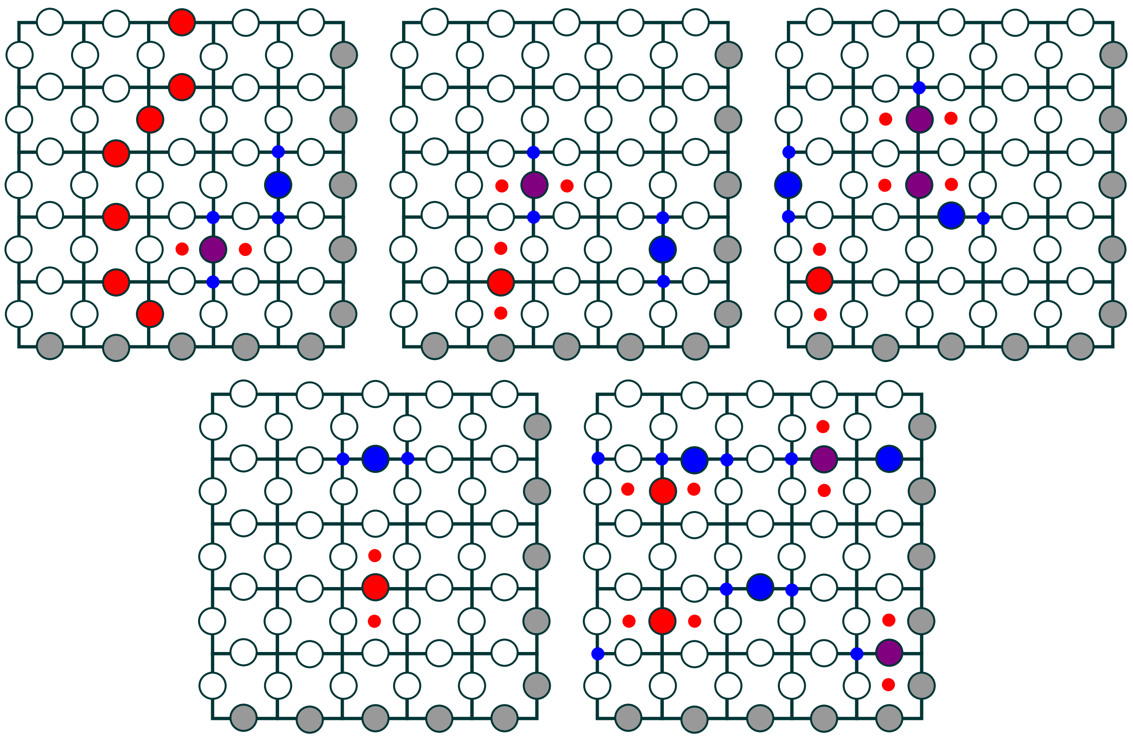
CHALMERS

UNIVERSITY OF TECHNOLOGY



Error correction for depolarising noise on a quantum system using deep RL

Master's thesis in Complex Adaptive Systems

David Fitzek
Mattias Eliasson

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2019

Error correction for depolarising noise on a quantum system using deep RL

DAVID FITZEK
MATTIAS ELIASSON

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2019

Error correction for depolarising noise on a quantum system using deep RL
DAVID FITZEK
MATTIAS ELIASSON

Supervisor: Mats Granath, Department of Physics, Gothenburg University
Examiner: Mats Granath, Department of Physics, Gothenburg University

Department of Physics
Division of Division name
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Several examples of error configurations, syndromes, of the toric code. The circles placed in the grid represents individual qubits and the colors represent different types of errors. Blue color represents a σ^z-error, red represents a σ^x-error and purple represents a σ^y-error. The smaller colored circles represents defects produced by the qubit error chains.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2019

Abstract

We implement a quantum error correction algorithm for the depolarized noise model on the topological toric code using deep reinforcement learning. An action-value Q-function encodes the discounted value of applying any of the three pauli operators (σ^x, σ^y, σ^z) to a specific qubit given the entire set of excitations on the torus. The Q-network is defined by a convolutional neural network (CNN), with one fully connected layer at the end. Considering the translational invariance of the torics code we can center every qubit and therefore naturally simplify the state space representation independently of the number of excitation pairs. We train the agent using experience replay and store the state from the algorithm to use it for mini-batch updates of the Q-network. We conclude that this approach, considering all three pauli operators, outperforms the Minimum Weight Perfect Matching (MWPM) algorithm and for small error rates is close to the asymptotic solution for very low error probabilites.

Contents

Abbreviations

MWPM	Minimum Weight Perfect Matching
CNN	convolutional neural network
ReLU	rectified linear unit
SGD	stochastic gradient descent
RL	reinforcement learning
NN	neural network

Symbol directory

A_v	Vertex operator
B_p	Plaquette operator
d	System size
D	Replay memory
E	Energy of the system
$\mathcal{H}$	Hilbert space
$\mathcal{L}$	Lattice
N_{per}	Number of perspectives
N	Size of replay memory
O	Observation
P	Perspective
p_L	Error correction fail rate
p_s	Error correction success rate
S	Syndrome
σ^x	Pauli x-operator
σ^y	Pauli y-operator
σ^z	Pauli z-operator
θ	Weights for policy Q-network
θ_t	Weights for target Q-network

Chapter 1

Introduction

Quantum mechanics inherits the potential to perform computational simulations and information processing tasks much faster than a classical computer. To outperform classical computers quantum systems must manipulate hundreds of qubits to execute a program and preserve the entangled quantum state over a long duration. To ensure that, it is necessary to introduce quantum error correction mechanisms. The basic principle behind this procedure is to facilitate the concept of redundancy, where a number of physical qubits is projected in a logical qubit. We are then able to perform error correction protocols and preserve the encoded information of the system [1].

For this work we make use of the torics code (Kitaev's surface code with periodic boundary conditions), which is a stabilizer formalism that projects a large number of physical qubits into a smaller number of entangled qubits. It is possible to perform the error correction protocols on classical computers and utilizing machine learning algorithms to secure the quantum information encoded by the physical qubits.

We have seen astonishing developments in the field of machine learning took place in all its different learning approaches, such as, supervised, unsupervised and reinforcement learning. Supervised learning techniques are accelerated by the use of CNNs and therefore leading to better performances and an overall improvement of its versatility, given big annotated data sets [4].

Moreover, reinforcement learning (RL) received a lot of attention due to some recent advances in the successful connection of function approximators, neural networks (NNs), embedded in the paradigm of RL. Mnih et al. [2] showed that the combination of deep learning and RL is a powerful tool to solve problems such as computer games. This set of problems is especially interesting, because there is no known solution (annotated dataset), but rather a dynamic environment through which an agent learns how to act for an optimal outcome.

RL is a possible tool to perform error correction codes on a classical computer. Recent applications of RL and quantum error correction have shown that it is a viable method. Andreasson and others have shown that for the uncorrelated error model the RL agent behaves as the MWPM algorithm if trained on solving the syndrom with the minimum amount of steps [8]. Moreover, the paper [9] proposes an approach to solve the correlated noise problem, considering bit and phase flip errors.

The aim of this thesis is to apply deep reinforcement learning to solve defects on the toric code produced by correlated errors on the qubits. More specifically a deep reinforcement learning algorithm, using the deep Q-learning technique [2], was developed and trained. The resulting algorithm will then be compared to the classical Minimum Weight Perfect Matching, MWPM, algorithm. The aim is to produce an algorithm that performs better than MWPM. The algorithm is developed for the toric code with defects generated by correlated errors. Due to hardware performance limitations the work has been limited to grid sizes, d, smaller than 9.

Several questions needs to be answered to see if the aim of the thesis have been reached. Can reinforcement learning be used to develop an algorithm that solves the uncorrelated errors on the toric code? Moreover, whether it is possible to develop and train a RL based algorithm that outperforms the MWPM algorithm and how the performance differs for varying grid sizes of the toric code.

The outline of the thesis is the following. We cover the basics of a stabilizer formalism, the torics code. Moreover, we introduce the concept of RL and its basic structure in chapter 3. In chapter 4 we describe the methods that we used to combine RL and quantum error correction. The results and a rigorous discussion is presented in chapter 5 and chapter 6 respectively. We show that we successfully trained an agent up to system size 7 and outperform the MWPM algorithm.

Chapter 2

Toric code

The Toric code is a model embedded on a two dimensional squared lattice grid. Considering periodic boundary conditions the toric code can be nested on a non-trivial surface such as a torus. The ground state degeneracy is 4^g, where g is the genus of the surface [5]. In this chapter we cover the model as it is initially described, its ground states and excitations and the rules for fusing these excitations together.

2.1 The Toric Code Model

We consider a lattice $\mathcal{L}$ embedded on an arbitrary two-dimensional surface. For this work we make use of a $d \times d$ lattice with periodic boundary conditions forming a torus. On every edge we construct a bi-dimensional local Hilbert space $\mathcal{H}_i$. The Hilbert space of the model is a tensor product of the local Hilbert spaces spanned by each qubit $\mathcal{H} = \bigotimes_{i \in \mathcal{L}} \mathcal{H}_i$. The lattice contains $N_v = d^2$ vertices and $N_p = d^2$ plaquettes. We define for each vertex v and plaquette p the following operators:

$$A_v = \bigotimes_{i \in star(v)} \sigma_i^x = \sigma_{i_1}^x \otimes \sigma_{i_2}^x \otimes \sigma_{i_3}^x \otimes \sigma_{i_4}^x, \qquad B_p = \bigotimes_{j \in square(p)} \sigma_j^z = \sigma_{j_1}^z \otimes \sigma_{j_2}^z \otimes \sigma_{j_3}^z \otimes \sigma_{j_4}^z \tag{2.1}$$

The index $i \in star(v)$ runs over all four edges around a specific vertex v and $j \in square(p)$ runs over all the edges enclosing the plaquette p on the lattice (see figure 2.1). The Pauli matrices and the identity matrix for completeness are given by:

$$\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma^y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.2}$$

The vertex and plaquette operator act over the entire Hilbert space $\mathcal{H}$, but trivially (acting with the identity operator $\mathbb{1}$) on every edge that is not part of v or p [5]. We will later use these operators for parity measurements of adjacent qubits and generate a syndrome map.

