Neural Network Based Decoder for Topological Codes

A Project Report

submitted by

DHEERAJ MOHANDAS PAI

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Neural Network Based Decoder for Topological Codes**, submitted by **Dheeraj Mohandas Pai**, to the Indian Institute of Technology, Madras, for the award of the dual degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Pradeep Kiran Sarvepalli
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT Madras, 600 036

Dr. Kaushik Mitra
Research Guide
Assistant Professor
Dept. of Electrical Engineering
IIT Madras, 600 036

Place: Chennai

Date: 4th June 2019

ACKNOWLEDGEMENTS

I express my sincere gratitude to my advisor Dr. Pradeep Sarvepalli for guiding me through this project and helping me conceptualise, understand the importance of the problem and motivating me through out the project. My heartful thanks to my co-advisor Dr. Kaushik Mitra for his invaluable inputs in Deep Learning without which I could not have proceeded in this inter-disciplinary project.

Special thanks to Chaitanya for his insightful discussions and his help in almost every hurdle in the project. Without his participation the project would have remained incomplete.

I am grateful to my labmates Abhishek and Arun for their useful insights. I thank my department and institute for providing me with opportunity and excellent resources to carry out this research work. I thank my friends for making five years of institute life joyful and memorable.

Last but not the least, I thank my parents and my sister for their constant emotional support throughout my stay in institute.

ABSTRACT

KEYWORDS: Quantum Error Correction, Neural Networks, Deep Learning, Topological Codes, Stabilizer Codes, Toric Codes, Subsystem Codes, Subsystem Color Codes

Recently, several deep learning techniques have been applied in quantum error correction. Torlai and Melko were the first to design a neural decoder for quantum codes. Varsamapoulus *et al.* came up with a two step decoder for quantum codes. Maskara *et al.* proposed a similar two step decoder and achieved near optimal threshold for triangular toric code and triangular toric code with a twist. Chinni *et al.* proposed a two step decoder based on pseudo-inverse of parity check matrix which achieved near-optimal threshold for color codes.

In this project we explore neural network based decoders for toric codes and subsystem color codes. We also take the two-step approach. The decoders proposed in Varsamapoulus *et al.*, Maskara *et al.*, Chinni *et al.* use a "naive" decoder in the first step. These naive decoders do not provide any threshold on their own. We implement a two step decoder where the naive decoder is replaced with a decoder that has a threshold. The motivation is to design a neural decoder with low training complexity. Our neural decoder for toric code achieves a threshold of 15.5% for depolarising error model. This performs better than the MWPM decoder which has a threshold of 14.8%.

We also explore the possibility of a neural decoder for subsystem color code. There has been no neural decoder for subsystem color code that has been designed till date. The existing state-of-art decoder provides a threshold of 1.9% while the theoretical upper limit for the threshold is 5.5%. The motivation is to find a neural decoder that can perform better than the state-of-art decoder. Our neural decoder for subsystem color code does not achieve any threshold for our proposed architecture and hyper-parameters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

DL	Deep Learning
ML	Machine Learning
QEC	Quantum Error Correction
QECC	Quantum Error Correcting Codes
FC	Fully Connected
NN	Neural Network
MWPM	Minimum Weight Perfect Matching

NOTATION

$\mathbf{A}$	Bold face capital letters denote matrices
$\mathbf{x}$	Bold face small letters denote row vectors
x_n	n^{th} element of $\mathbf{x}$
$\lvert x\rvert$	absolute value of x
$\mathbf{I}_N$	$N \times N$ identity matrix

CHAPTER 1

Introduction

Quantum computers compute and store data on qubits. Quantum error correction is used to protect the qubits from the external noise. Stabilizer codes are a class of quantum codes which are completely characterised by checks called as stabilizers.

Toric code and color codes are some examples of stabilizer code. Similar to stabilizer codes there is another class of codes known as subsystem codes. The subsystem code is completely characterised by a group of operators known as gauge operators. In this thesis we explore the neural decoder for toric codes and subsystem color codes.

1.1 Neural network based decoders

Recently, data-driven based neural network decoders have been proposed for quantum-error-correction for various codes and noise models by Torlai and Melko (2017); Varsamopoulos *et al.* (2018); Krastanov and Jiang (2017); Baireuther *et al.* (2018*a*); Chamberland and Ronagh (2018); Davaasuren *et al.* (2018); Jia *et al.* (2018); Breuckmann and Ni (2018); Baireuther *et al.* (2018*b*); Maskara *et al.* (2018); Chinni *et al.* (2019). Among them, Maskara *et al.* (2018); Chinni *et al.* (2019) have outperformed the non-neural decoders in terms of performance for various noise models on triangular color codes and triangular toric code with a twist. Recent work by Chinni *et al.* (2019) propose a neural decoder based on pseudo-inverse of the parity check matrix. They achieve a near optimal threshold of 10% for independent bit-flip/phase-flip error model on periodic color codes.

In this project we explore neural network based decoders for toric codes and subsystem color codes. The work is inspired by the the two-step approach proposed used in Varsamopoulos *et al.* (2018); Maskara *et al.* (2018); Chinni *et al.* (2019). They use a "naive" decoder in the first step to get an error estimate. This error estimate has the same pure error as the original error. Thus we can eliminate the pure error with this estimate. In the process of eliminating the pure error the error estimate could have introduced a new logical error in addition to the original logical error. A neural network is used to estimate the overall logical error. Thus the combined setup of the naive decoder and the neural network provides the final error correction. In Varsamapoulus *et al.*, Maskara *et al.*, Chinni *et al.* the naive decoders used in step one do not provide any threshold on their own. The combination of the two decoders give the threshold. We implement a two step decoder where the naive decoder is replaced with a decoder that has a threshold. The motivation is to explore and understand the effect of a "good" step one decoder on performance and training complexity.

We also explore the possibility of a neural decoder for subsystem color code. The theoretical upper limit to the threshold for the subsystem color code is 5.5% (Andrist *et al.* (2012)) while the state-of-art decoder proposed in Bombin *et al.* (2012) provides a threshold of 1.9% . Moreover, there has not been any neural network decoder designed for subsystem color code yet. This motivates us to explore neural network decoders for subsystem color codes.

1.2 Contributions of the Thesis

In this work, we study the decoding problem for toric codes and subsystem color code. We propose a neural decoder toric codes which achieves a threshold of 15.5% on depolarizing noise model. Our decoder performs better than the MWPM decoder. We achieve this performance from a single neural network for all the depolarising error rate. We also propose data augmentation techniques that will significantly improve the performance of the decoder at a lower training cost.

We also explore the possibility to design a neural-network based decoder for subsystem color code. We analyse the performance of the neural network decoder based on inverse of parity check matrix. This work is motivated by the work Chinni *et al.* (2019) who propose a neural decoder based on pseudo-inverse of parity check matrix for color codes.

The proposed subsystem color code decoders do not achieve any threshold for our chosen architecture and hyper-parameters.

The thesis is organized as follows,

Chapter 2 introduces QEC, stabilizer codes and subsytem codes with emphasis on toric codes and subsystem color code. We describe the decoding problem and how it can be reduced to a classification problem which can be solved by a neural network.

Chapter 3 broadly introduces neural networks (NN) and deep learning (DL). We discuss on the architecture and training procedure focusing on the neural decoders implemented in the current work.

Chapter 4 discusses on the neural decoder for toric codes. We briefly describe the decoder model. We describe the step one decoder. We describe the architecture, hyper-parameters and the training procedure used in the neural network. Finally, we discuss about the results.

Chapter 5 discusses on the neural decoder for subsystem color codes.We briefly describe the decoder model. We describe the step one decoder. We describe the architecture, hyper-parameters and the training procedure used in the neural network.

Chapter 6 provides the conclusion to the thesis.

CHAPTER 2

Quantum Error Correcting Codes

In this chapter, we summarize the necessary background on QECC. In section 2.1 we introduce the stabilizer formalism. In section 2.2 we describe how the quantum error correction problem can be reduced to a classification problem. In section 2.3 and 2.5 we introduce toric codes and subsystem color codes.

2.1 Stabilizer Codes

In this section we will briefly introduce stabilizer codes. Stabilizer codes are class of quantum codes which is completely characterised by checks called stabilizers. The Pauli group of 1 qubit is defined according to Eq. 2.1 where the Pauli operators X, Y, Z and the identity matrix I are defined in Eq. 2.2.

$$\mathcal{P}_1 \overset{\text{def}}{=} \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\} \tag{2.1}$$

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.2}$$

A Pauli group of n qubit $\mathcal{P}_n$ is the n- fold tensor product of $\mathcal{P}_1$.

$$\mathcal{P}_n = \mathcal{P}_1^{\otimes n} \tag{2.3}$$

Stabilizer codes are defined by an abelian subgroup of $\mathcal{P}_n$ called as stabilizers. We denote this subgroup as $\mathcal{S}$. The group $\mathcal{S}$ is considered up to a phase. The group $\mathcal{S}$ does not contain the element $-I$. The codespace, $\mathcal{C}$, is +1 eigenspace of $\mathcal{S}$.

$$\mathcal{C} = \{\, |\psi\rangle \in (\mathbb{C}^2)^{\otimes n} \mid S|\psi\rangle = |\psi\rangle \; \forall\, S \in \mathcal{S} \,\} \tag{2.4}$$

A centralizer of a subgroup A in a group B is the set of elements in B that commutes with every element in A. The centralizer $\mathcal{C}(\mathcal{S})$ of $\mathcal{S}$ in a group $\mathcal{P}_n$ is the set of elements in $\mathcal{P}_n$ that commutes with every element in $\mathcal{S}$. Mathematically, we can write,

$$\mathcal{C}(\mathcal{S}) = \{p \in \mathcal{P}_n |\, ps = sp \; for \; all \; s \in \mathcal{S}\} \tag{2.5}$$

Let $\mathcal{C}(\mathcal{S})$ be the centralizer of group $\mathcal{S}$ in the group $\mathcal{P}_n$. Elements of set $\mathcal{C}(\mathcal{S})\backslash\langle\mathcal{S}, \pm iI\rangle$ are the non-trivial logical operators, $\mathcal{L}$, and let $\mathcal{L}_g$ be it's generating set. The set $\mathcal{L}_g$ will has $2k$ generators, where k is the number of logical qubits. Each logical qubit i has two associated generators $\overline{X}_i, \overline{Z}_i$ for $1 \leq i \leq k$. $\overline{X}_i$ and $\overline{Z}_j$ commute if $i \neq j$ and anti-commute if $i = j$. A logical operator $\overline{L}_i$ will have the same effect on i^{th} qubit as the operator L will have on unencoded qubit.

The distance of the code d is an important characteristic of the code. The distance of the stabilizer code is defined as,

$$d = min\{\mathbf{wt}(e) | e \in \mathcal{C}(\mathcal{S})\backslash\langle\mathcal{S}, \pm iI\rangle\} \tag{2.6}$$

Stabilizer codes acting on n physical qubits and encoding k logical qubits and with a distance d are denoted by $[[n, k, d]]$.

Once we define $\mathcal{S}$ and it's generator set $\mathcal{S}_g$, we can define another set called pure errors, T with generator set $\mathcal{T}_g$. $\mathcal{T}_g$ consists of pure error $T_1, T_2 \ldots T_m$, where $m = |\mathcal{S}_g|$. A pure error T_i will anti-commute with exactly one stabilizer $S_i \in \mathcal{S}_g$ and commutes with all other stabilizers $S_j \in \mathcal{S}_g$. The pure error T_i will commute with other logical errors $L_i \in \mathcal{L}_g$ and pure errors $T_i \in \mathcal{T}_g$. Mathematically, we can define $\mathcal{T}_g$ as,

$$\mathcal{T}_g = \{\, \forall t \in \mathcal{T}_g \, \exists \, s \in S_g |\, ts = -ts, \forall s' \in S_g \backslash s, \; ts' = s't \,\} \tag{2.7}$$

We can observe that $\{\mathcal{S}_g, \mathcal{L}_g, \mathcal{T}_g\}$ together form the generating set of $\mathcal{P}_n$.

If E anti-commutes with the i^{th} stabilizer $S_i \in \mathcal{S}$ then, i^{th} bit of syndrome s_i is one. If it commuted then s_i is zero. Each stabilizer is analogous to a check in parity-check matrix in the classical error correction. Mathematically,

$$s_i = \begin{cases} 0 & \text{if } ES = SE \\ 1 & \text{if } ES = -SE \end{cases}$$

We represent the syndrome in a vector form $\mathbf{s} = (s_1, s_2, ..., s_m)$ where $m \geq n - k$.

