Quantum error correction for the toric code using deep reinforcement learning

Philip Andreasson,* Joel Johansson,† Simon Liljestrand,‡ and Mats Granath§

Department of Physics, University of Gothenburg, SE-41296 Gothenburg, Sweden

We implement a quantum error correction algorithm for bit-flip errors on the topological toric code using deep reinforcement learning. An action-value Q-function encodes the discounted value of moving a defect to a neighboring site on the square grid (the action) depending on the full set of defects on the torus (the syndrome or state). The Q-function is represented by a deep convolutional neural network. Using the translational invariance on the torus allows for viewing each defect from a central perspective which significantly simplifies the state space representation independently of the number of defect pairs. The training is done using experience replay, where data from the algorithm being played out is stored and used for batch upgrade of the Q-network. We find performance which is close to that achieved by the Minimum Weight Perfect Matching algorithm for code distances up to $d = 7$, which shows that the deep Q-network is highly versatile in dealing with varying numbers of syndrome defects.

I. INTRODUCTION

Much of the spectacular advances in machine learning using artificial neural networks has been in the domain of supervised learning were deep convolutional networks excel at categorizing objects when trained with big annotated data sets[1–3]. A different but also more challenging type of problem is when there is no a priori solution key, but rather a dynamic environment through which we want to learn to navigate for an optimal outcome. For these types of problems reinforcement learning (RL) [4] combined with deep learning has had great success recently when applied to problems such as computer and board games[5–8].

In physics the use of machine learning has seen a great deal of interest lately[9–13]. The most natural type of application of neural networks is in the form of supervised learning where the deep network can capture correlations or subtle information in real or artificial data. The use of deep reinforcement learning may be less obvious in general as the type of topics addressed by RL typically involve some sort of "intelligent" best strategy search, contrary to the deterministic or statistical models used in physics.

In this paper we study a type of problem where we are looking for an intelligent solution, namely a best strategy for error correction of a topological quantum code; the basic building block of a quantum computer. In the field of quantum computing, physics merge with computer science and smart algorithms are needed for error correction of fragile quantum bits[14, 15]. To use machine learning, and in particular reinforcement learning, has been suggested recently as a tool for quantum error correction and quantum control[16–18]. There are also very interesting prospects of utilizing the natural parallelization of quantum computers for machine learning itself[19], but we will be dealing with the more mundane task of putting deep learning at the service of quantum computing.

We consider Kitaev's surface code on a torus which is a stabilizer formalism that projects a large number of physical qubits into smaller number of entangled logical qubits[15, 20, 21]. Error (phase or bit flips) in the physical qubits are only manifested by the syndrome which indicates violated stabilizers but does not uniquely identify the physical errors. On a lattice of $2d^2$ physical cubits the logical qubits are separated by the code distance d meaning that chains of errors of length d or more may correspond to logical operations. On the torus these correspond to topologically non-trivial loops. The challenge is thus to find an algorithm for the error correction that can suggest which physical qubits to operate on without accidentally generating non-trivial loops. The problem is well suited for RL, similar in spirit to a board game, where the agent moves the defects of the syndrome, corresponding to suggesting operations on the physical cubits, and with reward given for successful error correction. It is a challenging problem given the very large state space that corresponds to a syndrome containing many defects, and given the fact that the success or failure cannot be assessed until the end of a complete episode.

In recent work [22] an application of reinforcement learning to error correction of the surface code was implemented. That work focuses on the important issue of error generated in the readout of the syndrome and used an auxiliary "referee decoder" to assist the performance of the RL decooder. In the present work we consider the simpler but conceptually more direct problem of error correction on a perfect syndrome, not corrupted by error. The problem can be addressed by the Minimum Weight Perfect Matching (MWPM) or Blossom algorithm[23, 24] and has also been the topic of many studies using methods such as renormalization group[25], cellular automata[26, 27], and a number of neural network based decoders typically using supervised learning[17, 28–33]. We find that by setting up a reward scheme that encourage the elimination of the syndrome in

* phiandr@student.chalmers.se
† gusjohjodt@student.gu.se
‡ simlilj@student.chalmers.se
§ mats.granath@physics.gu.se

as few operations as possible within the deep Q-learning (or deep Q-network, DQN)[6, 7] formalism we are able to arrive at an algorithm that is comparable in performance to MWPM. Although the present algorithm does not outperform the latter we expect that it has the potential to be more versatile when addressing correlated noise, measurement noise, or the surface code for varying geometries. Compared to the MWPM algorithm the RL algorithm also has the advantage that it provides step by step correction, meaning that it can readily adjust to the introduction of additional errors during the correction epsiode without recalculating the full error correcting strings of bit (or phase) flips.

The outline of the paper is the following. In the *Background* section we give a brief but self-contained summary of the main features of the toric code including the basic structure of the error correction and a similar summary of one-step Q-learning and deep Q-learning. (The reader familiar with these topics can readily skip ahead.) The following section, *RL Algorithm*, describes the formulation and training of the error correcting agent. In the *Results* section we shows that we have trained the RL agent up to system sizes of 7×7 with performance which is very close to the MWPM algorithm. We finally conclude and append details of the the neural network architecture and the RL and network learning hyperparameters.

