Deep reinforcement learning for quantum Szilard engine optimization

Vegard B. Sørdal[*] and Joakim Bergli
Department of Physics, University of Oslo, 0316 Oslo, Norway

(Received 9 April 2019; published 16 October 2019)

Machine learning techniques based on artificial neural networks have been successfully applied to solve many problems in science. One of the most interesting domains of machine learning, reinforcement learning, has a natural applicability for optimization problems in physics. In this paper, we use deep reinforcement learning and chopped random basis optimization to solve an optimization problem based on the insertion of an off-center barrier in a quantum Szilard engine. We show that by using designed protocols for the time dependence of the barrier strength, we can achieve an equal splitting of the wave function (1/2 probability to find the particle on either side of the barrier) even for an asymmetric Szilard engine in such a way that no information is lost when measuring which side the particle is found. This implies that the asymmetric nonadiabatic Szilard engine can operate with the same efficiency as the traditional Szilard engine with adiabatic insertion of a central barrier. We compare the two optimization methods and demonstrate the advantage of reinforcement learning when it comes to constructing robust and noise-resistant protocols.

DOI: 10.1103/PhysRevA.100.042314

I. INTRODUCTION

Machine learning is becoming an essential tool for data analysis and optimization in a wide variety of scientific fields from molecular [1] and medical sciences [2] to astronomy [3]. One of the most exiting developments in machine learning comes from combining reinforcement learning [4] with deep neural networks [5]. Reinforcement learning (RL) differs from supervised and unsupervised learning and is based on letting an agent learn how to behave in a desired way by taking actions in an environment and observing the effect of the action on the environment. In order to define the optimal behavior of the agent, we give it feedback in the form of a reward based on the effect of its previous action. If the action changes the environment into a more desirable state, we give it a positive reward, whereas if it had negative consequences, we give it a negative reward. Recently, RL has enjoyed increasing popularity in quantum physics and has been used to explore the quantum speed limit [6,7], protect qubit systems from noise [8], design new photonic experiments [9], and many other applications [10–12]. For an excellent review of the application of machine learning in physics, see Ref. [13].

We use deep reinforcement learning (DRL), specifically deep-Q learning (DQL) [5], and a deep deterministic policy gradient (DDPG) [14] to solve an optimization problem based on the barrier insertion of an asymmetric (off-center insertion) quantum Szilard engine, which we will motivate it the following paragraphs. The goal is to find barrier insertion protocols that effectively achieve equal splitting of the wave function of a single-particle box (SPB). We compare the results from DRL with those obtained by using chopped random basis optimization [15], a more traditional optimization algorithm.

The Szilard engine is a classic example of a information processing system, which can convert one bit of Shannon information (obtained by a binary measurement) into an amount $k_BT \ln 2$ of useful work [16]. This is performed by inserting a barrier in the center of a SPB, performing a measurement to determine which side of the barrier the particle is found (giving one bit of Shannon information), and then letting the compartment the particle occupies isothermally expand into the empty one resulting in a work extraction of $k_BT \ln 2$. This work is not free, however, since the information obtained has to be stored in memory, which, subsequently, has to be deleted at an energy cost of $k_BT \ln 2$ according to Landauer's principle [17]. For a detailed discussion on the relationship between Szilard's engine and Landauer's principle, see Ref. [18]. Both work extractions from a Szilard engine and Landauer's principle have recently been experimentally confirmed [19–22].

For the quantum version of the Szilard engine [23], there are some subtle differences in the entropy flow during insertion, expansion, and removal of the barrier [24]. Moreover, the position of the particle is now described by a quantum wave function, which is divided into two parts when inserting the barrier. When adiabatically inserting a barrier in the center of a quantum SPB in its ground state, the wave function is split in half in such a way that each half becomes a new ground state in each compartment when the barrier strength goes to infinity. The probability to find the particle on either side of the barrier after insertion becomes 1/2. However, as long as there is an asymmetry in the insertion of the barrier, i.e., it is not put exactly in the center, the adiabatic theorem guarantees that the particle will be found in the larger compartment [25]. Since the initial state is the ground state and the adiabatic theorem implies the time evolved state will stay in its instantaneous eigenstate, the particle always ends up in the global ground state. The global ground state is found in the larger compartment since the energy is proportional to

[*]vegardbs@fys.uio.no

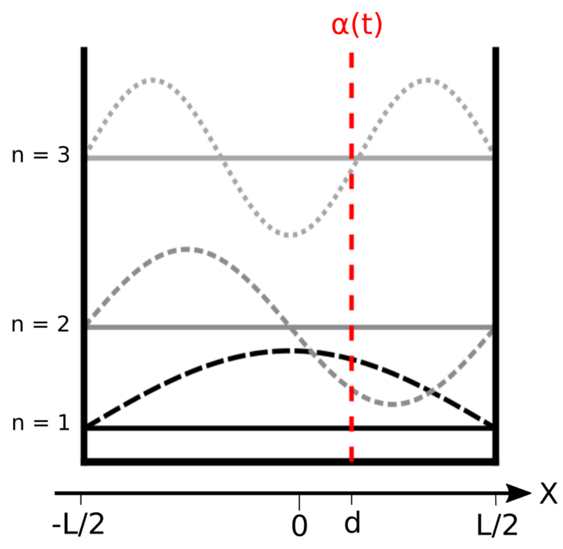

FIG. 1. Illustration of a single-particle box with total width L. The eigenfunctions and eigenenergies are shown for the initial state $\alpha(t) = 0$.

$L_{R(L)}^{-2}$, where $L_{R(L)}$ is the width of the compartment on the right(left) side of the barrier.

If we want to achieve equal probability on both sides of the barrier for asymmetric insertion, we have to insert the barrier nonadiabatically in such a way that we excite higher eigenstates. This will, in general, decrease the efficiency of the quantum Szilard engine since the measurement only determines which side the particle is found, not its exact eigenstate. However, there is one special way of obtaining exact splitting of the wave function without losing any information in the measurement for the asymmetric Szilard engine [26]: If we insert the barrier in such a way that the total wave function is a superposition of only the first and second eigenstates at the time of measurement, i.e., $|\Psi\rangle = (|\psi_1\rangle + |\psi_2\rangle)/\sqrt{2}$, the which-side measurement does not result in any information loss since the second eigenstate becomes the ground state of the smaller compartment. When one now measures which compartment the particle is in, one is certain that it is in the ground state of the respective compartment.

