A reinforcement learning approach for quantum state engineering

Jelena Mackeprang · Durga B. Rao Dasari · Jörg Wrachtrup

Abstract

Machine learning (ML) has become an attractive tool for solving various problems in different fields of physics, including the quantum domain. Here, we show how classical reinforcement learning (RL) could be used as a tool for quantum state engineering (QSE). We employ a measurement based control for QSE where the action sequences are determined by the choice of the measurement basis and the reward through the fidelity of obtaining the target state. Our analysis clearly displays a learning feature in QSE, for example in preparing arbitrary two-qubit entangled states and delivers successful action sequences that generalise previously found human solutions from exact quantum dynamics. We provide a systematic algorithmic approach for using RL for quantum protocols that deal with a non-trivial continuous state space, and discuss the scaling of these approaches for the preparation of larger entangled (cluster) states.

Keywords Quantum state engineering · Quantum control · Deep reinforcement learning

1 Introduction

Current advancements in quantum technology have rendered the research field and its applications increasingly complex. As machine learning (ML) has been successfully applied to various classical problems such as image recognition (Real et al. 2017; Krizhevsky et al. 2012), natural language processing (Lample et al. 2018) and board games (Silver et al. 2018), its applicability to quantum problems has gained increasing interest. Successful implementations of ML for problems in the field of quantum physics include neural networks for quantum state tomography (Torlai et al. 2018), detecting the speed-up of quantum walks (Melnikov et al. 2019) and classifying non-locality in quantum networks (Kriváchy et al. 2019).

There are three main variants of ML: supervised, unsupervised and reinforcement learning (RL). The latter is based on an abstract agent interacting with its environment and receiving information in the form of a reward. It has already been used in the optimisation of quantum control (Bukov et al. 2018; Niu et al. 2019; Chen et al. 2014), quantum state preparation (Bukov 2018), coherent quantum state transport (Porotti et al. 2019), quantum error correction (Nautrup et al. 2018; Sweke et al. 2018; Andreasson et al. 2019; Fösel et al. 2018) and much more.

Entangled quantum states are crucial resources for quantum optics and information tasks such as precision sensing, quantum communication and computing. Construction and maintenance of such states have been a long-term challenge involving the clever use of interactions between physical systems with ancillary degrees of freedom. One approach for preparing useful entangled states has been achieved through quantum measurements, where manipulating the state of a sub-system via sequences of (projective) measurements or other operations on this sub-system allows the rest of the system to be led to the desired entangled state. Here, we dub this approach quantum state engineering (QSE).

In this instance, we use a RL approach to obtain optimal control sequences for a QSE problem that generalise previous human solutions.

In RL, there is an agent and the environment. The agent (the learner), is supposed to reach a certain goal by performing actions \( a \) that change the state \( s \) of the environment (which is defined by everything outside of the agent). Neither the agent nor the environment necessarily have to be physical, i.e. the agent does not have to be an actual physical manifestation of an intelligent entity, like a robot.
The entire learning process is usually divided into so-called episodes, at the beginning of which the environment is reset to a starting state $s_0$. The episodes are in turn divided into discrete time steps $t$ (where the time does not necessarily have to correspond to real time), at which the agent selects an action to take. These actions end upon satisfying a certain criterion, e.g. when the maximum number of allowed time steps has passed or when the goal has been achieved.

At each time step, the agent is given the state $s$ of the environment, chooses an action $a$ that changes the state and then receives feedback in the form of a reward $r$. All rewards received after a time step $t$ accumulate to a return $G_t$ defined by the following:

$$G_t = \sum_{i=0}^{m} \gamma^i r_{t+i+1}$$

(1)

where $m$ is the number of steps taken per episode.

The return is a sum over all future rewards, discounted by a factor $\gamma$ with $0 \leq \gamma \leq 1$. The discount factor influences how important immediate rewards are to the agent as opposed to the long-term gain. If $\gamma = 0$, the agent’s only goal at every step will be maximising the first reward it receives after that step. If $\gamma = 1$, future rewards contribute equally to the sum in (1) and the agent will choose actions with regard to their long-term effect on the overall return. After having repeated this procedure for a long enough time, the agent is supposed to have the ability to choose the appropriate action at each state it encounters such that the return at each time step is maximised (Sutton and Barto 1998).

Usually, RL problems are formulated as a Markov decision process (MDP), which considerably simplifies the task. In an MDP, the probability of the environment entering the state $s'$ and the agent receiving a reward $r$ must only depend on the current state $s$ and the current chosen action $a$. In other words, it should not matter how the environment transitioned to a state $s'$ at a time step $t$ and what the agent did beforehand.

### 2 Modelling

In RL, the agent’s behaviour can be modelled by a policy $\pi(s, a)$, a probability distribution describing how likely the agent is to choose an action $a$ when encountering a given state $s$. For deterministic environments, i.e. where the reward $r$ is a deterministic function of the state $s$ and the action $a$, the policy $\pi(s, a)$ will just be one for one action and zero for the rest. The policy that maximises the overall return as defined in Eq. 1 is called the optimal policy $\pi^*(s, a)$. If one finds this optimal policy, the RL problem has been solved since the agent will now always select the action $a$ that maximises the return for any given $s$. Instead of trying to find a policy $\pi(s, a)$, the agent can also be tasked to find an action-value function $q(s, a)$ tied to a policy $\pi(s, a)$. The action-value $q(s, a)$ is defined as the expected return on the condition that the agent performed action $a$, that the environment was in state $s$ and that the agent subsequently follows the policy $\pi$. The action-value function corresponding to the optimal policy $\pi^*$ is called the optimal action-value function $q^*(s, a)$. (We have left out the subscript denoting the optimal policy.) Due to the agent acting according to the optimal policy, the optimal action-value function $q^*(s, a)$ gives back the maximum expected return for any initial $s$ and $a$.