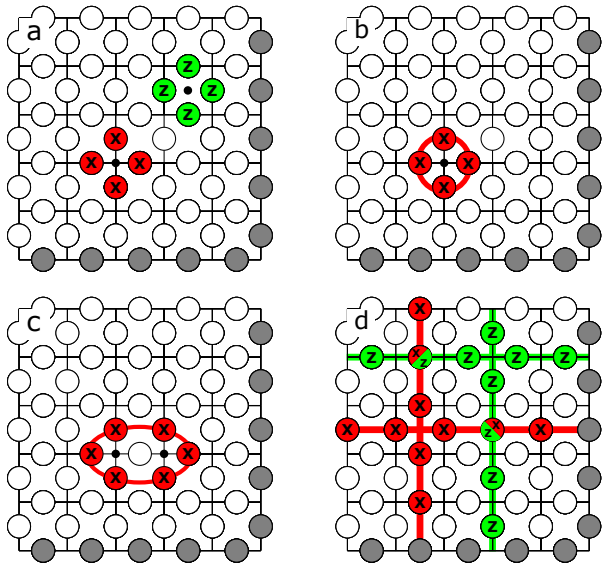

FIG. 1. A $d = 5$ toric code lattice with rings indicating the physical cubits and grey showing the periodic boundary conditions. a) Plaquette (green) and vertex (red) operators, as products of σ^z and σ^x Pauli matrices. b) A single vertex operator can be represented as a loop flipping the cubits that the it crosses. c) Two neighboring vertex operators make up a larger loop. d) The logical operators $X_{1/2}$ (red) and $Z_{1/2}$ (green) consist of loops winding the torus and are not representable in terms of products of vertex or plaquette operators.

II. BACKGROUND

II.1. Toric code

Here we recapitulate the main aspects of the topological toric code in an informal manner and from the perspective of an interacting quantum spin-Hamiltonian.[20, 21, 34]

The basic construction is a square lattice with a spin-$\frac{1}{2}$ degree of freedom on every bond, the physical qubits, and with periodic boundary conditions making up the torus, see Figure 1. The model is given in terms of a Hamiltonian

$$H = -\sum_{\alpha} \hat{P}_{\alpha} - \sum_{\nu} \hat{V}_{\nu} \,, \tag{1}$$

where α runs over all plaquettes and ν over all vertices (sites). The stabilizers are the plaquette operators $\hat{P}_{\alpha} = \prod_{i\in\alpha} \sigma_i^z$ and the vertex operators $\hat{V}_{\nu} = \prod_{i\in\nu} \sigma_i^x$, where σ^z and σ^x are the Pauli matrices. (Where, in the σ^z basis, $\sigma^z|\uparrow / \downarrow\rangle = \pm 1|\uparrow / \downarrow\rangle$ and $\sigma^x|\uparrow / \downarrow\rangle = |\downarrow / \uparrow\rangle$.) The stabilizers commute with each other and the Hamiltonian thus block diagonalizing the latter. On a $d\times d$ lattice of plaquettes $d^2 - 1$ plaquette operators are linearly independent (e.g. it is not possible to have a single -1 eigenvalue with all other $+1$) and correspondingly for the vertex operators. With $2d^2$ physical qubits and $2d^2 - 2$ stabilizers the size of each block is $2^{2d^2}/2^{2d^2-2} = 4$, corresponding in particular to a ground state which is 4-fold degenerate. These are the states that will serve as the logical qubits. (More precisely, given the 4-fold degeneracy it is a qudit or base-4 qubit.)

To derive the ground state consider first the plaquette operator in the σ^z-basis; clearly a ground state must have an even number of each spin-up and spin-down on every plaquette to be a $+1$ eigenstate of each plaquette operator. Let's consider the state with all spin-up $|\uparrow\uparrow\uparrow\cdots\rangle$; acting with a vertex operator on this flips all the spins around the vertex (see Fig. 1b) giving a state still in ground state sector of the plaquette operators as an even number of spins are flipped on the plaquettes surrounding the vertex. (As is also clear from the fact that all the stabilizer operators commute.) The $+1$ eigenstate of that particular vertex operator is thus the symmetric superposition of the two states. A convenient way to express the operation of one or several adjacent vertex operators is in turns of loop traversing the flipped spins. Such loops (fig. 1b-c) generated from products of vertex operators will always be topologically trivial loops on the surface of the torus since they are just constructed by merging the local loop corresponding to a single vertex operator. Successively acting with vertex operators on the states generated from the original $|\uparrow\uparrow\uparrow\cdots\rangle$ we realize that the ground state is simply the symmetric superposition of all states that are generated from this by acting with (trivial) loops $|\mathrm{GS}_0\rangle = \sum_{i\in\text{all trivial loops}} \mathrm{loop}_i|\uparrow\uparrow\uparrow\cdots\rangle$.

To generate the other ground states we consider the operators X_1 and X_2 (Fig. 1d) which are prod-

ucts of σ^x corresponding to the two non-trivial loops that wind the torus. (Deformations of these loops just correspond to multiplication by trivial loops and is thus inconsequential.) Correspondingly there are non-trivial loops of σ^z operators Z_1 and Z_2. The four ground states are thus the topologically distinct states $\{|\mathrm{GS}_0\rangle, X_1|\mathrm{GS}_0\rangle, X_2|\mathrm{GS}_0\rangle, X_2X_1|\mathrm{GS}_0\rangle\}$ distinguished by their eigenvalues of Z_1 and Z_2 being ± 1. For a torus with $d\times d$ plaquettes there are $2d^2$ physical qubits and the code distance, i.e. minimum length of any logical operator (X_i or Z_i), is d.