2.1.1 Binary representation

Let, E be the error operator acting on the code with n physical qubits. The operator E can be written in binarized vector $\mathbf{e} = (e_1, e_2, ..., e_{2n})$ of length $2n$. Firstly, every Y operator is considered as a combination of X and Z as $ZX = iY$. For $1 \leq i \leq n$, $e_i = 1$ if E has a X operator on i^{th} qubit. For $n + 1 \leq i \leq 2n$, $e_i = 1$ if E has a Z operator on $(i - n)^{th}$ qubit. The first n entries of $\mathbf{e}$ correspond to the X operators in E and the next n entries of $\mathbf{e}$ correspond to the Z operators in E.

A stabilizer S acting on the code of n physical qubits can be written in a binarized form as a binary vector $\mathbf{h} = (h_1, h_2, ..., h_{2n})$ of length $2n$. The first n entries of $\mathbf{h}$ correspond to the X operators in S and the next n entries of $\mathbf{h}$ correspond to the Z operators in S. For $1 \leq i \leq n$, $h_i = 1$ if S has a X operator on i^{th} qubit. For $n + 1 \leq i \leq 2n$, $h_i = 1$ if S has a Z operator on $(i - n)^{th}$ qubit.

The matrix that contain all the vectors $h_1, h_2, ..., h_{2n}$ as rows can be considered as parity check matrix and is denoted by $\mathbf{H}$. Let λ be the matrix as defined in label 2.9. For an error operator E with the binarized form $\mathbf{e}$ the syndrome $\mathbf{s}$ can be calculated as,

$$\mathbf{s} = \mathbf{H}\lambda\mathbf{e}^\top \tag{2.8}$$

$$\lambda = \begin{pmatrix} 0 & I_n \\ I_n & 0 \end{pmatrix} \tag{2.9}$$

2.1.2 Decoding Problem

Let E be the error operator. Given the syndrome s the decoding problem is to add an error estimate $\hat{E}$ to eliminate the syndrome without adding any additional logical error.

We discussed in Sec. 2.2 that the error E can be decomposed uniquely into T,L and S. Where T is the pure error. L is the logical error and S is the stabilizer error. The stabilizer error can be ignored as it does not affect the quantum information. Hence, the decoding problem is to find an error estimate $\hat{E}$ such that $\hat{E} = ES'$, where S' is some stabilizer in $\mathcal{S}$.

2.2 QEC as a classification problem

In the previous section we discussed that $\{\mathcal{S}_g, \mathcal{L}_g, \mathcal{T}_g\}$ form a generating set of $\mathcal{P}_n$ we can write $E = TLS$ as shown in Duclos-Cianci and Poulin (2010). Here $T \in \mathcal{T}$, $S \in \mathcal{S}$ and $L \in \mathcal{L}$. All the T, L, S are a function of the error E. The effect of S is trivial implying two error patterns E and $E' = SE$ will have same effect on the quantum information. In other words, S will not affect the quantum information stored in the encoded system. The effect of L is non-trivial. Two error patterns E and $E' = LE$ will not have the same affect on the encoded quantum information. Given syndrome vector we can uniquely identify T. The problem of error correction for stabilizer codes is finding the most likely L upto a stabilizer S, as shown in Eq. 2.10. Hence, decoding can be thought of as a classification problem with $2^{|\mathcal{L}_g|} = 2^{2k}$ number of classes.

$$L = \underset{\gamma \in \mathcal{L}}{\operatorname{argmax}}\, Pr\left(\gamma \mid s\right) = \underset{\gamma \in \mathcal{L}}{\operatorname{argmax}} \sum_{\delta \in \mathcal{S}} Pr\left(\gamma\delta \mid s\right) \tag{2.10}$$

2.3 Toric codes

In this section we briefly introduce the 2D toric code with periodic boundary.

Toric code is a stabilizer code. The code is completely characterised by the set of stabilizers. Qubits are placed on the edges. The toric code has two types of stabilizers. The Z type stabilizers are on faces with one Z operator on each edge. The X type stabilizers are on vertices with one X operator on each incident edge. The stabilizers and qubits are illustrated in Fig. 2.1.

The toric code encodes two logical qubits. It has four independent logical operators, upto a phase. A logical operator affects the information that is encoded in a qubit. For

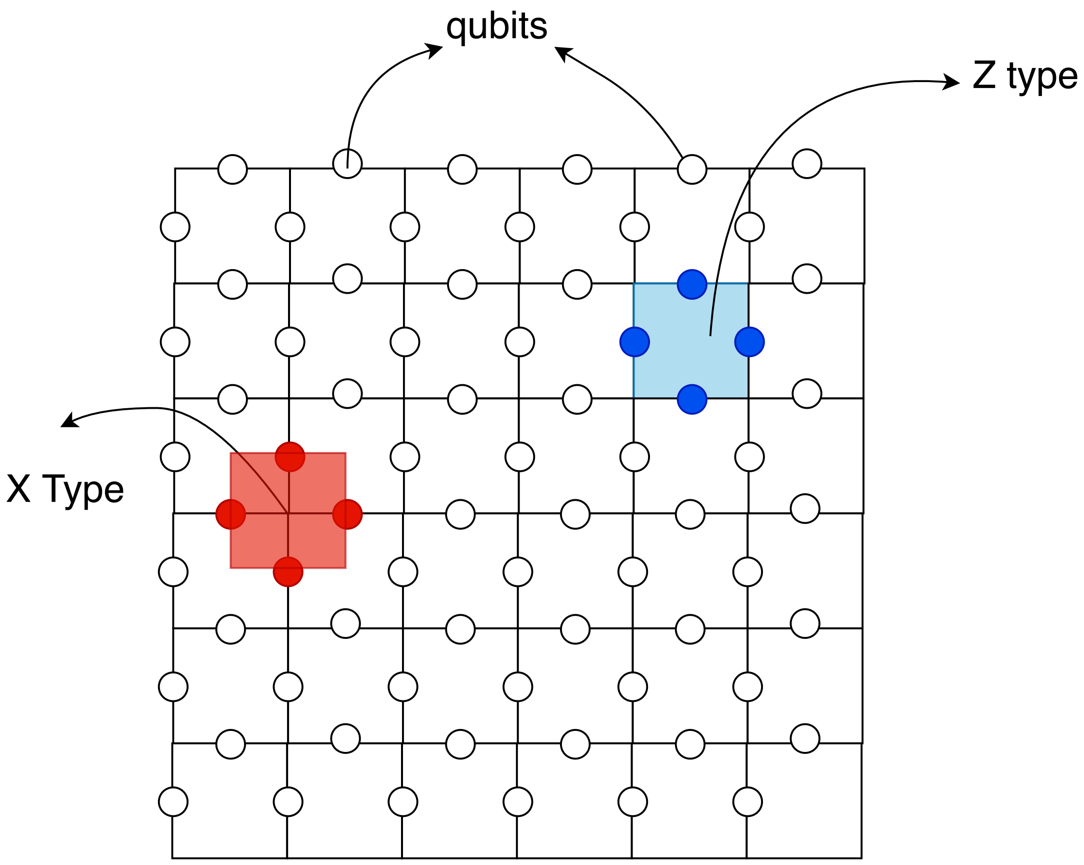


Figure 2.1: Periodic toric code on a illustrated with the qubits and the stabilizers (X and Z). Red dots and blue dots correspond to the X and Z operators on the physical qubits, respectively.

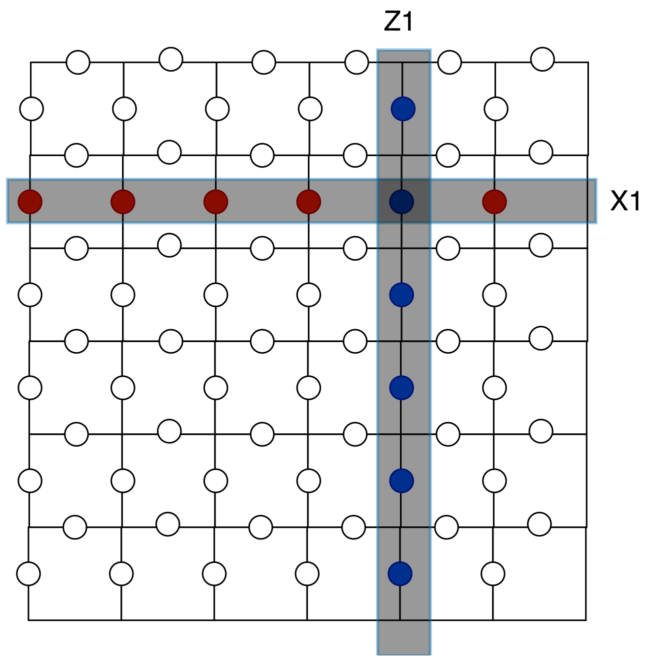
example, A logical Z operator operated on the encoded qubit has the same effect as an Z operator on an unencoded qubit. A logical operator commutes with the stabilizer but anti-commutes with one of the other logical operator. The logical operators of toric code is shown in the Figure 2.2.

We have to observe that multiple error pattern can result to the same syndrome.

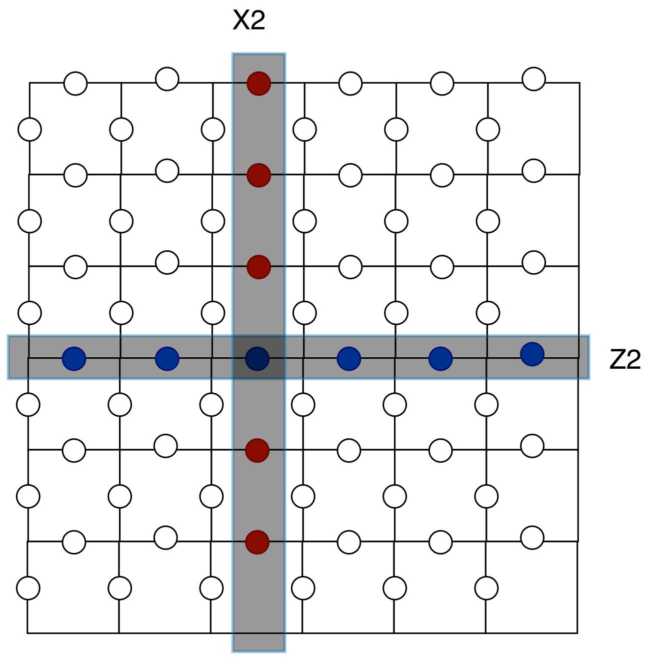
Consider the error pattern as shown in Fig. 2.3a. The blue dots indicate the Z error E. These errors cause the stabilizers to anti-commute with the error E and leave a syndrome. The violated checks are on the vertices, enclosed by the red box for better visualisation.

Consider a different error pattern E_1 as shown in the Fig. 2.3b. This error pattern also leaves the same syndrome as in Fig. 2.3a. Here, E and E_1 differ by a stabilizer S i.e, $E_1 = ES$.

Now, consider a different error pattern E_2 as shown in the Fig. 2.3b. This error



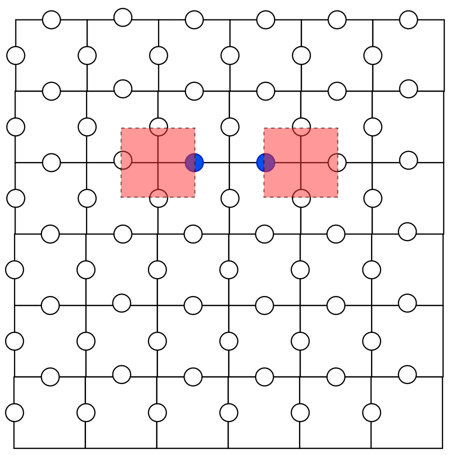
(a)

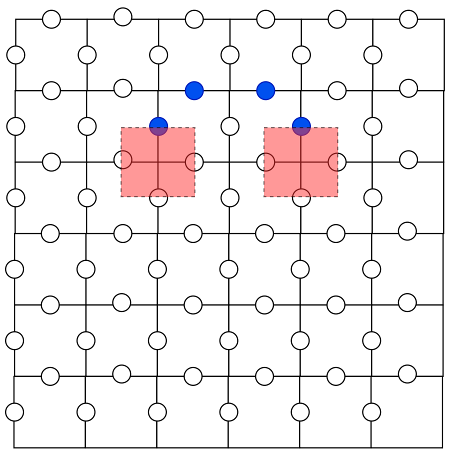


(b)

Figure 2.2: The toric code encodes two qubits. The logical operators of each qubit is shown in Fig 2.2a and 2.2b. A red dot and blue dot correspond to the X and Z operator on the physical qubit.

pattern also leaves the same syndrome as in Fig. 2.3a. Here, E and E_2 differ by a stabilizer S and a logical error L i.e, $E_2 = ELS$. Correcting the error E with E_2 will introduce a logical error.


(a) The toric code lattice with a phase-flip error E indicated using blue dots. The corresponding syndromes are indicated in the red boxes that enclose the vertices.


(b) The toric code with a different error E_1, where $E_1 = ES$. Both E and E_1 leave the same syndrome.

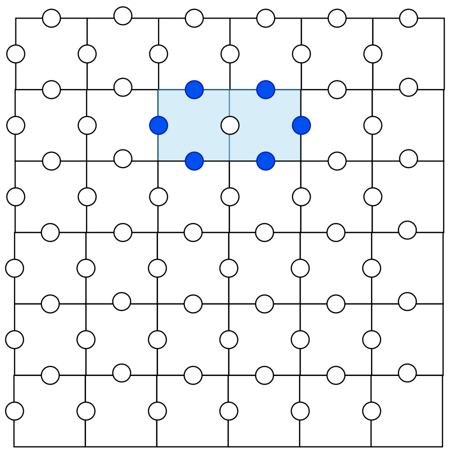
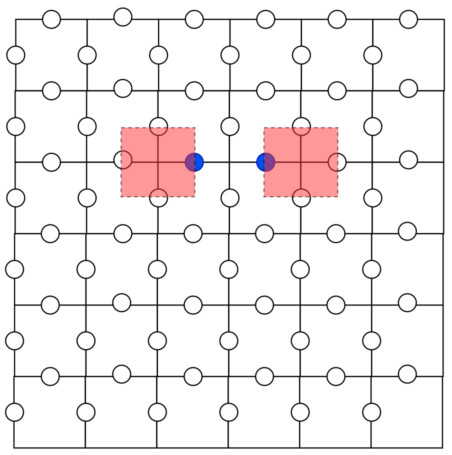
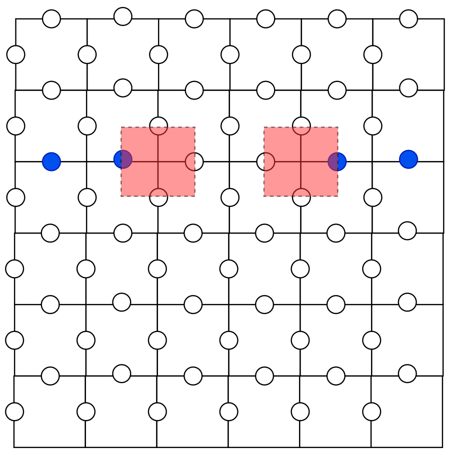

(c) The difference between E and E_1 is a stabilizer.

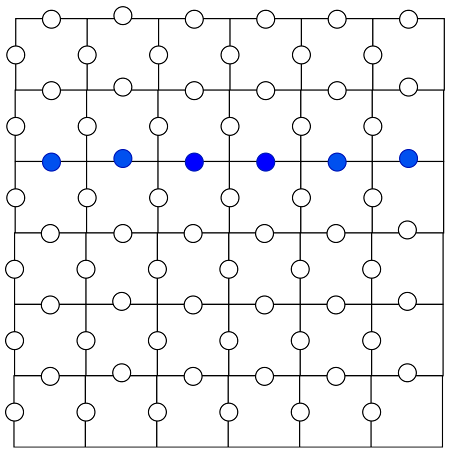
Figure 2.3: The error E and E_1 will leave the same syndrome. The difference between E and E_1 is a stabilizer. Correcting the error E with E_1 will not have any affect on the quantum information encoded in the system.



(a) The toric code lattice with a phase-flip error E indicated using blue dots. The corresponding syndromes are indicated in the red boxes that enclose the vertices.



(b) The toric code with a different Z error E_2, where $E_2 = E\overline{Z_2}S$. Both E and E_2 leave the same syndrome but will have different effect on the encoded quantum information.



(c) The difference between E and E_2 is a logical $\overline{Z_2}$.

Figure 2.4: The error E and E_2 will leave the same syndrome. But the difference between E and E_2 is a logical $\overline{Z_2}$. Correcting the error E with E_2 will introduce a logical Z on the second logical qubit.

2.4 Subsystem Codes

In this section we briefly introduce the subsystem codes. Subsystem codes were first introduced by Bombín (2010). Let $\mathcal{P}_n$ be the Pauli group on n qubits as described in 2.1. The subsystem color code is fully characterised by a non-abelian subgroup of $\mathcal{P}_n$ called gauge group $\mathcal{G}$. The stabilizer set $\mathcal{S}$ is the maximal abelian subgroup of $\mathcal{G}$ that does not contain $-I$.

Let $\mathcal{C}(\mathcal{G})$ be the centralizer of subgroup $\mathcal{G}$ in the group $\mathcal{P}_n$. We can write the group of stabilizers $\mathcal{S}$ as $\mathcal{S} = \mathcal{G} \cap \mathcal{C}(\mathcal{G})$, upto a phase. The group of gauge, stabilizers can be visualised by the Venn diagram 2.5. A detailed description of subsystem codes and it's construction is provided in Suchara *et al.* (2011); Gayatri and Sarvepalli (2018).

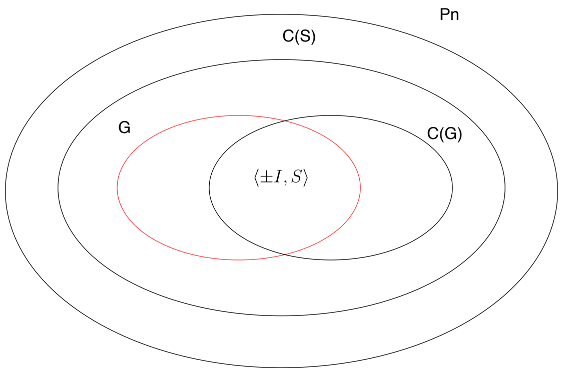


Figure 2.5: The Venn diagram illustrating the gauge group and stabilizer group in a subsystem code.

The codespace, $\mathcal{C}$, is +1 eigenspace of $\mathcal{S}$.

$$\mathcal{C} = \{\, |\psi\rangle \in (\mathbb{C}^2)^{\otimes n} \mid S|\psi\rangle = |\psi\rangle \;\forall\, S \in \mathcal{S} \,\} \tag{2.11}$$

The stabilizers form the checks of the subsystem code. The errors in $\mathcal{G}$ do not affect the information encoded in the code. The errors that do not belong to $\mathcal{C}(\mathcal{S})$ anti-commute with at least one element in $\mathcal{S}$ and hence are detected. The errors in $\mathcal{C}(\mathcal{S})\backslash\mathcal{G}$ are not detected by the checks.

2.4.1 Decoding Problem

Let E be the error operator. Given the syndrome s the decoding problem is to add an error estimate $\hat{E}$ to eliminate the syndrome without adding any additional logical error.

Similar to stabilizer codes we can we can uniquely decompose the error E into T,L and G. Where T is the pure error. L is the logical error and G is the gauge error. The gauge error can be ignored as it does not affect the quantum information. Hence, the decoding problem is to find an error estimate $\hat{E}$ such that $\hat{E} = EG'$, where G' is some gauge element in $\mathcal{G}$.

2.5 Subsystem Color Codes

In this section we briefly introduce the subsystem color codes. Subsystem color codes were first introduced by Bombín (2010).

The subsystem color code on the square-octagonal lattice is defined on a graph shown in figure 2.6. The qubits lie on the vertices. Observe that the graph has three kinds of edges. The dashed edge, solid edge and the hyper-edge (the triangle). The subsystem color code for square octagonal lattice has the gauge group which has three types of gauge elements, namely X, Y and Z type. The three types of gauges are shown in figure 2.7. The checks are the stabilizers shown in figure 2.9a. The stabilizers are derived from the gauges. An example is shown in figure. 2.8.

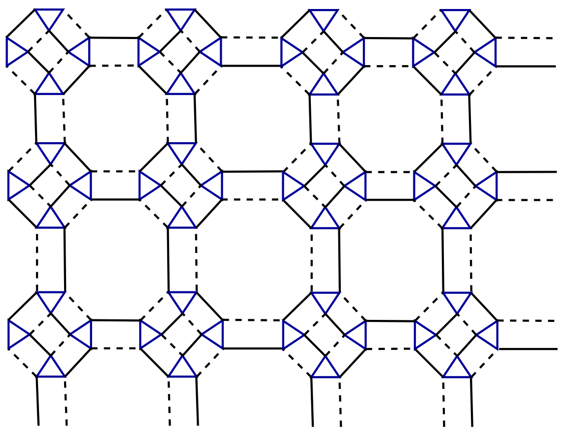


Figure 2.6: The figure illustrates the graphical structure of the subsystem color code. The Z-type gauges lie on the triangles and are shown in blue for better visualisation.

A triangle with two vertices with Z operator constitutes a Z-type gauge. We represent a X-type gauge with a solid line and Y-type gauge with dashed line. The square-octagonal subsystem color code encodes two logical qubits. It has four independent logical operators, upto a phase. A logical operator affects the information that is encoded in a qubit. For example, the logical $\overline{Z}$ has the same effect as an Z operator on an

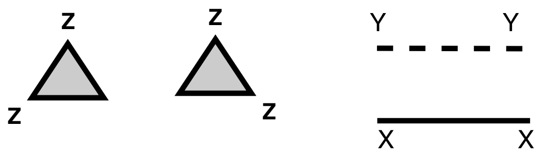


Figure 2.7: The figure illustrates the X, Y and the Z type gauge operators of the subsystem color code.

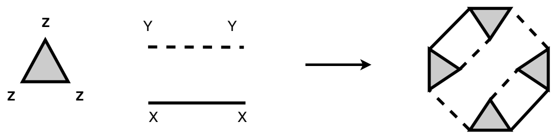
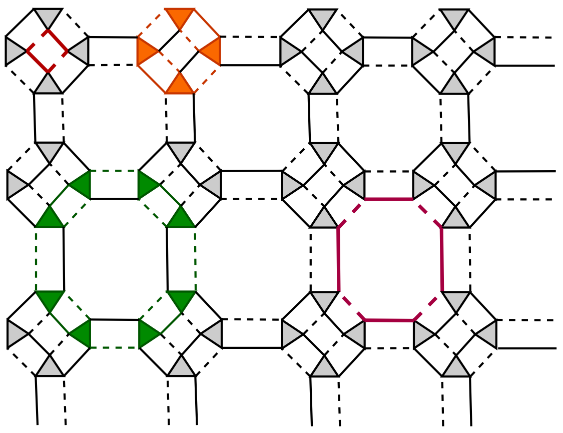
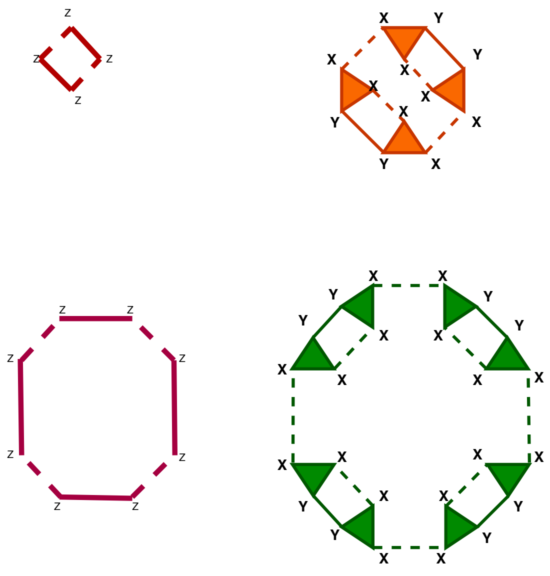


Figure 2.8: The figure illustrates the construction of stabilizer from the gauge group elements in the subsystem color code.



(a)



(b)

Figure 2.9: The figure illustrates the four types of stabilizers of the subsystem color code. Figure 2.9a shows the position of the stabilizers. The stabilizers are shown in color for better visualization.

