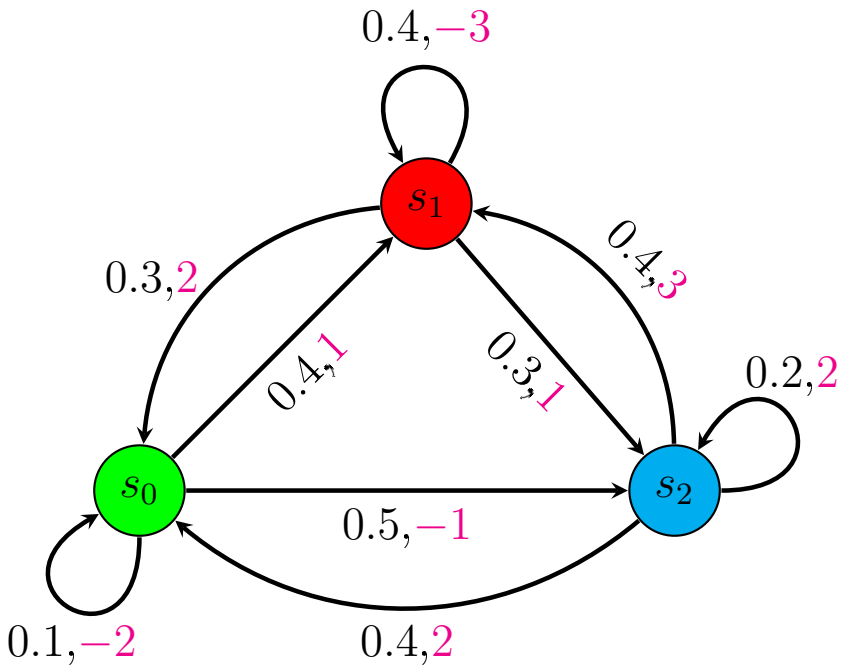



Figure 1: Markov decision process graph, transition probability is black and reward is in magenta.

which will be crucial because used to define the transition probability to model the interaction with the environment. To include the reward element of an MDP accumulated from present and future a new quantity needs to be defined called **return**:

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

The $\gamma$ is a variable called discount factor with values limited inside the range of 0 and 1 with extremes included, i.e. $\gamma \in [0, 1]$. The purpose of this variable is to limit the horizon of **return**, in case $\gamma$ is equal to 0 the immediate reward is considered if it is 1 all future steps will be considered.
Usually, the literature uses $\gamma \in [0.9, 0.999]$ to gradually consider less relevant further time steps rewards thanks to the k-power of $\gamma$ inside 1.
The RL learning approach's objective is to maximize the **return** quantity, equation 1 is not useful for the agent since it considers every possible chain that can be seen using the Markov reward process. This means that it can vary widely even for the same state despite that by calculating the expectation of return from any state and averaging a large number of chains, a new

quantity can be obtained called **value of state**:

$$V(s) = \mathbb{E}[G|S_t = s] = \mathbb{E}[\sum_{t=0}^{\infty} r_t \gamma^t] \tag{2}$$

These formulas consider the reward and state, but they are not sufficient to model the environment and agent, so it is necessary to define a set of rules to control the agent behaviour considering that the objective of RL is to maximize the return.
For these reasons, a new quantity is constructed: **policy**.It is formally determined as a probability distribution over actions for every possible state, i.e.:

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{3}$$

The policy is defined as a probability to add randomness of the agent that will be useful during the training phase. If the policy is fixed, transition and reward matrixes can be reduced using the policy's probabilities and decreasing the action dimension.

### 1.2.2 Q-learning

As described, the objective of RL is to maximize return, the issue is how to approximate the best optimal policy and values state to define the correct actions for the given state. Thankfully **Bellman optimality equation** can approximate the best action that can be taken on a deterministic and statistical case. The equation is:

$$V(s) = \max_{a \in A} \mathbb{E}_{s' \sim S}[(s, a) + \gamma V(s')] = \max_{a \in A} \sum_{s' \in S} p_{a, s \to s'}(r(s, a) + \gamma V(s')) \tag{4}$$

The interpretation of this formula is that the optimal value state is equal to the action which gives the maximum possible expected immediate reward, plus the discounted long-term return of next state. This definition is recursive because the value state is determined from values of immediate reachable states. The formula not only gives the best reward that can be obtained, but it even gives the best policy to obtain that reward. Formally the policy($\pi$) can now be defined as:

$$\pi(a|s) = \max_{a \in A} Q(s, a) \tag{5}$$

In order to simplify this formula it is possible to define other quantities, such as **value of action**:

$$Q(s, a) = \mathbb{E}_{s' \sim S}[r(s, a) + \gamma V(s')] = \sum_{s' \in S} p_{a, s \to s'}(r(s, a) + \gamma V(s')) \tag{6}$$

This quantity allows to define a pair of state and action, this is particulary important because it defines a category of learning called **Q-learning** which will be the focus of this thesis. As you can see using this new definition 4 becomes:

$$V(s) = \max_{a \in A} Q(s, a)$$

Thanks to this, the 6 can be even defined recursively and will be particulary useful later for the deep learning approach:

$$Q(s, a) = r(s, a) + \gamma \max_{a \in A} Q(s', a') \tag{7}$$

The problem is that in many situations the value of actions, rewards, transition probabilities aren't knowned.Due to this it the following **Q-learning algorithm** has been created:

**Algorithm 1** Q-learning

**Require:** Discount factor($\gamma \in [0, 1]$)
**Require:** Memory table of dimension $N$ containing tuple: state($s$),action($a$), next state($s'$), reward($r(s, s')$) and action value $Q(s, a)$
  $i \leftarrow 0$
  **for** $i < N$ **do**
    Apply random action $a$
    Store $(s, a, s', r)$ on memory table
    Store $Q(s, a)$ with random value
    $i \leftarrow i + 1$
  **end for**
  **while** Until goal is reached **do**
    Observe current state of sistem $s$
    Define $c(s, s')$ as counter of how many times action $a$ was taken from state $s$ to transition to state $s'$
    Calculate $p(s, s') = c(s, s') / \sum_{s' \in S} c(s, s')$
    Calculate $Q(s, a) = \sum_{s' \in S} p(s, s') * (r(s, s') + \gamma \, max_a Q(s', a))$
    Store $Q(s, a)$ inside memory table
    Select action $a$ from policy $\pi(s|a) = max_a Q(s, a))$
    Apply action $a$
  **end while**

This Q-learning algorithm presents major drawbacks such as:

- A large memory table is required to store all the values used to approximate the $Q(s, a)$ values for the $\pi(s|a)$
- Complete iteration over all possible states is required to extract the values and store them inside the memory table

To solve these troubles a new type of Q-learning algorithm called **Tabular Q-learning** 1.2.2 was invented. The main difference is the lack of necessity to iterate over all the states to optimize because only those obtained from the environment are used for optimization. Furthermore, the table will only contain the $Q(s, a)$, but this does not resolve the drawback of having a large memory table, but only diminish it.

**Algorithm 2** Tabular Q-learning

**Require:** Discount factor $\gamma \in [0, 1]$
**Require:** Learning rate $\alpha \in [0, 1]$
**Require:** Memory table of dimension $N$ containing action value $Q(s, a)$
  **loop**
    Create a table with initial values for $Q(s, a)$
    Select random action $a$
    Observe the tuple $(s, a, r, s')$
    Calculate $V(s') = \max_{a' \in A} Q(s', a')$
    Update $Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma V(s'))$
    Store $Q(s, a)$ inside table
    Test episode using $\pi(a|s) = \max_{a \in A} Q(s, a)$ with $Q(s, a)$ of stored values

    **if** goal is reached **then**
      break loop
    **end if**
  **end loop**

As noted, a memory table is required, but only to store the $Q(s, a)$ values that will be used to update itself and define which activities need to be taken following the policy. There is still one major problem, this algorithm will struggle if there is a large count of the observables state set. This, in many real-life situations, is quite common as in the Cartpole environment of the OpenAI gym where there are only 4 states, but the interval of values that can be taken is enormous. A solution to this problem would be to use a non-linear representation of action and state into a value. This is a typical "regression problem" in the field of machine learning and thanks to the capabilities of Neural Networks (NN) is possible to approximate any kind of linear or non-linear function given enough data. Fortunately, the amount of data is almost limitless since if more is needed, the only requirement is to interact more with the given environment. So now, it is time to introduce an algorithm that uses NN the **Deep Q-learning**.

## 1.3 Deep reinforcement learning

Deep reinforcement learning is an extension of the classic one due to the use of Deep Learning techniques such as the Neural Networks (NN). Neural Networks have been introduced by [3] who were inspired by the biological brain with neurons and connections. Then many limitations were observed due to the single-layer architecture and small amount of neurons that the hardware was able to simulate. With hardware advancements and the back-propagation algorithm introduced in [4] it was later possible to train neural networks with more layers and neurons. Moreover as demonstrated by [5], neural networks with only one single hidden layer can approximate any kind of linear or non-linear function, sadly the paper doesn't tell how many neurons are required. Even if the algorithms were introduced in the last century only in the previous two decades thanks to hardware advancements such as GPUs and a large number of data from the first databases it was possible to train these neural networks. Fortunately, reinforcement learning can increase the amount of data by increasing the time of interaction with the environment, but the question of how many neurons a neural network requires to approximate doesn't have a solution yet, only indications. Now that the neural networks have been briefly proposed is time to introduce the variation of Tabular Q-learning(1.2.2) that uses neural networks to solve the environment.

### 1.3.1 Deep Q-learning

Tabular Q-learning can solve the problem of iteration over time, but it still struggles with situations when the count of observable state set is large. This is a typical situation in real life where the set of states can be infinite, solutions have been proposed as using bins to discretize, but in many cases, it did not result successfully. A better solution is to create a nonlinear representation that maps state and action to a value. This is commonly called in the machine and deep learning field a "regression problem" and does not require neural networks, but this approach is the most popular one. Generally the following point are required to train a neural network to solve an environment:

1. Initialize $Q(s, a)$ with some initial approximation.
2. Interact with the environment to obtain tuple $(s, a, r, s')$.
3. Calculate loss, if episode ended is $\mathcal{L} = (Q(s, a) - r)^2$ otherwise is $\mathcal{L} = (Q(s, a) - (r + \gamma \max_{a' \in A} Q(s', a')))^2$. The loss can be defined in other ways but it needs to have a difference between value of actions from current state and the discounted reward or only reward if episode ended.
4. Update $Q(s, a)$ using an optimizer algorithm to minimize loss.

5. Repeat from step 2 until convergence or goal is reached.

Even if it looks simple, some modifications are required to ensure convergence. First, it is necessary to find a strategy to initially explore the environment and optimize the current approximation and later exploit the model, this is often referred to as "exploration versus exploitation". To obtain a correct solution, it is good to behave randomly at the beginning because the Q-value approximation is almost certainly bad and later starts to act using the Q-value obtained to choose the action. Usually, it is used the **epsilon-greedy method** where a random probability value is sampled from a uniform distribution and confronted with a probability $\epsilon$ that tells the algorithm to behave randomly. $\epsilon$ is usually initialized to 1 so that any kind of random value sampled is smaller than $\epsilon$ so that the model behaves randomly first, later $\epsilon$ is decreased to a final small value so that the model will not always behave randomly but use the Q-values approximated.
After this problem is solved, another matter appears linked to how an optimizer algorithm work on a NN. The gradients calculated from input data require that the data are i.i.d. otherwise, there would be incorrect estimations of the correct direction in which the parameters must be updated. Furthermore, the data must not be completely random but must reflect the past actions taken to have an experience that tells which actions are worst than others. To solve this, it is necessary to create a **large memory buffer** to store the past actions are taken, both random and by the agent.
The last problem that needs to be addressed is due to the steps correlation, the loss requires $Q(s, a)$ and $Q(s', a')$ and if calculated using the same NN, this can be a problematic because $s$ and $s'$ are highly correlated and this can lead to similar approximation. This influence the convergence of learning during the training process, to deal with it a copy of the NN is created and updated by copying parameters of the original one with a fixed interval, this NN is usually called *target network*. The pseudo-code 1.3.1 is taken from the original paper that was tested on atari games [6], after this paper numerous variations have been proposed to improve convergence and reduce the time of training, such as Double Q-Net [7] and many others. Due to the limited resources and great time required the thesis will be focused on the "classical" DQN, but it would be interesting to verify if the advantage would be more pronounced or reduced in other Q-learning algorithms.
The 1.3.1 gave a breakthrough in this field due to the ability to achieve a human-like, and in some cases even better, performance on multiple games of the atari. Thanks to this Deep Reinforcement Learning have been tested and applied in other contexts such as finance, medicine, robotics and many others. Before presenting quantum computing it is necessary to explain and describe another algorithm that can handle Markov Decision Process without the value iteration method: **policy gradient methods**.