II.1.1. Error correction

Errors in the physical qubits will take the state out of the ground state sector and thereby mask the encoded state. The task of the error correction procedure is to move the system back to the ground state sector without inadvertently performing a logical operation to change the logical qubit state. A σ^x error on a physical qubit corresponds to a bit-flip error. On the surface code this gives rise to a pair of defects (a.k.a. quasiparticles or anyons) in the form of neighboring plaquettes with -1 eigenvalues of the plaquette stabilizers. Similarly a σ^z error corresponds to a phase-flip error which gives rise to a pair of neighboring -1 defects on two vertices. A $\sigma^y = i\sigma^x\sigma^z$ simultaneously creates both types of defects. The most natural error process is to assume that x, y, z errors occur with equal probability, so called depolarizing or correlated noise. This however requires to treat correlations between x and z errors and the simpler uncorrelated noise model is often used, which is what we will consider in this work. Here x and z errors occur independently with probability p whereas y errors occur with probability p^2. Correcting independent x and z errors is completely equivalent (with defects either on plaquettes or on vertices) and it is therefore sufficient to formulate an error correcting algorithm for one type of error. In this work we will consider bit-flip errors and corresponding plaquette defects. Regardless of noise model and type of error an important aspect of the error correction of a stabilizer formalism is that the microscopic entanglement of the logical cubit states or its excitations does not have to be considered explicitly as errors act equivalently on all pure states that belong to the same stabilizer sector.

A crucial aspect of quantum error correction is that the actual bit-flip errors cannot be measured without collapsing the state into a partial basis and destroying the qubit. What can be measured without destroying the logical qubit are the stabilizers, i.e. for bit-flip error the ± 1 eigenvalue of the plaquette operators. The complete set of incorrect (-1) plaquettes makes up the syndrome of the state. The complete set of bit-flip errors will be produce a unique syndrome as the end-points of strings of bit-flip errors. The converse however is not true, which is what makes the task challenging. In order to do the error correction we need to suggest a number of physical bits

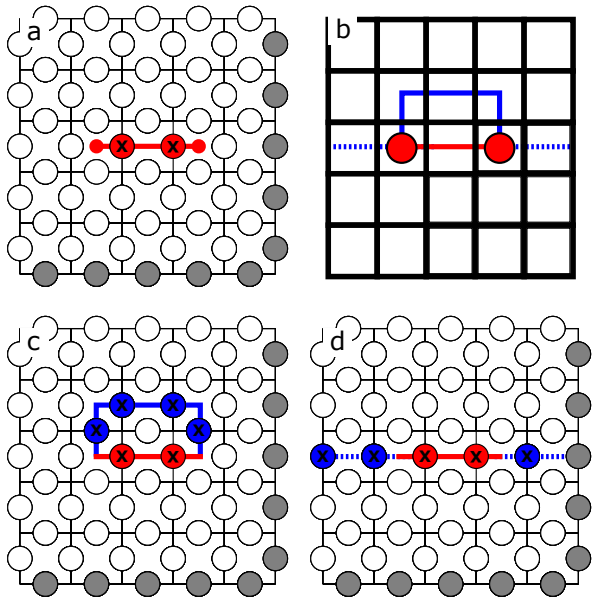

FIG. 2. Bit-flip errors (red 'X') and possible error correction bit-flips (blue 'X'). (a) Two neighboring errors and the corresponding error chain (red line) and syndrome (red dots). (b) Visualized in terms of the syndrome with error chain and two possible correction chains (blue) as expressed explicitly in (c) and (d). The error chain plus the correction chain in (d) constitutes a non-trivial loop and a logical bit-flip operation (as in Figure 1d), thus a failed error correction, in contrast to the trivial loop in (c).

that should be flipped in order to achieve the pair-wise annihilation of the defects of the syndrome. Consider a single pair of defects which have been created by a particular chain of errors. (See Figure 2.) The error correction needs to suggest a correction string connecting the two defects. If this is done properly the correction string and the error string form a trivial loop, thus returning the qubit to the original state. If instead the correction string and the error string together make up a non-trivial loop that winds the torus we have eliminated the error syndrome but changed the state of qubit (corresponding to a logical bit-flip), thus failed the task of correcting the error.

For the uncorrelated noise model it can be shown, by mapping to the random bond Ising model, that for $d \to \infty$ there is a critical threshold $p_c \approx 0.11$ below which the most probable correction chains to complement the error chain will with certainty form trivial loops, while for $p > p_c$ non-trivial loops occur with finite probability.[21] For a finite system, the sharp transition is replaced by a cross-over, as seen in Figure 7, where for increasing d the fraction of successful error correction evolves progressively towards 1 for $p < p_c$, and to 1/4 (thus completely unpredictable) for $p > p_c$.

For the uncorrelated noise model on the torus the most likely set of error chains between pairs of defects which is consistent with a given syndrome would be one that corresponds to the smallest number of total bit flips, i.e. the

shortest total error chain length. Thus, a close to optimal algorithm for error correction for this system is the Minimum Weight Perfect Matching (MWPM) algorithm[23]. (This algorithm is also near optimal for the problem with syndrome errors as long as it is still uncorrelated noise[21, 24].) The MWPM algorithm for the perfect syndrome corresponds to reducing a fully connected graph, with an even number of nodes and with edges specified by the inter-node distances, to the set of pairs of nodes that minimize the total edge length. This algorithm can be implemented efficiently[35] and we will use this as the benchmark of our RL results. In fact, as we will see, the RL algorithm that we formulate amounts to solving the MWPM problem. In this sense the work presented in this paper is to show the viability of the RL approach to this problem with the aim for future generalizations to other problems where MWPM is sub-optimal, such as the depolarizing noise model, or the corresponding Surface code on systems without periodic boundary conditions.

