Prospects of quantum computing approach to reinforcement learning

Matteo Conterno

July 3, 2022

Abstract

Reinforcement learning is one of three major techniques that allows a model to learn, especially this technique focus on how to create 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 by using quantum circuits with neural networks; analyzing and explaining how it is possible to create hybrid algorithms that exploit the advantages of both the classical and quantum algorithm, expanding the applicability not only on simulators, but even on quantum devices such as Rigetti and IonQ's processors through the amazon AWS braket service. Lastly it is analyzed how the quantum noise influence an optimal model obtained through a simulator, by testing it on a quantum processor to understand if it is possible to use the hybrid algorithm on the actual devices.

Contents

Acronyms

AI Artificial Intelligence. 5

DQN Deep Q-Network. 11

DRL Deep Reinforcement Learning. 5, 11

i.i.d. indipendent identical distributed. 10

MDP Markov Decision Process. 6, 7, 11

NN Neural Networks. 5, 9, 10

PPO Proximal Policy Optimization. 12

QC Quantum Computing. 5

QRL Quantum Reinforcement Learning. 5

SAC Soft-Actor Critic. 12

VQA Variational Quantum Algorithm. 5

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 already expert on these subjects fell free to skip at the next section.

1.1 Approaches of learning

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

- Data
- Model
- Approach of learning

Data is necessary in every AI model, the amount 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 his task using some learning approach. In order to check if the model has correctly learned, it will be later tested on unseen data and evaluated to understand if it is able to replicate the performances given a similar dataset. The approach of learning specify how the model can learn to complete his task. At the moment there are three major ways:

- **Supervised learning** : the data is already labeled, this means that we can define when the model is incorrect or correct.
- **Unspervised learning** : the data is not labeled, this means that we can't define easily when the model is correct or incorrect and the model must uncover some pattern.
- **Reinforcement learning** : an agent interact with the environment in order 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. In this thesis the focus is on what kind of advantage can be obtained by using a quantum algorithm such as Variational Quantum Algorithm (VQA), this technique can be even used in other context and applications. Futhermore it has been demostrated

that VQA and NN share some similarities and properties. The main drawback is the fact that when a VQA is trained on a classical device, there is a major overhead of time due to simulation of the quantum circuit implemented and the number of qubits is limited.

1.2 Reinforcement Learning

As said earlier a reinforcement learning approach consist of creating the most optimal agent able to complete a predefined task by interacting with the environment and taking some action. Questions arises about: how can be defined if an action is good or bad? How to model a dynamic environment? The answer is given by using a statistical model called Markov Decision Process (MDP) .

1.2.1 Markov Decision Process

In order to model a dynamic environment where an action can influence the sistem it is necessary to use the Markov Decision Process (MDP) which is an extension of Markov chains; these are a stochastic model that is able to describe a sequence of possible events that satisfy the Markov propriety, that is each event depends only on the previous event.
It is necessary to note that an MDP is based on Markov Chains that not only model the states, but even the time and this can be defined as continuous or discretized. MDP is an extension of Markov Chains because the agent can influence the state of environment and outcome, so a framework is necessary to define even 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.
The interaction between agent and environment can be defined by time, a discrete step implies to view it as distinct separate points in time uniquely definied and with a single state value can associated. The sequence of observation over times form a chain of states that is called **history**, this will be

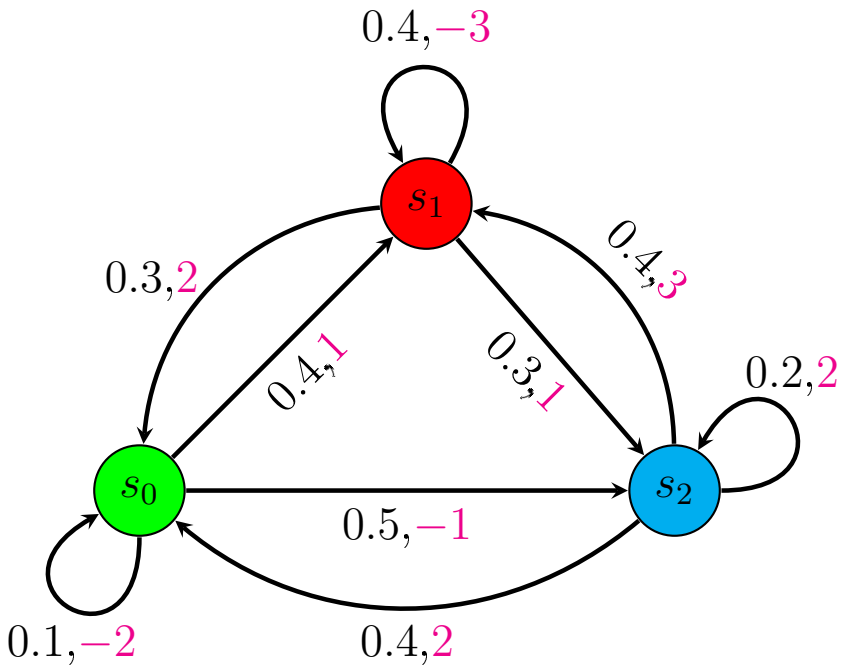


Figure 1: Markov decision process graph, transition probability is black and reward is in magenta.

particularly important because it will be used later to define the transition probability in order to model the interaction with environment.To include the reward element of an MDP accumulated from present and future a new quantity need 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 γ 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 γ is equal to 0 only the immediate reward will be counted and when it is 1 infinite future steps will be considered. Usually the literature use $\gamma \in [0.9, 0.999]$ in order to gradually consider less relevant further time steps rewards thanks to the k-power of γ inside 1. The RL learning approach has the objective to maximize the **return** quantity, but this equation is not useful for the agent due to the fact that it considers every possible chain that can be observed using the Markov reward process. This means that it can vary widely even for the same state however 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 considers the reward and state but they are not sufficient to model environment and agent, so it is necessary to define a set of rules to control the agent behaviour considering that the objective of RL that is to maximize the return, for these reasons a new quantity must be defined **policy**.This is formally determined as 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 in order to introduce randomness of the agent that will be useful during the training phase, if policy is fixed the transition and reward matrixes can be reduced using policy's probabilities and decreasing the action dimension.

1.2.2 Q-learning

As stated before the objective of RL is to maximize the return, problem is how to approximate the best optimal policy and values state in order to define the correct actions for given state. Fortunately the **Bellman optimality equation** is able to estimate approximately the best action to take on deterministic and statistical case.The equation for a general case is defined as:

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

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(π) 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 we don't know how the value of actions, rewards, transition probabilities.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 Tabular Q-learning present major drawbacks such as:

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

In order to resolve these drawbacks a new type of Q-learning algorithm called **Tabular Q-learning** was invented. The main difference is the lack of necessity to iterate over all the states to optimize because only the ones obtained from the environment will be used for optimization. Futhermore the table will only contain the $Q(s, a)$, this does not unfortunately resolve completely the drawback to have a large memory table, but at least reduce 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 and it will be used to update itself and to define which action need to be taken following the policy. There is still one major problem though, this algorithm will struggle if there is a large count of the observables state set. This, in many real life situation, is quite common for example on the Cartpole environment of OpenAI gym that we are going to use there are only 4 states, but the interval of values that can be taken is enormous. A solution of 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 power of Neural Networks (NN) it is possible to approximate any kind of linear or non-linear function given enough data. Fortunately the amount of data is almost limitless thanks to the fact that if new data is needed, the only requirement is to interact more with the given environment.So now, it will be introduced the **Deep Q-learning** algorithm.

