Project Report

An Experimentation Of:
Cloning a node in XOR model

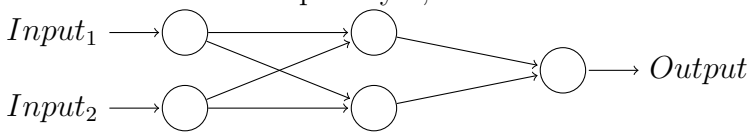
Thinh Truong
Supervised by Dr. Yuly Billig

11 August 2021

1 Introduction

The XOR problem is a classic problem in machine learning. However, a model with a hidden layer of two nodes may not product correct predictions and it depends a lot on the initial weights and biases. Therefore, in order to tackle this problem, one must increase the number of nodes in the hidden layer. In this report, I will explain the idea behind cloning a node, and point out the problems that I encountered with it.

In the XOR problem, given a model with two nodes in input layer, two nodes in hidden layer and one node in output layer, we can think of the following model.



Throughout the paper, let:

- The input layer be layer 1, the hidden layer be layer 2, and the output layer be layer 3.
- The learning rate ϵ be 1 to speed up training.
- w^l_{jk} be the weight from the k^{th} node in the $(l-1)^{th}$ layer to the j^{th} node in the l^{th} layer.
- b^l_j be the bias of the j^{th} node in the l^{th} layer.
- z^l_j be the weighted input of the j^{th} node in the l^{th} layer and $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$
- a^l_j be the activation of the j^{th} node in the l^{th} layer and $a^l_j = \sigma(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) = \sigma(z^l_j)$.

Using the above notations, we therefore get the following matrices for our XOR model.

Weight matrix 1
$$\begin{pmatrix} w_{11}^2 & w_{12}^2 \\ w_{21}^2 & w_{22}^2 \end{pmatrix}$$
Bias matrix 1
$$\begin{pmatrix} b_1^2 \\ b_2^2 \end{pmatrix}$$
Weight matrix 2
$$\begin{pmatrix} w_{11}^3 & w_{12}^3 \end{pmatrix}$$
Bias matrix 2
$$\begin{pmatrix} b_1^3 \end{pmatrix}$$

2 Cloning a node

2.1 The case where our model would not produce correct predictions

One case where training our two-node hidden layer model would not work is when the weights and biases are initialized as below:

Weight matrix 1
$$\begin{pmatrix} 4 & -6 \\ -4 & -5 \end{pmatrix}$$
Bias matrix 1
$$\begin{pmatrix} -3 \\ 0.5 \end{pmatrix}$$
Weight matrix 2
$$\begin{pmatrix} 4 & -4 \end{pmatrix}$$
Bias matrix 2
$$\begin{pmatrix} 0 \end{pmatrix}$$

The following image is the prediction of our initial model for all four cases.

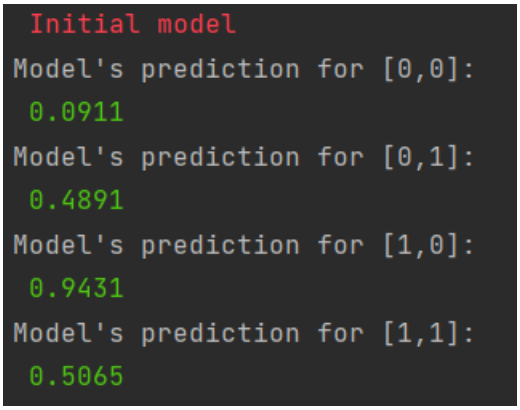


Figure 1: The predictions of the initial model.

This model starts with good predictions for [0,0] and [1,0] cases and bad predictions for the other two cases (see Figure 1). However, as we train the model, we will never get the predictions we expected . I choose 50000 epochs because it is a large enough number to show that the cost function will never converge to zero.

```
After training
Model's prediction for [0,0]:
 0.0065
Model's prediction for [0,1]:
 0.4999
Model's prediction for [1,0]:
 0.9935
Model's prediction for [1,1]:
 0.5001
```

Figure 2: The predictions of the model after 50000 epochs.