II. SINGLE-PARTICLE BOX

The SPB that constitutes a quantum-mechanical Szilard engine is defined by the potential $V(x) = 0$ for $x \in [-L/2, L/2]$, where L is the total width of the box and $V(x) = \infty$ elsewhere. The barrier is a δ-function potential inserted at $x = d \geqslant 0$. An illustration of the SPB is shown in Fig. 1 along with its three first eigenfunctions and eigenenergies before the barrier is inserted. If $d = 0$, the box is split symmetrically, i.e., the width of the left and right compartments is equal. However, for $d > 0$, the width of the left compartment becomes $L_L = L/2 + d$, whereas the width of the right compartment becomes $L_R = L/2 - d$. The time-dependent Hamiltonian of the insertion procedure is given by

$$\hat{H}(t) = -\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \alpha(t)\delta(x-d), \quad (1)$$

where $\alpha(t)$ is the strength of the barrier at time t and m is the mass of the particle. For the rest of this article, we set $\hbar = m = 1$. The total wave-function $|\Psi(t)\rangle$ can be expressed as a linear combination of the instantaneous eigenfunctions,

$$|\Psi(t)\rangle = \sum_n c_n(t)\,|\psi_n(t)\rangle\, e^{i\theta_n(t)}, \qquad \theta_n = -\frac{1}{\hbar}\int_0^t E_n(t')dt', \quad (2)$$

and the goal is to construct a protocol where $E_n(t)$'s are the instantaneous eigenenergies when the barrier strength is $\alpha(t)$, $|\psi_n(t)\rangle$'s are the instantaneous eigenfunctions, and $c_n(t)$'s are complex coefficients. The initial state is, therefore, given by $|c_1(0)|^2 = 1$, and the goal is to construct a protocol $\alpha(t)$, which brings us to a final state where $|c_1(T)|^2 = |c_2(T)|^2 = 1/2$, where T is the duration of the protocol. More details on how the instantaneous eigenstates are calculated and how the time evolution of the total wave function is numerically solved is given in Ref. [26].

We see now that the method of only exciting the first two eigenstates also results in the minimum work cost of inserting the barrier. The work in an isolated system evolving from time $t = 0$ to $t = T$ is given by

$$W = \Delta H = \sum_{n=1}^{\infty} E_n(T)|c_n(T)|^2 - E_1(0)|c_1(0)|^2. \quad (3)$$

If we require that the barrier strength $\alpha(T)$ at $t = T$ to be high enough to prevent any leakage between the compartments, then the energy eigenstates are the same for all protocols since they quickly converge to a fixed value as $\alpha(T) \to \infty$. Therefore, the work is determined only by the set $\{c_n(T)\}$, and since the energy is strictly increasing as a function of n, the minimum work required for an equal distribution is when $|c_1(T)|^2 = |c_2(T)|^2 = 1/2$.

If we want to get energy from a heat bath in this system as is typically performed for Szilard engines, the coupling between the system and the heat bath will result in a thermal density matrix,

$$\rho_B(T) = \frac{e^{-\beta E_n(T)}}{\sum_n e^{-\beta E_n(T)}}, \quad (4)$$

where β is the inverse temperature of the heat bath. This coupling is followed by an isothermal expansion of the barrier, extracting heat from the environment to perform work. If the isolated system is excited to higher states before coupling, then the energy transferred from the heat bath to the system upon coupling would be lower than the one transferred if only the two lowest eigenstates were excited (because the energy of the isolated system in the former case is closer to the energy of the thermal density matrix than for the latter case). This results in an overall decrease in the per cycle heat-to-energy conversion.

Our goal is to split the wave function of a single-particle box in the ground state by inserting a barrier off center in such a way that the first and second eigenstates have equal occupation probability of 1/2 and the probabilities to find the particle in all higher states are as close to zero as possible. However, finding a protocol for the barrier insertion which will achieve this goal is nontrivial since it will have

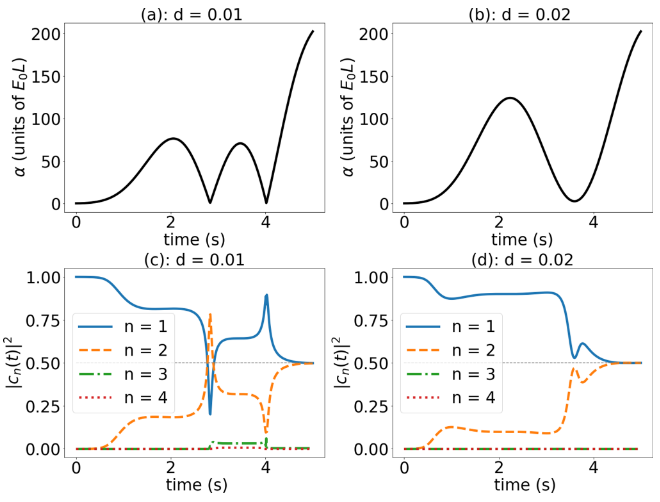

FIG. 2. Results from the CRAB optimization for $d = 0.01$ and $d = 0.02$. In (a) and (b), we show the protocols $\alpha(t)$, whereas in (c) and (d), we show the time evolution of $|c_n(t)|^2$. We see that the protocol in (b) gives negligible excitations to states $n > 2$ throughout its duration. However, the protocol in (a) excites the third eigenstate during the first discontinuity in $\dot{\alpha}(t)$ right before $t = 3$, but this excitation is depleted during the second discontinuity around $t = 4$.

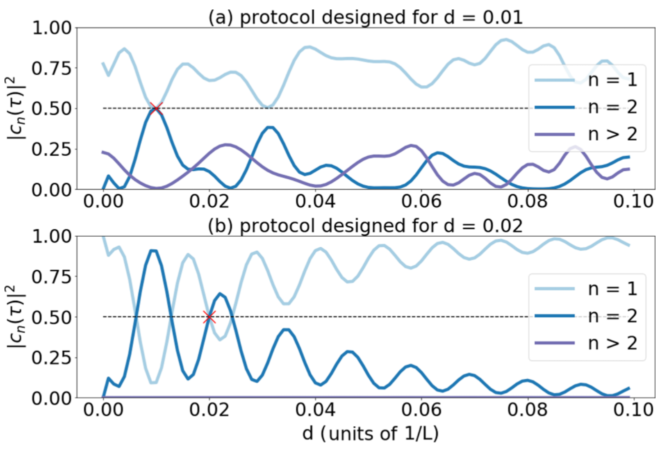

FIG. 3. Plot showing how the protocols designed for two specific asymmetries perform on other asymmetries. In (a), we show the results for the protocol designed for $d = 0.01$, whereas, in (b), we show the one designed for $d = 0.02$. The light blue and the blue lines show the occupation at $t = T$ for the first and second eigenstates, respectively, whereas the purple line shows the occupation of all eigenstates higher than the second, i.e., the unwanted excitations. The black dashed lines show the target $|c_n(T)|^2 = 0.5$, and the red crosses shows the asymmetry trained on.

to take advantage of complicated interference between the time-dependent eigenstates.

III. CHOPPED RANDOM BASIS OPTIMIZATION

We use chopped random-basis (CRAB) optimization [15] to find protocols $\alpha(t)$ that split the wave function in two equal halves for asymmetric barrier insertion in a quantum box. In CRAB optimization, we expand the protocol in a complete basis (the Fourier series in our case), in the following way:

$$\alpha(t) = \alpha_0(t)\left[1 + \lambda(t)\sum_{n=1}^{N_c} A_n\cos(\omega_n t) + B_n\sin(\omega_n t)\right]. \quad (5)$$

Here, $\alpha_0(t)$ is an initial guess for the optimal protocol, $\lambda(t)$ is a regularization function used to implement boundary conditions, and $\{A_n, B_n, \omega_n\}$ is a set of Fourier coefficients. We set the cutoff to $N_c = 10$, meaning we have a total of 15 free parameters. The Fourier coefficients are optimized to minimize the cost function,

$$C(\{A_n, B_n, \omega_n\}) = \sum_{n=1}^{2}[|c_n(T)|^2 - 0.5]^2. \quad (6)$$

We use a linear function for $\alpha_0(t)$ and fix the boundary conditions to be $\alpha_0(0) = 0$ and $\alpha_0(T) = 200E_0L$ [where $E_0 = \pi^2/2$ is the ground state at $\alpha(0) = 0$] and choose $\lambda(t) = \sin(\pi t/T)$. To minimize Eq. (6), one can use a gradient-free method, such as the Nelder-Mead [27] or the Powell method [28]. Using the Nelder-Mead method, we are able to almost exactly split the wave function in half, and results for $d = 0.01$ and $d = 0.02$ are shown in Fig. 2. In these examples, we obtained $|c_1(T)|^2 = 0.4986$, $|c_2(T)|^2 = 0.4979$, and $\sum_{n>2}|c_n(T)| \simeq 10^{-3}$ for $d = 0.01$, whereas, for $d = 0.02$, we got $|c_1(T)|^2 = 0.5001$, $|c_2(T)|^2 = 0.4999$, and $\sum_{n>2}|c_n(T)|^2 \simeq 10^{-5}$. In Figs. 2(a) and 2(b), we show example protocols for $d = 0.01$ and $d = 0.02$, respectively, whereas in Figs. 2(c) and 2(d), we show the time evolution of the probability to be in a given eigenstate $|c_n(t)|^2$.

The protocols obtained by CRAB are designed to split the wave function in two for a given asymmetry. They work extremely well for the asymmetry they were designed for. However, the protocols generalize poorly to other asymmetries as shown in Fig. 3. There we plot $|c_n(T)|^2$ as a function of the asymmetry d using the protocol designed for $d = 0.01$ and $d = 0.02$. We see that the performance of a protocol designed for a specific asymmetry dramatically reduces if it is applied to single-particle boxes of different asymmetries. An interesting feature is seen in Fig. 3(b) where the protocol designed for $d = 0.02$ achieves exact splitting for asymmetries other than the one that was used for training. However, even this protocol has bad performance in the regions between these points of exact splitting, so it would not be useful unless one knows the exact asymmetry of the single-particle box.

Real systems are noisy, and one may not know the exact asymmetry of the box, either due to a fluctuating Hamiltonian or due to limits of experimental precision. We, therefore, want to find protocols that perform well on a small interval of asymmetries. This can be performed by changing the cost function to an average of Eq. (6) for a set of asymmetries. In Fig. 4, we show example results obtained when using CRAB on three, five, and ten different asymmetries. The red crosses indicates the set of asymmetries that was trained on. At three asymmetries, Figs. 4(a) and 4(b), we obtain decent results, however, there are gaps in the areas between the specific points where we trained where the protocol performs poorly. When training on five asymmetries, Figs. 4(b) and 4(c), the protocol still performs better in the target area than the surroundings, however, the difference between $|c_1(T)|^2$ and $|c_2(T)|^2$ is quite large, and there are a lot of higher-order

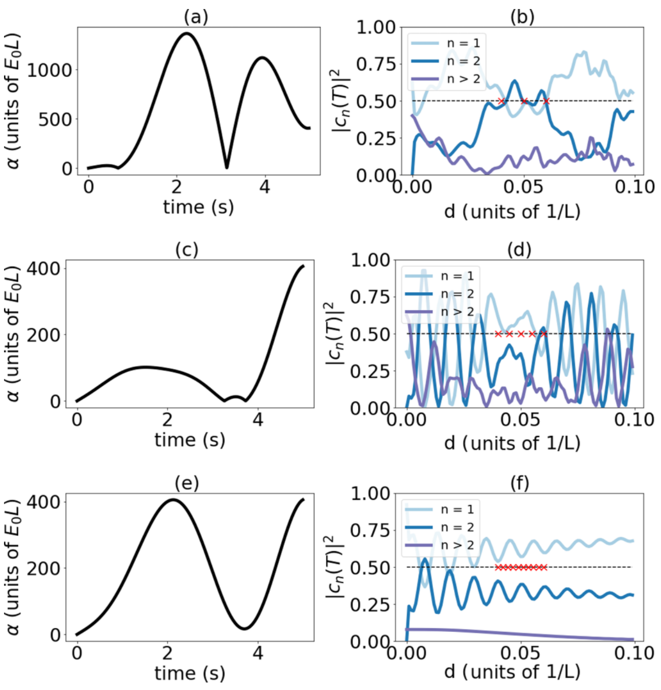

FIG. 4. Example protocols and their final distribution of $|c_n(T)|^2$ for a range of asymmetries. (a) and (b) show the results when training on three asymmetries, (c) and (d) show the same for five asymmetries, whereas (e) and (f) are trained on ten asymmetries. The specific asymmetries trained on is indicated by the red crosses. The training time was 11 h (a) and (b), 13 h (c) and (d), and 38 h (e) and (f). These protocols were obtained using $\alpha_0(T) = 400E_0L$, and all other parameters are the same as the single asymmetry case.

excitations. The worst protocol was obtained when training on ten asymmetries, shown in Figs. 4(e) and 4(f). The CRAB algorithm failed to converge to a good result when the number of asymmetries increased. This is most likely because the cost function becomes rugged and filled with local minima. CRAB is just one of many optimization methods that could be used to solve our problem. A set of recent exciting methods that has been used in physics before, but could become more widespread in the future, comes from deep learning. Specifically, deep reinforcement learning seems like a good candidate for optimization problems, so, in the following sections, we will introduce some key algorithms and apply them to our problem.

IV. DEEP Q LEARNING

We now give a short review of the DQL algorithm introduced in Ref. [5]. In the next section, we will show how this general algorithm is adapted to our problem. A schematic of the basic reinforcement learning protocol is shown in Fig. 5. At time t, the environment is in a given state s_t. The agent performs an action a_t which induces a state change in the environment from s_t to s_{t+1}. The agent then receives an observation of the new state of the environment s_{t+1}. After taking an action, the agent receives a reward $r_t = r(s_t, a_t, s_{t+1})$. The reward function $r(s_t, a_t, s_{t+1})$ is designed by us, according to what goal we want the agent to achieve.