**Algorithm 3** Deep Q-Networks

**Require:** Memory buffer of dimension $N$ containing tuple: state($s$),action($a$), next state($s'$), reward($r(s, s')$)
**Require:** Discount factor $\gamma \in [0, 1]$
**Require:** Learning rate $\alpha \in [0, 1]$
**Require:** Optimizer
**Require:** A neural networks for $Q$ and target network for $\hat{Q}$
**Require:** $\epsilon$ probability of randomness
Initialize $Q(s, a)$ and $\hat{Q}(s, a)$ with random weights and empty memory buffer, $\epsilon \leftarrow 1.0$
**while** goal not reached **do**
Observe state and define with probability $\epsilon$ if $a$ is random or $a = \arg\max_{a \in A} Q(s, a)$
Apply $a$, observe $r$ and $s'$, store in memory buffer the tuple $(s, a, r, s')$
Sample a batch of tuple $(s, a, r, s')$
For every tuple calculate $y = r$ if episode ended for that state, otherwise calculate $y = r + \gamma \max_{a' \in A} \hat{Q}(s', a')$
Calculate loss $\mathcal{L} = (Q(s, a) - y)^2$
Use optimizer to update $Q(s, a)$ parameters in order to minimize $\mathcal{L}$
Every $n$ steps copy weights of $Q$ to $\hat{Q}$
**end while**

### 1.3.2 Policy gradient methods

The DQN that has been illustrated is focusing mainly on approximating the value of actions($Q$) and the value of state($V$) so that afterwards the choice of which action to take is based on a greedy approach: the best action to take is the one that maximizes the return. This is not incorrect, but in some cases, it may not be optimal due to its environmental nature. For this reason, in those cases, it is better to focus on how to define the agent behaviour to consider even the possible alternatives. Another reason why these methods are used is the possibility to introduce **stochasticity** and the ability to work on **continous environment** differently from the DQN showed.
Now that the method is focused on policy, it is necessary to give a **policy representation** to work with it, most common way is to use a probability distribution over the possible actions. This allows an additional advantage to the neural network which is to have a smooth representation. In other words, if a variation is applied to the output weights it isn't abrupt. It is now time to find a way to optimize the weights to improve the policy. Thanks to the policy gradient theorem, it is known that the formula is:

$$\nabla J \approx \mathbb{E}[Q(s, a) \nabla \log \pi(a|s)] \tag{8}$$

The entire demonstration of the formula and application on Deep Reinforcement Learning can be found on [8]. The interpretation of this expression can be as follow: the policy gradient tells us the direction of update, the formula is proportional to the value of action taken $Q(s, a)$ and the gradient of log probability action taken. The formula is trying to increase the probability of good actions and rewards, simultaneously decreasing the probability of bad actions and outcomes. Finally, the $\mathbb{E}$ is the expectation value calculated using multiple values.
There are drawbacks to this approach, such as a high gradient variance that can influence the learning and exploration at the beginning of training. To avoid falling into a local minimum it is necessary to introduce some kind of uncertainty or *entropy*. Another problem with policy gradient methods is that is less sample-efficient denoting a bigger number of iterations to solve the environment. In order to tackle this problems new algorithms have been defined, notably Actor-Critic (A2C)[9], Proximal Policy Optimization[10] (PPO) and others which tries to reap the benefits of policy and value methods. This thesis will use the Soft-Actor Critic (SAC) algorithm on a robotic environment and later will confront it with the quantum type.

### 1.3.3 Soft Actor Critic

This algorithm can be seen as a variation of Actor-Critic, the latter can be simplified as the necessity to improve the behaviour and values of states approximated to solve the environment. To achieve these, two components are used: one to estimate the policy called **policy net or actor** and one to approximate the value of state called **value net or critic**.

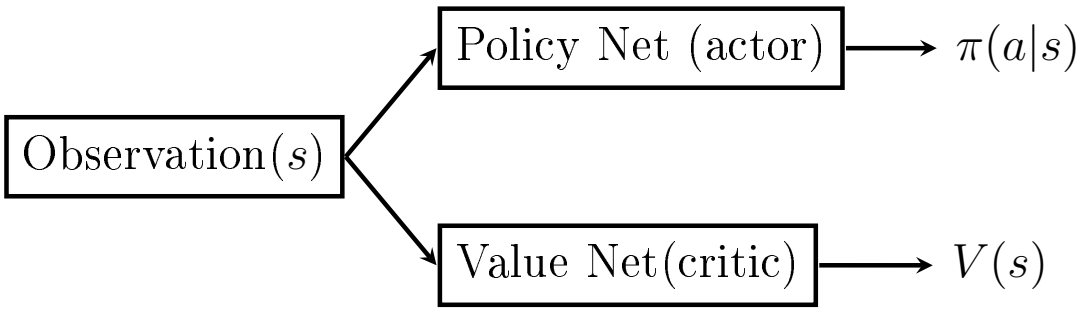



Figure 2: Schematic on actor-critic architeture

The problem with this algorithm is its on-policy nature to be sensible of hyperparameters' values. To tackle these problems and improve performance, stability and proficiency SAC was created.
SAC do it by introducing multiple components such as :

- a large buffer memory to efficiently update the weights using the previous history increasing stability and switching to an off-policy method instead of an on-policy one.

- A target network of the critic net is introduced to improve the stability and efficiency of learning to approximate the value state, differently from the DQN updating target network it will follow the procedure of **soft functions update**. Shortly, this means that the values will use a factor to combine new and old weights obtained from the update step and for this, a loss value of state ($J_V$) and loss of action ($J_Q$) function must be defined.

$$J_V(\psi) = \mathbb{E}_{s\sim S}\left[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}_{a_t\sim\pi_\phi}[Q_\theta(s_t,a_t) - \log\pi_\phi(a_t|s_t)])^2\right] \tag{9}$$

$$J_Q(\theta) = \mathbb{E}_{s\sim S,a\sim A}\left[\frac{1}{2}(Q_\theta(s_t,a_t) - r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}[V_{\bar{\psi}}(s_{t+1})])^2\right] \tag{10}$$

- the maximum entropy of reinforcement learning is introduced to increase the exploration of environment, improve learning and reduce sensibility to hyperparameters.The loss of this maximum entropy was transformed for this case from the "classical" one considering the use of a neural network and introducing an input noise vector($\epsilon_t$) sampled from the a fixed spherical gaussian distribution to calculate the action. Output action of neural network and input noise from currrent state can be expressed as follows $a_t = f_\phi(\epsilon_t; s_t)$ bringing to the loss formula:

$$J_\pi(\phi) = \mathbb{E}_{s\sim S,\epsilon_t\sim\mathcal{N}}\left[\log\pi_\phi(f_\phi(\epsilon_t;s_t)) - Q_\theta(s_t, f_\phi(\epsilon_t;s_t))\right] \tag{11}$$

For more details on how these formulas have been obtained and experimental results refer to the original paper [11], to avoid confusion with the paper pseudocode will be showed later considering that there are variations from the original implementation to create a quantum version, so that later it would be possible to confront classical and quantum algorithm.

## 1.4 Quantum computing

Quantum computing is a field that originated around the 1980s by numerous physicists, Paul Benioff was the first to introduce the idea of a Turing machine that used quantum mechanics to make the calculations. Richard Feynman and Yuri Manin suggested using it for efficient simulation of quantum mechanical systems. Furthermore, David Deutsch asked if using physical laws

it was possible to derive a stronger version of the Church-Turing thesis introducing the concept of Universal Quantum Computing, this conjecture isn't still demonstrated but has paved the way for the current concept. To introduce quantum computing it is necessary to understand what is a: quantum bit, circuit and logic gate. Always keep in mind that even if the following representation is an abstraction there is a physical implementation for all these components.

### 1.4.1 Quantum bits

Quantum bits or qubits are analogous to classical bits which are used as a fundamental concept for computation and information forming computational base states. As the bits can have two possible states 0 and 1, the qubits, thanks to the discretized energy on a microscopic level, can have a quantum state $|0\rangle$ and $|1\rangle$. Differently from their classical counterparts and thanks to quantum mechanics, the qubits can be in a possible *linear combination* even known as *superposition* of states:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{12}$$

The numbers $\alpha$ and $\beta$ are complex value also known as *amplitudes*, so a qubit can be represented as a two-dimensional complex vector space. A fundamental condition is applied to these numbers to respect the statistical view of quantum mechanics, when a qubit has been measured a probability of being in the quantum state is associated. These probabilities are $|\alpha|^2$ to be in-state $|0\rangle$ and $|\beta|^2$ in-state $|1\rangle$, for natural statistic normalization these number must have the condition $|\alpha|^2+|\beta|^2 = 1$.Geometrically this can be interpreted as the fact that qubit states must be normalized to have length 1 and be rewritten using real numbers as:

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\right) \tag{13}$$

with $\gamma, \theta, \varphi \in R$. Due to the fact that there are no observables effects with a global phase, it is possible to ignore $e^{i\gamma}$ giving the reduced formula:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle \tag{14}$$

This allows to introduce a three-dimensional sphere representation called **Bloch sphere** showed in figure 3, which allows to represented any possible superposition of states in a qubit.

This formalism can be extended for the case of multiple qubits, if for example a system is composed of two separable qubits $|\psi\rangle$ and $|\phi\rangle$ then it is possible

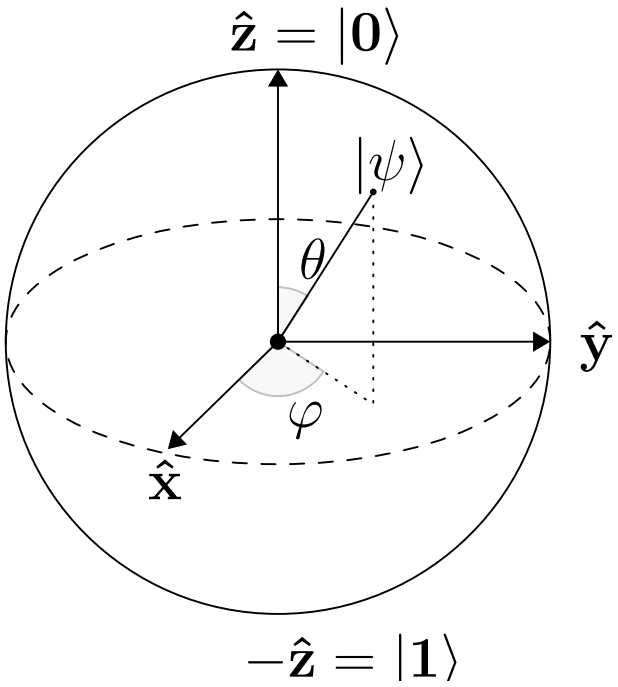



Figure 3: Representation of a qubit the Bloch sphere

to construct their representation $|\nu\rangle$ using the dot product as follows:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad |\phi\rangle = \gamma\,|0\rangle + \delta\,|1\rangle$$
$$|\nu\rangle = |\psi\rangle \otimes |\phi\rangle = \alpha\gamma\,|00\rangle + \alpha\delta\,|01\rangle + \beta\gamma\,|10\rangle + \beta\delta\,|11\rangle$$

