Neural Belief Propagation for Quantum Error Correcting Circuits

Sajag Kumar, Josias Old

May 2024

1 Neural Belief Propagation (NBP)

Belief propagation (BP) may fail for two reasons: cycles in the Tanner graph of codes and degenerate errors. For classical codes, while BP may still fail due to the presence of cycles in the Tanner graphs, it does not have to deal with degenerate errors. BP has a zoo of modifications and post-processing methods that ensure its convergence on loopy Tanner graphs. These algorithms were designed for classical codes and did not care about degenerate errors. Some of these work very well, even for quantum codes. Their success (probably) lies in the fact that they work by reweighing edges in the Tanner graph of the code to avoid oscillating messages along cycles; for quantum codes, this may lift some error degeneracies and improve performance. However, error degeneracy is not a problem but a feature that a decoder should exploit. For quantum codes, the decoder does not need to look for the actual error but for an error that is degenerate with the actual error. No post-processing method or modification of BP, which is developed for classical codes, does this. Developing such an algorithm is difficult as BP looks for the most likely qubitwise error. So, we resort to neural networks for help.

1.1 The algorithm

The BP algorithm can be mapped to a neural network. We further add learnable parameters that can be trained to exploit error degeneracy and improve BP's decoding performance. This neural network with these additional parameters is what we will call NBP network. In the following we discuss how BP update equations are altered to give the NBP algorithm.

1. Qubit to check messages from the BP algorithm are modified to include trainable weights B_q and $W_{c'q}$.

$$m_{q\to c} = l_v + \sum_{c' \in N(q)\setminus c} m_{c'\to q} \longrightarrow m_{q\to c} = l_q B_q + \sum_{c' \in N(q)\setminus c} m_{c'\to q} W_{c'q} \tag{1}$$

2. The check to qubit messages update equation remains the same. This step is the equivalent of the action of an activation function in a feed-forward neural network.

$$m_{c\to q} = (-1)^{s_c}\ 2\tanh^{-1}\left(\prod_{q'\,\in\, N(c)\,\setminus\, q}\tanh\left(\frac{m_{q'\to c}}{2}\right)\right) \tag{2}$$

   Here, s_c is the syndrome corresponding to check c.

3. The equation for computing beliefs is also modified in the same way as the qubit to check messages. The trainable weights $B_q^{\prime(t)}$ and $W_{c'q}^{\prime(t)}$ are different from the ones used in qubit to check messages.

$$b_q = l_q + \sum_{c\,\in\, N(q)} m_{c\to q} \longrightarrow b_q = l_q B_q + \sum_{c\,\in\, N(q)} m_{c\to q} W_{cq} \tag{3}$$

Later we will use detector error models instead of actual quantum circuits. Qubit and check nodes are then replaced by error and detector nodes, and the NBP equations for layer t are relabelled as follows:

$$m_{e\to d}^{(t+1)} = l_e B_e^{(t)} + \sum_{d'\,\in\, N(e)\,\setminus\, d} m_{d'\to e}^{(t)} W_{d'e}^{(t)} \tag{4}$$

$$m_{d\to e}^{(t+1)} = (-1)^{s_d}\ 2\tanh^{-1}\left(\prod_{e'\,\in\, N(d)\,\setminus\, e}\tanh\left(\frac{m_{e'\to d}^{(t)}}{2}\right)\right) \tag{5}$$

$$b_e^{(t+1)} = l_e B_e^{\prime(t)} + \sum_{d\,\in\, N(e)} m_{d\to e}^{(t+1)} W_{de}^{\prime(t)} \tag{6}$$

1.1.1 Loss function

The inferred recovery from the NBP algorithm is obtained by $\sigma(b_e)$, where

$$\sigma(x) = \frac{1}{1+e^x}. \tag{7}$$

For correct decoding we need that $\sum_{jk} H_{ij}(e_k + \sigma(b_k)) = 0$. Where H is the parity check matrix for the code and e is the actual error. The loss after layer n is defined as:

$$L_n(b,e) = \sum_i f\left(\sum_{jk} H_{ij}(e_k + \sigma(b_k))\right) \tag{8}$$

where

$$f(x) = \left|sin\left(\frac{\pi x}{2}\right)\right| \tag{9}$$

and the total loss is obtained by a weighted sum of losses after each layer,

$$L = \sum_n \rho_n L_n(b, e). \tag{10}$$

The weights ρ_n are also trainable. They are added to minimise the effect of number of layers on the decoding performance. In fig. 1 we compare the performance of NBP with different number of layers with trainable ρ_n's, as well as with all $\rho_n = 1/N$. In case ρ_n's are trainable, the decoding performance for NBP with 10 and 20 layers is almost the same. Whereas, when ρ_n's are constant the performance improves significantly when the number of layers are increased.

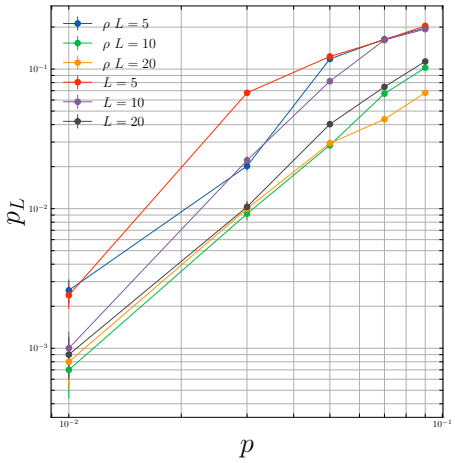

Figure 1: The effect of number of layers and weighted loss in the decoding performance of NBP decoder for $[[17, 1, 3]]$ surface under code capacity noise model.

In fig. 2 loss curves for two other loss functions are shown. Binary cross entropy measures how close the inferred recovery $\sigma(b_e)$ is from the actual error e, the loss function is $L = -\sum_v e_v \log(\sigma(b_v)) + (1 - e_v) \log(1 - \sigma(b_v))$. We can clearly see that the NBP algorithms fails to learn with this loss function. This is also expected because binary cross entropy encourages the network to do classical decoding and does not account for error degeneracy. The loss function we discussed above is He. The third loss function $[HL]e$ is obtained by replacing H in eq. 8 by $[HL]$ which is a matrix with logical operators concatenated below the parity check matrix. The motivation for using this loss function is to ensure that the predicted recovery does not flip any logical observables.

1.1.2 Residual Weights

To improve the training of the network we also include residual weights (r_i), defined as

$$m_{c\to q}^{(i+1)} = NBP(m_{c\to q}^{(i)}) + r_i m_{c\to q}^{(i)}. \tag{11}$$

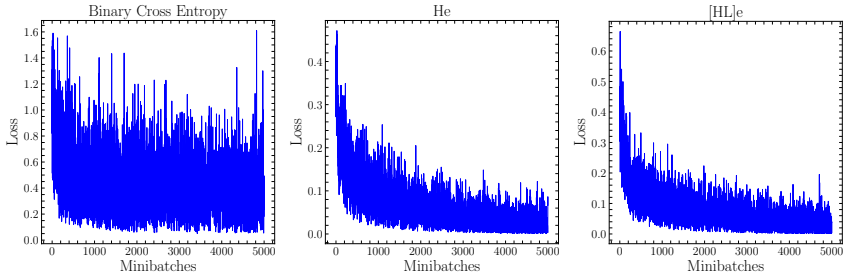

Figure 2: Loss curves for the training of NBP network for $[[17, 1, 3]]$ surface code under code capacity noise model with different loss functions.

As shown in fig. 3 the training is better with residual connections in the network than without them. The residual weights acts as message dampeners/amplifiers depending on their magnitudes.

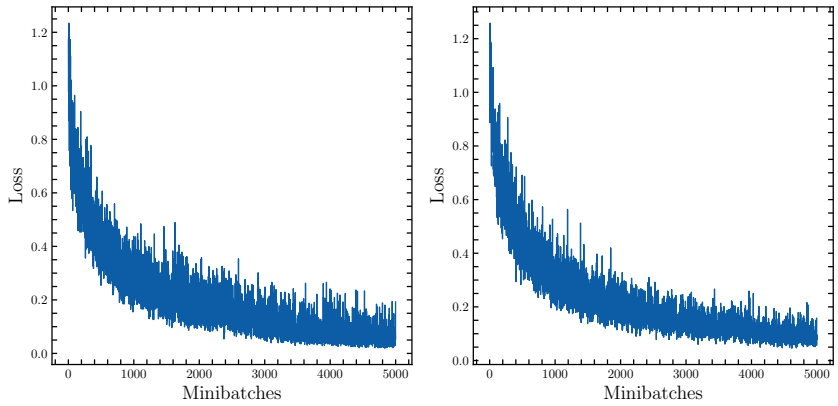

Figure 3: Loss curves for training of NBP network (20 layers) for decoding $[[17, 1, 3]]$ surface code under code capacity noise model (left) without residual connections and (right) with residual connections.

