Quantum Reinforcement Learning

Dalila Michelle Islas Sanchez*
School of Physics Astronomy, University of Nottingham, Nottingham, NG7 2RD, UK.
(Dated: September 2, 2022)

We consider the implementation of a quantum actor-critic algorithm, using a hybrid quantum-classical framework, a parametrized quantum circuit, as the analog of a neural network. This paper can be seen as a first approach to quantum reinforcement learning to understand how it works and, based on that, be able to make improvements in order to analyze more complex cases. In addition, analysis of the REINFORCE and deep Q learning algorithms are included, all applied to solve the standard cart-pole environment.

I. INTRODUCTION

Over the years, humanity has pursued ways to solve problems in the most efficient way possible. For this, they have primarily used the intrinsic relationship between computing and physics. An example of this is the Landauer's principle [1], which shows the relationship between thermodynamics and information [2]. In the same way, it gave birth to inventions such as the transistor, microprocessors, and eventually the digital computer [3]. To date, computers have been used to solve and optimize all kinds of processes. Taking this into account, to solve some problems, such as weather prediction [4], it is first necessary to simulate them [5]. However, let us not forget that we live in a quantum world; therefore, for this to be as accurate as possible, doing it on a computer of the same nature would be most appropriate [6]. That is why based on this idea and with the improvements and developments in the field of computer science, for instance GPUs or advanced data compression methods [7], nowadays much is said about quantum computing (QC) and adjacent technologies such as artificial intelligence (AI) or machine learning (ML). It is almost impossible not to notice ML because it is applied to almost anything we can think of, as it is becoming an active and essential part of our daily lives [8], used in many sectors including engineering, medicine, and science.

The use of ML today has made it possible to perform complicated tasks that were almost impossible or time-consuming to achieve previously (for instance, the use of ML and AI as a tool for rapid detection and management of COVID-19 [9]) autonomously since computers learn by themselves. To achieve this goal, there are different methods, classified according to their way of approaching each task: supervised, unsupervised, and reinforcement learning (RL). The first two are based on a training process using labeled or unlabeled information with relevant samples to solve classification or regression problems [10]. In contrast, in reinforcement learning, one of the key ideas is decision-making to achieve a goal. Each decision taken has an immediate consequence that, at the same time, influences actions in the future [11]. This formalism is known as Markov decision processes (MDP), which is shown in Fig. 1.

One way to see it applied is when trying to learn chess, a complex game in which possible moves or actions and the opponent's responses (which lead to a different state in the game and will guide the next moves) have to be taken into account. Despite not being obvious, each action is reflected in the future since the possible ranges of moves that will be taken later depend on it. It seeks to take actions that allow obtaining the most valuable pieces since this maximizes the possibility of winning the game in the long term [11].

On the other hand, quantum computing [12] is also gaining popularity lately. Although the beginnings of this theory date back to before the 80s [6], recently, there has been a significant increase in development. Proof of this is that the development of quantum computing during the last decade has been increasing, thanks to companies such as Google with Sycamore, a quantum processor with an array of 53 qubits [13], IBM with Eagle processor and Qiskit [14], and Intel with the first silicon qubits [15]. This development is in part due to commercial investment, but significant research is being done in academia at the same time. Examples of this are room-temperature studies for quantum circuits [16] and compiler structure for 3-qubit gates [17]. These companies, responsible for some of the most recent

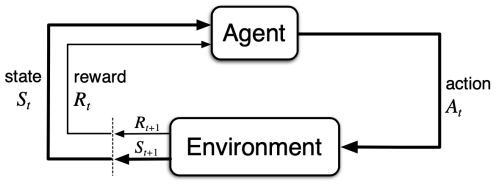


FIG. 1: The agent-environment Markov decision process to find the best actions to complete a goal, where an agent chooses an action that updates the environment, which in turn, feeds back the new state and reward. The diagram is taken from [11].

* ppxdi1@nottingham.ac.uk

advances, have provided us with tools such as IBM quantum [18], a cloud-based quantum processor to discover and experiment in order to promote scientific advances to overcome the challenges that still remain to be solved.

We are entering the era of Noisy Intermediate-Scale Quantum devices with more than 50 qubits capable of performing difficult calculations, even for today's supercomputers. However, these are not yet capable of error correction, which entails weighty restrictions regarding the number of qubits and size of the circuits [19]. Despite these advances, the reality is that there is still a long way to go in order to have a stable and efficient physical infrastructure. For instance, in the continuous development of quantum error correction [20], as quantum computing is probabilistic, the operation has to be repeated many times to obtain an accurate result. Since quantum operations are noisy, when these errors accumulate a calculation failure occurs, limiting its power [21]. That is why it is crucial to study and contribute to the state-of-the-art of these still emerging and robust technologies with the aim of taking advantage of their benefits.

It has been shown that in certain settings, quantum computers can provide polynomial [22] and exponential [23] speed up compared with classical computers. It is natural to ask, can these advantages be used in machine learning? Due to the versatility of RL and the power of quantum computing, it may be possible to tackle problems that are impossible to solve with conventional classical techniques or drastically improve performance. In this paper, we tackle the classic cart-pole RL problem using quantum approaches to RL.

In this paper, we considered three of the most significant and widely used algorithms of RL, to subsequently contrast the results obtained from each one: REINFORCE [24], deep Q learning [25] and actor-critic [26]; in a classical and quantum benchmark, replacing neural networks with quantum circuits in the latter. Quantum circuits are simulated in a classical computer using TensorFlow quantum (TFQ) [27]. An important part of this problem is the cart-pole environment or system, but thanks to OpenAI gym [28], an open source tool that offers a variety of environments in order to facilitate the study and development of artificial intelligence, it can be easily implemented. According to some of the results of this paper, one interesting point is that when comparing the quantum actor-critic algorithm with its classical analog, it did not show a significant advantage, indicating that a simpler function approximator works better for this case.

This paper is structured as follows: In section II we will talk about the necessary elements to design a parameterized quantum circuit (PQC). In Sect. III, the basics of reinforcement learning and the types of methods or algorithms used in this paper will also be described. An analysis of the results and performance will then be carried out in section IV. To finish, in Sects. V and VI we will discuss the difficulties and improvements that this project have.

II. SETUP: QUANTUM AGENT

In this section, we present the necessary elements of quantum circuits to introduce PQC that will replace the neural network in our quantum RL.

Quantum computing consists of computation based on the laws of quantum mechanics. It is interesting to mention that QC is also possible in some way thanks to the contributions of Alan Turing, his Turing Machine, and its efficiency in simulating algorithms [29]. What made later physicists like Paul Benioff suggest that it was possible to build a quantum computing model represented by Turing machines [30]. In the same way, mathematicians and physicists like Yuri Manin [31], and Richard Feynman [6] had already been speculating about the strong connection between physics and computation, creating the concept of having computers based on quantum mechanics. Moreover, David Deutsch, with his article [32] where it states that any physical process can be simulated in a universal device using the laws of physics. A device capable of not only performing any process that a classical computer can do but also performing quantum mechanical processes [33].

All the research and knowledge mentioned above has served the purpose of suggesting and exploring the use of quantum mechanics as a new paradigm of computation [34]. And thanks to those contributions that were the basis of both experimental and theoretical discoveries, such as one of the most famous quantum algorithms, Peter Shor's integer factorization [23], which has exponentially better scaling than the best classical algorithm.

A. Qubit

The bit is the basic unit of information in classical computing which can take two states. Like the classical bit, which can be either in an abstract state 0 or 1, the qubit also has two states in which it can be, but these behave differently from their analog classical. First there are the quantum states $|0\rangle$ and $|1\rangle$, which from what is commonly called the computational basis. Its representation in the Dirac bra-ket notation is as follows:

$$|0\rangle = \begin{bmatrix}1\\0\end{bmatrix}, |1\rangle = \begin{bmatrix}0\\1\end{bmatrix}. \quad (1)$$

The qubit can be in one of these two states, but it can also be in a superposition of these, i.e., it is a linear combination of $|0\rangle$ and $|1\rangle$, which are two orthonormal vectors. It can therefore be represented as a vector in a 2D complex space. The general way to define the state of a qubit would be:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad (2)$$

where α and β are complex numbers and the amplitudes of each state, whose squares represent the probability that the qubit is in each state [12]. These amplitudes have the constraint,

$$|\alpha|^2 + |\beta|^2 = 1. \quad (3)$$

This expression can be viewed as a normalization condition, which is simply that probabilities add to 1.

In the same way that in classical computing, one bit is not enough to perform complex operations, in quantum computing, more than one qubit is also needed to perform complex calculations. When introducing more qubits, the state can now be described as,

$$|\psi_{i_0,i_1,...,i_n}\rangle = \sum_{i_0,i_1,...,i_n=0}^{1} \psi_{i_0,i_1,...,i_n} |i_0, i_1, ..., i_n\rangle, \quad (4)$$

with $\sum_{i_0,i_1,...,i_n} |\psi_{i_0,i_1,...,i_n}|^2 = 1$, and where $|i_0, i_1, ..., i_n\rangle = |i_0\rangle \otimes |i_1\rangle \otimes \cdots \otimes |i_n\rangle$ is the tensor product. The number of parameters needed to describe a system of n qubits will be 2^n, whereas, in a classical system, it only needs n to specify the state of n bits. This supposes exponentially more parameters to specify the state than the classical case and, therein the potential advantage of quantum computing [35].

B. Quantum logic gates

As mentioned in section II A, introducing more qubits and manipulating them in a certain way allows quantum computing to be built in the same way as classical computing: based on circuits, using wires (qubits) that transmit information and quantum gates that act on it to represent calculations [12]. That is why there is the equivalent to classical logic gates (e.g. NOT, AND, OR) and they are quantum logic gates. These logic gates receive a set of qubits as input, and apply some kind of unitary transformation to them.

These quantum gates on a single qubit can be represented in a matrix form, by *unitary* [36] 2×2 matrices. One of the most important are the analogous to the classical NOT logic gate, which in QC would be the inversion gate, also known as the X gate, or the Pauli X gate [12],

$$X \equiv \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}. \quad (5)$$

The action of this logic gate is that when applied to state $|0\rangle$, state $|1\rangle$ is obtained, and when applied to state $|1\rangle$, state $|0\rangle$ is obtained, i.e., it flips the states $|0\rangle \leftrightarrow |1\rangle$, as shown in Eq (6).

$$\begin{aligned} X|0\rangle &= \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0\\1\end{bmatrix} = |1\rangle, \\ X|1\rangle &= \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}1\\0\end{bmatrix} = |0\rangle. \end{aligned} \quad (6)$$

In the same way, other essential gates are the Pauli Y gate and Pauli Z gate:

$$Y \equiv \begin{bmatrix}0 & -i\\i & 0\end{bmatrix}; Z \equiv \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}. \quad (7)$$

The states $|0\rangle$ and $|1\rangle$ are eigenstates of Z, hence commonly referred to as Z-basis.

Similarly the Hadamard gate is very useful, and is represented as follows:

$$H \equiv \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}. \quad (8)$$

This gate is used to create superposition states, that is, to pass them from the state $|0\rangle$, to a state of equal probability for $|0\rangle$ and $|1\rangle$ [12], i.e.,

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle. \quad (9)$$

Pauli gates are helpful to define other important ones, which are the following rotation gates [12]:

$$\begin{aligned} R_X(\theta) &= exp\{-i\frac{\theta}{2}X\} = \cos\frac{\theta}{2}\mathbb{1} - i\sin\frac{\theta}{2}X, \\ R_Y(\theta) &= exp\{-i\frac{\theta}{2}Y\} = \cos\frac{\theta}{2}\mathbb{1} - i\sin\frac{\theta}{2}Y, \\ R_Z(\theta) &= exp\{-i\frac{\theta}{2}Z\} = \cos\frac{\theta}{2}\mathbb{1} - i\sin\frac{\theta}{2}Z. \end{aligned} \quad (10)$$

This gates are important because any single qubit gate can be constructed using combinations of these rotations [12].

Likewise, gates can be constructed to apply to a system of qubits instead of an individual qubit, and can generate quantum entanglement between them [35]. One of the most commonly used and experimentally implemented is the Controlled-NOT (CNOT). This gate acts on two qubits, the *control qubit* and the *target qubit* [12]. In case the *control qubit* is set to 1, the state of the *target qubit* is flipped, otherwise, nothing is done, i.e. $|00\rangle \rightarrow |00\rangle$; $|01\rangle \rightarrow |01\rangle$; $|10\rangle \rightarrow |11\rangle$; $|11\rangle \rightarrow |10\rangle$. Its matrix representation is as follows,

$$CNOT \equiv \quad \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \qquad (11)$$

The CNOT gate cannot be implemented using only single qubit gates. This is an entangling gate that can generate quantum entanglement between qubits. Entanglement can be viewed as a computational resource not available in classical computing [35].

Another two-qubit operation is the controlled-Z (CZ) gate, which works similarly to the CNOT gate if the *control qubit* is in state $|1\rangle$, but instead applying the Z gate [37]. Its matrix representation is as follows:

$$CZ \equiv \quad \equiv \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \qquad (12)$$

Although in this paper only the most basic and essential gates for our project were explained, there is still a wide variety of interesting gates.

C. Quantum circuits

Quantum circuits, like the classical ones, are used to define operations and algorithms, but in this case they will work on qubits. Here we introduce the schematic representation of the quantum circuit.

In QC, a logic gate is graphically represented by its symbol inside a square box, and black lines can be seen as quantum wires and represent a single qubit. The time goes from left to right, and the different operators of the circuit are computed sequentially. The measurements are left at the end of the circuit to obtain information about the final state of the qubit projecting at the computational basis [33], as shown in the circuit in Fig. 2.

