# ML Assignment | Practical-3

## Problem 1:

1) To solve the problem, let's apply perceptron learning algorithm, step by step over the sample data. We are given an initial weight vector $w = [1, 1]$ and sample data point of respective classes.

(i) Initialize the weight vector, $w = [1, 1]$

(ii) Iterate through the ~~weight vector~~ data points and update the weight vector as we encounter ~~a [illegible]~~ a mis classification.

Iteration 1:

(i) $(1, 1)$ is correctly classified as $+1$;
$w = [1, 1]$

(ii) $(-1, -1)$ is incorrectly classified as $+1$,
$\therefore w\ [1, 1] + (-1 \times -1, -1 \times -1) = [2, 2]$

(iii) $(0, 0.5)$ is incorrectly classified as class 1: $w = [2, 2] + (-1 \times 0, -1 \times 0.5) = [2, 1.5]$

(iv) $(0.1, 0.5)$ is incorrectly classified as 1:

$w = [2, 1.5] + (-1 \times 0.1, -1 \times 0.5) = [1.9, 1]$

(v) $(0.2, 0.2)$ is correctly classified as 1:

$w = [1.9, 1]$

(vi) $(0.9, 0.5)$ is correctly classified as 1:

$w = [1.9, 1]$

<u>Iteration 2</u>:

(i) $(1, 1)$ is correctly classified as 1: $w = [1.9, 1]$

(ii) $(-1, -1)$ is correctly classified as $-1$: $w = [1.9, 1]$

(iii) $(0, 0.5)$ is correctly classified as $-1$: $w = [1.9, 1]$

(iv) $(0.1, 0.5)$ is correctly classified as $-1$: $w = [1.9, 1]$

(v) $(0.2, 0.2)$ is correctly classified as 1: $w = [1.9, 1]$

(vi) $(0.9, 0.5)$ is correctly classified as 1: $w = [1.9, 1]$

Therefore the perceptron learning algorithm converges after 2 iterations.

2). Let's go through stepwise update of the weight vector using Perceptron learning algorithm for the given sample data and initial weight vector $w = [1, 1]$.

Initial weight vector: $w = [1, 1]$

Decision boundary: $x_2 = -x_1 + 0$

Iteration 1:

(i) $(1, 1)$ is correctly classified as 1: $w = [1, 1]$

(ii) $(-1, -1)$ is incorrectly classified as 1:

$$w = [1, 1] + (-1 \times -1, -1 \times -1)$$
$$= [2, 2]$$

Decision boundary: $x_2 = -x_1 + 1$

(iii) $(0, 0.5)$ is incorrectly classified as 1:

$$w = [2, 2] + (-1 \times 0, -1 \times 0.5) = [2, 1.5]$$

Decision boundary: $x_2 = -x_1 + 0.75$

(iv) $(0.1, 0.5)$ is incorrectly classified as 1:

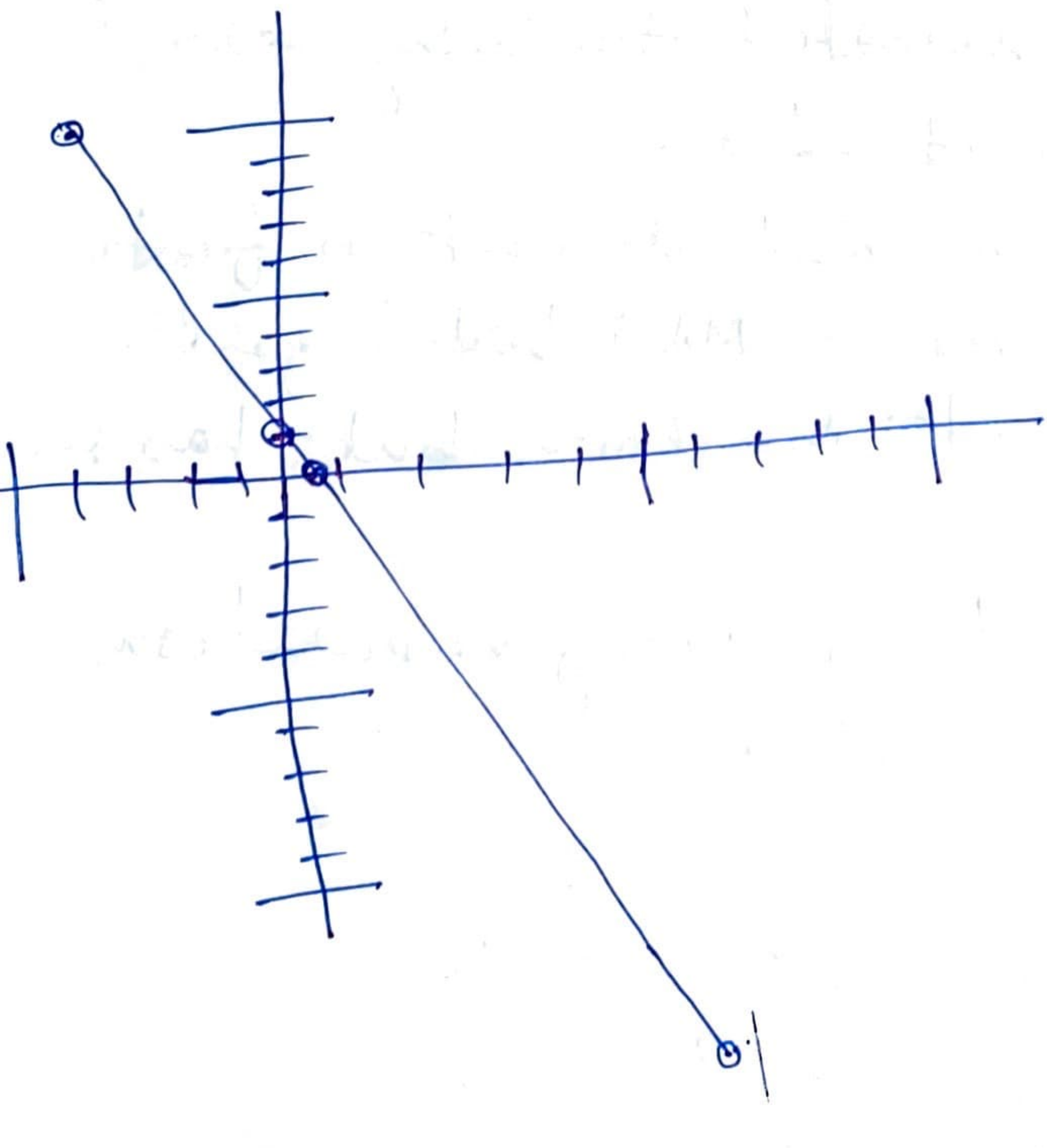
$$w = [2, 1.5] + (-1 \times 0.1, -1 \times 0.5) = [1.9, 1]$$

Decision boundary: $x_2 = -1.9/1 \times x_1 + 1$

Iteration 2:

At this point all the samples are correctly classified by the decision boundary. The final decision boundary is as follows:

$$x_2 = -1.9 \times x_1 + 1$$



Problem 4:

1) Stochastic gradient descent updates the model parameter using a single randomly selected training example at each iteration, while mini batch Gradient descent updates the model parameters using a small subset of randomly selected training examples at each iteration.

Stochastic gradient descent is faster but more noisy, Mini batch gradient ~~des~~ descent is slower but has less variance

2) We have the following calculations:

$f = w_1^3 a_1^2 + w_2^3 a_2^3$

(i) $\partial f / \partial z_1^2$:

$\hookrightarrow f = w_1^3 a_1^2 + w_2^3 a_2^2$

$z_1^2 = w_{11} a_1^1 + w_{21} a_2^1$

f depends on $z_1^2$ through $a_1^2$ which in turn depends on $z_1^2$, so we need the chain rule twice.

$\partial f / \partial z_1^2 = \partial f / \partial a_1^2 \times \partial a_1^2 / \partial z_1^2$

$\partial f / \partial a_1^2 = w_1^3$, $\partial a_1^2 / \partial z_1^2 = \sigma'(z_1^2)$,

where, $\sigma'$ is the derivative of the activation function at $z_1^2$.

So, $\partial f / \partial z_1^2 = w_1^3 \times \sigma'(z_1^2)$

(ii) $\partial f / \partial z^2$:

f - depends on $a_2^2$ through $w_2^3$, so we have $\partial f / \partial a_2^2 \times \partial a_2^2 / \partial z_2$

$\partial f / \partial a_2^2 = w_2^3$, $\partial a_2^2 / \partial z_2 = \sigma'(z^2)$

So, $\partial f / \partial z^2 = w_2^3 \times \sigma'(z^2)$

(iii) $\partial f / \partial z^1$:

~~f - depends on $a_1^2$ through $w_2^3$, so we have $\partial f / \partial a_2^2 \times \partial a_2^2 / \partial z^2$~~

f - depends on $a_1$ through $w_{11}$, so we have $\partial f / \partial a_1^1 \times \partial a_1^1 / \partial z_1^1$

$\partial f / \partial a_1^1 = w_1^3 \times w_{11}$, $\partial a_1^1 / \partial z_1 = \sigma'(z^1)$

So, $\partial f / \partial z^1 = w_1^3 \times w_{11} \times \sigma'(z^1)$

(iv) $\partial f / \partial w_{11}$:

f - depends on $a'_1$ through $w_{11}$, so we have $\partial f / \partial w_{11} = \partial f / \partial a'_1 \times \partial a'_1 / \partial w_{11}$

$\partial f / \partial a'_1 = w_1^3 \times w_{1r}$ $\partial a'_1 / \partial w_{11} = x_1$

So, $\partial f / \partial w_{11} = w_1^3 \times x_1$.