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 has been introduce by [1] who largely inspired by the biological brain with neurons and connections. At that time many limitations were observed due to the single layer architecture and small amount of neurons that the hardware was able to simulate, only thanks to the backpropagation algorithm introduced in [2] it was later possible to train neural networks with more layers and neurons. Furthermore as demonstrated by [3] a neural network with only one single hidden layer is able to approximate any kind of linear or non-linear function, but the paper doesn't tell how many neurons are required. Even if the algorithms were introduced in the last century only in the last two decades thanks to hardware advancements such as GPUs and large amount of datas from the first databases it was possible to train effeciently these neural networks. Fortunately reinforcement learning can increase the amount of data by increasing the time of interaction with the environment, but the question on how many neurons a neural networks requires to approximate doesn't have a solution, but only indications. Now that the neural networks have been briefly introduced it 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 is able to solve the problem of iteration over time, but it still struggles with situations when the count of observable state set is very large. This is a typical situation in real life where the set of states can be even infinite, solutions have been proposed such as use bins to discretize, but in many cases it did not result successful. A better solution is to create a nonlinear representation, instead of a linear one, that maps both state and action to a value. This is commonly called in machine and deep learning as "regression problem" and in general 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 reached.

Even if it looks simple, some modifications are required in other to ensure convergence. First of all when it is necessary to store the experience we need to find a strategy to explore the environment and optimize the current approximation, this is often referred to "exploration versus exploitation". In order to have an efficient solution it is good to behave randomly at the beginning because Q-value approximation is bad at that time and later start to act using the Q-value obtained. Usually it is used the **epsilon-greedy method** where a random probability value is sampled from a uniform distribution and confronted with a probability ϵ that tells the algorithm to behave randomly. ϵ is usually initialized to 1 so that any kind of random value sampled is smaller to ϵ so that the model behave randomly and later ϵ is decreased to a final small value so that the model will not always behave randomly but use the Q-values approximated. After that this problem is solved, there is another matter linked to how an optimizer algorithm work on a NN. The gradients calculated from input data requires that the data are i.i.d., otherwise there would be incorrect estimations of the correct direction on which the parameters must be updated. Furthermore the data must not be completely randomic but must reflect the past actions taken in order to have an experience that tells which action are worst than others. To solve this, it is necessary to create a **large memory buffer** in order to store the past actions taken both randomic and by exploiting the agent. Last problem that needs to be addressed is due to the steps correlation, when we calculate the loss we use $Q(s, a)$ and $Q(s', a')$ using the same NN, this can be a problem 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 following pseudo is taken from the original paper that was tested on atari games [4], after this paper numerous variations have been proposed in order to improve convergence and reduce time of training, such as Double Q-Net [5] and many others. Due to the limited resources and 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.

The 1.3.1 gave a breakthrough on this field thanks to the fact that was able to achieve a human-like, and in some cases even better, performance on multiple games of the atari. Thanks to this Deep Reinforcement Learning has been tested and applied in other contexts such as finance, medicine, robotics and many others. Before we introduce quantum computing it is necessary to explain and describe another algorithm that is able to 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: ϵ 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 ϵ 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 we have illustrated is focusing mainly on approximating the value of actions(Q) and the value of state(V) so that afterward the choice on which action to take is based on a greedy approach: the best action to take is the one that maximize the return. This is not incorrect, but in some cases it may not be good due to the environment consequences. For this reason in those cases it is better to focus on how to define the agent behaviour in order to consider vene the possible alternatives. Another reason why these methods are used is due to the fact that can introduce **stochasticity** and can 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** in order to work with it, the most common way is to use a probability distribution over the possible actions. This gives an additional advantage to neural network which is to have a smooth representation, in other words if a variation is applied on the weights outputs variate. It is now time for the final step, to find a way to optimize the weights in order to improve the policy. Thanks to the policy gradient theorem, it is known that the formula we need to use is:

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

the complete demonstration of the formula and application on Deep Reinforcement Learning can be found on [6]. The meaning 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. Simply we are trying to increase the probabilty of good actions and rewards, at the same time decreasing the probability of bad actions and outcomes. Finally, the $\mathbb{E}$ is expectation value due to the fact that will be used with multiple values. There are some drawbacks for this approach such as a high gradients variance that can influence the learning and exploration at the beginning of training and to avoid falling in a local minima it is necessary to introduce some kind of uncertainty or *entropy*. There is another problem of policy gradient methods and that is the fact that are usually less sample-efficient and this implies a bigger numeber of iterations to solve the environment. In order to tackle this problems new algorithms have been defined, notably Actor-Critic (A2C) [[7]], Proximal Policy Optimization[8] (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 one can be simplified as the necessity to improve the behaviour and values of states approximated in order to solve the environment. In order to achieve this two neural networks are used: one to approximate the policy called **policy net or actor** and one to approximate the value of state called **value net or critic**. The problems of this algorithm is the fact that is on-policy and sensible to

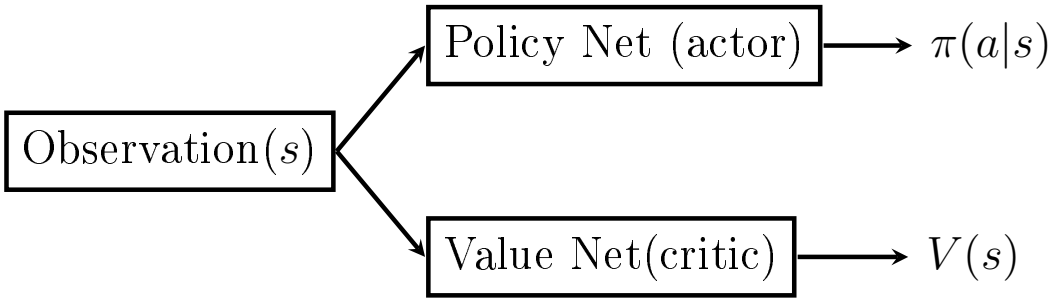


Figure 2: Schematic on actor-critic architeture

hyperparameters, in order to improve performance, stability and efficiency the SAC was created. SAC is able to do it introducing multiple components such as :

- a large buffer memory is needed to efficiently update the weights using

previous history increasing in this way 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 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, to do this a loss value of state (J_V) and action (J_Q) function must be defined.

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

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

- 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(ϵ_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 [9], to avoid confusion the 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 first to introduce the idea of a Turing machine that used quantum mechanics to make calculation. Richard Feynman and Yuri Manin suggested to use it for efficient simulation on quantum mechanics systems. Futhermore 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 current concept. In order to introduce quantum computing it is necessary to understand what is a quantum bit, circuit and logic gate, keeping in mind that even if the following representation is an abstraction there is a physical implementation for all this components.

