Homework 2
Computational Intelligence

Baktash Ansari

99521082

Q1:

For the NAND gate we will have:

X1	X2	Y
0	0	1
0	1	1
1	0	1
1	1	0

We should use just one neuron that input the X = [x1, x2] and W= [w1, w2] and b = 1 and α = 0.1. For the W I randomly choose these values: W = [-3, 4]

The formula is like below:

- Approximation of gradient(E)

$$E(w(n)) = \frac{1}{2}e^2(n) \qquad e(n) = d(n) - \sum_{j=0}^{m} x_j(n)w_j(n)$$

$$\frac{\partial E(w(n))}{\partial w(n)} = e(n)\frac{\partial e(n)}{\partial w(n)} = e(n)[-x(n)^T]$$

- Update rule for the weights becomes:

$$w(n+1) = w(n) + \eta x(n)e(n)$$

We have:
X = [[0, 0, 1, 1], [0, 1, 0, 1]]
W = [-3, 4]
b = 1
D = [1,1,1,0]
F = W1X1 + W2X2 + 1

For input (0, 0) and output 1:
y = -3 * 0 + 4 * 0 + 1 = 1
b = 1 + 0.1(1 - 1) = 1
w1 = -3 + 0.1(1 - 1) = -3
w2 = 4 + 0.1(1 - 1) = 4

For input (0, 1) and output 1:
y = -3 * 0 + 4 * 1 + 1 = 5
b = 1 + 0.1(1 - 5) = 0.6
w1 = -3 + 0.1(1 - 5) = -3.4
w2 = 4 + 0.1(1 - 5) = 3.6

For input (1, 0) and output 1:
y = -3.4 * 1 + 3.6 * 0 + 0.6 = -2.8
b = 0.6 + 0.1(1 + 2.8) = 0.98
w1 = -3.4 + 0.1(1 + 2.8) = -3.02
w2 = 3.6 + 0.1(1 + 2.8) = 3.98

For input (1, 1) and output 0:
y = -3.02 * 1 + 3.98 * 1 + 0.98 = 1.94
b = 0.98 + 0.1(0 - 1.94) = 0.786
w1 = -3.02 + 0.1(0 - 1.94) = -3.214
w2 = 3.98 + 0.1(0 - 1.94) = 3.786

Q2:

a:

The activation function in a neural network determines whether a neuron should be activated by calculating the weighted sum and adding bias. Linear activation functions take the form f(x) = x, meaning the output is proportional to the input. They are used primarily in the output layer and do not change the proportionality of the input data. However, a neural network without an activation function is essentially just a linear regression model. On the other hand, non-linear activation functions can have curves and are used to compute complex functions. They allow the model to create complex mappings between the network's inputs and outputs and can approximate functions that do not follow linearity. Non-linear functions also make back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Thus, the main difference between linear and non-linear activation functions is that linear functions preserve the proportionality of the input data, while non-linear functions can change it and allow for more complex input-output mappings.

B:

In a Multi-Layer Perceptron (MLP) model, the weights and biases play a crucial role in the learning process. Here's how the training progress might be affected in the situations you mentioned:

Random Bias and Zero Weights: If the weights are initialized to zero, the derivative with respect to the loss function is the same for every weight, meaning all weights have the same value in subsequent iterations. This makes the hidden units symmetric and continues for all the n epochs during training. This leads to the model being unable to break symmetry and might as well be learning with just one hidden unit. The bias being random might introduce some initial asymmetry, but it's not enough to sustain meaningful learning, as the weights play a more significant role in the learning process.

Zero Bias and Random Weights: In this case, the model can still learn effectively. The weights being randomly initialized can break symmetry and thus every neuron can learn different features of the input. The bias term is used to shift the activation function towards the right or left, which might be useful in some cases, but initializing it to zero won't prevent the network from learning. The bias will be updated during the training process and will adjust to a value that minimizes the loss.