## Problem 5:

1) Tuning hyper parameter using a test dataset can lead to overfitting, which means that the model will perform well on the test dataset but will perform poorly on new or unseen data. This happens because the model learns the pattern specific to the test dataset rather or generalized data.

Additionally, using test dataset for hyper parameters can bias the evaluation on model's performance as the dataset is no longer an unbiased measure of the model's generalization ability. It's better to use a separate validation dataset for hyperparameter tuning and keep the test dataset reserved for final evaluation of the model's performance.

2> Here are two strategies for addressing overfitting problem in neural network:

(i) Regularization: It is a technique used to prevent overfitting by adding a penalty term to the loss function. This penalty term penalizes large weights and biases and encourages the model to use smaller weights and biases.

(ii) Data Augmentation: It is a technique to increase the size of the training dataset by applying the transformation to the existing data. This allows the model to learn from a larger and more diverse dataset, which can prevent overfitting.

3) The input layer size for a neural network is determined by the number of input features in the dataset. If there are n- input features, then input layer would be of size n.

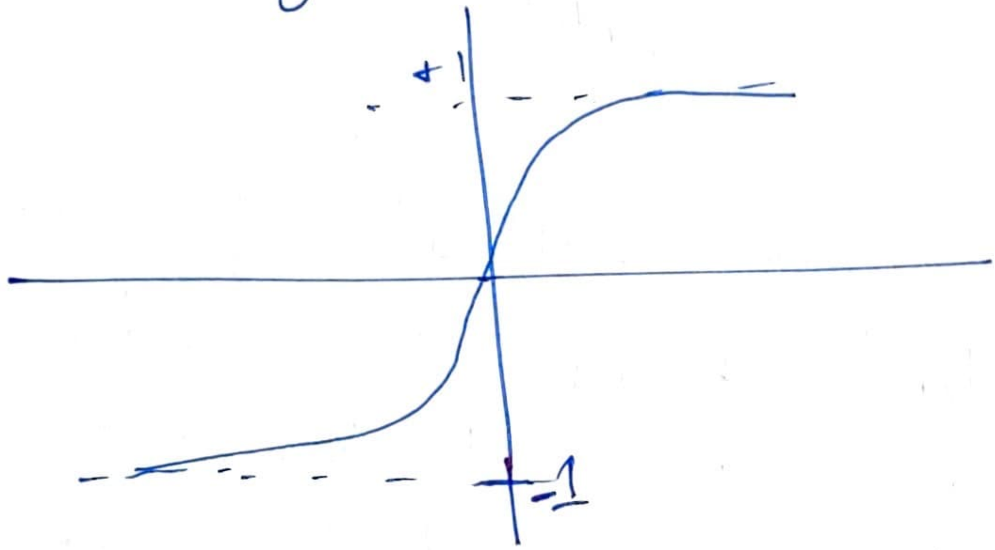
The output layer size depends on the type of the problem being solved. For a binary classification, the output layer's size would be of 1, while for a multi-class classification with 'K' classes, the output layer size would be of size - 'K'.

For a regression problem, the o/p layer size would be of 1 if there is a single target and more than 1 if there are multiple target variables.

4) The sigmoid activation function is commonly used non-linear activation function in neural networks. It takes an input value and maps it to value between 0 and 1. Here's the formula:

$$f(x) = 1/(1+e^{-x})$$

The sigmoid function is commonly used to introduce non-linearity into the o/p of a neural network. It is often used as a activation function of an output function of a binary classification problem. The o/p of a sigmoid function can be interpreted as a probability, where values close to 1 indicates high probability of the positive class and values close to 0 indicate high probability of the negative class.




5) The learning rate of a model is a hyperparameter that controls how much the weights of the network are adjusted during the training process. It determines the step size at each iterations of the optimization algorithm, thus affects directly how quickly a network converges to a solution.

A very large learning rate can cause the model to overshoot the optimal solution and may result in ~~a~~ instability or divergence of the training process. ~~B~~

~~determination the .. sol~~

This can lead to poor performance on the training set and generalize poorly on the test set. On the other hand, a learning rate that is too small can result in slow convergence and may get stuck in local optima. Therefore, it is important to choose an appropriate learning rate that balances between convergence speed and optimization stability.