Here, we apply the Q-learning approach first introduced by Watkins and Dayan (1992) where the agent tries to directly predict the optimal action-value function $q^*(s, a)$ as opposed to the optimal policy $\pi^*(s, a)$. The equation governing this RL algorithm relates the optimal action-value $q^*(s, a)$ of a state $s$ and action $a$ with the optimal action value of the successive state $s_{t+1}$ over all possible actions and is called the Bellman optimality equation:

$$q^*(s, a) = E_{s', a'} \left[ r_{t+1} + \gamma \max_{a'} q^*(s_{t+1}, a') \mid s_t = s, a_t = a \right]$$

(2)

Here, $E$ denotes an expectation value that is taken with respect to the optimal policy $\pi^*(s, a)$. Here, we deal with a deterministic environment, so the expectation value can be left out. In the following, we will treat all policies as deterministic.

We implement RL for a state space described by sets of continuous variables where the actions the agent can choose transfer the system from one state to the other. Our goal is for the state of the environment to reach a state that fulfils a predetermined criterion. The reward should therefore inform the agent whether or not the criterion has been satisfied. The fact that we are dealing with mixed quantum states constitutes a problem. Usually, when the goal is to lead the system described by a density matrix to a state that satisfies a certain condition, the RL state space $S$ has to consist of all independent entries of the entire density matrix in order for the Markov criterion to be fulfilled. This means that we deal with an exponentially growing, multi-dimensional, continuous, state space, whose optimal action-value function $q(s, a)$ has to be approximated.

One class of function approximators that have been shown to master large, high dimensional, continuous input data are neural networks (Mehta et al. 2018). Hence, we use the deep Q-network (Mnih et al. 2015) (DQN) and the double deep Q-network (Hasselt et al. 2016) (DDQN) algorithms that combine classical RL with artificial neural networks (ANNs). In both algorithms, an ANN takes on
the role of the agent’s “brain” which approximates the optimal action-value function \( q(s, a) \) returning the action-value \( q(s, a) \) for any state \( s \) represented by the set of continuous variables and any action \( a \).

At the beginning, the outputs \( q(s, a) \) for the inputs \( s \) and \( a \) of the ANN will be random, but over time, as the agent gains experience by observing its received rewards, its internal parameters determining its output are adapted by taking advantage of Eq. \( 2 \).

We use a simple feed-forward neural network that consists of multiple layers containing a number of nodes. Each node is connected to the nodes of the previous layer and its output is determined by a parameter assigned to each connection, the weight \( w \), and to the node itself, the bias \( b \), and an activation function. If \( x \) is the input vector passed element-wise from all previous nodes to one node via the connections and \( w \) is the weight vector consisting of the weights of all of these connections leading to the node, its output \( y \) is given by Mehta et al. (2018):

\[
y = a(w \cdot x + b) \tag{3}
\]

with \( a(x) \) being the activation function and \( b \) the bias of the node. The node will then pass this output to the next layer’s nodes and so on. As a result, the total output is a complex, intricate function of all weights and biases in the network. Non-linearities are implemented with the help of non-linear activation functions. More on neural networks can be found in Mehta et al. (2018). When a neural network is used to approximate a function \( f : X \to Y \), it gets trained with input values \( x \in X \) and targets \( y \in Y \). At every training step, it optimises its weights and biases with regard to a loss function that measures how close the approximated values are from the targets. An example for a loss function is the mean-squared error.

Here, the function \( f : X \to Y \) to be approximated by the neural network is the action-value function \( q : S \times A \to Q \) where \( Q \) is the space of all possible optimal action values. At each training step (of the entire learning algorithm), the agent “remembers” transition points \( (s, a, s', r) \) after passing through a certain number of episodes, i.e. an observed state \( s \), a selected action \( a \), the observed subsequent state \( s' \) and the received reward \( r \), and stores them in a so-called replay memory. After the agent has carried out a fixed number of episodes, a number of transition points determined by the batch size is taken out of the replay memory and the targets for these given points are calculated according to the following equation:

\[
y_t(s, a) = \begin{cases} \ r, & \text{if } s' \text{ is terminal} \\ \ r + \gamma \max_{a'} Q_t(s', a'), & \text{else} \end{cases} \tag{4}
\]

The deep Q-network then performs an optimisation step to minimise the loss. This is repeated until the agent’s performance is deemed satisfying.

As one can see, the targets for the action-value function as predicted by the neural network are updated using its current approximation also given by the network. The action \( a' \) that maximises the action value for a given state \( s \) is determined via the same function that also outputs the action value for said action \( a' \) and for the state \( s \) for the calculation of the new targets. The ever-changing targets may lead to instability in the function approximation process. The double DQN (DDQN) algorithm tries to avoid this by introducing a second neural network, the target network, that is utilised to calculate the new targets. While the main network determines the maximising action \( a' \), the target network gives the action value for said action and state. Both networks have the same architecture. The network that the agent uses as a reference and whose weights are constantly updated is the main network. Meanwhile, the internal parameters of the target network are slowly adjusted to the ones of the main network over the course of the entire learning process. In other words, at each step, the agent decides what action to take. To make this decision, it looks at the \( Q_t(s, a) \) values given by the main network. Then, the main network’s targets are updated by Hasselt et al. (2016):

\[
y_t(s, a) = \begin{cases} \ r, & s' \text{ terminal} \\ \ r + \gamma Q_t(s', \text{amax } Q \circ Q_t(s', a')), & \text{else} \end{cases} \tag{5}
\]

where \( Q_t \) is the action-value function given by the target network. First, the action that maximises the output of the main network is determined and then the new targets for the main network are calculated with the target network’s \( Q_t \) function taking that action as an argument. The main network performs an optimisation step and the weights and biases of the target network are slowly adjusted, either by copying all of the main network’s weights and biases to the target network after a fixed number of training steps or performing a soft update (Lillicrap et al. 2015) at every training step:

\[
w_t = (1 - \alpha)w_t + \alpha w_{\text{main}}, \quad b_t = (1 - \alpha)b_t + \alpha b_{\text{main}} \tag{6}
\]

Here, \( w_t \) and \( b_t \) are the weights and biases of the target network, \( w_{\text{main}} \) and \( b_{\text{main}} \) the weights and biases of the main network and \( \alpha \) is the update rate.