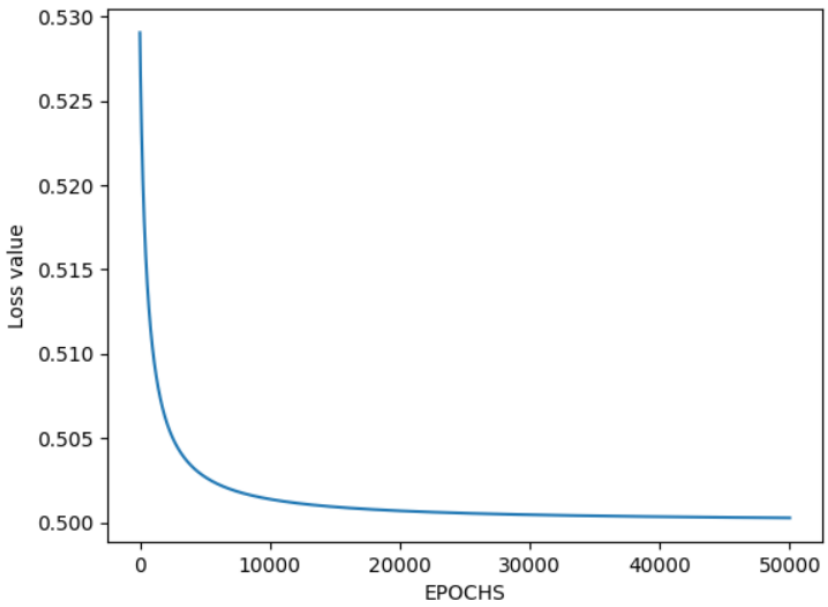


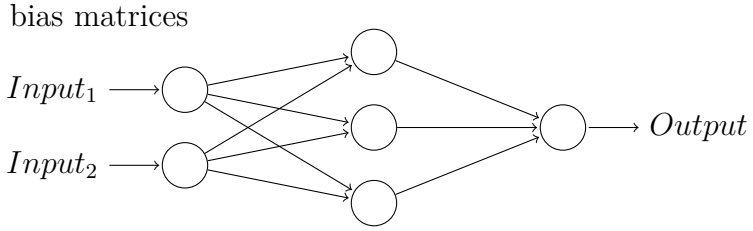
Figure 3: The graph of cost function after 50000 epochs.

2.2 The theory of cloning a node

In order to overcome local minimum in the above model, we can add a node in the hidden layer so that there is a new dimension in our cost function. Cloning a node means we copy a

node from the hidden layer, and create an exact same node with the same input weights and bias, but decrease the output weights of both nodes by half. The reason we have to halve the output weights is because we would want to maintain all the features from the previous model, and so, to keep a_1^3 the same after adding a new node with the same activation as the original node, the output weights must be reduced by half.

In this case, let us clone the second node in the hidden layer from the initial model, where we skip training the model for 50000 epochs, and place the new node in the third position. This results in a new model with three nodes in the hidden layer and the following weight and bias matrices



Weight matrix 1

$$\begin{pmatrix} 4 & -6 \\ -4 & -5 \\ -4 & -5 \end{pmatrix}$$

Bias matrix 1

$$\begin{pmatrix} -3 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Weight matrix 2

$$\begin{pmatrix} 4 & -2 & -2 \end{pmatrix}$$

Bias matrix 2

$$\begin{pmatrix} 0 \end{pmatrix}$$

Therefore, as expected, we would get the same predictions as the as before.

```
After make clone
Model's prediction for [0,0]:
 0.0911
Model's prediction for [0,1]:
 0.4891
Model's prediction for [1,0]:
 0.9431
Model's prediction for [1,1]:
 0.5065
```

Figure 4: The prediction of the model after cloning a node.

However, if we keep training our model with the normal backpropagation method, the two copies of the cloned node will always be updated in the same way and that would make our new node pointless. Therefore, initially, we would have to distinguish the two copies. In

this report, let consider $\frac{\delta C}{\delta a_j^l}$ as the auxilary partial derivative of the j^{th} node in the l^{th} layer . Thus, I alternate updating the partial derivatives of the cost function with respect to weights and biases of each node based on their auxilary partial derivative.

