Quantum agents in the Gym: a variational quantum algorithm for deep Q-learning

Andrea Skolik,[1, 2] Sofiene Jerbi,[3] and Vedran Dunjko[1]
[1]*Leiden University, Niels Bohrweg 1, 2333 CA Leiden, The Netherlands*
[2]*Volkswagen Data:Lab, Ungererstraße 69, 80805 Munich, Germany*
[3]*Institute for Theoretical Physics, University of Innsbruck, Technikerstr. 21a, A-6020 Innsbruck, Austria*
(Dated: March 30, 2021)

Quantum machine learning (QML) has been identified as one of the key fields that could reap advantages from near-term quantum devices, next to optimization and quantum chemistry. Research in this area has focused primarily on variational quantum algorithms (VQAs), and several proposals to enhance supervised, unsupervised and reinforcement learning (RL) algorithms with VQAs have been put forward. Out of the three, RL is the least studied and it is still an open question whether VQAs can be competitive with state-of-the-art classical algorithms based on neural networks (NNs) even on simple benchmark tasks. In this work, we introduce a training method for parametrized quantum circuits (PQCs) to solve RL tasks for discrete and continuous state spaces based on the deep Q-learning algorithm. To evaluate our model, we use it to solve two benchmark environments from the OpenAI Gym, Frozen Lake and Cart Pole. We provide insight into why the performance of a VQA-based Q-learning algorithm crucially depends on the observables of the quantum model and show how to choose suitable observables based on the RL task at hand. We compare the performance of our model to that of a NN for agents that need similar time to convergence, and find that our quantum model needs approximately one-third of the parameters of the classical model to solve the Cart Pole environment in a similar number of episodes on average. We also show how recent separation results between classical and quantum agents for policy gradient RL can be adapted to quantum Q-learning agents, which yields a quantum speed-up for Q-learning. This work paves the way towards new ideas on how a quantum advantage may be obtained for real-world problems in the future.

I. INTRODUCTION

Variational quantum algorithms are among the most promising candidates to show quantum advantage in the near-term. Quantum machine learning has emerged as one field that is amenable to applications of VQAs on noisy intermediate-scale quantum (NISQ) devices. Many proposals for QML algorithms have been made in supervised [1–5] and unsupervised [6–11] learning. In contrast, RL is a subfield of machine learning that has received less attention in the QML community [12, 13], and especially proposals for VQA-based approaches are only now emerging [14–18]. RL is essentially a way to solve the problem of optimal control. In a RL task, an agent is not given a fixed set of training data, but learns from interaction with an environment. Environments are defined by a space of states they can be in, and a space of actions that an agent uses to alter the environment's state. The agent chooses its next action based on a policy (probability distribution over actions given states) and receives a reward at each step, and the goal is to learn an optimal policy that maximizes the long-term reward the agent gets in the environment. State and action spaces can be arbitrarily complex, and it's an open question which types of models are best suited for these learning tasks. In classical RL, using NNs as function approximators for the agents' policy has received increased interest in the past decade. As opposed to learning exact functions to model agent behavior which is infeasible in large state and action spaces, this method of RL only approximates the optimal function that models the agents' behavior. One classical RL algorithm that has gained much popularity in this context is called *Q-learning* [19], where the goal is to learn *Q-values*, numerical values which for a given state-action pair represent the expected future reward. Based on these values, a Q-learning agent chooses the appropriate action in a given state, where a higher Q-value corresponds to a higher expected reward. The NN's role in a Q-learning algorithm is to serve as a function approximator for the Q-function. These types of RL algorithms have been shown to play Atari arcade games as well as human players [20], and even reach super-human levels of performance on games as complex as Go [21], Dota [22] and StarCraft [23].

It is thus natural to ask whether RL algorithms can be adapted to work with a VQA approach, and whether such algorithms could offer an advantage over their classical counterparts. RL is one of the hardest modes of learning in current ML research, and especially NN-based approaches are known to require careful tuning of model architectures and hyperparameters to perform well. It is thus to be expected that quantum models in a VQA-based RL approach also need to be selected carefully. This question is directly related to choices made when defining an architecture for a VQA. There are three important factors to consider: the structure (or *ansatz*) of the model, the data-encoding technique, and the readout operators. For the choice of structure, there is a trade-off between the expressivity and trainability of a model, as certain structures are subject to the so-called barren plateau phenomenon [24]. This phenomenon prevents successful training of models with a large number of qubits and layers for highly expressive structures like random circuits. On the other hand, overparametrization has been observed to simplify optimization landscapes and lead to faster convergence for certain VQAs [25, 26]. Apart from that, the choice of structure is also limited by hardware constraints like the topology of a certain quantum device. While the model structure is an important factor in training VQAs that has received much attention in the QML community [27–33], the authors of [34] have shown that the technique used to encode data into the model plays an equally important role, and that even highly expressive structures fail to fit simple functions with an insufficient data-encoding strategy. A less explored architectural choice in the context of QML is that of the observables used to read out information from the quantum model. Considering that the readout operator of a quantum model fixes the range of values it can produce, this choice is especially important for tasks where the goal is to fit a real-valued function with a given range, as is the case in many RL algorithms. This is in contrast to NNs, which have no restriction on the range of output values and can even change this range dynamically during training. In Q-learning, the goal is to approximate the real-valued optimal Q-function, which can have an arbitrary range based on the environment. Crucially, this range can change depending on the performance of the agent in the environment, which is an impediment for quantum models with a fixed range of output values.

A first step to study the influence of architectural choices on PQC-based RL algorithms has been made in the context of policy gradients in [18], who point out that data-encoding and readout strategies play a crucial role in these types of RL tasks. Previous work on Q-learning with PQCs has also addressed certain other fundamental questions about the applicability of VQAs in this context. A VQA for Q-learning in discrete state spaces was introduced in [14], where the quantum model's output is followed by a layer of additive weights, and it has been shown that the model successfully solves two discrete-state environments. A VQA for Q-learning in environments with continuous and discrete state spaces has been proposed in [15], who simplify the continuous environments' potentially infinite range of input values to a restricted encoding into angles of one initial layer of rotation gates, and use measurements in the computational basis to represent Q-values. Notably, none of the models that were run for the continuous state-space environment Cart Pole reach a performance that is considered to be solving the environment according to its original specification [35].