1.4.1 Quantum bits

Quantum bits or qubits are the analogous of classical bits which are used as a fundamental concept for computation and information forming a computational bases 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 the 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 α and β are complex even known as *amplitudes*, so a qubit can be represented as a two-dimensional complex vector space. A fundamental condition is applied on these numbers to respect the statistical view of quantum mechanics, when a qubit is measured a probability of being on the quantum state is associated. These probabilities can be represented as $|\alpha|^2$ 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 in order 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 to construct their representation $|\nu\rangle$ using the dot product as follows:

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

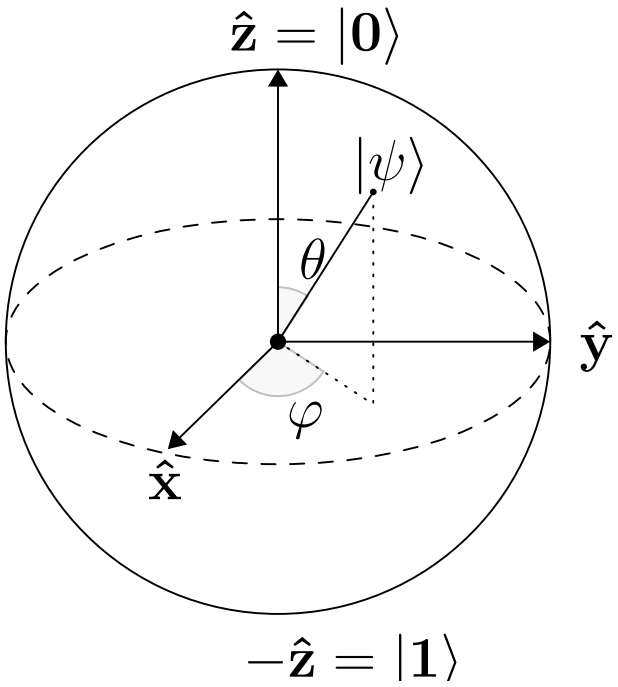


Figure 3: Representation of a qubit the Bloch sphere

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 we had 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 certainity 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 at the basis of many quantum algorithms and has been proven to be valid even on distance of galaxies. This doesn't violate Einstein 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* amd *logic gates*, quantum computing does the same. Main difference is that for example quantum gates such as the equivalent quantistic NOT act linearly instead of non-linearly, the reason is due to the fact that if non-linearity was allowed then empirically there would be paradoxes, such as violations of the second law of thermodynamics, faster than light communications and time travels. Taking advantage of the linear proprierty it is possible to give a matrix form of such as NOT, in fact a NOT gate classicaly invert 0 to 1, what happens on 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 equal to 1. This can be translated to the condition that any matrix U representing a quantum gate, must be unitary. This can be formalized using the *adjoint* of $U^\dagger$, which means complex conjugating and transposing the elements of U, as the condition $U^\dagger U = 1$. This can be even interpreted as the fact that any kind of quantum operation applied is reversible, so by applying another one

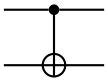
is possible to return at the initial state, which quantistically is the adjoint of operation applied. The most used single qubit gates are the following:

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

there are other gatex such as S, T and I, for the thesis they will not be necessary, but if interested feel free to check the book(put reference Nielsen Chang).The interesting part is that any kind of single qubit gate can be obtained from the generic 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 as an example 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**. These, differently from the single qubits using multiple of them, for example one that will be used extensively is the *controlled-NOT* or CNOT gate which has the following form:

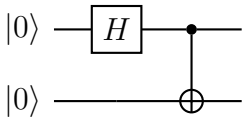


$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This gate function by using a control qubit which is the black dot and only if the control qubit is $|1\rangle$ applies to the target qubit $\oplus$ which is the addition modulo two operation that on a two basis computational state means to flip it. The horizontal line repreent the wire used to transfer the qubit. Formally CNOT is defined as:

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

The import point is that this circuit is like it applies an X gate, in fact it is possible to subsitute with other single qubit gate. 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 because it can introduce 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 it is not necessary to introduce onle last component the measurement gate, in fact after the qubits are manipulated it is necessary to extract the information on these. In order 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 consist of extracting information, but for the quantistic case this means 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 in order to have the correct expectation value of $< \sigma_z >$ it is necessary to repeat multiple times the measurement to sample the possible values, his is usually represented as S which is an abbreviation for *shots*. In order to have the correct expectation with an error of at most ϵ conventional statistics can be used for example using the Bernoulli distribution. 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 to a z-value. Estimating the error of a Bernoulli trial can be defined in different ways, but the most preferred ways is by using the Wilson score interval [10]. 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 phenomenon that needs to be considered when a quantum device is effectively used and that is **quantum decoherence**. The phenomenon is refeered to the effect of losing quantum coherence, this is obtained when the phase relation of the wave functions associated to different states is not valid anymore. This effects is mainly due to the fact that qubits are not completely isolated and interactions with environment introduce entanglement between them. The effect of quantum decoherence is problematic because implies 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 costant decays for statistically say when quantum decoherence affect the qubits. Application of quantum gates influence the system by increasing the possibility of decoherence, for this reason **quantum error correction** has been developed. Problem is that these algorithms for error correction requires a lot of qubits and actually there are not physical devices with this capacity. In order 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 quantistic, there are different ways to encode: *basis encoding, amplitude encoding, angle encoding.* There are even others in [11], but these will be the ones used in this thesis. Basis encoding is based on the fact that any bit is substituted with the qubit having corrisponding state value, for example 01001 becomes $|01001\rangle$ using this strategy. As it is possible to notice this representation is not qubit efficent because it requires the same number of qubits for the associated bit of value, but the routine don't require many operations. Another strategy of encoding is called *amplitude encoding*, which consist of taking the value of a classical vector x, normalize the values inside by normalizing the elements and associate to them

an amplitude, like in this exxample:

$$x \to \sum_{k=1}^{2^n} |x_k|^2 = 1$$
$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_{2^n} \end{bmatrix} \leftrightarrow |x\rangle = \sum_j x_j |j\rangle$$

this kind of representation requrires 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 angle the classical value.

2 Variational circuits applied to deep reinforcement learning

Variational circuit 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: find 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 concept that comes from quantum and classical algorithms, futhermore it can be applied to actual quantum devices with 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), due to the fact that 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 successfull in Reinforcement Learning (RL). A consideration that needs to be done is the fact that an ideal variational circuit must be small as possible in order to make it work on a quantum device and reduce the possibility of information loss due to decoherence, this type of circuit is referred as **shallow circuit**. An extensive paper that describes these circuit is [12].