Due to few preceding interactions between the agent and the environment at the beginning of the learning process, it is essential to make sure that the agent explores a large number of possibilities before determining an action-value function to be optimal; otherwise, the agent could fail to find the optimal policy even though it is performing relatively well. In other words, it could generate a seemingly large return for each time step, but there actually could be an even better strategy that it had not found yet. This
is called the exploitation-exploration dilemma: The agent must balance to what extent it deviates from its current policy to explore other options and how heavily it exploits policies it has already found. Too little exploration could lead to accumulated re-occurring transitions stored in the replay memory causing the neural network to over-fit. A possible strategy to try to avoid this problem is to let the agent choose the action \( a \) with the highest action value according to its own approximation for a given state \( s \) at a time step \( t \) with a probability \( 1 - \epsilon \) and let it choose an action randomly with a probability \( \epsilon \). This is called an epsilon-greedy policy (Sutton and Barto 1998). We adapt this policy for our problem and linearly decrease \( \epsilon \) at each step of the DQN/DDQN algorithm until it has reached a final predetermined minimum value \( \epsilon_{\text{min}} \).

With this, we show that the agent can find the same combinations as humans but can also come up with its own solutions.

### 3 Quantum state engineering

Here, we implement RL for a central spin model composed of a central spin-1/2 system coupled to a spin environment. Such a model can be physically realisable using solid-state spins in diamond, where a single central electron spin, the nitrogen vacancy (NV) centre, is coupled to an internal \(^{13}\text{C}\) nuclear spin bath (Doherty et al. 2013; Wrachtrup and Finkler 2016). The central spin \( S \) is a spin one \((S = 1)\) system of which we choose a two-level subspace, and the nuclear spins \((I)\) are spin-half \( I = 1/2 \) particles, each with a two-level Hilbert space. The nuclear spins that constitute the spin bath are assumed to be weakly coupled, and hence non-interacting for the present study.

With the magnetic field aligned along the central spin axis (say NV \(z\)-axis), the Hamiltonian that determines the dynamics is given by Greiner et al. (2017):

\[
H = S^{(z)} \otimes \sum_k g_k(r) \cdot I_k + \omega \sum_k 1 \otimes I_k^{(z)}
\]

Here, \( g_k \) stands for the strength of the dipolar coupling between the NV (central) spin and the \( k \)th nuclear spin, dependent on the spatial separation \( r \) between these spins. \( S^{(z)} \) and \( I_k^{(z)} \) stand for the spin components along the \( z \)-axis of the system and \( I_k \) for the spin components along the three axes \( x, y \) and \( z \) of the \( k \)th nuclear spin. The identity operator is denoted by \( 1 \).

Because of the random spatial location of the spins with respect to the NVC, the couplings become in-homogeneous such that the bath has neither conserved quantities nor preferred symmetries. The nuclear spins precess under the external field \( \omega \) (the nuclear Zeeman term) that is acting along the \( z \)-direction and is assumed to be uniform. The dynamics generated by the Hamiltonian (7) can be exactly solved (Greiner et al. 2017).

The time evolution governed by it can be determined from the Schrödinger equation and is given by the following:

\[
\rho'(t + \tau) = U(\tau) \rho(t) U^\dagger(\tau)
\]

where \( U(\tau) = \exp(-i(t - \tau)H) \) and \( \rho'(t + \tau) \) is the time-evolved density matrix of the total system after a time interval \( \tau \).

To study the central spin model within the paradigm of RL, we determine the agent to be a virtual experimentalist combining projective measurements via optical read out, as demonstrated in Robledo et al. (2011), and time evolutions determined by Eq. 8 over fixed time intervals \( \tau \).

We employ here projective single-shot measurements as in Greiner et al. (2017), which give a deterministic result. The goal of the agent is to identify the sequences of projections of the central spin that result in the spin bath forming the target state. The environment is everything outside of the agent, i.e. the entire physical system including the central spin and the nuclear spins. Because of our choice of algorithms, the problem has to be formulated as a MDP. Therefore, we describe the states by the independent entries of the density matrix \( \rho \) belonging to all spins.

At each step of an episode, the agent lets the system freely evolve for a predetermined time \( \tau \) and then performs either of the following actions:

1. Project the central spin onto the state \( |z+\rangle \) by applying the projection operator \( P_{z+} = |z+\rangle \langle z+| \otimes \prod_{k=1}^\infty 1 \)
2. Project the central spin onto the state \( |z-\rangle \) by applying the projection operator \( P_{z-} = |z-\rangle \langle z-| \otimes \prod_{k=1}^\infty 1 \)
3. Project the central spin onto the state \( |x+\rangle = \frac{1}{\sqrt{2}} (|z+\rangle + |z-\rangle) \) by applying the projection operator \( P_{x+} = |x+\rangle \langle x+| \otimes \prod_{k=1}^\infty 1 \)
4. Project the central spin onto the state \( |x-\rangle = \frac{1}{\sqrt{2}} (|z+\rangle - |z-\rangle) \) by applying the projection operator \( P_{x-} = |x-\rangle \langle x-| \otimes \prod_{k=1}^\infty 1 \)
5. Project the central spin onto the state \( |y+\rangle = \frac{1}{\sqrt{2}} (|z+\rangle + i|z-\rangle) \) by applying the projection operator \( P_{y+} = |y+\rangle \langle y+| \otimes \prod_{k=1}^\infty 1 \)
6. Project the central spin onto the state \( |y-\rangle = \frac{1}{\sqrt{2}} (|z+\rangle - i|z-\rangle) \) by applying the projection operator \( P_{y-} = |y-\rangle \langle y-| \otimes \prod_{k=1}^\infty 1 \)
7. Do nothing, i.e. apply the identity operator \( 1 \) to \( \rho \)

As the projective measurements are non-unitary transformations, the time-evolved state \( \rho(t + \tau) \) must be appropriately normalised after every operation, for it to represent a valid density matrix. The density matrix \( \rho_{t+\tau} \) at the time \( t + \tau \) is derived from \( \rho_{t+\tau} \) following (8) and by applying
one of the projection operators $P_{\pm}$ listed above (or none of them) and then normalised:

$$
\rho_{t+\tau} = \frac{P_{\pm}^{\dagger} \rho_{t+\tau} P_{\pm}^{\dagger}}{\text{tr}(P_{\pm}^{\dagger} \rho_{t+\tau} P_{\pm}^{\dagger})}
$$

(9)

