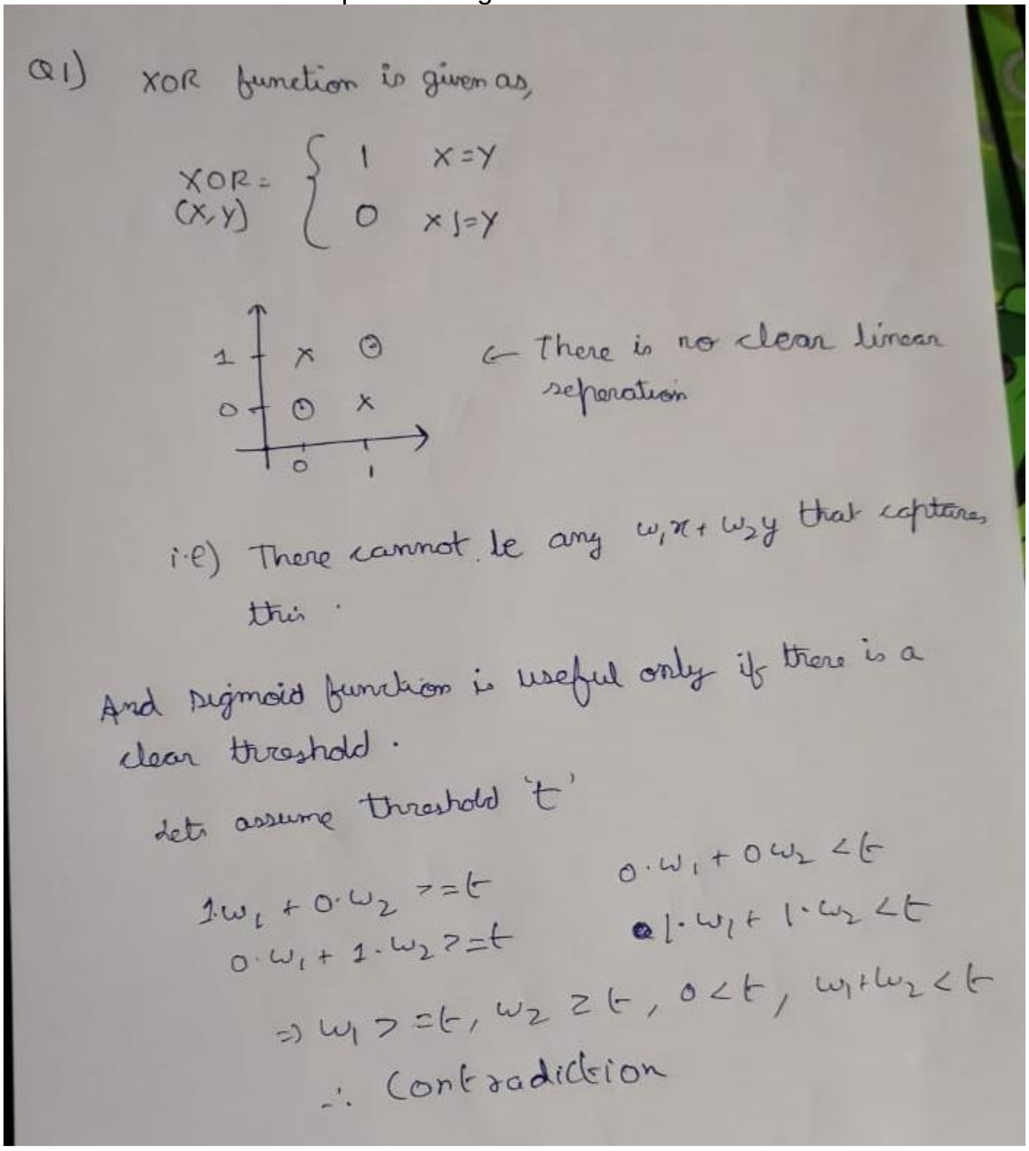
Q1)

Linear neuron **cannot** compute XOR logic.

Proof:

Assume XOR can be computed using a linear neuron



Q1) XOR function is given as,

$$XOR(x,y) = \begin{cases} 1 & x = y \\ 0 & x != y \end{cases}$$

← There is no clear linear seperation

i.e) There cannot be any $w_1x + w_2y$ that captures this.

And sigmoid function is useful only if there is a clear threshold.