These initial works prompt a number of vital follow-up questions related to the architectural choices that are required to succeed in arbitrary RL environments with a quantum Q-learning agent. We address these questions in form of our main contributions as follows: first, we propose a VQA to solve discrete- and continuous-state RL environments and explain the intricate relationship between the environment's specification and the requirements on the readout operators of the quantum model. We show how a quantum Q-learning agent only succeeds if these requirements are met. Second, to enable the model to match the environment's requirements on the range of output values, we make this range itself trainable by introducing additional weights on the model outputs. We show how the necessity of these weights can be inferred from the range that the optimal Q-values can take in an environment. Third, we study the performance of our model on two benchmark environments from the OpenAI Gym [36], Frozen Lake and Cart Pole. For the continuous-state Cart Pole environment, we also study a number of data encoding methods and illustrate the benefit of previously introduced techniques to increase quantum model expressivity, like data re-uploading [37] or trainable weights on the input data [18, 37]. Additionally, the state space dimension of both environments is small enough so that inputs can be directly encoded into the quantum model without the use of a dimensionality reduction technique. This makes it possible to directly compare our model to a NN performing the same type of Q-learning algorithm to evaluate its performance. Specifically, we compare our results on the Cart Pole environment to those of a classical NN, and find that our model is competitive with the classical model. Our model produces agents which solve the environment with roughly one-third of the parameters required by a NN, where we compare configurations that need a similar number of episodes to reach this goal. Finally, we briefly discuss the recent results demonstrating a classical/quantum separation in RL with policy gradients [18], and show how those results can be adapted to prove the existence of environments where the learning of optimal Q-functions can only be achieved by quantum agents, barring some widely-believed complexity-theoretic assumptions.

The remainder of this paper is structured as follows: in section II we give an introduction to RL, followed by a description of our quantum RL model in section III. We present numerical results in section IV and discuss our findings in section V.

II. REINFORCEMENT LEARNING

In RL, an agent doesn't learn from a fixed data set as in other types of learning, but by making observations on and interacting with an environment [38]. This distinguishes it from the other two main branches of ML, supervised and unsupervised learning, and each of the three comes with its individual challenges. In a supervised setting, an agent is given a fixed set of training data that is provided with the correct labels, where difficulties arise mainly in creating models that don't overfit the training data and keep their performance high on unseen samples. In unsupervised learning, training data is not labeled and the model needs to discover the underlying structure of a given data set, and the challenge lies in finding suitable loss functions and training methods that enable this. RL also comes with a number of challenges:

there is no fixed set of training data, but the agent generates its own samples by interacting with an environment. These samples are not labeled, but only come with feedback in form of a reward. Additionally, the training data keeps changing throughout the learning process, as the agent constantly receives feedback from interacting with its environment.

An environment consists of a set of possible states $\mathcal{S}$ that it can take, and a set of actions $\mathcal{A}$ which the agent can perform to alter the environment's state. Both state and action spaces can be continuous or discrete. An agent interacts with an environment by performing an action a_t at time step t in state s_t, upon which it receives a reward r_{t+1}. A tuple (s, a, r, s') of these four quantities is called a *transition*, and transition probabilities from state s to s' after performing action a in a given environment are represented by the *transition function* $P^a_{ss'}$,

$$P^a_{ss'} = P(s'|s, a). \tag{1}$$

The reward function is designed to evaluate the quality of the agent's actions on the environment based on the learning task at hand, and the agent's goal is to maximize its total reward over a sequence of time steps, called the return G_t

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

where $\gamma \in [0, 1]$ is a discount factor introduced to prevent divergence of the infinite series. Environments often naturally break down into so-called *episodes*, where the sum in eq. (2) is not infinite, but bounded by a fixed number of steps called *horizon* H. An example of this are environments based on games, where one episode comprises one game played and an agent learns by playing a number of games in series. The function that models the agent's behavior in the environment is called the *policy* $\pi(s, a)$, which gives the probability of taking action a in a given state s. The performance of the algorithm is evaluated based on the state-value function $V_\pi(s)$,

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s], \tag{3}$$

which is the *expected return* when following policy π starting from state s, and one way to formulate the goal of a RL algorithm is to learn the optimal policy π_* which maximizes the expected return for each state. Note that the task is to maximize an expected value, and that the reward r_t in eq. (2), and therefore G_t are random variables. There are a number of different flavors of RL algorithms, and we restrict our attention to a specific type of algorithm called *Q-learning*.

A. Q-learning

In Q-learning, we are not interested in the state-value function as shown in eq. (3), but in the closely related action-value function $Q_\pi(s, a)$,

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a], \tag{4}$$

which also gives us the expected return assuming we follow a policy π, but now additionally conditioned on an action a. We call the optimal Q-function $Q_*(s, a) = \max_\pi Q_\pi(s, a)$, and an optimal policy can be easily derived from the optimal values by taking the highest-valued action in each step, as

$$\pi_*(s, a) = \underset{a}{\operatorname{argmax}}\ Q_*(s, a). \tag{5}$$

The goal in Q-learning is to learn an estimate, $Q(s, a)$, of the optimal Q-function.

In its original form, Q-learning is a tabular learning algorithm, where a so-called *Q-table* stores Q-values for each possible state-action pair [19]. When interacting with an environment, an agent chooses its next action based on a policy, where with a given probability the next action is chosen depending on the Q-values as

$$a_t = \underset{a}{\operatorname{argmax}}\ Q(s_t, a), \tag{6}$$

where a higher value designates a higher expected reward when action a is taken in state s_t as opposed to the other available actions. If the next action is not chosen based on the highest Q-values, it depends on the policy which defines the probability with which other, possibly sub-optimal, actions are picked. The Q-values are updated with observations made in the environment by the following update rule,

$$\begin{aligned} Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \cdot \max_a Q(s_{t+1}, a) \\ - Q(s_t, a_t)], \end{aligned} \tag{7}$$

where α is a learning rate, r_{t+1} is the reward at time $t + 1$, and γ is a discount factor. In the limit of visiting all (s, a) pairs infinitely often, this update rule is proven to converge to the optimal Q-values in the tabular case [39]. Q-values can take an arbitrary range, which is determined by the environment's reward function and the discount factor γ, which controls how strongly expected future rewards influence the agent's decisions. Depending on γ, the optimal Q-values for the same environment can take highly varying values, and can therefore be viewed as different learning environments themselves. In practice, it is not necessary that an agent learns the optimal Q-values exactly. As the next action at step t is chosen according to eq. (6) and a policy π, it is sufficient that the action with the highest expected reward has the highest Q-value for the sake of solving an environment presuming a deterministic policy. In other words, for solving an environment only the *order* of Q-values is important, and the task is to learn this correct order by observing rewards from the environment through interaction. The situation is slightly less trivial for non-deterministic policies, where each action is not only chosen based on the highest Q-value, but in a probabilistic manner given by the policy. The policy in the Q-learning algorithm we study in this work is given by the Q-values, and a probability with which an agent performs the action suggested by eq. (6), or one of the other available actions randomly. Here, if the agent chooses its action based on eq. (6), the action with the highest Q-values is chosen deterministically, so in that sense the order of Q-values is essential here.

Obviously, this tabular approach is intractable for large state and action spaces. For this reason, the Q-table was replaced in subsequent work by a Q-function approximator which doesn't store all Q-values individually [40, 41]. In the seminal work [20], the authors use a NN as the Q-function approximator which they call *deep Q-network* (DQN) and the resulting algorithm the *DQN algorithm*, and demonstrate that this algorithm achieves human-level performance on a number of arcade games. In this work, the agent chooses actions based on an ϵ-greedy policy, which means with probability ϵ, a random action is performed and with probability $1-\epsilon$ the agent chooses the action which corresponds to the highest Q-value for the given state. Typically ϵ is chosen large in the beginning and then decayed in subsequent iterations, to ensure that the agent can sufficiently explore the environment at early stages of training by being exposed to a variety of states. The authors of [20] also utilize two other improvements over previous approaches which we adopt: (i) *experience replay*: past transitions and their outcomes are stored in a memory, and for updates sampled at random from this memory to remove temporal correlations between transitions, (ii) adding a second so-called *target network* to compute the expected Q-values for the update rule, where the target network has an identical structure as the DQN, but is not trained and only sporadically updated with a copy of the parameters of the DQN to increase stability during training. We refer to the original paper for further details on the necessity of these techniques to stabilize training of the DQN algorithm.

The DQN is then trained almost in a supervised fashion, where the training data and labels are generated by the DQN itself through interaction with the environment. At each update step, a batch $\mathcal{B}$ of previous transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ is chosen from the replay memory. To perform a model update, we need to compute $\max_a Q(s_{t+1}, a)$. When we use a target network, this value is not computed by the DQN, but by the target network $\hat{Q}$. To make training more efficient, in practice the Q-function approximator is redefined as a function of a state parametrized by $\vec{\theta}$, $Q_{\vec{\theta}}(s) = \vec{q}$, which returns a vector of Q-values for all possible actions instead of computing each $Q(s,a)$ individually. We now want to perform a supervised update of $Q_{\vec{\theta}}$, where the target value is given by the update rule in eq. (8). To compute the target value for a state s that we have taken action a on in the past, we take a copy of $Q_{\vec{\theta}}(s)$ which we call $\vec{q_\delta}$, and only the ith entry of $\vec{q_\delta}$ is altered where i corresponds to the index of the action a, and all other values remain unchanged. The estimated maximum Q-value for the following state s_{t+1} is computed by $\hat{Q}_{\vec{\theta}_\delta}$, and the update rule for the i-th entry in $\vec{q_\delta}$ takes the following form

$$\vec{q_\delta}_i = r_{t+1} + \gamma \cdot \max_a \hat{Q}_{\vec{\theta}_\delta}(s_{t+1}, a), \tag{8}$$

where $\vec{\theta}_\delta$ is a periodically updated copy of $\vec{\theta}$. The loss function $\mathcal{L}$ is the mean squared error (MSE) between $\vec{q}$ and $\vec{q_\delta}$ on a batch of sample transitions $\mathcal{B}$,

$$\mathcal{L}(\vec{q}, \vec{q_\delta}) = \frac{1}{|\mathcal{B}|} \sum_{b \in \mathcal{B}} \sqrt{\sum_{i=1}^{|A|} (\vec{q}_{b_i} - \vec{q_\delta}_{b_i})^2}. \tag{9}$$

Note that because $\vec{q_\delta}$ is a copy of $\vec{q}$ where only the i-th element is altered via the update rule in eq. (8), the difference between all other entries in those two vectors is zero.

III. QUANTUM Q-LEARNING

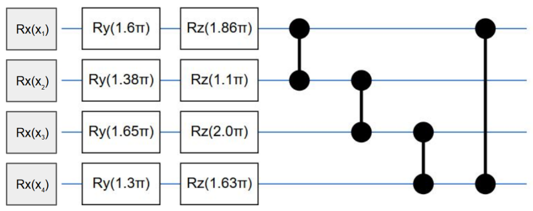


FIG. 1. PQC architecture used in this work. Each layer consists of a parametrized rotation along the Y and Z axes on each qubit, and a daisy chain of CZ gates. The grey boxes with X rotations at the beginning correspond to data encoding gates. When data re-uploading is used, the whole circuit pictured is repeated in each layer, without data re-uploading only the variational part without the initial X rotations is repeated.

In this work, we adapt the DQN algorithm to use a PQC as its Q-function approximator instead of a NN. For this, we use a hardware-efficient ansatz [42] as shown in fig. 1. This ansatz is known to be highly expressive, and is susceptible to the barren plateau phenomenon for a large number of qubits and layers, although this is not an issue for the small state and action spaces we consider here. All other aspects of the Q-learning algorithm described in section II A stay the same: we use a target network, an ϵ-greedy policy to determine the agent's next action, and experience replay to draw samples for training the Q-network PQC. Our Q-network PQC is then $U_{\vec{\theta}}(s)$ parametrized by $\vec{\theta}$ and the target network PQC is $\hat{U}_{\vec{\theta}_\delta}(s)$, where $\vec{\theta}_\delta$ is a snapshot of the parameters $\vec{\theta}$ which is taken after fixed intervals of episodes δ and the circuit is otherwise identical to that of $U_{\vec{\theta}}(s)$.

A. Encoding environment states

Depending on the state space of the environment, we distinguish between two different types of encoding in this work:

Discrete state space: Discrete states are mapped to bitstrings and then input into the model, where on an all-zero state the bits corresponding to ones in the input state are flipped.

Continuous state space: For continuous input states, we scale each component x of an input state vector $\vec{x}$ to $x' = \arctan(x) \in [-\pi/2, \pi/2]$ and then perform a variational encoding, which consists of rotations in the X direction by the angles x'.

As shown in [34], when data is encoded into a PQC by local rotation gates along the X-axis, the PQC can only model simple sine functions of its input. To further increase the expressivity of the circuit, the data encoding can be repeated in two ways: either in parallel by increasing the number of qubits and duplicating the data encoding on them, or in sequence in an alternating fashion with the variational layers of the circuit. The latter is also referred to as *data re-uploading* in [37]. Where needed, we will introduce data re-uploading to our model in section IV.

The formalism introduced in [34] establishes a connection between PQCs and partial Fourier series by showing that the functions a given PQC can model can be represented as a Fourier series, where the accessible frequency spectrum depends on the eigenvalues of the data encoding gates, and the coefficients depend on the architecture of the variational part of the PQC and the observable that defines the readout operation. They show that in models as ours, where data is encoded in form of Pauli rotations, only Fourier series up to a certain degree can be learned, where the degree depends on the number of times the encoding gate is repeated. Additionally, the scale of the input data must match the scale of the frequencies of the modeled function for the model to fit the target function exactly. Making the scaling of input data itself trainable to increase a PQC's expressivity has been suggested in [18, 37], which we will also use by introducing a weight $\vec{w}_d$ on the input data. The input value x_i' then becomes:

$$x_i' = \arctan(x_i \cdot w_{d_i}) \ , \tag{10}$$

where w_{d_i} is the weight for input x_i. We will illustrate the advantage of these enhanced data-encoding strategies numerically in section IV.

B. Computing Q-values

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

$$Q(s,a) = \langle 0^{\otimes n} | \, U^{\dagger}_{\vec{\theta}}(s) O_a U_{\vec{\theta}}(s) \, | 0^{\otimes n} \rangle \ , \tag{11}$$

where O_a is an observable and n the number of qubits, and our model outputs a vector of Q-values for each possible action as described in section II A. The exact definition of observables depends on the size of the action space, where the output of the model is a vector of Q-values $\vec{q} \in \mathbb{R}^{|A|}$. The type of measurements we perform to estimate Q-values will be described in more detail in section IV for each environment. The way Q-values are read out from the PQC is an important factor that determines the success at solving the environment at hand. A key difference between PQCs and NNs is that a PQC has a fixed range of output defined by its measurements, while a NN's range of output values can change arbitrarily during training depending on its weights and activation function. To understand why this is an important difference in a RL setting, we need to recall that Q-values are an estimate of the expected return

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi\left[\sum_{k=0}^{H} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right] . \end{aligned}$$

In each episode performed during training, the model needs to have the ability to match the range of possible Q-values in order to approximate the optimal Q-function. This means that the observables in a PQC-based Q-learning agent need to be chosen with care, and highly depend on the specific environment. We will provide detailed descriptions and motivation for our choice of observables for each of the benchmark environments we study in section IV.

C. Prospects on quantum advantage

A recent work on policy-gradient RL with PQCs introduces three RL environments based on the discrete logarithm problem which can't be learned by any polynomial-time classical agent [43], but where an efficient quantum learner can achieve near-perfect performance [18]. The proof of this statement comprises two parts: first, the authors show that no classical agent can solve these environments efficiently, assuming the widely-believed hardness of the discrete logarithm problem, and second, they construct a quantum agent that achieves close-to-optimal performance, where performance is measured by a value function as that in eq. (3).

This same construction can be used directly to obtain analogous claims for the Q-learning case. In particular, the classical hardness of obtaining optimal policies immediately implies that good approximations of Q-functions can't be obtained either, as optimal Q-functions imply optimal policies. The quantum learnability holds as well,

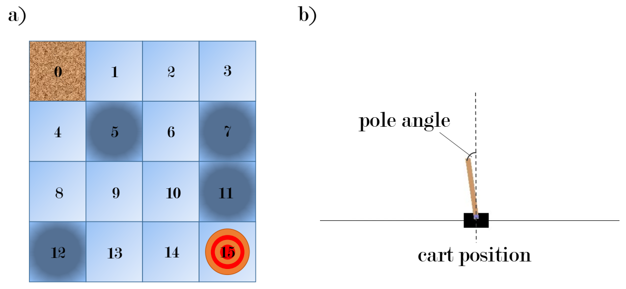


FIG. 2. Gym environments solved by the quantum model. a) Frozen Lake environment, where an agent needs to learn to navigate from the top left of a grid to retrieve the Frisbee at the bottom right without falling into any of the holes (dark squares), b) Cart Pole environment, which consist of learning to balance a pole on a cart which moves left and right on a frictionless track.

in essence: the existence of an agent which attains optimal policies implies that optimal Q-values can be approximated as well. Specifically, with access to an optimal policy, any given Q-value can be estimated via direct Monte Carlo sampling, since the optimal Q-function is defined as the expected return of an optimal policy.

However, all these results utilize artificially constructed learners, and also involve sampling from the environment to compute Q-function estimates. It is an open question whether the same performance guarantees can be achieved with a learning algorithm that uses Q-learning updates of the type of eq. (7).

IV. NUMERICAL RESULTS

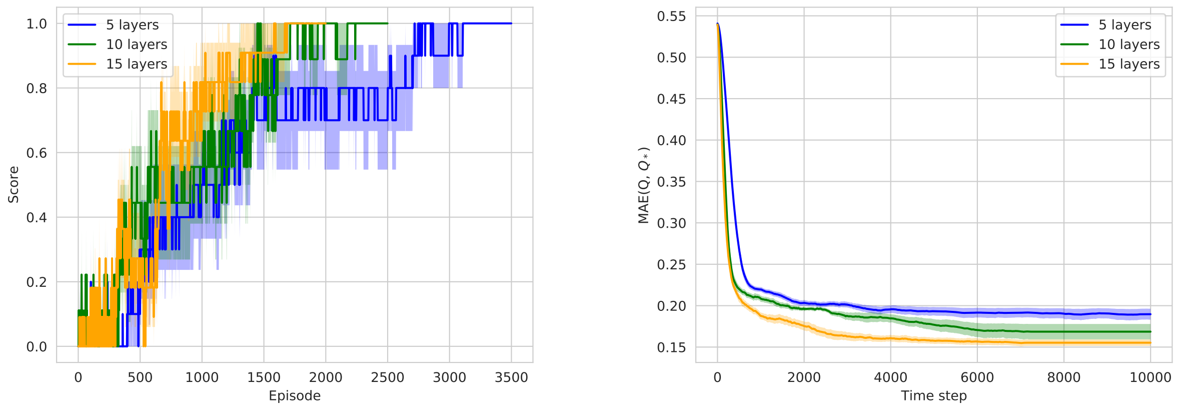
In this section, we present results for our PQC model on two benchmark RL tasks from the OpenAI Gym [36], Frozen Lake v0 [44] and Cart Pole v0 [35] (see fig. 2). We ran an extensive hyperparameter search for both environments, and present our results for the best sets of hyperparameters. A detailed description of the hyperparameters we tuned and their best values can be found in appendix A. Our experiments were run with TensorFlow Quantum [45] and Cirq [46].

A. Frozen Lake

The Frozen Lake (FL) environment consists of a 4x4 grid representing a frozen surface, where the agent can choose to move one step up, down, left or right. The goal is to cross the lake from the top left corner to the bottom right corner where the goal is located. However, some of the grid positions correspond to holes in the ice, and when the agent steps on them the episode terminates and it has to start again from the initial state. In each episode, the agent is allowed to take a maximum number of steps m_{max}. The episode terminates if one of the following conditions is met: the agent performs $m_{max} = 200$ steps, reaches the goal, or falls into a hole. For each episode in which the goal is reached the agent receives a reward of 1, and a reward of 0 otherwise. The environment is considered solved when the agent reaches the goal for 100 contiguous episodes. (See [44] for full environment specification.)

As the FL environment is discrete and the dimensions of the state and action spaces are small, it is not a natural use-case for Q-function approximation. However, the first work which introduced PQCs as Q-function approximators only did so on two discrete environments, FL and the cognitive radio task [14], and for completeness we also include results of our model on the FL environment. We note that we don't directly compare our results to those in [14], as they use an altered version of the environment which differs from that given in the original specification [44]. In their version, each transition is rewarded instead of just the one to the goal state, which makes the learning task easier as the agent receives immediate rewards for every action taken. The FL environment is also interesting from the perspective that there are only 64 Q-values which we can compute exactly, and therefore we can directly compare the Q-values learned by our model to the optimal Q-values Q_*, which we do in fig. 3 b).

The FL environment has 16 states (one for each square on the grid) of which four are holes (marked as darker squares in fig. 2 a), and 4 actions (top, down, left, right). We encode each position on the grid as one of the computational basis states of a 4-qubit system, without use of trainable input data weights or data re-uploading. The optimal Q-values for each state-action pair can be computed as $Q_*(s,a) = \gamma^\beta$ (cf. eq. (4)), where β is the number of steps following the shortest path to the goal from the state s' that the agent is in after the transition (s,a). We will now motivate our choice of observables for the FL agent by studying the range the optimal Q-values can take. Note that these optimal Q-values are defined for the tabular case only, and serve as a reference for the Q-values we want our Q-function approximator to model. We know that only one transition, that from state 14 to the goal state 15, is rewarded. This corresponds to a Q-value $Q_*(14,R) = \gamma$. As the only other state adjacent to the goal (state 11) is a hole, no other transition in this environment is rewarded. Through the recursive Q-value update rule (see eq. (7)), all other Q-values depend on $Q_*(14,R)$, and are smaller due to the discount factor and the zero reward of all other transitions. In case of a function approximator, the Q-values may not be the same as the optimal values, but the relationship between $Q(14,R)$ and all other Q-values still applies. This means that we have an upper bound on the range of Q-values that we want to model which only depends on $\gamma \leq 1$ and stays constant over all episodes. Therefore we don't expect that Q-values need to become larger than γ for our



(a) average scores (b) mean absolute error with optimal Q-values

FIG. 3. Agents with varying depth playing the Frozen Lake environment. The environment is solved when the agent reaches the goal (receives a score of 1) for 100 contiguous episodes. a) Average score over 10 agents for circuits of depth 5, 10, and 15, respectively. All agents manage to solve the environment, higher circuit depth leads to lower time to convergence. Shaded area shows standard deviation from the mean. b) Mean absolute error between agents' Q-values and the optimal Q-values Q_* for all (s, a) pairs over time steps in episodes, where one time step corresponds to one transition in the environment. Shaded region shows standard error of the mean.

agent to solve the environment, and only become larger in practice if the initialization of our model happens to yield higher values for some state-action pairs. Motivated by this, we represent the Q-values for the four actions as the expectation values of a measurement in the computational basis with the operator Z_i for each of the four qubits $i \in \{1, \ldots, 4\}$, which we scale to lie between $[0, 1]$ instead of $[-1, 1]$. Note that even when parameter initialization yields Q-values higher than the largest optimal Q-value, they will still be close to this value as both optimal Q-values and those of our model are upper-bounded by 1. Figure 3 a) shows the average scores of ten agents, each configuration trained with a circuit depth of 5, 10, and 15 layers, respectively. All agents manage to solve the environment, and the time to convergence decreases as the number of layers increases. Figure 3 b) shows the averaged mean absolute error (MAE) between the optimal Q-values and the Q-values produced by the agents at each time step during training. The agents trained on circuits of depth 15 reach the lowest values and converge earlier to an average MAE that is roughly 0.05 lower than that of the agents trained on a circuit of depth 5.

B. Cart Pole

In the Cart Pole v0 environment, an agent needs to learn to balance a pole upright on a cart that moves left and right on a frictionless track. Its state space is continuous and consists of the following variables: cart position, cart velocity, pole angle, and pole velocity at the tip. The cart position is bounded between ± 2.4, where values outside of this range mean leaving the space that the environment is defined in and terminating the episode. The pole angle is bounded between $\pm 41.8°$. The other two variables can take infinite values, but are bounded in practice by how episode termination is defined. An episode terminates if the pole angle is outside of $\pm 12°$, the cart position is outside of ± 2.4, or the agent reaches the maximum steps per episode $m_{max} = 200$. For each step of the episode (including the terminal step) the agent receives a reward of one. At the beginning of each episode, the four variables of the environment state are randomly initialized in a stable state within the range [-0.05, 0.05]. The episode score is computed as the cumulative reward of all steps taken in the episode. The environment is solved when the average score of the last 100 episodes is ≥ 195. (See [35, 38] for full environment specification.)

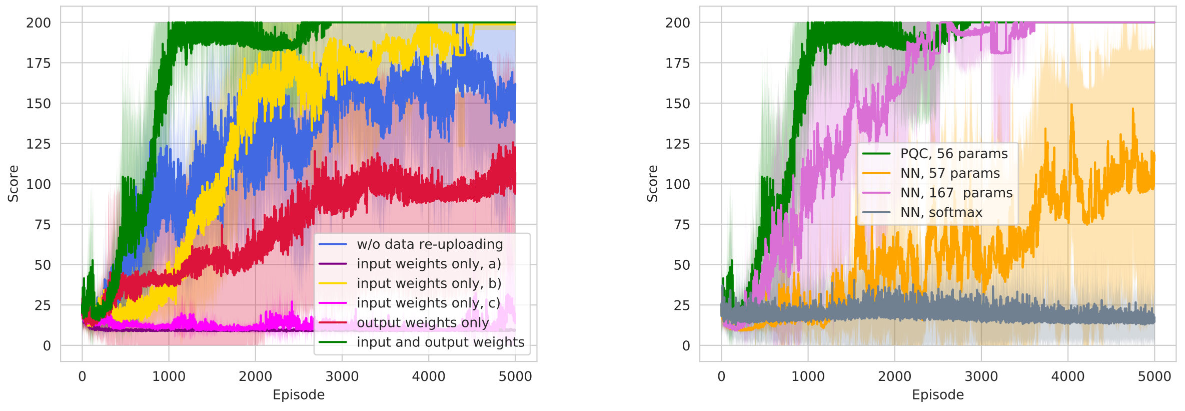
As in section IV A, we now motive our choice of observables depending on how rewards are received in this environment. For this, we recall that a Q-value gives us the expected return for a given state-action pair,

$$Q_\pi(s, a) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$

Cart Pole is an episodic environment with a maximum number of time steps $H = 200$ in the version of the environment we study here, so the Q-value following optimal policy π_* from a stable state s is

$$Q_*(s, a) = \sum_{k=0}^{H-1} \gamma^k.$$

When following an arbitrary policy π and starting in a



(a) average scores with varying trainable weights

(b) average scores compared to NN

FIG. 4. Agents training on the Cart Pole environment, illustrating the importance of trainable weights and data re-uploading. The environment is solved when an agent has an average reward $\geq$ 195 for the past 100 episodes, after which training is stopped. Results are averaged over 10 agents each. a) Agents with same ansatz and hyperparameter configurations, except for which weights are trained and how data is encoded. Purple line at the bottom ("input only, a)") shows agents with trainable weights only on the input (w_d), and a fixed range of output values $[0, 1]$. As described in section III B, this prohibits the reward mechanism from taking effect, and the agents can not distinguish from good and bad actions. Yellow line ("input only, b)") shows the same configuration, but with fixed output range of $[0, 90]$, where the upper bound is motivated by the range of optimal Q-values. The magenta curve ("input only, c)") shows the same configuration but with a fixed output within range $[0, 180]$. Red line (middle) shows agents with trainable weights on outputs only (w_o). Average scores increase over the course of 5000 episodes, albeit much slower than for the green line (top), where both input and output weights are trainable. The blue line shows agents trained with the same hyperparameters as those in the green curve, but without data re-uploading. b) Our model performs better than a NN with 3 dense layers and 57 parameters, and similarly to a NN with 3 dense layers with 167 parameters. A NN with a final softmax layer that restricts its range of output values to $[0, 1]$, similar to a PQC with fixed range of outputs, fails to solve the environment.

random stable state of the environment, the Q-value is

$$Q_\pi(s,a) = \sum_{k=0}^{h-1} \gamma^k,$$

where $h \leq H$ is the length of the episode which is determined by the policy. The longer the agent balances the pole, the higher h, with $h = H$ the maximum number of steps allowed in an episode. When not considering random actions taken by the ϵ-greedy policy, h depends solely on the performance of the agent, which changes as the agent gets better at balancing the pole. Consequently, the Q-values we want to approximate are lower bounded by the minimum number of steps it takes to make the episode terminate when always picking the wrong action (i.e., the pole doesn't immediately fall by taking one false action alone), and upper bounded by the Q-values assuming the optimal policy, where $h = H$. We stress that this upper bound applies to the optimal policy in one episode only, and that in practice the upper bound of the magnitude of Q-values during training depends on the performance of the agent as well as the number of episodes played. Compared to the range of expectation values of computational basis measurements these values can become very high, e.g. for $\gamma = 0.99$ we get max $Q_*(s,a) \approx 86$. Even when considering that Q-values need not necessarily be close to the optimal values to solve an environment, the range given by computational basis measurements is clearly too small compared to the frequency with which rewards are given and the number of episodes needed until convergence.

There are two ways to approach this issue: (i) multiply PQC outputs by a fixed factor to increase their range which accommodates the theoretical maximum Q-value, (ii) allow the PQC to change its range of outputs during training. Multiplying the outputs of the PQC by a fixed factor increases the range of values, but at the cost of potentially being close to the estimated maximum from the beginning. In particular, as Q-values are initialized randomly depending on the initial parameters of the PQC, the Q-values for actions of a specific state might have large differences. Considering that the reward which controls the magnitude of change given by the Q-value updates in eq. (8) is comparatively small and actions are picked based on the policy and $\text{argmax}_a Q(s,a)$, it may take a long time before subsequent updates of Q-values will lead to the agent picking the right actions. Even if we consider models that are initialized such that all Q-values are close to zero in the beginning, the actual changes in the rotation angles that the PQC needs to perform for Q-values of large ranges can become very small. Especially on NISQ devices, these changes might be impractically small to be reliably performed and measured on hardware. For these reasons, we focus on option (ii). We add a trainable weight $w_o \in \mathbb{R}$ to each readout operation, so that the output Q-value $Q(s,a)$ becomes

$$Q(s,a) = \frac{\langle 0^{\otimes 4}| \, U_{\vec{\theta}}(s)^\dagger O_a U_{\vec{\theta}}(s) \, |0^{\otimes 4}\rangle + 1}{2} \cdot w_{o_a}, \qquad (12)$$

where $O_L = Z_1 Z_2$ and $O_R = Z_3 Z_4$ are Pauli-ZZ operators on qubits $(1,2)$ and $(3,4)$ respectively, corresponding to actions left and right, and each Q-value has a separate weight w_{o_a}. We make the weights multiplicative in analogy to weights in a NN. This gives the model the possibility to naturally increase the magnitude of Q-values as it becomes better at balancing the pole over the course of the played episodes. We also add trainable weights on the input vectors and data-encoding as described in section III A.

To illustrate the effect of trainable weights on the input and output values, we show agents trained with trainable input weights, trainable output weights, and both trainable input and output weights, respectively, in fig. 4 a). This figure illustrates how an agent with no trainable output weights and a fixed range of outputs of $[0,1]$ ("input only, a)") stays at an extremely low score during all 5000 episodes, as it fails to fit a good Q-function approximation. Agents where we extend this range to $[0,90]$ by multiplying the output with a fixed factor of 90 perform much better ("input only, b)") and eventually solve the environment, but training is unstable and these agents take roughly 2000 episodes longer than those in the green curve, which have trainable weights on their output values. We picked the factor of 90 based on the magnitude of optimal Q-values, where the highest value for our choice of $\gamma = 0.99$ is approximately 86. As described above, the magnitude of Q-values crucially depends on the agent's ability to balance the pole in each episode, and as a general trend it will increase over the course of training for agents that perform well. How large the final Q-values of a solving agent are therefore also depends on the number of episodes it requires until convergence, so a range which is upper bounded by 90 presumes agents that converge relatively quickly. Considering the range of final Q-values of agents in the green curve, they can become as high as approximately 176 for agents that converge late. However, as we see for agents with a fixed output range of $[0,180]$ ("input only, c)"), increasing the range to accommodate agents that converge later prohibits agents from increasing their score at all and they perform similarly to those with a fixed range of output values in $[0,1]$. The top, green curve shows how agents with trainable output weights overcome this issue, and all agents in this configuration solve the environment after 3000 episodes at most. The middle, red curve shows agents trained with trainable output weights only. These agents increase their average score over time, but don't solve the environment within 5000 episodes, which illustrates the benefit of making both input and output weights trainable. All the agents mentioned before were trained with data re-uploading, where the data encoding is repeated in each layer of the PQC as depicted in fig. 1. The blue curve shows agents with trainable input and output weights, but without data re-uploading, where we see a clear drop in performance compared to the green curve. The fact that agents with trainable input weights and data re-uploading perform much better than those without, emphasizes the importance of matching the PQC's expressivity to the learning task at hand, as described in [34].

In fig. 4 b) we compare our model to classical NNs with a similar number of parameters. A NN with 3 dense layers (4, 5 and 2 units, respectively) and 57 parameters, which has one more parameter than our PQC model, doesn't manage to solve the environment. The NN with 3 dense layers (9, 10, 2 units) and 167 parameters performs similarly to our PQC model. Additionally, we evaluate the performance of the latter NN with a Softmax layer applied after the last Dense layer, which restricts its output values to $[0,1]$. This is similar in nature to a PQC model which performs measurements in the computational basis without a trainable output weight, and we see that the NN also fails completely in this scenario. This further strengthens our claim that a range of output values that matches the requirements of the environment is essential to solve certain types of RL environments with Q-learning. In all experiments we performed, training is stopped after an agent solves the environment, which means that it balances the pole successfully for 100 episodes in a row, and the score for subsequent episodes in the figure is set to the highest possible value.