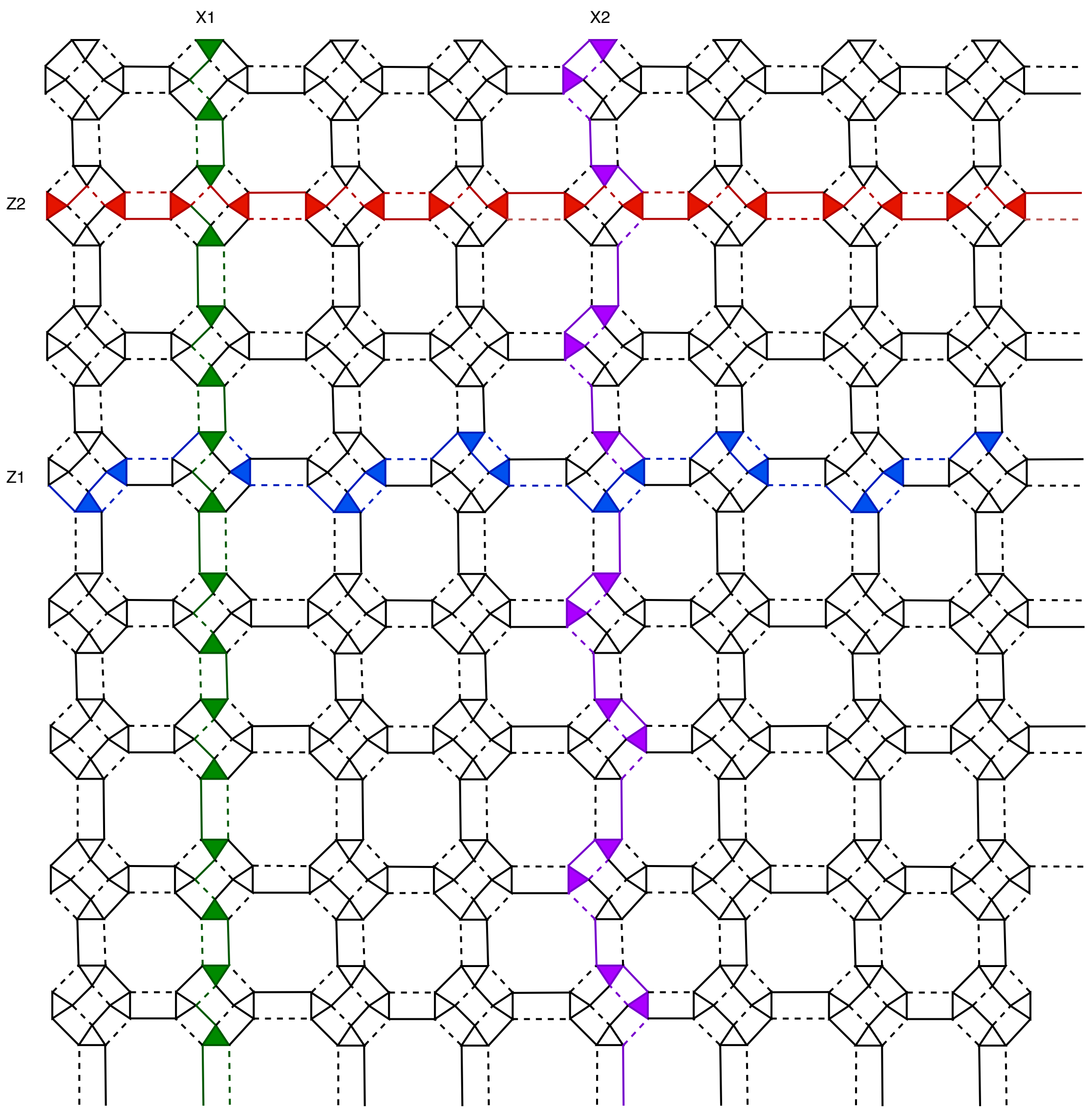


Figure 2.10: The figure illustrates the four independent logical operators of the subsystem color code.

unencoded qubit. A logical operator commutes with the stabilizer but anti-commutes with one of the other logical operator. The logical operators of subsystem color code is shown in the Fig. 2.10. Both stabilizers and the logical operators are hypercycles. The stabilizers act trivially on the codespace. The logical errors act non-trivially on the codespace.

CHAPTER 3

Neural Networks and Deep Learning

3.1 Overview of Machine learning

Traditional methods of computation (Non-machine learning algorithms) have explicitly mentioned steps to compute the output for a given input. In the case of data driven methods the algorithm learns these steps using the data. Machine learning (ML) algorithm belong to these data-driven methods. We do not explicitly program the but the algorithm learns these steps with the help of training data. ML algorithms have two steps. In the first step is called training. During training the machine learning model will take in sufficient amount data and learn the pattern the data represents. The second step is inference. During inference the trained machine learning model is used to predict the output for the given input. The performance of the machine learning model is measured by running the machine learning model for various inputs and calculating the accuracy of the prediction. In this chapter we focus on the specific framework of machine learning called neural networks.

3.2 Neural Networks

Artificial neural networks is a machine learning framework inspired by the neural circuit of the biological brain. A neural network is constructed by connecting number of basic unit called neuron. A neuron takes in input vector, multiplies it with a weight vector, adds a bias and sends it as an output (As shown in Fig. 3.1). The set of neurons which take the same input form a neural network layer. This output is passed through an non-linear function and then fed as input to the another set of neurons. This continues till the last layer where the neural network outputs the prediction. Fully connected neural network is a framework where each neuron in a layer takes input from every neuron of the previous layer. We use fully connected neural networks in our neural decoders.

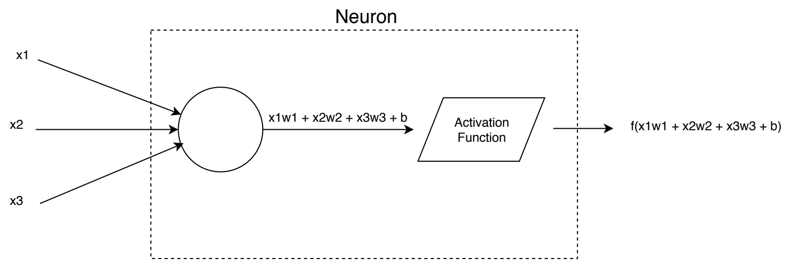


Figure 3.1: A single neuron in an neural network.

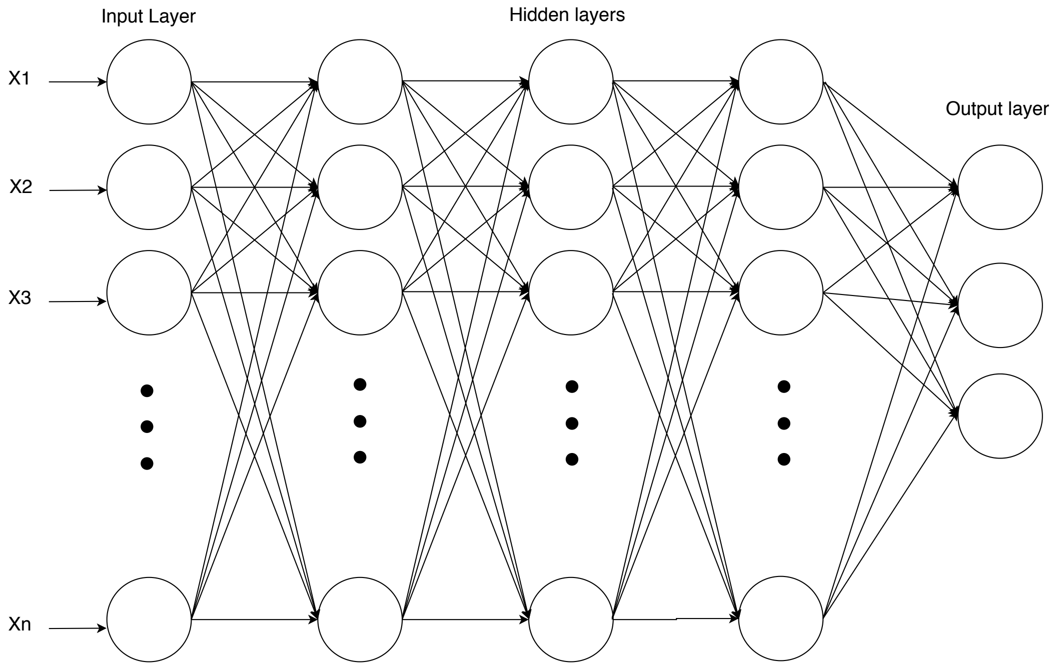


Figure 3.2: An example of a fully connected neural network.

Fig. 3.2 illustrates an example of fully connected neural network with three hidden layers.

In a classification problem the number of neurons in the output layer will be same as the number of classes. For the given input the neural network will output the probability for that particular class. In the case of the neural decoder the neural network must take in the syndrome and predict the homology. The number of neurons in the input layer will be the size of syndrome. The number of neurons in the output layer is the number of homology classes. The intermediate layers are called as hidden layers. The number of neurons in the hidden layer is defined by the user.

As discussed before the neural network has two phases. During training the neural network is given with labelled input and output samples. During inference the trained neural network will produce the output for the given input. The performance of the neural network is measured by calculating the accuracy of the prediction. In the case of our neural decoder the neural network is trained with the syndrome, homology pair. During inference the trained neural network will predict the correction homology for the given syndrome.

3.2.1 Activation function

A single layer of a neural network is nothing but a linear transformation. Two linear transformation on the same input space will still remain a linear transformation. Clearly having a neural network with no non-linearity will not be helpful in any real life prediction. To introduce non-linearity the output of every neuron is passed through a non-linear function called as an activation function. We use the `ReLU` activation function that is defined as Eq. 3.1 for hidden layers. And soft-max activation function for the output layer.

$$ReLU(x) = max(x, 0) \tag{3.1}$$

3.2.2 Loss Function

For a given input the neural network will predict an output. But this prediction need not always be the ground truth. A suitable function is used to quantify the errors in prediction. This function is called the activation function. Once the error is quantified using the loss function the aim of the neural network training would be to reduce the loss function. The loss function is computed at each step during training. The weights of the neural networks are updated to minimize the loss the function. Cross-entropy loss is a well suited for classification(Eq. 3.2). We use cross entropy loss for our decoders.

$$\ell_{CE}\left(\mathbf{y}, \hat{\mathbf{y}}\right) = -\sum_{i} y_i \log\left(\hat{y}_i\right) \tag{3.2}$$

3.2.3 Weight Initialization

Weight initialization plays a major role in determining the time taken for the training. Large weights significantly increase the value of gradient. Smaller weights will diminish the gradients. Both cases are not preferred while training. There are several techniques of weight initialization that overcome this problem. Xavier normal initialization is one such technique widely used for weight initialization (Glorot and Bengio (2010)). We initialize the weights according to Xavier normal initialisation in our neural decoder.

3.2.4 Data Augmentation

Training a neural network needs significant samples of labelled data. Sometimes obtaining labelled data would be computationally expensive. Data augmentation is a standard procedure used in deep learning community to increase the number of labelled dataset without significantly increasing the computational cost. For example, an image of a cat would remain an image of the cat if it is rotated by small angles. Fig. 3.3 shows an example of data augmentation. We exploit a similar technique for our toric code decoder to improve the performance of the neural decoder.

3.2.5 Training Procedure

The neural network is trained with the labelled data. At the very beginning, the neural network is initialised according to the Xavier's initialisation. In each iteration the input is given to the first layer. The output of the first layer is input to the second and so on. The output of the final layer is taken to compute the loss function. Once the loss function is computed the weights are updated such that the value of loss function is reduced. This is done by the gradient descent algorithm. The training is done till the loss function saturates. The entire training process is illustrated in Fig. 3.4

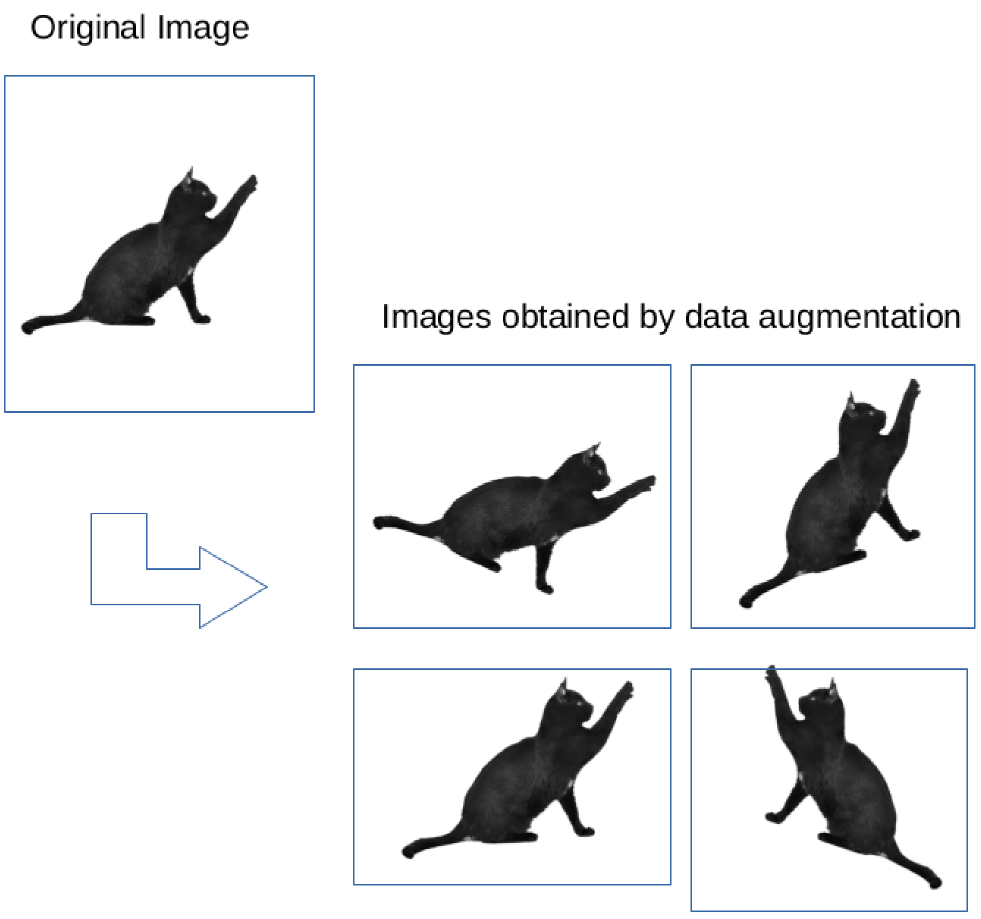


Figure 3.3: Data augmentation over an image of cat.

