MSc in Photonics

PHOTONICS**BCN**

Universitat Politècnica de Catalunya (UPC)
Universitat Autònoma de Barcelona (UAB)
Universitat de Barcelona (UB)
Institut de Ciències Fotòniques (ICFO)

http://www.photonicsbcn.eu

Master in Photonics

MASTER THESIS WORK

Quantum error correction - decoders

Eduard Bordas Marchante

Supervised by Dr. Felix Huber (ICFO) & Prof. Dr. Antonio Acín (ICFO)

Presented on date 9th September 2019

Registered at

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Quantum error correction – decoders

Eduard Bordas Marchante

ICFO - The Institute of Photonic Sciences, Mediterranean Technology Park, Castelldefels (Barcelona), Catalonia, Spain

E-mail: treballeduard@gmail.com

Abstract. Error correction is used to correct any errors that may arise when sending any message, as errors are prone to appear due to noise. For classical codes, there are many decoders used, for example, belief propagation. Unfortunately, these classical decoders are less efficient for quantum codes. A proposed remedy to improve the decoders efficiency is to use a neural network for decoding. In this thesis, we will implement belief propagation on the toric code and check its efficiency. We will see that for few errors, belief propagation works, but it fails for other cases. We also introduce the modifications needed for neural belief propagation, a modification of the original algorithm that integrates a neural network on the algorithm structure.

Keywords: Stabilizer operator, toric code, belief propagation, neural belief propagation.

1. Introduction

Error correction for quantum codes presents a different challenge compared to classical codes because a) we cannot make multiple copies of a code due to the no-cloning theorem, b) we cannot measure the code directly, as it would collapse the wave function and break the entanglement of the qubits, c) quantum errors are continuous, unlike classical errors which are discrete.

Thus, it is difficult to use classical decoding algorithms, like *belief propagation*, on quantum codes, like the *toric code*, with the algorithm performance often leaving much to be desired. However, due to the toric code simplicity, it provides a good framework on where to test modifications for the classical decoding algorithms to improve their performance at error correction for quantum codes.

With this idea, Liu and Poulin proposed an integration of a neural network into belief propagation, a *neural belief propagation*. The goal of this thesis is to reproduce some aspects from their paper [1].

The thesis is structured as follows; Section 1 provides an explanation of the toric code, Section 2 describes the belief propagation algorithm and its neural network implementation, Section 3 is about how to implement belief propagation into the toric code and the results obtained and the final Section is the conclusions.

2. The Toric code

2.1. Stabilizer codes

Before describing the Toric code, we review the framework of stabilizer codes.

The Pauli matrices are given by

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{1}$$

As with Pauli matrices, elements in the Pauli group (henceforth Pauli operators) are both unitary and Hermitian, and two Pauli operators must either commute or anti-commute. The n-qubit Pauli group is generated by $\mathscr{P}_n = \mathscr{P}_1 \times \mathscr{P}_1 \times ...$, where $\mathscr{P}_1$ is the Pauli group for one qubit, defined as $\mathscr{P}_1 = \langle i, X, Y, Z \rangle$.
The *stabilizer group* S is a commutative subgroup of the Pauli group, generated by the product of generators, where the generators that form the stabilizer group are independent, that is, no generator can be expressed as the product of the rest of generators. The number of elements in the stabilizer group will be given by $m = 2^{n-k}$, where n is the number of physical qubits and k is the number of generators [2]. The codespace is formed by all the states that are in the +1 eigenspace of the stabilizer group. Therefore, the expected value of S_j, a stabilizer operator, which is an element of the stabilizer group, is

$$\langle S_j \rangle = \langle \psi | S_j | \psi \rangle = 1\,, \tag{2}$$

where $|\psi\rangle$ is a state of the code.
Quantum codes can be described by their parameters $((n,k,d))_2$, where n is the number of physical qubits, k the dimension of the code space and d the distance. To define the distance we must introduce the weights. Weights of an *error* are the number of terms in the tensor product comprising the error which are not equal to the identity. Where we refer to an error as a Pauli operator that acts on the distribution of qubits that we are considering and it changes its state to something else. Thus, the distance is defined as the minimum weight of a set of tensor operators such that they commute with the stabilizer group. Also, should a code have a distance $d \leq 2t+1$, it will be able to correct arbitrary errors on any t qubits [3].
Stabilizers are used in error correction by employing the anti-commutation relationship of the Pauli matrices which form the stabilizer operators with the Pauli operators forming the errors. Ideally, the expected value of a stabilizer operator acting on a code with errors would be