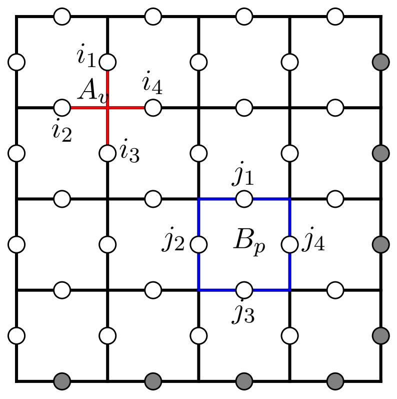


Figure 2.1: Graphical representation of the toric code. On every edge there are physical qubits located and represented as circles. The grey marked qubits serve as an indication for the periodic boundary condition. The vertex operator A_v and plaquette operator B_p are visualized.

2.2 Algebra of Operators

The underlying algebra of the plaquette and vertex operators can be understood by studying the commutation and anticommutation relation of the Pauli operators [5].

$$[\sigma^a, \sigma^b] = 2i\epsilon_{abc}\sigma^c, \tag{2.3}$$

$$\{\sigma^a, \sigma^b\} = 2\delta_{ab}\mathbb{1} \tag{2.4}$$

From this we can deduce the following relations,

$$[\sigma^x, \sigma^x] = 0 \tag{2.5}$$

$$[\sigma^z, \sigma^z] = 0 \tag{2.6}$$

$$\sigma^x\sigma^z = -\sigma^z\sigma^x \tag{2.7}$$

$$\sigma^x\sigma^y = -\sigma^y\sigma^x \tag{2.8}$$

$$\sigma^z\sigma^y = -\sigma^y\sigma^z \tag{2.9}$$

Consider a vertex and plaquette operator sharing two edges as in figure (see figure 2.2). The commutation relation between A_v and B_p can be show as

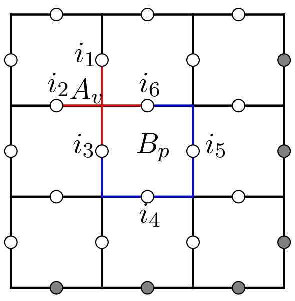


Figure 2.2: Two adjacent plaquette B_p and vertex operators A_v. The constellation of the vertex and plaquette operator is arbitrary as long as they share two edges.

$$A_vB_p = \sigma^x_{i_1} \otimes \sigma^x_{i_2} \otimes \sigma^x_{i_3}\sigma^z_{i_3} \otimes \sigma^x_{i_4} \otimes \sigma^x_{i_5} \otimes \sigma^x_{i_6}\sigma^z_{i_6} \bigotimes_{j\notin v,p} \mathbb{1} \tag{2.10}$$

$$= ... \otimes -\sigma^z_{i_3}\sigma^x_{i_3} \otimes ... \otimes -\sigma^z_{i_6}\sigma^x_{i_6} \otimes ... \tag{2.11}$$

$$= (-1)^2 B_pA_v \tag{2.12}$$

We used the commutation relation 2.7 and introduced the minus signs in the second line. Two adjacent vertex and plaquette operators always have two shared edges. Therefore, we can conclude

$$[A_v, B_p] = 0. \tag{2.13}$$

Due to the involuntory nature of the Pauli matrices ($(\sigma^x)^2 = (\sigma^y)^2 = (\sigma^z)^2 = \mathbb{1}$) it is easy to see that A_v^2 and B_p^2 reverse the changes and leaves the grid unchanged [5].

$$A_v^2 = \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} \otimes \mathbb{1} = B_p^2 \tag{2.14}$$

Applying the same operator to two adjacent ones (either plaquette or vertex) will result in a state, where the shared edge will remain unchanged. Moreover, applying the operator on the entire lattice leaves the entire lattice invariant. We can write,

$$\prod_{\text{all } v} A_v = \bigotimes_{i\in\mathcal{L}} \mathbb{1}_i \qquad \prod_{\text{all } p} B_p = \bigotimes_{j\in\mathcal{L}} \mathbb{1}_j \tag{2.15}$$

The eigenstates of the plaquette and vertex operator are defined as follows:

$$A_v\left|\psi_v\right\rangle = a_v\left|\psi_v\right\rangle \qquad B_p\left|\psi_p\right\rangle = b_p\left|\psi_p\right\rangle \tag{2.16}$$

Due to the involuntory properties of the operators (see equation 2.14) the only eigenvalues that the operators can take are ± 1 [5] .

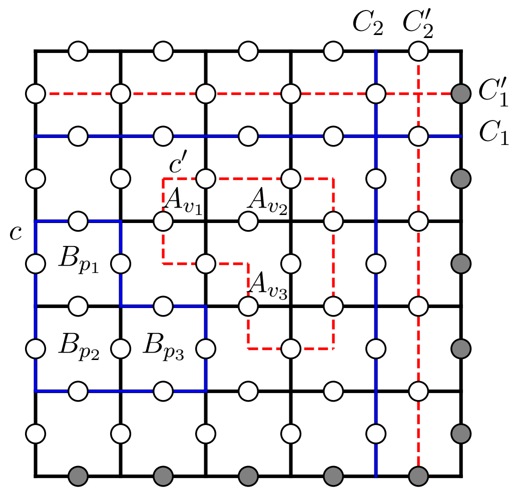


Figure 2.3: Two contractible loops and four paths are shown. The three plaquette operators B_p (solid blue line labeled as c) and the three vertex operators A_v (dotted red line labeled as c') describe the trivial loops formed by these operators. The non-contractible paths C_1 and C_1' characterize a non-trivial path in the horizontal direction. Paths C_2 and C_2' define non-contractible loops in vertical direction. Non-contractible loops cannot be represented as a combination of plaquette B_p or vertex operators A_v.

2.3 Trivial and nontrivial loops

We can describe every closed contractible loop on the dual (vertex space) and direct (plaquette space) lattice as a product of operators. We remember that, due to the involuntory nature of the Pauli operators, shared edges stay invariant. Therefore, the contractible loops seen in figure 2.3 can be described as

$$Z(c) = \bigotimes_{i\in c} \sigma_i^x = B_{p_1} B_{p_2} B_{p_3} \tag{2.17}$$

$$X(c') = \bigotimes_{j\in c} \sigma_j^z = A_{v_1} A_{v_2} A_{v_3}. \tag{2.18}$$

Considering that the lattice is embedded on a torus we can define closed loops that cannot be represented in terms of A_v and B_p. The operators describing these non-contractible loops are $X(C_1')$, $X(C_2')$, $Z(C_1)$, $Z(C_2)$ and can be seen in figure 2.3 [5]. Figure 2.4 shows the torus form of the torics code considering the property of periodicity.

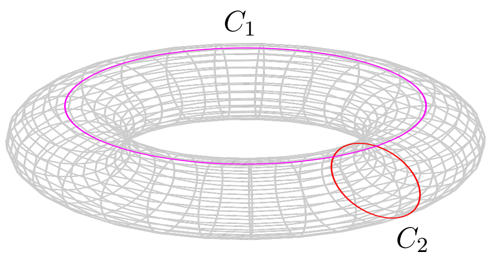


Figure 2.4: Two non-contractible loops on a torus [12].

2.4 Hamiltonian and Ground States

We define the Hamiltonian as the sum of all the plaquette and vertex operators over the lattice as

$$H = -\sum_v A_v - \sum_p B_p. \tag{2.19}$$

To find the ground state $|\Psi_0\rangle$ we need to find the condition such that the energy of the Hamiltonian is minimized (Due to the minus signs for each term we aim to maximize the eigenvalues of A_v and B_p). The lower bound on the energy is obtained when the state satisfies,

$$B_p |\Psi_0\rangle = |\Psi_0\rangle \quad \forall p, \qquad \text{and} \qquad A_v |\Psi_0\rangle = |\Psi_0\rangle \quad \forall v. \tag{2.20}$$

Therefore any state that fulfills these conditions is a ground state of the system.

Let us consider a state with all spins up $|\uparrow\uparrow\uparrow ...\rangle$. A ground state must minimize the Hamiltonian. Considering first the plaquette operators, B_p, it is evident that an even number of spin up and spin down on every plaquette will ensure an eigenvalue of $+1$ for each of these operators. This will minimize equation 2.19. However, if we introduce a pair of flipped spins we can see that adjacent vertex operators need another spin flipped and therefore only a closed loop of flipped spins fulfills the constraint in equation 2.20 regarding the vertex space.

We act with a vertex operator on the previously defined state and flip all the qubits in the range of the operator. As we have already shown in section 2.2, adjacent vertex and plaquette operators share always two edges and therefore obeying the constraint 2.20.

The operation of several adjacent vertex or plaquette operators is visualized in figure 2.3. Loops created as a product of the stabilizer operators are always trivial loops. Therefore, acting with vertex operators on the state described with all spins up $|\uparrow\uparrow\uparrow ...\rangle$ The ground state of the toric code is an equal weight superposition of all loop configurations [8].

$$|\Psi_0\rangle = \sum_{i \in \text{all trivial loops}} \text{loop}_i \, |\uparrow\uparrow\uparrow ...\rangle \tag{2.21}$$

To generate the other ground states we consider the operators that generate the nontrivial loops discussed in section 2.3. The operators $X(C'_1)$, $X(C'_2)$, $Z(C_1)$, $Z(C_2)$ and any deformation of these loops corresponding to multiplications with trivial loops define the four distinct ground states. There are four topologically distinct states defined by the nontrivial loops that can wind the torus.

$$|\Psi_0\rangle\,, \tag{2.22}$$
$$X(C'_1)\,|\Psi_0\rangle\,, \tag{2.23}$$
$$X(C'_2)\,|\Psi_0\rangle\,, \tag{2.24}$$
$$X(C'_1)X(C'_2)\,|\Psi_0\rangle \tag{2.25}$$

2.5 Elementary Excitations

The elementary excitations are a result of violating the constrains defined in equation 2.20. We know that the eigenvalues of the plaquette and vertex operator can only be ± 1, thus an elementary excitation refers to an eigenstate such that some of the operators eigenvalues are -1 [5]. Plaquette excitations are also known as a result of bit flip errors (σ^x) and vertex excitations correspondingly as phase flip errors (σ^z).

2.5.1 Vertex Excitations: Charges

We consider the following state:

$$|\Psi_i^z\rangle = \mathbb{1} \otimes \mathbb{1} \otimes ... \otimes \sigma_i^z \otimes ... \otimes \mathbb{1} \otimes \mathbb{1}\,|\Psi_0\rangle \tag{2.26}$$

Where Ψ_0 can be any of the four ground states. The resulting state is not a ground state anymore. We can deduce from the anticommutation relation of the Pauli operators (see equation 2.7) that the vertex operators do not commute with the phase shifted qubit (see blue qubit in figure 2.5).

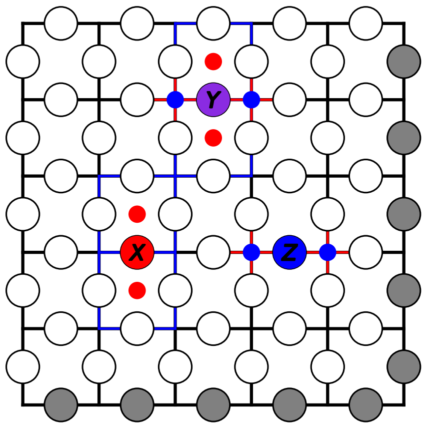


Figure 2.5: Vertex and plaquette excitations: shown are the excitations as a result of a Pauli σ^x (red colored qubit), a Pauli σ^y (purple colored qubit) and a Pauli σ^z (blue colored qubit) operator applied to a qubit of the torics code. We can see that the Pauli σ^x operator creates two excitations in the plaquette space. The Pauli σ^z operator produces two excitations in the vertex space and the Pauli σ^y operator constructs excitations in both spaces.

$$
\begin{aligned}
A_v \left|\Psi_i^z\right\rangle &= A_v \sigma_i^z \left|\Psi_0\right\rangle && (2.27)\\
&= -\sigma_i^z A_v \left|\Psi_0\right\rangle && (2.28)\\
&= -\sigma_i^z \left|\Psi_0\right\rangle && (2.29)\\
&= -\left|\Psi_i^z\right\rangle && (2.30)
\end{aligned}
$$

From the first to second line we used the anticommutation relation defined in equation 2.7. We know that the vertex operator acts trivial on the ground state, thus we can explain the simplification from line two to three [5].

We can see that the Pauli operator σ^z creates two excitations. The energy of the system is now

$$E_v = -(N_p + N_v - 2) \tag{2.31}$$

Let us now consider an open path created by e.g two adjacent Pauli operators σ^z. The Pauli operators will anti-commute with the vertex operators on the endpoints of the path and create two vertex excitations named charges. The excitations can be moved by extending the open path on the lattice. The energy of the state doesn't change since the number of vertex operators affected by the path operator doesn't change. Finally, moving the excitations toward each other results in a closed loop. This operator can be written as a product of vertex and plaquette operators and as we have already shown in section 2.4 do these loops act trivially on the ground state [5].

For our problem we also allow the σ^y operator. As we can conclude from the anticommutation relation (see equation 2.8), a σ^y error introduces as well excitations in the vertex space (see purple qubit in figure 2.5)[5].

2.5.2 Plaquette Excitations: Fluxes

The plaquette excitations are as well a violation of the constraints defined in equation 2.20. We can use a very similar reasoning as already shown in subsection 2.5.1. We know that the plaquette operator anticommutes at the end of the path operator with the σ^x operator. We can see that the excitations come always in pairs and the energy of this state is given by

$$E_v = -(N_p + N_v - 2). \tag{2.32}$$

The energy of the system does not depend on the path itself. There are many path operators that create an excited state (see figure 2.5) [5].

Similar to the vertex space the σ^y operator creates two excitations in the plaquette space (see purple qubit in figure 2.5). This is due to the anticommutation relation specified in equation 2.9 [5].

2.6 Error correction

Errors introduced to the system will alter its state and violate the ground state condition defined in equation 2.20. Therefore, the logical qubit encoded state is masked by the error and it is necessary to find a procedure to bring the system back to its previous ground state and not inadvertently generate nontrivial loops and change the logical qubit state.

A fundamental aspect of quantum error correction is that the actual 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 eigenvalues (± 1). The complete set of measurements defines the syndrome, whereas every -1 eigenvalue corresponds to an excitation either in the plaquette or vertex space [8].

Given the complete set of bit flip and phase shift errors there is one unique syndrome, however the converse is not true. Therefore, there are several possible error constellations corresponding to a specific syndrome (see figure 2.7). Moreover, considering the syndrome and due to the involuntory nature of the Pauli operators, it is a valid procedure to correct errors by suggesting the qubits that should be altered and apply the corresponding Pauli operator in order to achieve a pair wise annihilation of the excitations of the syndrome.

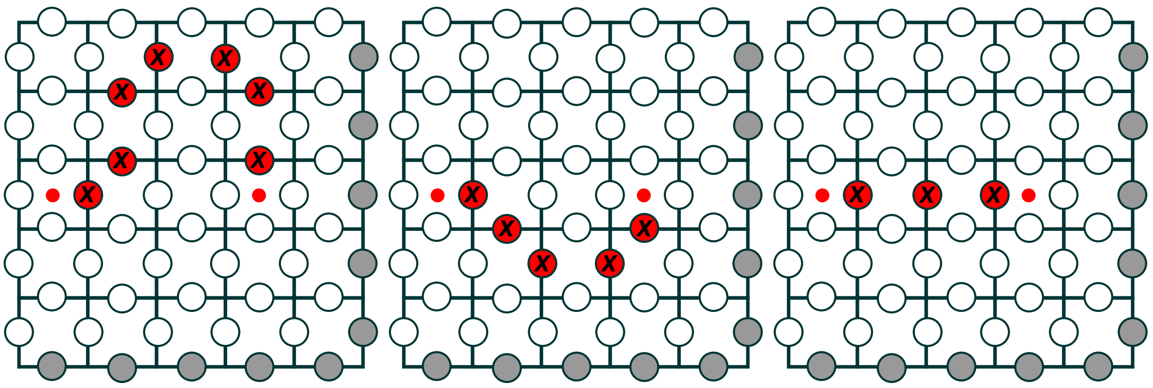


Figure 2.7: Three strings of Pauli σ^x operators correspond to the same syndrome. There is no unique mapping from the syndrome to the actual state of the system.

There are two distinct error model that are widely studied in the context of topological codes. The depolarizing noise model and the uncorrelated noise model. For the uncorrelated noise model the most likely error chain is the smallest number of total qubit flips and shifts. A close to optimal error correction algorithm is the MWPM algorithm. The core idea is to reduce a fully connected graph, with an even number of nodes to the set of pairs of nodes minimizing the total edge length [6]. We will utilize this algorithm to benchmark the results from the RL approach. For our work we consider the depolarizing noise model, therefore we expect the RL algorithm to outperform the MWPM algorithm. We examine correlations between the plaquette and vertex space [3].

Chapter 3

Reinforcement learning

Reinforcement learning is a machine learning paradigm where an agent is trying to maximize cumulative rewards given for different actions taken in an environment [11]. As the agent takes different actions and receives different rewards it will develop a policy which describes what actions to take in different states. The optimal action value function, given state s and action a,

$$Q(s_t, a_t) = \max_{\pi} \mathbf{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^n r_{t+n}] \tag{3.1}$$

describes the maximum cumulative reward for a sequence of n actions following the policy π. γ is the discount factor taking values lower than 1. This can also be expressed with the one step Q-function or the Bellman equation:

$$Q(s_t, a_t) = r + \gamma \max_{a'} Q(s_{t+1}, a'), \tag{3.2}$$

s_{t+1} and a' corresponds to the next state and action taken in the next state. The action value function, or Q-function, describes the best possible actions to take in a given state. It is the goal of the agent to learn the correct Q-function and thus learn a good policy related to a reward scheme.

This learning is done through exploration of the state space by trying different actions, as the agent enacts different actions it will iteratively update the action value function. This is commonly done with the Bellman Equation, $Q_i(s_t, a_t) = r + \gamma \max_{a'} Q(s_{t+1}, a')$, this will converge the optimal action value function for a large number of steps. As the state space can be huge it is common to use a function approximation of the Q-function, e.g. a neural network.

3.1 Deep Reinforcement learning

One way of representing the Q-function is to use a neural network, this method is called Deep Q-learning. It has been used with great success to learn an algorithm a policy to play Atari 2600 games [2]. In the algorithm, which can be seen in algorithm 1, an agent chooses ϵ-greedily the move with the highest Q-value.

The Q-values are produced by feeding the state into a neural network, the policy Q-network. The output of the network has a output dimension corresponding to the number of possible actions, the output simply represent the Q-values for the different action in a given space. From training the parameters of the network the output should converge to values close to the one step Q-function, equation 3.2.

Algorithm 1: Deep Q-learning algorithm

```
Initialize policy Q-network, $Q$, with random weights $\theta$
Initialize target Q-network, $Q_t$, and set $\theta_t = \theta$
Initialize replay memory, $D$, with capacity, $N$
while terminal state not reached do
    Select random action $a_i$ with probability $\epsilon$
    Select $a_i = \text{argmax}(Q(s_i, a_i; \theta))$ otherwise
    Execute action $a_i$ and take note of resulting state $s_{i+1}$
    Store transition ($s_i$, $a_i$, $r_i$, $s_{i+1}$) in replay memory
    Sample a minibatch of transitions ($s_j$, $a_j$, $r_j$, $s_{j+1}$)
    Set $y_j = \begin{cases} r_j & \text{if terminal state reached} \\ r_j + \gamma \max_\pi(Q_t(s_{j+1}, a'; \theta_t)) & \text{otherwise} \end{cases}$
    Do backpropagation on $(y_j - Q(s_j, a_j; \theta))^2$
    Every $C$ step set $\theta_t = \theta$
end
```

The algorithm starts by initializing a memory used for storing transitions and randomly initializing the parameters for the policy network. The algorithm chooses actions ϵ-greedily and store any action taken together with the state, resulting state of the action and reward of the action.

After executing an action, the algorithm reaches the phase called experience replay, during this phase the algorithm tries to correct the parameters of the policy network via backpropagation. A minibatch of transitions is sampled uniformly from the memory and targets are generated for each transition in the minibatch. Targets, y_i, are generated with the help of a second neural network, the target network. It is a network with the same structure as the policy network, it is initialized as a copy of the policy network in the beginning of the algorithm.

To decrease the instability introduced from using a constantly moving target the target network parameters are kept constant and are only updated with a predefined regularity. When the target network weights are updated it is updated by copying the parameters from the policy network.

The targets used for backpropagation takes the form of

$$y_i = \begin{cases} r_i & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_\pi(Q_t(s_{i+1}, a'; \theta_t)) & \text{otherwise.} \end{cases} \tag{3.3}$$

This target is then used to calculate the error used for a gradient descent step on the policy Q-network. The error is backpropagated using stochastic gradient descent or any other standard optimizer, such as RMS-prop.

3.2 Prioritised learning

One limitation with algorithm 1 is that transitions are given equal importance when sampled during experience replay, transitions that will learn the algorithm nothing are as likely to be sampled as very important moves. To help with this an modification called Prioritized Learning has been developed as an modification to Deep Q-learning and it has also been used for learning an algorithm to play Atari 2600 games [10].

Prioritized learning contains several modifications compared to Deep Q-learning, one is that each transition needs to be ranked according to importance. A transition with a high rank will be more likely to be sampled during experience replay. The importance is captured by the absolute value of the temporal-difference, or TD-error, $|\delta_j| = |r_j + \gamma \max(Q_t(s_{j+1}, a'; \theta_t)) - Q(s_j, a_j; \theta)|$ and is used for the rank or priority p_j of transition j. The TD-error is also used for the loss when backpropagating.

The probability to sample a transition j during experience replay takes the following form

$$P(j) = \frac{p_j^\alpha}{\sum_k p_k^\alpha}. \tag{3.4}$$

The parameter α controls to what extent the prioritisation is used, $\alpha = 0$ corresponds to uniform sampling.

One convenient consequence of sampling transitions uniformly is that the sampling will not contain any bias, this will not be the case for prioritised learning. To correct for this bias weighted importance sampling is used, the weights take to form of:

$$w_j = \left(\frac{1}{N \cdot P(j)}\right)^{\beta}.$$

The parameter β controls to what extent the non-uniformly probabilities are compensated, with $\beta = 1$ corresponding to full compensation. When backpropagating the product of the weight and TD-error, $w_j \cdot \delta_j$, is used as the loss.

Chapter 4

Methodology

We have introduced in section 3.1 the concept of a function approximator to capture the action value function for any problem that can be formulated as a reinforcement learning problem. We adopt this approach and use a neural network that suggests a recovery chain step by step to bring the system back to one of the four described ground states, solely based on the information obtained from the syndrome. The environment state (the actual state of each physical qubit) is hidden from the agent. The agent uses a CNN with one fully connected output layer to approximate the Q-values of possible actions given a syndrome.

4.1 State space representation

We make use of the periodic boundary condition of the torics code, such that given a syndrome, S, we generate several perspectives, P_i. A perspective centers one qubit that is adjacent to an excitation. Thus, a perspective captures the relative position of all the other excitations of the syndrome. The set of all perspectives is defined as an observation, O (see figure 4.1). The agent predicts which action gives the highest cumulative reward and acts accordingly.

The number of possible actions varies with the number of excitations and therefore also the number of perspectives. To ensure a constant sized output of the Q-network we let the agent predict on every perspective separately. To leverage the parallelism of the computer it is convenient to consider the observation, O, as a batch of the size of number of perspectives, N_{per}. $Q(P_i, a_i; \theta)$ represents the expected cumulative reward given the perspective and action. The full Q-function given a syndrome is described by $\{Q(P_i, a_i; \theta\}_{P \in O}$ or $Q(O_i, a_i; \theta)$. After evaluating the action Q-values for every qubit the action is determined by a greedy policy.

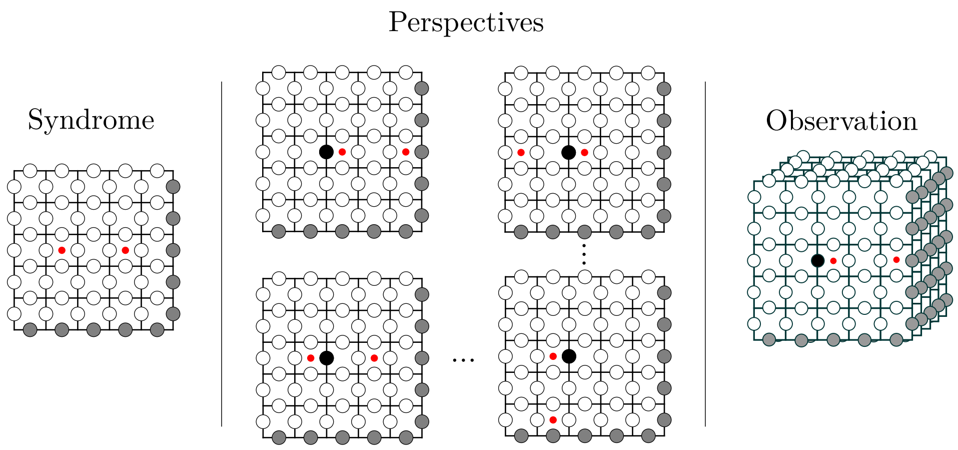


Figure 4.1: State formulation. Left: A representation of the syndrome. Middle: Given a centralized qubit (see black qubit), we can extract all the different perspectives (shown is just a subset of the perspectives, P). Right: All perspectives, P, define the observation, $O = \{P_1, P_2, ..., P_{N_{per}}\}$.

The new syndrome is sent to the agent and the procedure continues until the terminal state or any predefined stopping criteria is reached. For the implementation presented in this work we added an additional constraint that after 50 steps any sequence is terminated and a new one is initialized.

4.2 Model architecture

We use an architecture in which there is a separate output unit for each action and only the state representation (in our case the observation, O) is the input to the neural network. The output vector approximates the Q-values of each individual action. Therefore, it is possible to compute every Q-value for all possible actions with one forward pass through the network, given the input state, O.

The network architecture depends on the grid size, the bigger the grid the more complex we choose the network. Nevertheless, the general structure of the neural network is for all grid sizes the same. The input is a three dimensional binary matrix of size $(d \times d \times 2)$ containing all the excitations in the plaquette and vertex space respectively (1 for excitation, 0 otherwise).

We use n-convolutional layers with zero padding (except for the first layer), whereas the kernel size is (3×3) with stride 1 for all the convolutional layers. Due to the periodicity of the environment we apply a periodic padding to the first convolutional layer of the network. The final hidden layer is a fully connected layer and the output layer is a fully connected layer with a single output for each valid action. The possible actions are the Pauli σ^x, σ^y and σ^z-operators respectively (see

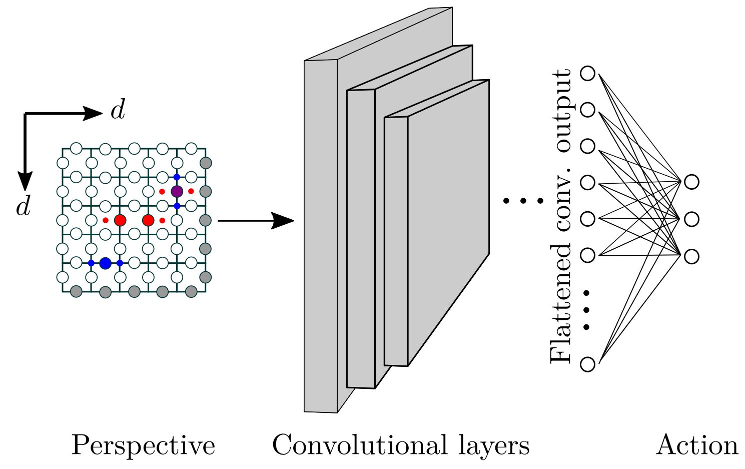


Figure 4.2: Structure of the deep Q-network. The input is a batch of perspectives, P_i, or the observation, O, constructed from the syndrome, S_i. The output is the action Q-value, $Q(P_i, a_i; \theta)$, of applying a Pauli operator to one of the qubits and therefore altering the syndrome. The output is a table of dimension $(3 \times N_{per})$, containing for every perspective, P the proposed action. The hidden layers consist of convolutional layers, followed by one fully connected layer.

figure 4.2). Every layer is followed by a rectified linear unit (ReLU) activation function, except the final output layer (for a more detailed description of the model see appendix A.2).

4.3 Training the neural network

The agent is trained using the deep Q-network (DQN) algorithm (see algorithm 1). The algorithm takes advantage of experience replay, meaning that every transition of the agent is stored in a memory buffer (see section 3.1). The network is trained by using a minibatch of random samples from the memory buffer. By utilizing random samples of experience, the temporal correlation of the data is minimized.

Moreover, using a separate Q-network to compute the target ensures a more stable training procedure. The target Q-network is systematically synchronized with the Q-network.

The training starts with a self play phase. Given a random syndrome, the agent suggests an action a_i, based on the Q-values and given all the different perspectives, P_i. An ϵ-greedy policy is used and therefore, with probability $(1-\epsilon)$ the agent suggests the action with the highest Q-value, otherwise a random action is proposed. The action is performed on a qubit and alters the state of the environment. The agent receives a reward, r_i, and the syndrome for the next state, S_{i+1}, given action a_i. The transition is stored as a tuple, $T = (P_i, a_i, r_i, S_{i+1}, t_{i+1})$, whereas

t_{i+1} is a boolean containing the information whether S_{t+1} is a terminal state (the system is in one of the four ground states and there are no excitations left).

The reward design is captured in equation 4.1, where E represents the energy of the system (meaning the count of excitations in the plaquette and vertex space). Therefore we can define an intermediate reward and simplify the training procedure for the agent. Doing that we also introduce an additional constraint which might introduce an unexpected behaviour of the agent.

$$r_i = \begin{cases} 100 & \text{if episode terminates at step } i+1 \\ E_i - E_{i+1} & \text{otherwise,} \end{cases} \tag{4.1}$$

The self play phase is followed by a self improvement phase. For that we utilize stochastic gradient descent (SGD) and the tuples stored in the memory buffer, D. A minibatch of N transitions is sampled from the memory buffer, D, $\{T = (P_i, a_i, r_i, S_{i+1}, t)\}_{i=1}^{N}$ with replacement. We also used a method selecting samples depending on their importance (Prioritized learning uses the TD-error as a reference to measure the significance of an experience, for more details see section 3.2). The training target value for the policy Q-network is given by,

$$y_i = \begin{cases} r_i & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_{a'} Q(O_{i+1}, a'; \theta_t) & \text{otherwise.} \end{cases} \tag{4.2}$$

whereas γ represents the discount factor and the target network is used to predict the future cumulative reward. It is worth mentioning that the replay memory only stores the syndrome, S_{i+1}, therefore it is necessary to generate the observations, O_{i+1} and select, given the different perspectives, the action maximizing the cumulative reward (see equation 4.2).

Now we have described the reinforcement learning problem as a supervised learning problem and therefore we can use gradient descent to minimize the difference between the Q-value prediction of the target Q-network and the policy Q-network. We update the network parameters according to $-\nabla_\theta \sum_i (y_i - Q(P_i, a_i; \theta))^2$. The training sequence is repeated over and over again until the agent converges to the optimal policy. Figure 4.3 captures the procedure in a flowchart. A pseudo code description is presented in algorithm 2. A more detailed description regarding the hyperparameters and the network architecture can be found in appendix A.2.

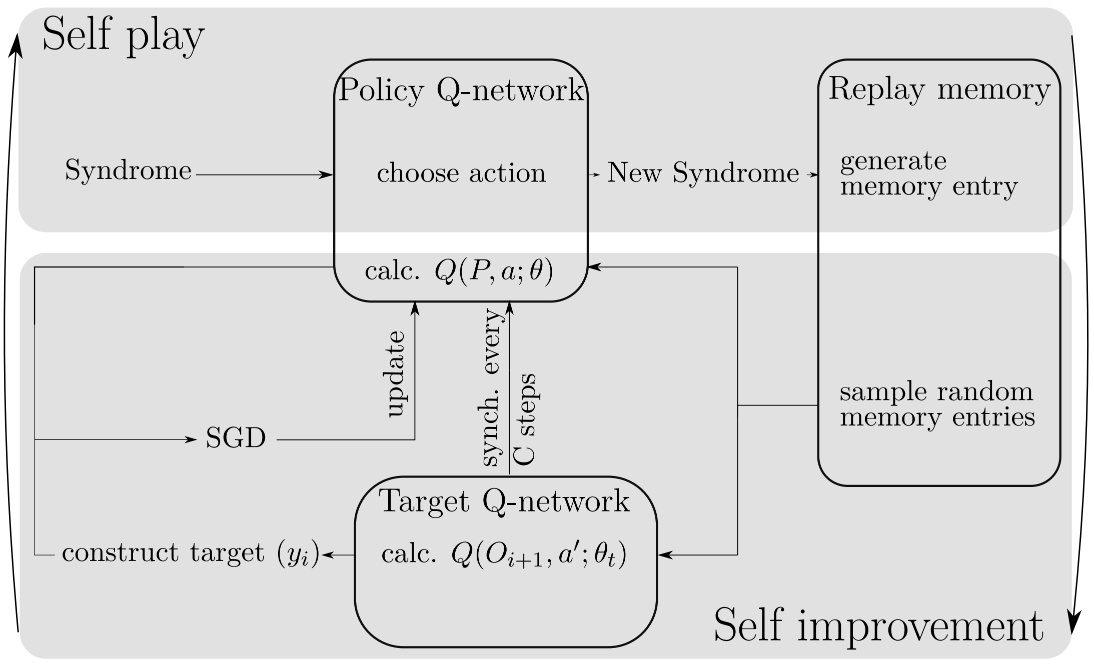


Figure 4.3: Flowchart of the training procedure. The learning sequence consists of a self play and self improvement phase. It is an iterative process. During the self play phase the policy Q-network is used to generate memory samples and these are stored in the replay memory. In the self improvement phase random memory entries are sampled and used to compute the loss between the target Q-network and the policy Q-network. We update the weights of the policy Q-network via stochastic gradient descent.

Algorithm 2: Training the reinforcement learning agent decoder

```
Initialize policy Q-network, $Q$, with random weights $\theta$
Initialize target network, $Q_t$, and set $\theta_t = \theta$
Initialize replay memory, $D$, with capacity, $N$
for episode = 1, M do
    Generate initial syndrom, $S$
    while terminal state not reached do
        Get observation $O$, whereas $P \in O$
        with probability $\epsilon$ select random action, $a_i$,
        otherwise select $a_i = \text{argmax}_a \{Q(P_i, a_i; \theta\}_{P \in O}$
        Execute action, $a_i$ and observe reward $r_i$ and syndrome $S_{i+1}$
        Store transition $(P_i, a_i, r_i, S_{i+1}, t_{i+1})$ in $D$
        Sample random minibatch of transitions from $D$
        Set $y_i = \begin{cases} r_i & \text{if episode terminates at step } i+1 \\ r_i + \gamma \max_{a'} Q(O_{i+1}, a'; \theta_t) & \text{otherwise} \end{cases}$
        Perform gradient descent step on $(y_i - Q(P_i, a_i; \theta))^2$ with respect to the network
          parameter $\theta$
        Every C steps synchronize the target network with the policy netowork, $\theta_t = \theta$
    end
end
```

4.4 Benchmarking

After successfully finishing the training phase of the agent, it is necessary to evaluate its performance. Therefore, we compare its success rate, p_s, (the rate for conserving the ground state) to the success rate of the MWPM algorithm.

We predict on a set of randomly initialized syndromes and let the agent and the MWPM algorithm find a correction chain that brings the system back to its ground state energy. We expect that all the syndromes, for both algorithms, are always terminating in the ground state energy (all excitations are annihilated). Nevertheless, a logical qubit error can occur when a correcting error chain introduces non-trivial loops around the torus. To check for non-trivial loops, we keep track of the actual qubit state (this state is hidden to the agent) and use that to verify that the ground state is conserved.

For the limit of small error rates it is possible to derive an exact expression for the rate of logical failure under the assumption of MWPM error correction and therefore providing a solid tool to benchmark the performance of the RL agent. For a detailed derivation see appendix A.1.

We test the algorithm on various error probabilities to evaluate its performance. We will investigate error rates between $0.05 - 0.18\%$ and with syndromes generated with $\left\lceil \frac{d}{2} \right\rceil$ errors, since this is the minimum numbers of errors needed to produce a syndrome where MWPM will fail to keep the ground state (see chapter 6 for more details).

4.5 Minimum weight perfect matching

For the MWPM-algorithm we used an implementation called Blossom V [7]. The MWPM-algorithm solves graphs generated from syndromes generated in the same way as for the RL-based algorithm. This approach is a near to optimal for the uncorrelated noise problem (solving the syndrome in the vertex and plaquette space independently).

For our work we consider the depolarizing noise model, therefore we expect the RL algorithm to outperform the MWPM algorithm. We examine correlations between the plaquette and vertex space. With a probability of $(1-p)$ there will be no error and with a probability of $(p/3)$ a σ^x, σ^y or σ^z is introduced. The optimal decoder threshold for the toric code has been estimated to be at 18.9% [3].