II.2. Q-learning

Reinforcement learning and Q-learning is described in detail for example the excellent book by Sutton and Barto[4], and here we will only recapitulate in an informal way the main and for us most pertinent details.

Reinforcement learning is a method to solve the problem of finding an optimal policy of an agent acting in a system where the actions of the agent causes transitions between states of the system. The policy $\pi(s, a)$ of an agent describes (probabilistically perhaps) the action a to be taken by the agent when the system is in state s. In our case the state will correspond to a syndrome, and an action to moving a defect one step. The optimal policy is the one that gives the agent maximal return (cumulative discounted reward) over the course of its interaction with the system. Reward r_{t+1} is given when the system transitions from state $s_t \to s_{t+1}$ such that the return starting at time t is given by $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots$. Here $\gamma \leq 1$ is the discounting factor that quantifies how we want to value immediate versus subsequent reward. As will be discussed in more detail, in the work presented in this paper a constant reward $r = -1$ will be given for each step taken, so that in practice the optimal policy will be the one that minimizes the number of actions, irrespectively of the value of γ. (Although in practice, even here the value of γ can be important for the convergence of the training.)

One way to represent the cumulative reward depending on a set of actions and corresponding transitions is by means of an action-value function, or Q-function. This function $Q(s, a)$ quantifies the expected return when in state s taking the action a, and subsequently following some policy π. In one-step Q-learning we quantify Q according to $Q(s, a) = r + \gamma \max_{a'} Q(s', a')$, with $s \xrightarrow{a} s'$, which corresponds to following the optimal policy according to our current estimate of Q. In order to learn the value of the Q-function for all states and actions we should explore the full state-action space, with the policy given by taken action a according to $\max_a Q(s, a)$ eventually guaranteed to converge to the optimal policy. However, an unbiased exploration gets prohibitively expensive and it is therefore in general efficient to follow an ϵ-greedy policy which with probability $(1 - \epsilon)$ takes the optimal action based on our current estimate of $Q(s, a)$ but with probability ϵ takes a random action. From what we have learned by this action we would update our estimate for Q according to

$$Q(s, a) \leftarrow Q(s, a) + \alpha[(r + \gamma \max_{a'} Q(s', a')) - Q(s, a)], \quad (2)$$

where $\alpha < 1$ is a learning rate. This procedure is then a trade-off between using our current knowledge of the Q function as a guide for the best move to avoid spending extensive time on expensive moves but also exploring to avoid missing out on rewarding parts of the state-action space.

II.2.1. Deep Q-learning

For a large state-action space it is not possible to store the complete action-value function. (Disregarding symmetries, for a $d \times d$ system with N_S defects, the state space has size $\binom{d^2}{N_S}$, $\sim 10^{13}$ for $p \approx 10\%$ and $d = 7$.) In deep Q-learning[7], the action-value function is instead represented by a deep neural network with the input layer corresponding to some representation of a state and the output layer corresponding to the value of the possible actions. The idea is that similarities in the value of different regions of the state-action space may be stored in an efficient way by the deep network. Parametrizing the Q-function by means of neural network we write $Q(s, a, \theta)$, where θ represents the complete set of weights and biases of the network. (We use a convolutional network with $\sim 10^6$ parameters for the $d = 7$ problem.) As outlined in more detail in the following sections the latter can be trained using supervised learning based on a scheme similar to one step Q-learning.

III. RL ALGORITHM

The decoder presented in this paper is a neural network-based agent optimized using reinforcement learning to observe toric code syndromes and suggesting recovery chains for them step by step. The agent makes use of a deep convolutional neural network to approximate Q values of actions given a syndrome. We will refer to this network as the Q network.

In a decoding session, a syndrome S corresponding to the coordinates of N_S defects e_i $(i = 1, ..., N_S)$ is fed to the algorithm as input. The syndrome is the state of the system as visible to the agent. The syndrome at any time step is that generated by accumulated actions

of the agent on the syndrome given by the initial random distribution of bit-flips. There is also a hidden state corresponding to the joint set of initial and agent flipped qubits. After the complete episode resulting in a terminal state with an empty syndrome, an odd number of non-trivial loops (in either X_1 or X_2) indicates a failed error correction. In the algorithm used in this work however, the success/fail information does not play any explicit role in the training, except as external verification of the performance of the agent. Instead reward $r = -1$ is given at every step until the terminal state regardless of whether the error correcting string(s) gave rise to an unwanted logical operation. Taking the fewest number of steps to clear the syndrome is thus the explicit target of the agent, corresponding to actuating the MWPM algorithm. (An alternative formulation with different dependence on γ would be to reward $+1$ at the terminal step.)

It would seem very natural to base the RL reward scheme on the success/failure information from the hidden state. However, we found it difficult to converge to a good agent based on this, for the following reason: given a particular starting syndrome, consistent with a distribution of different error strings, most of these are properly corrected by the MWPM algorithm whereas a minority are not. As the syndrome is all that the agent sees, it has no chance to learn to distinguish between these two classes, thus trying to use it for training will only obscure the signal. Nevertheless, for future more advanced tasks, such as dealing with depolarizing noise, and to aim for super-MWPM performance it will be necessary to explore the use of the fail/success information for the reward scheme despite the quite weak training signal.