The previous form of $|00\rangle$ is only an abbreviation for the form $|0\rangle \otimes |0\rangle$, normalization must be still valid so the sum of all amplitudes squared must give 1, in other words $|\alpha\gamma|^2 + |\alpha\delta|^2 + |\beta\gamma|^2 + |\beta\delta|^2 = 1$ if that is not the case it is possible to normalize it dividing by the root square of the squared sum. As it can be seen with only two qubits there are four quantum states, if for example there are $N$ qubits then there would be $2^N$ possible states giving a tremendous advantage on the information that can be stored instead of the classical approach. Furthermore thanks to the fact that this is a multi dimensional space a matrix representation to give the following formalism:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix} \qquad |1\rangle = \begin{bmatrix}0\\1\end{bmatrix} \qquad |\psi\rangle = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \qquad |\phi\rangle = \begin{bmatrix}\gamma\\\delta\end{bmatrix}$$
$$|\nu\rangle = |\psi\rangle \otimes |\phi\rangle = \begin{bmatrix}\alpha\gamma & \alpha\delta\\\beta\gamma & \beta\delta\end{bmatrix}$$

In case a measurement is applied on subsets of qubits, for example on the first qubit measuring if the state is $|0\rangle$ then the post-measurement state will

be:

$$|\nu'\rangle = \frac{\alpha\gamma\,|00\rangle + \alpha\delta\,|01\rangle}{\sqrt{|\alpha\gamma|^2 + |\alpha\delta|^2}} \tag{15}$$

the denominator is due to normalization after measurement in order to respect the condition that squared amplitudes summed is 1.
Quantum mechanics allows even some particular states called *Bell states or EPR pair* which are not separable such as this one:

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{16}$$

even if it seems to be completely normal, notice that in case a measurement is applied on any of the two qubits it is possible to know immediately with certainty what state is in the other one. So there is a *correlation* between the qubits, point is that this type of correlation is **stronger than any kind that could exist classicaly**. This correlation is known as **entanglement**, is the basis of many quantum algorithms and has been proven to be valid even on large distances. This doesn't violate Einstein's general relativity because the information, even if using this quantum property, in any case can't exceed the speed of light. Now that the separated and entangled states have been introduced it is time to see how the qubits can be manipulated using the **gates and circuits**.

### 1.4.2 Quantum gates and circuits

As classical computation uses *wires* and *logic gates*, quantum computing does the same. The main difference is that non-linearity isn't allowed, an example is a quantum NOT which acts linearly instead of non-linearly.
The motive is that if non-linearity was allowed in these operations inconsistency would be present, such as violations of the second law of thermodynamics, faster than light communications and time travel.
Taking advantage of the linearity it is possible to give a matrix form to the NOT gate which classically inverts 0 to 1, in quantum computing is:

$$\alpha\,|0\rangle + \beta\,|1\rangle \rightarrow \beta\,|0\rangle + \alpha\,|1\rangle$$

in matrix form it can be translated as:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \rightarrow X \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \beta \\ \alpha \end{bmatrix}$$

In addition to the linearity of quantum gates, it is necessary to remember that these operations must respect the condition of normalization, in other words their sum must be equal to 1. Translating the condition on linear algebra

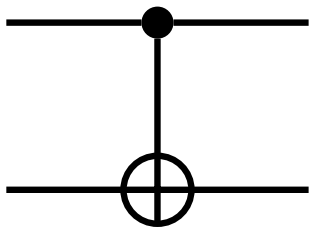
means that any matrix $U$ representing a quantum gate, must be unitary.
This can be formalized using the *adjoint* represented as $U^\dagger$, which means complex conjugating and transposing the elements of U, transforming the condition of normalization to $U^\dagger U = 1$. This can be even interpreted as the fact that any kind of quantum operation applied is reversible, so by applyingtwo times the same operation is possible to return at the initial state, which quantistically is the adjoint of operation applied. The most used single-qubit gates are the following:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

there are other gates such as S, T and I, but for the thesis they will not be necessary. If interested feel free to check the book [2].
The interesting part is that any kind of single qubit gate can be defined in this form:

$$U(\theta, \phi, \lambda) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda} \sin(\frac{\theta}{2}) \\ e^{i\phi} \sin(\frac{\theta}{2}) & e^{i(\phi+\lambda)} \cos(\frac{\theta}{2}) \end{bmatrix} \tag{17}$$

for confirmation notice that $H = U(\frac{\pi}{2}, 0, \pi)$.
Now as there a single qubit gates, it is possible to extend the concept introducing **multiple qubit gates** or **multi-qubit gates**. These differently from the single ones use multiple of them, as an example a multi-qubits gate that will be used extensively is the *controlled-NOT* or CNOT gate which has the following form:



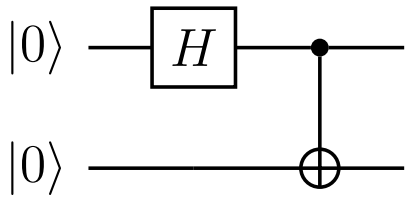
$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This gate function by using a control qubit (black dot) and activates only if the control qubit is $|1\rangle$ by applying to the target qubit ($\oplus$ )addition modulo

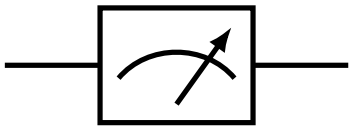
two operation that on a two basis computational state means to flip it. The horizontal line represent the wire used to transfer the qubit. Formally CNOT transforms the basis states in this way:

$$|00\rangle \to |00\rangle \qquad |01\rangle \to |01\rangle \qquad |10\rangle \to |11\rangle \qquad |11\rangle \to |10\rangle$$

The main point is that this circuit is like applying an $X$ gate on the target qubit, but it is even possible to substitute it with other single-qubit gates allowing more kinds of operations. There are other multi-qubit gates such as *toffoli*, *swap* and many others, but they will not be used in this thesis. The reason for which **CNOT is largely used refers to his addiction of the entanglement effect on qubits**, this is particularly important to efficiently explot the quantum advantage. For example the 16 can be obtained with the following circuit:




Now that gates and wires have been introduced, is necessary to introduce one last component the measurement gate, in fact after the qubits are manipulated information must be extracted. To do that, a measurement basis can be decided and quantum computing allows more than the classical basis of $|0\rangle$ and $|1\rangle$, but only this one will be used. As said the measurement consists of extracting information, but for the quantistic case this means to distinguish the possible states, this is represented with the following symbol on circuits:



This measurement on the computational basis consist on using projectors of the possible eigenspaces, for the case of a generic qubit state projectors are $P_0 = |0\rangle\langle 0|$ and $P_1 = |1\rangle\langle 1|$. The probability associated to be in one state, for example $|0\rangle$ can be defined as:

$$p(0) = tr(P_0 |\psi\rangle\langle\psi|) = \langle\psi| P_0 |\psi\rangle = |\alpha_0|^2$$

The fully observable corresponding to a computational basis measurement is the Pauli-Z operator or the Z gate:

$$\sigma_z = |0\rangle \langle 0| - |1\rangle \langle 1| = Z$$

To work with quantum computing it is necessary to use statistics and to have the correct expectation value of $\langle \sigma_z \rangle$ it is necessary to repeat multiple times the measurement to sample the possible values, which is usually represented as $S$ which is an abbreviation for *shots*. To have the correct expectation with an error of at most $\epsilon$ conventional statistics can be used for example using the Bernoulli distribution. The error gives a confidence interval $[p-\epsilon, p+\epsilon]$ which tells the proportion of samples that are in the interval. Statistically, this can be associated with a $z$-value. Estimating the error of a Bernoulli trial can be defined in different ways, but the most preferred way is by using the Wilson score interval [12]. Following that formula the overall error of estimation is bounded by the following equation:

$$\epsilon \leq \sqrt{z^2 \frac{S + z^2}{4S^2}}$$

which can be inverted to calculate the S required:

$$S \leq \frac{\epsilon^2 \sqrt{\frac{z^4(16\epsilon^2+1)}{\epsilon^4}} + z^2}{8\epsilon^2}$$

### 1.4.3 Quantum decoherence

There is a phenomenon that needs to be considered when a quantum device is effectively used and that is **quantum decoherence**. The phenomenon is referred to as the effect of losing quantum coherence, which presents when the phase relation of the wave functions associated with different states is not valid anymore. This is caused by the not complete isolation and interactions with the environment introducing entanglement between them.
The effect of quantum decoherence is problematic because implies the loss of information stored. This effect for quantum computing introduces non-linearity and is usually considered through quantities called *decoherence times*. These are defined as $T_1$ and $T_2$, $T_1$ refers to *relaxation time* which is the time taken to transition from $|1\rangle \rightarrow |0\rangle$. $T_2$ refers instead to *dephasing time* that is the time for which qubits phase remains intact. These times are usually combined and used as constant decays for a statistical description of when quantum decoherence affects the qubits.
The application of quantum gates influences the system by increasing the possibility of decoherence, for this reason, **quantum error correction** has been developed. The problem is that these algorithms for error correction

require a lot of qubits and actually there are no physical devices with this capacity.
To deal with the quantum decoherence from environment and gates, new types of algorithms have been created with the requirement to use the least number of gates and operations required to successfully extract information.

### 1.4.4 Encoding algorithms

Working with quantum computers requires that information which needs to be analyzed and transformed needs to be stored inside qubits, there are different ways to encode information: *basis encoding, amplitude encoding, angle encoding.*
Basis encoding is based on the fact that any bit is substituted with the qubit having a corresponding state value, for example, 01001 becomes $|01001\rangle$ using this strategy. As it is possible to notice this representation is not qubit efficient because it requires the same number of qubits for the associated bit of value, but the routine doesn't require many operations. Another strategy of encoding is called *amplitude encoding*, which consists of taking the value of a classical vector $x$, normalising the values inside by normalizing the elements and associating to them an amplitude, like in this example:

$$x \to \sum_{k=1}^{2^n} |x_k|^2 = 1$$

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{2^n} \end{bmatrix} \leftrightarrow |x\rangle = \sum_j x_j |j\rangle$$

this kind of representation requires very few qubits, but the routine required to encoded it is not applicable on the actual physical devices for their limited capabilities.
The last strategy is the *angle encoding* which consist on applying a rotation of the classical value along one axis, this is used because it can exhibit some similarities and proprierties to the amplitude encoding. It is not qubit efficient, but the routine can be applied using actual devices, formally it can be defines as follows:

$$x \to |x\rangle = \otimes_i^n R(x_i) |0^i\rangle$$

where $R$ can be $R_x, R_y, R_x$ so it is only necessary to apply one single rotation gate over every qubit using as the angle the classical value.

# 2 Variational circuits applied to deep reinforcement learning

Variational circuits or *parametrised circuit*, are a type of quantum computing model that can be used in the field of machine learning. The problems that this model tries to tackle are: finding the best parameters to solve the problem required and applicability to actual physical quantum devices with their limitations. This kind of circuit is *hybrid* because it uses concepts that come from quantum and classical algorithms, furthermore, it can be applied to actual quantum devices with a classical computer. The usual approach for this kind of circuit is to use quantum algorithms as the machine learning model and apply the training on data using classical algorithms. In this thesis, it will be used as a layer for Neural Networks (NN), since these kinds of circuits can be even viewed as *quantum neural networks* thanks to their approximation ability to functions and because deep learning models are particularly successful in Reinforcement Learning (RL). A consideration that needs to be done is the fact that an ideal variational circuit must be small as possible to make it work on a quantum device and reduce the possibility of information loss due to decoherence, this type of circuit is referred to as **shallow circuit**. An extensive paper that describes these circuits is [13]. Many consideration and concepts are taken from: [14].