$$\langle S_j \rangle = \langle \psi | E^\dagger S_j E | \psi \rangle = \langle \psi | - E^\dagger E S_j | \psi \rangle = -\langle \psi | S_j | \psi \rangle = -1\,, \tag{3}$$

where $|\psi\rangle$ is a state of the code, E is an error and S_j is a stabilizer operator. If an error E commutes with all the stabilizer operators S_j, then it could not be detected by measuring the stabilizers, because then we cannot distinguish the case of error E happening with an errorless case, as both give an eigenvalue of +1 when measuring the stabilizers operators.
The values we obtain when we measure the stabilizer operators are called *syndromes*. As the stabilizer operators act on multiple qubits, it is possible that different combinations of errors might give the same syndrome. This phenomena is called *error degeneracy*.
In order to perform logical operations on the encoded states, the *logical operators* are used. The logical operators must not be detected by the stabilizers, therefore, they must commute with the stabilizer group, such group is called the *normalizer group*.

2.2. The Toric code

The toric code is defined on a periodic $L \times L$ lattice where qubits are placed in each edge of the lattice.
In the toric code, the stabilizer operators take the form of *plaquette operators*, given by the tensor product of Pauli Z operators acting on the 4 qubits on the edges of the plaquette. And the *vertex operator*, given by the tensor product of Pauli X operators acting on the 4 qubits adjacent to a vertex [2]. This two operators are given by

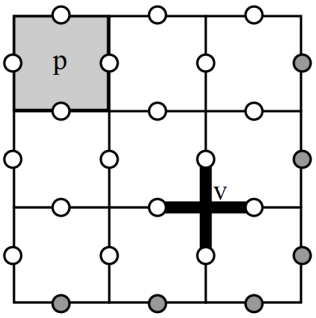


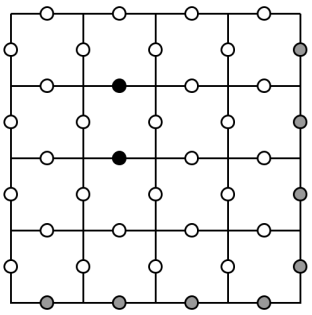
Figure 1: A lattice representing a toric code with 18 physical qubits, with a plaquette operator, indicated by p, and a vertex operator, indicated by v.

$$A_p = \prod_{i \in p} X_i \qquad\qquad B_v = \prod_{i \in v} Z_i \tag{4}$$

for the plaquette and vertex operators, as shown in Eqs. (4) at the left and right equation respectively. Figure 1 provides an example on how the stabilizer operators act on the toric code lattice. Because the toric code is defined in a periodic lattice, the qubits on the limits of the lattice are in fact the same qubit as the qubit in the opposite limit, indicated by the gray circles in Figure 1.
A toric code lattice of size L consists of a number of physical qubits n equal to $2L^2$, where we can only encode k qubits as defined by $m = n - k$, where m is the number of independent stabilizers operators. The easiest way to calculate the maximum m is by the product of all the stabilizer operators minus one. Therefore, the maximum m will be equal to $L^2 - 1$ for both plaquette and vertex operators. Thus, in the toric code we can encode $2L^2 - 2(L^2 - 1) = 2$ qubits, as m includes both plaquette and vertex operators. Consider a string of errors acting on the toric code, as seen in Figure 2; in such case, we will only measure a value of -1 for the syndromes connected to the ends of the strings of errors, as an even number of qubits with errors acting on the same stabilizer operator commute with the operator. Should a string of errors form a closed loop, it will not be detected by a syndrome measurement.