III.1. State-space formulation

Due to the periodic boundary conditions of the code, the syndrome can be represented with an arbitrary plaquette as its center. Centering a defect e_i, we define the *perspective*, P_i, of that defect, consisting of the relative positions of all other defects in the syndrome. The set of all perspectives given a syndrome we define as an *observation*, O, as exemplified in Figure 3. (The syndrome, observation and perspective all contain equivalent information but represented differently.) The agent will be given the option of moving any defect one plaquette in any direction (left, right, up, or down), corresponding to performing a bit flip on one of the physical qubits enclosing the plaquette containing the defect. Clearly the total number of available actions varies with the number of defects, which is inconvenient if we want to represent the Q-function in terms of a neural network. In order for the Q network to have a constant-sized output regardless of how many defects are present in the system, each perspective in the observation is instead sent individually to the Q network. Thus, $Q(P, a, \theta)$ represents

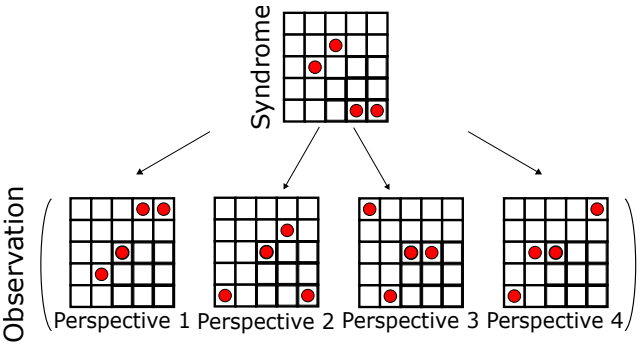

FIG. 3. State formulation. The toric code syndrome, defines an "observation" that contains the centralized "perspectives" for each defect.

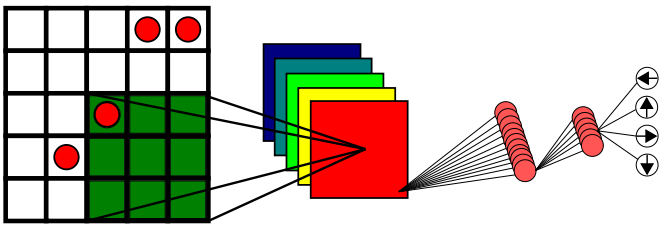

FIG. 4. Structure of the deep Q-network. The input layer is a $d \times d$ matrix corresponding to the "perspective" P, of one defect of the syndrome. (Using translational symmetry on the torus, any defect can be placed at the center.) The output layer gives the action value $Q(P, a, \theta)$ of moving the central defect to any of the four neighboring plaquettes $a = U, D, R, L$, given the current training state of network parameters θ. The hidden layers consist of a convolutional layer (of which a 3×3 filter is indicated on the input layer) and several fully connected layers. (For details, see Appendix.) Successively scanning all defects using the same network gives the full action value function of the syndrome.

the value of moving the central defect $a = L, R, U, D$, given the positions of all other defects specified by the perspective P, for network parameters θ. The network with input and output is represented graphically in Figure 4. The full Q-function corresponding to a syndrome is given by $\{Q(P, a, \theta)\}_{P \in O}$. When the Q value of each action for each defect has been obtained, the choice of action and defect is determined by a greedy policy. The new syndrome is sent to the algorithm and the procedure is repeated until no defects remain.

III.2. Training the neural network

Training of the decoder agent was done using the Deep Q Network (DQN) algorithm [7]. This algorithm utilizes the technique of experience replay in which the experience acquired by the agent is stored as transition tuples in a memory buffer. When updating the Q network (given by parameters θ), a batch of random samples is drawn from this memory buffer. By taking random samples of

experience, the temporal correlation of the data is minimized, resulting in a more stable training procedure of the neural network. To further increase the stability of the training, the DQN algorithm makes use of a target Q network (with parameters θ_T) to compute update targets. The target Q network is periodically synchronized with the updated Q network.

Algorithm 1 Training the reinforcement learning agent decoder

while syndrome defects remain **do**
  Get observation O from syndrome ▷ See figure 3
  Calculate $Q(P, a, \theta)$ using Q-network for all perspectives $P \in O$.
  Choose which defect e to move with action a using ϵ-greedy policy
  $P \leftarrow$ perspective of defect e
  Perform action a on defect e
  $r \leftarrow$ reward from taking action a on defect e
  $O' \leftarrow$ observation corresponding to new syndrome
  Store transition tuple $T = (P, a, r, O')$ in memory buffer
  Draw a random sample of transition tuples
  for each transition tuple T_i in sample **do**
    Construct targets y_i using target network θ_T and reward r_i according to Eqn. 3.
  end for
  Update Q-network parameters θ
  Every n iterations, synchronize the target network with network, setting $\theta_T = \theta$
end while