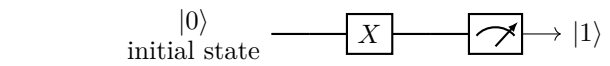


FIG. 2: Quantum circuit with an initial state $|0\rangle$ and then an application of a X gate, to finally perform the measurement, obtaining as a result of $|1\rangle$.

D. Variational quantum circuits

Nowadays, hybrid methods use classical computers to simplify the algorithms since they limit the use of quantum computers to specific tasks in which its use may provide an advantage. This makes them perfect candidates for the first real applications of quantum computers in the near future [38]. One of the most promising hybrid methods is the use of parameterized quantum circuits or variational quantum circuits (PQCs or VQCs), which are formed by fixed gates (for instance, an X gate) as well as gates whose action depends on an adjustable variable (such as a quantum rotation of a given angle), i.e. single-qubit rotations [39].

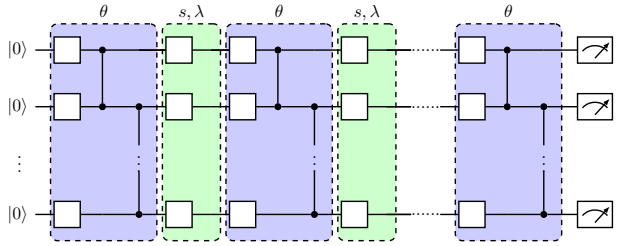


FIG. 3: Parametrized quantum circuit with data re-uploading. The blank squared boxes are general parametrized single qubit gates, that can have learnable parameters θ, a CZ entangling gates. As well as encoding gates with the following data: a s state, and λ learnable scaling weights. This PQC has several data re-uploading, this can be done by duplicating the encoding gates along with the variational and encoding angles as many times as necessary.

Variational circuits are mathematically equivalent to unitary operators in a space whose dimension grows exponentially with the number of qubits [40]. In the field of quantum computing, variational circuits stand out, where parameterized circuits are optimized in order to minimize a given cost function [41].

Hence for the realization of this project, we worked with PQC, that can be viewed as a quantum analogue of a neural network, that are used in a classical case. For this specific case, the input of the circuit is the states of the quantum agent s, which takes values corresponding to the position and velocity of the cart, as well as the angle and the angular velocity of the pole. As an output it has the expectation values that will be used for instance to approximate the Q values, or will be used

for both an actor (policy), and another one for a critic (value-function approximator) component [42], whose details will be discussed later.

To have a better function approximation, it is necessary to implement several re-uploadings [43], alternated with variational gates, as shown in Fig. 3. This model is hybrid because the quantum agent is repeatedly interacting with a classical environment exchanging states s, actions a and rewards r as in the Fig. 1, and its trainable parameters are optimised iteratively using gradient descent on a classical computer see e.g. [24, 44].

Depending on the state, the agent will act according to the policy or Q values, receiving a reward as feedback for each action performed [45]. The PQC has different parameters, such as θ, λ, and w, being the policy parameters, encoding gate input-scaling parameters, and trainable observable weights respectively. At the measurement, the expectation value of the observables (weighted Hermitian operator O_a) are obtained, $w_a\langle O_a\rangle_{s,\theta}$, this operator is used to extract numerical information from the gates as output e.g. the probability of taking one of two actions [46].

More concretely we considered a 4-qubit circuit, as shown in Fig. 4 as is the specific case used in this paper. This circuit is a Controlled PQC, and it is composed of R_X, R_Y and R_Z rotation gates (parameterized by different θ, Eq. (10)), as well as CZ entangling gates (Eq. (12)) to entangle the qubits, that is, alternating layers of encoding unitaries, with variational and encoding angles.

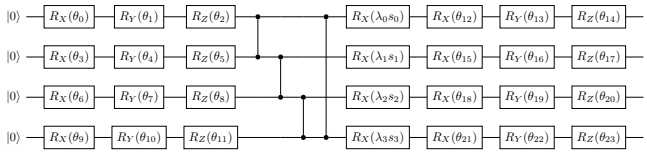


FIG. 4: Parameterized quantum circuit with data re-uploading for our specific case. The shown circuit consists of 4-qubits; a single layer is shown as an example. It has a parameterized rotation in X, Y, and Z axes, with a learnable parameter θ, and CZ entangling gates. The encoding gates have the λ scaling parameter and s of an X rotation.

The number of qubits corresponding to the 4-dimension input is equal to the dimension of the state of environment vector. The number of layers is variable between 1-8, although in Fig. 4, a single layer is shown as an example. These layers are mathematical operations different from those of a neural network, but they are called layers for practical purposes [46]. The number of actions is two, the space of actions of the environment. As its observables $O_a = \sum_i w_{a,i} H_{a,i}$, where it has the Hermitian operator $H_{a,i}$ which are the tensor products of the Pauli Z matrices, since it shares the computational basis, i.e., it can be decomposed in terms of the Z-basis [24]. Specifically we use the $Z_0Z_1Z_2Z_3$ Pauli product for REINFORCE, and Z_0Z_1 for action 0 and Z_2Z_3 for action 1 for deep Q learning [47]. For the actor-critic Z_0Z_1, Z_2z_3 are used for actions of the actor policy, while $Z_0Z_1Z_2Z_3$ is used for the critic value function. From these observables we can compute the output of for example a policy for REINFORCE or actor, depending on the algorithm used.

III. METHODOLOGY

In this section, the different methods used in this project will be reviewed, analyzing their utilities, capabilities and limitations. As well as a deepening of some concepts addressed in the section I.

The environment used in this project is the Cart-Pole-v1, an OpenAI Gym environment based on Michie and Chambers's pole-balancing problem, an important benchmark in the field of reinforcement learning [48]. It comprises of a pole attached and balanced upright to a movable cart. To solve this classical environment, the aim is to prevent it from falling, applying the appropriate forces, 0 or 1, actions which correspond to the cart accelerating left or right, a reward of +1 will be given at each step if the pole stays upright, with a maximum reward of 500. Within the environment, the state is defined by four values: cart position (-4.8, 4.8), cart velocity $(-\infty, \infty)$, pole angle (-24°, 24°), and angular velocity $(-\infty, \infty)$ [28]. It has discrete actions with continuous states. Every time the cart position or pole angle are out of bounds, it will then restore to its upright and initial position. Its graphical representation can be seen in Fig. 5.

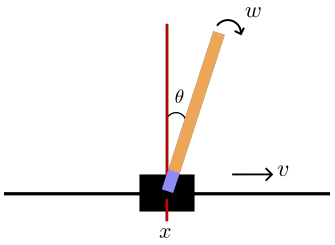


FIG. 5: The car-pole system is a post hinged to a car that moves along a track. It has four states: trainer position x, with values between ±4.8, angular velocity w with values $\pm\infty$, angle between the vertical axis and the pole θ (±24°), and w, the angular velocity of the pole with values between $\pm\infty$.