C:

Among the neural networks I have learned (Perceptron, Adaline, Madaline, and MLP), the Multilayer Perceptron (MLP) generally has a greater generalization capability, while the Perceptron has a lesser capability.

Here's why:

1. Perceptron
   - The Perceptron is a single-layer neural network that can only learn linearly separable functions. This means it can only classify data that can be separated by a straight line or hyperplane.
   - It has a relatively limited capacity to learn complex patterns or relationships in the data.

2. Adaline
   - Adaline, while an improvement over the Perceptron, is still a single-layer model with a linear activation function.
   - It can learn linear relationships and perform regression tasks, but it is still limited in its capacity to model non-linear relationships.

3. Madaline
   - Madaline is an extension of Adaline and can handle multiple classes, but it's still a shallow model composed of multiple linear units.
   - While it can handle more complex decision boundaries compared to the Perceptron or Adaline, it may struggle with highly non-linear problems.

4. MLP (Multilayer Perceptron)
   - MLPs have multiple hidden layers, which allow them to learn highly non-linear relationships in the data.
   - The presence of multiple layers and non-linear activation functions in each node enables MLPs to approximate complex functions to a high degree of accuracy.
   - With appropriate training, MLPs can generalize well to a wide range of tasks.

D:

Advantages:

> This method can provide precise updates to the weights, which can lead to more accurate model training why? Because it can include the speed of changing weights to the training procedure.
>
> It allows you to understand how the network is learning and how the weights are being updated at each stage.

Disadvantages:

Calculating this H value and its inverse can be computationally expensive, especially for large networks. This can make the training process slower.

Storing these values requires O(n^2) memory (where n is the number of parameters), which can be prohibitive for large models.

In cases where the error surface is non-convex, the method can lead to poor convergence or get stuck in local minima.

Q3:

Circle data:

Linear: the result of this mlp with linear activation function is awful; Because it can't change the representation of data in the hidden layer and hyperplanes are not fit with the data.
Sigmoid: the result of sigmoid is good for this data representation because it can't change the representation in a way that hyperplanes can be in a circle shape and categorize the area.
Tanh: the result of Tanh is so similar to sigmoid because these two functions have a lot in common except the output domain of their values.
ReLU: the result of ReLU I think is the best result because it can find the borders perfectly and we can generalize it easily

Exclusive or

Linear: again the result is awful and the reason is the same as before.
Sigmoid: the result of sigmoid in this dataset is not acceptable. It can create not bad borders but massive data is predicted wrongly.
Tanh: it is better than sigmoid because of negative values that sigmoid is always positive but also some proportion of data still predicted wrongly.
ReLU: again the best result is the ReLU. it can completely train perfectly and create accurate borders for this data. I think the reason for this ability is negative values that can remove some hidden outputs and regularize data.