A training sequence begins with an acting stage, where a syndrome is sent to the agent, which uses the Q network Q to suggest a defect perspective, P, and an action, a. An ϵ-greedy policy is used by the agent, meaning that it will suggest the action with the highest Q-value with probability $(1-\epsilon)$. Otherwise a random action is suggested. The action is performed on the defect, e, corresponding to P, resulting in a reward, r, and a new observation, O', derived from the resulting syndrome. The whole transition is stored as a tuple, $T = (P, a, r, O')$, in a memory buffer. After this, the training sequence enters the learning stage using stochastic gradient descent. First, a random sample of transitions, $\{T_i = (P_i, a_i, r_i, O_i')\}_{i=1}^{N}$, of a given batch size, N, is drawn with replacement from the memory buffer. (Here the discrete C_4 rotational symmetry of the problem is enforced by including all four rotated versions of the same tuple.) The training target value for the Q-network is given by

$$y_i = r_i + \gamma \max_{P' \in O_i'; a'} Q(P', a', \theta_T)\,, \tag{3}$$

where γ is the discount factor and where the more slowly evolving target network parametrized by θ_T is used to predict future cumulative award. After this, gradient descent is used to minimize the discrepancy between the targets of the sample and the Q network predictions for it, upgrading the network parameters schematically according to $-\nabla_\theta \sum_i (y_i - Q(P_i, a_i, \theta))^2$. A new training sequence is then started, and with some specified rate, the weights of the target Q network are synchronized with the Q network. A pseudocode description of the procedure is presented in algorithm 1 and an illustration of the different components and procedures of the training algorithm and how they relate to each other is found in Figure 5.

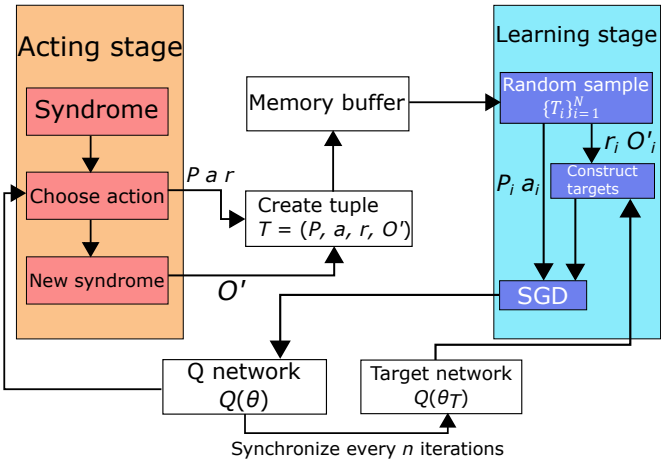

FIG. 5. Flowchart of the training procedure. A learning sequence consists of an acting stage followed by a learning stage. (P, a, r) is (perspective,action,reward) corresponding to what defect is acted on with what action and giving what reward. O' is the observation (see Fig. 3) corresponding to the resulting syndrome. A pseudocode representation is given in Algorithm 1. (SGD is Stochastic Gradient Descent.)

IV. RESULT

Data sets with a fixed error rate of 10% were generated to train the agent to operate on a code of a specified size. The syndromes in a data set is fed one at a time to the agent, which operates on it until no errors remain. The data sets also contain information about the physical qubit configuration (the hidden state) of the lattice, which (as discussed in section III) is used to check the success rate of the decoder. This is compared to the performance of the MWPM decoder on the same syndromes [35]. The initial training of the agent is illustrated in Figure 6 for lattice size $d \times d$, with $d = 3, 5, 7$.

The proficiency of the well converged agents are shown in figure 7 and compared to the MWPM performance. Given our specified reward scheme, which corresponds to using as few operations as possible, we achieve near optimal results with a performance which is close to that of the MWPM decoder. For larger system sizes $d \geq 9$ we were not successful at converging to close to MWPM performance. We expect that this can be resolved by using a larger neural network and parallelizing the exploration, to be explored in future work.

As a demonstration of the operation of the trained agent and the corresponding Q-network we present

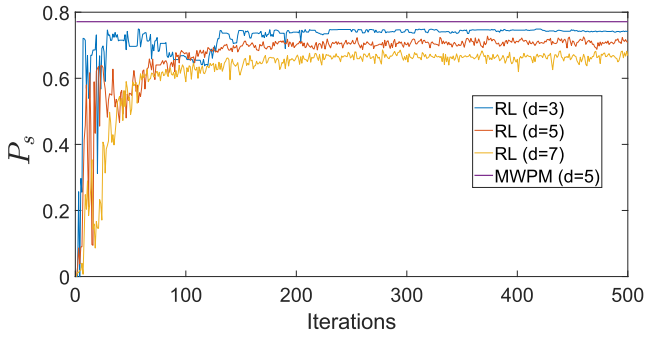

FIG. 6. Early training convergence of the Q network agent. Success rate P_s versus number of iterations. One iteration corresponds to annihilating all the defects of a single syndrome. (The very early below 1/4 success rate is an artifact of using a max count for the number of error correcting steps for the validation.)

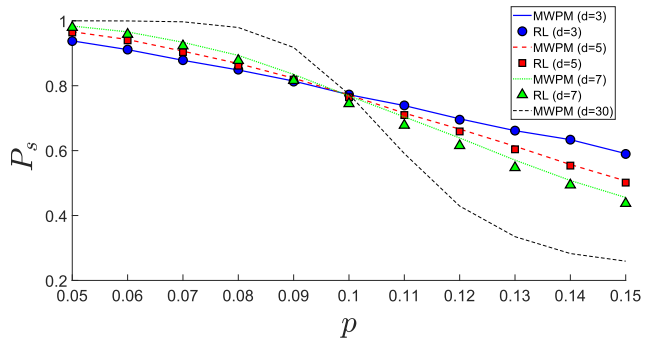

FIG. 7. Error correction success rate P_s of the converged agents versus bit-flip error rate p, for system size $d = 3, 5, 7$, and compared to the corresponding results using MWPM. (The MWPM decoder for $d = 30$ is included as a reference for the approach to large d.)

in Figure 8 the action values $Q(S, a)$ for two different syndromes. (As discussed previously, $Q(S, a) = \{Q(P, a, \theta)\}_{P \in O}$, where O is the observation, or set of perspectives, corresponding to the syndrome S.) The size of the arrows are proportional to the discounted return R of moving a defect one initial step in the direction of the arrow and then following the optimal policy. In Fig. 8a, the values are written out explicitly. The best (equivalent) moves have a return $R = -3.57$ which corresponds well to the correct value $R = -1-\gamma-\gamma^2-\gamma^3 = -3.62$ for following the optimal policy to annihilate the defects in four steps, with reward $r = -1$ and discount rate $\gamma = .95$. Figure 8b shows a seemingly challenging syndrome where the fact that the best move does not correspond to annihilating the two neighboring defects is correctly captured by the Q-network.