Lets assume threshold 't'

$1 \cdot w_1 + 0 \cdot w_2 >= t$ $\quad$ $0 \cdot w_1 + 0 w_2 < t$

$0 \cdot w_1 + 1 \cdot w_2 >= t$ $\quad$ $1 \cdot w_1 + 1 \cdot w_2 < t$

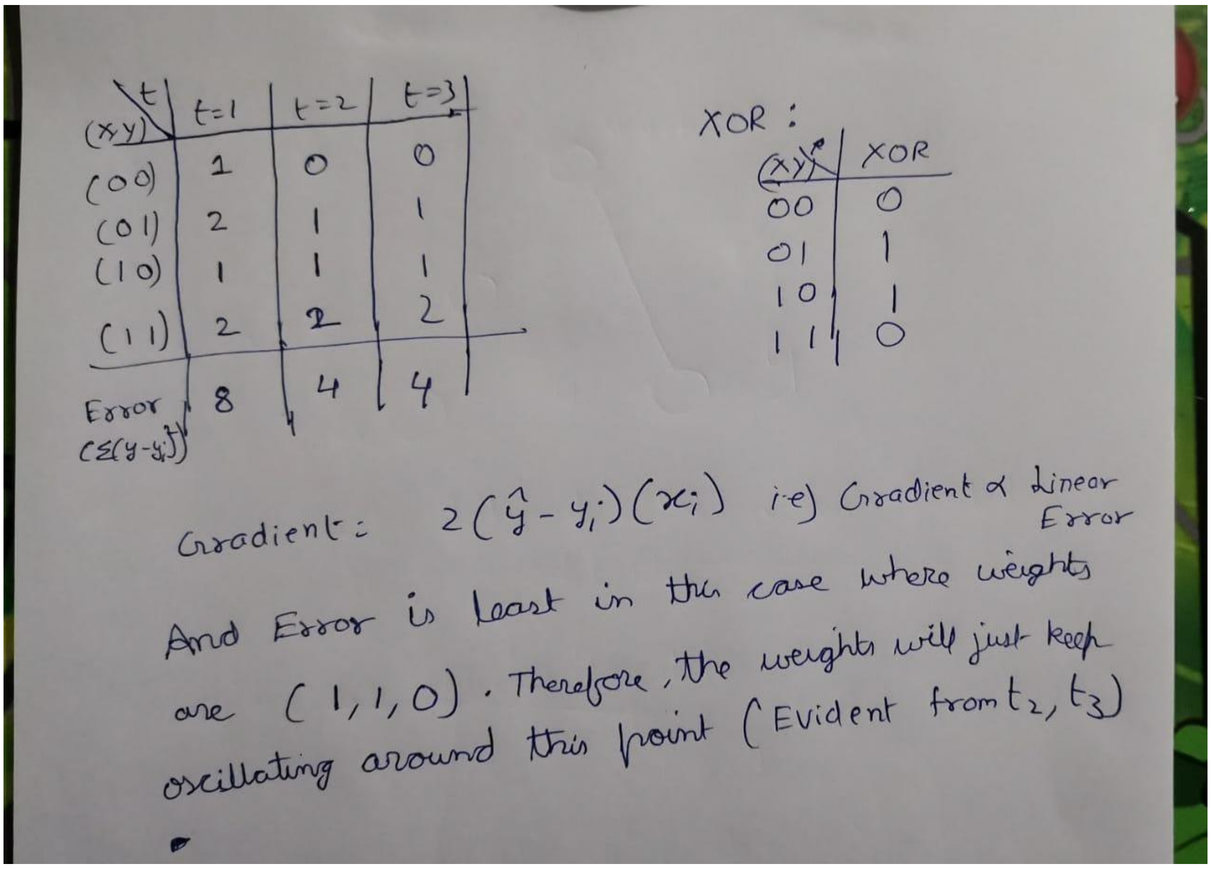
$\Rightarrow w_1 >= t, w_2 \geq t, 0 < t, w_1 + w_2 < t$

$\therefore$ Contradiction

Let’s consider the current case,

Weights along time stamps: <0,1,1>, <1,1,0>, <1,1,0>

Lets use the function of XOR.