2.1 Variational Circuit

Variational circuits have been fromulated by [13]-[14], but the idea was introduced years before. This circuit are based from the fact that gate can have an associated a parameter, the parametrized and unparametrized gates that will be used in the circuit will form an **ansatz**.In order 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 using a method called **parameter shift** as it will be seen later.
Thanks to the proprierties of quantum mechanics the entire circuit can be considered as a single unitary gate of the form $U(x, \theta)$ due to their dependecies on the input data. Usually the internal structure of $U(x, \theta)$ consist on of an embedding block $S(x)$ and a parametrized one $W(\theta)$ in order to decompose $U(x, \theta) = S(x)W(\theta)$, these blocks can contains quantum fixed gates. Graphycally this can be seen as:

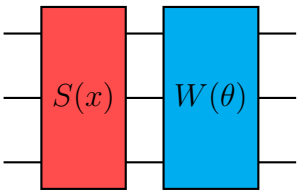


Figure 4: ususal decomposition of quantum circuit

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

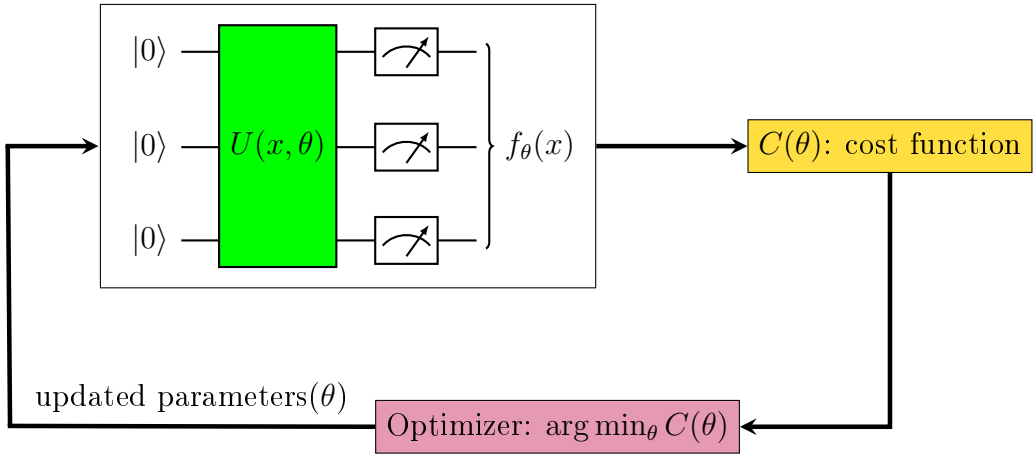


Figure 5: Schematic of a variational quantum algorithm

Furthermore these variational circuits can be interpreted, with minor conceptual changes, to 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 indicating $\mathcal{M}$ as an 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 formula 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:

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

In case the measurement applied is based on the Z-gate($\mathcal{M} = Z$) which will be the one iused 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 ϵ 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 quantity 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 (VQA) can be interpreted as supervised and unsupervised model 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 $\psi(x,\theta) = U(x,\theta)|0\rangle$. We 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:

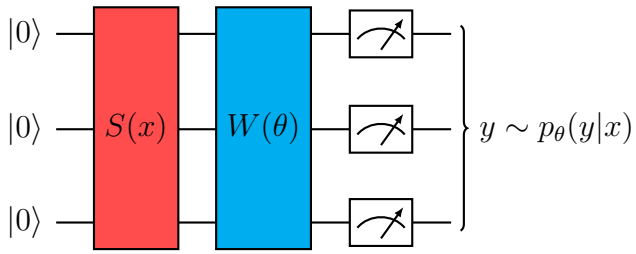


Figure 6: Variational quantum algorithm for conditional probability

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.

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:

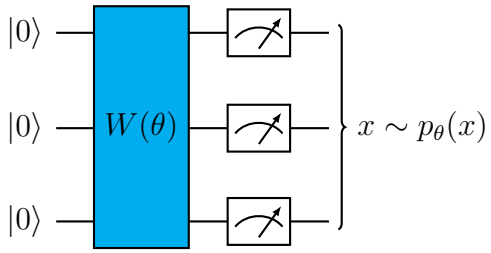


Figure 7: Variational quantum algorithm for unsupervised approach

So it is possible to notice that the only differences between the unsupervised and supervised model are two: the absence of embedding layer and data basis for measurement instead of the output basis.Probabilistic quantum model is strictly correlated to deteminsitc one due to the fact that $p_\theta(x) = |\langle x|\psi(\theta)\rangle|^2$

can be even seen as $\langle x| \left(|\psi(\theta)\rangle \langle\psi(\theta)|\right) |x\rangle$ with basis measurement of $\mathcal{M} = |\psi(\theta)\rangle \langle\psi(\theta)|$. This is important because it define a clear connection betwenn the insights on designs applied. Futhermore it is possible to notice that VQA are generative models, but it is difficult to extract this distribution due to the measurements requried. Lastly unsupervised quantum models are even 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

For generality it is possible to express the quantum circuit using in the model as an alternation between encoding an parametrized gates repeated for multiple layers as:

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

embedding gate can be defined as having the form $S_i(x_i) = e^{-ix_i G_i}$ where G_i without losing genrality 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 [15] demonstrate the linear combination of such models.

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 $e^{-ix_i G_i}$: Then f_θ can be written as*

$$f_\theta(x) = \sum_{w_1 \in \Gamma_1} \cdots \sum_{w_N \in \Gamma_N} c_{w_1 \ldots w_n}(\theta) e^{iw_1 x_1} \ldots e^{iw_N x_N}$$

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_θ 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 linearity if only the variational cirucit is applied, the frequencies are defined by the embedding layer and weights are dependent on parameters and frequencies. This means that for an expressive Variational Quantum Algorithm the embedding and parameters layer must be repeated multiple times to have enough expressitivity. This means that unfortunately non-linearity is not present on these circuits, unless measurement or specific gate are introduced as it will be seen later.

2.1.4 Variational algorithm training

As previously stated this type of cicruit is parametrized and the parameters are optimized by calculating a cost function and the gradient. Due to the fact that cost fucntion is dependendant in first place from output function and afterward to parameters, the chain rule needs to be applied obtaining the following equation:

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

This form has two components $\frac{\partial C(\theta)}{\partial f(\theta)}$ and $\frac{\partial f(\theta)}{\partial \mu}$ required to exactly calculate the differentiation. First one can be obtained using classical computation, while the second one cannot be evaluated using classical quantities because it depend 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 library 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 μ. 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 is able to estimate the analytic gradient, while the finite difference method focus on approximating it**. Furthermore the parameter-shift rule is not dependant on how big or small must be the variation in order to correctly calculate the gradient and requires very few evaluations. There is a problem with the gradients of variational circuits, that is a significant presence of **barren plateaus**. This term refers to the fact that sometimes the cost function can presents zone where the gradient is highly probable to be zero leading to small variantons on gradient. Mathematically speaking this means that:

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

This is a problem for the correct optimization of parameters due to the fact that gradient is particularly small leading to a non global minimal solution or

requiring a great amount of training time 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. Difference between classical and quantum model is the fact that these regions are ***exponentially more large*** on the quantum case as demonstrated by [16]. The paper furthermore explains that increasing the number of gates, referred even as layers due to repetition of specific gates, and qubits leads to an exponential decay on the variance. This is an ulterior motive for which the variational circuit that is defined must be the most shallow in order to avoid problems on trainability. There has been some strategy addressed to partially solve this problem such as initialization using correlated parametrized circuit as suggested by [17], but this represent one of the major challenge to solve in order to use deepest circuits. A deeper and more detailed explaination can be found in [18].