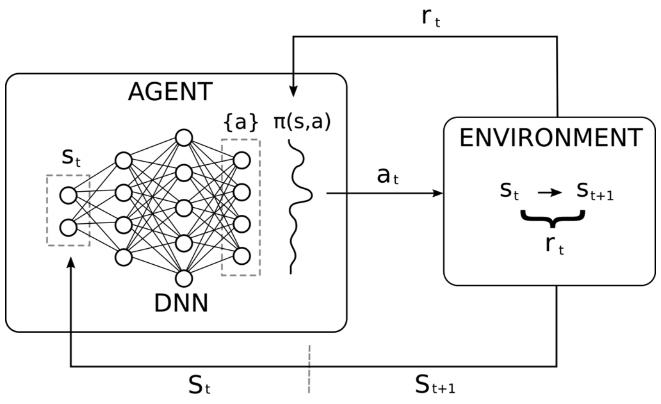

FIG. 5. Schematic showing the basic setup of deep Q learning. The current state of the system s_t is fed as input nodes into a deep neural network (DNN). The output nodes are the set of all possible actions $\{a\}$, and their values are the estimated Q value for the given state-action pair. The policy $\pi(s, a)$ is given by the action node with the highest output value or by a random action if the agent is exploring. The action determined by the policy is performed in the environment, inducing a state change from $s_t \rightarrow s_{t+1}$. Associated with this state change, a reward r_t is given, which is used to determine how good the given action was in this state. This reward is fed back into the DNN and used to update its weights according to Eq. (10). Schematic adapted from Ref. [29].

The behavior of the agent is determined by its policy $\pi(a_t|s_t)$, which is the probability of taking the action a_t in given the observation s_t. If the agent is in state s_t, the Q function (quality function) $Q_\pi(s_t, a_t)$ gives the expected cumulative reward given that the action a_t is performed and the policy π is followed for all proceeding states,

$$\begin{aligned} Q(s_t, a_t) &= E_{s_{t+1}}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots |s_t, a_t, \pi] \\ &= E_{s_{t+1}}[r_t + \gamma Q(s_{t+1}, a_{t+1})|s_t, a_t, \pi]. \end{aligned} \tag{7}$$

Here, $\gamma \leqslant 1$ is a discount parameter, which determines how much the agent values the immediate reward compared to the future reward. If $\gamma < 1$, the agent will value the future reward less than the immediate reward, which is useful for learning in stochastic environments where the future is more uncertain. The optimal Q-function $Q^*_\pi(s_t, a_t)$ is the maximum expected cumulative reward obtained by taking the action a_t in state s_t and then acting optimally thereafter, and it is shown to obey the Bellman optimality equation [4],

$$Q^*_\pi(s_t, a_t) = E_{s_{t+1}}\Big[r_t + \gamma \max_{a_{t+1}} Q^*_\pi(s_{t+1}, a_{t+1})|s_t, a_t\Big]. \tag{8}$$

If we have $Q^*_\pi(s, a)$ for all possible state-action pairs, it is clear that we can find the optimal policy π^* by choosing $a_t = \arg\max_{a'} Q^*_\pi(s_t, a')$, i.e., following the policy:

$$\pi^*(a_t|s_t) = \arg\max_{a'} Q^*_\pi(s_t, a'). \tag{9}$$

The key idea introduced in Ref. [5], is to estimate the optimal Q function using a neural network $Q^*_\pi(s, a) \simeq Q^*_\pi(s, a, \theta)$, where θ is the weights and biases of the neural network. This neural network is called a deep-Q network (DQN) and is updated by performing gradient ascent on the mean-squared error of the current predicted $Q^*_\pi(s, a, \theta)$, whereas using the Bellman equation as the target. The loss function for the DQN is, therefore,

$$L(\theta) = E_{s_{t+1}}\{[Q^*_\pi(s_t, a_t, \theta) - y_t]^2\}, \quad (10)$$

where

$$y_t = r_t + \gamma \max_{a_{t+1}} Q^*_\pi(s_{t+1}, a_{t+1}, \theta). \quad (11)$$

To create the neural network, we used TENSORFLOW's implementation of the KERAS API [30,31] with Adam [32] as the optimizer. The network consists of three hidden layers with 24, 48, and 24 neurons, respectively, as well as 2 input neurons and 20 output neurons. When the network is initialized, its predictions for the optimal Q^*_π values are, of course, totally wrong. So, if we always chose the actions that maximize the current predicted Q^*_π values, the agent would not learn anything. We need to let the agent explore the state-action space by randomly performing actions. A typical exploration policy is the ϵ-greedy policy. The agent chooses random actions with probability ϵ or the ones with the highest Q^*_π value (greedily) with probability $1 - \epsilon$. As time goes and the agent explores more of the environment, ϵ is decreased so that it focuses more on the areas of the state-action space with higher Q^*_π values by taking deterministic actions. Typically, we start by taking completely random actions $\epsilon = 1$ and let ϵ converge to some finite number $\epsilon \sim 0.05$ so that there is always some exploration going on. As seen in Eq. (10), a single update of the network weights requires the following input: The current state s_t, the action chosen a_t, the immediate reward r_t, and the next state s_{t+1}. We call this tuple $e_t = (s_t, a_t, r_t, s_{t+1})$ that the network trains on an experience. Instead of training on consecutive experiences, we store them all in memory $M_N = \{e_0, e_1, \dots, e_N\}$ and then train on randomly drawn batches of samples from the memory. The memory has a finite capacity, and new experiences replace older ones when the memory is full. There are three main advantages of training on the memory: It is data efficient since a single experience can be drawn many times. Only training on consecutive experiences is inefficient since the network tends to forget previous experiences by overwriting them with new experiences. The time correlation of consecutive experiences means that the network update due to the current experience determines what the next experience will be so training can be dominated by experiences from a certain area in the state-action space. Finally, we see that, in Eq. (10), the current weights of the network determine both the target Q^*_π value and the predicted Q^*_π value from the Bellman equation. Thus, every network update changes the target Q^*_π value that we are trying to reach and makes it hard for the network weights to converge. A simple way to circumvent this problem is to use two neural networks, one for the target Q^*_π-value (θ^-), and one for the current Q^*_π-value (θ). The target network is softly updated during training according to $\theta^- \leftarrow \theta^-(1-\tau) + \theta\tau$, where τ is a hyperparameter that determines how close the two networks are in the network parameter space.