During each episode, the agent takes a maximum number $n_e$ of steps. It terminates and a positive reward $r_+$ is given if the partial trace of density matrix $\rho(t+\tau)$ over the Hilbert space of the central spin, denoted by $\rho_1$, following the time evolution and projection, has a fidelity $F$ with the target state $|\Psi\rangle$, that exceeds a certain threshold $\theta$, for example $\theta = 0.99$. We define the fidelity $F(\sigma, \rho)$ as follows:

$$
F(\sigma, \rho) = \text{tr}\sqrt{\sqrt{\rho} \sigma \sqrt{\rho}}.
$$

(10)

As long as the fidelity is not greater than $\theta$ and the agent has not performed the maximum number of actions, a negative reward $r_-$ is given. This way, and by using all independent entries of the density matrix as the state representation, we make sure that the reward received at some time $t$ and the subsequent state $s'$ only depends on the current chosen action $a$ (one of the projection operators or just the identity matrix) and the current state $s$, thereby fulfilling the Markov condition. In the following, we will consider a bath consisting of two nuclear spins, i.e. Eq. 7 simplifies to the following:

$$
H = S^{(z)} \otimes I_1 + \omega \mathbb{1} \otimes I_1^{(z)} + S^{(z)} \otimes I_2 + \omega \mathbb{1} \otimes I_2^{(z)}
$$

(11)

The specific values for the rewards $r_+$ and $r_-$ will be defined below.

The entire protocol is summarised in Fig. 1.

4 Construction of the Bell states

4.1 fixed central spin starting state

We consider the simplest case, i.e. a spin bath composed of two nuclear spins. For this, we use the DQN algorithm. The state of the environment is best described by a three-spin density matrix with 64 complex entries of which 35 are independent. Those 35 independent entries are then divided into their real and imaginary parts, resulting in an input of 70 continuous variables.

At first, the environment is initialised as the starting state $s_0$ belonging to $\rho_0$ defined in Eq. 12 at the beginning of each episode. The central spin is initialised as the $|x\rangle$ state following the promising results of the experiments reported by Johannes Greiner et al. (2017). They found that if the central spin of the previously described system is always brought into a superposition state before any projective measurements are applied, the spin bath can be purified with the help of a sequence of projective measurements and time evolutions, just as the agent is supposed to find. The nuclear spins are initialised in a completely mixed state:

$$
\rho_0 = |x\rangle \langle x| + \mathbb{1} \otimes \left( \frac{1}{2} \mathbb{1} + \frac{i}{2} \sigma_x \right) \otimes \left( \frac{1}{2} \mathbb{1} + \frac{i}{2} \sigma_x \right)
$$

(12)

The density matrix of the nuclear spins $I_1$ and $I_2$ is written in the $|m_z = \pm 1\rangle = |\pm\rangle$ basis.

As mentioned above, each episode consists of a fixed number of repetitions and terminates when a maximum overlap of the nuclear spins’ density matrix $\rho_1$ with the target state is achieved, or when all the $n_e$ actions have been performed. We choose the four maximally entangled Bell states $|\Phi^\pm\rangle$ and $|\Psi^\pm\rangle$ as our target states:

$$
|\Phi^\pm\rangle = 1/\sqrt{2} (|z\rangle |z\rangle \pm |z\rangle |z\rangle)
$$

(13)

$$
|\Psi^\pm\rangle = 1/\sqrt{2} (|z\rangle |z\rangle \pm |z\rangle |z\rangle)
$$

(14)

The system is simulated with the parameters from Eq. 7 set to $\omega = 1/2$ and $g_k = (1, 0, 0)$ $\forall k$ in relative units, i.e. the nuclear spins form a linear chain of spins arranged along the $x$-direction of the NVC. In Greiner et al. (2017), the authors say that the optimal choice for $\tau$ should be such that $\tau \sim 1/\omega$

That would be $\tau = 2$ in our case. To give the agent more flexibility, we divide the time interval by two, i.e. $\tau = 1$ relative to $\omega$. This way, the agent can theoretically select sequences with differing time intervals by choosing the option “do nothing”.

The reward and the indicator “done” for whether or not an episode has terminated are defined by Eq. 16, in which

![Fig. 1 Depiction of our RL scheme. The agent influences the central spin’s, and therefore the nuclear spin’s, state by applying one of the projection operators (or none of them) after letting the entire system freely evolve for a time $\tau$. The spin’s density matrix is then analysed and all necessary information is given to the agent who then decides over its next action, i.e. what projective measurement it performs, completing the circle](image)
$m$ stands for the number of steps taken and $P_{i\pm}$ and $\rho'_{i+1}$ are defined in Eq. 9. As the denominator in Eq. 9 can become too small for a computer to handle, some sort of punishment has to be introduced if the agent chooses a projective measurement that results in a computing error due to a denominator that is too close to zero. Receiving this reward $r_{\text{fatal}}$ also results in the end of the episode. Summing up, the reward $r$ is defined as follows:

$$(r, \text{done}) = \begin{cases} 
(r_+, 1) & F(\rho_{i}, |\Psi\rangle \langle \Psi|) > \theta \\
(r_-, 0) & F(\rho_{i}, |\Psi\rangle \langle \Psi|) < \theta \land m_e < n_e \\
(r_-, 1) & F(\rho_{i}, |\Psi\rangle \langle \Psi|) < \theta \land m_e = n_e \\
(r_{\text{fatal}}, 1) & \text{tr} (P_{i\pm} \rho'_{i+1} P_{i\pm}^\dagger) \approx 0 
\end{cases}$$

(16)

Here, $m_e$ stands for the number of steps taken and $n_e$ for the maximum number of steps per episode. When implementing definition Eq. 16, we realise the last case by creating an exception for a division by zero error raised by the computer that terminates the episode and outputs the fatal reward $r_{\text{fatal}}$.

We would like to comment on the role of the discount factor $\gamma$ for this approach. This RL task is episodic, so the discount factor could also be set to 1, as the sum in Eq. 1 would still converge. However, in DQN and DDQN, the neural network only approximates the optimal action-value function $q(s, a)$. Therefore, there always is a certain uncertainty to the actions the agent deems optimal that lay far in the future. To account for these uncertainties while also making sure the agent cares about the long-term future, we do not use a discount factor of 1, but 0.99.

