Reinforcement learning for error correction in quantum computers

Meng Hua [1] **Jing Luo** [1] **Syed Raza** [1]

Abstract

Topological error-correcting codes are one of the most promising routes for realizing large-scale, fault-tolerant quantum computers. One of the biggest hurdles for realizing quantum computers is their sensitivity to noise from the environment and from measurements. In this project, we implement techniques from Reinforcement Learning to model error-correction for various noise models in quantum computation. We implement this technique on a popular error-correcting code known as the surface code.

1. Introduction

Quantum Computers are on the verge of revolutionizing computing. One of the biggest hurdles in realizing large-scale, fault-tolerant quantum computers is their sensitivity to noise from environment and measurements. There have been advances in reinforcement learning (RL) problems, where a prior solution are not available and the agent reacts to a dynamic environment and learns to navigate to an optimal policy. Recently, there has been a tremendous success in combining reinforcement learning techniques with deep learning to tackle problems such as board games, computer games with human and super-human like control. Applying these techniques to problems in physics is an exciting new area of research in physics. There have been a few recent papers that employ reinforcement learning for error-correction in quantum computing models (Sweke et al., 2018; R. Sweke, 2018; Andreasson et al., 2018). In this project, we map the problem of error-correction to a board game where the opponent is the environment which introduces various noise models that create errors in the system. The agent is trained to find the optimal strategy for error-correction.

Most of the work presented in this report is reproducing the results obtained in reference (Sweke et al., 2018). The figures used in the report for illustrative purposes are taken from reference (Sweke et al., 2018). We try to extend the model by using more advanced techniques such the rainbow DQN agent but have not been able to generate meaningful data yet. It seems to be a promising direction.

The outline of the project report is as follows. We begin with a brief and non-technical introduction to quantum computation and error-correction in both classical and quantum computers. In Sec. 2, we describe our particular platform for quantum computation known as the surface code. We describe a lattice of physical qubits which make up one logical qubit and describe the qubit flip operations. In Sec. 3, we map the conditions and constraints of the noise models and the qubit lattice to a board game and explain the rules of the game. Once the problem is mapped to a board game, we can use techniques from reinforcement learning which have been successfully applied in the past for games such as AlphaGo. We give a brief description of Q-learning and Deep-Q learning in Sec. 4, the techniques will be applied later to our error-correction models. In Sec. 5, we discuss the noise model where both physical and measurement errors can be introduced into the system. In this model, the rules of the game are modified and the goal is now to maximize the lifetime of the logical qubit. The methods employed in this project are described in Sec. 6, the environment, parameters, and the RL model are also explained. The results and plots obtained are provided in Sec. 7. Some sample testing examples are also shown to give an idea of how the agent is performing. In Sec. 8, we discuss the advanced agents that we tried to implement in this project which go beyond the scope of the paper (Sweke et al., 2018). Finally, in Sec. 9 we conclude our project and discuss some possible future directions.

1.1. Quantum Computation

The universe, especially at the smallest scale, is governed by the laws of quantum mechanics. These laws are very different from the classical laws, such as electrons can tunnel through barriers with a certain probability, particles can be entangled with each other at long-length scales. Current classical computers are not able to fully simulate chemical and physical processes at the atomic scale because of quantum mechanics. To simulate and compute a quantum

[*]Equal contribution [1]Department of Physics, University of Virginia, Charlottesville, USA. Correspondence to: Meng Hua <mh2td@virginia.edu>, Jing Luo <jl4yr@virginia.edu>, Syed Raza <raza@virginia.edu>.

Reinforcement Learning Course Project, Department of Systems and Information Engineering, University of Virginia, Spring 2019.

mechanical world we need a quantum computer. Quantum computers exploit the laws of quantum mechanics and can solve a subset of problems exponentially faster than traditional computers.

One such problem is the problem of factorization. The reason one can send messages and bank transactions safely over the internet is because of encryption. Encryption is based on the principle that it is very hard to factorize large numbers into prime numbers unless you have a key. However, Shor's algorithm (Shor, 1994) shows that a quantum computer can factorize exponentially faster and encryption will break down.

Meanwhile quantum computers are extremely powerful for solving some problems, they will not be a replacement for traditional computers. They will also not be able to solve NP-hard problems.