V. DQL RESULTS

For our system, we defined the state to be a tuple of the strength of the δ barrier and the time t, i.e., $S = \{\alpha(t), t\}$. The available actions are a set of $\dot\alpha(t)$, given by

$$A = \{\dot\alpha^\pm_n(t) = \pm 10 \times 2^n \quad \text{for } n = 1, 2, \dots, 10\}.$$

The initial state is $S = \{\alpha(t) = 0,\ t = 0\}$, and the goal is to reach a state where $|c_0|^2 = |c_1|^2 = 1/2$ at the end of the protocol $t = T$. A sequence of selected actions from time $t = 0$ to $t = T$ defines a protocol $\alpha(t)$. The number of times the agent chooses an action per protocol is given by N_t, and the time step is, therefore, $dt = T/N_t$. The environment that the agent acts in is the quantum-mechanical SPB with initial states $|c_1|^2 = 1$ and $|c_{n>1}|^2 = 0$ and time evolution given by the Schrödinger equation $i\partial_t |\psi(t)\rangle = H(t)|\psi(t)\rangle$, which we solve as in Ref. [26]. The sequential process for one episode is then:

(1) The initial state is s_0: $(\alpha_0 = 0,\ t = 0)$.

(2) The agent chooses an action based on s_0, e.g., $a_0 = \dot\alpha^+_3$.

(3) The next state is then s_1: $(\alpha_0 + \dot\alpha^+_3 dt, t + dt)$.

(4) Repeat (2) $\to$ (3) for $s_1, s_2, \dots$ until $t = T$.

(5) Solve the Schrödinger equation for the given protocol (set of all states $\{s_n, t_n\}$), and calculate reward. Repeat from (1) until a maximum number of episodes is reached.

The reward function we used is defined by

$$r(t) = \begin{cases} 0, & \text{if } t < T \text{ and } \alpha \in [0, \alpha_{\max}], \\ -10, & \text{if } t < T \text{ and } \alpha \notin [0, \alpha_{\max}], \\ 100 \exp\left(-\sum_{n=1}^{2} \frac{[|c_n(T)|^2 - 0.5]^2}{\sigma}\right), & \text{if } t = T, \end{cases} \quad (12)$$

where σ determines how sharp we want the reward distribution to be. We use a Gaussian function instead of the quadratic one we used for CRAB such that the terminal state reward is always positive. This is performed because it makes it easier to infer the agents behavior when looking at the reward as a function of episode number. To motivate the agent to be precise when splitting the wave function, we should give it a very high reward when it gets close to exact splitting, thus, we include a $\sigma < 1$ to make the distribution sharper. If it gets almost the same reward for a worse result, which is easier to obtain, it will prefer the easier option.

If the agent chooses actions such that $\alpha(t) < 0$, we give it a punishment of -10 and set $\alpha(t) = 0$, and for actions that would give $\alpha(t) > \alpha_{\max}$, we punish and set $\alpha(t) = \alpha_{\max}$. We do this to keep the state space bounded. The space of possible protocols grows exponentially with dt^{-1}, so it is impractical to set dt so small that we get approximately continuous $\dot\alpha(t)$. The accuracy of our numerical solution of the quantum time evolution decreases if we have discontinuous $\dot\alpha(t)$, so, to circumvent this problem, we use a cubic spline to interpolate the final protocol before calculating the reward.

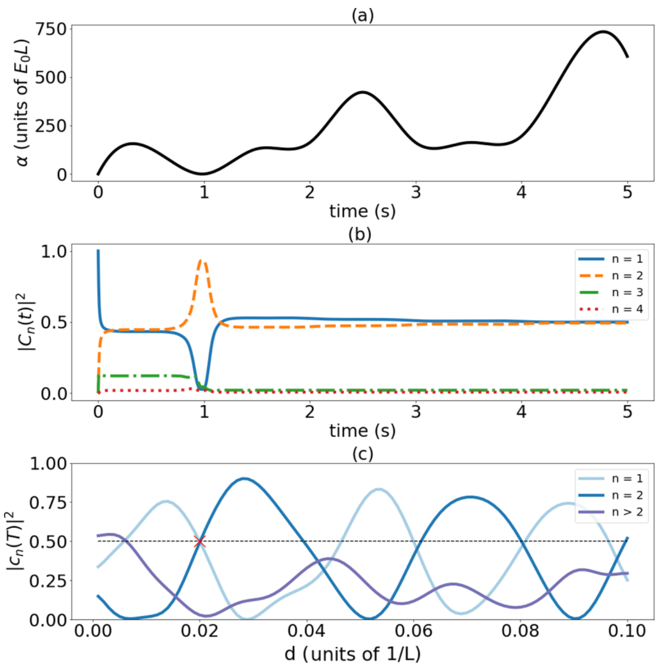

FIG. 6. Results from DQL when training on a single asymmetry $\epsilon = 0.02$. In (a), we show the protocol $\alpha(t)$, whereas in (b), we show the time evolution of $|c_n(t)|^2$ for the asymmetry we trained on. Similar to the protocol in Fig. 2(a), there is a good amount of excitations for the third eigenstate in the very beginning of the protocol, which is then depleted around $t = 1$ s. In (c), we show how the protocol generalizes to other asymmetries by plotting the distribution $|c_n(T)|^2$ at $t = T$ for asymmetries in the range of $d \in [0.01, 0.1]$. The parameters of this protocol were $T = 5$ s, $N_t = 10$, $\alpha_{\max} = 800E_0L$, and $\sigma = 0.05$.

In Fig. 6, we show an example protocol learned by the DQL agent and the corresponding time evolution of $|c_n(t)|^2$ when training on a single asymmetry ($\epsilon = 0.02$) for 10 000 episodes. The final distribution was $|c_1(T)|^2 = 0.4996$, $|c_2(T)|^2 = 0.4935$, and with higher excitations, $\sum_{n>2} |c_n|^2 \simeq 10^{-2}$. The results, when training on a single asymmetry, tended to be worse for DQL than for direct CRAB optimization. There are many ways to improve the results obtained by DQL; we can add actions to or change the action space, we can train for a longer time or increase the number of actions per episode N_t. Alternatively, one could implement algorithms similar to DQL that can perform actions in a continuous action space, such as the DDPG [14]. However, most of these changes would also increase the necessary training time.

One of the main benefits of DQL is that it is a model-free algorithm, so the task of generalizing a protocol for a range of asymmetries is easily achieved. One only has to let the agent train on random samples of the set of asymmetries one wants the protocol to be optimized to, which is essentially the same as training with a noisy Hamiltonian [7,8]. Since the agent tries to maximize the expected cumulative reward, this added stochasticity is no hindrance. How much the agent values a given state-action pair is averaged over the random samples from the memory, which is proportionally filled with the number of asymmetries we train on.