## 2.1 Variational Circuit

Variational circuits have been formulated by [15]-[16], but the idea was introduced years before. This circuit is based on the fact that a gate can have an associated parameter, the parametrized and not parametrized gates that will be used in the circuit will form an **ansatz**.To optimize the parametrized circuit it is necessary to define a cost function $C(\theta)$ and find an optimizer algorithm able to find the set parameters for which this cost function is minimized. This optimizer usually is based on gradient methods to reach the set of parameters that minimize the cost function, fortunately, the gradient of a variational circuit can be calculated quite easily on a quantum computer using a method called **parameter shift** as it will be seen later.
Thanks to the properties of quantum mechanics the entire circuit can be considered as a single unitary gate of the form $U(x, \theta)$, where the $x$ is due to their dependencies on the input data. Usually the internal structure of $U(x, \theta)$ consist on of an encoding block $S(x)$ and a parametrized one $W(\theta)$, resulting in $U(x, \theta) = S(x)W(\theta)$, these blocks can contains quantum fixed gates. Graphycally this can be seen as:

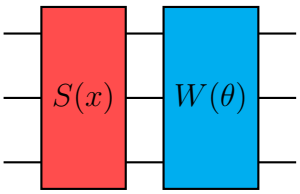



Figure 4: ususal decomposition of quantum circuit

So in the end a variational circuit can be graphically summarized as follows:

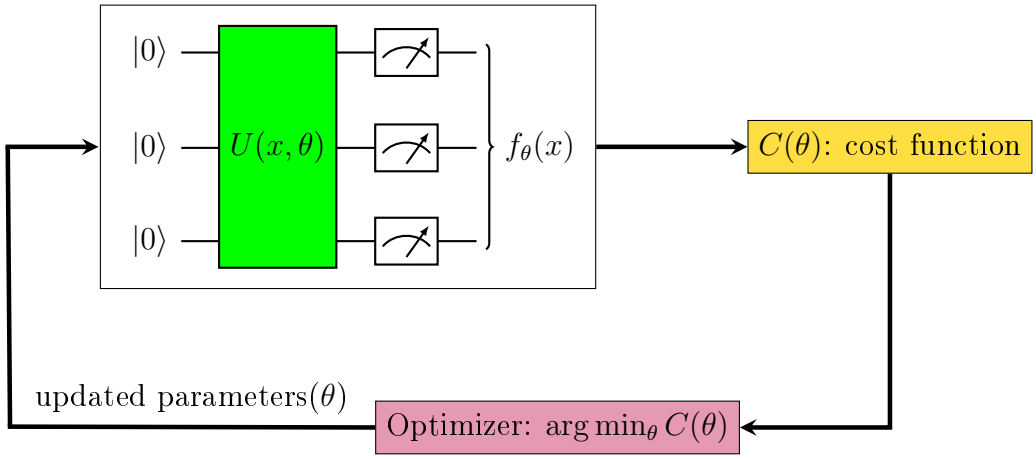



Figure 5: Schematic of a variational quantum algorithm

Furthermore these variational circuits can be interpreted, with minor conceptual changes, as deterministic or probabilistics machine learning models and can be ingtegrated, as it will be seen in this thesis, as a component of neural networks.

### 2.1.1 Deterministic quantum model

**Definition 1** *Considering a data domain $X$, the quantum circuit $U(x,\theta)$ with $x \in X, \theta \in \mathbb{R}^n$ depends on input data and indicates $\mathcal{M}$ as a hermitian operator associated to a quantum observable. It is possible to denote $|\psi(x,\theta)\rangle$ as $U(x,\theta)|0\rangle$ then it is possible to define the output of this variational circuit as:*

$$f_\theta(x) = \langle\psi(x,\theta)|\,\mathcal{M}\,|\psi(x,\theta)\rangle = \langle\mathcal{M}\rangle_{x,\theta} \tag{18}$$

This definition tells that even if quantum computing output is statistical, an average value based on the measurement applied can be extracted. For example if measurement is written in diagonal basis such as : $\mathcal{M} = \sum_i \mu_i |\mu_i\rangle \langle\mu_i|$, then the output function will be of the form of:

$$f_\theta(x) = \sum_i \mu_i | \langle\mu_i|\psi(x,\theta)\rangle |^2 = \sum_i \mu_i p(\mu_i)$$

If the measurement applied is based on the Z-gate($\mathcal{M} = Z$), which will be the one used in the following variational quantum algorithm, the result can be rewritten by considering the eigenvalues and eigenstates obtaining the following result:

$$f_\theta(x) = | \langle 0|\psi(x,\theta)\rangle |^2 - | \langle 1|\psi(x,\theta)\rangle |^2 = p(0) - p(1)$$

This quantity can be even calculated by performing $S$ shots sampling the eigenvalues $\mu(s) \in \mu_i$ and averaging over the results, obtaining the following form:

$$\hat{f}(x) = \frac{\sum_{i=1}^{S} \mu_i}{S}$$

This value can be estimated with an error $\epsilon$ applying $O(\epsilon^{-2})$ measurement, this means that if for example, the error required is 0.01, then the measurement needed to be applied is of the order in 1000s. This deterministic function will be used mainly for approximating the q-value and other quantities necessary for the reinforcement learning algorithm.

### 2.1.2 Probabilistic quantum model

The inherit theory of quantum mechanics allows for these models to be expressed as probabilistic, in fact as it will be seen later Variational Quantum Algorithm can be interpreted as supervised and unsupervised models applying minimal modifications on the quantum circuit.

**Definition 2 (supervised probabilistic quantum model)** *Let $X$ be an input and $Y$ an output domain, and $U(x,\theta)$ be an input and parameter-dependent unitary so that $\psi(x,\theta) = U(x,\theta)|0\rangle$. It is possible associate each eigenvalue or outcome of a measurement observable with a possible output $y$, so that $\mathcal{M} = \sum_{y\in Y} y |y\rangle \langle y|$. A supervised probabilistic quantum model for a conditional distribution is then defined:*

$$p_\theta(y|x) = | \langle y|\psi(x,\theta)\rangle |^2 \tag{19}$$

Due to $|y\rangle$ being a basis, the normalization is required so $\sum_{y\in Y} |y\rangle \langle y| = \mathbb{1}$ and probability sum to 1.Representing this kind of variational quantum circuit using the previous ansatz, it can be represented as:

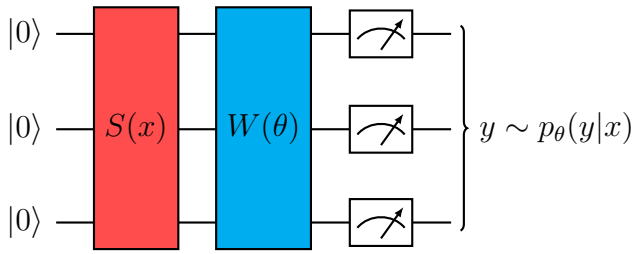



Figure 6: Variational quantum algorithm for conditional probability

This kind of circuit has been used for classification and regression tasks mainly, but as it will be seen that can be used in reinforcement learning to decide which action to take in a discrete environment such as the Cartpole.

**Definition 3 (unsupervised probabilistic quantum model)** *Let $X$ be an input domain, $W(\theta)$ a unitary that depends on some parameters with $|\psi(\theta)\rangle = W(\theta)|0\rangle$, and $\mathcal{M} = \sum_{x \in X} x |x\rangle \langle x|$ a measurement in diagonal basis with outcomes that correspond to the inputs $x$. An unsupervised probabilistic quantum model is defined by the distribution:*

$$p_\theta(x) = |\langle x|\psi(\theta)\rangle|^2 \tag{20}$$

The circuit can be graphically defined as:

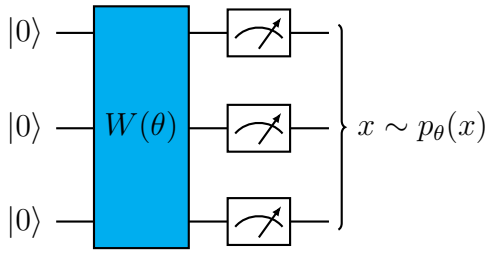



Figure 7: Variational quantum algorithm for unsupervised approach

So it is possible to notice that the only differences between the unsupervised and supervised models are two: the absence of an encoding layer and the data basis for measurement instead of the output basis.Probabilistic quantum model is strictly correlated to deterministic one due to the fact that $p_\theta(x) = |\langle x|\psi(\theta)\rangle|^2$ can be even seen as $\langle x| (|\psi(\theta)\rangle \langle\psi(\theta)|) |x\rangle$ with basis measurement of $\mathcal{M} = |\psi(\theta)\rangle \langle\psi(\theta)|$. This is important because it define a clear connection between the insights on designs applied in both applications. Furthermore, it is possible to notice that VQA are generative models, but it is

difficult to extract this distribution due to the measurements required. Lastly, unsupervised quantum models are known as *Born machines* from the Born rule that links quantum states and probability.

### 2.1.3 Quantum models as linear combinations of periodic functions

Generally, it is possible to express the quantum circuit as an alternation between encoding and parametrized gates repeated for multiple layers:

$$U(x,\theta) = W_{N+1}(\theta) = \prod_{i=1}^{N} S_i(x_i)W_i(\theta) \tag{21}$$

The encoding gate can be defined as having the form $S_i(x_i) = e^{-ix_iG_i}$ where $G_i$ is called generator and, without losing generality, can be assumed a diagonal operator $diag(\lambda_i^1, \ldots, \lambda_i^1)$ with $d$ dimension of Hilbert space. Then a theorem proven by [17] demonstrate that such models are linear combination of functions.

**Theorem 1** *Let $X \in \mathbb{R}^n$, $Y = \mathbb{R}$ and $f_\theta : X \to Y$ be a deterministic quantum model with a circuit $U(x,\theta)$ as defined in 21. Accordingly the i-th feature is encoded by gate $e^{-ix_iG_i}$, then $f_\theta$ can be written as*

$$f_\theta(x) = \sum_{w_1 \in \Gamma_1} \cdots \sum_{w_N \in \Gamma_N} c_{w_1 \ldots w_n}(\theta) e^{iw_1x_1} \ldots e^{iw_Nx_N}$$

*with the frequency spectrum of the i-th feature, $i = 1, \ldots, N$ is given by*

$$\Gamma_i = \{\lambda_s^i - \lambda_t^i | s,t \in 1, \ldots, d\}$$

*This frequency spectrum is the set of all values produced by differences between any two eigenvalues of $G_i$. We are guaranteed that $0 \in \Gamma$, and for each $w \in \Gamma$ there is $-w \in \Gamma$ too with $c_w(\theta) = c_{-w}^*(\theta)$. This symmetry guarantees that $f_\theta$ is real-valued and that the sum can be rewritten with cosine and sine functions.*

This theorem is important because it tells that there is no non-linearity if only the variational circuit is applied, frequencies are defined by the encoding layer and weights are dependent on parameters and frequencies. This means that for an expressive Variational Quantum Algorithm, the encoding and parameters layer must be repeated multiple times to have enough expressivity. So non-linearity is not present on these circuits unless measurement or specific gates are added as will be seen later.

### 2.1.4 Variational algorithm training

As previously stated this type of circuit is parametrized and the parameters are optimized by calculating a cost function and the gradient. Since cost function is dependent in first place on output function and afterwards to parameters, the chain rule needs to be applied to obtain the following gradient:

$$\frac{\partial C(\theta)}{\partial \mu} = \frac{\partial C(\theta)}{\partial f(\theta)} \frac{\partial f(\theta)}{\partial \mu} \tag{22}$$

This form has two components required $\frac{\partial C(\theta)}{\partial f(\theta)}$ and $\frac{\partial f(\theta)}{\partial \mu}$, to exactly calculate the differentiation. The first can be obtained using classical computation, while the second cannot be evaluated using classical quantities because it depends directly on the quantum circuit. In fact using the parameter-shift rules it possible to calculate the gradient of a quantum circuit compared to the parameters $\frac{\partial f(\theta)}{\partial \mu}$ , while for the other quantity libraries such as *Tensorflow* and *Pytorch* can apply **automatic differentiation** to approxiamte the result.