We note that unlike for the FL environment, it is not straightforward to compute optimal Q-values for Cart Pole as its state space is continuous. A trained model that is known to implement the optimal policy (i.e., correct ordering of Q-values for all (s,a)-pairs) could be used as a baseline to compare other models to, but the magnitudes of Q-values can highly vary even among agents that solve the environment so this comparison will not provide much insight, which is why we refrain from including it here. Nonetheless, we provide a visualization of the Q-values learned by one of our best-performing quantum models in appendix B. We observe that these Q-values have a maximum value close to what we expected from an optimal agent (i.e., 86).

V. CONCLUSION

In this work, we have proposed a quantum model for deep Q-learning which can solve environments with discrete and continuous state spaces. We have illustrated the importance of picking the observables of a quantum model such that it can represent the range of optimal Q-values that this algorithm should learn to model. One crucial difference between PQCs and classical methods based on NNs, namely the former's fixed range of output values defined by its measurement operators, was identified as a major impediment to successfully perform Q-learning in certain types of environments. Based on the range of optimal Q-values, we illustrate how an informed choice can be made for the quantum model's observables. We also introduce trainable weights on the observables of our model, to achieve a flexible range of output values as given by a NN and show how this is crucial to solve the Cart Pole environment. We use a number of data encoding techniques to enhance the expressivity of our model, namely data re-uploading [37] and trainable weights on the input [18, 37], and numerically show their benefit for the Cart Pole environment, which has a continuous input state space. Our results illustrate the importance of architectural choices for QML models, especially for a RL algorithm as Q-learning that has very specific demands on the range of output values the model can produce. To evaluate the performance of our model, we compare it to a classical approach, where the same DQN algorithm is used with a NN as the Q-function approximator. We compare quantum and classical agents that require a similar number of episodes before solving the environment and find that our model requires roughly one-third of the number of parameters that the classical model does. Additionally, we explained how a recent proof of quantum advantage for policy gradient RL agents [18] directly encompasses Q-learning agents as well. However, this proof only guarantees that this quantum learner can be constructed in general, and not that the optimal policy can be learned by policy gradients or (deep) Q-learning. It is an interesting open question if this performance can also be achieved by a learning algorithm that uses Q-value updates as shown in eq. (7). Our results show that a quantum Q-learning algorithm is competitive with a NN on a simple benchmark environment, but also that these types of quantum RL models may offer guaranteed learning advantages over all classical agents, at least in restricted special cases. This opens up the path to future investigations of possible quantum advantages of these types of quantum agents in relevant settings.

ACKNOWLEDGEMENTS

AS thanks Casper Gyurik for many valuable discussions over the course of this project. This work was in part supported by the Dutch Research Council (NWO/OCW), as part of the Quantum Software Consortium program (project number 024.003.037). VD also acknowledges the support by the project NEASQC funded from the European Union's Horizon 2020 research and innovation programme (grant agreement No 951821), and partial funding by an unrestricted gift from Google Quantum AI. SJ acknowledges support from the Austrian Science Fund (FWF) through the projects DK-ALM:W1259-N27 and SFB BeyondC F7102. SJ also acknowledges the Austrian Academy of Sciences as a recipient of the DOC Fellowship. AS acknowledges funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 828826.

[1] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[2] Maria Schuld, Alex Bocharov, Krysta M Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3):032308, 2020.

[3] Maria Schuld and Nathan Killoran. Quantum machine learning in feature hilbert spaces. *Physical review letters*, 122(4):040504, 2019.

[4] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[5] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[6] Mohammad H Amin, Evgeny Andriyash, Jason Rolfe, Bohdan Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018.

[7] Brian Coyle, Daniel Mills, Vincent Danos, and Elham Kashefi. The born supremacy: Quantum advantage and training of an ising born machine. *npj Quantum Information*, 6(1):1–11, 2020.

[8] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Variational quantum boltzmann machines. *Quantum Machine Intelligence*, 3(1):1–15, 2021.

[9] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.

[10] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. Quantum generative adversarial networks for learning and loading random distributions. *npj Quantum Information*, 5(1):1–9, 2019.

[11] Shouvanik Chakrabarti, Huang Yiming, Tongyang Li, Soheil Feizi, and Xiaodi Wu. Quantum wasserstein generative adversarial networks. In *Advances in Neural Information Processing Systems*, pages 6781–6792, 2019.

[12] A Hamann, V Dunjko, and S Wölk. Quantum-accessible reinforcement learning beyond strictly epochal environments. *arXiv preprint arXiv:2008.01481*, 2020.

[13] Sofiene Jerbi, Lea M Trenkwalder, Hendrik Poulsen Nautrup, Hans J Briegel, and Vedran Dunjko. Quantum enhancements for deep reinforcement learning in large spaces. *PRX Quantum*, 2(1):010328, 2021.

[14] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational quantum circuits for deep reinforcement learning. *IEEE Access*, 8:141007–141024, 2020.

[15] Owen Lockwood and Mei Si. Reinforcement learning with quantum variational circuit. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 16, pages 245–251, 2020.

[16] Shaojun Wu, Shan Jin, Dingding Wen, and Xiaoting Wang. Quantum reinforcement learning in continuous action space. *arXiv preprint arXiv:2012.10711*, 2020.

[17] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[18] Sofiene Jerbi, Casper Gyurik, Simon Marshall, Hans J Briegel, and Vedran Dunjko. Variational quantum policies for reinforcement learning. *arXiv preprint arXiv:2103.05577*, 2021.

[19] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[21] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[22] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[23] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[24] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature communications*, 9(1):1–6, 2018.

[25] Bobak Toussi Kiani, Seth Lloyd, and Reevu Maity. Learning unitaries by gradient descent. *arXiv preprint arXiv:2001.11897*, 2020.

[26] Roeland Wiersema, Cunlu Zhou, Yvette de Sereville, Juan Felipe Carrasquilla, Yong Baek Kim, and Henry Yuen. Exploring entanglement and optimization within the hamiltonian variational ansatz. *PRX Quantum*, 1(2):020319, 2020.

[27] M Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1–12, 2021.

[28] Samson Wang, Enrico Fontana, Marco Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J Coles. Noise-induced barren plateaus in variational quantum algorithms. *arXiv preprint arXiv:2007.14384*, 2020.

[29] Andrea Skolik, Jarrod R McClean, Masoud Mohseni, Patrick van der Smagt, and Martin Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3 (1):1–11, 2021.

[30] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. Entanglement induced barren plateaus. *arXiv preprint arXiv:2010.15968*, 2020.

[31] Sukin Sim, Peter D Johnson, and Alán Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12):1900070, 2019.

[32] Sukin Sim, Jhonathan Romero Fontalvo, Jérôme F Gonthier, and Alexander A Kunitsa. Adaptive pruning-based optimization of parameterized quantum circuits. *Quantum Science and Technology*, 2021.

[33] Xiaoyuan Liu, Anthony Angone, Ruslan Shaydulin, Ilya Safro, Yuri Alexeev, and Lukasz Cincio. Layer vqe: A variational approach for combinatorial optimization on noisy quantum computers. *arXiv preprint arXiv:2102.05566*, 2021.

[34] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. The effect of data encoding on the expressive power of variational quantum machine learning models. *arXiv preprint arXiv:2008.08605*, 2020.

[35] Openai gym wiki, cartpole v0, https://github.com/openai/gym/wiki/cartpole-v0.

[36] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

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

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[39] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[40] Long-Ji Lin. *Self-supervised Learning by Reinforcement and Artificial Neural Networks*. PhD thesis, PhD thesis, Carnegie Mellon University, School of Computer Science . . . , 1992.

[41] Francisco S Melo and M Isabel Ribeiro. Q-learning with linear function approximation. In *International Conference on Computational Learning Theory*, pages 308–322. Springer, 2007.

[42] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[43] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. A rigorous and robust quantum speed-up in supervised machine learning. *arXiv preprint arXiv:2010.02174*, 2020.

[44] Openai gym wiki, frozen lake v0, https://github.com/openai/gym/wiki/frozenlake-v0.

[45] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J Martinez, Jae Hyeon Yoo, Sergei V Isakov, Philip Massey, Murphy Yuezhen Niu, Ramin Halavati, Evan Peters, et al. Tensorflow quantum: A software framework for quantum machine learning. *arXiv preprint arXiv:2003.02989*, 2020.

[46] Cirq, https://quantumai.google/cirq.

Appendix A: Model hyperparameters

In the following, we give a detailed list of the hyperparameters used for both the quantum model and the neural networks shown in section IV. Table I shows the hyperparameters for the quantum models, table II shows those for the classical models.

The hyperparameters that we searched over for each model were the following (see explanations of each hyperparameter in corresponding table):

- *Frozen Lake v0*: number of layers in circuit, update model, update target model, η
- *Cart Pole v0, quantum model*: batch size, update model, update target model, η, train w_d, train w_o, η_{w_d}, η_{w_o}
- *Cart Pole v0, classical model*: number of units per layer, batch size, update model, update target model, η

	Frozen Lake v0	**Cart Pole v0**	**Hyperparameter explanation**
qubits	4	4	number of qubits in circuit
layers	5, 10, 15	5	number of layers as defined in fig. 1
γ	0.8	0.99	discount factor for Q-learning
train w_d	no	yes, no	train weights on the model output as defined in section III A
train w_o	no	yes, no	train weights on the model input as defined in section III B
η	0.001	0.001	model parameter learning rate
η_{w_d}		0.001	input weight learning rate
η_{w_o}		0.1	output weight learning rate
batch size	11	16	number of samples shown to optimizer at each update
ϵ_{init}	1	1	initial value for ϵ-greedy policy
ϵ_{dec}	0.99	0.99	decay of ϵ for ϵ-greedy policy
ϵ_{min}	0.01	0.01	minimal value of ϵ for ϵ-greedy policy
update model	5	10	steps in episode after which model is updated
update target model	10	30	steps in episode after which model parameters are copied to target model
size of replay memory	10000	10000	size of memory for experience replay
data re-uploading	no	yes, no	use data re-uploading as defined in section III A

TABLE I. Detailed hyperparameter values that were used in the models we present in fig. 3 and fig. 4.

	NN(4, 5)	**NN(9, 10)**	**Hyperparameter explanation**
γ	0.99	0.99	discount factor for Q-learning
η	0.01	0.01	model parameter learning rate
batch size	16	16	number of samples shown to optimizer at each update
ϵ_{init}	1	1	initial value for ϵ-greedy policy
ϵ_{dec}	0.99	0.99	decay of ϵ for ϵ-greedy policy
ϵ_{min}	0.01	0.01	minimal value of ϵ for ϵ-greedy policy
update model	5	5	steps in episode after which model is updated
update target model	10	10	steps in episode after which model parameters are copied to target model
size of replay memory	10000	10000	size of memory for experience replay

TABLE II. Detailed hyperparameter values for the classical models we present in and fig. 4. NN(4, 5) corresponds to the NN with two dense layers with 4 and 5 units each, NN(9, 10) to that with 9 and 10 units in each hidden layer.

Appendix B: Visualization of a learned Q-function

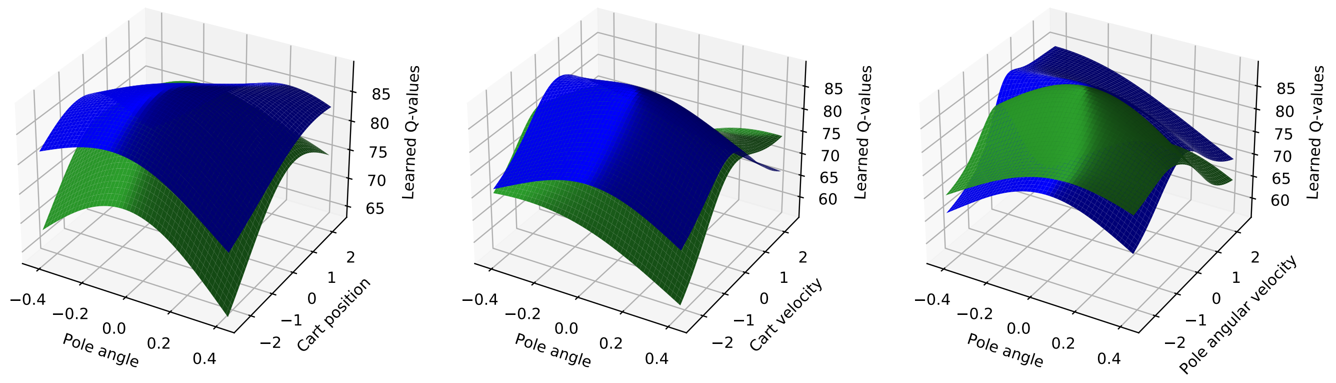


FIG. 5. Visualization of the approximate Q-function learned by a quantum Q-learning agent solving Cart Pole. Due to the 4 dimensions of the state space in Cart Pole, we represent the Q-values associated to the actions "left" (green) and "right" (blue) on 3 subspaces of the state space by fixing unrepresented dimensions to 0 in each plot. As opposed to the analogue values (i.e., unnormalized policy) learned by policy-gradient PQC agents in this environment [18], the approximate Q-values appear nicely-behaved, likely due to the stronger constraints that Q-learning has on well-performing function approximations.