Contents

1 Problem 1

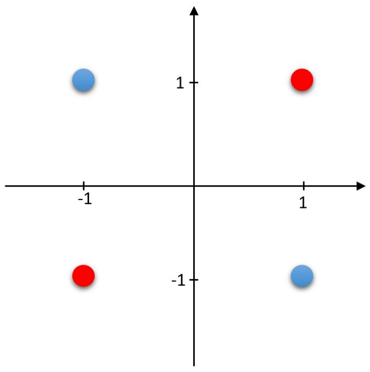
In this problem, we will be focusing on a single Artificial Neuron (a single unit from a neural network). In each part, you should check whether the given Boolean function can be represented by a single unit or not. If the answer is yes, you should state two weights and a threshold and show that with these parameters, the given Boolean function can be represented by a single unit (you have to write your calculations). If the answer is no, you should explain and state the reason. (10 pts)

1.a XNOR Function

x_1	x_2	y
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

Solution

Let's take a look at plotted data. This function's data isn't in a linear shape. Thus, a single unit of a neural network can't solve this nonlinear classification problem.



1.b NAND Function

x_1	x_2	y
-1	-1	1
-1	1	1
1	-1	1
1	1	-1

Solution

We can solve this problem using the fixed-increment learning algorithm. Initially, the weights of the input vector$(1,x_1,x_2)$ are set to zero. Also, the learning rate (η) is set to 1.

$$W_0(b) = W_1 = W_2 = 0$$
$$\alpha = 1$$

$$Activation\ function(F) = \begin{cases} +1 & Y_{in}>0 \\ 0 & Y_{in} = 0 \\ -1 & Y_{in}<0 \end{cases}$$

$$Y_{in} = W_0 + (W_1 \times X_1) + (W_2 \times X_2)$$
$$Y_{in}^1 = 0 + (0)(-1) + (0)(-1) = 0$$
$$Y = F(Y_{in}) = 0 \rightarrow Y \neq target(+1)$$

update weights:

$$W_i^{'} = W_i + \eta\ t\ x_i$$
$$W_0^{'} = 0 + (1)(1)(1) = 1$$
$$W_1^{'} = 0 + (1)(1)(-1) = -1$$
$$W_2^{'} = 0 + (1)(1)(-1) = -1$$

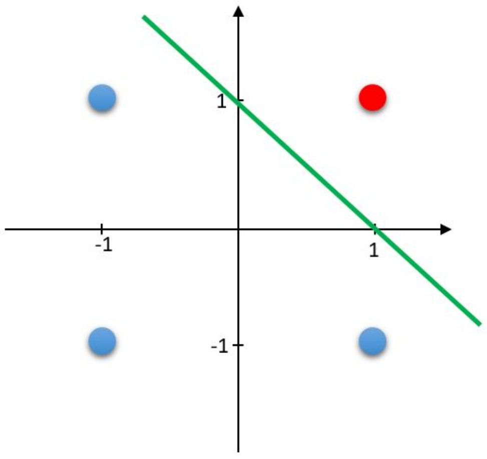
calculate Y_{in} with new weights:

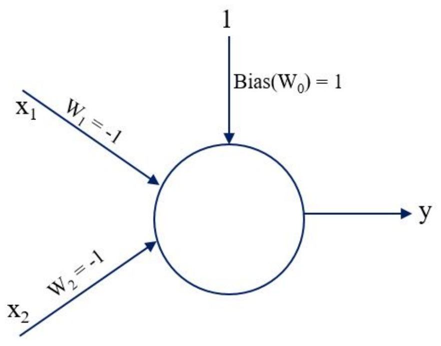
$$Y_{in}^1 = 1 + (-1)(-1) + (-1)(-1) = 3$$
$$Y = F(Y_{in}) = 1 \rightarrow Y = target(+1)$$

$$Y_{in}^2 = 1 + (-1)(-1) + (-1)(1) = 1$$
$$Y = F(Y_{in}) = 1 \rightarrow Y = target(+1)$$

$$Y_{in}^3 = 1 + (-1)(1) + (-1)(-1) = 1$$
$$Y = F(Y_{in}) = 1 \rightarrow Y = target(+1)$$

$$Y_{in}^4 = 1 + (-1)(1) + (-1)(1) = -1$$
$$Y = F(Y_{in}) = -1 \rightarrow Y = target(-1)$$