For all cases mentioned in the following, $\theta$ is set to 0.99, $r_+$ to 10, and $r_-$ to −1. The episodes consist of 50 steps for $|\Psi^\pm\rangle$, $|\Psi^-\rangle$ and $|\Phi^+\rangle$. The reward seems to be sparsely distributed for $|\Phi^-\rangle$; hence, we set the number of steps per episode to 60 to help the agent receive more positive rewards. The fatal reward must always be smaller than the negative number of steps per episode, as the negative reward is −1 and it should not be more rewarding for the agent to perform an action that terminates the episode with the fatal reward than continuing to look for the positive reward. The agent carries out a constant, sufficiently large number of episodes at each training step. As described above, it remembers the transitions and stores them in a replay memory. We have implemented an $\epsilon$-greedy policy with a linearly decaying $\epsilon$. To illustrate the agent’s improving performance over time, Fig. 2 shows the average undiscounted sum of all rewards received during the episodes at each training step in the case of the Bell state $|\Psi^-angle$ where the system starts out with a density matrix given by Eq. 12:

We show the same for the target states $|\Phi^-\rangle$, $|\Phi^+\rangle$ and $|\Psi^+\rangle$ Fig. 3.

Both in Figs. 2 and 3, $\epsilon$ decays to a constant value $\epsilon_{\text{end}}$ that is 0.1 in all cases except for $|\Phi^+\rangle$ where it is 0.01. As the number of steps needed to reach the positive reward is relatively low, an agent that performs well could accumulate too much redundant data in the replay memory, thereby inducing over-fitting or forgetting about other policies than the one according to which it is acting. To avoid this, $\epsilon$ does not decay to zero in Figs. 2 and 3 but to 0.1 for $|\Psi^-\rangle$, $|\Psi^+\rangle$ and $|\Phi^-\rangle$ and to 0.01 for $|\Phi^+\rangle$. For the latter, the agent seems to struggle more, which is why a larger $\epsilon$ hinders convergence.

To make sure that it has found a policy superior to one where it randomly chooses an action at any time step, we

Fig. 2. Average over the total sum of all received rewards during the fixed number of episodes completed at each training step of the DQN algorithm for the target state $|\Psi^-\rangle$. The dashed line indicates the linearly decaying $\epsilon$. ($\epsilon$-greedy policy, see section Modelling.) When it is small enough, it stops decaying and stays constant at $\epsilon = 0.1$. The maximum number of steps per episode $n_e$ is set to 50 and $r_{\text{fatal}} = -51$. At the beginning of each episode the central spin is initialised in the state $|x^+\rangle$. Initially, the agent is rarely successful in achieving its goal but over time, its progress improves until it stays stagnant.
Fig. 3 Average over the total sum of all received rewards during the fixed number of episodes completed at each training step of the DQN algorithm for different target states, starting state: $|+\rangle$. On the dashed red line’s right side, the linearly decaying $\epsilon$ has reached its minimum value and stays constant. For $|\Phi^+\rangle$ and $|\Psi^+\rangle$ $n_e$ is equal to 50 and $r_{\text{fatal}}$ to $-51$, for $|\Phi^-\rangle$ $n_e$ is equal to 60 and $r_{\text{fatal}}$ to $-61$. As for $|\Psi^-\rangle$, the agent on average receives a growing number of rewards per episode over time.

Table 1 lists the sequences with the highest success rate the agent has found for the respective target states, where the success rates are the products of the success probabilities

compare the agent’s performance at a small $\epsilon$ value to its behaviour at $\epsilon = 1$, i.e. when acting according to a completely random policy (Fig. 4).

Fig. 4 Comparison of a randomly acting agent ($\epsilon = 1$, red dots) to a trained one ($\epsilon = 0.1$, green triangles) for different target states. Each point or triangle represents the total sum of received rewards during an episode. The agent receives a reward of 10 and terminates the episode if a fidelity of 0.99 with the target state is reached and receives a reward of $-1$ if the fidelity is below this threshold, i.e. the higher the total reward per episode, the fewer steps the agent needed to construct the desired target state. The trained agents perform better than the ones acting according to a random policy.
for each central spin read out, i.e. the denominator in Eq. 9. The purity $\text{tr} \left( \rho P_i^\dagger \rho P_i \right)$ of the nuclear spin state, the fidelity $F(\rho_i, |\Psi^-\rangle|\Psi^-\rangle)$ of it and the desired Bell state, their trace distance and the success probability $\text{tr} \left( P_i^\dagger \rho P_i \right)$ of the projective measurement (or lack thereof) at each step $i$ are visualised in Fig. 5 at the example of the first sequence in Table 1 for the target state $|\Psi^+\rangle$. Here, the trace distance of two density matrices $\rho$ and $\sigma$ is defined as follows:

$$D(\rho, \sigma) = \sqrt{(\rho - \sigma) \cdot (\rho - \sigma)^\dagger}.$$  

(17)

Figure 5 shows a monotonically decreasing trace distance and a monotonically increasing fidelity which begs the question as to why the reward could not have been simply defined as the fidelity of the nuclear spin state with the target state at each step of the episode. The reason why this does not work can be seen at the example of one sequence the agent has found for the target state $|\Phi^+\rangle$ in Fig. 6.

The data plotted in Fig. 6 clearly demonstrates that the reward must be defined as it is in Eq. 16, i.e. only returning a positive value if the fidelity has exceeded the threshold $\theta$, since the fidelity does not increase continuously with the episode step $i$. On the contrary, it plunges to nearly zero at the fourth step, i.e. an increasing or decreasing fidelity does not give the agent any information on how to reach its end goal.

For the case of $|\Psi^-\rangle$, the agent has found the same sequence as Greiner et al. (2017) (fourth row of Table 1) where they showed that for a linear chain containing an even number of spins arranged along the $x$-axis of the system.
the steady state of the nuclear spin ensemble evolves to the following:

\[ |\Psi\rangle = \otimes_{i=1}^{N} |\Psi^\pm\rangle \]  

when repeatedly applying \( P_x^+ \) after the ensemble has freely evolved for a fixed time interval \( \tau \). (\( N \) is the total number of spins divided by two.) However, the agent has also found many other sequences consisting of projections onto superposition subspaces, i.e. the \( |x\pm\rangle \) and \( |y\pm\rangle \) subspaces, as one can see in Table 1. To further demonstrate this, the number of appearances of action combinations over the course of successful and unique episodes taken from a total set of 3000 episodes are illustrated in the histogram Fig. 7.

Interestingly, the agent very much prefers repeating the same action over choosing another one except for the \( P_z^+ \) projection and the option “do nothing”. Apparently, repeatedly selecting either \( P_x^+ \), \( P_x^- \), \( P_y^+ \) or \( P_y^- \) during a sequence leads to \( |\Psi^-\rangle \). This means that the agent has found other sequences besides the one described by Greiner et al., so it generalised the solutions found by them.

4.2 Random (pure) central spin starting state

Apart from initialising the environment in a definite state \( s_0 \) whose corresponding density matrix is defined by Eq. 12, we also let the agent search for combinations of projective measurements when the state of the central spin is initialised in a random pure state \( |\xi\rangle \) at the beginning of each episode:

\[ \rho_0 = |\xi\rangle \langle \xi| \otimes \left( \begin{array}{cc} 1 & 0 \\ 0 & \frac{1}{2} \end{array} \right) \otimes \left( \begin{array}{cc} 1 & 0 \\ 0 & \frac{1}{2} \end{array} \right) \]  

In the following, we will only discuss the result for the target state \( |\Psi^-\rangle \). As randomness leads to a larger variety of states the agent can encounter, and therefore to increased noise, we utilise the double DQN algorithm for more stable learning. We simulate the system with the same \( g_k \) and \( \omega \) as before, but \( \tau \) is now set to 2 since the agent has frequently chosen to let the system evolve freely for the time \( \tau = 2 \) during the last step. (This is possible by selecting the action “do nothing”.) The maximum number of steps per episode \( n_e \) is chosen to be 50 and \( r_{\text{fatal}} \) is set to -51.

The agent now has to try out more combinations than before until it encounters a positive reward. To ensure that these experiences stand out to the agent from the vast number of all unsuccessful attempts, we increase the positive reward \( r_+ \) to 50. The agent’s progress is shown in Fig. 8, and it seems to be unstable which may be due to over-fitting of the under-laying neural networks (Mehta et al. 2018).

For the analysis of a trained agent’s behaviour, we extract the agent’s copy at the 1900th, 2000th, 2290th and the 2500th training step where its performance has local maxima. Its behaviour at these training steps is compared to the performance of an untrained agent in Fig. 9.

To compare this agent’s behaviour with the previous ones, we set the starting states for the trained agent to \( |x^+\rangle \) and \( |x^-\rangle \) and observe the sequences it chooses. Tables 2...
Fig. 9 Comparison of the agent’s copies ($\epsilon = 0.01$) at different training steps during the learning process shown in Fig. 8 to untrained ones ($\epsilon = 1$). The trained agents receive more rewards per episode than the untrained ones.

Table 2 Shortest sequences of time evolutions and subsequent projective measurements found by the agent’s copies at different training steps that lead to the $|\Psi^+\rangle$ state for the $|x+\rangle$ starting state

| Training step | Sequence                                      | Fidelity  | Success rate |
|---------------|-----------------------------------------------|-----------|--------------|
| 1900          | No positive reward reached before the maximum number of steps |           |              |
| 2000          | $-U(\tau)\cdot P_{x+} \cdot P_{x+} \cdot U(\tau) \cdot P_{x+} \cdot U(\tau) \cdot P_{x+}$ | 99.227    | 25.391       |
| 2290          | $-U(\tau)\cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{y-}$ | 99.227    | 12.695       |
| 2500          | $-U(\tau)\cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{y+} \cdot U(\tau) \cdot P_{x+}$ | 99.227    | 6.348        |

As opposed to before, this agent was trained using the double DQN algorithm.

Table 3 Shortest sequences of time evolutions and subsequent projective measurements that lead to the $|\Psi^+\rangle$ state for the $|x-\rangle$ starting state found by the double DQN agent’s copies at different training steps

| Training step | Sequence                                      | Fidelity  | Success rate |
|---------------|-----------------------------------------------|-----------|--------------|
| 1900          | $-U(\tau)\cdot P_{x-} \cdot U(\tau) \cdot P_{x-} \cdot U(\tau) \cdot P_{x-}$ | 99.227    | 25.391       |
| 2000          | No positive reward reached before the maximum number of steps |           |              |
| 2290          | $-U(\tau)\cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{y-}$ | 99.227    | 12.695       |
| 2500          | $-U(\tau)\cdot P_{y-} \cdot U(\tau) \cdot P_{y-} \cdot U(\tau) \cdot P_{x+} \cdot U(\tau) \cdot P_{x+}$ | 99.227    | 6.348        |
and 3 list the shortest successful sequences for \(|x^+\rangle\) and \(|x^-\rangle\) as the central spin starting states the trained agents acting with \(\epsilon = 0.01\) can find.

Together with the histogram, in Fig. 10, they show that for random or the \(|x^\pm\rangle\) starting states, the agent prefers repeating projective measurements onto a single subspace which belongs to \(|x^+\rangle\), \(|x^-\rangle\), \(|y^+\rangle\) or \(|y^-\rangle\). Apart from that, it also mixes projective measurements onto these subspaces. We therefore conclude that in order for the two nuclear spins to end up in the \(|\psi^-\rangle\) state, one has to repeatedly perform projections onto one of the superposition subspaces, no matter what the state of the central spin was in the beginning. Hence, with the help of deep reinforcement learning, we have generalised the findings of Greiner et al. (2017) for the case of two nuclear spins.

5 Conclusion and discussion

In this study, we applied the technique of deep reinforcement learning to the task of quantum state engineering. An agent took the role of the experimentalist. It was tasked with finding sequences of projective measurements and time evolutions of an NVC electron spin coupled to a nuclear spin bath that polarised the nuclear spin bath in a desired fashion.

We looked at the \(m_3 = \pm 1\) subspace of the central spin and two \(^{13}\text{C}\) spins that is was coupled to and instructed agents to find sequences that take the nuclear spins from a completely mixed state to the maximally entangled Bell states. The agent did indeed find sequences composed of projective measurements and free evolution periods that lead to the formation of any of the four Bell states. For the formation of singlet state, \(|\psi^-\rangle\), the agent found an identical sequence as Greiner et al. in (2017). However, it also generalised these human found solutions by finding many more sequences that lead to the formation of a similar entangled state.