2.1.5 Variational algorithm as neural networks

Variational circuits are sometimes called "quantum neural networks". This name is partially reasonable due some resemblance to classical neural network such as the optimization of parameters and structure linearity for deep learning. Problem is that these circuits does not express non-linearity unless particularly encoding strategy or measurement is applied. In a interesting wa, non-linear activation function can be obtained by applying different types of encoding with slight modifications as suggested by [18], but the papers to create the ansatz for reinforcement learning does not apply them. It would be intersting to see if these 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 to a layer o f the neural network. 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-

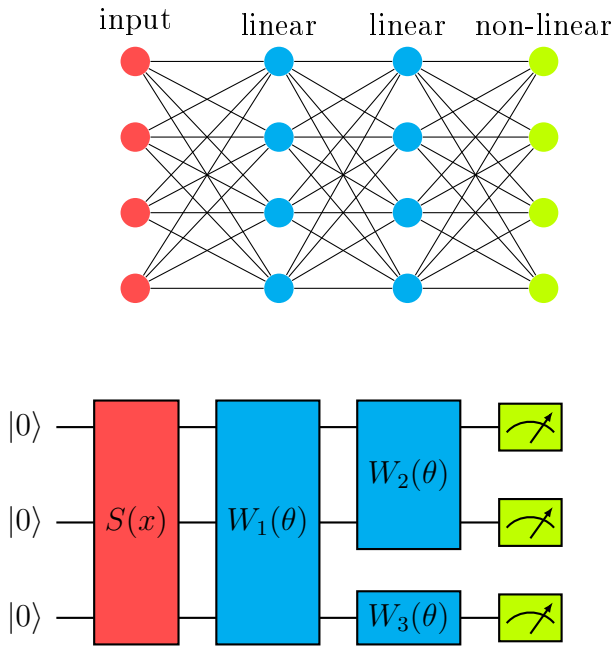


Figure 8: Variational quantum algorithm as a neural network

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 simbol $\mathbb{1}$ refers to an identity matrix 2×2, 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 layer of neural network with linear activation and vqa because both of them use matrix operations.

2.1.6 Data reuploading

As it was already explained before, 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 significant presence of non-linear functions for the q-values and actor component. In order to introduce this non-linearity and deal with the **non-cloning** property of quantum computing a new type of ansatz has been defined: ***data reuploading***. This concept has been introduced in the paper [19] and it wants to introduce non-linearity by reapplying the embedding layer multiple times in order to have a more composite function expressivity. So the strategy is to not use more qubits, but apply multiple layer increasing the depth of final circuit. This

seems to work particularly well for reinforcement learning using simulators. Question is if this kind of depth would be able to run omn a quantum computer and will be a ble 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 qauntum models on the environment called Cartpole-v0. This is part of the library Open-AI gym ([20], link), which contains multiple environments used for benchmarking reinforcement learning algorithms.
The Cartpole environment is composed by a pole attached to a cart:

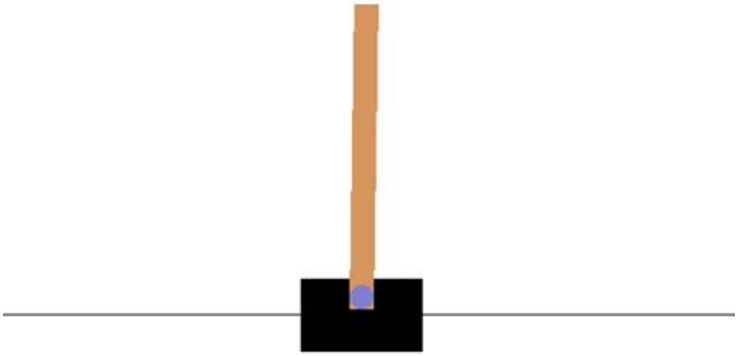


Figure 9: Cartpole environment of Open-AI gym

The environment state is an array of 4 values representing: (position, velocity, pole angle, pole angular velocity). Position and angle are limited in value, while the others are not. The possible action that can be taken is a single value tha can be 0 or 1, which respectivelly means push left or right. The reward given by the environment is that for every step taken the total reward is increased by 1 even on termination. The conditions for which an episode is stopped are: the cartpole reaches extrems of the environment, angle pole

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

2.2.1 Ansatz for VQA

Different ansatz have been used for this work, the starting point was form the paper [21] which has a corresponfing github repository that can be used for the code. Variational circuit proposed for the paper is based from the following layer:

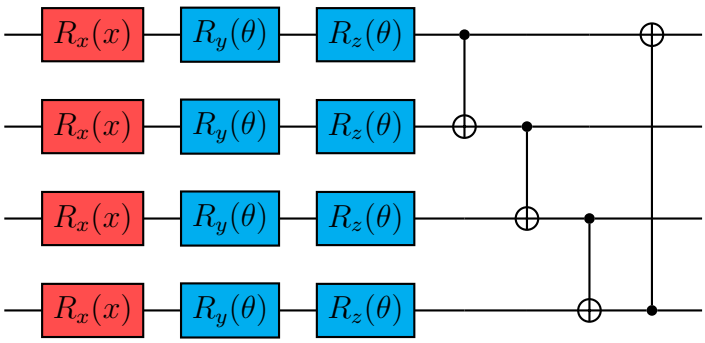


Figure 10: Variational quantum algorithm for cartpole

As it can be seen, this kind of layer applies the data-reuploading approach already mentioned. This seems vital in order to achieve a good performance in respect to the classical one, for more detail please consult the paper. This kind of layer is repeated multiple times in order to achieve enough depth for the expresssivity required to approximate the best q-value function. The measurement applied at the end uses probabilistic approach, in fact due to the nature of cartpole a single action is required, go left or right, so in order to reduce the dimension output a measurement composed on two qubits corresponding to $Z \otimes Z$ is applied. After the measurement value with maximum probability is chosen and executed. In order to increase the performance a classical layer of weights on input and output is added, this leads to the folloring final structure. The original implementation used as libraries pennylane and pytorch, this made the algorithm take around 11 hours for 5000 episodes.

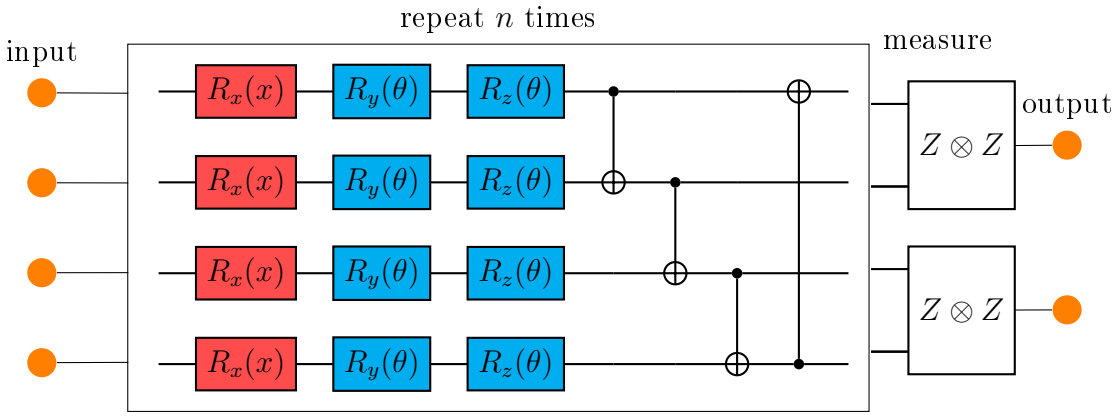


Figure 11: Final structure for cartpole

Due to the requirement of benchmarking the code has been rewritten using tensorflow quantum. This allowed a speedup by reducing the time required ...
By applying multiple runs and confronting the runs made by the quantum algorithm and different kind of neural network this result is obtained:

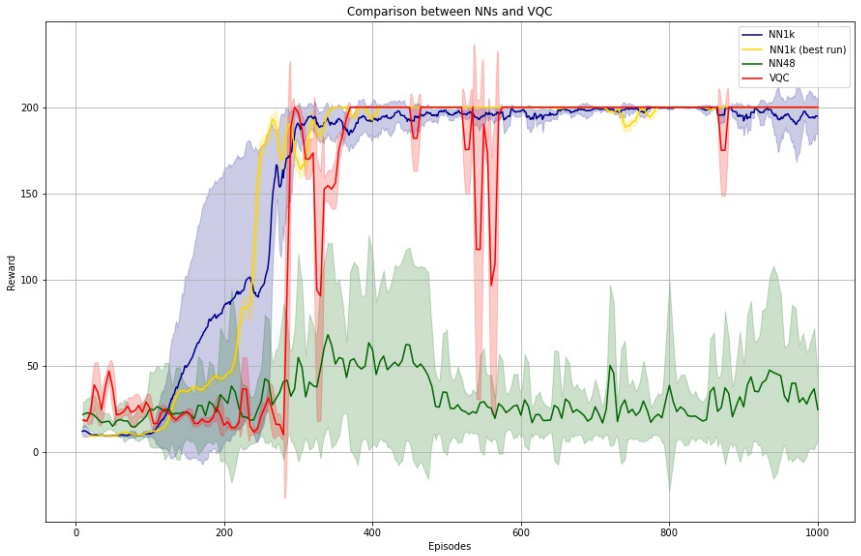


Figure 12: In this image it has been used the mean and the variance of 10 runs of neural networks and vqa

As it can be seen from the figure 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 that a quantum advantage on the number of parameters and expressivity of the modes with few layers. The paper referenced is more expressive and even more detailed on all hyperparameters that has been tested, the configuration used for this benchmark is:

hyperparameter	NN(1256 params)	NN(48 params)	VQA
γ	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
ϵ start	0.1	0.1	1
ϵ decay	0.99	0.99	0.99
ϵ 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 hyperpaprameters ϵ used for the ϵ-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:

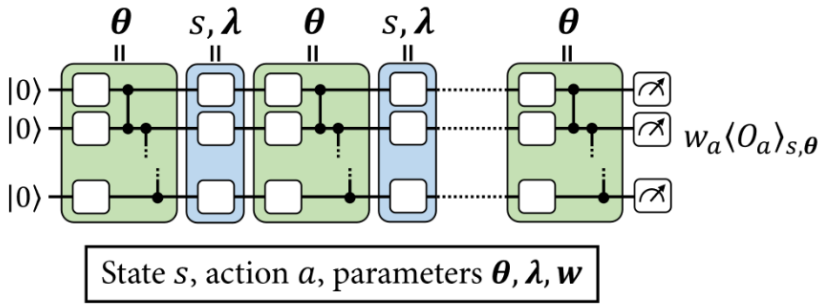


Figure 13: Ansatz of variational circuit with data reuploading of the tensorflow quantum library

The differences between this kind of ansatz and 11 are: the lack of classical weights input-output and most importantly the encoding layer is parametrized. This means that the expressivity of this model can be tuned by finding the parameters that are able to give the best encoding to find afterward best approximation. Now that the results of Cartpole has been showed proving that

there can be a qantum advantage using less parameters, it is now change for another environment which is based on the robotic arm and can be a demonstration on the quantum capabilities applied on possible future industrial application.

2.3 Variational quantum algorithm on robotic arm

2.3.1 Environment

The previous environment is used for benchmarking and to have a partial view on the capability of this algorithm. It is now time to test it on something more difficult and with application on the industry: a robotic arm. This technology is applied in many field 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 from prof.Noah Klarmann from the university of Rosenheim who I would like to thank.This environment is formed of an arm composed of different links whose number can be defined, this influence both the space of state and action, with a randomic point which need to be reached by the arm using his extremis. The links are independent and can move on any direction with a fixed maximum velocity, this 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:

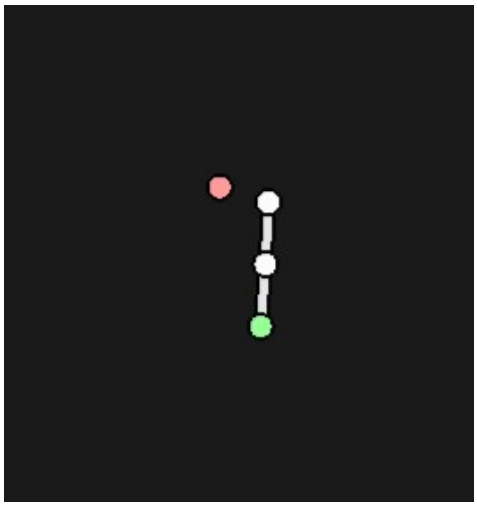


Figure 14: Robotic arm with 2 links, the termination point of arm is green and the objective point is red

The total return is calculated 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 eukledian distance between arm termination and point. In case the

arm is able to effectively reach the point a positive reward of +5 is given and the episode is considered ended. The termination conditions are two: the steps taken is equal to 250 or the point is reached. The state tuple is composed in case of 2 links: (target x, targe y, end effector x, end effector y, link angle 1, link angle 2) in case of more link the tuple increase considerinfg more angles. The action tuple has length equal to links and for the case of 2, it is : (link 1 velocity, link 2 velocity).There isn't an exact condition for which the environment is considered solved, but from multiple runs and afterward rendering the environment was decided to be considered solved when the mean reward was above 40 for the case of 2 links. Differerently from the cartpole enviornment, robotic arm can be multidiscrete or continous and for application it was decided to use the continuos one and use the Soft-Actor Critic (SAC) by substituting a complete neural network approach with an 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 [22].