**Definition 4 (parameter-shift rules)** *Let $f_\mu = \langle \mathcal{M} \rangle_\mu$ be a quantum expectation value that depends on a classical parameter $\mu$. A parameter-shift rule is an identity of the form:*

$$\partial_\mu f_\mu = \sum_i a_i f_{\mu + s_i} \tag{23}$$

*where $a_i$ and $s_i$ are real scalar value.*

As it can be noticed this formula is similar to the finite difference method which is:

$$\frac{\partial f_\theta}{\partial \mu} \approx \frac{f_\theta - f_{\theta + \Delta\theta}}{||\Delta\theta||}$$

The main difference is that **the parameter-shift rule method can estimate the analytic gradient, while the finite difference method focuses on approximating it**. Furthermore, the parameter-shift rule is not dependent on how big or small the variation is to correctly calculate the gradient and requires very few evaluations. There is a problem with the gradients of variational circuits, the significant presence of **barren plateaus**. This term refers to the fact that sometimes the cost function can present a zone where the gradient is highly probable to be zero leading to little variations in gradient. Mathematically this means that:

$$Var[\partial_\mu f_\mu] = 0 \tag{24}$$

This is a problem for the correct optimization of parameters since the gradient is small, leading to a non-global minimal solution or requiring a lot of

training before actually leaving this zone. The **barren plateaus** are present in Neural Networks and is something studied and discussed, but no real solution has been actually found to deal with it. The difference between the classical and quantum model is the fact that these regions are ***exponentially larger*** on the quantum case as demonstrated by [18]. The paper furthermore explains that increasing the number of gates, referred to even as layers due to repetition of specific gates, and qubits leads to exponential decay in the variance. An ulterior motive for which the variational circuit must be shallow is to avoid problems with trainability. There have been some strategies addressed to partially solve this problem such as initialization using correlated parametrized circuit, as suggested by [19], but this represents one of the major challenges to solve to use deepest circuits. A deeper and more detailed explanation can be found in [14].

### 2.1.5 Variational algorithm as neural networks

Variational circuits are sometimes called "quantum neural networks". This name is partially reasonable due to some resemblance to classical neural networks, such as the optimization of parameters and structure linearity for deep learning. The problem is that these circuits do not express non-linearity unless a particularly encoding strategy or measurement is applied. Interestingly, a non-linear activation function can be obtained by applying different types of encoding with slight modifications as suggested by [14], but the papers used for reference to create the ansatz for reinforcement learning do not apply them. It would be interesting to see if this encoding may result in a better performance on future tests.
A possible schematic on how Variational Quantum Algorithm can be interpreted as Neural Networks is given by figure 8.From this schematic, it can be observed an important fact the only non-linearity present is due to the measurement applied at the end.
To introduce more non-linearity gates such as *depolarizing gates* and even *noise* can be added, but this is not a complete solution. Furthermore, it can be noticed how every gate applied can be associated with a layer of the neural network. A possible representation that can be used is to define a formalism linked to connectivity by generalizing a single qubit gate as:

$$W = \begin{bmatrix} z & u \\ -u* & z* \end{bmatrix}$$

This matrix representation must respect the condition of normalization. In case the single qubit gate is applied on only one qubit, for example $i$, a multi-

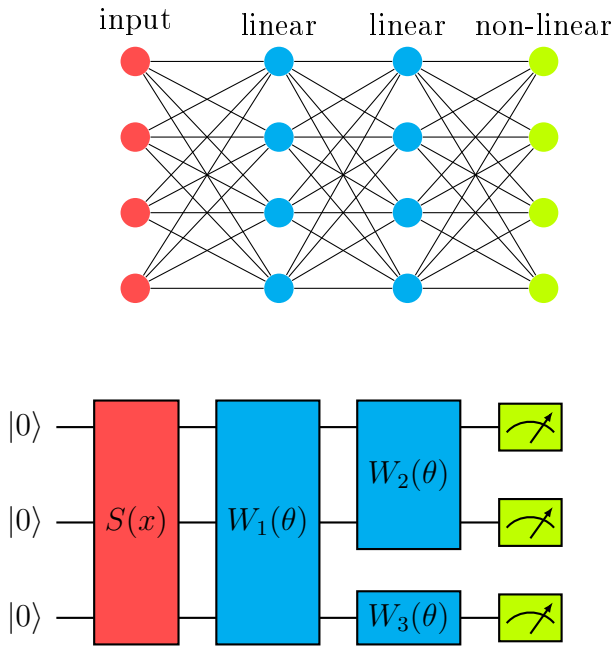



Figure 8: Variational quantum algorithm as a neural network

qubit representation can be defined by extending the previous definition:

$$W_i = \mathbb{1} \otimes \cdots \otimes \underbrace{W}_{i \text{ position}} \otimes \cdots \otimes \mathbb{1}$$

The symbol $\mathbb{1}$ refers to an identity matrix $2 \times 2$, this means that a single qubit gate applied leads to a sparse representation and the matrix able to represent any kind of operation applied on multiple qubits has shape $2^n \times 2^n$. This is interesting and gives a connection between the layer of neural network with linear activation and VQA because both of them use matrix operations.

### 2.1.6 Data reuploading

As previously explained, the Variational Quantum Algorithm are linear except for the final points where measurement is applied. This can be problematic for the field of deep reinforcement due to the significant presence of non-linear functions in the q-values and actor components. To introduce this non-linearity and deal with the **non-cloning** property of quantum computing, new types of ansatz has been defined: ***data reuploading***. This concept has been introduced in the paper [20] and it wants to introduce non-linearity by reapplying the encoding layer multiple times to have a more composite function expressivity. So the strategy is to not use more qubits but to apply multiple layers increasing the depth of the final circuit. This seems to work

particularly well for reinforcement learning using simulators.
The question is if this kind of depth would be able to run on a quantum computer and will be able to achieve the same performance. This is something that may be required further work.

## 2.2 Quantum Deep Q-learning applied to Cartpole

Now that Variational Quantum Algorithm has been explained it is now time to see the results obtained by applying quantum models on the environment called Cartpole-v0. This is part of the library Open-AI gym ([21], link), which contains multiple environments used to benchmark reinforcement learning algorithms.
The Cartpole environment is composed by a pole attached to a cart:

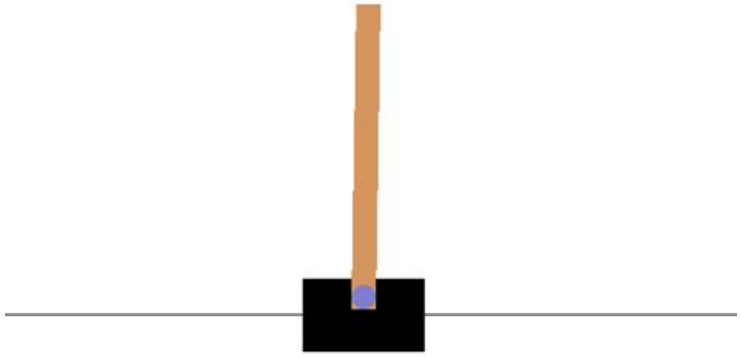


Figure 9: Cartpole environment of Open-AI gym

The environment state is an array of 4 values representing: (position, velocity, pole angle, and pole angular velocity). Position and angle are limited in value, while the others are not. The possible actions that can be taken are linked to a single value that can be 0 or 1, which respectively means push left or right. The reward is given by the environment for any step taken increasing the total return by 1. The conditions for which an episode is stopped are: the cartpole reaches the environment extremes, the angling pole is equal to

or greater than 12° or the number of steps taken is equal to 200.
The goal is to reach a reward greater or equal to 175 for 100 episodes. Technically when this condition is reached the training should be stopped, but it will be extended for 1000 episodes to check the stability and convergence of the algorithm. To confront the advantage between Neural Networks and Variational Quantum Algorithm the trend of reward, the number of episodes taken to reach the goal, the time taken and the number of parameters will be counted. This has been decided from the reference of other papers that used this way to benchmark classical and quantum models.

### 2.2.1 Ansatz for VQA

Different ansatz has been used for this work, the starting point was from the paper [22] which has a corresponding Github repository that can be used for the code. Variational circuit proposed for the paper is based from the following layer:

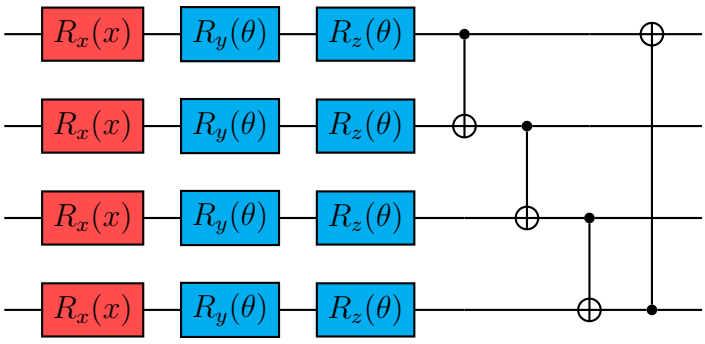



Figure 10: Variational quantum algorithm for cartpole

As it can be seen, this kind of layer applies the data-reuploading approach already mentioned. This seems vital to achieve good performance concerning the classical one, for more information please consult the paper. The layer is repeated multiple times to achieve enough depth for the expressivity required to approximate the best q-value function. The measurement applied at the end uses a probabilistic approach, in fact, due to the nature of the environment a single action is required, go left or right, so to reduce the dimension output a measurement composed of two qubits corresponding to $Z \otimes Z$ is applied. After the measurement value with maximum probability is chosen and executed. To increase the performance, parameters on input and output are added, leading to the following final structure 11.

The original implementation used libraries Pennylane and PyTorch, this made the algorithm take around 11 hours for 5000 episodes. Due to the requirement of benchmarking the code has been rewritten using TensorFlow

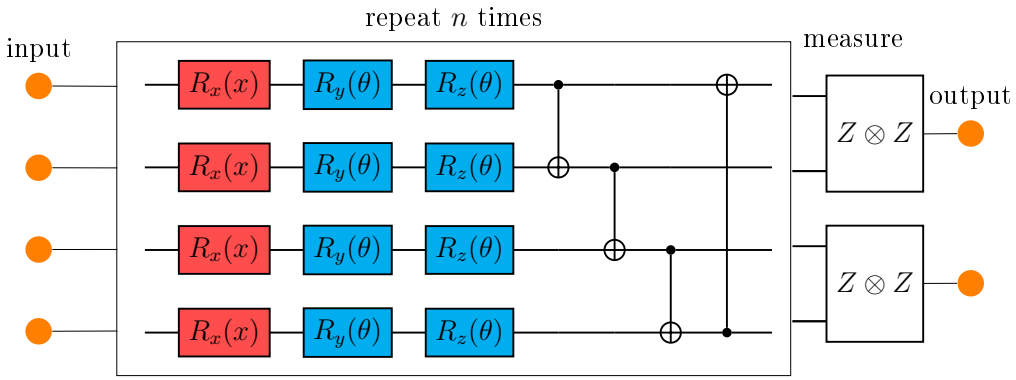



Figure 11: Final structure for cartpole

quantum, this allowed a speedup by reducing the time required by at least 11-12 times. By applying multiple runs and confronting the runs made by the quantum algorithm and different kinds of neural networks this result is obtained:

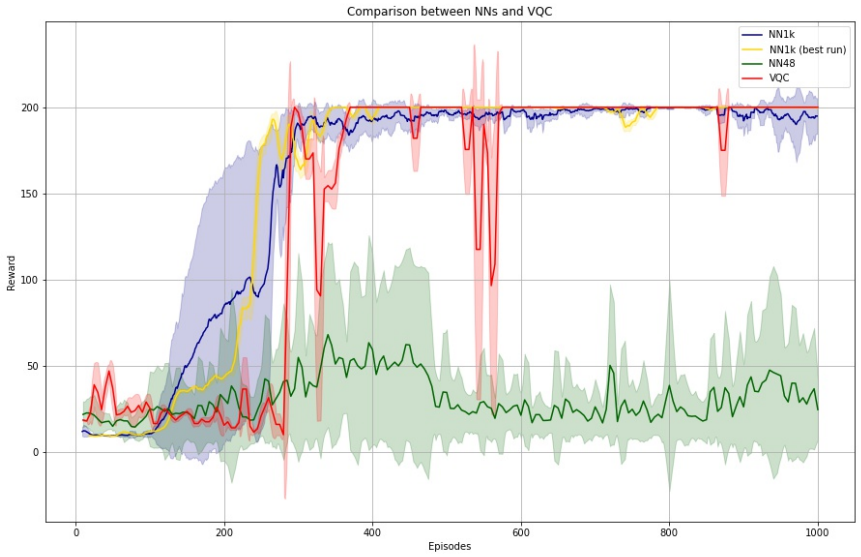



Figure 12: In this image it has been used the mean and the variance of 10 runs of neural networks and vqa