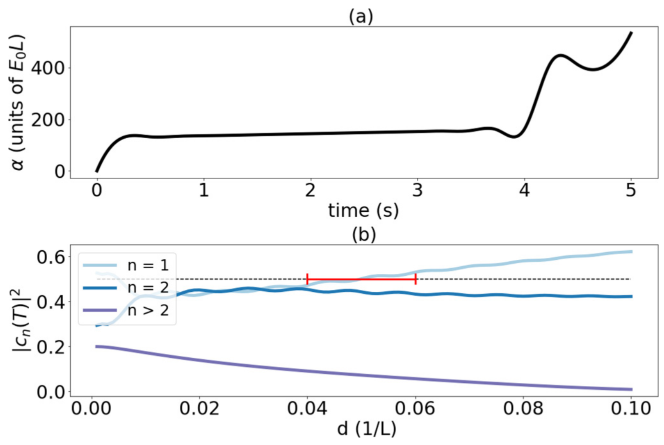

FIG. 7. Results from DQL when training on ten different asymmetries in the range of $d \in [0.04, 0.06]$. In (a), we show the protocol obtained, whereas in (b), we show $|c_n(T)|^2$ all asymmetries up to $d = 0.1$ where the red bar indicates the range of asymmetries we trained on. When compared to Fig. 3, we see that the protocol performs much better overall than the ones designed for one specific asymmetry, particularly, in the range we trained on. The parameters of this protocol were $T = 5$ s, $N_t = 20$, $\alpha_{\max} = 800E_0L$, and $\sigma = 0.05$.

As an example, say one could determine the asymmetry with a given accuracy $d = 0.05 \pm 0.01$. An example protocol that was obtained when training on multiple asymmetries (ten equally spaced samples in the range of $d \in [0.04, 0.06]$) is shown in Fig. 7. As seen in Fig. 7(b), this protocol performs better on the full range of asymmetries than the ones designed for a single asymmetry, shown in Fig. 3. The excitation to states higher than the two first eigenstates is largest for small asymmetries. This is due to the fact that, when $d \to 0$, the wall is inserted close to the central node of the second eigenstate and the central antinode of the third eigenstate as shown in Fig. 1. Therefore, excitations to the second eigenstate become less likely, whereas the opposite is true for excitations to the third eigenstate. Since this is an intrinsic property of the system, it is impossible to find protocols that avoid excitations for $d \to 0$. For $d = 0$, the ground state of the left and right compartments constitute a doubly degenerate global ground state, and to achieve an equal splitting of the wave function, one has to insert the barrier adiabatically [26].

From Figs. 7 and 8, we can infer that the way the agent created a robust protocol (for this example) was to choose a large initial value of the action $\dot{\alpha}(t)$ (from 0 to 1 s) in such a way that the first and second eigenstates were quickly almost evenly occupied. At later times, the protocol slightly modifies the occupation in such a way that the average reward of all asymmetries trained on increases by a small amount. When training on multiple asymmetries, the type of protocol shown in Fig. 6 no longer works because it takes advantage of the exact timing of complicated interference between the eigenstates, which strongly depends on the exact asymmetry. It is unlikely that there exists special protocols that simultaneously achieve this for many asymmetries, therefore, the most important feature of the robust protocols is the early and rapid increase to a high value of $\alpha(t)$. If they exist, it is reasonable

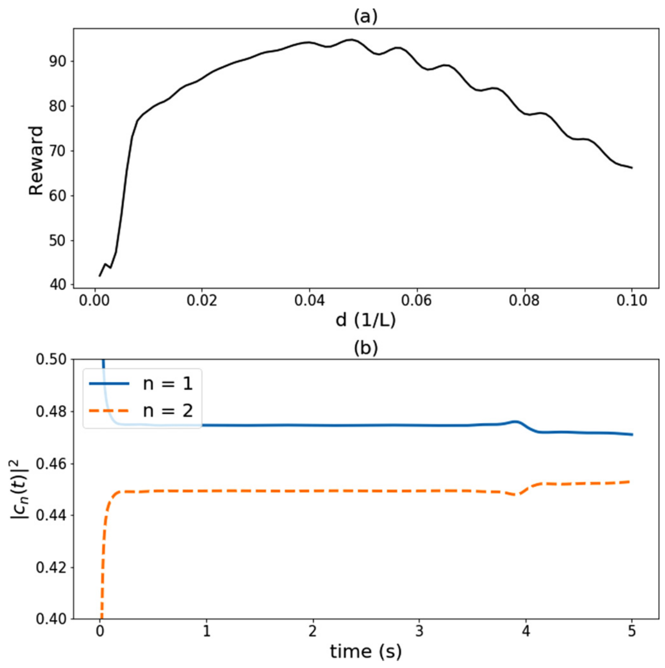

FIG. 8. Plot of the (a) reward as a function of asymmetry d and the time evolution of (b) $|c_n(t)|^2$ for $d = 0.548$. These results are obtained by applying the protocol from Fig. 7. We see that the reward is maximized around the training asymmetries.

to expect that they can only be found for small time steps, however, even when decreasing the time step down to $dt = 0.01$ using $T = 1$ s and $N_t = 100$, the obtained protocols were qualitatively similar to the ones shown.

In Fig. 9, we see the total cumulative reward received per episode in a scatter plot as well as a running average. We see that, in the early episodes, where the agent mostly performs random actions, there are many episodes with a negative cumulative reward. This is because there is an equal probability that the agent chooses negative and positive $\dot{\alpha}$'s, and since the initial state is $\alpha(t = 0) = 0$, there is a high

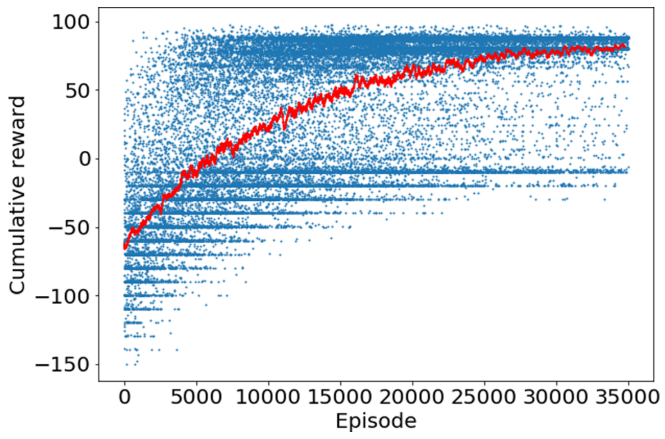

FIG. 9. Scatter plot of the reward received per episode when training on multiple asymmetries, shown in blue dots and a running average shown in red. The probability to take random actions is gradually reduced with the number of episodes, leading to a final protocol which the agent determines to be the best.

probability that the agent chooses actions which give $\alpha(t) < 0$, resulting in a punishment of -10 every time. In this early stage, the agent explores and learns about its environment. As the probability to take random actions decreases (according to the ϵ-greedy protocol) with each episode, the agent takes more deterministic actions based on its experience, and the reward per episode increases steadily. The stochasticity observed in the rewards for final episodes is due to a finite final exploration rate $\epsilon = 0.05$. We obtain the final protocol after training by setting $\epsilon = 0$ and let the agent act deterministically. The efficiency of the protocol obtained by training on a range of asymmetries can be increased by implementing the same changes as for the one designed for a single asymmetry.