Algorithm: ALTERNATING NODE UPDATE METHOD

Input: Let e be the position of the original node, f be the position of the new node in the hidden layer l.

Output: Update the original node when its auxilary partial derivative is positive and update the new node when its auxilary partial derivative is negative.

if $\frac{\delta C}{\delta a_e^l} < 0$ **then**
- $\frac{\delta C}{\delta b_e^l} = 0$
- **for** *k be each node number in the input layer* **do**
  - $\frac{\delta C}{\delta w_{ek}^l} = 0$
- **end**
- **for** *j be each node number in the output layer* **do**
  - $\frac{\delta C}{\delta w_{je}^{l+1}} = 0$
- **end**

if $\frac{\delta C}{\delta a_f^l} > 0$ **then**
- $\frac{\delta C}{\delta b_f^l} = 0$
- **for** *k be the node number in the input layer* **do**
  - $\frac{\delta C}{\delta w_{fk}^l} = 0$
- **end**
- **for** *j be the node number in the output layer* **do**
  - $\frac{\delta C}{\delta w_{jf}^{l+1}} = 0$
- **end**

After training the model with this method a certain number of epochs, the two copies of the cloned nodes will be two distinguishable nodes and we then can proceed to train the model normally. With the new node in the hidden layer, we essentially add a new dimension for the cost function to overcome the local minimum, and expect the new model will perform well with the predictions.

3 Current obstacles with cloning a node

With the alternating node update method, initially, the cost value goes down, as expected. We would expect the graph of the cost function to continue decreasing while the two nodes are differentiated. However, after around 300 epochs, the curve starts to increase, which reaches the peak at around 2700 epochs and then plunges and stays stabilized after that.

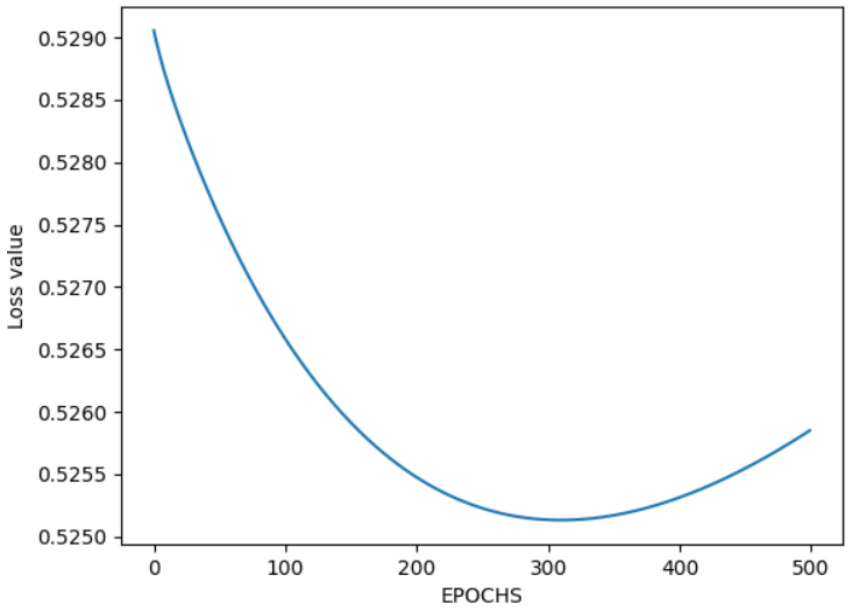


Figure 5: The graph of cost function after 500 epochs trained with alternating node update method.