Algorithm 4 Variational Quantum Sac

Require: initial policy parameters θ, initial action-value estimate parameters ϕ_1 and phi_2, γ, α, ρ, empty experience replay D.
Initialize the hybrid quantum-classical policy network with θ.
Initialzie two action value networks with ϕ_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 ϕ_i by minimizing: $\mathbb{E}_B[(Q_{\phi_i}(S, A) - y(R, S', d))^2]$ for $i = 1, 2$.
Update θ 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

So from this algorithm it is possible to understand that there 5 components necessary: an actor policy, two action-value approximator and others two called target action-value approximator. The reason for which it is necessary the target components is similar to the DQN case, due to the fact that usually two consecutive states are not independent and requiring action-values to be it in order to correctly update the weights an independent copy is required. The reason for which both of action and target components is double can be explained from other advances of DQN which showed that using two components and taking 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 randomically 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 form DQN tries to incorporate Neural Networks layers and Variational Quantum Algorithm. An image showing the actual structure is present on the paper:

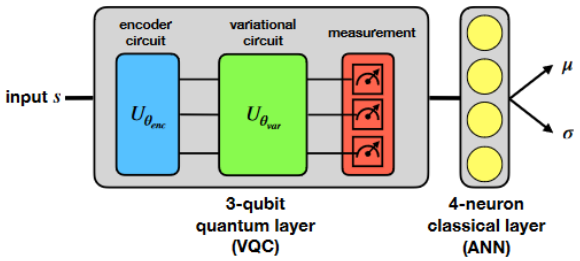


Figure 15: Hybirid actor policy architecture, image taken from [22]

So of the 5 component present on this algorithm only one uses the VQA with NN, this can be a limitation and afterword the results using an algorithm for which every component applies it. All the test showed, are the ones with an environment that uses 2 links, this is due to a lack of time and resources.

2.3.3 Results with hybrid actor

Using the previous ansatz and algorithm it was decide 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 test 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, in order 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 this runs weren't repeated multiple times is the time and resources required, in fact using multiple cpu or applying in the future on a gpu may reduce significantly this time. Unfortunately on the time of writing this is not possible, but it may be applied in the future.
Confronting result obtained using hybrid and classical components gives:

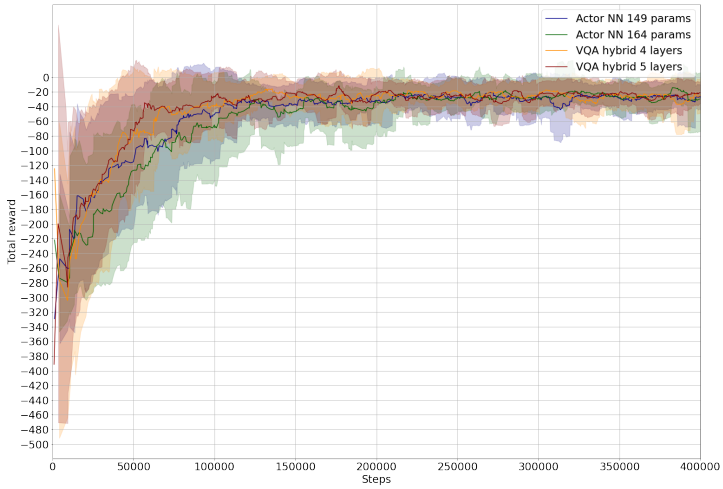


Figure 16: Moving average with time window of 30 with mean and standard deviation of the trend.

From this an interesting fact can be noticed: all of the test have more or less the same trend and reach the same return at around 150000 steps. As can be seen the hybrid algorithm present an higher steep curve confronting to the classical ones and is able to reach the plateau slightly faster, but there isn't a clear qurnum advantage due to the similar number of parameters and time step taken. This is quite suspicious and could mean that the critical component may nit be the acto, but the critic. To confirm this suspect notice the plot of only the actor component that use Neural Networks:

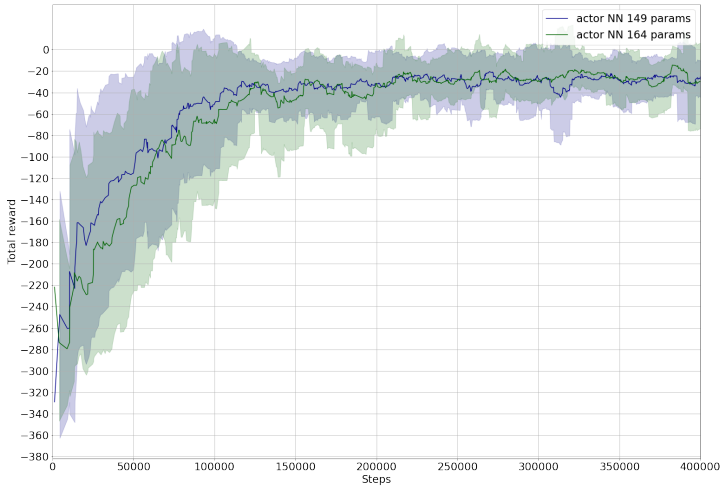


Figure 17: Plot of the runs that uses neural netowrks for all the components.

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 confronting them by resusing the same architecture a plot is obtained:

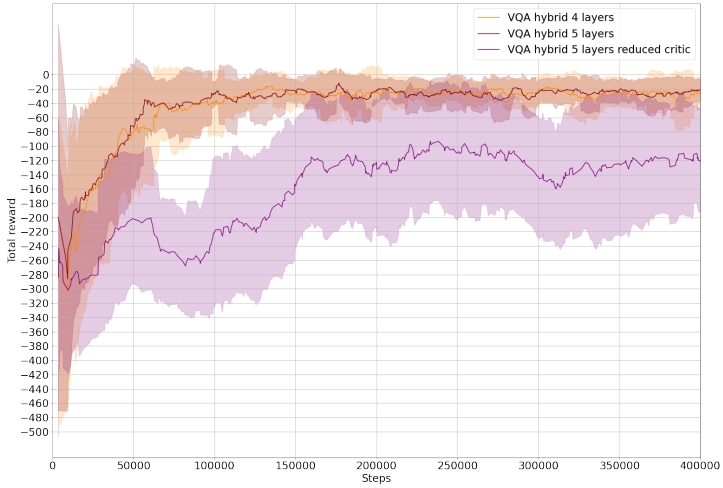


Figure 18: Plot confronting vqa with reduced parameters of critic and previous ones.

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

Hyperparameters	VQA 4 layers	VQA 5 layers
γ	0.99	0.99
α	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	(8,64,64,1)	(8,64,64,1)
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
γ	0.99	0.99
α	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	(8,16,16,1)	(8,16,16,1)
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
γ	0.99	0.99
α	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 toal parameters is so high links to the fact that there are 4 critic components that are required in order for the algorithm to work and learn.

2.3.4 Results with hybrid actor and critic

Now that the previous test have demonstrated that critic is the component that determines the most on performace the following test will use the hybrid algorithm used for the actor and apply it on critic. During the first tests we noticed that the previous structure used for 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 modifided by adding a hidden layer before the output to increase the expressivity of non-linearity. This is the schematic:

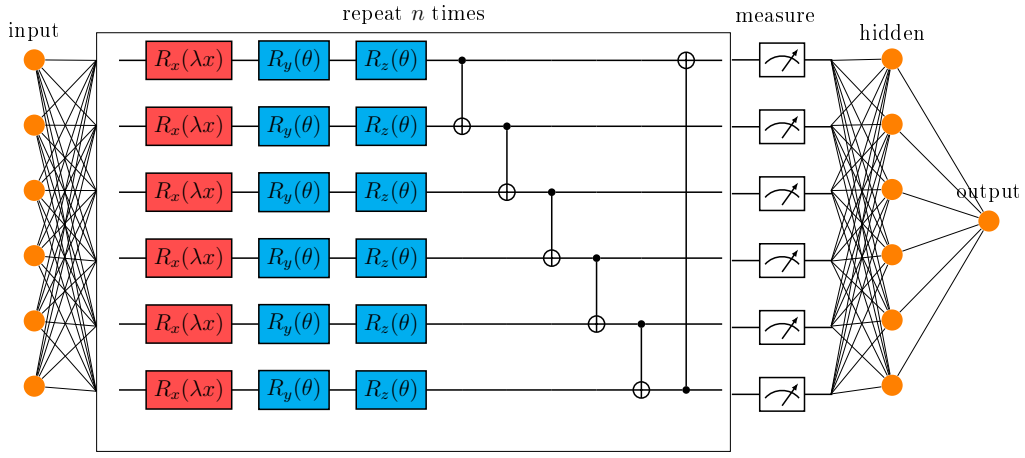


Figure 19: Structure for critic hybrid

As it can be seen from the plot 20 by converting all components to hybrid

architecture it was able to have a performance veery similar to the ones with only hybrid actor. As it will be seen later this means that a smaller component that uses few parameters is able to reach the same performance of one with more on the Neural network. Reason for which this line is interrupted after only 250000 steps is maily 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.

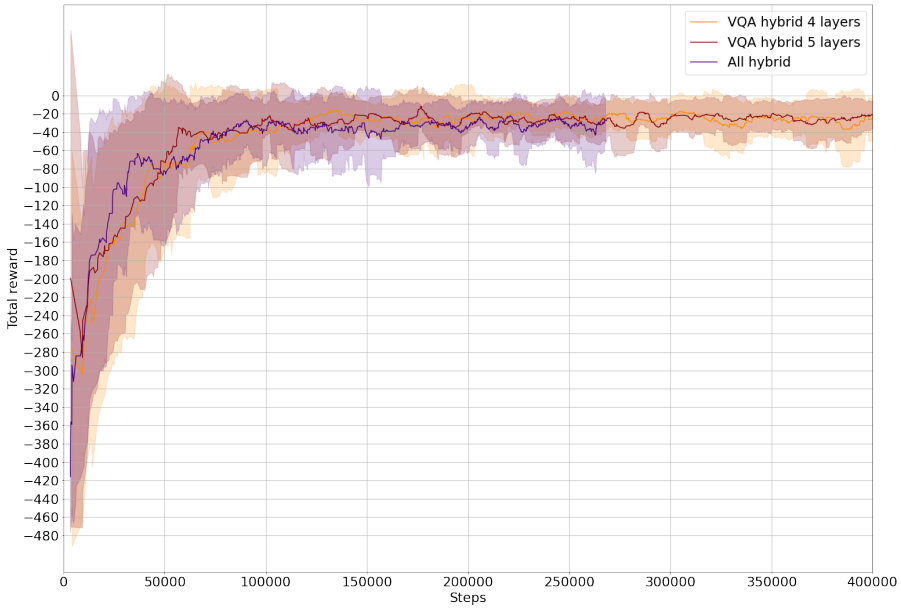


Figure 20: Comparision between only actor hybrid and all components hybrid.

Afterward to have a detailed cased for which multiple configuration with different size of the neural network were conducted in order 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 on 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 on using 21 neurons is not casual, but it has been decided in order to have a similar number of parameters equal to the run that uses all hybrid components.

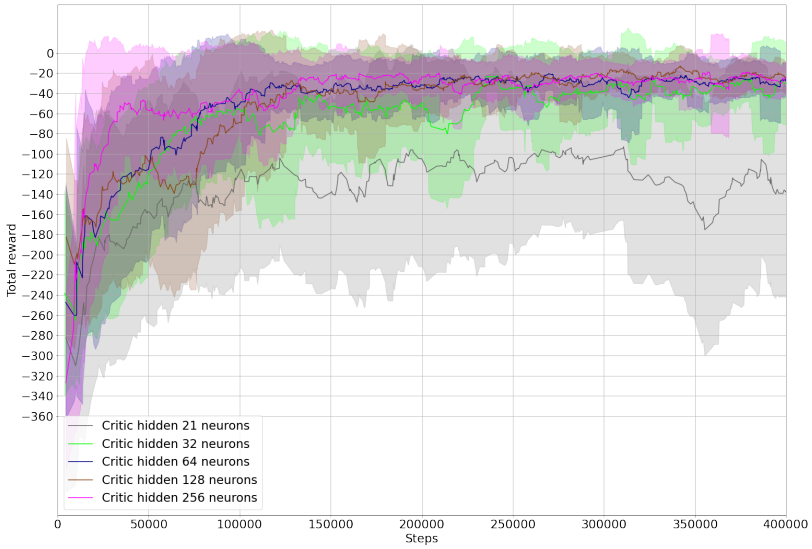


Figure 21: All classical runs with an actor component of 149 parameters

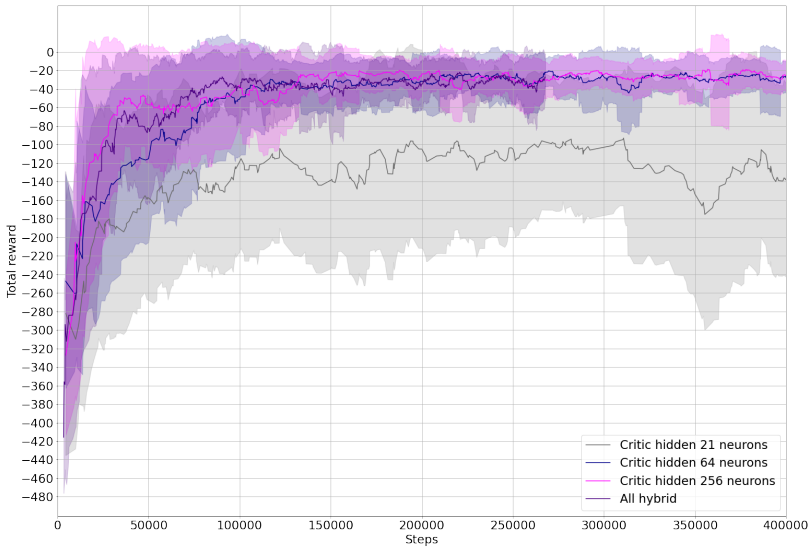


Figure 22: Plot of best classical runs and all-hybrid

So by looking at 22 an important result can be obtained, the all-hybrid run is able to reach a similar performance of all neural networks components that use an 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 all-hybrid model then it will not be able to reach the objective**.

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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).

[11] Maria Schuld and Francesco Petruccione. "Representing Data on a Quantum Computer". In: *Machine Learning with Quantum Computers*. Cham: Springer International Publishing, 2021, pp. 147–176. ISBN: 978-3-030-83098-4. DOI: 10.1007/978-3-030-83098-4_4. URL: https://doi.org/10.1007/978-3-030-83098-4_4.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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