Unlike other ML methods, in RL, the agent is goal-directed and must explore the environment to achieve it. The RL formalism is mainly based on Markov Decision Processes (MDP) [11]. One way to represent

this process is through the diagram in Fig. 1, where the agent takes actions and sends them to the environment, while the environment sends back information to the agent in the form of states, and rewards. Typically an objective is achieved by maximization of rewards or *expected return* in a certain number of steps in the environment, $R_{t+1}, R_{t+2}, R_{t+3}, ..., R_T$, specifically, the sequence of rewards obtained until the agent reaches a terminal state of the environment, these are called episodes [11]. For tasks that require many steps to achieve the goal, the sum of rewards in the episode can be represented as $R_{t+1}, \gamma R_{t+2}, \gamma^2 R_{t+3}, ..., \gamma^k R_{t+k+1}$, this is called a *discounted return*, where γ is a *discount rate* that goes from $0 \leq \gamma \leq 1$, and determine the present value of future rewards [11]. Therefore RL is handy for implementing agents that act independently in a given environment, e.g., application in robotics and videogames.

The agent chooses its actions according to its strategy, which it will update depending on the feedback obtained, i.e., preferring it if it receives a positive reward or changing it if negative. This strategy can be seen as a table of probabilities or predicted rewards. Depending on the agent's state, specific actions may have a higher probability of achieving positive feedback. The agent is based on actions that have been shown to be effective, but to find them, the agent must try actions that have not been taken before, that is, there must be a trade-off between exploration and exploitation [11]. The major problem with this type of algorithms is the refinement of that table, since the agent learns based on trial-error during the training phase. In cases where the environment is large, i.e., with large and continuous sets of states, this phase can last for a long time, and all possibilities cannot be stored in memory. That is why alternatives are sought that minimise training time; some options are using neural networks or quantum computing, taking advantage of the state superposition to evaluate all actions simultaneously, thus reducing training time.

In the quantum case, like deep reinforcement learning, we use a PQC as a function approximator instead of a table, like is doing in the tabular methods. The difference in the classical case is that the function approximator is a neural network, while in the quantum case this function approximator is a PQC.

RL algorithms can generally be classified into three different criteria: value-based, policy-based, or model-based. In the *value-based* approach, the value function is computed for each state and used to assess the policy. Whereas for a *policy-based* approach, the policy is iteratively evaluated and improved [49]. Both methods know the environment through simulation interactions; that is, they do not know the model of the environment, and therefore they are called *model-free*, e.g. Temporal difference, Sarsa, Q-learning, and Monte Carlo methods. This property is an advantage over more complex *model-based* methods, e.g. dynamic programming, which need a comprehensive knowledge of the dynamics of the environment by the agent. This is known as the transition function, which allows moving from one state to another by planning and anticipating future states and rewards [11]. On the other hand, the two approaches mentioned above can be combined in a method called actor-critic, where an actor or policy and a critic or value-function approximator are introduced, which what it does is calculate the value function, while the actor uses those values to update the policy [49].

A. REINFORCE

The first algorithm we consider is REINFORCE or Monte-Carlo policy gradient, which is a policy-based RL algorithm. The quantum version was studied previously in ref [24]. The idea of some algorithms of this family (policy gradient methods) is to learn the policy or *parametrized policy* with respect to θ, i.e., $\pi_\theta(a|s)$, which can choose actions without requiring a value function [11].

The REINFORCE algorithm then, consists of directly applying the idea that PQC approximates the policy $\pi_\theta(a|s)$. In this case, a SOFTMAX-PQC policy is used, which employs a non linear activation function softmax_β to the expectation values that can be between -1 and 1, to normalize them. It is defined as follows:

$$\pi_\theta(a|s) = \frac{e^{\beta w \langle O_a \rangle_{s,\theta}}}{\sum_{a'} e^{\beta w \langle O_{a'} \rangle_{s,\theta}}}, \tag{13}$$

where $\langle O_a \rangle_{s,\theta}$ is the expectation value of the observables per action, $\langle O_{a=0} \rangle = \langle Z_0 Z_1 Z_2 Z_3 \rangle$, and $\langle O_{a=1} \rangle = \langle O_{a=0} \rangle$, and because we have the softmax, this give two values that add up to one. Here θ are the rotation angles $\theta \in [0, 2\pi]^{|\theta|}$, where the dimension of this parameter is defined in the norm $|\theta|$, $\langle O_a \rangle$ are the observables per action, and $\lambda \in \mathbb{R}^{|\lambda|}$ are the scaling parameters, both defined in section II C, $\beta \in \mathbb{R}$ is the inverse temperature parameter, this controls how close to a step function [24] So we have the parameters θ, λ, and w. In other words, from the observables, we calculate the output of this agent's policy that is the probability of taking any of the two available actions. This is made in an additional post-processing softmax layer.

The loss function for this type of algorithms is:

$$\mathcal{L}(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{s_0,a_0,r_1,s_1,a_1,\ldots\in\mathcal{B}} \left(\sum_{t=0}^{H-1} \log(\pi_\theta(a_t|s_t)) \sum_{t'=1}^{H-t} \gamma^{t'} r_{t+t'}\right). \tag{14}$$

In this case it is learning the policy that maximises $\mathcal{L}(\theta)$, defined in the Eq. (14). The agent will use it to apply backpropagation to perform a gradient ascending algorithm to optimize it updating the policy-model, using several re-uploadings, similar to [50].

This loss function is defined b y $\mathcal{B}$ batches of episodes $(s_0, a_0, r_1, s_1, a_1, \ldots)$, and where $\sum_{t'=1}^{H-t} \gamma^{t'} r_{t+t'}$ are the discounted returns $G_{i,t}$ with a discount rate γ. The hyperparameters used for this agent are a discount rate of $\gamma = 1$. We use a separate optimizer for the different learning parameters $\alpha_1 = 0.1$, $\alpha_2 = 0.01$, and $\alpha_3 = 0.1$. We also use a batch size of $\mathcal{B} = 10$, and an inverse temperature parameter that we fix it to β =1. The pseudocode of the algorithm can be seen in Algorithm 1.

Algorithm 1: REINFORCE with PQC policies, using batch of episodes, in our case N=2000, $\beta = 1$, $\gamma = 1$, and $\alpha = 0.1, 0.01$, where M = $\frac{N}{\mathcal{B}}$, and $\mathcal{B} = 10$ is the number of episodes per batch. Based on [24].