One interesting aspect of the close to MWPM performance of the fully trained agent is the ability of the Q-network to suggest good actions independently of how many defects are in the syndrome. A 7×7 system with $p = 10\%$ would start out with a syndrome with maybe

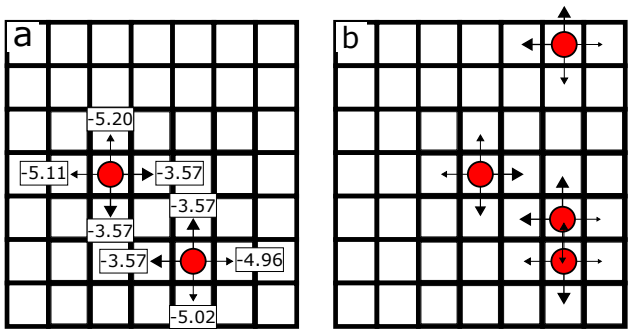

FIG. 8. Action value function produced by the Q-network for two different syndromes and code distance $d = 7$. The magnitude of the arrows indicate the expected return from taking a next step along the arrow and after that following the optimal policy. The optimal policy for the next move corresponds to the action with the biggest arrow(s). In (a) the expected return is written out explicitly, where the best moves are consistent with the constant reward of -1 per step and discounting rate $\gamma = 0.95$ used in the training.

20 defects, which is successively pair-wise reduced down to two and finally zero defects, all based on action-values given by the same Q-network (θ). The network is thus surprisingly versatile and capable, given the enormous reduction of the number of adjustable parameters compared to representing and training the full Q-value function as a list.

V. CONCLUSIONS

In conclusion, we have shown that reinforcement learning is a viable method for quantum error correction of the toric code for moderate size systems using uncorrelated bit-flip (or phase-flip) noise. By training an agent to find the shortest paths for the error correction chains we are able to achieve accuracy close to that using a Minimum Weight Perfect Matching decoder. In order to accomplish this we used the deep Q-network (DQN) formalism using a deep neural network to encode the action-value function.[6, 7] The construction also made good use of the translational invariance on the torus to be able to efficiently reduce the state space representation. For future work it will be interesting to see how the formalism generalizes to more advanced problems that include depolarizing noise, syndrome noise, as well as surface codes with boundaries. For such problems the training would certainly be more challenging but probably within reach using the machinery of deep Q-learning. The versatility of the deep Q-network should perhaps not be surprising given its successful application to other challenging problems, but it also gives good hope for using the DQN approach for the noise models where the MWPM algorithm is suboptimal.

ACKNOWLEDGEMENTS

We thank Niklas Forsström, Gustav Karlsson, and Elias Hannouch for contributing to the early stages of this work.

Appendix: Network architecture and training parameters

The reinforcement learning agent makes use of a deep convolutional neural network to approximate the Q values for the possible actions of each defect. The network (see Fig. 4) consists of an input layer which is $d \times d$ matrix corresponding to a perspective (binary input, 0 or 1, with 1 corresponding to a defect), and a convolutional layer followed by several fully-connected layers and an output layer consisting of four neurons, representing each of the four possible actions. All layers have ReLU activation functions except the output layer which has simple linear activation. The network architecture is summarized in Table I and II. We also included explicitly a count of the number of parameters (weights and biases) to emphasize the huge reduction compared to tabulating the Q-function. The latter requires of the order $\binom{d^2}{N_S}$ entries, for N_s defects, where N_s will also vary as the syndrome is reduced, with initially $N_S \sim 2pd^2$ as each isolated error creates a defect pair.

TABLE I. Network architecture d=5. FC=Fully connected

#	Type	Size	# parameters
0	Input	5x5	
1	Convolutional	512 filters; 3x3 size;	
		2-2 stride; 2048 neurons	5 120
2	FC	256 neurons	524 544
3	FC	128 neurons	32 896
4	FC	64 neurons	8 256
5	FC	32 neurons	2 080
6	FC (output)	4 neurons	132
			573 028

In Table III we list the hyperparameters related to the Q-learning and experience replay set-up, as well as the neural network training algorithm used. The full RL algorithm is coded in Python using Tensorflow and Keras for the Q-network. A single desktop computer was used, with training converging over a matter of hours (for $d = 3$) to days (for $d = 7$).

TABLE II. Network architecture d=7.