2 Problem 2

Construct an OR function for an ADALINE neuron. (Choose suitable initial weights, initial bias, and learning rate, and explain your reason.) (Write only four steps) (15 pts)

Solution

Step 1: Set weights, bias, and Learning rate(α)

$$
\begin{gathered}
W_0 = 0.4 \\
W_1 = 0.2 \\
W_2 = 0.1 \\
\alpha = 0.1
\end{gathered}
$$

Step 2: Calculate Y_{in}

$$
\begin{gathered}
Y_{in} = W_0 + (W_1 \times X_1) + (W_2 \times X_2) \\
Y_{in}^1 = 0.4 + (0.2)(-1) + (0.1)(-1) = 0.1 \\
\rightarrow Y_{in} \neq target
\end{gathered}
$$

Step 3: Update the weights

$$
\begin{gathered}
W_i^{'} = W_i + \alpha\ (t - Y_{in})\ X_i \\
W_0^{'} = 0.4 + (0.1)(-1 - 0.1)(1) = 0.29 \\
W_1^{'} = 0.2 + (0.1)(-1 - 0.1)(-1) = 0.31 \\
W_0^{'} = 0.1 + (0.1)(-1 - 0.1)(-1) = 0.21
\end{gathered}
$$

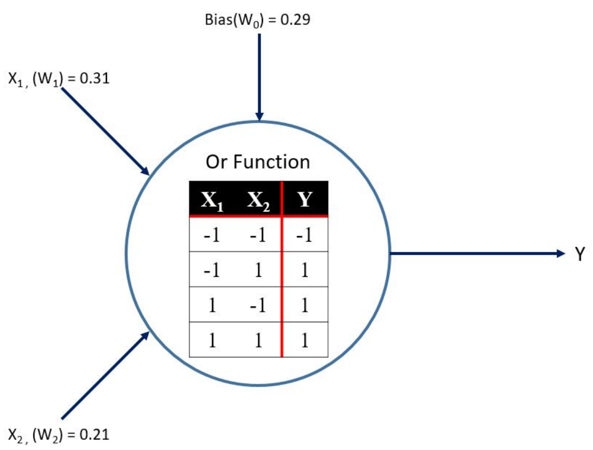
Step 4: Calculate Y_{in}

$$
\begin{gathered}
Y_{in} = W_0 + (W_1 \times X_1) + (W_2 \times X_2) \\
Y_{in}^1 = 0.29 + (0.31)(-1) + (0.21)(-1) = -0.23
\end{gathered}
$$

$$Y_{in}^2 = 0.29 + (0.31)(-1) + (0.21)(1) = 0.19$$

$$Y_{in}^2 = 0.29 + (0.31)(1) + (0.21)(-1) = 0.39$$

$$Y_{in}^4 = 0.29 + (0.31)(1) + (0.21)(1) = 0.81$$



X_1	X_2	Y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

3 Problem 3

In this section you have to do some research.

3.a

Explain the limitations of the Perceptron.

Perceptron networks have several limitations.

- **First**, the output values of a perceptron can take on only one of two values (0 or 1) due to the hard-limit transfer function.
- **Second**, perceptrons can only classify linearly separable sets of vectors. If a straight line or a plane can be drawn to separate the input vectors into their correct categories, the input vectors are linearly separable. If the vectors are not linearly separable, learning will never reach a point where all vectors are classified properly. Note, however, that it has been proven that if the vectors are linearly separable, perceptrons trained adaptively will always find a solution in finite time. You might want to try demop6. It shows the difficulty of trying to classify input vectors that are not linearly separable.
- **Third**, Unfortunately, there is no proof that such a training algorithm converges for perceptrons. On that account the use of train for perceptrons is not recommended.

3.b

Explain Sigmoid, tanh, Softmax, ReLU and ELU activation functions and make a comparison between them.

An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations. The purpose of an activation function is to add non-linearity to the neural network.