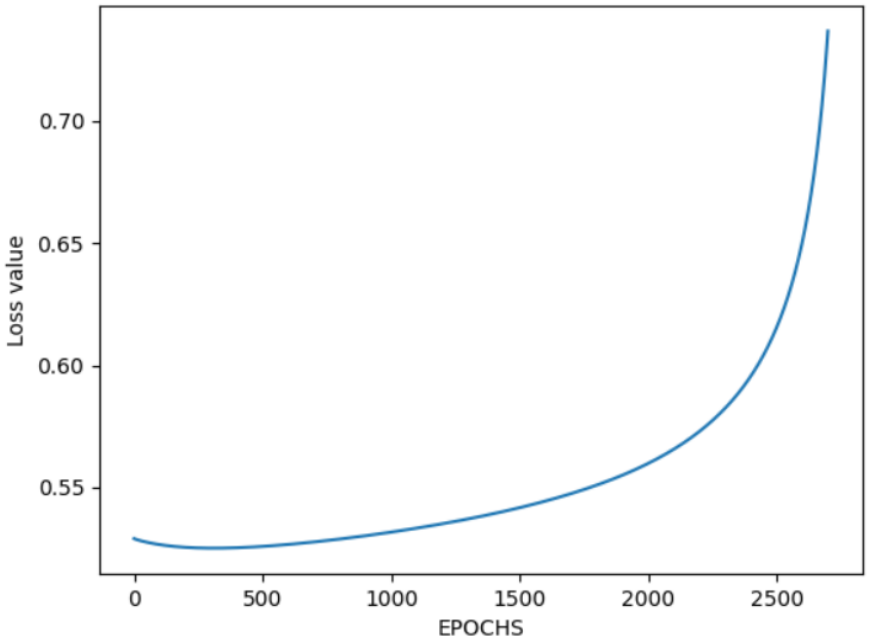


Figure 6: The graph of cost function after 2700 epochs trained with alternating node update method.

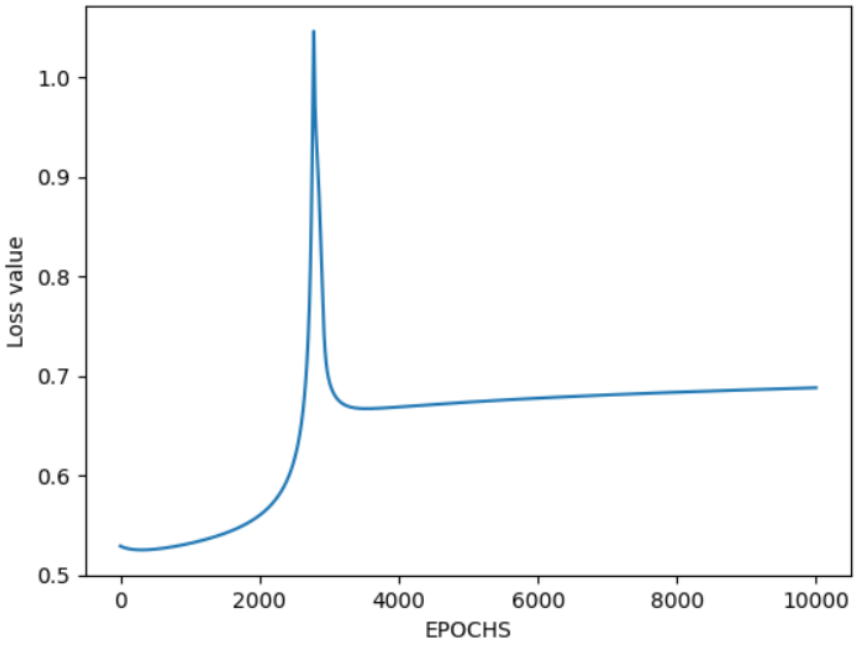


Figure 7: The graph of cost function after 10000 epochs trained with alternating node update method.

The resulting graphs produce a surprising behavior of the cost function. As of now, I cannot mathematically reason why it does behave this way. However, there are a few things that I have observed during the training.

Let consider the $\frac{\delta C}{\delta w_{jk}^l}$ matrices and $\frac{\delta C}{\delta b_j^l}$ matrices of the initial model, model after 200 epochs and after 400 epochs.