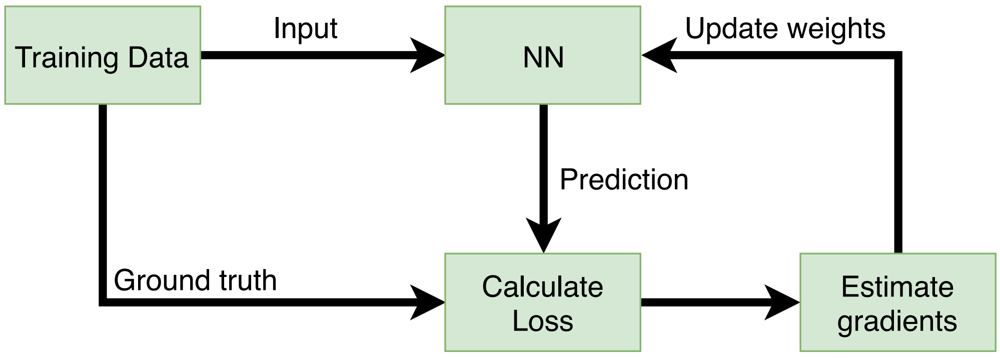


Figure 3.4: Training procedure of a neural neural network.

3.2.6 Curriculum Learning

Curriculum learning was first introduced in Bengio *et al.* (2009). It is a procedure where we train the neural network with "easier" patterns first and then proceed to "harder" patterns. The authors train neural network to predict the shape of the input image. It is shown that the neural network that is trained with "easier" shapes first and then trained over "harder" shapes perform better than the neural network trained randomly on any of the shapes. In our decoder we train the neural networks for lower error rate and then proceed to higher error rates. Same approach was taken by Maskara *et al.* (2018); Chinni *et al.* (2019) in the proposed neural decoder.

CHAPTER 4

Error Correction for Periodic Toric Code using Neural Networks

In this chapter we describe our work on the toric code. In section 4.1 we recollect the decoding problem for the toric code. In section 4.2 we describe how the two step neural decoder is modelled. We briefly describe the non-neural decoder. In section 4.3 we describe the neural network architecture and the hyper-parameters. In section 4.4 we explain the training procedure. In section 4.5 we explain the results obtained from our experiments.

4.1 Introduction

Let us recollect that decoding problem for the toric code can be considered as a classification problem as described in 2.2. Let E be the error operator. The error E can be decomposed uniquely into T,L and S. Where T is the pure error. L is the logical error and S is the stabilizer error. The stabilizer error can be ignored as it does not affect the quantum information. Hence, the decoding problem is to find an error estimate $\hat{E}$ such that $\hat{E} = ES'$, where S' is some stabilizer. Given the syndrome s the decoding problem is to add an error estimate $\hat{E}$ to eliminate the syndrome without adding any additional logical error.

$$E = TLS \tag{4.1}$$

Let E be the error operator and $\mathbf{H}$ be the parity check matrix computed according to 2.1.1. We can compute the binary representation of the error $\mathbf{e}$ according to 2.1.1. The syndrome s can be calculated as

$$\mathbf{s}^\top = \mathbf{H}\lambda\mathbf{e}^\top \tag{4.2}$$

.

A code of distance d that encodes k logical qubit in n physical qubits is denoted by $[[n, k, d]]$. The smallest number of physical qubits n to have a code of distance d is $2d^2$. Though out our experiments we take the code with the smallest number of physical qubits. As the number of physical qubits and stabilizers are dependent on the distance we take distance d as the free parameter. Hence the code of distance d will be $[[2d^2, 2, d]]$ code.

4.2 Decoder Model

The decoder is modelled in a two step process. Let E be the error operator and $\mathbf{e}$ be it's binarized form. In the first step we use the non-neural decoder to calculate an estimate $\left(\hat{E}\right)$ of the actual error (E). This approach is inspired by the previous works by Varsamopoulos *et al.* (2018); Maskara *et al.* (2018); Chinni *et al.* (2019). In the first step we use the minimum weight perfect matching (MWPM) decoder to calculate an estimate $\left(\hat{E}\right)$. In the second step, we use a neural network to predict the resultant logical error that could have occurred by the initial error E and estimated error $\hat{E}$.

$$\hat{\mathbf{e}}^\top = MWPM(\mathbf{s}^\top) \tag{4.3}$$

The decoder in step one finds an error estimate $\hat{E}$. This is done with the help of MWPM decoder. The syndrome pattern is transformed into a complete graph. Each violated check is a vertex v. Each edge $e = vv'$ will have the weight equal to the smallest number of qubit string that connect checks corresponding v and v'. An example of the transformation is given in Figure 4.2. Once the graph is constructed we obtain the edges corresponding minimum weight perfect matching using the MWPM algorithm. The error estimate $\hat{E}$ is nothing but the error operator on the qubits that correspond to the edges in the MWPM.

Both $\hat{\mathbf{e}}$ and $\mathbf{e}$ will leave the same syndrome. Hence, the pure error T corresponding to E and $\hat{E}$ will also be the same.

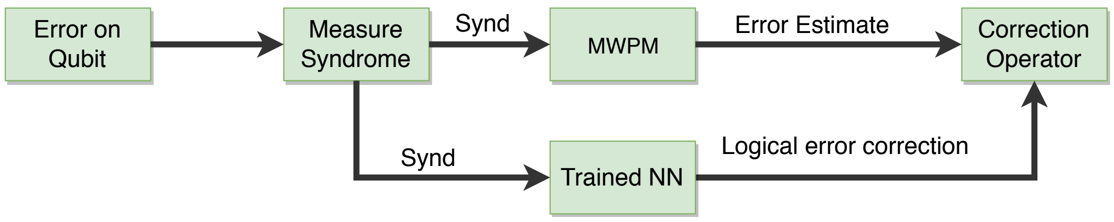


Figure 4.1: The two step framework used in the neural decoder for the toric code.

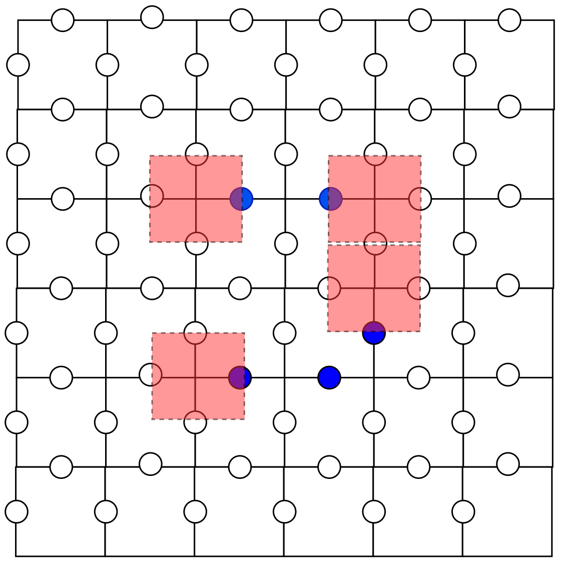
The error estimate ê and e need not be the same. There exist multiple error patterns that leave the same syndrome. Each syndrome correspond to a unique pure error T. The error operators $\hat{e}$ and E will have same pure error T but might have different logical errors. Applying this initial estimate $\hat{E}$ onto the system might result in logical errors. This can be concluded through the following equations,

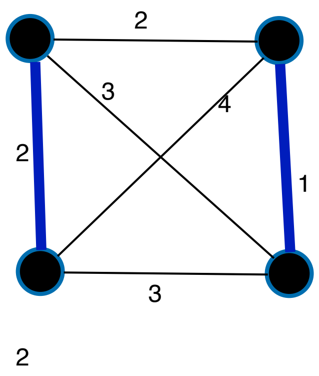
$$\begin{aligned} E &= TLS \\ \hat{E} &= T\hat{L}\hat{S} \\ \hat{E}E &= T\hat{L}\hat{S}\,TLS \\ \implies \hat{E}E &= (\pm)\,\hat{L}L\hat{S}S \\ \implies \hat{E}E &= (\pm)\,L^{\star}S^{\star} \end{aligned} \tag{4.4}$$

The above set of equations have been reproduced from Chinni *et al.* (2019) for the sake of completeness. The reason for occurrence of $(\pm)$ in Eq. (4.4) is because the Pauli operators T, $\hat{S}$ might commute or anti-commute. This is of little interest to us because we estimate the error up to a global phase.

$$\implies \mathbf{H}\hat{\mathbf{e}}^{\top} = \mathbf{H}\mathbf{e}^{\top} = \mathbf{s}^{\top} \tag{4.5}$$

By applying the estimate $\hat{E}$ onto the system the pure error is eliminated. The remaining errors are the logical error $L^{\star}$ and the stabilizer error $S^{\star}$. The stabilizer error $S^{\star}$ acts trivially on the codespace. If we can predict the resultant logical error $L^{\star}$ the decoding succeeds. The estimation of $L^{\star}$ is done by the Neural Network (NN). Given the syndrome, the neural network must predict $L^{\star}$. A neural network trained with samples of $\mathrm{s}^{\top}, L^{\star}$ can predict $L^{\star}$ given $\mathrm{s}^{\top}$. Our two step decoder can be illustrated in Fig. 5.1. The final error correction will be,


(a) Initial error pattern E and the corresponding syndrome



(b) The graph obtained from the syndrome pattern in 4.2a

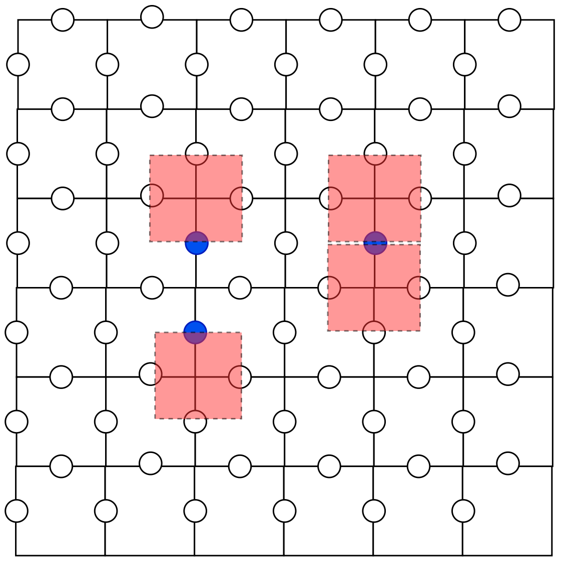

(c) The resultant error estimate $\hat{E}$ obtained from the MWPM algorithm. E and $\hat{E}$ differ by a stabilizer.