1.2. Error-Correction

One of the main hurdles in realizing large-scale quantum computers is their sensitivity to noise. In classical computers, information is stored as bits, *0's* and *1's*. Noise from the surrounding environment can flip one of the bits, corrupting the information. A simple way around this is building redundancy into the system, like by encoding a *0* as multiple copies *000*. So even if the original bits of information *010* stored as *000111000* gets corrupted, *010111001*. A majority rule can be used to decode the information correctly. In quantum computers, the information is stored as quantum bits or 'qubits'. Each qubit is the superposition of both *0* and *1* state $q = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. It collapses to one of the sates on observation similar to Schrodinger's cat. Unlike the classical case, redundancy cannot be built into the quantum model as making copies of the qubits will collapse the system and is not allowed by the no-cloning theorem.

A clever way is to store the information non-locally so it is protected from noise even if it acts locally. Information can be stored in a quantum memory made up of logical qubits which are an assortment of physical qubits. Noise can flip or introduce error into the individual physical qubits but local operations of error-corrections can be applied and the global information in the logical qubit is still protected.

2. Surface Code

In this section, we will describe the error-correction in a particular model known as the surface code (Kitaev, 2003). We will not be going into the physics behind it but will explain the rules of allowed error-corrections as a board game.

In Fig. 1a, the lattice board represents one logical qubit comprised of an array of *5x5* physical qubits. Each vertex of the checkerboard is one logical qubit. The green circles X and Z represent an operation that flips the qubit at that particular vertex of the lattice. The combination of X and Z qubit flips represent stabilizers that are used to make measurements but will not be relevant for now. Fig. 1b represents a string of X and Z qubit flip operations. Such a string of operations that connect the opposite boundaries of the lattice change the state of our logical qubit and destroys it. The line does not have to be a straight line and can wind and turn but it has to connect the opposing boundaries to break down the logical qubit.

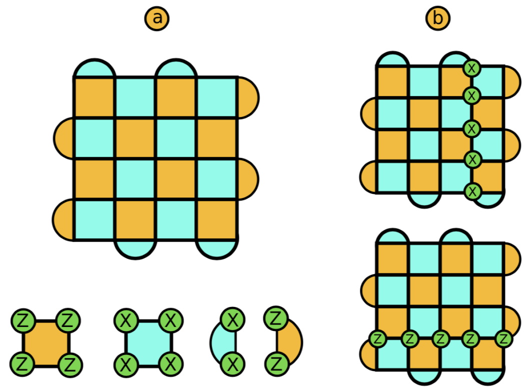

Figure 1. (a) represents a 5×5 lattice of physical qubits that make up one logical qubit. (b) represents the string of X and Z physical qubit flips that connect opposing boundaries and destroy the logical qubit

Fig. 2a represents the qubit flips that can occur because of external noise. The errors incurred by the qubit flip are represented by red dots in the plaquettes of the lattice and are known as syndromes. Notice that X and Z qubit flip operations represent a different pattern of syndromes in the lattice. The Y qubit flip operation is just a combination of X and Z. Fig. 2b-d represent some examples of syndromes caused by qubit flips. Notice that two red syndromes in the same plaquette equal to nothing. Another important thing to notice here is that different combinations of qubit flips can result in the same syndrome pattern, there is no one-to-one correspondence between qubit flips and the syndrome patterns. This is important as there are no fixed error-correcting moves, given a particular syndrome pattern. There is a probability distribution of actions that can be taken for a particular syndrome pattern, just like a board game. In this project, we have mapped the problem of quantum error-correction to rules of a board game to come up with the optimal probability distribution of error-correcting moves.

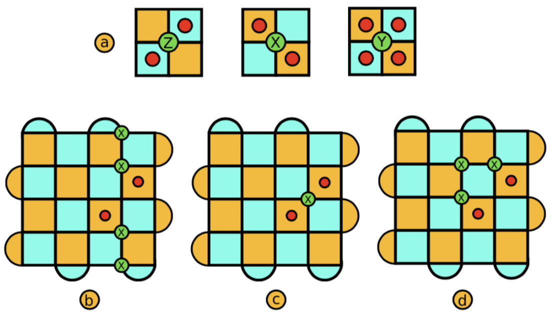

Figure 2. (a) represents the syndrome pattern (red dots) introduced by various physical qubit flips (Z, X and Y). (b) represents some examples of these qubit flips on the lattice. Notice that two syndromes on a single plaquette equals to no syndromes.

3. Rules of the game

To summarize the rules of the game, the opponent can make a move or a series of moves by introducing qubit flips X or Z (green circles). These moves (green circles) are hidden from the player but what the player can observe is the pattern of syndromes, denoted by red dots. The player's goal is to get rid of the red dots by making qubit flips. The goal is to have a clean board with no red dots (syndromes). However, if the player's qubit flips result in a string of X or Z qubit flips which connect the opposing boundaries like in Fig. 1b then the player loses.

Below are some examples of winning and losing games. Notice that in all three examples, the player was successfully able to get rid of the syndromes (red dots) but it lost in Fig. 5 as the player's move combined with the opponent's hidden move make a string of qubit flips connecting the opposite boundaries, hence breaking down the logical qubit and losing the game. Another thing to notice is that both Fig. 3 and Fig. 4, the player ends up winning by a different set of moves. As the opponent's moves are always hidden, there is no set strategy to win every time and a probability distribution of actions is required that can maximize the chance of winning over many runs.

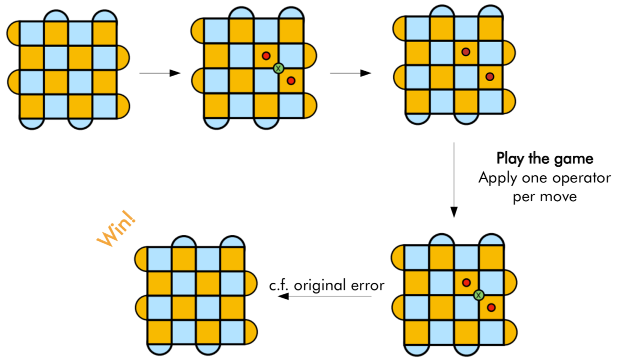

Figure 3. Example of winning the game.

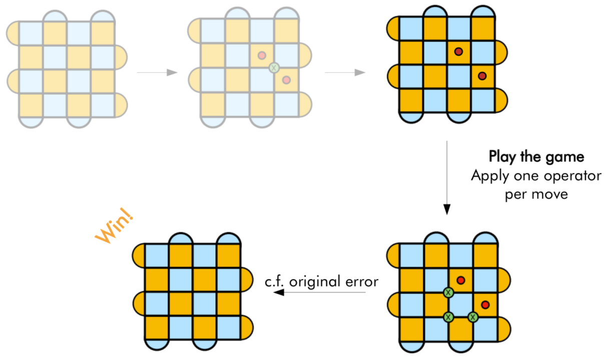

Figure 4. Example of winning the game with a different set of moves

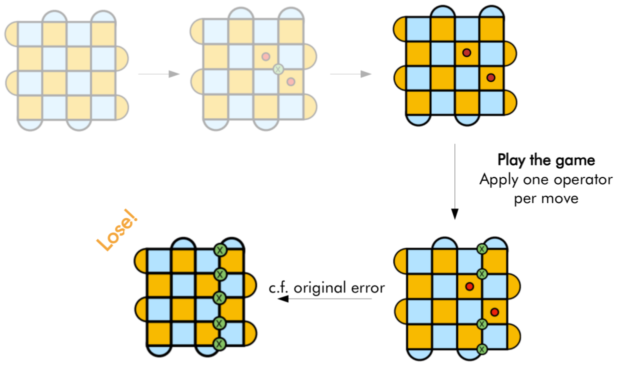

Figure 5. Example of losing the game. Although the player was successful in removing the syndromes, the moves of the player combined with hidden move of the opponent makes up a string of qubit flips and a logical error.

4. Q-learning and Deep-Q learning

Reinforcement learning and Q-learning are described in detail in the excellent book (Sutton & Barto, 1998). In this section, we will briefly describe some relevant details for this project.

Given a dynamic environment, an agent can transition between different states denoted by s by actions denoted by a. Reinforcement learning is a technique to find the optimal strategy for the agent, the policy $\pi(s, a)$ probabilistically describes actions for the agent when the system is in state s. The optimal policy is the one where the agent gives maximal return from its interactions with the environment. The agent receives a reward r_{t+1} when the agent interacts with the environment at time step t and and the system transitions from state $s_t \rightarrow s_{t+1}$. The reward accumulates over time and is given by $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots$. Here, $\gamma \leq 1$ and is called the discount factor. It quantifies the tradeoff between immediate reward vs reward accumulated later in time.