#	Type	Size	# parameters
0	Input	7x7	
1	Convolutional	512 filters; 3x3 size;	
		2-2 stride; 2048 neurons	5 120
2	FC	256 neurons	1 179 904
3	FC	128 neurons	32 896
4	FC	64 neurons	8 256
5	FC	32 neurons	2 080
6	FC (output)	4 neurons	132
			1 228 388

TABLE III. Hyperparameters

Parameter	Value
discount rate γ	0.95
reward r	-1/step; 0 at finish
exploration ϵ	0.1
max number of steps per syndrome	50
mini batch size, N	32
target network update rate	100
memory buffer size	1 000 000
optimizer	'Adam'
learning rate	0.001
beta_1	0.9
beta_2	0.999
decay	0.0

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems* (2012) pp. 1097–1105.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, "Deep learning," nature **521**, 436 (2015).

[3] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio, *Deep learning*, Vol. 1 (MIT press Cambridge, 2016).

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

[5] Gerald Tesauro, "Temporal difference learning and td-gammon," Communications of the ACM **38**, 58–68 (1995).

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller, "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602 (2013).

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

[8] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, *et al.*, "Mastering the game of go without human knowledge," Nature **550**, 354 (2017).

[9] Louis-François Arsenault, Alejandro Lopez-Bezanilla, O Anatole von Lilienfeld, and Andrew J Millis, "Machine learning for many-body physics: the case of the anderson impurity model," Physical Review B **90**, 155136 (2014).

[10] Evert PL Van Nieuwenburg, Ye-Hua Liu, and Sebastian D Huber, "Learning phase transitions by confusion," Nature Physics **13**, 435 (2017).

[11] Juan Carrasquilla and Roger G Melko, "Machine learning phases of matter," Nature Physics **13**, 431 (2017).

[12] Giuseppe Carleo and Matthias Troyer, "Solving the quantum many-body problem with artificial neural networks," Science **355**, 602–606 (2017).

[13] Xun Gao and Lu-Ming Duan, "Efficient representation of quantum many-body states with deep neural networks," Nature communications **8**, 662 (2017).

[14] Michael A Nielsen and Isaac Chuang, "Quantum computation and quantum information," (2002).

[15] Barbara M Terhal, "Quantum error correction for quantum memories," Reviews of Modern Physics **87**, 307 (2015).

[16] Alexey A Melnikov, Hendrik Poulsen Nautrup, Mario Krenn, Vedran Dunjko, Markus Tiersch, Anton Zeilinger, and Hans J Briegel, "Active learning machine learns to create new quantum experiments," Proceedings of the National Academy of Sciences , 201714936 (2018).

[17] Thomas Fösel, Petru Tighineanu, Talitha Weiss, and Florian Marquardt, "Reinforcement learning with neural networks for quantum feedback," Phys. Rev. X **8**, 031084 (2018).

[18] Marin Bukov, Alexandre G. R. Day, Dries Sels, Phillip Weinberg, Anatoli Polkovnikov, and Pankaj Mehta, "Reinforcement learning in different phases of quantum control," Phys. Rev. X **8**, 031086 (2018).

[19] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd, "Quantum machine learning," Nature **549**, 195 (2017).

[20] A Yu Kitaev, "Fault-tolerant quantum computation by anyons," Annals of Physics **303**, 2–30 (2003).

[21] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill, "Topological quantum memory," Journal of Mathematical Physics **43**, 4452–4505 (2002).

[22] Ryan Sweke, Markus S Kesselring, Evert PL van Nieuwenburg, and Jens Eisert, "Reinforcement learning decoders for fault-tolerant quantum computation," arXiv preprint arXiv:1810.07207 (2018).

[23] Jack Edmonds, "Paths, trees, and flowers," Canadian Journal of mathematics **17**, 449–467 (1965).

[24] Austin G Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average o(1) parallel time," Quantum Information and Computation **15**, 0145–0158 (2015).

[25] Guillaume Duclos-Cianci and David Poulin, "Fast decoders for topological quantum codes," Physical review letters **104**, 050504 (2010).

[26] Michael Herold, Earl T Campbell, Jens Eisert, and Michael J Kastoryano, "Cellular-automaton decoders for topological quantum memories," npj Quantum Information **1**, 15010 (2015).

[27] Aleksander Kubica and John Preskill, "Cellular-automaton decoders with provable thresholds for topological codes," arXiv preprint arXiv:1809.10145 (2018).

[28] Stefan Krastanov and Liang Jiang, "Deep neural network probabilistic decoder for stabilizer codes," Scientific reports **7**, 11003 (2017).

[29] Savvas Varsamopoulos, Ben Criger, and Koen Bertels, "Decoding small surface codes with feedforward neural networks," Quantum Science and Technology **3**, 015004 (2017).

[30] Paul Baireuther, Thomas E O'Brien, Brian Tarasinski, and Carlo WJ Beenakker, "Machine-learning-assisted correction of correlated qubit errors in a topological code," Quantum **2**, 48 (2018).

[31] Nikolas P Breuckmann and Xiaotong Ni, "Scalable neural network decoders for higher dimensional quantum codes," Quantum **2**, 68 (2018).

[32] Christopher Chamberland and Pooya Ronagh, "Deep neural decoders for near term fault-tolerant experiments," arXiv preprint arXiv:1802.06441 (2018).

[33] Xiaotong Ni, "Neural network decoders for large-distance 2d toric codes," arXiv preprint arXiv:1809.06640 (2018).

[34] Figures in this section were inspired by the lecture notes: Dan Browne, "Topological codes and computation a lecture course given at the university of innsbruck," http://bit.do/topological (2014).

[35] Vladimir Kolmogorov, "Blossom v: a new implementation of a minimum cost perfect matching algorithm," Mathematical Programming Computation **1**, 43–67 (2009).