1	-1	1	1
1	1	1	1
1	-1	1	1
1	1	1	1

(a)

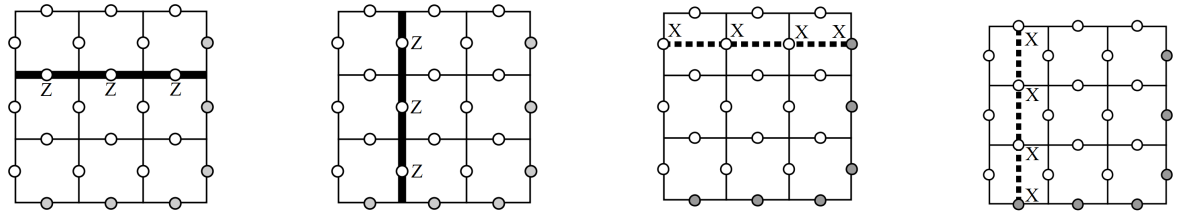


(b)

Figure 2: An example of a Z syndrome of a toric code, shown in Subfigure 2a containing a string of X Pauli errors, shown in Subfigure 2b. Errors are denoted by a black circle.

To complete the code description, we introduce the logical operators [2]. The logical operators must commute with the plaquette and vertex operators. As seen in Figure 3, the logical operators Z_1^L and Z_2^L are given by a closed loop of the tensor product of a Z Pauli operator acting on a horizontal string of qubits and acting on a vertical string of qubits, for Z_1^L and Z_2^L respectively. The logical operators X_1^L and

X_2^L are given by a closed loop of the tensor product of a X Pauli operator acting on a horizontal string of qubits and acting on a vertical string of qubits, for X_1^L and X_2^L respectively.



(a) (b) (c) (d)

Figure 3: Encoded logical operators Z_1^L, Z_2^L, X_1^L and X_2^L, shown in Subfigures 3a, 3b, 3c and 3d, for the two encoded qubits in the toric code.

3. Belief propagation for decoding

Belief propagation is an iterative algorithm which can be used to obtain the conditional probability distribution of an unknown variable, given known variables. Belief propagation is based on Bayes' law, a statistical method used to update the probabilities for a hypothesis as more information becomes available.

Bayes' law equation reads

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E)}, \tag{5}$$

where $P(H|E)$ is the probability of obtaining the hypothesis H after the evidence E is given and $P(E|H)$ is the probability of obtaining the evidence after the hypothesis is given.

Thus, we will introduce the *variable nodes*, these are unknown variables and form an hypothesis H. And the *check nodes*, these are known variables and form the evidence E. For example, in the toric code the variable nodes are the errors on the qubits and the check nodes are the syndromes. The algorithm works by repeatedly passing *messages* between the variable and check nodes, where each set of messages depends on the previous ones. Messages are real valued functions which contains the 'influence' one node exerts on the others. With influence we mean that they signal how much the conditional probability must change according to nodes connected to the first node. The initial messages are only formed by the prior probability of the values of the variable node. But when we calculate the next set of messages to be send, they will include a function of the previous messages, with the particularity that this messages must come from a different origin compared to their destination.

Let H be a parity check matrix and associate its rows to variable nodes and its columns to check nodes. The entry H_{ij} contains 1 if variable node i and check node j are connected and H_{ij} contains 0 otherwise. Its visual representation, the Tanner graph, can be useful to quickly understand how the nodes are connected.

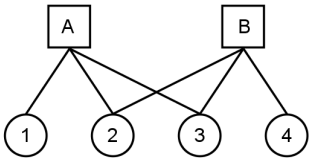


Figure 4: The Tanner graph representation of Eq. (6). The variable nodes are represented by circles and the check nodes by squares.

For example, the Tanner graph of Figure 4 corresponds to the parity check matrix of (6).

$$H = \begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \tag{6}$$

It is important to note that since the values of the check nodes and the variable nodes are related by $c_i = v_j H_{ij}$, where c_i is the value of check node i and v_j is the value of variable node j, one could think that we could obtain the values of v by $c_i H_{ij}^{-1} = v_j$. However, H is not of full rank, so H^{-1} is not well defined.

3.1. The belief propagation algorithm

Before starting the algorithm, the observed values and the *prior belief*, a function of the probability of the values the variable node may take, are needed.
The algorithm structure is described by the following [4] :

(i) We start the algorithm by initializing all the messages values to 0.

(ii) The messages from the check nodes to the variable nodes are given by

$$\mu_{c\to v} = \sum_{v'\in N(c)/v} f_v \mu_{v'\to c}\,, \tag{7}$$

where $N(c)$ are the elements connected to neighborhood c. Since the message is the joint probability of all nodes, by performing a sum of all the neighboring nodes we will obtain the marginal probability for the node we are calculating. Eq. (7) contains information from the observed check nodes, as the prediction must be compatible with the values observed.

(iii) The messages from the variable nodes to the check nodes are given by

$$\mu_{v\to c} = l_v + \sum_{c'\in N(v)/c} \mu_{c'\to v}\,, \tag{8}$$

where l_v is the prior belief.

(iv) We iterate (ii) and (iii) until the messages values from both Eqs. (7) and (8) converge.

(v) We obtain the marginal probabilities of the variable nodes by using

$$\mu_v = l_v + \sum_{c\in N(v)} \mu_{c\to v}. \tag{9}$$

With the marginal probabilities calculated, we know which variable nodes values are more likely according to belief propagation given the check nodes observed.

Note that in order to compute the messages value in each iteration, the function used will depend on if we are updating the messages from one type of node or the other. Two types of messages are sent, messages from the check nodes to the variable nodes and messages from the variable nodes to the check nodes. It may help to understand how belief propagation works by taking into consideration that when the messages are sent according to belief propagation, they are not send between node and node. Instead, each message corresponds to a connection $c \to v$ or $v \to c$. Therefore, each node will send multiple messages to the other nodes. Also, note that the bigger the absolute value of the message, the more important the influence of the connection is.
An example of how the messages are connected in Eqs. (7) and (8) for the code in Figure 4 is provided by Figure 5. Note that in Figure 5, the messages sent exclude direct connections between the messages from the same check and variable node.

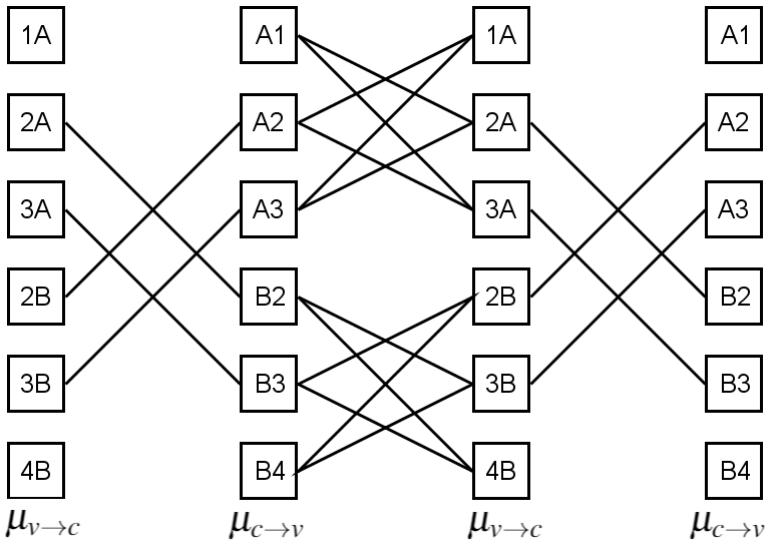


Figure 5: An example of the network that corresponds to the connections between the check nodes and variable nodes for the code in Figure 4. The first column are the initial messages, initialized to a value equal to 0. Each iteration comprises a $\mu_{c\to v}$ column and a $\mu_{v\to c}$ column, therefore, this example shows the connections for the initial messages, the first iteration and half the second iteration.

3.2. Neural belief propagation

We introduce *Neural belief propagation*, a modification on the belief propagation algorithm in order to improve its performance [1].

A *neural network* is a network of connected nodes, bearing resemblance to the connections in the neurons in the brain. The weight of these connections can be modified by a computer itself in order to improve its performance to obtain the desired output from an input. For example, neural networks are used to recognize images and categorize images. In this case, the input would be the image and the output would be what the image is. A neural network structure is actually quite similar to Figure 5. Thus, it is logical to assume that a step to improve belief propagation performance would be to incorporate elements from neural networks into the belief propagation algorithm. Thus, neural belief propagation adds a set of *weights* $w_{c'v,vc'}$ and *biases* b_v to the messages passed in the belief propagation algorithm, this weights and biases will be trained. Training is performed by modifying the weights and biases values according to the predictions obtained compared to a training set of data, for which we have both the input and output that the neural network should obtain.

A *loss function*,expressed by the symbol $\mathscr{L}$, will be used to compare the predictions with the real values, and how much will the biases or weights need to change will be given by

$$\nabla {b_v}^t = -l_r \times \frac{\partial \mathscr{L}}{\partial {b_v}^t}, \tag{10}$$

where l_r is the *learning rate*, an adjustable parameter. Eq. (10) is for the changes for the biases, for the equation for the weights we need to substitute the biases for the weights.

4. Belief propagation applied to the Toric code

4.1. Decoding the toric code

In the toric code, a qubit having an error or not will be the variable nodes and the syndromes will be the check nodes. Thus, the belief propagation algorithm explained previously will be changed to the following equations [4]:

The messages from the syndromes to the errors, as seen in Eq. (7), are modified by

$$\mu_{c\to v}^{t+1} = (-1)^{s_c} 2 \operatorname{arctanh}\left(\prod_{v' \in N(c)/v} \tanh\left(\frac{\mu_{v'\to c}^{t}}{2} \right) \right), \tag{11}$$

where s_c is the measured syndrome of the code we have obtained for that specific element.
It is important to note that when we perform the product of the messages, we are only taking into consideration those messages that come from a different qubit.
The messages from the errors to the syndromes, as seen in Eq. (8), are given by

$$\mu_{v\to c}^{t+1} = l_v + \sum_{c' \in N(v)/c} \mu_{c'\to v}^{t}, \tag{12}$$

where l_v is the prior belief, defined as $l_v = \log(\frac{p(e_v=0)}{p(e_v=1)})$. We refer to the state of having errors on qubit v, e_v, as $e_v = 1$ if there is an error at the qubit at position v, or if there is not, $e_v = 0$.
Note that Eq. (11) will assign a sign to the messages, it is important to remark that the values of the real syndrome, S_c, must be changed to 1 for real syndrome values of -1 and 0 for real values of the syndrome of 1. As the algorithm proceeds, the messages value for the variable nodes will shift to more positive or more negative values as more iterations are calculated.
After enough iterations for the messages values to converge, we will use

$$\mu_v = l_v + \sum_{c \in N(v)} \mu_{c\to v}^{T} \tag{13}$$

to obtain the error belief, which will indicate if an error or not is more likely according to belief propagation on every qubit given the syndromes. Unlike the previous case, for this equation we will consider the messages from all the syndromes connected to the qubit. A higher positive/negative value in the error belief indicates a lower/higher probability of having an error on the considered qubit, respectively.
If we compare Eqs. (11), (12) and (13) to the general equations, (7), (8) and (9), we see that the latter equations are mostly untouched, at most, we can appreciate that the neighborhoods used are the toric code ones. The former equation, however, changes the most of the three equations by introducing a $(-1)^{S_c}$ factor, used to update the predictions with the observed syndromes and by specifying the form of the function used in the general case.

4.2. Decoding using neural belief propagation

When applying the neural network, the equations used are mostly untouched, the only difference being the introduction of the weights and biases into Eqs. (12) and (13).
Thus, the messages from the syndromes to the errors are given by

$$\mu_{c\to v}^{t+1} = (-1)^{s_c} 2 \operatorname{arctanh}\left(\prod_{v' \in N(c)/v} \tanh\left(\frac{\mu_{v'\to c}^{t}}{2} \right) \right). \tag{14}$$

and the messages from the errors to the syndromes are given by

$$\mu_{v\to c}^{t+1} = l_v b_v^t + \sum_{c' \in N(v)/c} \mu_{c'\to v}^{t} w_{c'v,vc'}^{t}, \tag{15}$$

In neural belief propagation, we will calculate the error belief:

$$\mu_v = l_v b_v^t + \sum_{c \in N(v)} \mu_{c\to v}^{t} w_{cv,vc}^{t} \tag{16}$$

for each iteration, as the predictions are needed to properly train the weights and biases. It's easy to see that if the weights and biases are set to 1, we obtain the equations used in the initial belief propagation, (12) and (13).
The loss function will take the form

$$\mathscr{L} = \sum_i \left| \sin\left(\frac{\pi \sum_j H_{ij} \left[e_j + \sigma(\mu_j) \right]}{2} \right) \right|. \tag{17}$$

And with the partial derivative of $\mathscr{L}$ for the weights or biases, we will know how much each weight or bias must be corrected as seen in Eq. (10). We must remark that the values of the real error, e_j, must be changed to 1 for real error values of -1 and 0 for real values of the error of 1, as this are the values the sigmoid will give for errors and non-errors.

5. Simulation of the Toric code

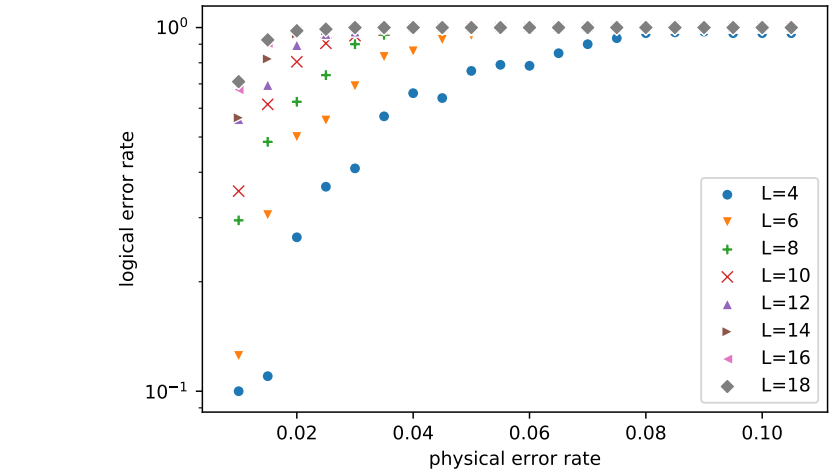
We simulate the toric code by keeping track of the syndromes. First, in order to simulate a toric code of size L, we will generate a matrix of size $L \times L$. This matrix will serve as a template for a toric code and we will make a copy to have the toric code components for the X and Z Pauli errors, as each type of error will not interfere with the other.
The next step is to randomly flip errors to the toric code, done by randomly assigning multiplying an element of the matrices by -1 with probability p_{err}. Simultaneously, we keep track of the syndromes, S_X for Z Pauli errors and S_Z for X Pauli errors. In the simulation, we know the real error for both X and Z lattices, the probability of error p_{err} and the syndromes. But in a real case, we will only know the latter two.
To simulate the decoding, we use belief propagation, with only inputting the syndromes and p_{err}. Thus, the error predictions are obtained, which are then applied to the real code. After the belief propagation prediction and correction on the real code, two types of logical errors may appear on the corrected code:

(i) *Flagged logical errors*, a type of logical error where the states we measure would not be a codeword.
(ii) *Unflagged logical errors*, a type of logical error where the states we measure would be a codeword, but not the original codeword.

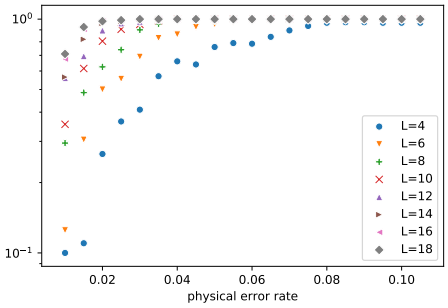
In order to detect if a logical error happened, first we measure the syndromes of the corrected code. Should we measure an eigenvalue of -1, it will indicate that a flagged logical error happened. If all the eigenvalues of the syndrome are 1, then either belief propagation properly predicted the real error pattern or an unflagged logical error happened. Unflagged logical error can be detected by checking if these two conditions are fulfilled: All the syndrome elements of the lattice are 1 and we detect an error on the qubit matrices.
The simulation using neural belief propagation is quite similar, the only difference being that we need to previously train the weights and biases neural belief propagation uses with a training set consisting of known errors on the toric code with its corresponding syndromes. Once trained, we can use neural belief propagation to predict the error pattern from the prior belief and the syndromes measured as before.