A useful way to represent the action based reward is the action-value function or the Q-function. This function $Q(s, a)$ quantifies the expected return when the agent in state s takes an action a following a policy π. For one-step Q-learning $Q(s, a) = r + \gamma max_{a'} Q(s', a')$ is the optimal policy based on the current estimate of Q when the agent transitions from state s to s' via action a. The values of the Q-function are updated for all states and actions and the system eventually converges to an optimal policy. To make exploration efficient, the tradeoff between explorative vs exploitative strategies is quantified through the $\epsilon - greedy$ policy. The system chooses an action randomly with probability ϵ and chooses the current estimate of the optimal policy with probability $1 - \epsilon$. This ensures that the agent does not spend extensive time on expensive moves but also does not missing out parts of the state-action space that are more rewarding.

For systems with a large state-action space like AlphaGo and other board games, it becomes increasingly impossible to store Q-function values. In deep Q-learning, the Q-function is stored in an efficient way by the deep neural network. The idea behind it is that because of similarities between different regions of the state-action space, the deep network is able to find patterns and store them in an efficient way. This is similar to how neural networks can identify increasingly more advanced features by finding patterns when classifying pictures, as the network becomes deeper. The Q-function $Q(s, a, \theta)$ for a convolutional neural network is parameterized by θ, which represents the complete set of weights and biases of the network. This can store the Q-function in a much more efficient way.

5. Fault-tolerant error correction

The environment (opponent) introduces a single or a series of qubit flips (X or Z) on the lattice of physical qubits. These flips are hidden from the player and his goal is to identify qubit flip moves that clear the board of physical qubits if syndromes. These moves could also be the same as the flip moves introduced by the opponent or a different set of moves as long as they do not introduce a logical error. The input to the agent is a syndrome pattern and output is a qubit flip pattern, that removes the syndromes.

Until now, we have only considered external noise that caused syndromes in our qubit system and we called them physical errors. However, in a more realistic model, there is also the possibility of noise due to measurement errors. It is like playing against two opponents who can take turns and introduce syndromes into the qubit and it is almost impossible to do error-correction in one turn. Fig. 6 shows how opponent 1 introduces noise (represented in green) due to physical errors and introduces syndrome into the system which we denote by true syndromes. The syndrome pattern is further perturbed by the measurement error (represented in red). As there is not enough information for the player to make a move that gets rid of the errors. The player uses a different strategy where it lets the opponents make their moves and keeps gathering information in the form of slices of syndrome patterns. There is now more information available for the player and it can make its error correction move to get the clear board.

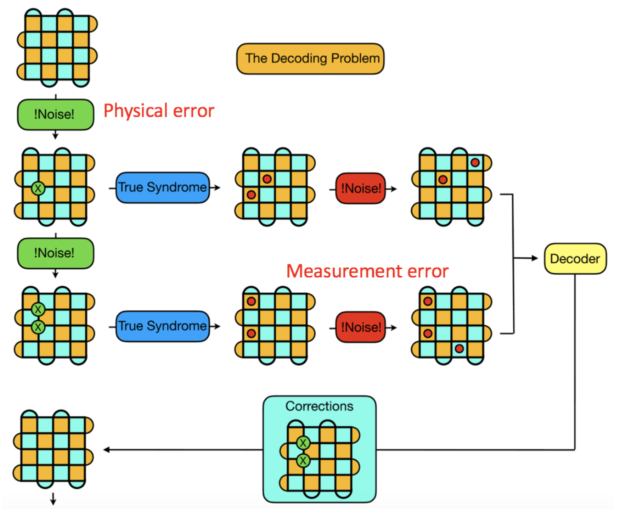

Figure 6. This is the case where opponent introduces both physical and measurement errors due to noise. Subsequent layers of syndromes are fed to the decoder to do corrections.

Unlike the game with just physical errors, the player does not win when it clears the board of syndromes. In this case, the player lets the opponent keep on introducing more noise (syndromes) even if it has cleared the board. The game only ends in a loss when a string of flips is introduced on the physical qubits which connect the opposite ends. This means that there is a logical error and the logical qubit is destroyed. The goal for the player is to keep the game running for as long as possible before a logical error is introduced.

This refers to the physical case of extending the lifetime of the qubit by error-correction. We want to have a logical qubit with as long a lifetime as possible and protect it from decoherence due to noise introduced by the environment and measurement.

6. Methods

We will now define the state space $\mathcal{S}$, action space $\mathcal{A}$ and the terminal state $S_{terminal} \subset \mathcal{S}$. At time step *t*, the environment is acted upon by action A_t and the environment generates a tuple $[S_{t+1}, R_{t+1}, T_{t+1}]$. In this tuple S is the state, R is the reward and T is a boolean value denoting if it is a terminal state or not. For the syndrome volume case, the state space consists of $S_t = \{S_{sv,t}, h_t\}$ where $S_{sv,t}$ is the syndrome volume at time step t and h_t is the action history, a list of actions performed by the agent since the syndrome volume was generated. If all errors have been corrected and there is no logical error, then the reward $R_{t+1} + 1$, otherwise $R_{t+1} + 0$. If the decoder is succesfully able to decode the syndrome, then S_{t+1} is not a terminal state and $T_{t+1} = 0$. If the decoder incoreacctly decodes the syndrome, then $T_{t+1} = 1$ and the episode is over. As mentioned in the previous section, states are the state-volume consisting of action values (qubit flips) and slices of syndromes. Actions are the qubit flips that the agent can make.

The states now have a syndrome volume (consecutive slices of syndromes after both physical and measurement errors). We will now discuss how to encode this syndrome volume state. For a $d \times d$ surface code, there is $(2d+1)\times(2d+1)$ matrix for each syndrome layer that keeps track of the anyons and the stabilizer orientation. The number of syndrome layers can be user-defined and make up the syndrome volume. Similarly, two additional action history layers are also present which keep track of the qubit flips on each vertex. The action history for the two layers is also encoded in a $(2d + 1) \times (2d + 1)$ matrix, a layer for each Z and X qubit flips. Finally, the state volume is a combination of a user-defined number of syndrome layers and two action history layers as shown in Fig. 7.

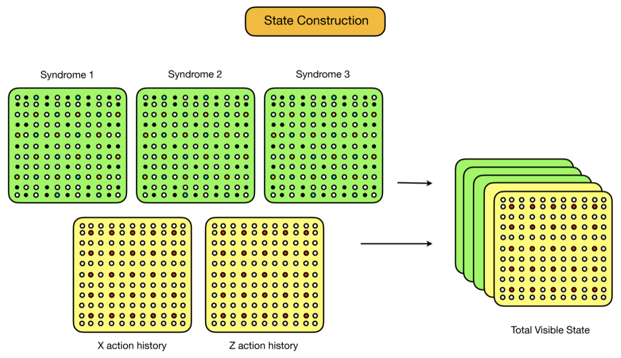

Figure 7. The state is constructed with a combination of syndrome layers and two action history layers. This combined state is the input to a convolutional neural network.

Now that we have encoded the volume state of the environment for each time step, a deep convolutional neural network can be used to parameterize the Q-function of our agent. This is illustrated in Fig. 8 where input is a state volume comprising syndrome layers and two action history layers, then there are a user-defined number of convolutional layers and feed-forward layers. The final output layer provides Q-values with respect to the input states. We used this technique as a black box and the approach is based on seminal work in references (Wang et al., 2016; Mnih et al., 2015).

These Q-values are generated at each time step and we can

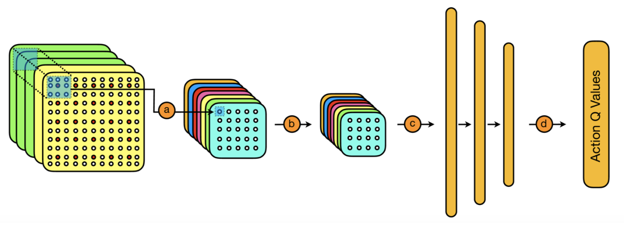

Figure 8. A convolutional Neural network with a combined state as input, a feed forward layer and the final output layer which encodes $q(S_t, a)$ for an action a

use the traditional Q-learning algorithms to find the optimal Q-functions. We use a $\epsilon - greedy$ policy, where the agent explores and chooses an action at random with probability ϵ. The agent will choose the maximum Q-value rest of the times (probability of $1 - \epsilon$).