As the entire density matrix was included into the state representation and the reward was defined as given in Eq. 16, the transition probability of the environment and the probability of the agent receiving a reward \(r\) only depended on the current state \(s\) and selected action \(a\). Therefore, the problem constituted a Markov decision process, allowing it to be solved with a deep Q- and double deep Q-network algorithm.

The implementation of a deep reinforcement learning algorithm for a larger spin ensemble is straightforward and could theoretically lead to more sophisticated quantum protocols for the task of creating highly entangled quantum states in a large nuclear spin bath. However, as the number of entries of the density matrix describing the entire system grows exponentially with the number of spins, the DQN and DDQN with a state representation as the one we have chosen, will become very difficult to scale. (For RL to work, the environment’s dynamics have to be constantly simulated all over again. In our case, this is what took the most time.) For this reason, we suggest to study this further by reducing the dimension of the state representation and formulating the problem as a partially observable Markov decision process (POMDP), where the state representations are replaced by observations that do not fulfil the Markov property, which could then be handled by a recurrent neural network (Hausknecht and Stone 2015).

After reducing the state representations to observations and thereby improving the scalability of the algorithm, one could then try to implement the RL learning protocol in real life. For example, the agent could perform its actions on an actual NV centre and optimise its policy for observables accessible during an experiment. The costliest part of our algorithm, the simulation of the quantum dynamics, would therefore be outsourced to a real quantum system. Also, in the future, when there may be devices suited to simulate the dynamics of a large spin ensemble, the task of computing the system dynamics could be managed by such a device.
However, when implementing our scheme in an experiment, one needs to consider the following: The measurement sequences obtained here are probabilistic and one may have to post-select them over many experimental runs. This seems feasible for a finite number of measurements as was recently demonstrated experimentally (Bhaktavatsala Rao et al. 2019), where the authors have performed four sequential measurements and repeated the experiment \(\sim 10^5\) times to obtain statistics.

Another possibility would be to use all previous actions as the state representations, i.e. the agent would change the environment’s state from \(P_{x+} - P_{x-}\) to \(P_{x+} - P_{y+} - P_{y-}\) by selecting the action \(P_{y+}\), and implement the projective simulation algorithm proposed by Hans J. Briegel and Gemma De las Cuevas (2012). Projective simulation has been used to create photonic quantum experiments where the agent receives the entire optical setup as the state representation and changes the state of the environment by adding an optical element to the setup (Melnikov et al. 2018). This would be analogous to a problem where the agent takes in the entire history of projections and time evolution periods as the state representation and changes the state by adding another action, i.e. another projection (or none). An analogous state space was also discussed in Bukov (2018).

To conclude, we have theoretically described how measurement-induced quantum state engineering could be achieved within the framework of RL. We found that the agent could find optimal measurement sequences that lead the system towards a desired entangled state. In general, RL could be applied for other quantum protocols if they have some sort of trial-and-error nature.

However, combining machine learning on a classical computer with problems including a quantum system becomes increasingly difficult due to an exponential growth in the parameter space. One would only be able to fully harness the abilities of RL in quantum information processing upon reducing/optimising the information flow to the agent and further replacing the classical computation of quantum dynamics with a quantum simulation (given that the quantum simulator is powerful enough) or by an actual experimental setup.

Acknowledgements The first author would like to thank Emily Miraglia and Noah Diller-Schatz for their generous help.

Funding information This study is supported by the Max Planck Society as well as the EU via the project ASTERIQS and QIA, the ERC grant SME-L and the DFG (FOR 2724).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Appendix

A.1 Neural network architecture

For the DQN algorithm used for the fixed initial state, we use a three-layered neural network with 256 nodes each and rectified linear units as activation functions (Nair and Hinton 2010). For training them, we use the Adam optimiser (Kingma and Ba 2014) with a learning rate set to 0.001 and \(\epsilon\) belonging to the optimiser (not to be confused with \(\epsilon\) belonging to the \(\epsilon\)-greedy strategy) to \(10^{-6}\). This may seem excessive but after trying out different several different architectures this one seemed to work best.

For the DDQN algorithm used for the random initial state, we use a two-layered neural network with 128 nodes and the same activation functions as above. The learning rate and \(\epsilon\) stay the same. The architecture 256-256-256 led to over-fitting, hence the reduced number of layers and nodes. We tested other combinations as well but none of them led to convergence.

In all cases, the batch size of the sample of transitions points taken from the replay memory for the neural network to train on is 32.

A.2 Hyper-parameters

During learning in Section 4.1, \(\epsilon\) decays to a final \(\epsilon_{\text{end}}\) according to the following:

\[
\epsilon = \max\left(\epsilon_0 - \lambda \frac{\text{number of training steps taken yet}}{N}, \epsilon_{\text{end}}\right)
\]  

(20)

where \(\epsilon_0\) is the initial \(\epsilon\) factor, \(\lambda\) is a fixed decay rate and \(N\) is the maximum number of steps of the entire learning process.

The hyper-parameters used for Section 4.1 applied to construct the states \(|\Phi^+\rangle, |\Psi^-\rangle, |\Phi^-\rangle\) and \(|\Psi^+\rangle\) are summarised in Tables 4, 5, 6 and 7. In all cases, the number of episodes the agent experienced before the neural network performed an optimisation step is 50.

For the random initial state, we changed the way \(\epsilon\) decays to the following:

\[
\epsilon = \max\left(\epsilon_{\text{end}}, 1 - \frac{\text{number of training steps taken yet}}{\epsilon_{\text{steps}}}\right)
\]  

(21)

Table 4 Hyper-parameters for the construction of \(|\Phi^+\rangle\). (Fixed initial state.)

| \(\epsilon_0\) | \(\lambda\) | \(\epsilon_{\text{end}}\) | \(N\) |
|---|---|---|---|
| 1 | 3 | 0.1 | 2000 |
This means that $\epsilon$ decays from 1 to $\epsilon_{\text{end}}$ in $\epsilon_{\text{steps}}$ steps. We used Eq. 21 instead of Eq. 20 to make the decay of $\epsilon$ independent from the number of training steps such that we could stop the learning procedure at any moment. The hyper-parameters used for the random initial state are summarised in Table 8 where $\alpha$ stands for the update rate (see Eq. 6).

## References

Andreasson P, Johansson J, Liljestrand S, Granath M (2019) Quantum error correction for the toric code using deep reinforcement learning. Quantum 3:183. https://doi.org/10.22331/q-2019-09-02-183

Bhaktavatsala Rao DD, Yang S, Jesenški S, Tekin E, Kaiser F, Wrachtrup J (2019) Observation of nonclassical measurement statistics induced by a coherent spin environment. Phys Rev A 100:022,307. https://doi.org/10.1103/PhysRevA.100.022307

Briegel HJ, De las Cuevas G (2012) Projective simulation for artificial intelligence. Sci Report 2:400. https://doi.org/10.1038/srep00400.

Bukov M (2018) Reinforcement learning for autonomous preparation of Floquet-engineered states: Inverting the quantum Kapitza oscillator. Phys Rev B 98:224,305. https://doi.org/10.1103/PhysRevB.98.224305

Bukov M, Day AG, Sels D, Weinberg P, Polkovnikov A, Mehta P (2018) Reinforcement learning in different phases of quantum control. Phys Rev X 8(3):031,086. https://doi.org/10.1103/PhysRevX.8.031086

Chen C, Dong D, Li H, Chu J, Tarn T (2014) Fidelity-based probabilistic Q-learning for control of quantum systems. IEEE Trans Neural Netw Learn Syst 25(5):920–933. https://doi.org/10.1109/TNNLS.2013.2283574

Doherty MW, Manson NB, Delaney P, Jelezko F, Wrachtrup J, Hollenberg LCL (2013) The nitrogen-vacancy colour centre in diamond. Phys Report 528(1):1–45. https://doi.org/10.1016/j.physrep.2013.02.001. arXiv:1302.3288

Fösel T, Tighineanu P, Weiss T, Marquardt F (2018) Reinforcement learning with neural networks for quantum feedback. Phys Rev X 8:031,084. https://doi.org/10.1103/PhysRevX.8.031084

Greiner JN, Dasari DBR, Wrachtrup J (2017) Purification of an unpolarized spin ensemble into entangled singlet pairs. Scientific Reports 7. https://doi.org/10.1038/s41598-017-00603-z. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5428879/

Hasselt HV, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. In: Proceedings of the 13th AAAI conference on artificial intelligence, AAAI Press, Phoenix, Arizona, AAAI 16, pp 2094–2100. http://dl.acm.org/citation.cfm?id=3016100.3016191

Hausknecht M, Stone P (2015) Deep recurrent Q-learning, for partially observable MDPs. arXiv:1507.06527 [cs]

Kingma D, Ba J (2014) Adam: a method for stochastic optimization. International Conference on Learning Representations

Kriváčy T, Cai Y, Cavalcanti D, Tavakoli A, Gisin N, Brunner N (2019) A neural network oracle for quantum nonlocality in networks. 1907.10552

Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet classification with deep convolutional neural networks. Neur Inf Process Syst 60:25. https://doi.org/10.1145/3065386

Lample G, Ott M, Conneau A, Denoyer L, Ranzato M (2018) Phrase-based, & neural unsupervised machine translation. arXiv:1804.07755 [cs]

Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous control with deep reinforcement learning. 1509.02971

Mehta P, Bukov M, Wang CH, Day AGR, Richardson C, Fisher CK, Schwab DJ (2018) A high-bias, low-variance introduction to machine learning for physicists. arXiv:1803.08823 [cond-mat, physics:physics, stat]

Melnikov AA, Nautrup HP, Krenn M, Dunjko V, Tiersch M, Zeilinger A, Briegel HJ (2018) Active learning machine learns to create new quantum experiments. Proc Nat Acad Sci 115(6):1221–1226. https://doi.org/10.1073/pnas.1714936115. https://www.pnas.org/content/115/6/1221

Melnikov AA, Fedichkin LE, Alodjants A (2019) Detecting quantum speedup by quantum walk with convolutional neural networks. 1901.10632

Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://www.nature.com/articles/nature14236

Nair V, Hinton GE (2010) Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th international conference on international conference on machine learning, Omnipress, USA, ICML’10, pp 807–814. http://dl.acm.org/citation.cfm?id=3104322.3104425

Nautrup HP, Delfosse N, Dunjko V, Briegel HJ, Friis N (2018) Optimizing quantum error correction codes with reinforcement learning. 1812.08451

Niu M, Boixo S, Vea S (2019) Universal quantum control through deep reinforcement learning. npj Quantum Inf 5(33):447–450. https://doi.org/10.1038/s41534-019-0141-3
Porotti R, Tamascelli D, Mea Restelli (2019) Coherent transport of quantum states by deep reinforcement learning. Commun Phys 2(61):61. https://doi.org/10.1038/s42005-019-0169-x

Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le Q, Kurakin A (2017) Large-scale evolution of image classifiers. arXiv:1703.01041 [cs]

Robledo L, Childress L, Bernien H (2011) High-fidelity projective read-out of a solid-state spin quantum register. Nature 477:574–578. https://doi.org/10.1038/nature10401

Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, Lanctot M, Sifre L, Kumaran D, Graepel T, Lillicrap T, Simonyan K, Hassabis D (2018) A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362(6419):1140–1144. https://doi.org/10.1126/science.aar6404. https://science.sciencemag.org/content/362/6419/1140

Sutton RS, Barto AG (1998) Reinforcement learning: an introduction

Sweke R, Kesselring MS, van Nieuwenburg EPL, Eisert J (2018) Reinforcement learning decoders for fault-tolerant quantum computation. 1810.07207

Torlai G, Mazzola G, Carrasquilla J (2018) Neural-network quantum state tomography. Nature Phys 14:447–450. https://doi.org/10.1038/s41567-018-0048-5

Watkins CJCH, Dayan P (1992) Q-learning. Mach Learn 8(3):279–292. https://doi.org/10.1007/BF00992698

Wrachtrup J, Finkler A (2016) Single spin magnetic resonance. J Magn Reson 269:225–236. https://doi.org/10.1016/j.jmr.2016.06.017

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.