`Initial model`

Alternating node update method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} 0.001922699 & 0.006607202 \\ 0.00034808 & 0.005548522 \\ -0.000102995 & -0.000102995 \end{pmatrix} \qquad \begin{pmatrix} 0.00452357 \\ 0.005896602 \\ -0.007195426 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002092904 & -0.002984376 & 0.009444445 \end{pmatrix} \qquad \begin{pmatrix} 0.006864792 \end{pmatrix}$$

Normal trainning method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} 0.001922699 & 0.006607202 \\ 0.000245085 & 0.005445527 \\ 0.000245085 & 0.005445527 \end{pmatrix} \qquad \begin{pmatrix} 0.00452357 \\ -0.001298824 \\ -0.001298824 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002092904 & 0.006460069 & 0.006460069 \end{pmatrix} \qquad \begin{pmatrix} 0.006864792 \end{pmatrix}$$

`After 200 epochs`

Alternating node update method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} -0.000589661 & 0.004670982 \\ 0.000264909 & 0.003371733 \\ -0.000168785 & -0.000168785 \end{pmatrix} \qquad \begin{pmatrix} 0.000988999 \\ 0.003636642 \\ -0.005198084 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002882543 & -0.001950929 & 0.006852048 \end{pmatrix} \qquad \begin{pmatrix} -0.000386748 \end{pmatrix}$$

Normal trainning method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} -0.000589661 & 0.004670982 \\ 0.000204223 & 0.003311048 \\ 0.000401603 & 0.008857716 \end{pmatrix} \qquad \begin{pmatrix} 0.000988999 \\ -0.000955161 \\ 0.004398805 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002882543 & 0.003645883 & 0.002783048 \end{pmatrix} \qquad \begin{pmatrix} -0.000386748 \end{pmatrix}$$

`After 400 epochs`

Alternating node update method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} -0.000705775 & 0.003942532 \\ 0.000184971 & 0.002402513 \\ -0.000243775 & -0.000243775 \end{pmatrix} \qquad \begin{pmatrix} 0.000396406 \\ 0.002587484 \\ -0.003846889 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002585391 & -0.001450028 & 0.005161626 \end{pmatrix} \qquad \begin{pmatrix} -0.00027088 \end{pmatrix}$$

Normal trainning method

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 1} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 1}$$

$$\begin{pmatrix} -0.000705775 & 0.003942532 \\ 0.000141997 & 0.002359538 \\ 0.000434988 & 0.012453106 \end{pmatrix} \qquad \begin{pmatrix} 0.000396406 \\ -0.000536601 \\ 0.009528755 \end{pmatrix}$$

$$\frac{\delta C}{\delta w_{jk}^l} \text{ matrix 2} \qquad \frac{\delta C}{\delta b_j^l} \text{ matrix 2}$$

$$\begin{pmatrix} -0.002585391 & 0.002216078 & 9.65872 \cdot 10^{-5} \end{pmatrix} \qquad \begin{pmatrix} -0.00027088 \end{pmatrix}$$

As we see, even from the initial model, the gradients of the third node in the $\frac{\delta C}{\delta w_{jk}^l}$ matrix 1 are positive numbers if we train the model normally. However, due to the way how we update the new node (third node) when we train the model with the alternating node update method, the gradients of the third node in the $\frac{\delta C}{\delta w_{jk}^l}$ matrix 1 are negative numbers, which

means we would then update the weights of the third node in the direction that would increase the value of the cost function the most. As the number of epochs increases and we keep updating the weights of the third node in the opposite direction of the normal training method, the gradients of the third node in the $\frac{\delta C}{\delta w_{jk}^{l}}$ matrix 1 in later epochs become large positive numbers. Therefore, it makes some predictions worse, especially the [1,1] case where the output depends a lot on the weights of the third node, and so, the graph of the cost functions would start to increase.

4 Concluding remarks

I have briefly described the theory of cloning a node, and the implementation of it. Although it works as expected initially, the graph of the cost function eventually blows up before we can reach the target predictions with the normal training method. Therefore, there is still a question whether it is efficient to clone a node to keep the features from the previous training that may be modified during the alternating node update method before the target predictions are reached?