As it can be seen from the plot a neural network with the same number of vqa parameter is unable to reach the goal of cartpole and in order to have the same trend a neural network with 1256 parameters.
This means a quantum advantage is present in the number of parameters and

expressivity of the modes with few layers. The paper referenced is more expressive and more exhaustive on all hyperparameters that have been tested, the configuration used for this benchmark is:

| hyperparameter | NN(1256 params) | NN(48 params) | VQA |
|---|---|---|---|
| $\gamma$ | 0.99 | 0.99 | 0.99 |
| optimizer | Adam | Adam | Adam |
| batch size | 64 | 64 | 16 |
| learning rate | 0.001 | 0.001 | 0.01 |
| buffer memory | 100000 | 100000 | 10000 |
| $\epsilon$ start | 0.1 | 0.1 | 1 |
| $\epsilon$ decay | 0.99 | 0.99 | 0.99 |
| $\epsilon$ final | 0.001 | 0.001 | 0.01 |
| Loss | Smooth-L1 | Smooth-L1 | Smooth-L1 |
| neuron layers | (4,32)(32,32)(32,2) | (4,8)(8,2) | (4,)(2,) |
| vqa layers | None | None | 5 |

As it can be seen from the hyperparameters $\epsilon$ used for the $\epsilon$-greedy algorithm used is decreased from starting to ending value applying a formula called linear decay.
A better and more efficient ansatz was proposed by TensorFlow quantum exactly for this kind of problem giving to the following structure of the Variational Quantum Algorithm:

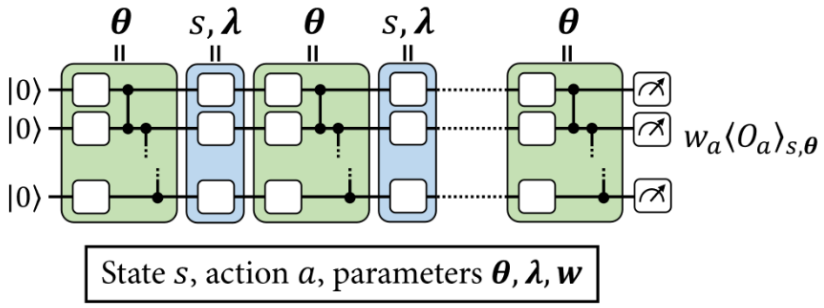



Figure 13: Ansatz of variational circuit with data reuploading of the tensorflow quantum library

The differences between this kind of ansatz and 11 are the lack of classical weights input-output and above all the encoding layer is parametrized. This means that the expressivity of this model can be tuned by finding the parameters that can give the best encoding to find afterwards the best approximation.
Now that the results of Cartpole have been shown proving that there can

be a quantum advantage using fewer parameters, it is now changed with another environment which is based on the robotic arm and can demonstrate the quantum capabilities applied to possible future industrial applications.

## 2.3 Variational quantum algorithm on robotic arm

### 2.3.1 Environment

The previous environment is used for benchmarking and has a partial view of the capability of this algorithm. It is now time to test it on something more complex and with application in the industry: a robotic arm.
This technology is applied in many fields of industry such as manufacturing, cars and many others. The environment that will be used to simulate a robotic arm has been created and offered by prof. Noah Klarmann from the university of Rosenheim I would like to thank. This environment is formed of an arm composed of different links whose number can be defined, that influence both the space of state and action, with a random point which needs to be reached by the arm using its extreme.
The links are independent and can move in any direction with a fixed maximum velocity, which means that for every possible point that can be reached the number of steps required is not always the same. As it can be seen from this rendering:

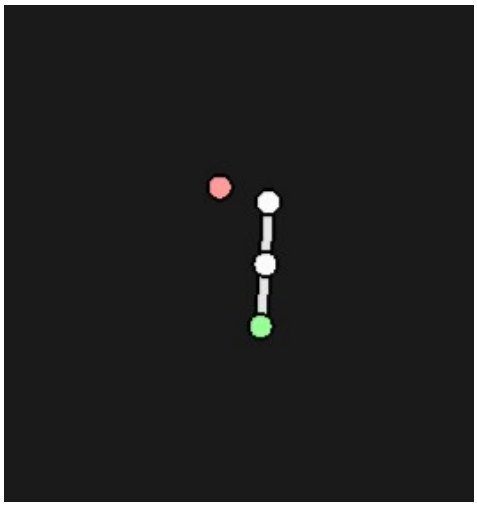


Figure 14: Robotic arm with 2 links, the termination point of arm is green and the objective point is red

The total return is calculated for every step taken, the value is negative for every step that the arm was unable to reach the determined point and is defined as the euclidean distance between arm termination and point. If it

can effectively reach the target point, a positive reward of $+5$ is given. The possible termination conditions are two: the steps taken are equal to 250 and the target point is reached.
The state tuple is composed, in the case of 2 links, as (target x, targe y, end effector x, end effector y, link angle 1, link angle 2) in the case of more links the tuple length increase adding more link angles. The action tuple has a length equal to links and in the case of 2 is : (link 1 velocity, link 2 velocities).
There isn't an exact condition for which the environment is considered solved, but from multiple runs and afterwards the environment rendering, it was decided to be considered solved when the mean reward was above $-40$ for the case of 2 links. Differently from the cartpole environment, the robotic arm can be multi-discrete or continuous and for the application, it was decided to use the continuous one and use the Soft-Actor Critic (SAC) by substituting a complete neural network approach with a hybrid which uses the VQA.

### 2.3.2 Quantum SAC

A paper has been already published which uses a quantum SAC approach to solve the pendulum-v0 environment, the details can be found in [23] and the pseudocode is reported in 2.3.2. So from the pseudo-code, it is possible to understand that there 5 components: an actor policy, two action-value approximators and two called target action-value approximators. The reason for which the target components must be similar to the DQN case, is because usually two consecutive states are not independent and require action-values to be it to correctly update the weights an independent copy is required. The reason for which both action and target components are double can be explained by other advances of DQN which showed that using two components and taking a minimum of the two improves performance and stability. The final component which is an actor policy, for the case of SAC this component will output two values: mean and variance of a gaussian distribution. From this the action will be sampled random meaning that this approach is statistical, it can be converted to deterministic by using the mean extracted from the component.
This policy actor in the paper is defined as a quantum-classic hybrid approach that differently from DQN tries to incorporate Neural Networks layers and Variational Quantum Algorithm. An image showing the actual structure present on the paper is 15 .
So of the 5 components present in this algorithm, only one uses the VQA with NN, this can be a limitation and afterwards, the results use an algorithm for which every component applies it. All the tests showed, are the ones with an environment that uses 2 links, this is due to a lack of time and resources.

**Algorithm 4** Variational Quantum Sac

**Require:** initial policy parameters $\theta$, initial action-value estimate parameters $\phi_1$ and $phi_2$, $\gamma$, $\alpha$, $\rho$, empty experience replay $D$.
Initialize the hybrid quantum-classical policy network with $\theta$.
Initialzie two action value networks with $\phi_1$ and $phi_2$ respectively.
Set target-action value networks parameters: $\phi_{targ,1} \leftarrow \phi_1$ and $\phi_{targ,2} \leftarrow \phi_1$.

**for** each time-step **do**
Observe state $S$, select action $A \sim \pi_\theta(\cdot|S)$ and execute $A$ in the environment.
Observe next state $S'$, reward $R$, and binary done signal $d$ to indicate whether $S'$ is terminal state or not.
Store $(S, A, R, S', d)$ in $D$.
Reset the environment if $d = 1$.
Sample a batch of transitions $B = (S, A, R, S', d)$ from $D$ randomly.
Compute target values $y(R, S', d) = R+\gamma(1-d)(min_{i=1,2}Q_{\phi_{targ,i}}(S', A')-\alpha \log \pi_\theta(A', S'))$ where $A' \sim \pi(\cdot, S')$.
Update $\phi_i$ by minimizing: $\mathbb{E}_B[(Q_{\phi_i}(S, A) - y(R, S', d))^2]$ for $i = 1, 2$.
Update $\theta$ by maximizing: $\mathbb{E}_B[min_{i=1,2}Q_{\phi_i}(S, \tilde{A}_\theta) - \alpha \log \pi_\theta(\tilde{A}_\theta|S)]$.
Do a soft update for target action-value networks: $\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1 - \rho)\phi_i$ for $i = 1, 2$.
**end for**

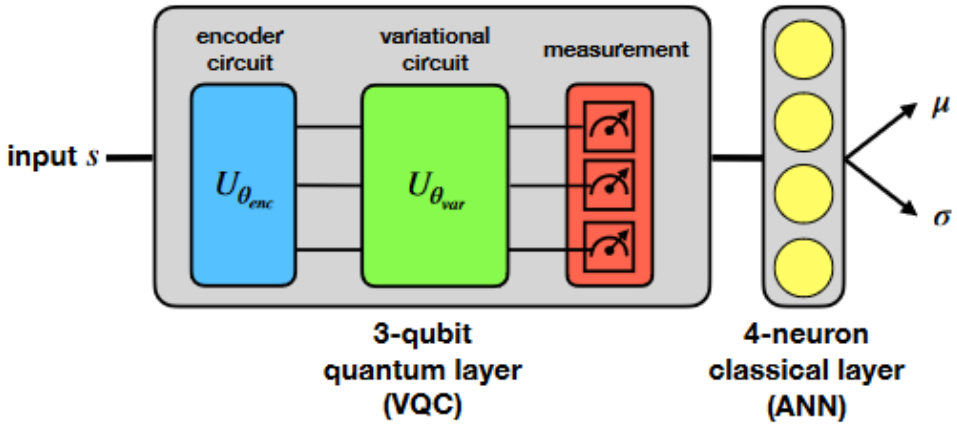



Figure 15: Hybirid actor policy architecture, image taken from [23]

### 2.3.3 Results with hybrid actor

Using the previous ansatz and algorithm it was decided to apply the hybrid architecture only on the actor component applying 4 and 5 layers giving a total of respectively 100 and 112 parameters. To confront it with a NN two tests were carried out using 149 and 164 parameters leading to this interesting plot. The runs could not be repeated for the lack of time and resources required, in fact, the hybrid architecture took almost 19 hours to reach this point. Even if the classical one took only 4, to benchmark them it would require at least 1 week and as it will be noticed this kind of solution is not optimal as it will be later noticed. One of the reasons for which these runs weren't repeated multiple times is the time and resources required, in fact using multiple CPUs or applying in the future on a GPU may reduce significantly this time. Unfortunately at the time of writing, this is not possible, but it may be applied in the future.
Confronting results obtained using hybrid and classical components gives:

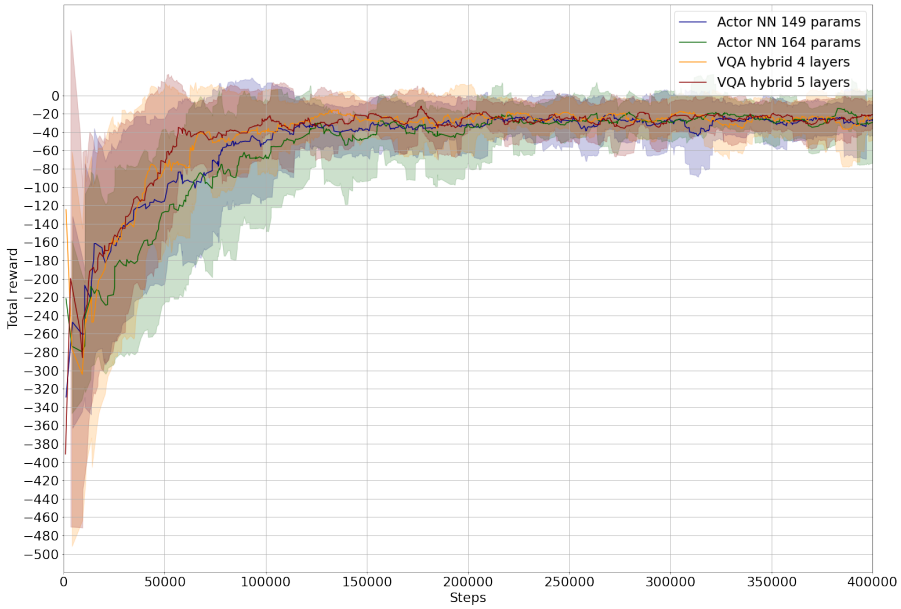



Figure 16: Moving average with a time window of 30 with a mean and standard deviation of the trend.