1.2 Results

In this section we present results for $[[17, 1, 3]]$ surface code. The NBP network used for decoding is 20 layers deep for all noise models. The NBP networks are trained on 10000 minibatches of 120 errors each and 10000 Monte Carlo (MC) samples are taken for decoding. The error bars represent standard MC errors.

1. *Code capacity noise model.* The performance of BP is worse for larger distances. While NBP performs better than BP at all distances, the performance of NBP does not improve as d is increased.

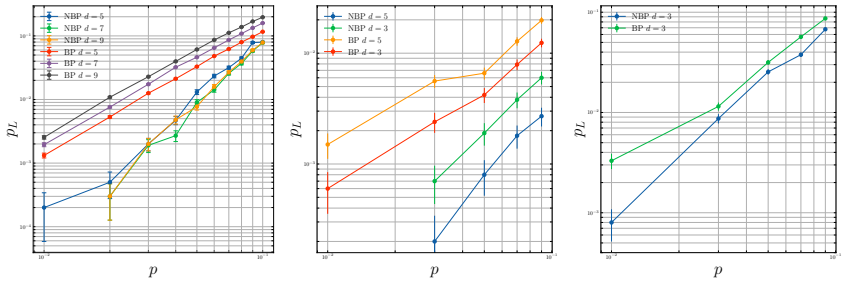

Figure 4: Comparison of decoding performance of NBP and BP for $[[17, 1, 3]]$ surface code under (left) code capacity, (center) phenomenological, and (right) circuit level noise models.

2. *Phenomenological noise model.* We only train for $d = 3$ and 5, training NBP for larger d is slow. The performance is ideal. For $p = 0.001$ there are 0 failures for 10000 samples.

3. *Circuit level noise model.* We only train for $d = 3$, training for larger d runs into memory overflow problems. While the decoding performance of NBP is still better than BP, we have not checked if increasing d will result in better performance for NBP.

1.3 Next steps

1.3.1 Interpreting learned weights

As can be seen in fig. 5 only some edges in the detector error model of the code are weighted. It would be nice to understand what these weighted edges correspond to in the actual quantum error correcting circuit. Since the weight of only a fraction of edges are being learned we can reduce the time and space complexity of training and decoding significantly, if we only train the edges that will be weighted.

1.3.2 Improving the accuracy

1. *OSD postprocessing.* We can do OSD on the output of the NBP decoder. While this seems to guarantee a better or at worst the same performance at NBP, the attempts hitherto have proven otherwise, the decoding performance actually worsens with OSD.

2. *Adding more trainable weights* For example we can

   - Change $W_{d'e}$ to $W_{d'e,ed}$, i.e. the weights depends not only on the detector to error messages being summed but also on the error to detector message they are being summed for.

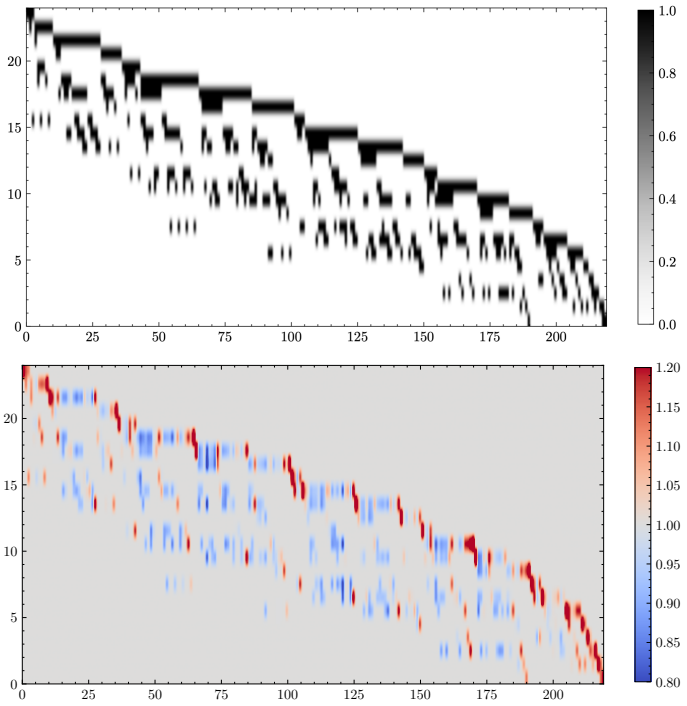

Figure 5: (Top) Check matrix for the $[[17, 1, 3]]$ surface code under circuit level noise. (Bottom) Learned weights (W_{cq}) for the fifteenth layer.

- Add weights in the detector to error messages. This is equivalent to having a different activation function for each node in a layer.

This will give the network more flexibility to learn but will also increase memory requirements.