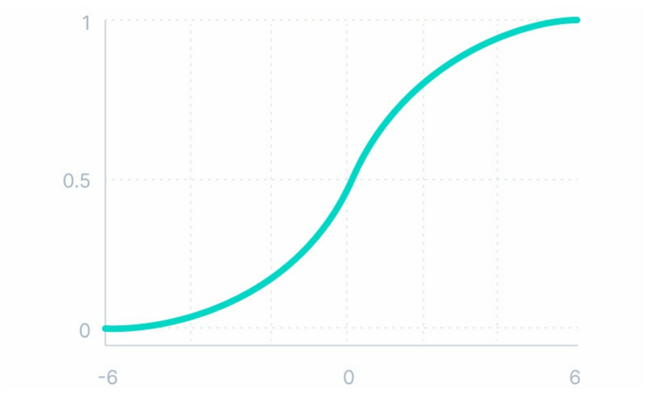
Sigmoid Function

This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below. The advantages of using sigmoid as an activation function are as follows:

- t is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.
- The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

Mathematically it can be represented as:

$$Sigmoid\ Function = \frac{1}{1 + e^{-x}}$$



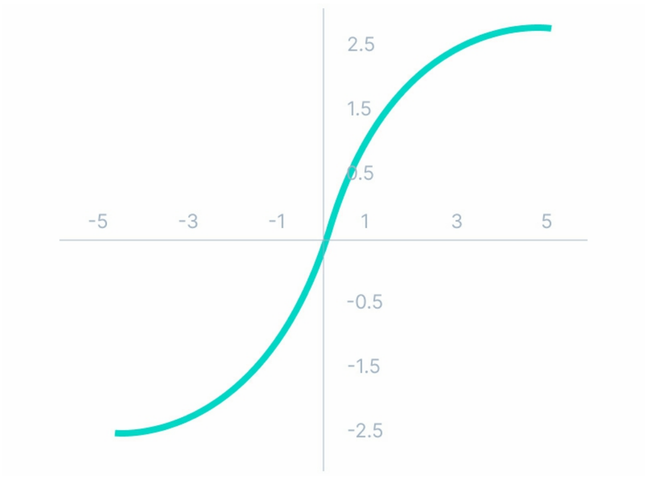
Tanh Function

Tanh function is very similar to the sigmoid activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0. Advantages of using this activation function are:

- The output of the tanh activation function is Zero centered; hence we can easily map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as its values lie between -1 to; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centering the data and makes learning for the next layer much easier.

Mathematically it can be represented as:

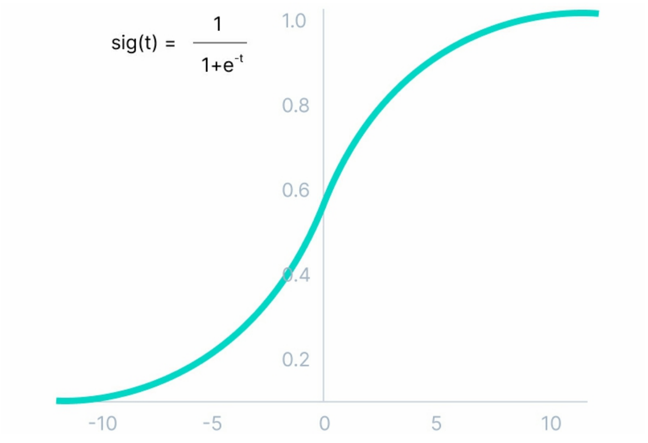
$$Tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$



Softmax Function

It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class. It is most commonly used as an activation function for the last layer of the neural network in the case of multi-class classification. Mathematically it can be represented as:

$$SoftMax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_i)}$$



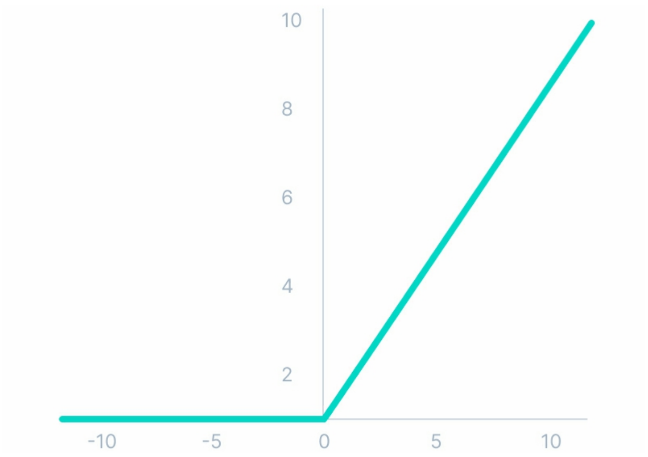
ReLU Function

ReLU stands for Rectified Linear Unit. Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient. The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0. The advantages of using ReLU as an activation function are as follows:

- Since only a certain number of neurons are activated, the ReLU function is far more computationally efficient when compared to the sigmoid and tanh functions.
- ReLU accelerates the convergence of gradient descent towards the global minimum of the loss function due to its linear, non-saturating property.