Input : A PQC policy π_θ
Initialise: Parameters θ and ω
for *t = 1,..., M* **do**
  Generate N episodes $\{(s_0, a_0, r_1, ..., s_{H-1}, a_{H-1}, r_H)\}_i$ following π_θ
  for *i = 1,..., N* **do**
    Compute the returns $G_{i,t} \leftarrow \sum_{t'=1}^{H-t} \gamma^{t'} r^i_{t+t'}$
    Compute the gradients $\bigtriangledown_\theta \log \pi_\theta(a^i_t|s^i_t)$ using a SOFTMAX-PQC policy
  end
  Compute $\triangle\theta = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H-1} \bigtriangledown_\theta \log \pi_\theta(a^i_t|s^i_t)(G_{i,t})$
  Update $\theta \leftarrow \theta + \alpha \triangle \theta$
  Compute average rewards R_{avg}
  if $R_{avg} \geq$ *500* **then**
    break
  end
end

B. Deep Q Learning

Q learning is a model-free and value-based method, i.e., it does not estimate a model of the environment and operates by approximating the action-value function $Q_\pi(s, a)$. In the case of finite actions and states, $Q_\pi(s, a)$ can be seen as a strategy table, like the one mentioned above, where the rows run through the states and the columns the actions [51]. Q learning works satisfactorily when the environment is simple; however, a problem appears when the number of different states and actions is continuous. To solve that, deep Q learning uses a neural network to approximate the action-value function. Deep Q learning uses two neural networks: a main neural network parametrized by θ to estimate $Q(s, a; \theta)$; and a target neural network with θ', to approximate the Q values of the next action and state a', s'. On the other hand, this algorithm is also off-policy; that unlike the on-policy approach that uses the same policy it is inferring to find the optimal one, off-policy algorithms use $Q_*(s, a) = \max_\pi Q_\pi(s, a)$. That is, the actions taken, $a_t = \text{argmax}_a Q(s_t, a)$, differ from the policy to be learned, thus obtaining better results by incorporating, for example, exploration through an ϵ-greedy policy. An experience replay is introduced to store past transitions of the agent.

For the quantum case, a PQC as a function approximator is used, $Q_\theta(s, a) = \langle O_a \rangle_{s,\theta}$, where the main PQC network, $U_\theta(s)$ is parametrized by θ, while the target PQC network is $U_{\theta_\delta}(s)$, that is, the target PQC is copied from the main PQC after certain interval of episodes. Where δ are the specific intervals of episodes after the model parameters are copied, with the aim of stabilizing the Q-value function, in our case this parameter is set to 30. The quantum version was studied previously in ref [24, 40, 46]. Here, $\langle O_a \rangle_{s,\theta}$ is the expectation value of O_a as an output, also weighted in the same way as section III A, but by an action-specific weight, with the difference that the Pauli product for each action is $O_{a=0} = Z_0 Z_1$ and $O_{a=1} = Z_2 Z_3$, these are re-scaled to be between 0 and 1 in an additional layer applying on $(1 + \langle Z_0 Z_1 \rangle_{s,\theta})/2$ and $(1 + \langle Z_2 Z_3 \rangle_{s,\theta})/2$.

In this case the loss function (Eq. 15), which is the mean squared error between the action-value Q and the target Q values, is calculated over a batch of iterations (s, a, r, s') of the replay memory and θ' parameters to update the Q-values performing gradient descent,

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{s,a,r,s'\in\mathcal{B}} \left(Q_\theta(s, a) - [r + \gamma \max_{a'} Q_{\theta'}(s', a')]\right)^2. \tag{15}$$

The pseudocode of the algorithm for this method is presented in Algorithm 2.

C. Actor-Critic

The Reinforce algorithm is conceptually straightforward, but the estimator that the use of the gradient has significant variance, making it infeasible for complex and lengthy environments. That is why the idea of baseline function $b(s)$ is introduced, that is a function

Algorithm 2: Deep Q-learning with PQC target and Q-function approximator, with batch of interactions, experience replay, and a ϵ-greedy policy. In our case M = 2000. Based on [40].

Initialise: Replay memory $\mathcal{D}$ to capacity N
PQC action-value function Q with random weights
for *episode = 1,..., M* **do**
  Initialise: state s_1 and encode into the quantum state
  for t = *1,...,* ∞ **do**
    With probability ϵ select random action a_t,
    otherwise select $a_t = \max_a Q_*(s_t, a; \theta)$
    Execute action a_t and observe reward r_t and next state s_{t+1}
    Set $s_{t+1} = s_t, a_t$ and preprocess $\theta_{t+1} = \theta(s_{t+1})$
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Set

$$y_j = \begin{cases} r_j \\ r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta) \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
    if *done* **then**
      break
    end
  end
  Compute average rewards R_{avg}
  if $R_{avg} \geq$ *500* **then**
    break
  end
end

that depends only on the state, and not on the action. So for this case, the parameterized policy $\pi_\theta(s, a)$ which we call the actor, to propose and update an action to a given state. As well as the approximate value function $V_\phi^{\pi_\theta}(s)$ parameterized by θ, which we call the critic [26] to evaluate the proposed action to the given state by the actor, i.e, the expected return according to the specific policy. The actor-critic then combines a policy-based and value-based components to take advantage of each other. The goal is to choose actions based on a policy that maximises the expected return. This type of algorithms are very active research areas of RL [25].
These methods use temporal difference to compute the error δ of the estimate V on a transition (s, a, s', a') from state (s, a) to (s', a'): $\delta = r + \gamma V(s') - V(s)$, with $\gamma \in [0, 1)$.

In this case we use the same PQC for the actor component as in section III A to learn a policy, i.e., to generate the probability distribution of each action with respect to the state, this because the actor is a policy gradient method. For the critic we also use it, however, this is different in the way that it is only necessary one output instead of two. We then modified it to ensure a critic value as a single output by modifying the last layer by making that the only number of available actions is 1. We removed the softmax layer used in REINFORCE. Hence, in this way we can obtain one output, i.e., $\lambda\langle Z_0 Z_1 Z_2 Z_3\rangle$ of the PQC that will provide the probability of two outcomes.

The losses for this algorithm were calculated separately, being the actor loss (Eq. (16)),

$$L_{actor} = -\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)[G(s_t, a_t) - V_\theta^\pi(s_t)], \quad (16)$$

based on policy gradient methods per episode, where, $G = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$ is the returns.

While for the critic loss, Eq. (17):

$$L_{critic} = L_\delta(G - V_\theta^\pi), \quad (17)$$

where $L_\delta(a)$ is the Hubber loss defined by $\frac{1}{2}(G - V_\theta^\pi)^2$ if $|a| \leq \delta$ or $\delta \cdot \left(|a| - \frac{1}{2}\delta\right)$ otherwise [26]. In our case we use a value of 1 for δ.

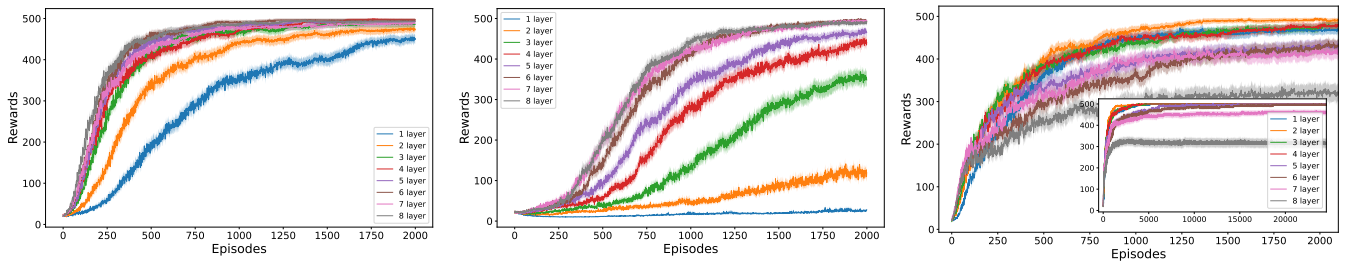
The pseudocode of the algorithm for this method is presented in Algorithm 3.

IV. RESULTS

This section shows the results obtained for the tests on the REINFORCE, deep Q-learning, and actor-critic quantum methods using parametrized quantum circuits and classical methods using neural networks. Due to its random nature, 100 runs were made for each method mentioned above, both quantum and classical making use of tensorflow and tensorflow quantum [27].

All the tests were executed both in Google Colab, in a virtual machine *mlis*1 and *mlis*2 each with 2 2080Ti GPUs, as well as in an on-site High-Performance Computer (HPC) of the University of Nottingham.

First, in Fig. 7 we show an instance of the rewards during the actor-critic method learning. It can be seen that it manages to perform decently within the first 150 episodes approximately but with many ups and downs, but it is not until episode 900 that it manages to stabilize and consistently maintain a good reward, i.e., the maximum reward that is 500 for the Cart-Pole v1. Secondly, we made 100 runs of each because a



(a) Quantum REINFORCE (b) Quantum Deep Q learning (c) Quantum Actor-Critic

FIG. 6: Average rewards vs. episodes over 100 runs, layers from 1-8, with a standard error of the mean confidence interval for quantum REINFORCE on Fig. 6a, deep Q-learning on Fig. 6b, and actor-critic on Fig. 6c. In Fig. 6a, it can be seen that the agent generally becomes better as the number of layers increases, learning faster and converging to the maximum reward. However, with seven layers, it learns faster, but it takes longer to reach the maximum reward, while for eight layers, it improves considerably. On the other hand, in Fig. 6b ff it is clearly seen that a greater number of layers is necessary to have a better performance, possibly due to the complexity of its Q-function. On the contrary, Fig. 6c, which shows an inset with 20,000 episodes, illustrates that increasing the number of layers to eight gives worse results than having a single layer.

Algorithm 3: Actor-critic with a PQC policy, and a PQC value function approximator with a discount factor $\gamma = 0.99$, and with maximum steps per episode of $N = 10000$. Based on [26].

Input : A PQC $\pi_\theta(s, a)$, and $V_\phi^{\pi_\theta}(s)$

while *True* **do**
  Initialise: Initial state s_0, policy parameters θ, and total episodes rewards R
  for t = *1,.., N* **do**
    Take action $a_t \sim \pi_\theta(a_t, s_t)$,
    Observe next state $s_{t+1}(s_t, a_t, s_{t+1})$
    Observe reward r_{t+1}
    Accumulate episode reward in R
    if *done* **then**
      break
    end
  end
  Update running reward
  Update $V_\theta^\pi(s)$
  $A^\pi(s_i, a_i) = r(s_i, a_i) + \gamma V_\theta^\pi(s') - V_\theta^\pi(s)$
  $\nabla_\theta \approx \sum_i \nabla_\theta \log \pi_\theta(a_i|s_i) A^\pi(s_i, a_i)$
  $\theta \leftarrow \theta + \delta \nabla_\theta$
  if *running reward* $\geq$ *350* and R = *500* **then**
    break
  end
end

particular run will be random each time and probably with different behavior; this can be seen in the red plot, which is more clearly how the agent is learning and performing on average. However, it is not possible to notice if it stabilises in the range of 2000 episodes.

On the other hand, the three algorithms are compared in Fig. 6. Where it was made an average of the 100 runs that were performed, filtered by each number of layers, ranging from 1 to 8. All with a maximum number of 2000 episodes, except for the actor-critic method, where up to 80,000 episodes shown in inset of 6c.

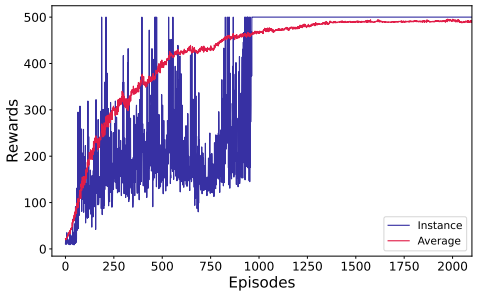


FIG. 7: Here we can see an instance of actor-critic agent performance during 2000 episodes. It can be seen that after inconsistent learning, it manages to reach the maximum reward of 500 from episode 900. On the other hand, we have the agent performance as an average, where we can see the actual behavior of the 100 runs, which generally learn faster than the instance. However, it clearly did not get the maximum reward in that number of episodes. Until episode 1900, it begins to get closer to the maximum reward; nevertheless, it is not possible to appreciate if it manages to stabilise.

Specifically for the quantum actor-critic, we can see that the agent performs better within the first four layers. Starting with a single layer makes the agent learn faster; by adding more layers, it improves up to a certain point, and if we go beyond that, for instance, three layers, it worsens again.

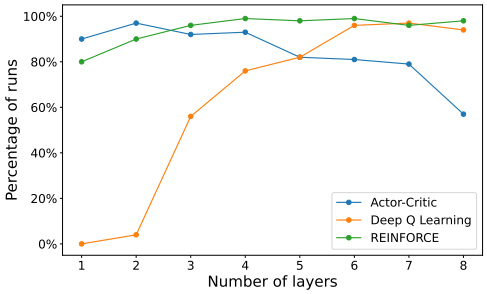


FIG. 8: Percentage of runs achieved the maximum reward of 500 for REINFORCE, deep Q-learning and actor-critic methods. This plot shows that for the case of five layers, both actor-critic and deep Q-learning have the same percentage of runs 82%, which is an improvement for deep Q-learning, but not for the actor-critic agent, which keeps getting worse until reaching its minimum, which is 60%. REINFORCE agent was the only one that maintained the same type of behavior with a different number of layers. In general, it can be seen that deep Q-learning and REINFORCE improved with more number of layers while actor-critic could not after four layers. This behavior agrees with that seen in Fig. 6c.

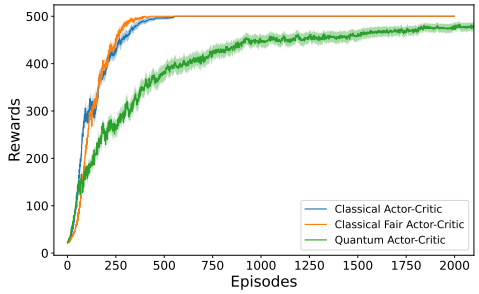


FIG. 9: In this plot, we compare three different agents: a PQC with four layers corresponding to 78 learnable parameters, a classical neural network with 898 learnable parameters, and a classical-fair neural network with 79 learnable parameters. This shows that the particular architecture for the quantum agent is not entirely suitable for this problem because the classical agents performed significantly better with fewer as well as a fair number of parameters.

On the other hand, the deep Q-learning method is almost entirely different. It can be seen clearly that both one layer and two layers did not perform well; that is, more layers are needed to perform better. This could be due to the complexity of its Q-function, which makes the process harder to learn. Each agent except Actor-Critic improved with more number of layers. It is interesting to point out that deep Q-learning needs more parameters to learn, while the REINFORCE and Actor-Critic both have good performance with the difference in how they learn, as is the case that even with one layer can perform decently.

In Fig. 8 we show the percentage of runs that succeded, i.e., how many runs achieved the maximum reward of 500. As we saw in Fig. 6, the number of layers plays an essential role in the agent's actual performance. Both REINFORCE and actor-critic started with similar behavior with 80% or more achieving the maximum reward. For a single layer none of the 100 runs manages to finish or achieved the full reward for deep Q-learning. REINFORCE consistently gets better, whereas the actor-critic with the first layers has a percentage of up to more than 80%, but when the number of layers increases, the percentage that succeeds decreases. This means that having a more expressive function approximator is really important for deep Q-learning. However, for the actor-critic, it is slowing down learning so much that it is no longer successful; therefore, a considerably simpler function approximator is better for the actor-critic.

For Fig 9, we compare PQC with four layers with classical neural network, and with fair classical NN that has comparable number of parameters to PQC for the actor-critic. We choose four layers because it was successful in all cases. However it can be seen a comparison not fair, that is why we choose a classical-fair version with 79 parameters, only one more than the actor-critic. For the classical case, the number of parameters is 898 while for the quantum case, is only 78 parameters, in the figure can be seen that the classical and classical fair perform better than the quantum. These results show that the classical case works better than the quantum version, even with a comparable number of parameters. In fact, the number of parameters does not seem to make a significant difference in the classical case. This may be because the particular PQC we are using is not optimal for this problem, or perhaps there is some scope for designing an improved quantum circuit. It is also possible that something else is going on, causing the learning to slow down that we do not know yet.

V. DISCUSSION

An important point to mention is that trying to make an agent learn correctly is not straightforward; it is necessary to experiment with hyperparameters in order to get a good result. We chose to stick with the hyperparameters provided in the recent research papers, where they realised a considerable grid search to determine the best ones. The only difference is the case of the classical method in which the PQC was replaced with a neural network. There, we did that search ourselves and found that the most difficult of each was deep Q-learning, whose Q function is more complex than the others. It would be interesting to

see if the quantum case of actor-critic can be further improved by optimizing hyperparameters, however, that is not something we investigate in this paper. For this specific case, the quantum agent is not better than the classical version, and the classical agents are definitely performing better. Perhaps we need to address a more complicated problem because the cart-pole is a straightforward environment that is very well suited for the classical algorithms.

The actor-critic, being the main focus of this paper, we chose to represent the actor and critic by separate networks. However, an alternative will be to represent them as a single neural network with a single PQC, previous works have shown that this can actually obtain better results. An overview for future work would be a comparison with a combined actor-critic PQC and see which works better. Another reason of this decision is that we already have a policy-based architecture that makes it easy to change from a two-output configuration to a one-output configuration. If we had chosen the other version, things would be more complicated because, in that case, a three-output architecture would be needed, which implies changes in the quantum circuit and the method. That is why I cannot say that one option is better than the other; simply, the chosen version worked better for our structure.

VI. CONCLUSION

In general, this project aims to understand the concepts and ideas of quantum computing, mainly the implementation of variational quantum circuits, by applying reinforcement learning algorithms, especially the actor-critic method, using the TFQ framework.

In this project, we address the problem of the cart-pole. We focus on the quantum actor-critic algorithm using PQC. From the results, it can be seen that although it has a good performance, it is not better than the other algorithms, both quantum and classical. This is to be expected due the size and nature of this environment, as the benefit of the superposition of states that quantum computing offers is not being used. The advantages will be seen when more qubits or layers are used, and when performing tasks 'impossible' even for a powerful classical computer.

It is, however, an exciting project because, as far as we know, there are not many works related to actor-critic algorithms with a quantum circuit architecture, and of the few that there are, they have taken a different approach than this paper. Also, the fact of delving more into RL topics as well as learning about quantum circuits, which in my perspective, are some of the most complex topics, has been a great challenge for me. However, it is interesting to know that due to the state of development in this area, it is still possible to make new contributions.

This project still has many improvements and different aspects that can continue to be carried out. For example, the environment approached, in this case with an actor-critic algorithm that has a probabilistic policy, that is, we have two possibilities or discrete actions, moving to the left or right, being the output the probability between these two; and also a set of continuous states. Another perspective is to apply to problems with continuous actions such as the inverted pendulum.

ACKNOWLEDGMENTS

I want to thank the University of Nottingham for the access to the computational resources necessary to carry out this project. Specifically to the $mlis1$ and $msli2$ machines, as well as the HPC. I also would like to thank all the support, patience, commitment, and significant contributions of my supervisor Dr. Adam Smith at all stages of this project.

[1] I. Georgescu, 60 years of landauer's principle, Nature Reviews Physics **3**, 770 (2021).
[2] A. S. Wright, The Physics of Forgetting: Thermodynamics of Information at IBM 1959–1982, Perspectives on Science **24**, 112 (2016).
[3] G. O'Regan, *Mathematics in Computing An Accessible Guide to Historical, Foundational and Application Contexts* (Springer, 2020).
[4] P. Bauer, A. Thorpe, and G. Brunet, The quiet revolution of numerical weather prediction, Nature **525**, 47 (2015).
[5] Y. Li, L. Ma, and Y. Shi, Exploration on computer simulation method in physics education, in *Education and Educational Technology* (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 517–522.
[6] R. P. Feynman, Simulating physics with computers, International Journal of Theoretical Physics **21**, 467 (1982).
[7] A. L. Fradkov, Early history of machine learning, IFAC-PapersOnLine **53**, 1385 (2020), 21st IFAC World Congress.
[8] I. H. Sarker, Machine learning: Algorithms, real-world applications and research directions, SN Computer Science **2**, 160 (2021).
[9] Y. Mohamadou, A. Halidou, and P. T. Kapen, A review of mathematical modeling, artificial intelligence and datasets used in the study, prediction and management of covid-19, Applied Intelligence **50**, 3913 (2020).
[10] K. P. Murphy, *Machine learning : a probabilistic perspective* (MIT Press, Cambridge, Mass., 2013).
[11] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction* (MIT press, 2018).
[12] M. A. Nielsen and I. L. Chuang, *Quantum Computation*

and Quantum Information (Cambridge University Press, 2010).

[13] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, Buell, and et al., Quantum supremacy using a programmable superconducting processor, Nature **574**, 505 (2019).

[14] A. Eddins, M. Motta, T. P. Gujarati, S. Bravyi, A. Mezzacapo, C. Hadfield, and S. Sheldon, Doubling the size of quantum simulators by entanglement forging, PRX Quantum **3** (2022).

[15] A. M. J. Zwerver, T. Krähenmann, T. F. Watson, L. Lampert, H. C. George, R. Pillarisetty, S. A. Bojarski, P. Amin, S. V. Amitonov, J. M. Boter, Caudillo, and et al., Qubits made by advanced semiconductor manufacturing, Nature Electronics **5**, 184 (2022).

[16] S. Krastanov, M. Heuck, J. H. Shapiro, P. Narang, D. R. Englund, and K. Jacobs, Room-temperature photonic logical qubits via second-order nonlinearities, Nature Communications **12**, 191 (2021).

[17] C. Duckering, J. M. Baker, A. Litteken, and F. T. Chong, Orchestrated trios: Compiling for efficient communication in quantum programs with 3-qubit gates, in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21 (Association for Computing Machinery, New York, NY, USA, 2021) pp. 375–385.

[18] Ibm quantum, https://quantum-computing.ibm.com/ (2021).

[19] J. Preskill, Quantum computing in the NISQ era and beyond, Quantum **2**, 79 (2018).

[20] Y. Zhao, Y. Ye, H.-L. Huang, Y. Zhang, D. Wu, H. Guan, Q. Zhu, Z. Wei, T. He, S. Cao, F. Chen, T.-H. Chung, H. Deng, D. Fan, M. Gong, C. Guo, S. Guo, L. Han, N. Li, S. Li, Y. Li, F. Liang, J. Lin, H. Qian, H. Rong, H. Su, L. Sun, S. Wang, Y. Wu, Y. Xu, C. Ying, J. Yu, C. Zha, K. Zhang, Y.-H. Huo, C.-Y. Lu, C.-Z. Peng, X. Zhu, and J.-W. Pan, Realization of an error-correcting surface code with superconducting qubits, Phys. Rev. Lett. **129**, 030501 (2022).

[21] J. Preskill, Quantum computing 40 years later (2021).

[22] L. K. Grover, A fast quantum mechanical algorithm for database search (1996).

[23] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM Journal on Computing **26**, 1484 (1997).

[24] S. Jerbi, C. Gyurik, S. C. Marshall, H. J. Briegel, and V. Dunjko, Parametrized quantum policies for reinforcement learning (2021).

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

[26] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, Natural actor–critic algorithms, Automatica **45**, 2471 (2009).

[27] M. Broughton, G. Verdon, T. McCourt, A. J. Martinez, J. H. Yoo, S. V. Isakov, P. Massey, R. Halavati, M. Y. Niu, A. Zlokapa, E. Peters, O. Lockwood, A. Skolik, S. Jerbi, V. Dunjko, M. Leib, M. Streif, D. Von Dollen, H. Chen, S. Cao, R. Wiersema, H.-Y. Huang, J. R. McClean, R. Babbush, S. Boixo, D. Bacon, A. K. Ho, H. Neven, and M. Mohseni, Tensorflow quantum: A software framework for quantum machine learning (2020).

[28] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, Openai gym, arXiv preprint arXiv:1606.01540 (2016).

[29] A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem, Proceedings of the London Mathematical Society **2**, 230 (1936).

[30] P. Benioff, The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines, Journal of Statistical Physics **22**, 563 (1980).

[31] Y. I. Manin, *Computable and uncomputable (in Russian)*, 128 p (Sovetskoe Radio, Moscow, 1980).

[32] D. David, Quantum theory, the church–turing principle and the universal quantum computer, Proc. R. Soc. Lond. A **400**, 97 (1985).

[33] A. Matuschak and M. Nielsen, Quantum computing for the very curious, https://quantum.country/qcvc.

[34] 40 years of quantum computing, Nature Reviews Physics **4**, 1 (2022).

[35] A. Smith, *A practical introduction to quantum computing*, PHYS4039 Physics of Deep Learning, University of Nottingham (2022).

[36] For it to be a *unitary* matrix, it must be true that $U^\dagger U = I$, where $U^\dagger$ is the adjoint of U, and I is the identity matrix.

[37] M. S. ANIS, Abby-Mitchell, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, V. Ajith, I. Y. Akhalwaya, and et al, Qiskit: An open-source framework for quantum computing (2021).

[38] Y. Du, T. Huang, S. You, M.-H. Hsieh, and D. Tao, Quantum circuit architecture search for variational quantum algorithms, npj Quantum Information **8**, 62 (2022).

[39] Y. Du, M.-H. Hsieh, T. Liu, and D. Tao, Expressive power of parametrized quantum circuits, Physical Review Research **2**, 10.1103/physrevresearch.2.033125 (2020).

[40] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, Variational quantum circuits for deep reinforcement learning (2019).

[41] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Parameterized quantum circuits as machine learning models, Quantum Science and Technology **4**, 043001 (2019).

[42] S. Wu, S. Jin, D. Wen, and X. Wang, Quantum reinforcement learning in continuous action space (2020).

[43] "In the neural network, every neuron receives input from all neurons of the previous layer. The single-qubit classifier receives information from the previous processing unit and the input (introduced classically). It processes everything all together and the final output of the computation is a quantum state encoding several repetitions of input uploads and processing parameters" [50].

[44] R. Sweke, F. Wilde, J. Meyer, M. Schuld, P. K. Faehrmann, B. Meynard-Piganeau, and J. Eisert, Stochastic gradient descent for hybrid quantum-classical optimization, Quantum **4**, 314 (2020).

[45] S. Gammelmark and K. Mølmer, Quantum learning by measurement and feedback, New Journal of Physics **11**, 033017 (2009).

[46] O. Lockwood and M. Si, Reinforcement learning with quantum variational circuits (2020).

[47] Parametrized quantum circuits for reinforcement learning (2022).

[48] R. S. S. A. G. Barto and C. W. Anderson, "neuronlike adaptive elements that can solve difficult learning control problems,", IEEE Transactions on Systems, Man, and Cybernetics **SMC-13**, 834 (1983).

[49] A. Alharin, T.-N. Doan, and M. Sartipi, Reinforcement learning interpretation methods: A survey, IEEE Access **8**, 171058 (2020).

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

[51] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing atari with deep reinforcement learning (2013).

[52] E. Farhi, J. Goldstone, and S. Gutmann, A quantum approximate optimization algorithm (2014).

[53] N. Zettili, *Quantum mechanics: concepts and applications*, 2nd ed. (Wiley-Blackwell, Chichester, 2009).