Figure 4.2: Finding an error estimate using MWPM algorithm.

$$E^\star = L^\star \hat{E} \tag{4.6}$$

By operating system with the error correction $E^\star$, the system gets into the original state up to a stabilizer and a phase as shown in Eq. 4.7.

$$E^\star E = L^\star \hat{E} E = (\pm)\, L^\star L^\star S^\star = (\pm)\, S^\star \tag{4.7}$$

4.3 Architecture

In this section we describe the architecure, hyper-parameters and the training procedure of the neural network used in the step-two. We use a fully-connected architecture where every neuron in one layer has input from every neuron of the previous layer. The output of the network is a vector. Each element of the vector correspond to a logical error operator. We follow the best practices that is followed by the deep learning community. We use cross-entropy as our loss function which is best suited for classification problem. We use the Adam optimizer to update the weights. Kingma and Ba (2014) show that the Adam optimizer performs better than other known optimizers in convergence of loss. We have 1D batch normalization layer after every layer in the neural network. Batch normalization has shown to boost training speed and convergence of loss (Ioffe and Szegedy (2015)). We use `ReLU` as the activation function for input and hidden layers. `ReLU` has shown to overcome the vanishing gradient problem that is pertinent in other activation functions like `Sigmoid` and `Tanh`.

Table 4.1: The values of the hyper-parameters used in the neural network used for the toric code decoder.

parameters / $[[n,k,d]]$[a]	h_d[b]	f_d[c]	b_d[d]	α[e]	$t_{d,p_{err}}$[f]	T_d[g]	N_t[h]
$[[50,2,5]]$	5	5	500	0.001	1×10^7	6×10^7	1
$[[98,2,7]]$	7	7	500	0.001	3×10^7	1.8×10^8	1
$[[162,2,9]]$	10	10	500	0.001	5×10^7	3×10^8	20

[a]Parameters of the code
[b]Number of hidden layers
[c]Hidden dimension factor
[d]Batch size
[e]Learning rate
[f]Number of training samples per each p_{err}
[g]Total number of training samples for all p_{err} combined
[h]Number of epochs

4.4 Training Procedure

The training samples are generated as follows. Error pattern E are randomly generated based on the depolarizing error probability p_{err}. The syndrome is computed according to the equation 4.2. The syndrome is input to the MWPM decoder to obtain the error estimate $\hat{E}$. The logical error caused by E and $\hat{E}$ is obtained by $E\hat{E} = L^{\star}S^{\star}$. We use data augmentation techniques to increase the total training dataset as described in section 4.4.1. We train the neural network with syndrome as the input and $L^{\star}$ as the output. The training procedure is illustrated in Figure. 4.3.

The neural network needs to be trained to predict the correct output. We have labeled data of input (syndromes s from Eq. (4.2)) and the corresponding output ($L^{\star}$).

We use cross-entropy function (ℓ_{CE}) as our loss function.

Once the loss is calculated the weights of the neural network are updated according to the update rule specified by the optimizer. The process flow for the neural network training is explained in 3.2.5.

With the syndrome vector s as the input, a trained NN should be able to correctly predict the correction homology class $L^{\star}$. We employ a curriculum learning procedure as described in 3.2.6. We generate training samples at a fixed depolarizing error rate p_{err}. We use a data augmentation technique describe in section 4.4.1 to increase the data samples. We train our NN for that error rate until the loss function saturates. We

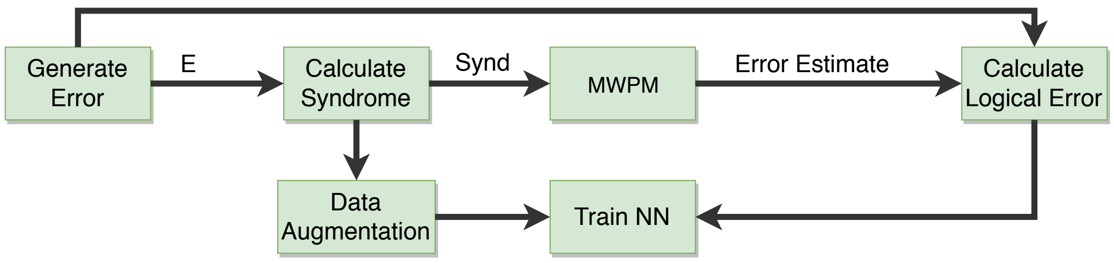


Figure 4.3: The training procedure for the toric code decoder.

then change the p_{err} to a higher value. The process is repeated for multiple error rates under the threshold. In our experiments, we have trained our NN for the error rates $\{0.14, 0.15, 0.16, 0.17, 0.18, 0.19\}$. We employ the standard best practices used by the machine learning community. We use Xavier normal initialization for the parameters in fully-connected layers and Gaussian normal initialization for the parameters in batch-normalization layer before we start training.

The hyper-parameters we have used for our networks are listed in the Table 4.1. b_d denotes the batch size. The input layer will have the dimension of the syndrome vector s i.e $|\mathrm{s}|$. f_d characterises the number of nodes in the hidden layers. α denotes the learning rate. The number of nodes in the hidden layer is f_d times the number of input nodes $= f_d \times |\mathrm{s}|$. $t_{d,p_{err}}$ denotes the training samples for each p_{err}. T_d denotes the total number of samples used. We use *PyTorch*[1], widely used open-source deep learning framework to train and test our neural networks.

4.4.1 Data Augmentation

In this section we describe our data augmentation technique. We show that this technique significantly increases the accuracy for a fixed training set.

The performance of neural network scales with the size of training set. This is considered as one of the major advantages of neural networks over traditional machine learning techniques. It is common procedure in deep learning to amplify the available data with simple techniques like translation, rotations etc.

In our case the training data is generated using the MWPM decoder which has a complexity of $O(n^3) = O(d^6)$. Where n is the number of qubits and d is the distance of the code. For a distance d code we show that we can amplify the data by a fac-

[1] https://pytorch.org/

tor of d^2 by shifting the syndrome pattern. Note that the code has periodic boundary due to which the shifting the syndrome pattern is possible irrespective of the position of the syndrome. As shifting the syndrome pattern does not change the relative distance between the stabilizers and hence the graph that is input to the MWPM does not change and so does the output. Thus, we can obtain more data samples without actually running MWPM. This technique will increase the number of data samples without significantly increasing the cost of generating data. Fig. 4.4 illustrates the example of the data augmentation.

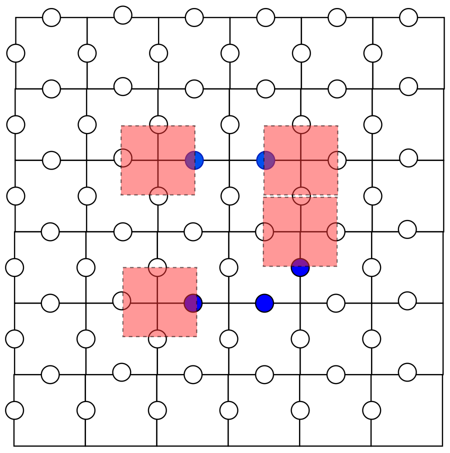
4.4.2 Analysis of data augmentation on toric code

We observe that the data augmentation technique significantly reduces the decoding error probability of the neural decoder. The following plot show the performance of neural decoder with and without data augmentation. Keeping all other parameters same.

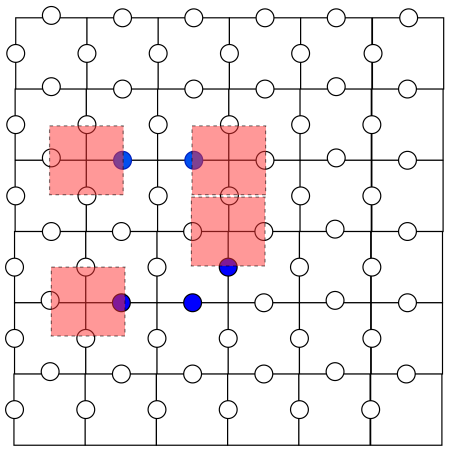
Table 4.2: The values of the hyper-parameters used in the neural network.

parameters / $[[n,k,d]]$[a]	h_d[b]	f_d[c]	b_d[d]	α[e]	$t_{d,p_{err}}$[f]	T_d[g]
$[[50,2,5]]$	7	7	500	0.001	3.8×10^6	2.38×10^7
$[[98,2,7]]$	7	7	500	0.001	9×10^5	5.4×10^6
$[[162,2,9]]$	9	9	500	0.001	5×10^5	3×10^6

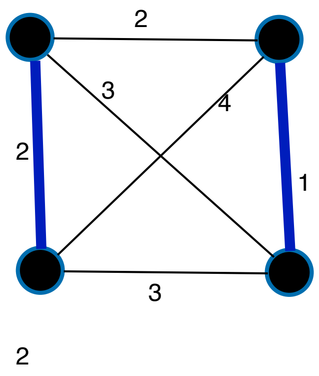
[a]Parameters of the code
[b]Number of hidden layers
[c]Hidden dimension factor
[d]Batch size
[e]Learning rate
[f]Number of training samples per each p_{err}
[g]Total number of training samples for all p_{err} combined



(a) Initial Syndrome pattern

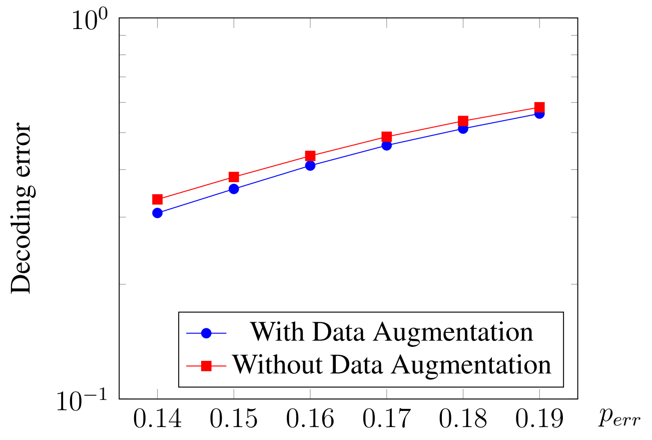


(b) Syndrome pattern after data augmentation.

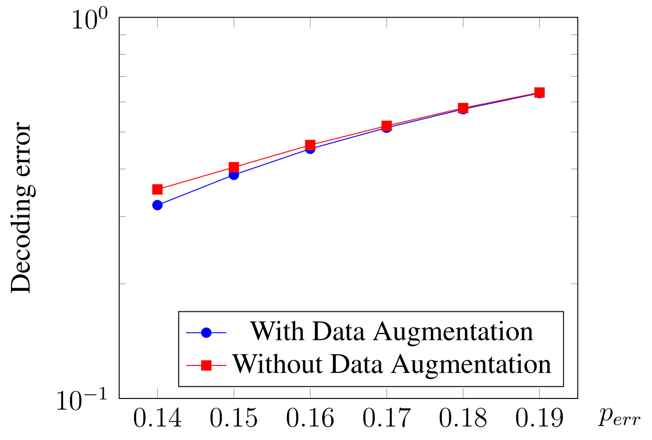


(c) The underlying graph generated to calculate MWPM which is same for both (a) and (b)

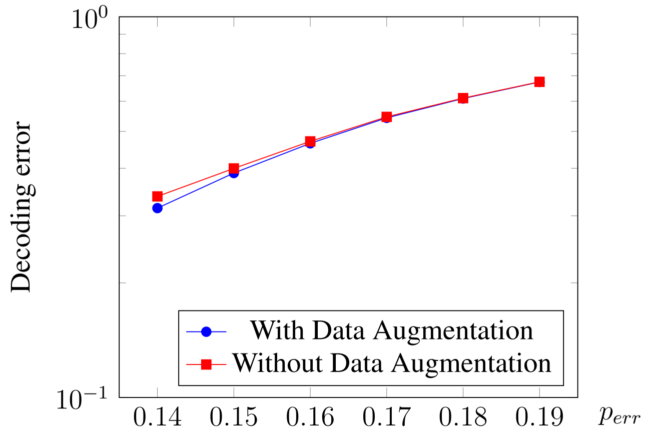
Figure 4.4: The syndrome pattern is shifted to it's left by one step from 4.4a to 4.4b and the initial estimate by MWPM E' is also shifted by one step. This because the underlying graph 4.4c used to calculate MWPM is unchanged.



(a) $[[50, 2, 5]]$



(b) $[[98, 2, 7]]$



(c) $[[162, 2, 9]]$

Figure 4.5: The decoding error rates of our neural decoder with and without data augmentation for distance 5 (4.5a), 7 (4.5b) and 9 (4.5c). The hyper-parameters used for the training is given in table. 4.2.

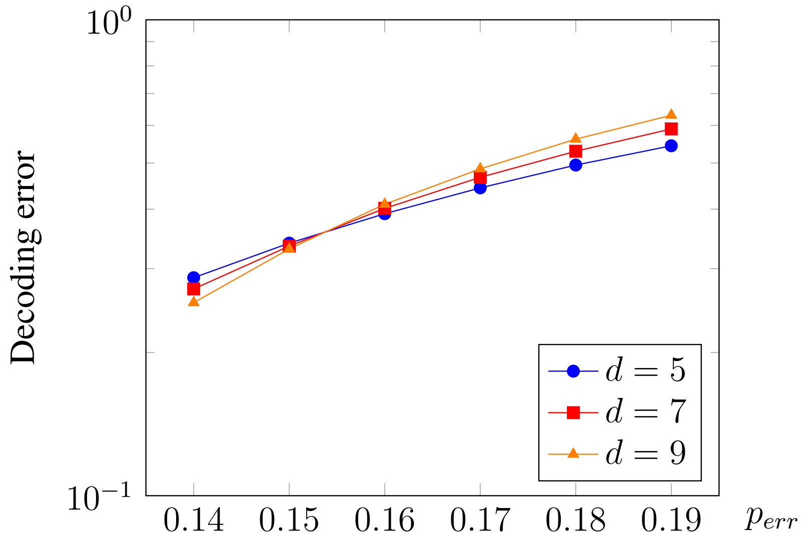
4.5 Results

We describe our simulation results for toric code decoder with depolarization noise model in this section. As described earlier in the Section 4.2, our decoder is a two step decoder where we use a MWPM decoder in step-one and then NN decoder in step two. The performance of MWPM decoder is shown in the Fig. 4.6. The MWPM decoder provides a threshold of 14.8%. Our neural decoder provides a threshold of 15.5%.

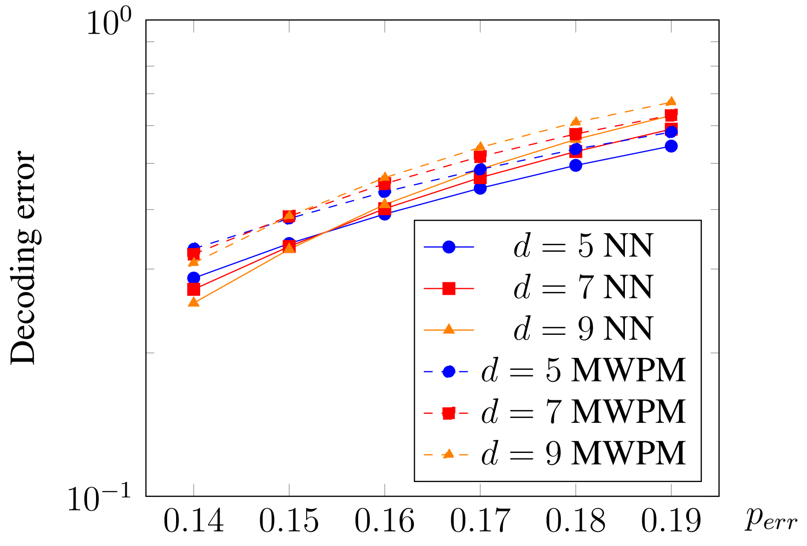
The performance of our fully-connected NN trained according to the training procedure mentioned in Section 4.4 is shown in the Fig. 4.6. We report the final threshold achieved by our neural decoder is 15.5% for depolarizing error model. We also observe that the neural decoder has lower decoding error rate compared to the MWPM decoder.

4.5.1 Comparison with other decoders

The proposed neural decoder for toric code achieves the threshold of 15.5% for depolarising noise model. The neural decoder for triangular toric code with a twist proposed by Maskara *et al.* (2018) achieves a threshold of 17.8% for depolarizing error. The neural decoder for triangular toric code with a twist proposed by Krastanov and Jiang (2017) achieves a threshold of 16.4% for depolarizing error. We also point out that MWPM decoder has much higher complexity than the naive decoder proposed in Maskara *et al.* (2018). This will significantly increase the training cost.



(a)



(b)

Figure 4.6: Figure. 4.6a shows the performance of our neural decoder. Figure 4.6b compares the performance of neural decoder with the MWPM decoder

CHAPTER 5

Error Correction for Subsystem Color Code using Neural Networks

In this chapter we describe our work on the subsystem color code. In section 5.1 we recollect the decoding problem for the subsystem color code. We briefly describe the construction of the non-neural decoder. In section 5.2 we describe how the two step neural decoder is modelled. In section 5.3 we describe the neural network architecture and the hyper-parameters. In section 5.4 we explain the results obtained from our experiments.

5.1 Introduction

In this section, we describe our problem formulation for correction of depolarizing errors on subsystem color codes.

Let us recollect that decoding problem for the subsystem color code can be considered as a classification problem as described in 2.4. Let E be the error operator. The error E can be decomposed uniquely into T,L and G. Where T is the pure error. L is the logical error and G is the gauge. The gauge can be ignored as it does not affect the quantum information. Hence, the decoding problem is to find an error estimate $\hat{E}$ such that $\hat{E} = EG'$, where G' is some gauge operator. Given the syndrome s the decoding problem is to add an error estimate $\hat{E}$ to eliminate the syndrome without adding any additional logical error.

Our subsystem color code decoder is inspired by the work of Chinni *et al.* (2019). They propose a neural decoder for topological color codes based on psuedo-inverse of parity check matrix. The decoder uses two-step approach. The first step with a non-neural decoder to eliminate T. The second step uses the neural network to predict L. We use the same technique to design a decoder for subsystem color codes.

$$E = TLG \tag{5.1}$$

Let E be the error operator. We can write the binarized form of E as $\mathbf{e}$ according to 2.1.1. Let $\mathbf{H}$ is the parity-check matrix for the subsystem color code computed according to 2.1.1. syndrome can be calculated as

$$\mathbf{s}^\top = \mathbf{H}\lambda\mathbf{e}^\top \tag{5.2}$$

In the case of subsystem color codes we have two dependent stabilizers. Hence, $\mathbf{H}$ is not full rank. We remove the two rows corresponding to two dependent stabilizers from the $\mathbf{H}$ matrix and denote it as $\mathbf{H}_f$. There is no specific rule to decide which two stabilizers should be removed. For our experiments we remove the last two stabilizers of the $\mathbf{H}$ matrix. The syndrome that is calculated with $\mathbf{H}_f$ is denoted by $\mathbf{s}_f$. $\mathbf{H}_f$ is a full rank matrix. Now, we can calculate the pseudo-inverse of $\mathbf{H}_f$ denoted as $\mathbf{G}$. We use in-built functions of *SageMath*[1] to compute the inverse.

$$\mathbf{H}_f\mathbf{G} = \mathbf{I} \tag{5.3}$$

$$\mathbf{s}_f^\top = \mathbf{H}_f\lambda_f\mathbf{e}^\top \tag{5.4}$$

A code of distance d that encodes k logical qubit in n physical qubits is denoted by $[[n, k, d]]$. The smallest number of physical qubits n to have a code of distance d is $3d^2$. The number of stabilizers would be d^2. Though out our experiments we take the code with the smallest number of physical qubits. As the number of physical qubits and stabilizers are dependent on the distance we take distance d as the free parameter. Hence the code of distance d will be $[[3d^2, 2, d]]$ code.

[1]`http://doc.sagemath.org/html/en/reference/matrices/sage/matrix/matrix_symbolic_dense.html`

5.2 Decoder Model

The decoder is modelled in a two step process. Let E be the error operator and $\mathbf{e}$ be it's binarized form. In the first step we use the non-neural decoder to calculate an estimate $\left(\hat{E}\right)$ of the actual error (E).

Given the syndrome $\mathbf{s}^\top$ we calculate $\mathbf{s}_f^\top$ by removing the entries corresponding the dependent rows. Now we calculate the error estimate $\hat{\mathbf{e}} \in \mathbb{GF}_2^n$, the binary representation of the operator $\hat{E}$ as follows,

$$\hat{\mathbf{e}}^\top = \lambda \mathbf{G}(\mathbf{s}_f^\top) \tag{5.5}$$

Both $\hat{\mathbf{e}}$ and $\mathbf{e}$ will leave the same syndrome. Hence, the pure error T corresponding to E and $\hat{E}$ will also be the same.

$$\begin{aligned} \mathbf{H}_f \lambda \hat{\mathbf{e}}^\top &= \mathbf{H}_f \lambda \mathbf{e}^\top = \mathbf{s}_f^\top \\ \implies \mathbf{H} \lambda \hat{\mathbf{e}}^\top &= \mathbf{H} \lambda \mathbf{e}^\top = \mathbf{s}^\top \end{aligned} \tag{5.6}$$

The error estimate $\hat{\mathbf{e}}$ and $\mathbf{e}$ need not be the same. There exist multiple error patterns that leave the same syndrome. Each syndrome correspond to a unique pure error T. The error operators $\hat{e}$ and E will have same pure error T but might have different logical errors. Applying this initial estimate $\hat{E}$ onto the system might result in logical errors. This can be concluded through the following equations,

$$\begin{aligned} E &= TLG \\ \hat{E} &= T\hat{L}\hat{G} \\ \hat{E}E &= T\hat{L}\hat{G}\,TLG \\ \implies \hat{E}E &= (\pm)\,\hat{L}L\hat{G}G \\ \implies \hat{E}E &= (\pm)\,L^\star G^\star \end{aligned} \tag{5.7}$$

The above set of equations are analogous to the two step decoder for stabilizer codes as shown in Chinni *et al.* (2019). The Pauli operators T, $\hat{G}$ might commute or anti-commute. Hence, we get $(\pm)$ in Eq. (5.7). But we do not need to estimate this as the error is estimated up to a global phase.

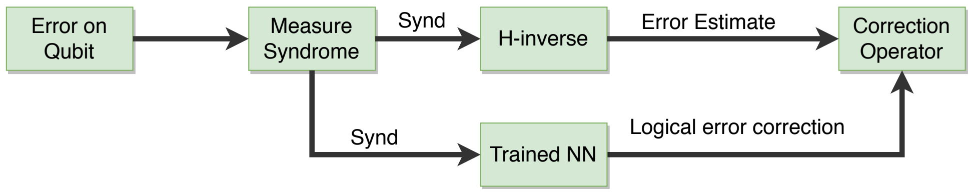


Figure 5.1: The two step decoder framework used for the subsystem color code

By applying the estimate $\hat{E}$ onto the system we observe the pure error is eliminated. If we can predict the resultant logical error $L^\star$ the decoding succeeds. The estimation of $L^\star$ is done by the Neural Network (NN). Given the syndrome, the neural network must predict $L^\star$. A neural network trained with samples of $\mathrm{s}^\top, L^\star$ can predict $L^\star$ given $\mathrm{s}^\top$. Our two step decoder can be illustrated in Fig. 5.1. The final error correction will be,

$$E^\star = L^\star \hat{E} \tag{5.8}$$

Given the syndrome it is always possible to get the error estimate $\hat{E}$. If the neural network predicts the correct resultant logical error $L^\star$ we can go back to the original state, up to a gauge and a phase as shown in Eq. 5.9.

$$E^\star E = L^\star \hat{E} E \tag{5.9}$$
$$\implies E^\star E = (\pm)\, L^\star L^\star G^\star \tag{5.10}$$
$$\implies E^\star E = (\pm)\, G^\star \tag{5.11}$$

5.3 Architecture and Training Procedure

In this section we describe the architecure, hyper-parameters and the training procedure of the neural network used in the step-two. The architecture used is similar to the architecture used for our toric code decoder as explained in section 4.3. We use a fully-connected neural network. We use cross-entropy as our loss function, Adam optimizer to update the weights, `ReLU` as the activation function for input and hidden layers. We use 1D batch normalization layer after every layer in the neural network.

The training samples for the neural network are generated as follows. Error pattern

Table 5.1: The values of the hyper-parameters used in the neural network.

$[[n,k,d]]$ [a] \ parameters	h_d [b]	f_d [c]	b_d [d]	α [e]	$t_{d,p_{err}}$ [f]	T_d [g]
$[[192,2,8]]$	3	3	500	0.001	1×10^6	4.5×10^7
$[[432,2,12]]$	5	5	500	0.001	1×10^6	4.5×10^7
$[[768,2,16]]$	7	7	500	0.001	4×10^7	1.8×10^8

[a]Parameters of the code
[b]Number of hidden layers
[c]Hidden dimension factor
[d]Batch size
[e]Learning rate
[f]Number of training samples per each p_{err}
[g]Total number of training samples for all p_{err} combined

E are randomly generated based on the depolarizing error probability p_{err}. The syndrome is computed according to the equation 5.4. The error estimate $\hat{E}$ corresponding to the syndrome is estimated with the pseudo-inverse of H as shown in 5.5. The logical error caused by E and $\hat{E}$ is obtained by Eq. 5.7. We train the neural network with syndrome as the input and $L^\star$ as the output. The training procedure is illustrated in figure. 5.2.

The neural network is trained according to the training procedure described in 3.2.5. We have labeled data of input (syndromes s from Eq. (5.2)) and the corresponding output ($L^\star$). We use cross-entropy function (ℓ_{CE}) as our loss function. The neural network takes syndrome (s), and outputs probability distribution over all the possible classes.

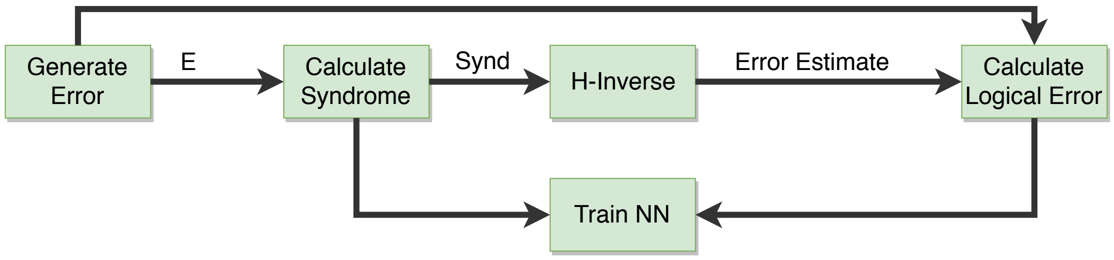


Figure 5.2: The training procedure for the subsystem color code decoder.

Given a syndrome vector s, a trained NN should be able to correctly predict the correction homology class $L^\star$ for all error rates under the threshold. We employ a curriculum learning procedure as described in 3.2.6. We generate training samples at a fixed depolarizing error rate p_{err}. We train our NN at this error rate until the loss function saturates. Then we increase the p_{err}. We repeat this process for several p_{err} in the increasing order. In our experiments, we have trained our NN for the error rates $\{0.005, 0.006, 0.007, \ldots, 0.05\}$. The weights of the fully connected layers are initialized according to the Xavier normal initialization as described in 3.2.3. The parameters in batch-normalization layer are initialized according to Gaussian normal initialization.

The hyper-parameters used for the neural networks are listed in the Table 5.1.

b_d denotes the batch size. The input layer will have the dimension of the syndrome vector s i.e $|\mathrm{s}|$. f_d characterises the number of nodes in the hidden layers. α denotes the learning rate. The number of nodes in the hidden layer is f_d times the number of input nodes $= f_d \times |\mathrm{s}|$. $t_{d,p_{err}}$ denotes the training samples for each p_{err}. T_d denotes the total number of samples used. We use *PyTorch*[2] to train and test our neural networks.

5.4 Results

In this section we describe our results for subsystem color code decoder with depolarization noise model. The performance of neural network decoder is shown in the Fig. 5.3. We observe that the decoding error rate is increasing with increase in distance. Hence, we conclude that the decoder does not have any threshold for the given set of hyper-parameters. We have not yet identified why the architecture has failed.

[2] `https://pytorch.org/`

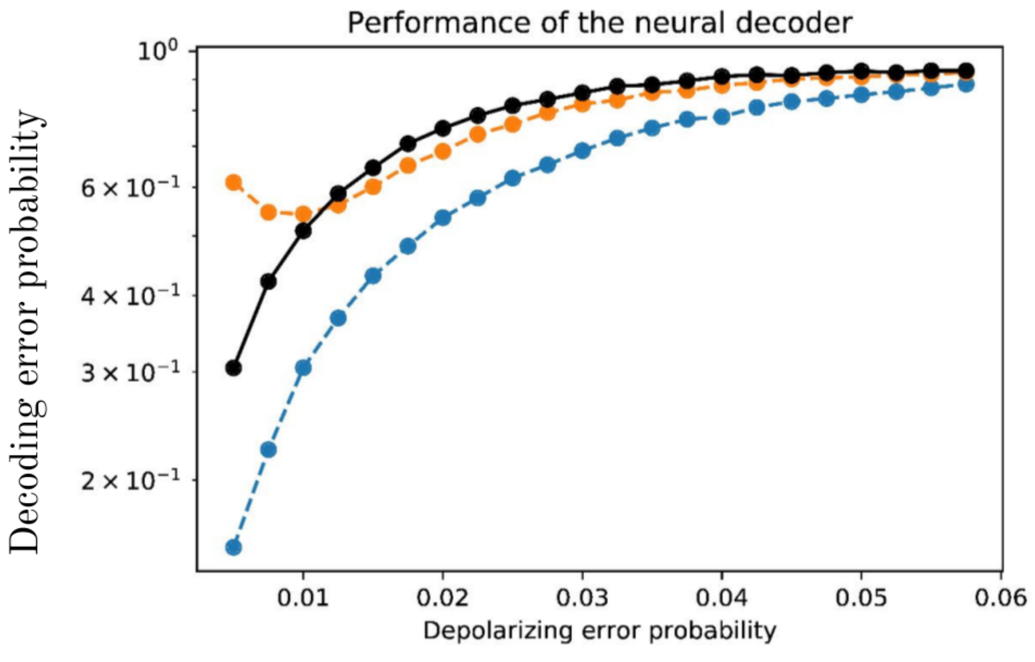


Figure 5.3: Performance of neural decoder for subsystem color.

CHAPTER 6

Conclusion

In this thesis we studied two classes of quantum codes. In section 4.5 we showed that our neural decoder performs better than the non-neural MWPM decoder both in terms of threshold and decoding error rate. But it does not outperform other neural decoders proposed in Maskara *et al.* (2018); Chinni *et al.* (2019). This is surprising considering the fact that Maskara *et al.* (2018); Chinni *et al.* (2019) use a naive decoder in step one while we use a much better performing decoder. We feel the following might be a potential reasons for the same. The decoders proposed in the Maskara *et al.* (2018); Chinni *et al.* (2019) use a "naive" decoder in the first step. The "naive" step-one decoder has very high decoding error rate. The neural network gets sufficient samples of almost all homology classes. But in the proposed decoder in 4 the MWPM decoder has a good decoding accuracy on its own. Hence, the dataset has overwhelming samples corresponding to logical error correction $L^\star = \overline{I}$. Essentially, the neural network does not get much "useful" data to correct the error made by the MWPM decoder. Another way to validate this is to replace the MWPM decoder with another decoder which has a good decoding accuracy.

The neural decoder for the subsystem color code does not achieve any threshold for our chosen hyper-parameters. As stated before, one the major challenges in designing neural decoder is the choice of architecture, hyper-parameters and the training procedure. For example, a "smart-sampling" technique provides the threshold in Krastanov and Jiang (2017) and the curriculum learning procedure increases the threshold from 7.2% to 10% in Chinni *et al.* (2019). We have chosen the hyper parameters comparable to that used in Chinni *et al.* (2019). We also observe that the curriculum learning performs better than training the neural network for one fixed depolarizing error rate p_{err}. We observe that as the curriculum learning progresses to higher p_{err} the accuracy of the neural network reduces for smaller p_{err}. One approach to overcome this would be to have different neural network for different p_{err}. As stated before, the architecture and hyper-parameters play a major role in designing the neural decoder. One can replace the fully connected neural network with a convolutional neural network(CNN) and deploy a much deeper network. CNNs are well suited to "learn" local correlations as shown in Krizhevsky *et al.* (2012). As the stabilizers and gauges in subsystem color codes are inherently local a CNN with suitable hyper-parameters might perform better.

REFERENCES

1. **Andrist, R. S.**, **H. Bombin**, **H. G. Katzgraber**, and **M. Martin-Delgado** (2012). Optimal error correction in topological subsystem codes. *Physical Review A*, **85**(5), 050302.

2. **Baireuther, P.**, **M. Caio**, **B. Criger**, **C. Beenakker**, and **T. O'Brien** (2018*a*). Neural network decoder for topological color codes with circuit level noise. *arXiv preprint arXiv:1804.02926*. URL https://arxiv.org/abs/1804.02926.

3. **Baireuther, P.**, **T. E. O'Brien**, **B. Tarasinski**, and **C. W. J. Beenakker** (2018*b*). Machine-learning-assisted correction of correlated qubit errors in a topological code. *Quantum*, **2**, 48. ISSN 2521-327X. URL https://doi.org/10.22331/q-2018-01-29-48.

4. **Bengio, Y.**, **J. Louradour**, **R. Collobert**, and **J. Weston**, Curriculum learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-516-1. URL http://doi.acm.org/10.1145/1553374.1553380.

5. **Bombín, H.** (2010). Topological subsystem codes. *Physical review A*, **81**(3), 032301.

6. **Bombin, H.**, **G. Duclos-Cianci**, and **D. Poulin** (2012). Universal topological phase of two-dimensional stabilizer codes. *New Journal of Physics*, **14**(7), 073048. URL https://doi.org/10.1088%2F1367-2630%2F14%2F7%2F073048.

7. **Breuckmann, N. P.** and **X. Ni** (2018). Scalable Neural Network Decoders for Higher Dimensional Quantum Codes. *Quantum*, **2**, 68. ISSN 2521-327X. URL https://doi.org/10.22331/q-2018-05-24-68.

8. **Chamberland, C.** and **P. Ronagh** (2018). Deep neural decoders for near term fault-tolerant experiments. *arXiv preprint arXiv:1802.06441*. URL https://arxiv.org/abs/1802.06441.

9. **Chinni, C.**, **A. Kulkarni**, and **D. M. Pai** (2019). Neural decoder for topological codes using pseudo-inverse of parity check matrix. *arXiv preprint arXiv:1901.07535*.

10. **Davaasuren, A.**, **Y. Suzuki**, **K. Fujii**, and **M. Koashi** (2018). General framework for constructing fast and near-optimal machine-learning-based decoder of the topological stabilizer codes. *arXiv preprint arXiv:1801.04377*. URL https://arxiv.org/abs/1801.04377.

11. **Duclos-Cianci, G.** and **D. Poulin**, A renormalization group decoding algorithm for topological quantum codes. *In 2010 IEEE Information Theory Workshop*. 2010. URL https://www.doi.org/10.1109/CIG.2010.5592866.

12. **Gayatri, V. V.** and **P. K. Sarvepalli**, Decoding topological subsystem color codes and generalized subsystem surface codes. *In 2018 IEEE Information Theory Workshop (ITW)*. IEEE, 2018.

13. **Glorot, X.** and **Y. Bengio** (2010). Understanding the difficulty of training deep feed-forward neural networks. *Journal of Machine Learning Research - Proceedings Track*, **9**, 249–256.

14. **Ioffe, S.** and **C. Szegedy** (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*. URL `https://arxiv.org/abs/1502.03167`.

15. **Jia, Z.-A.**, **Y.-H. Zhang**, **Y.-C. Wu**, **L. Kong**, **G.-C. Guo**, and **G.-P. Guo** (2018). Efficient machine learning representations of surface code with boundaries, defects, domain walls and twists. *arXiv preprint arXiv:1802.03738*. URL `https://arxiv.org/abs/1802.03738`.

16. **Kingma, D. P.** and **J. Ba** (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. URL `https://arxiv.org/abs/1412.6980`.

17. **Krastanov, S.** and **L. Jiang** (2017). Deep Neural Network Probabilistic Decoder for Stabilizer Codes. *Scientific reports*, **7**(1), 11003. URL `http://dx.doi.org/10.1038/s41598-017-11266-1`.

18. **Krizhevsky, A.**, **I. Sutskever**, and **G. E. Hinton**, Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*. 2012. URL `https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks pdf`.

19. **Maskara, N.**, **A. Kubica**, and **T. Jochym-O'Connor** (2018). Advantages of versatile neural-network decoding for topological codes. *arXiv preprint arXiv:1802.08680*. URL `https://arxiv.org/abs/1802.08680`.

20. **Suchara, M.**, **S. Bravyi**, and **B. Terhal** (2011). Constructions and noise threshold of topological subsystem codes. *Journal of Physics A: Mathematical and Theoretical*, **44**(15), 155301.

21. **Torlai, G.** and **R. G. Melko** (2017). Neural decoder for topological codes. *Phys. Rev. Lett.*, **119**, 030501. URL `https://link.aps.org/doi/10.1103/PhysRevLett.119.030501`.

22. **Varsamopoulos, S.**, **B. Criger**, and **K. Bertels** (2018). Decoding small surface codes with feedforward neural networks. *Quantum Science and Technology*, **3**(1), 015004. URL `http://stacks.iop.org/2058-9565/3/i=1/a=015004`.