Mathematically it can be represented as:

$$ReLU(x) = \max(0, x)$$



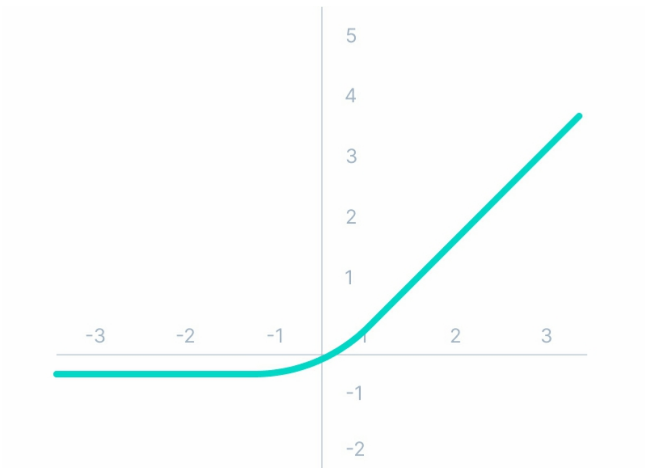
Exponential Linear Units (ELUs) Function

Exponential Linear Unit, or ELU for short, is also a variant of ReLU that modifies the slope of the negative part of the function. ELU uses a log curve to define the negativ values unlike the leaky ReLU and Parametric ReLU functions with a straight line. ELU is a strong alternative for f ReLU because of the following advantages:

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.
- Avoids dead ReLU problem by introducing log curve for negative values of input. It helps the network nudge weights and biases in the right direction.

Mathematically it can be represented as:

$$ELU = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x<0 \end{cases}$$



3.c

Optimizers are algorithms or methods used to adjust the parameters of the neural network, such as weights to minimize the loss function. In the course, Gradient Descent and Stochastic Gradient Descent(SGD) were explained. Please explain the disadvantages of SGD and explain the following two optimization algorithms and compare them with SGD:

- SGD with momentum
- RMS-Prop

(For this question, you should search the topics. You can use websites like medium.com and towardsdatascience.com) (25 pts)

Disadvantages of Stochastic Gradient Descent

- Due to frequent updates the steps taken towards the minima are very noisy. This can often lead the gradient descent into other directions.
- Also, due to noisy steps it may take longer to achieve convergence to the minima of the loss function.
- Frequent updates are computationally expensive due to using all resources for processing one training sample at a time.
- It loses the advantage of vectorized operations as it deals with only a single example at a time.

Mini Batch Stochastic Gradient Descent with Momentum

Momentum simulates the inertia of an object when it is moving, i.e. the way of the previous update is retained to a some percentage during the update, while the current update gradient is used to optimize the final update direction. Stability is increased to a certain extent, so learning is faster, and also have the ability to get rid of local optimization.

RMS-Prop

The RMSProp algorithm full form is called Root Mean Square Prop, which is an adaptive learning rate optimization algorithm proposed by Geoff Hinton. RMSProp tries to resolve Adagrad's radically diminishing learning rates problem by using a moving average of the squared gradient. It utilizes the magnitude of the recent gradient descents to normalize the gradient as shown in the equation below. While Adagrad accumulates all previous gradient squares, RMSprop just calculates the corresponding average value, so it can eliminate the problem of quickly dropping of learning rate of the Adagrad. In RMSProp learning rate gets adjusted automatically and it chooses a different learning rate for each parameter. This algorithm divides the learning rate by the average of the exponential decay of squared gradients.

4 References

[1] http://matlab.izmiran.ru/help/toolbox/nnet/percep11.html#:~:text=Perceptron%20networks%20have%20several%20limitations,linearly%20separable%20sets%20of%20vectors.

[2] https://www.v7labs.com/blog/neural-networks-activation-functions

[3] https://medium.com/@divakar_239/stochastic-vs-batch-gradient-descent-8820568eada1

[4] https://www.oreilly.com/library/view/learn-arcore-/9781788830409/e24a657a-a5c6-4ff2-b9ea-9418a7a5d24c.xhtml

[5] https://towardsdatascience.com/optimizers-88694509311c