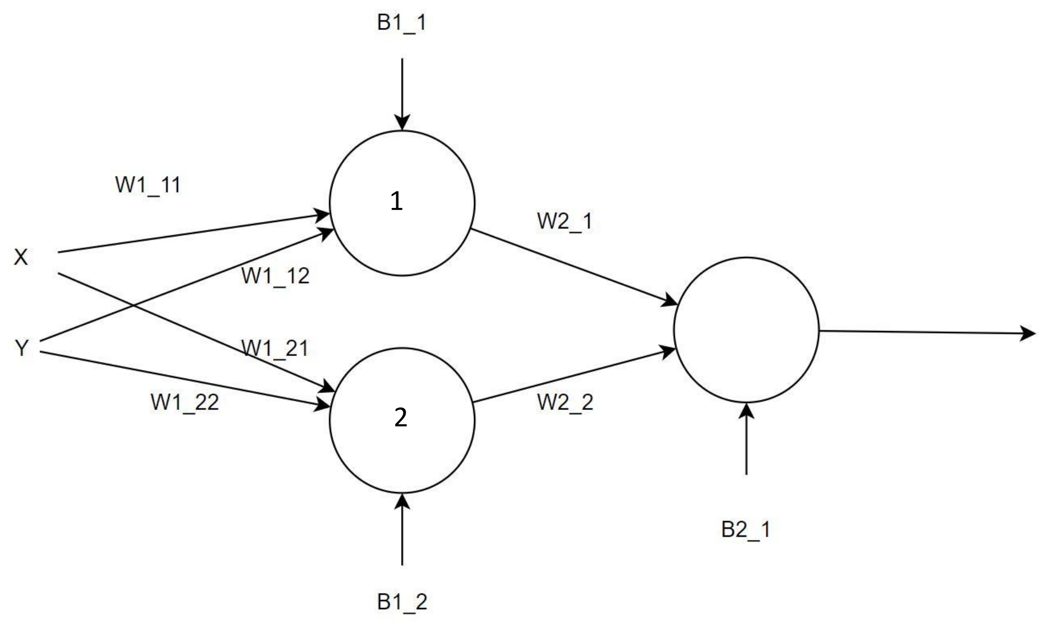
Q1. First of all, we should prepare the input data. If we consider two inputs for our NOR function, there exist 4 combinations of different inputs. By considering bias as a member of weights we should input a 3*1 vector to the neuron.

Q2. Same as part 1. But note that in this part after the train, I get negative numbers for 1 class and positive numbers for 0 class. But the classifier works correctly and needs an additional activation function which multiplies the output by -1. I decreased the learning rate per iteration and tried to prevent gradients from exploding, so then the system converges. Results available in the notebook file.

Q3. Madaline is and of multiple Adaline. Now consider we have a linearly separable data which can be separated by AND of multiple lines, so we can classify this dataset with Madaline. Each Adaline checks the condition for each line and a AND function at the end concludes that all the conditions are held or not. So if we could separate our data with AND of multiple lines, then we can solve that problem with Madaline.

A. Yes, it's possible to make a Madaline to classify these inputs. If we consider the circles as class 1 and x's as class 0, we can classify them with the network described in the picture. Data is separable with and of two lines y = 0.5 and y = -0.5.



$$W1 = \begin{bmatrix} 0 & -1 \\ 0 & 1 \end{bmatrix}, \quad B1 = [0.5 \quad 0.5], \quad W2 = [1 \quad 1], \quad B2 = [\,-1.5\,]$$

For example, let's consider X = 1, Y = 0. Classifier should return 1.
Neuron 1 calculates $0 \times 1 + -1 \times 0 + 0.5 = 0.5 \geq 0$. So it returns 1. Neuron 2

calculates $0 \times 1 + 1 \times 0 + 0.5 = 0.5 \geq 0$. Then neuron 3 calculates $1 \times 1 + 1 \times 1 - 1.5 = 0.5 \geq 0$. So network returns 1 at the end.

B. If we are not able to use OR function in Madaline and have only an AND at the end, **no** we cannot solve this problem with Madaline. But it's able to solve it via MLP by AND of 4 lines and OR of them with one line at the left. It's not solvable with Madaline which discussed in the class.

Q4. Implemented with binary cross entropy loss and 6 layers with [512, 256, 128, 64, 32, 10] neurons and a softmax layer at the end and also changed the outputs to one-hot vectors for simplicity. More details on the notebook.

Q5. I implemented same network as Q4, but adding tanh activation function. Having 6 layers with [512, 256, 128, 64, 32, 10] neurons and a softmax layer with **cross entropy loss**.

Suppose o is the output vector and y is one-hot vector representing the true label. Because we are implementing

Forward propagation

To improve training, I implemented batch normalization algorithm at the beginning. And because it has no parameters before this layer, there is no need to implement backward propagation for this layer. So first, we should standardize the mean and variance of the input data.

$$A_0 = x, \qquad Z_1 = W_1^T A_0 + b_1, \qquad A_1 = \mathrm{Z}_1, \qquad Z_2 = W_2^T A_1 + b_2, \qquad A_2 = Z_2,$$
$$Z_3 = W_3^T A_2 + b_3, \qquad \dots, \qquad o = softmax(A_6)$$

Calculating loss:

$$J = -\sum_{i=0}^{|y|} y_i \log o_i$$

As we have one-hot vector for y, so cost function has only one element (I didn't consider batches in the report, but it should be considered when implementing. So we need to average loss over a batch), so we can write:

$$J = -y \log o$$

Backward propagation

Starting from the loss function.

$$\frac{\partial J}{\partial o} = -\frac{1}{o}$$

For the softmax layer, calculate the derivatives for index k of A_6. Here A is equal to A_6 and $o_i = \frac{e^{A_i}}{\sum_{j=0}^{|A|} e^{A_j}}$

$$\frac{\partial J}{\partial A_k} = \frac{\partial}{\partial A_k}\left(-\sum_{i=0}^{|y|} y_i \log o_i\right) = -\sum_{i=0}^{|y|} \frac{\partial}{\partial A_k} y_i \log o_i = -\sum_{i=0}^{|y|} \frac{y_i}{o_i}\frac{\partial o_i}{\partial A_k}$$

$$\frac{\partial o_i}{\partial A_k} = \begin{cases} i \neq k & -o_i \times o_k \\ i = k & o_i(1-o_i) \end{cases}$$

$$\frac{\partial J}{\partial A_k} = \sum_{i\neq k}^{|y|} \frac{y_i}{o_i} o_i \times o_k - \frac{y_i}{o_k} o_k(1-o_k) = o_k \sum_{i\neq k}^{|y|} y_i - y_k + y_k \times o_k = o_k\left(\sum_{i\neq k}^{|y|} y_i + y_k\right) - y_k$$
$$= o_k - y_k$$

So to vectorize it, we can write:

$$\frac{\partial J}{\partial A_6} = o - y$$

For each activation function g we have $A_l = g(Z_l)$, and we can calculate $\frac{\partial J}{\partial Z_6}$.

$$\frac{\partial J}{\partial Z_6} = \frac{\partial A_6}{\partial Z_6}\frac{\partial J}{\partial A_6} = g'(Z_6)\frac{\partial J}{\partial A_6}$$

Then we can calculate $\frac{\partial J}{\partial W_6}$ and $\frac{\partial J}{\partial b_6}$, because $Z_6 = W_6^T A_5 + b_6$ with avoiding shape convention we have

$$\frac{\partial J}{\partial W_6} = \frac{\partial Z_6}{\partial W_6}\left(\frac{\partial J}{\partial Z_6}\right)^T = \frac{\partial Z_6}{\partial W_6}.\left(\frac{\partial J}{\partial Z_6}\right)^T = A_5.\left(\frac{\partial J}{\partial Z_6}\right)^T$$

$$\frac{\partial J}{\partial b_6} = \frac{\partial J}{\partial Z_6}\frac{\partial Z_6}{\partial b_6} = \frac{\partial J}{\partial Z_6}$$

$$\frac{\partial J}{\partial A_5} = \frac{\partial Z_6}{\partial A_5}\frac{\partial J}{\partial Z_6} = W_6\frac{\partial J}{\partial Z_6}$$

And now we can do the same till we reach first layer. As we see, we should cache values of A to be able to run backprop.

After calculating gradients, should update the parameters based on the gradient descent rule with learning rate η.

$$\theta = \theta - \eta\nabla_{\mathrm{J}}(\theta)$$