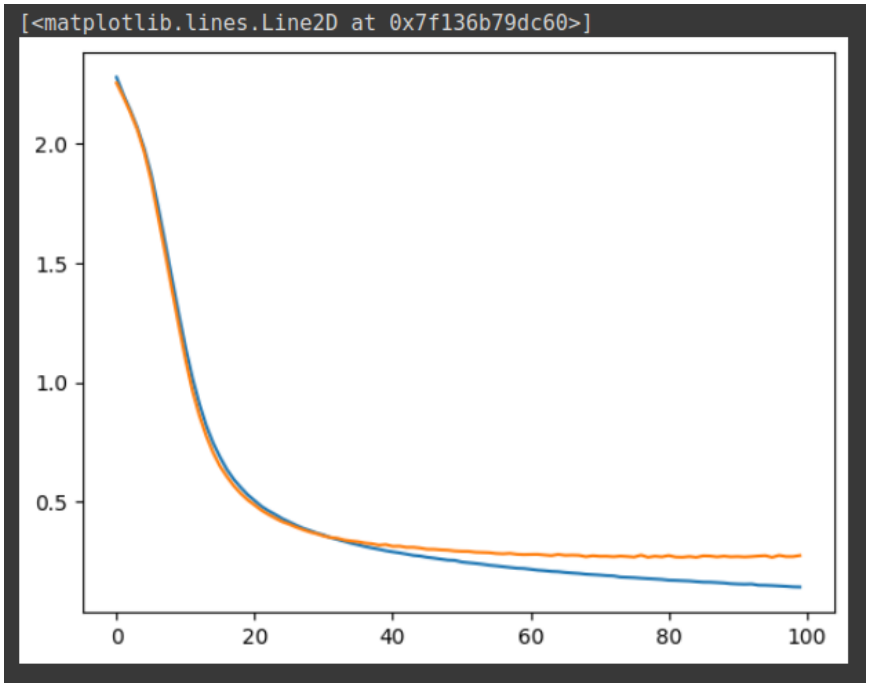
Gaussian

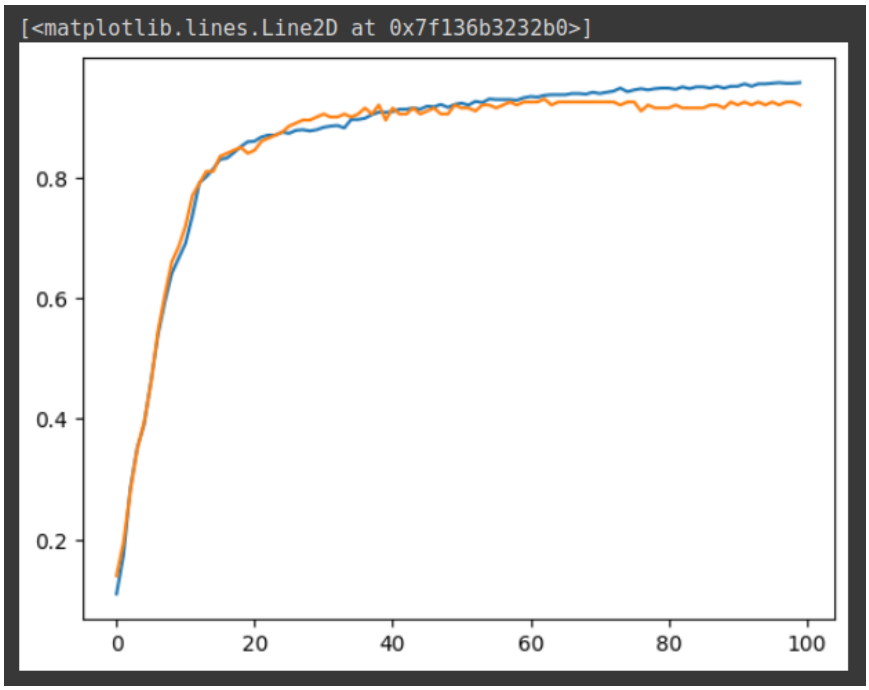
This dataset is so simple. Therefore, all activation functions can do perfectly, even linear ones!

Spiral

Linear: this dataset is the most complex one so linear creates a very bad result.
Sigmoid: sigmoid creates a very bad result too. Because the dataset is so complex.
Tanh: same as sigmoid. Very bad!
ReLU: I think a little better than previous ones, but still very bad!

Q4:





We have an input dimension of 25 so we can set 25 neurons for input and because we have 10 label classification tasks, I set the output layer with softmax because of multi-label classification and one middle layer with 15 neurons with ReLu.

Finally, the model fits very good and as we can see in the plots, error is very low and accuracy is very high.

Q6:

I used the MNIST dataset Tensorflow datasets
I use two layers. The first layer with 128 neurons and ReLU
And second layer with 64 neurons and ReLU
I added a dropout layer between layers one and two to prevent overfitting
For the last layer, I used the 10 neurons layer for the 10-label classification problem and the Softmax activation function for classification.
I use 128 neurons for the input layer experimentally. Because more neuron costs more time to train and increase the probability of overfitting.
Plots: