Machine Learning Assignment (Fract-3)

Debonil Ghosh (M21AIE225)

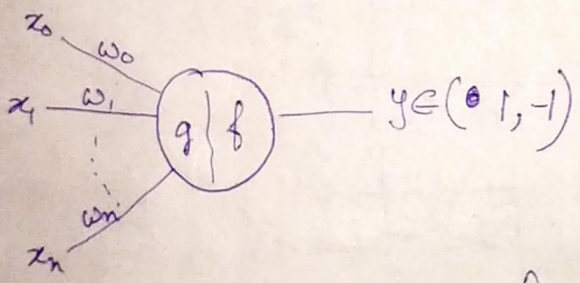
[Hand Written Part]

Q1. Perceptron

x_1	x_2	Class
1	1	+1
-1	-1	-1
0	0.5	-1
0.1	0.5	-1
0.2	0.2	+1
0.9	0.5	+1

Sample data

Initial decision boundary $w^T x = 0$ as $w = [1, 1]$

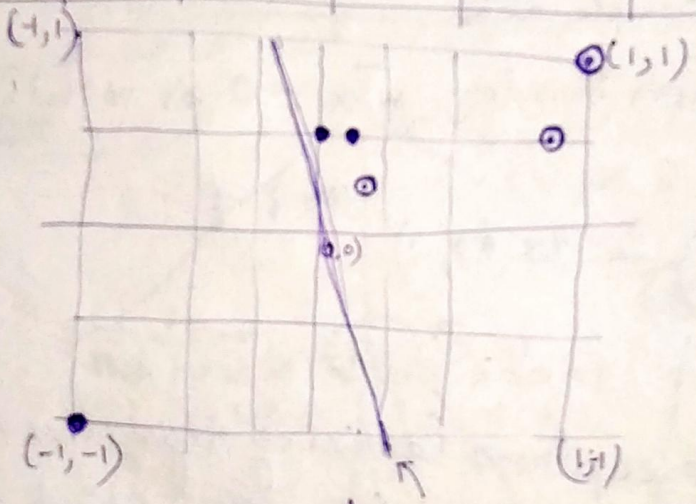


1.1. The perceptron learning Algorithm will converge in 4 Steps (iteration), which can be verified from the next answer.

1.2

Step/Iteration 1:

x_1	x_2	x_0	Class	w_1	w_2	w_0	$w^T x$	Correct	w updated
1	1	1	1	1	1	1	3	✓	—
-1	-1	1	-1	1	1	1	-1	✓	—
0	0.5	1	-1	1	1	1	1.5	×	[1, 0.5, 0]
0.1	0.5	1	-1	1	.5	0	0.35	×	[0.9, 0, -1]
0.2	0.2	1	1	0.9	0	-1	-0.82	×	[1.1, 0.2, 0]
0.9	0.5	1	1	1.1	0.2	0	1.09	✓	—

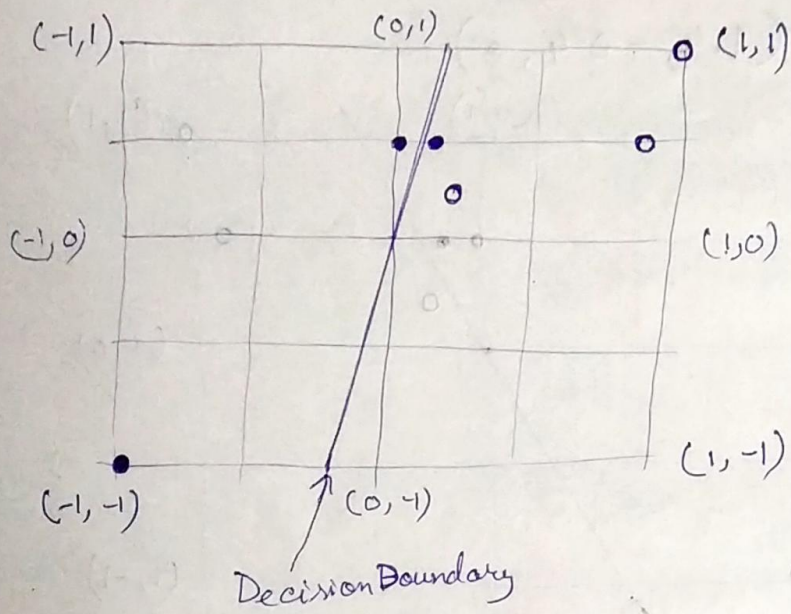


decision boundary after iteration 1
($w = [1.1, 0.2, 0]$)

Step-2:

x_1	x_2	x_0	Class	w_1	w_2	w_0	$w^T . x$	Correct	w updated
1	1	1	1	1.1	0.2	0	1.3	✓	—
-1	-1	1	-1	1.1	0.2	0	-1.3	✓	—
0	.5	1	-1	1.1	0.2	0	0.1	×	[1.1, -0.3, -1]
0.1	0.5	1	-1	1.1	-0.3	-1	-1.04	✓	—
0.2	0.2	1	1	1.1	-0.3	-1	-0.84	×	(1.3, -0.1, 0)
0.9	0.5	1	1	1.3,	-0.1,	0	1.12	✓	—

∴ Decision Boundary after step 2 is $w = (1.3, -0.1, 0)$



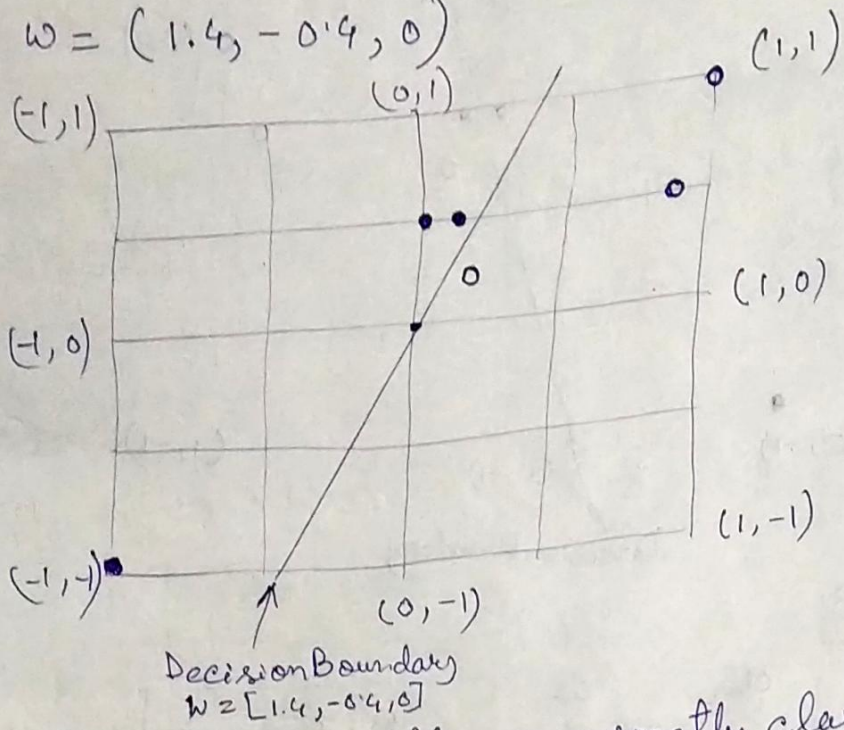
☐ Step-3:

x			Class	w			$w^T \cdot x$	Correct	w updated
x_1	x_2	•		w_1	w_2	w_0			
1	1	1	1	1.3	-0.1	0	1.2	✓	—
-1	-1	1	-1	1.3	-0.1	0	-1.2	✓	—
0	0.5	1	-1	1.3	-0.1	0	-0.05	✓	—
0.1	0.5	1	-1	1.3	-0.1	0	0.08	X	[1.2, -0.6, -1]
0.2	0.2	1	1	1.2	-0.6	-1	-0.88	X	[1.4, -0.4, 0]
0.9	0.5	1	1	1.4	-0.4	0	1.06	✓	—

☐ Step-4

x			Class	w			$w^T \cdot x$	Correct	w updated
x_1	x_2			w_1	w_2	w_0			
1	1	1	1	1.4	-0.4	0	1	✓	—
-1	-1	1	-1	1.4	-0.4	0	-1	✓	—
0	0.5	1	-1	1.4	-0.4	0	-0.2	✓	—
0.1	0.5	1	-1	1.4	-0.4	0	-0.06	✓	—
0.2	0.2	1	1	1.4	-0.4	0	0.2	✓	—
0.9	0.5	1	1	1.4	-0.4	0	1.06	✓	—

Final decision boundary (as w is not updated in step 4) is $w = (1.4, -0.4, 0)$



After step 3, all samples are correctly classified in step 4.

1.3 Prove that Perceptron Learning Algorithm converges in a finite number of steps.

Ans Suppose at timestamp t, we inspected the point P_i, and found that $w^T \cdot P_i \leq 0$

we make a correction $w_{t+1} = w_t + P_i$

Let β be the angle between w^* and w_{t+1}

$$\text{then, } \cos\beta = \frac{w^* w_{t+1}}{\| w_{t+1} \|}$$

$$\text{Numerator} = w^* \cdot w_{t+1} = w^* (w_t + p_i)$$

$$= w^* w_t + w^* p_i$$

$$\geq w^* w_t + \delta \quad \left(\delta = \min\{w^* \cdot r_i \mid v_i\} \right)$$

$$\geq w^* (w_{t-1} + p_i) + \delta$$

$$\geq w^* w_{t-1} + w^* p_i + \delta$$

$$\geq w^* w_{t-1} + 2\delta$$

Everytime we make a correction a quality δ is added. Say if k corrections are made, a quantity of $k\delta$ is added.

$$\therefore \quad \cos\beta = \frac{(w^* w_{t+1})}{\| w_{t+1} \|}$$

$$\text{Numerator} \geq w^* w_0 + k\delta \quad \text{(proved by induction)}$$

$$\text{Denominator}^2 = \| w_{t+1} \|^2$$

$$= (w_t + p_i)$$

$$= w_t^2 + 2 w_t p_i + p_i^2 \leq w_t^2 + p^2$$

$$\leq \| w_{t-1} \|^2 + 1 \qquad (\because w_t, p \leq 0)$$

$$\leq \| w_{t-1} \|^2 + k$$

$$\cos\beta \geq \frac{w^* w_0 + k\delta}{\sqrt{\| w_0 \|^2 + k}}$$

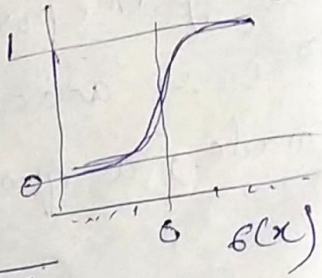
Thus $\cos\beta$ is proportional to $\sqrt{k}$

$\therefore$ k has to be finite so that $\cos\beta \leq 1$

$\therefore$ there can only be a finite number of corrections (k) to w and the perceptron Algorithm will converge.

Q.3

3.1. Compute Derivatives of the following activation functions:



1. Sigmoid:

$$\text{Sigmoid}(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d\,\sigma(x)}{dx} = \frac{d}{dx}\left(\frac{1}{1+e^{-x}}\right)$$

$$= \frac{(1+e^{-x})\frac{d}{dx}\cdot 1 - 1 \times \frac{d}{dx}e^{-x}}{(1+e^{-x})^2} \quad \text{(applying quotient rule)}$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$$

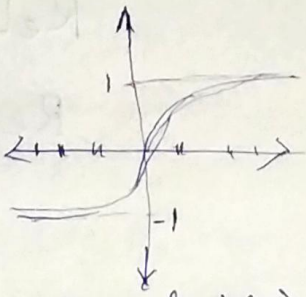
$$= \frac{1+e^{-x}}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = \sigma(x)\left(1-\sigma(x)\right)$$

$$\therefore \frac{d}{dx}\sigma(x) = \sigma(x)(1-\sigma(x))$$ Ans

2. tanh :

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



$f(x) = \tanh(x)$

$$\frac{d}{dx}T(x) = \frac{d}{dx}\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

applying quotent rule,

$$\frac{d}{dx}\frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^x + e^{-x}\left(\frac{d}{dx}(e^x - e^{-x})\right) - e^x - e^{-x}\left(\frac{d}{dx}(e^x + e^{-x})\right)}{(e^x + e^{-x})^2}$$

$$= \frac{(e^x + e^{-x})(e^x + e^{-x}) - \{(e^x - e^{-x})(e^x - e^{-x})\}}{(e^x + e^{-x})^2}$$

$$= \frac{(e^x + e^{-x})^2}{(e^x + e^{-x})^2} - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

$$= 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

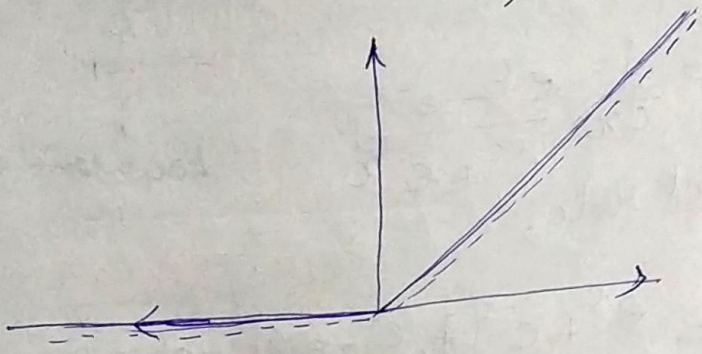
$$\frac{d}{dx}T(x) = 1 - (T(x))^2$$

$$\therefore \frac{d}{dx}\tanh = 1 - (\tanh)^2$$

3. ReLU:

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

$$ReLU = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$



$f(x) = ReLU$

$$f'(x) = \lim_{x \to 0} \frac{\max(0, x+\Delta x) - \max(0, x)}{\Delta x}$$

At $x > 0$

$$f'(x) = \lim_{x \to 0} \frac{x + \Delta x - x}{\Delta x} = 1$$

At $x < 0$

$$f'(x) = \lim_{x \to 0} \frac{0 - 0}{\Delta x} = 0$$

At $x = 0$

$$f'(x) = \lim_{x \to 0^+} \frac{0 + \Delta x - 0}{\Delta x} = 1$$

$$f'(x) = \lim_{x \to 0^-} \frac{0 - 0}{\Delta x} = 0$$

LHL $\neq$ RHL

$$\therefore \frac{d}{dx} ReLU = \frac{d}{dx} \max(0, x)$$

$$= \begin{cases} 1 & , x > 0 \\ 0 & , x < 0 \\ \text{Does not exist} & , x = 0 \end{cases}$$

3.2 What are the strategies you will follow to avoid overfitting in neural network.

Ans:

When a model of neural network learns training data very well but does not perform well on testing dataset, then the situation is called Overfitting. Several precautions may be taken to avoid overfitting. and these are following:

① Simplifying the model: By reducing the number of Internal layes and nodes in a layer, making network smaller.

② Early Stopping: Stopping the learning or training process of the model before it crosses over fitting.

③ Data Augmentation: By making small changes into the existing dataset, we make the model consider each augmented model as a distinct dataset. this will reduce overfitting of a ~~naural~~ neural network.