Chapter 5

Results

Figure 5.1 shows the success rate, i.e. the fraction of error corrections not changing the ground state, of the MWPM and RL-based algorithm for two different grid sizes ($d = 5, 7$). The success rate decreases as the error probability, p, increases. The RL-based algorithm performs better than the MWPM-based algorithm. For $d = 7$ we can observe a higher success rate than for $d = 5$ considering error probabilities lower than $p = 0.15$, and for error probabilities lower than $p = 0.15$ we see a lower succes rate.

The results for low error probabilities can be seen in figure 5.2. The fail rate, p_L $(1 - p_s)$, is shown for the RL-algorithm, considering two different grid sizes ($d = 5, 7$). Moreover, a theoretical function evaluated for the given error probabilities, p, is shown for the MWPM and for the RL agent (for a detailed derivation see appendix A.1). As expected, the fail rate decreases as the error probability decreases for all cases. The RL-algorithm provides a lower error correction fail rate, compared to the fail rate of the theoretical performance of the MWPM algorithm. Nevertheless, the experimental fail rate is worse than the fail rate suggested by the theoretical values for the agent. For $d = 7$ we see lower fail rates compared to $d = 5$.

In table 5.1 we can see the fail rate of the algorithm when solving syndromes consisting of $\left\lceil \frac{d}{2} \right\rceil$ errors compared to a theoretical value of the fail rate. We notice that for both grid sizes, d, the experimental value is similar to the theoretical, for $d = 7$ the difference is larger.

Table 5.1: Comparison of theoretical and experimental faile rate, p_L, considering the RL approach

	theoretical	experimental
d = 5	1.51-3	1.45e-3
d = 7	2.12e-5	3.10e-5

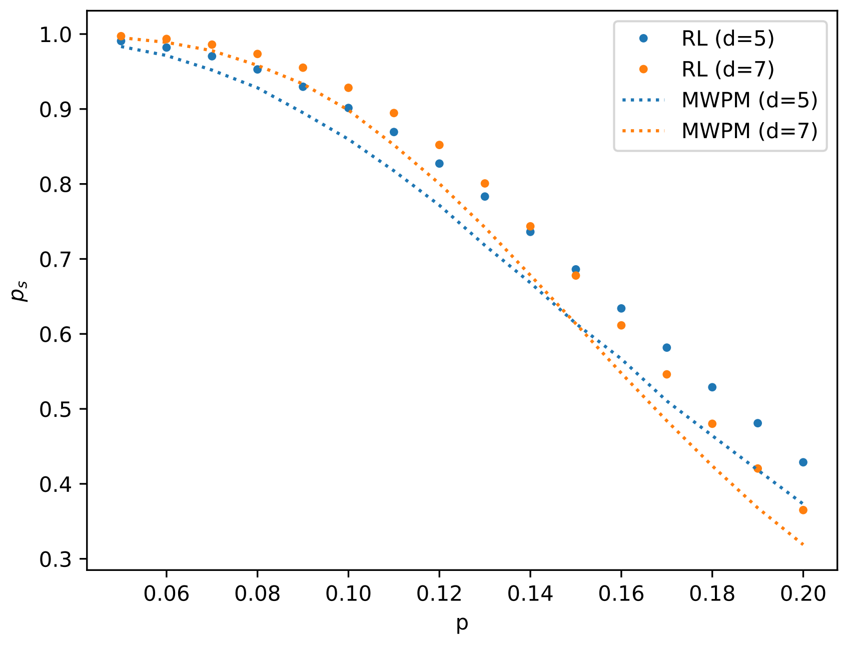


Figure 5.1: The error correction success rate, p_s, of the converged agents versus the error probability p, for system sizes $d = 5, 7$, compared to the corresponding results using the MWPM algorithm. It is visible that the RL-based algorithm outperforms the MWPM-based algorithm.

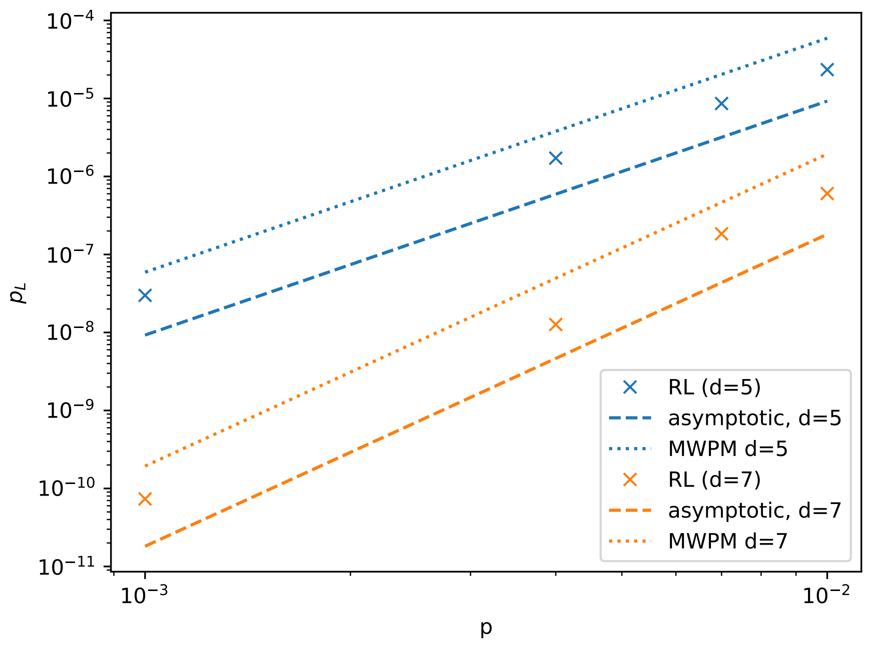


Figure 5.2: The error correction fail rate, $p_L = 1 - p_s$, of the converged agents as a function of very low error probabilities, p. The dashed and dotted lines show an approximation for the theoretical fail rate of the agent and the theoretical fail rate for the MWPM-algorithm respectively (see appendix A.1).

Chapter 6

Discussion

From figure 5.1 and figure 5.2 it can be clearly seen that an RL-based algorithm performs better than MWPM. The difference in performance seem to increase as the error probability increases. As can be further noticed in figure 5.1 for the higher grid size, $d = 7$, the success rate of the algorithm is higher than for $d = 5$ until a threshold at around $p = 0.15$. For error probabilities higher than that threshold the success rate decreases as the grid size increases, this behaviour is observed for both the RL algorithm and the MWPM algorithm. This is inline with results from RL algorithms used on the toric code with only bit flip errors [8].

One explanation for the better performance of the RL-based algorithm compared to MWPM, is that the RL-based algorithm can use the σ^y-operator. MWPM treats defects in the plaquette and vertex space separately, as it does not use the σ^y-operator. The RL-based algorithm does use the σ^y-operator since the reward scheme will give a higher return for using as few moves as possible and use of the σ^y-operator can decrease the number of moves used in some cases.

One example of a case where the RL-based algorithm will succeed whereas MWPM will fail to keep the correct ground state can be seen in figure 6.2. The RL-based algorithm will use two σ^y- and one σ^x-operator to solve the syndrome, MWPM will use two σ^x-operators around the torus, together with two σ^z-operator, and thus changes the ground state. This shows that the RL-algorithm can successfully solve a class of problems that MWPM fails on.

The reward scheme used during training (see equation 4.1) will reward solving syndromes with the minimum number of moves. This should lead to a theoretical algorithm with behaviour identical to MWPM for syndromes consisting of only σ^x- and σ^z-errors. If any syndromes also contains σ^y-errors using σ^y-operators will lead to a solution with a lower amount of moves. MWPM would need at least to moves to solve defects arisen from a σ^y-error.

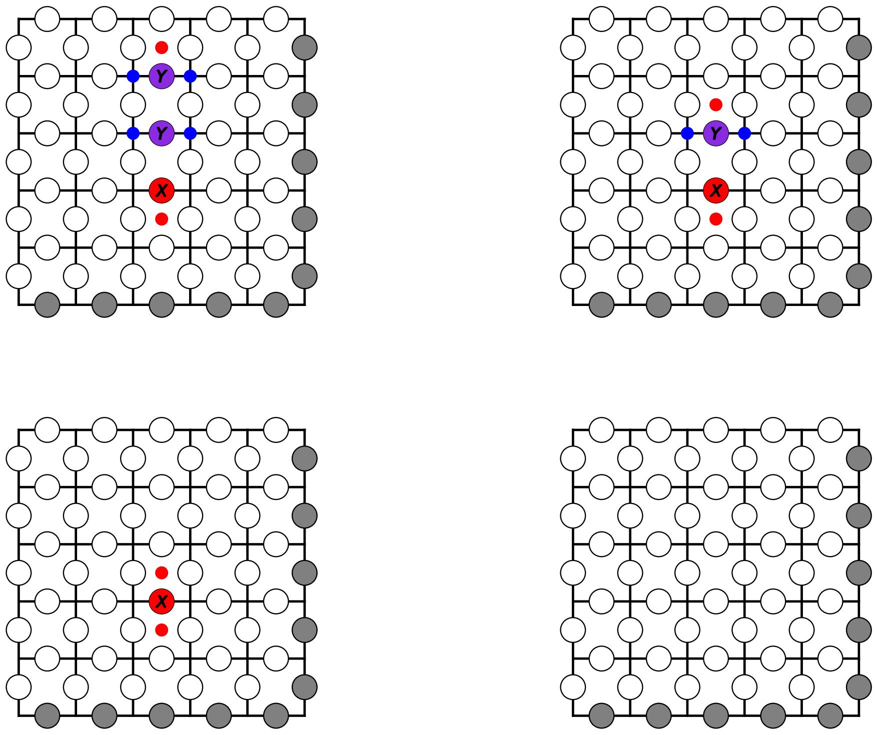


Figure 6.2: An example of a successful, i.e. keeping the ground state, solution of a syndrome done by a RL-based algorithm. The algorithm starts by using two σ^y-operators and then finishes by using a σ^x-operator. This leads to a unchanged ground state in contrast to what MWPM would have produced.

6.1 Performance on very low error probabilities

The minimum number of errors needed for the MWPM and the RL-algorithm to fail is at least $\left\lceil \frac{d}{2} \right\rceil$, this can be easily seen with the following reasoning. $\left\lceil \frac{d}{2} \right\rceil$ is the minimal length of a row(or column) of errors needed to produce a syndrome where the defects are placed in such a way that the shortest correction chain is around the torus, any less number of errors would produce a syndrome where the shortest solution does not go around the torus. Therefore, $\left\lceil \frac{d}{2} \right\rceil$ errors need to be placed on a column or row to create a situation where the MWPM will always produce an correction chain that changes the ground state. For MWPM it is a sufficient condition to have $\left\lceil \frac{d}{2} \right\rceil$ of either σ^x, σ^z or σ^y, or any combination of $\left\lceil \frac{d}{2} \right\rceil - 1$ σ^y-errors together with either a σ^x or σ^z-error, on a row or a column to fail the error correction.

The RL-based algorithm also fails on any syndrome containing a row or column of either $\left\lceil \frac{d}{2} \right\rceil$ σ^x or a $\left\lceil \frac{d}{2} \right\rceil$ σ^z-errors, however it will succeed on syndromes consisting of $\left\lceil \frac{d}{2} \right\rceil$ σ^y-errors. The RL-based algorithm will also succeed on any syndrome consisting of 2 σ^y-errors and either $\left\lceil \frac{d}{2} \right\rceil - 2$ σ^x or $\left\lceil \frac{d}{2} \right\rceil - 2$ σ^z error on a row or column, as can be seen in figure 6.2.

In cases where a row or column contains exactly one σ^y-error and, $\left\lceil \frac{d}{2} \right\rceil - 1$, σ^x- or σ^z-errors together on a row or column, there is a 50% chance of conserving the ground state. This is due to the fact two different error correction chains use the minimum number of actions. One winds the torus and creates a non-trivial loop, and one correction chain simply reverses the error chain. One example of this case can be seen in figure 6.3, for initial syndrome 1b, which shows the initial syndrome and the result from the two different error corrections.

It is important to note that if the σ^y-error, in example 1b in fig 6.3, is not adjacent to the other errors the RL-algorithm will succeed in keeping the ground state of the system. This is a result of the reward scheme giving an immediate reward when eliminating pairs of defects. The lone σ^y-error produces two pairs of defects, and the σ^x or σ^z-errors produces a pair of defect, thus giving the RL-algorithm an opportunity of eliminating two pairs of defects with one operation. This should produce a higher immediate Q-value for choosing to operate on the σ^y-error first and thus producing a state that the RL-algorithm can solve without going around the torus.

Initial syndrome 1a in figure 6.3 is an example of a class of syndromes where the RL-algorithm can fail whereas MWPM will always succeed, this is another case where there is a 50% chance of the algorithm to learn to not keep the ground state. Using a σ^y-operator on the σ^z-error in the middle will create an error chain of length $\left\lceil \frac{d}{2} \right\rceil$ σ^x errors, which is a sufficient condition for the RL-algorithm to chose a solution chain around the torus. These cases do not seem to be common enough to affect the overall performance in any significant way.

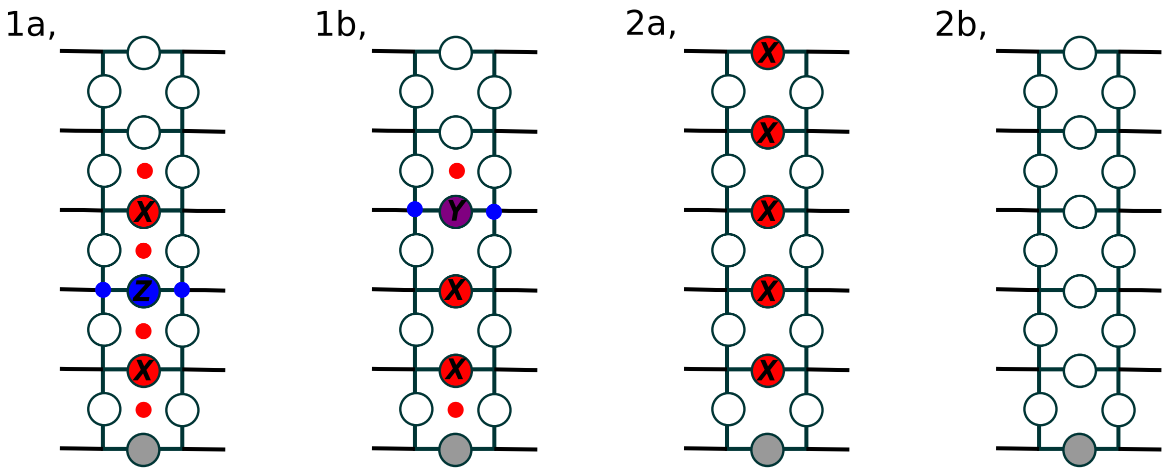


Figure 6.3: Example of two syndromes on a column that have two possible optimal correction chains containing the same amount of operations. 1a show one of the initial syndromes and 1b show the other initial syndrome, 2a shows one example of the result of an optimal correction chain and 2b show the other example of the result of an optimal correction chain. Both correction chains used 3 operations.

For very low error probabilities the distribution of syndromes leading to a failed error correction, i.e. changing the ground state, will be dominated by the minimum numbers of errors required for failure, i.e. $\left\lceil \frac{d}{2} \right\rceil$ errors. With this knowledge it is possible to derive expressions for an asymptotic behaviour of the fail rate, p_L, in the limit when the error probability approaches zero. The asymptotic behaviour of the RL-based solution takes the form of

$$p_{L_{RL}} = 4d \cdot \left[\binom{d}{\lceil \frac{d}{2} \rceil} \cdot \left(\frac{p}{3}\right)^{\lceil \frac{d}{2} \rceil} + \left\lceil \frac{d}{2} \right\rceil \binom{d}{\lceil \frac{d}{2} \rceil - 1} \cdot \left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil - 1} \right] \tag{6.1}$$

The asymptotic behaviour for MWPM takes the form of

$$p_{L_{MWPM}} = 4d \left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil} \cdot \sum_{N_y=0}^{\lceil \frac{d}{2} \rceil} 2 \cdot \binom{d}{\lceil \frac{d}{2} \rceil - N_y} \binom{d - \lceil \frac{d}{2} \rceil + N_y}{N_y} \tag{6.2}$$

for further details of the derivation see appendix A.1

As can be seen in figure 5.2 the RL algorithm outperforms MWPM for low error probabilities. This is because of the use of the σ^y-operator as discussed above. The RL-algorithm does not reach the theoretical asymptotic values for the error probabilities tested in figure 5.2 as syndromes with a higher amount of errors than $\left\lceil \frac{d}{2} \right\rceil$ are still relatively common for $p_L = 10^{-3}$. However, we can see that the fail rate seem to converge to the asymptotic behaviour of the theoretical expression derived for the agent as the error probability reaches zero.

One other possible reason for the RL-algorithm not reaching the theoretical values of the RL-algorithm is the cases where it is a chance of 50 % of success and 50 % of failure. The RL-algorithm is completely deterministic, it will either learn to fail or succeed on the cases with a theoretical 50 % chance of success. In the worst case scenario the algorithm would fail on all of these cases and in the best case it would learn to succeed on all such cases. The worst case scenario would lead to a significantly worse performance compared to equation 6.1, but the best case would lead to a better performance.

When comparing the fail rates, see table 5.1, for syndromes containing only $\left\lceil \frac{d}{2} \right\rceil$ errors it can be seen that the RL-algorithm does more or less reach the theoretical value derived from the reward scheme. The case of syndromes generated by $\left\lceil \frac{d}{2} \right\rceil$ errors should be a good measure of how the algorithm performs in the asymptotic case, since it measures the the performance on a subset of syndromes that will generate the only theoretical cases of asymptotic failure. This shows that the algorithm more or less reaches the theoretical asymptotic behaviour.

It is noticed for all tested cases that the RL-based algorithm failed to keep the ground state on several syndromes where it theoretically should not. For a grid size of 5 the number of these unexpected failures are low but they increase as the grid size is increased. This implies that lack of training can be attributed to the unexpected failures, a higher grid size leads to a higher state space leading to an expected higher training time for a given hardware setup. As the Q-networks are trained even more the rate of unexpected failures should decrease.

Chapter 7

Conclusion

During this thesis a deep reinforcement learning based algorithm has been developed. The resulting algorithm have been trained on randomly generated syndromes, using depolarized error generation. A reward scheme that both rewards the minimum amount of moves used to solve a syndrome and eliminating individual pairs of defects was used. The trained algorithm was evaluated using syndromes generated from the same distribution as used during the training. The algorithm was evaluated using both high error probabilities and very low error probabilities to evaluated performance of the algorithm as in the limit when the error probability approaches zero.