VI. DEEP DETERMINISTIC POLICY GRADIENT

Our set of possible actions for the DQL algorithm is somewhat arbitrarily chosen. For our specific control problem, there are infinitely many protocols that can achieve our goal, so the exact set chosen is not critically important. However, the performance of the algorithm depends on this choice, and the optimal protocols we find can always be defined by some subset of the total action space. That is, not all actions are used for the optimal protocol, so we could retroactively reduce the action space after learning which actions were needed. For many control problems in physics, it is more natural to let the action values be drawn from a continuous set on some interval $A \in [a_{\min}, a_{\max}]$. For DQL, this is not possible since the optimal policy $\pi^*(a_t|s_t)$ comes from taking the maximum argument of a finite-dimensional $Q^*(s_t, a_t)$.

When the action space is continuous, the optimal Q-function $Q^*(s, a)$ is assumed to be differentiable with respect to the action a. In the deep deterministic policy gradient [14], the goal is to find a deterministic policy $\mu(s)$, which gives us the optimal action to take for any state $a^* = \mu(s)$. This deterministic policy is approximated by another neural network $\mu(s) \simeq \mu(s, \phi)$, where ϕ's are the parameters of the network. The Q function is as in DQL also approximated by a neural network, and the essence of introducing the deterministic policy is to replace the largest Q value for a state-action pair in the following way:

$$\arg\max_{a'} Q^*_\pi(s_t, a', \theta) \rightarrow Q^*[s_{t+1}, \mu(s_{t+1}, \phi), \theta]. \tag{13}$$

The Q network is updated in the same way as in DQL by using the Bellman equation, but instead of Eq. (11), the target for the loss function now becomes

$$y_t = r_t + \gamma Q^*[s_{t+1}, \mu(s_{t+1}, \phi), \theta]. \tag{14}$$

As for the policy network, it was shown in Ref. [33] that its weights can be updated in proportion to the gradient of the Q function,

$$\phi_{k+1} = \phi_k + \lambda E_{s\in B}\{\nabla_\phi Q^*[s, \mu(s, \phi), \theta]\}, \tag{15}$$

where λ is the learning rate, which determines the step size of the gradient ascent. Since the gradient will, in general, move the weights in different directions for different states, an average over a batch of experiences is taken. By applying the chain rule to Eq. (15), we can decompose it into a product of the gradient of the policy with respect to its network weight,

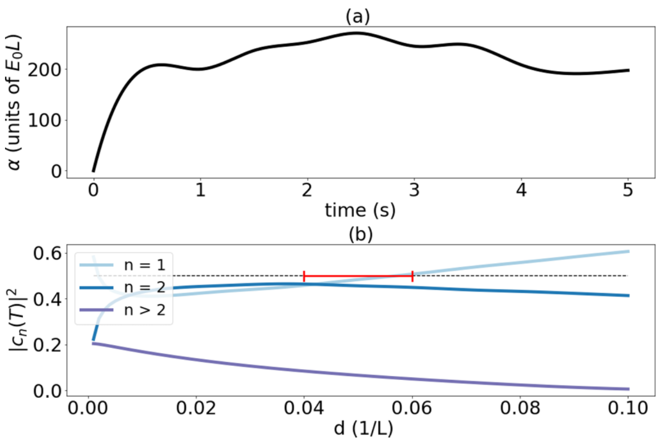

FIG. 10. Results from DDPG when training on ten different asymmetries in the range of $d \in [0.04, 0.06]$. In (a), we show the protocol itself, whereas in (b), we show how the protocol performs on a range of asymmetries $d \in [0, 0.1]$. The red bar marks the range we trained on. The parameters of this protocol were $T = 5$ s, $N_t = 20$, $\alpha_{\max} = 800E_0L$, and $\sigma = 0.05$, and we trained for 20 000 episodes.

and the gradient of the Q function with respect to the actions,

$$\nabla_\phi Q^*[s, \mu(s, \phi), \theta] = \nabla_\phi \mu(s, \phi) \nabla_a Q^*(s, a, \phi)|_{a=\mu(s,\phi)}. \tag{16}$$

Exploration in DDPG is driven by adding noise to the policy, sampled from some distribution N_t suited to the environment, which is annealed over time,

$$\mu'(s_t) = \mu(s_t, \phi) + N_t. \tag{17}$$

We use a Gaussian white-noise process and annealed its standard deviation from $\sigma_N = 0.3$ to $\sigma_N = 10^{-4}$ over the course of the training. DPPG is called an actor-critic model, and the sense is that the policy is an actor, taking actions in an environment, and the Q function acts as a critic, determining how good the actions where and feeding the result back to the actor.

For the DPPG algorithm, we used an adapted implementation from Keras-RL [34], which includes the same modifications we used for DQL; i.e., experience replay and different networks for the target and current Q function and policy. The policy network takes as input the same state tuple as for DQL $S = \{\alpha(t), t\}$, which is connected to three hidden layers with the same architecture as DQL; 24, 48, and 24 neurons, respectively, and outputs a single value, the action $\dot{\alpha}(t)$. The Q-function network takes as input the action value suggested by the policy as well as the state $S = \{\alpha(t), t\}$ again connected to three hidden layers with the same architecture as DQL and outputs a value which is its estimation of the optimal Q value of the state-action pair. The output actions forming the policy network are clipped at $|\dot{\alpha}(t)| \leqslant 1000E_0L/s$, and we use the same reward function as for DQL. In Fig. 10(a), we show a protocol obtained from DDPG when training on ten asymmetries in the range of $d \in [0.04, 0.06]$, and in Fig. 10(b), the performance of the protocol on a range of asymmetries from $d \in [0, 0.1]$. As expected, the best results are obtained for the range of asymmetries we trained on, indicated by a red bar. A rigorous comparison between DQL and DPPG is difficult, partly due to the large amount of hyperparameter tweaking needed to optimize each algorithm but largely due to the arbitrary choice of discrete action values for DQL: For our example problem, there is no natural set of available actions to choose. As mentioned earlier, the performance of DQL for our problem depends on the set of actions chosen, and therefore, a fair comparison of the algorithms is complicated. The choice between discrete and continuous-action algorithms has to be taken based on the specific problem one wants to solve. For our SPB problem, there are infinitely many good solutions, and since we interpolate the protocol at the end of each episode, both DQL and DPPG are well suited.

We used a 3.40-GHz CPU, and the training time for the most resource-intensive computation (the protocol in Fig. 7) was about 48 h, so increased training time is something that more advanced computation systems can handle. The most computationally intensive part of the training by a large margin was solving the Schrödinger equation after each episode. As for the hyperparameters of the neural networks, we used a learning rate $\lambda = 10^{-3}$, target network update every $\tau = 10^{-3}$ time steps, and a replay memory size between 10% and 50% of the total number of experiences. The ε-greedy exploration policy was a linear decrease from $\varepsilon = 1$ to $\varepsilon = 0.05$.

VII. DISCUSSION AND SUMMARY

We have used CRAB optimization and deep reinforcement learning to construct protocols $\alpha(t)$ for the time-dependent strength of a barrier inserted asymmetrically in a single-particle box in such a way that the wave function is split in two equal halves. These results imply that the asymmetric quantum Szilard engine can reach the same efficiency in information-to-work conversion as the symmetric one. Using CRAB optimization, the protocols we obtain perform very well for the specific asymmetry we optimize for. However, the algorithm generalizes poorly when simultaneously training on a set of asymmetries to simulate a noisy environment. Although more time consuming than CRAB optimization, we can also use DRL to find high performing protocols when training on single asymmetries. One of the biggest strengths of reinforcement learning-based techniques is the possibility to perform robust and noise-resistant optimization. When training on a range of different asymmetries simultaneously, DRL can be used to find the protocols that perform best on the average of all the asymmetries sampled. Both DQL and DDPG were able to find good protocols for our example SPB problem, but, in general, the choice between discrete and continuous-action algorithms has to be made on the basis of what specific problem one wants to solve. The advantage of using reinforcement learning for quantum control is multifaceted: Having model-free algorithms makes it simple to change the optimization criterion to make the agent solve different problems within the same environment, one only has to change the reward function to suit the new goal. Furthermore, since the agent is not tailored to any specific environment, it can easily be adopted to work in entirely different systems (e.g., we can use the agents constructed here to perform state transfer in qubit systems [6]). Finally, the stochastic nature of the agents learning procedure is advantageous when one wants to perform robust optimization which can perform well with noise. These points all suggest that reinforcement learning can become a useful tool in physics.

It is important to make clear that we only compared DQL and DDPG to a single other algorithm and that other traditional algorithms may perform as well or even better than DRL. Particle swarm optimization and Monte Carlo methods are some possible candidates. For future work, it is important to test and compare these algorithms taking into account data requirement, computational effort, time, and ease of use.

ACKNOWLEDGMENT

We thank Y. M. Galperin for reading the paper and making valuable comments.

[1] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, Machine learning for molecular and materials science, Nature (London) **559**, 547 (2018).

[2] D. Shen, G. Wu, and H.-Il Suk, Deep learning in medical image analysis, Annu. Rev. Biomed. Eng. **19**, 221 (2017).

[3] J. R. Primack, A. Dekel, D. C. Koo, S. Lapiner, D. Ceverino, R. C. Simons, G. F. Snyder, M. Bernardi, Z. Chen, H. Domínguez-Sánchez *et al.*, Deep learning identifies high-z galaxies in a central blue nugget phase in a characteristic mass range, Astrophys. J. **858**, 114 (2018).

[4] R. Bellman, A markovian decision process, J. Math. Mech. **6**, 679 (1957).

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, Human-level control through deep reinforcement learning, Nature (London) **518**, 529 (2015).

[6] M. Bukov, A. G. R. Day, D. Sels, P. Weinberg, A. Polkovnikov, and P. Mehta, Reinforcement Learning in Different Phases of Quantum Control, Phys. Rev. X **8**, 031086 (2018).

[7] X.-M. Zhang, Z.-W. Cui, X. Wang, and M.-H. Yung, Automatic spin-chain learning to explore the quantum speed limit, Phys. Rev. A **97**, 052333 (2018).

[8] T. Fösel, P. Tighineanu, T. Weiss, and F. Marquardt, Reinforcement Learning with Neural Networks for Quantum Feedback, Phys. Rev. X **8**, 031084 (2018).

[9] A. A. Melnikov, H. P. Nautrup, M. Krenn, V. Dunjko, M. Tiersch, A. Zeilinger, and H. J. Briegel, Active learning machine learns to create new quantum experiments, Proc. Natl. Acad. Sci. USA **115**, 1221 (2018).

[10] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, Quantum machine learning, Nature (London) **549**, 195 (2017).

[11] D. Dong, C. Chen, H. Li, and T.-J. Tarn, Quantum reinforcement learning, IEEE Trans. Syst., Man, Cybernet., Part B—Cybernet. **38**, 1207 (2008).

[12] L. Lamata, Basic protocols in quantum reinforcement learning with superconducting circuits, Sci. Rep. **7**, 1609 (2017).

[13] P. Mehta, M. Bukov, C.-H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, A high-bias, low-variance introduction to machine learning for physicists, Phys. Rep. **810**, 1 (2019).

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning, arXiv:1509.02971.

[15] T. Caneva, T. Calarco, and S. Montangero, Chopped random-basis quantum optimization, Phys. Rev. A **84**, 022326 (2011).

[16] L. Szilard, über die entropieverminderung in einem thermodynamischen system bei eingriffen intelligenter wesen, Z. Phys. **53**, 840 (1929).

[17] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. **5**, 183 (1961).

[18] C. H. Bennett, Notes on landauer's principle, reversible computation, and maxwell's demon, Stud. Hist. Philos. Sci. B: Studies In History and Philosophy of Modern Physics **34**, 501 (2003).

[19] S. Toyabe, T. Sagawa, M. Ueda, E. Muneyuki, and M. Sano, Experimental demonstration of information-to-energy conversion and validation of the generalized jarzynski equality, Nat. Phys. **6**, 988 (2010).

[20] A. Bérut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz, Experimental verification of landauers principle linking information and thermodynamics, Nature (London) **483**, 187 (2012).

[21] J. V. Koski, V. F. Maisi, J. P. Pekola, and D. V. Averin, Experimental realization of a szilard engine with a single electron, Proc. Natl. Acad. Sci. USA **111**, 13786 (2014).

[22] M. D. Vidrighin, O. Dahlsten, M. Barbieri, M. S. Kim, V. Vedral, and Ian A. Walmsley, Photonic Maxwells Demon, Phys. Rev. Lett. **116**, 050401 (2016).

[23] S. Lloyd, Quantum-mechanical maxwells demon, Phys. Rev. A **56**, 3374 (1997).

[24] S. W. Kim, T. Sagawa, S. De Liberato, and M. Ueda, Quantum Szilard Engine, Phys. Rev. Lett. **106**, 070401 (2011).

[25] J. Gea-Banacloche, Splitting the wave function of a particle in a box, Am. J. Phys. **70**, 307 (2002).

[26] V. B. Sørdal and J. Bergli, Quantum particle in a split box: Excitations to the ground state, Phys. Rev. A **99**, 022121 (2019).

[27] J. A. Nelder and R. Mead, A simplex method for function minimization, Comput. J. **7**, 308 (1965).

[28] M. J. D. Powell, An efficient method for finding the minimum of a function of several variables without calculating derivatives, Comput. J. **7**, 155 (1964).

[29] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (ACM, New York, 2016), pp. 50–56.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).

[31] F. Chollet *et al.*, Keras, https://keras.io (2015).

[32] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[33] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *ICML'14 Proceedings of the 31st International Conference on International Conference on Machine Learning, Beijing, China, 2014* (ACM, New York, 2014).

[34] M. Plappert, keras-rl, https://github.com/keras-rl/keras-rl (2016).