④ Regularization: It can be used to reduce the complexity of a model. The best way is to add penalty to the loss function in proportion to the weight of the model.

Q. 3.3

$x = [1,1]^T$, $y = [1,1]^T \in \mathbb{R}^n$

$f: \mathbb{R}^n \Rightarrow \mathbb{R}^n$ with $f(z) = z_1 x + z_2 y$ for any $z = [z_1, z_2]^T \in \mathbb{R}^n$.

$z = g(r) = [r^n, r^3]$ where $r \in \mathbb{R}$.

$$\frac{\partial f}{\partial r} = \frac{\partial f}{\partial z} \times \frac{\partial z}{\partial r} = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} 2r & 3r^n \end{bmatrix}$$

$$= \begin{bmatrix} 2r x_1 & 3r^n y_1 \\ 2r x_2 & 3r^n y_2 \end{bmatrix}_{\substack{x=[1,1] \\ y=[1,1] \\ r=2}} = \begin{bmatrix} 4 & 12 \\ 4 & 12 \end{bmatrix}$$

Q. 3.4

① Mean Squar Error (MSE): Mean Square Error calculates the mean of distance of set data points from a regression line. It calculates the difference between the truth and prediction and square it. The reason behind the squaring the result is to make the negative difference not relevant.

$$MSE = (Prediction - Truth)^2$$

for n number of data points,

$$MSE = \frac{(Pred_1 - Truth_1)^2 + (Pred_2 - Truth_2)^2 + \cdots + (Pred_n - Truth_n)^2}{n}$$

② Binary Cross-Entropy (BCE): Binary Cross Entropy is the negative average of the log of corrected predicted probabilities. It compares each of the predicted probabilities to actual class output which can be either 0, or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close of far from the actual value.

$$\text{Binary Cross Entropy} = -\frac{1}{N}\sum_{i=1}^{N}\left(\log(p_i)\right)$$

(3) Categorical Cross Entropy: It is a loss function that is used in multiclass classification task. These are tasks where an example can only belong to one out of many possible categories and the model must decide which one.

$$\text{Categorical Cross Entropy} = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

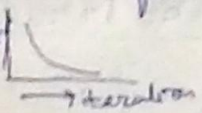
where, N is the number of rows
M is the number of classes

Q. 3.5: Explane the following variants of Gradient Descent in brief:

(i) Batch Gradient Descent: Batch Gradient Descent computes the gradient of the cost function w.r.t. to the parameter θ for the entire training dataset

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

With Batch Gradient Descent function cost function reduces smoothly.

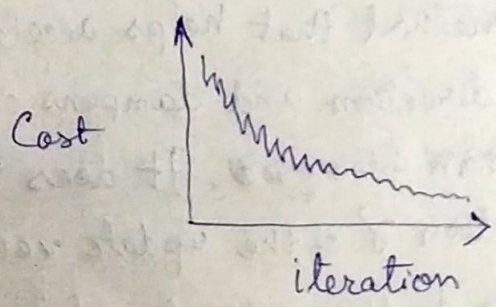


(ii) **Stochastic Gradient Descent** : Stochastic Gradient Descent performs a parameter update for each training example x^i and label y^i:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

Batch GD performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update.

SDG does away with this redundancy by performing one update at a time it is therefore usually much faster and can also be used to learn online.



Parameter updation in case of SGD is not that smooth, as it is possible to learn noise as well.

(iii) Mini Batch Gradient Descent: Mini-batch Gradient Descent

takes the best of both worlds (BGD and SGD) and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

this way it (a) reduces the variance of the parameter update, which can lead to more stable convergence; (b) can make use of highly optimized matrix optimizations common to deep learning libraries.

(iv) Momentum Based Gradient Descent:

Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. ~~as can be seen in the image~~. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

(v) Adam Solver: Adaptive Moment Estimation (ADAM) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop. Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. The decaying averages of past and past squared gradients m_t and v_t respectively follows.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$