From this an interesting fact can be noticed: all of the tests have more or less the same trend and reach the same return at around 250000 steps. As can be seen, the hybrid algorithm presents a higher steep curve confronting the classical ones and can reach the plateau slightly faster, but there isn't a clear quantum advantage due to the similar number of parameters and time steps taken. This is quite suspicious and could mean that the critical component may not be the actor, but the critic. To confirm this suspect notice the plot of only the actor component that uses Neural Networks:

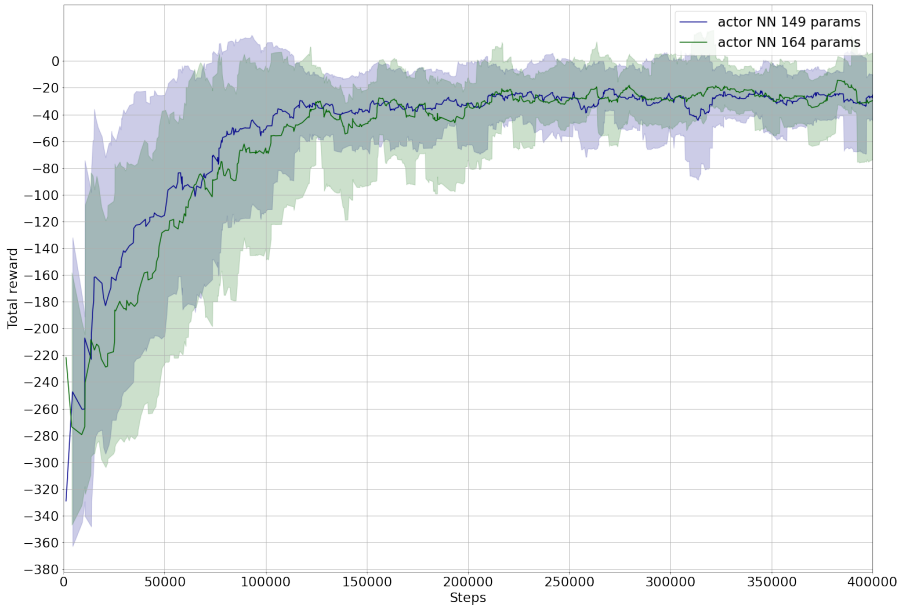



Figure 17: Plot of the runs that uses neural netowrks for all the components.

As it can be noticed an increase on the number of parameters doesn't determine a great variation, instead if a plot using the hybrid algorithm and reducing the critic component size using same architecture the following plot is obtained:

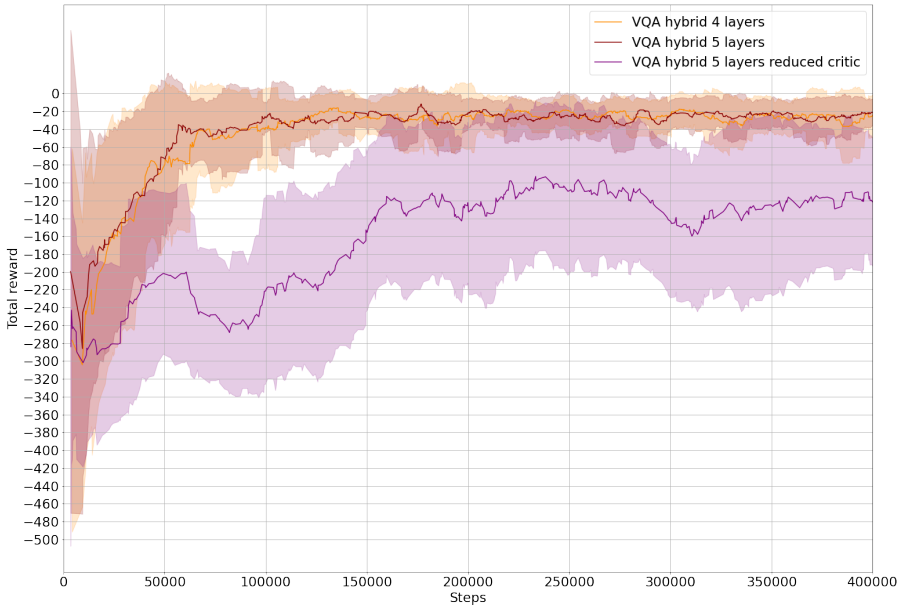



Figure 18: Plot confronting vqa with reduced parameters of critic and previous ones.

So effectively the most impacting component on the ability to reach the goal of this algorithm is not the actor, but the critic ones! Here can be found the table containing information on the structure of components where it can be noticed that except for the number of params the test runs use almost everywhere the same exact hyperparameters:

| Hyperparameters | VQA 4 layers | VQA 5 layers |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6, VQA, (1,1)) | (6, VQA, (1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 100 | 112 |
| critic neurons | (8,64)(64,64)(64,1) | (8,64)(64,64)(64,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 4608 | 4608 |
| total params | 18532 | 18544 |

| Hyperparameters | VQA 4 layers reduced | VQA 5 layers reduced |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6, VQA, (1,1)) | (6, VQA, (1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 100 | 112 |
| critic neurons | (8,16)(16,16)(16,1) | (8,16)(16,16)(16,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 384 | 384 |
| total params | 1636 | 1648 |

| Hyperparameters | Actor NN 149 | Actor NN 164 |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6,24,(1,1)) | (6,26,(1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 149 | 164 |
| critic neurons | (8,64,64,1) | (8,64,64,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 4608 | 4608 |
| total params | 18581 | 18596 |

The reason for which the total parameters are so high links to the fact that 4 critic components are required for the algorithm to work and learn.

### 2.3.4 Results with hybrid actor and critic

Now that the previous test has demonstrated that critic is the component that determines the most performance the following test will use the hybrid algorithm used for the actor and apply it to the critic. During the first tests, we noticed that the previous structure used for the actor isn't good for the critic due to maybe a high non-linearity of the map required. For this reason, the previous structure was modified by adding a hidden layer before the output to increase the expressivity of non-linearity. This is the schematic:

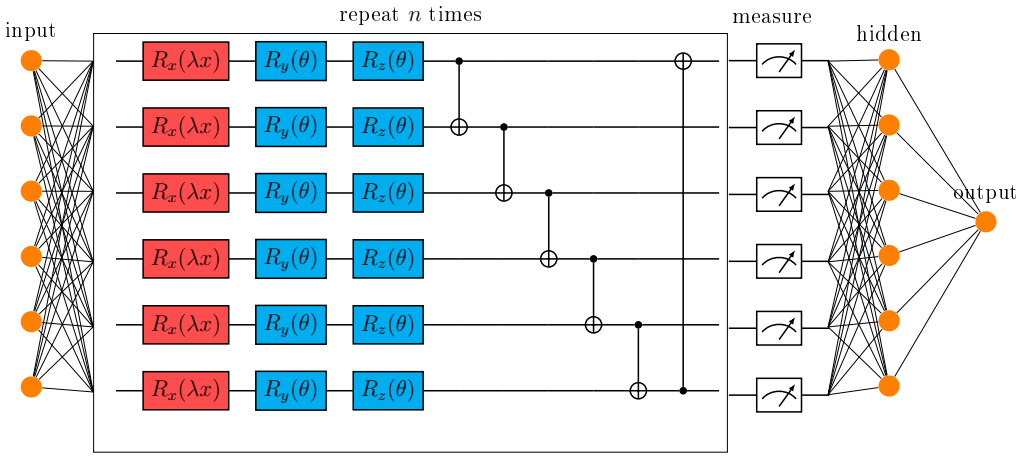



Figure 19: Structure for critic hybrid

As it can be seen from the plot 20 by converting all components to hybrid architecture it was able to have a performance very similar to the ones with the only hybrid actor. As it will be seen later this means that a smaller component that uses few parameters can reach the same performance as one with more on the Neural network. The reason for which this line is interrupted after only 250000 steps is mainly due to the high time required, in fact, to reach that step the fully hybrid algorithm took more than a week and after concluding that the algorithm was performing very similarly to the actor hybrid it was stopped.

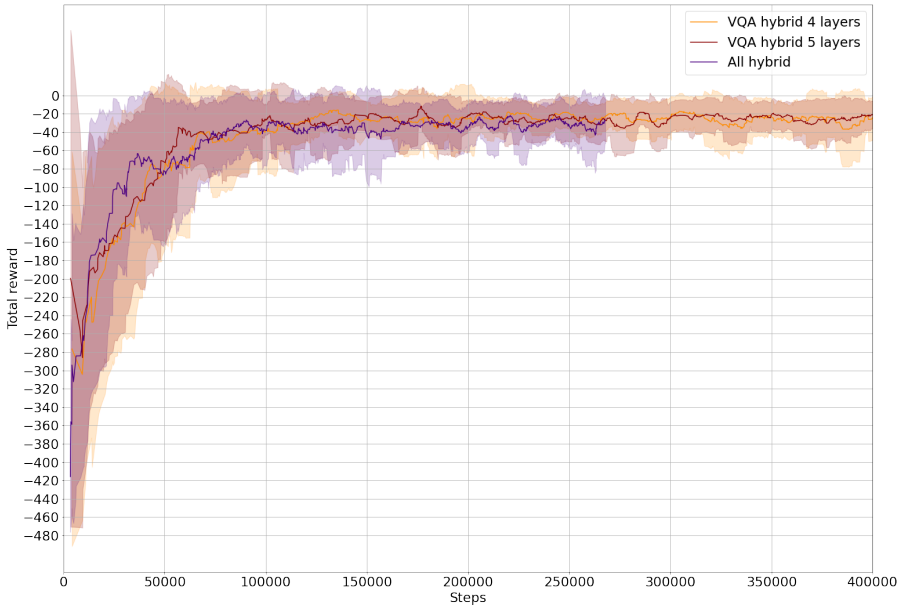



Figure 20: Comparision between only actor hybrid and all components hybrid.

Afterwards to have a detailed cased for which multiple configurations with different sizes of the neural network were conducted to understand what kind of advantage was expected to confront the run of all-hybrid components and all neural networks. As it can be noticed from 21 increasing the size of Neural Networks and keeping the actor equal for all runs doesn't mean that there is an increase in performance and unfortunately, the run that uses the least number of neurons is not able to reach the objective to have a score at least -40 as average. The choice of using 21 neurons is not casual, but it has been decided to have a similar number of parameters equal to the run that uses all hybrid components.