7. Results

We implement deepQ learning in the framework described in the previous section. The following packages were used; Python3 (with Numpy and Scipy), tensorflow, keras-RL, and gym. We use a $d = 5$ surface-code lattice and $\epsilon - greedy$ policy. The discount factor $\gamma = 0.99$. Similar to reference (Sweke et al., 2018), the convolutional deepQ network consists of three convolutional layers, a single feed-forward layer, and final output layer. A single convolutional layer is described with $[n, w, s]$, where n is the number of filters, w is the filter width and s is the stride. The feed forward layer is described with $[n, d]$, where n is the number of neurons and d is the dropout rate. The deepQ network has the following structure,

$$[[64, 32, 2], [32, 2, 1], [32, 2, 1], [512, 0.2], [|\mathcal{A}|, 0]]$$

Here $|\mathcal{A}|$ is the size of the action space.

Two kinds of noise errors are considered, bit-flip noise (X qubit flip) and depolarizing noise (X, Z, or X and Z qubit flip(s) with equal probability). The probability of these physical errors is denoted by p_{phy}. Similar noise errors can also occur on measurement and are denoted by p_{meas}.

The goal is to maximize the average lifetime of the logical qubit. The lifetime is defined as the number of episodes it takes till the terminal state when a string of qubit flips connect opposing boundaries and a logical error is introduced. We plot the average qubit lifetime against distribution of errors with different probabilities. For simplicity, we consider the physical error p_{phy} and p_{meas} to be equal. We show two plots, for both bitflip noise and depolarizing noise as shown in Figs. 9 and 10. The red line represents the deepQ agent and the bluw line represents the case when no RL algorithm is employed and the logical qubit interacts with the environment. There is an almost two orders of magnitude of improvement in average qubit lifetime due to the deepQ error-correcting agent.

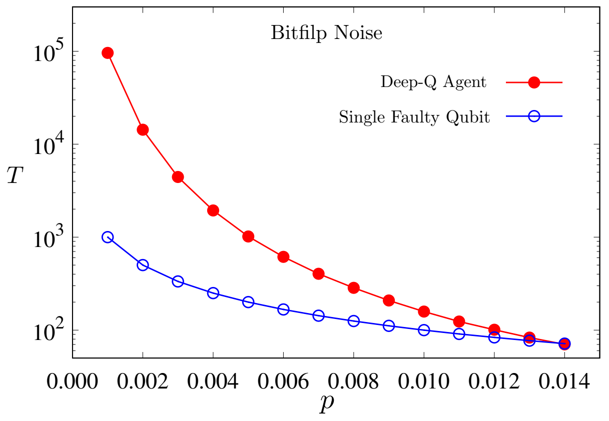

Figure 9. Bitflip noise model. Average qubit lifetime T plotted against different probabilities of physical and measurement errors $p = p_{phy} = p_{meas}$. Red line represents the performance of the agent trained on the deepQ network. Blue line represents the agent without any RL strategy.

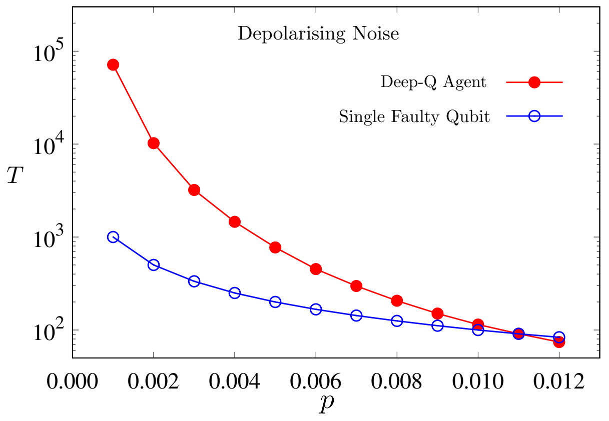

Figure 10. Depolarising noise model. Average qubit lifetime T plotted against different probabilities of physical and measurement errors $p = p_{phy} = p_{meas}$. Red line represents the performance of the agent trained on the deepQ network. Blue line represents the agent without any RL strategy.

8. Rainbow agent

One of the techniques we tried to use in this project is the Rainbow agent. Although we came close and spent a lot of time on it but in the end were not fully able to implement it. There are very few ready to use packages and the algorithm is too complicated to implement in a short period of time. We give a brief description of the Rainbow algorithm and why it is so appealing to implement.

Rainbow algorithm (Hessel et al., 2017) is an extension of DeepQ Network (DQN) algorithm with a combination of six advanced reinforcement learning algorithms proposed by the DeepMind group. The six algorithms are N-step return Q-learning, Distributional reinforcement learning, Dueling networks, Noisy Networks, Double DQN, and Prioritized Experience Replay. This agent is able to overcome several limitations of DQN algorithms and make full use of the advantages of the six algorithms. N-step returns Q-learning uses a n-step target for bootstrap. Double DQN can reduce the overestimation of Q-values. Polarized Experience Replay samples more of the states that have larger TD error in DQN. Dueling Network separates the Q network into state value and advantage value of action on that state then merges them together to get the Q-network. Distributional Reinforcement learning approximates the returns as a distribution instead of an expectation value. Noisy Networks introduce stochasticity to the weights and agents policy which can be used to aid efficient exploration.

In Atari games, the Rainbow agent shows much better performance than all the six algorithms training individually.

9. Conclusion and future work

In this project, we showed how reinforcement learning and deep neural network techniques can be combined to effectively train an agent to do optimal error-corrections in quantum computers for various noise models. We showed how the average qubit lifetime can be massively improved using this deepQ network agent. By mapping the quantum error-correction procedure to a board game, we were able to train the agent to come up with optimal strategies to do error-correction. We were able to reproduce the main results of (Sweke et al., 2018) and tried to implement some extensions by using more advanced DQN agents.

In the future, we would like to extend this model for other surface codes and noise models. In the current model, the logical qubit was stationary, there were no computing operations done with the logical qubit and in the future, we would like to study dynamic qubits. The RL algorithms only tried to maximize the lifetime of the logical qubit which is a first step towards building an autonomous and flexible agent for quantum error corrections. A future direction could be extending this RL model to also include error correction when quantum operations are being performed and the logical qubit is being manipulated to different states along the Bloch sphere. Another interesting avenue to explore would be surface codes with non-Abelian anyons (Nayak et al., 2008) which can possibly be used for universal quantum computation.

Recently, there have been many exciting works trying to incorporate quantum algorithms for machine learning applications which could lead to much faster algorithms. It would be interesting to see if a quantum analog of the RL algorithms can be theorized which may have an advantage over classical algorithms.

References

Andreasson, P., Johansson, J., Liljestrand, S., and Granath, M. Quantum error correction for the toric code using deep reinforcement learning, 2018. URL https://arxiv.org/pdf/1811.12338.pdf.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv e-prints*, art. arXiv:1710.02298, Oct 2017.

Kitaev, A. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, January 2003. doi: 10.1016/s0003-4916(02)00018-0. URL https://doi.org/10.1016/s0003-4916(02)00018-0.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. doi: 10.1038/nature14236. URL https://doi.org/10.1038/nature14236.

Nayak, C., Simon, S. H., Stern, A., Freedman, M., and Sarma, S. D. Non-abelian anyons and topological quantum computation. *Reviews of Modern Physics*, 80(3):1083–1159, September 2008. doi: 10.1103/revmodphys.80.1083. URL https://doi.org/10.1103/revmodphys.80.1083.

R. Sweke, M.S. Kesselring, E. v. N. J. E. Deepq decoding. https://github.com/R-Sweke/DeepQ-Decoding, 2018.

Shor, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE Comput. Soc. Press, 1994. doi: 10.1109/sfcs.1994.365700. URL https://doi.org/10.1109/sfcs.1994.365700.

Sutton, R. S. and Barto, A. G. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 0262193981. URL http://www.worldcat.org/oclc/37293240.

Sweke, R., Kesselring, M. S., van Nieuwenburg, E. P. L., and Eisert, J. Reinforcement learning decoders for fault-tolerant quantum computation, 2018. URL https://arxiv.org/pdf/1810.07207.pdf.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1995–2003. JMLR.org, 2016. URL http://dl.acm.org/citation.cfm?id=3045390.3045601.