5.1. Results

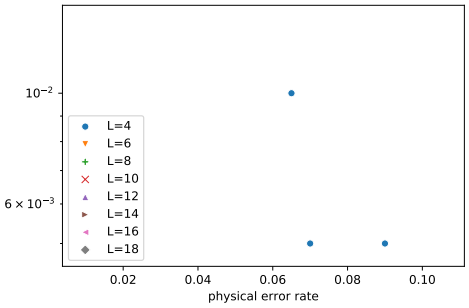
Subfigure 6a is a plot of the evolution of the logical error rate as a function of the physical error rate obtained by applying belief propagation to toric code lattices of $L = 4, 6, 8, 10, 12, 14, 16$ and 18 with randomly generated errors. As we can see, the smaller sized lattices have a lower logical error rate for small physical error rates. However, as the physical error rate increases, the belief propagation predictions fail more frequently, up to the point where most predictions have a logical error by the bigger physical error rates for all lattice sizes. We can separate the two types of logical errors into Subfigure 6b and Subfigure 6c. As we can see, most of the logical errors obtained are flagged logical errors, with unflagged logical errors being almost non-existant, with only a few cases detected in 100 iterations.



(a)



(b)



(c)

Figure 6: Subfigure 6a is a plot of the flagged logical error rate obtained for the physical error rate. The logical error rate is separated into the flagged logical error rate for Subfigure 6b and into the unflagged logical error rate for Subfigure 6c. The results obtained are the average values of a 100 measures applying belief propagation for 12 cycles to different sized toric code lattices. Note that Subfigures 6a and 6b use a range of values between 10^{-1} and 10^{0} to plot the logical error rate and flagged logical error rate, respectively. Instead, Subfigure 6c uses a range of values between 5×10^{-3} and 10^{-2} to plot the unflagged logical error rate.

Another interesting result is on which conditions belief propagation manages to properly predict the real error pattern. Belief propagation properly predicts the error if only there is one error on the whole toric code, or if there are more than one error, but they are separated enough. The minimum distance so that belief propagation properly predicts the real error is of at least one stabilizer operator between them, for which the qubits connected to it are errorless.

6. Conclusions

Belief propagation provides a solution to predicting errors in the toric code, which does not always completely work. However, it can be further optimized using neural belief propagation, with an increase in its performance as seen in Liu and Poulin's paper [1].

While we represented the cells of the toric code as squares, we can also represent the cell geometry with other types of geometries, like an hexagonal lattice or the kagome lattice. This lattices have the property

that their dual is not themselves, like the square lattice that we have used. Instead, they are triangular lattices and rhombic star lattices, for the hexagonal and kagome lattices respectively [2]. Therefore, it could be interesting to apply Belief propagation and neural belief propagation on this other lattices, or even more complex lattices like the Kitaev honeycomb model or even other types of code, like the compass codes [5] [6].

Acknowledgments
I would like to thank my advisor for his help and encouragement through this work, my friends and family for their support and finally the reader, for his or her attention.

References

[1] Y.-H. Liu and D. Poulin, *"Neural belief-propagation decoders for quantum error-correcting codes,"* Phys. Rev. Lett. **122**, 200501 (2019).
[2] D. Browne,*"Lectures on Topological Codes and Quantum Computation,"* retrieved from `bit.do/topological`, (2014).
[3] Nielsen, M. and Chuang, I. (2010). *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge: Cambridge University Press. doi:10.1017/CBO9780511976667
[4] E. Nachmani, Y. Be'ery and D. Burshtein, *"Learning to decode linear codes using deep learning,"* in Proc. IEEE Annu. Allerton Conf. Commun., Control, and Computing (Allerton), pp. 341–346, (2016).
[5] Y. C. Lee, C. G. Brell, and S. T. Flammia, *"Topological quantum error correction in the Kitaev honeycomb model,"* J Stat Mech-Theory E, 083106 (2017).
[6] M. Li, D. Miller, M. Newman, Y. Wu, and K. R. Brown, *"2-D compass codes,"* Phys. Rev. X **9**, 021041 (2019).