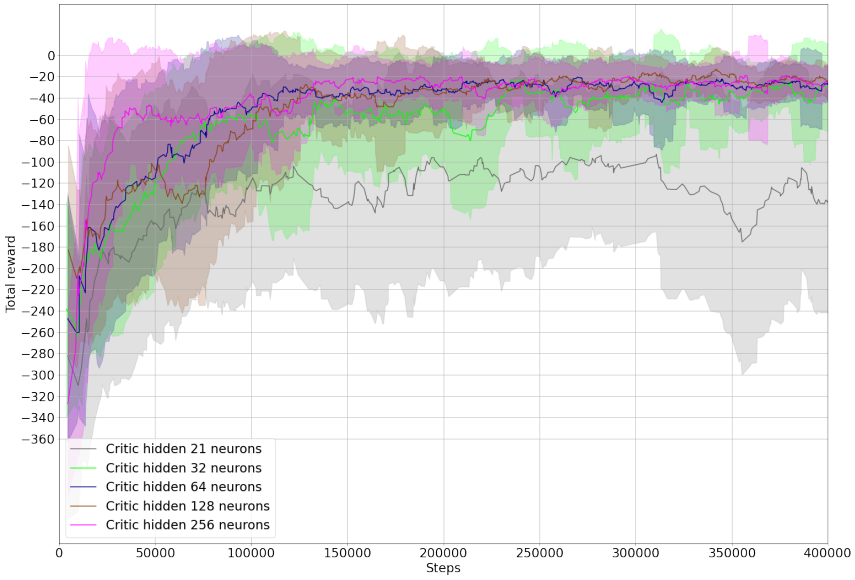



Figure 21: All classical runs with an actor component of 149 parameters

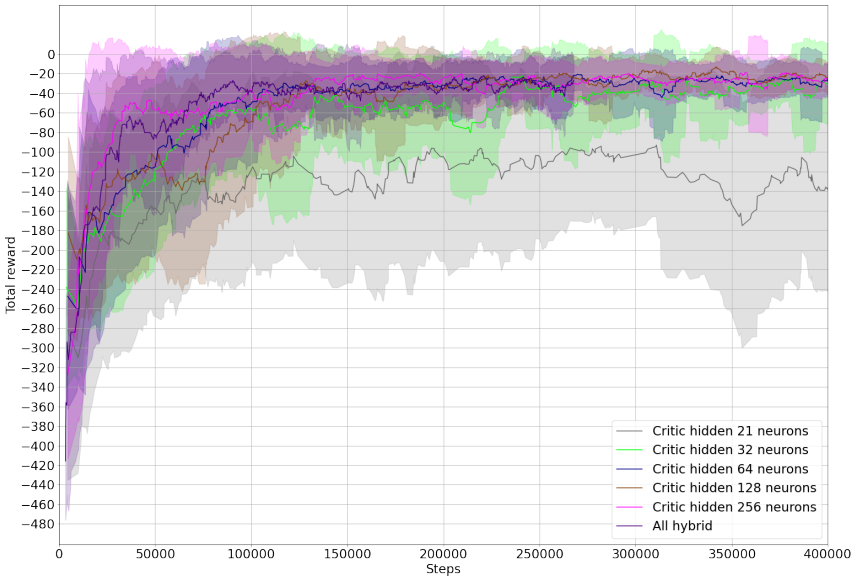



Figure 22: Plot of classical runs and all-hybrid

So by looking at 22, an important result can be obtained, the all-hybrid run can reach a similar performance of all neural network components that use a higher number of neurons to reach the objective with a similar trend and time steps required. **But if the neural network model uses a similar parameter number of the all-hybrid model then it will not be able to reach the objective**. Furthermore, all the test runs have almost all the same hyperparameters for every one of them, to have a more equal confrontation of these methods. The configurations used for these runs are:

| Hyperparameters | Critic hidden 21 neurons | Critic hidden 32 neurons |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6,24)(24,(1,1)) | (6,24)(24,(1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 149 | 149 |
| critic neurons | (8,21)(21,21)(21,1) | (8,32)(32,32)(32,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 651 | 1344 |
| total params | 2753 | 5525 |

| Hyperparameters | Critic hidden 64 neurons | Critic hidden 128 neurons |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6,24)(24,(1,1)) | (6,24)(24,(1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 149 | 149 |
| critic neurons | (8,64)(64,64)(64,1) | (8,128)(128,128)(128,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 4672 | 17536 |
| total params | 18837 | 70293 |

| Hyperparameters | Critic hidden 256 neurons | All hybrid |
|---|---|---|
| $\gamma$ | 0.99 | 0.99 |
| $\alpha$ | 0.2 | 0.2 |
| learning rate | 0.0003 | 0.0003 |
| memory size | 1000000 | 1000000 |
| optimizer | Adam | Adam |
| actor neurons | (6,24)(24,(1,1)) | (6,VQA(5 layers),(1,1)) |
| actor act. func. | (linear,relu,linear) | (linear,relu,linear) |
| actor params | 149 | 112 |
| critic neurons | (8,256)(256,256)(256,1) | (8,VQA(20 layers),8,1) |
| critic act. func. | (linear,relu,relu,linear) | (linear,relu,relu,linear) |
| critic params | 67840 | 650 |
| total params | 271.509 | 2712 |

Focusing on the number of parameters and the previous plots a conclusion can be extrapolated: to have similar trends between the hybrid model and neural networks, which are the ones with 64 neurons for hidden layers or more, it is required to have at least 7 times the number of parameters!. In fact from the previous table hybrid model uses around 2700 parameters, while the neural network with 64 uses 19000. Furthermore, as it can be seen from plot 22 even the biggest neural network is not always able to follow the trend of the hybrid and reach the objective with the same number of steps and notice that uses around 100 times the parameters of hybrid.

# 3 Conclusion

## 3.1 Results obtained

So in this thesis after a first dedicated to introducing the basis of Deep Reinforcement Learning and quantum mechanics it was later presented the Variational Quantum Algorithm to simulate and replace Neural Networks, observing what kind of advantage was possible to obtain in the context of reinforcement learning. To understand if there was a quantistic advantage and on which aspect was present the ansatz proposed in the paper [22] was tested as a quantum variation of the Deep Q-Network on Cartpole environment.
To compare it, a model with neural networks was run with the most number of hyperparameters equal between the two. From results obtained and plot 12, it was demonstrated that an effective advantage was obtained: an impressive smaller number of parameters was required to reach the objective and a neural network with the same number of weights was unable to do it.
This suggested that there is a more powerful expressivity and approximation capability compared to the neural network in this case. Unfortunately, such kind of environment is extremely simple to solve and doesn't completely demonstrate this kind of ability and extension to more complex cases.
For this reason and to use the possible advantages and applications in the industry a different environment was tested: the robotic arm. This required changing the algorithm used previously due to continuous state and actions which the DQN is unable to deal for its discrete nature.
The new algorithm that was used is called Soft-Actor Critic (SAC), differently from the DQN it is policy gradient-based and uses 5 components to learn. These are actor policy which must calculate the mean and standard deviation, 2 action values and 2 target action values that must update policy and approximate the correct functions to learn and work. From a previous paper, [23] it was defined as a partial quantum SAC which demonstrated an advantage on the number of parameters by using a hybrid actor component. From the test conducted later on and in particular from plot 18 an important conclusion was extrapolated: the critical component that determines the performance of the algorithm is the action-values components. So for this conclusion, another run using all components as a hybrid was conducted with an additional classical layer for the critical components to introduce enough non-linearity. Furthermore, more tests using all components leveraging neural networks were run increasing every time the dimension of neurons on hidden layers. Confronting the trend on plot 22 it was noticed that to have a similar trend for the classical and hybrid run, several more parameters between 7 to 100 times and not always the steps required needed to be similar. Furthermore running an example where all components using the neural network have the same parameters number of hybrid, this model

was unable to reach the goal of environment suggesting a quantum advantage on the number of parameters required and ability of expressivity useful for industrial applications.

## 3.2 Future directions

Possible future directions that should be interesting to take are:

- applies multiple runs on the hybrid algorithm possibly reducing the time required to run.
- introduces the noise on algorithms execution to understand the possible non-linearity and performance impacts on the ability to learn and the difference in weights.
- develops better ansatz for the VQA to reduce the dependence of non-linearity by neural network layers and minimize the depth of the circuit.
- test this VQA on a real quantum computing using the parameter-shift method to calculate the gradients and use it later on a CPU to optimize the parameters showing performance and time reduction compared to this run on a simulator.
- tests these algorithms on multiple environments to see if are better or worse compared to "classical" ones.
- extending the introduction of VQA on other algorithms of DRL such as PPO, DDPG and others to confirm quantum advantage.

# Bibliography


[1] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, 2018. ISBN: 1788834240.

[2] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511976667.

[3] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[4] Ronald J. Williams David E. Rumelhart Geoffrey E. Hinton. "Learning representations by back-propagating errors". In: *Nature* 323 (1986), pp. 533–536. DOI: https://doi.org/10.1038/323533a0.

[5] Halbert White Kurt Hornik Maxwell Stinchcombe. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 5 2 (1989), pp. 359–366. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[6] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[7] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-learning". In: *CoRR* abs/1509.06461 (2015). arXiv: 1509.06461. URL: http://arxiv.org/abs/1509.06461.

[8] Richard S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: NIPS'99 (1999), pp. 1057–1063.

[9] Ziyu Wang et al. "Sample Efficient Actor-Critic with Experience Replay". In: *CoRR* abs/1611.01224 (2016). arXiv: 1611.01224. URL: http://arxiv.org/abs/1611.01224.

[10] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *CoRR* abs/1707.06347 (2017). arXiv: 1707.06347. URL: http://arxiv.org/abs/1707.06347.

[11] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *CoRR* abs/1801.01290 (2018). arXiv: 1801.01290. URL: http://arxiv.org/abs/1801.01290.

[12] Edwin B. Wilson. "Probable Inference, the Law of Succession, and Statistical Inference". In: *Journal of the American Statistical Association* 22.158 (1927), pp. 209–212. ISSN: 01621459. URL: http://www.jstor.org/stable/2276774 (visited on 06/17/2022).

[13] M. Cerezo et al. "Variational quantum algorithms". In: *Nature Reviews Physics* 3.9 (Aug. 2021), pp. 625–644. DOI: 10.1038/s42254-021-00348-9. URL: https://doi.org/10.1038%2Fs42254-021-00348-9.

[14] Maria Schuld and Francesco Petruccione. "Variational Circuits as Machine Learning Models". In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 177–215. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_5. URL: https://doi.org/10.1007/978-3-030-83098-4_5.

[15] Edward Farhi and Hartmut Neven. *Classification with Quantum Neural Networks on Near Term Processors*. 2018. DOI: 10.48550/ARXIV.1802.06002. URL: https://arxiv.org/abs/1802.06002.

[16] Marcello Benedetti et al. "A generative modeling approach for benchmarking and training shallow quantum circuits". In: *npj Quantum Information* 5.1 (May 2019). DOI: 10.1038/s41534-019-0157-8. URL: https://doi.org/10.1038%2Fs41534-019-0157-8.

[17] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. "Effect of data encoding on the expressive power of variational quantum-machine-learning models". In: *Physical Review A* 103.3 (2021). DOI: 10.1103/physreva.103.032430. URL: https://doi.org/10.1103%2Fphysreva.103.032430.

[18] Jarrod R. McClean et al. "Barren plateaus in quantum neural network training landscapes". In: *Nature Communications* 9.1 (Nov. 2018). DOI: 10.1038/s41467-018-07090-4. URL: https://doi.org/10.1038%2Fs41467-018-07090-4.

[19] Edward Grant et al. "An initialization strategy for addressing barren plateaus in parametrized quantum circuits". In: *Quantum* 3 (Dec. 2019), p. 214. DOI: 10.22331/q-2019-12-09-214. URL: https://doi.org/10.22331%2Fq-2019-12-09-214.

[20] Adrián Pérez-Salinas et al. "Data re-uploading for a univeral quantum classifier". In: *Quantum* 4 (Feb. 2020), p. 226. DOI: 10.22331/q-2020-02-06-226. URL: https://doi.org/10.22331%2Fq-2020-02-06-226.

[21] Greg Brockman et al. *OpeanAI Gym*. 2016. eprint: arXiv:1606.01540.

[22] Andrea Skolik, Sofiene Jerbi, and Vedran Dunjko. "Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning". In: *Quantum* 6 (May 2022), p. 720. DOI: 10.22331/q-2022-05-24-720. URL: https://doi.org/10.22331%2Fq-2022-05-24-720.

[23] Qingfeng Lan. *Variational Quantum Soft Actor-Critic*. 2021. DOI: 10.48550/ARXIV.2112.11921. URL: https://arxiv.org/abs/2112.11921.