When reviewing the results of the thesis it can be concluded that it is possible to successfully solve correlated errors on the toric code with deep reinforcement learning. It can also be concluded that it is possible to supersede, with regard to conserving the ground state of the system, MWPM performance with an RL-based algorithm. This is at least possible for the grid sizes, $d = 5, 7$, analyzed in this thesis. The RL-based algorithm outperforms MWPM in both asymptotic behaviour for low error probabilities and for high error probabilities. The RL-based algorithm did not reach the theoretical performance calculated from the reward scheme used, we believe that this can be reached with even more training of the algorithm.

7.1 Suggestions for further research

It is possible that this result will not hold for even higher grid sizes, more research is needed on larger grid sizes to draw conclusion if this is the case and why eventual limitations arise. Since the implementation used in this thesis is dependent on the boundary conditions of the toric code more work is needed for implementing deep reinforcement learning on the surface code. Research into implementation on time dependent systems is also an interesting area of further research.

Bibliography

[1] al., Brown et. “Quantum memories at finite temperature”. In: *arXiv:1411.6643v4 [quant-ph]* (Nov. 2016).

[2] al., Mnih et. “Human-level control through deep reinforcement learning”. In: *Nature, vol. 518, pp. 529-533* (Feb. 2015. doi:10.1038/nature/14236).

[3] Browne, D. “Topological Codes and Computation”. In: *A lecture course given at the University of Innsbruck* (May 2014).

[4] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *CVPR09*. 2009.

[5] Jimenez, J. P. I. *Gauge and Matter Fields on a Lattice - Generalizing Kitaev’s Toric Code Model.* São Paulo, 2015.

[6] Kolmogorov, V. “Blossom V: a new implementation of a minimum cost perfect matching algorithm”. In: *Mathematical Programming Computation* 1.1 (July 2009), pp. 43–67. URL: `https://doi.org/10.1007/s12532-009-0002-8`.

[7] Kolmogorov, Vladimir. *Blossom V: A new implementation of a minimum cost perfect matching algorithm.* 2009.

[8] P. Andreasson, J. Johansson, Liljestrand, S., and Granath, M. “Quantum error correction for the toric code using deep reinforcement learning”. In: (Apr. 2019).

[9] P. Baireuther T. E. O’Brien, B. Tarasinski and Beenakker, C. W. J. “Machine-learning-assisted correction of correlated qubit errors in a topological code”. In: *arXiv:1705.07855v3 [quant-ph]* (Jan. 2018).

[10] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. “Prioritized Experience Replay”. In: *ICLR*. Feb. 2016.

[11] Sutton, R. S. and Barto, A. *Rienforcement Learning*. 2nd ed. The MIT Press, 2018.

[12] Wikimedia. *Pointcare conjecture*. 2019. URL: `https://en.wikipedia.org/wiki/Poincar%C3%A9_conjecture`.

List of Figures

Appendix

A.1 Small error rate

By using the points, regarding the conditions of failure, discussed in section 6.1 it is possible do derive a theoretical expression of the asymptotic failure rate for low error probabilities. Here we derive the probability considering a low error probability where error chains of length $\lceil \frac{d}{2} \rceil$ are dominating the failed syndromes. We assume the error probability for σ^x, σ^y and σ^z to be equally likely, thus $p_{x,y,z} = \frac{p}{3}$, where p is the probability of an error. The probability for a logical qubit change for low error probabilities, considering our RL based approach, ($p_{L_{RL}}$) is given by

$$p_{L_{RL}} = p_{xxx} + p_{zzz} + p_{yxx} + p_{yzz} + p_{xzx} + p_{zxz}. \tag{A.1}$$

With p_{xxx} and p_{zzz} being the probability of having three σ^x and σ^z-errors respectively. p_{yxx} and p_{yzz} being the probability of having on σ^y-error and two σ^x or σ^z-errors (analog for $p_{xzx} + p_{zxz}$). The probability of three errors of the same type is

$$p_{xxx} = p_{zzz} = 2d \cdot \binom{d}{\lceil \frac{d}{2} \rceil} \cdot \left(\frac{p}{3}\right)^{\lceil \frac{d}{2} \rceil}. \tag{A.2}$$

The probability of having one σ^y-error together with either two σ^x or σ^z-errors takes the form of

$$p_{yxx} = p_{yzz} = \underbrace{\frac{1}{2}}_{\text{p of success}} \underbrace{2d}_{\text{nbr of rows and columns}} \cdot \underbrace{\left\lceil \frac{d}{2} \right\rceil \cdot \binom{d}{\lceil \frac{d}{2} \rceil - 1}}_{\text{nbr of ways to place errors}} \cdot \underbrace{\left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil - 1}}_{\text{p of getting xy, or zy}} . \tag{A.3}$$

A similar expression holds for $p_{xzx} + p_{zxz}$.

$$p_{xzx} = p_{zxz} = \underbrace{\frac{1}{2}}_{\text{p of success}} \underbrace{2d}_{\text{nbr of rows and columns}} \cdot \underbrace{\left\lceil \frac{d}{2} \right\rceil \cdot \binom{d}{\lceil \frac{d}{2} \rceil - 1}}_{\text{nbr of ways to place errors}} \cdot \underbrace{\left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil - 1}}_{\text{p of getting xz, or zx}} . \tag{A.4}$$

Inserting equation A.2, A.3, A.4 in A.1 and simplifying we obtain the following probability of failure in the case for very low p:

$$p_{L_{RL}} = 4d \cdot \left[\binom{d}{\lceil \frac{d}{2} \rceil} \cdot \left(\frac{p}{3}\right)^{\lceil \frac{d}{2} \rceil} + \left\lceil \frac{d}{2} \right\rceil \binom{d}{\lceil \frac{d}{2} \rceil - 1} \cdot \left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil - 1} \right] \tag{A.5}$$

It is possible to derive in a similar way an expression for the MWPM algorithm

$$p_{L_{MWPM}} = p_{xxx} + p_{zzz} + p_{yxx} + p_{yzz} + ... + p_{yyy} \tag{A.6}$$

N_y denotes the number of σ^y operators in the error chain. The MWPM algorithm fails on any combination of only σ^x or σ^z and any combination of σ^x or σ^z and σ^y.

$$p_{L_{MWPM}} = \underbrace{2}_{\text{x and z errors}} \cdot \underbrace{2d}_{\text{nbr of rows and columns}} \cdot \underbrace{\left(\frac{2p}{3}\right)^{\lceil \frac{d}{2} \rceil}}_{\text{p of getting xy, or zy}} \cdot \underbrace{\sum_{N_y=0}^{\lceil \frac{d}{2} \rceil} 2 \cdot \binom{d}{\lceil \frac{d}{2} \rceil - N_y} \binom{d - \lceil \frac{d}{2} \rceil + N_y}{N_y}}_{\text{nbr of ways to place errors}} . \tag{A.7}$$

Another useful representation is to calculate the ratio of syndromes with $\left\lceil \frac{d}{2} \right\rceil$ errors that lead to a failure compared to the total number of syndromes with $\left\lceil \frac{d}{2} \right\rceil$ errors.

$$f_{RL} = \frac{4d \cdot \left[\binom{d}{\lceil \frac{d}{2} \rceil} + \lceil \frac{d}{2} \rceil \binom{d}{\lceil \frac{d}{2} \rceil - 1} \right]}{\binom{2d^2}{\lceil \frac{d}{2} \rceil} \cdot \lceil \frac{d}{2} \rceil^3} \tag{A.8}$$

Accordingly for the MWPM

$$f_{MWPM} = \frac{\displaystyle\sum_{N_y=0}^{\lceil \frac{d}{2} \rceil} 2 \cdot \binom{d}{\lceil \frac{d}{2} \rceil - N_y} \binom{d - \lceil \frac{d}{2} \rceil + N_y}{N_y}}{\binom{2d^2}{\lceil \frac{d}{2} \rceil} \cdot \lceil \frac{d}{2} \rceil^3} \tag{A.9}$$

A.2 Model definition and Hyperparameters

The different hyperparameters used, and a short description of them, for the RL-based algorithm can be observed in table A.1. The structure of the deep neural network used for most of the training can be seen in table A.2. As can be seen the network consists of mostly convolutional 2 dimensional layers of decreasing size. All layers except the first used zero-padding. The first layer used padding with periodic boundary conditions. The model was trained on a desktop computer utilizing a gpu.

Table A.1: List of hyperparameters and their values

Hyperparameter	Value	Description
minibatch size	32	Number of training samples used for stochastic gradient descent update.
trainint steps	10000	Total amount training steps per epoch
replay memory size, N	10000	Total amount of stored memory samples.
target network update frequency, C	1000	The frequency with which the target network is updated with the policy network (refers to parameter C in algorithm 1).
discount factor, γ	0.95	Discount factor γ used in the Q-learning update.
learning rate	0.00025	The learning rate used by Adam.
initial exploration	1	Initial value of ϵ in ϵ-greedy exploration.
final exploration	0.1	Final value of ϵ in ϵ-greedy exploration.
replay start size	1000	A random policy generates training samples to populate the replay memory before the learning starts.
optimizer	RMSprop	RMSprop is an optimization algorithm used to update network weights.
max steps per episode	50	Number of steps before every episode is terminated.

Table A.2: Network architecture d=5; Every convolutional layer has a kernel size of 3 and stride 1. Periodic padding is applied to the first convolutional layer. The other convolutional layers work with zero padding.

#	Type	Size	#parameters
1	Conv2d	128	2,432
2	Conv2d	128	147,584
3	Conv2d	120	138,360
4	Conv2d	111	119,991
5	Conv2d	104	104,000
6	Conv2d	103	96,511
7	Conv2d	90	83,520
8	Conv2d	80	64,880
9	Conv2d	73	52,633
10	Conv2d	71	46,718
11	Conv2d	64	40,960
12	Linear	3	1,731
			899,320