$(x\,y) \backslash t$	$t=1$	$t=2$	$t=3$
(0 0)	1	0	0
(0 1)	2	1	1
(1 0)	1	1	1
(1 1)	2	2	2
Error $(\Sigma(y-y_i)^2)$	8	4	4

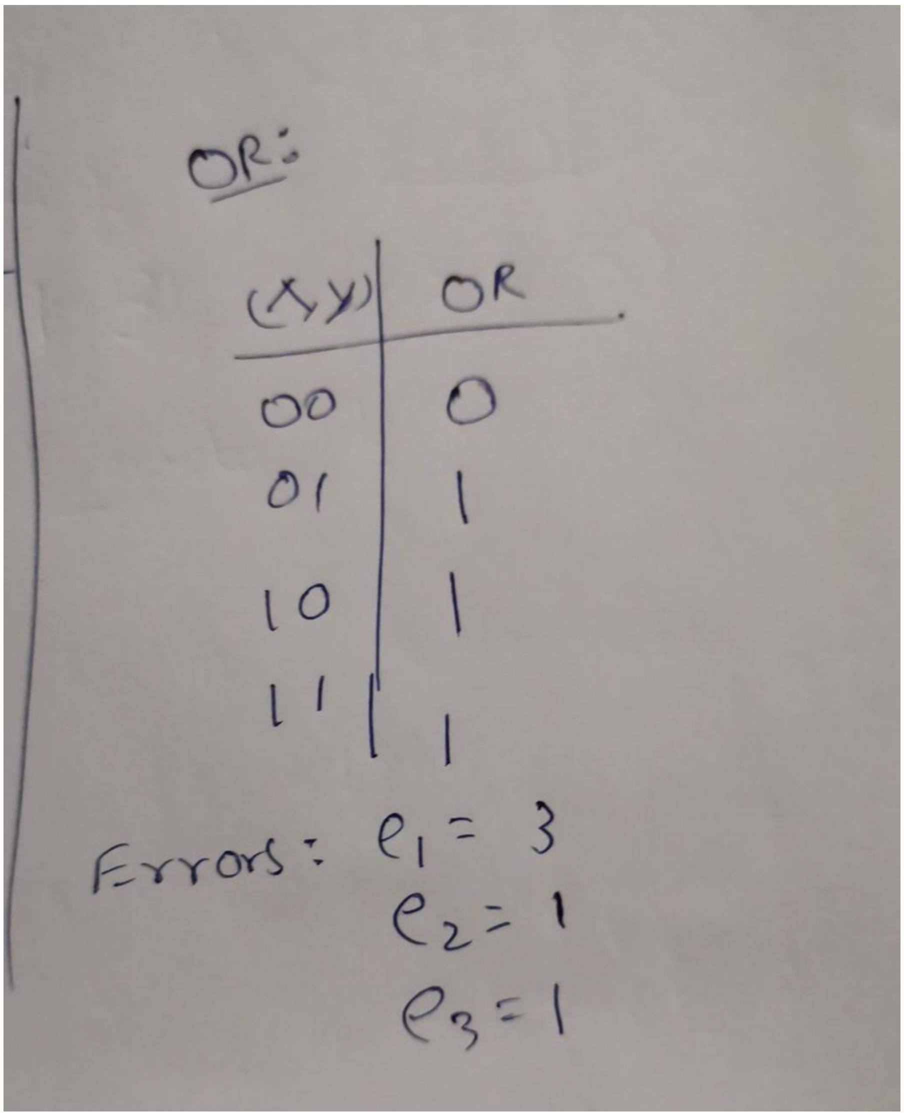
XOR :

$(x\,y)$	XOR
0 0	0
0 1	1
1 0	1
1 1	0

Gradient: $2(\hat{y} - y_i)(x_i)$ i.e) Gradient $\propto$ Linear Error

And Error is Least in the case where weights are (1, 1, 0). Therefore, the weights will just keep oscillating around this point (Evident from t_2, t_3)

We can see the model isn't convergent. However, if we take function OR.



OR:

(x,y)	OR
00	0
01	1
10	1
11	1

Errors: $e_1 = 3$

$e_2 = 1$

$e_3 = 1$

Here the data is linearly separable and can be convergent.

Therefore, the model is convergent dependent on the case. But not always

Q2)

Ensemble Learning -> Employ multiple learners and combine their prediction

There are many methods to perform Ensembing

- Bagging boosting, voting
- Error-correcting output codes
- Stacked generalization
- Cascading

Yes, Ensemble always performs better than single classifier especially in the following conditions

1. Without sufficient training data
2. Learning algorithm leads to different local optimals easily
3. Potentially valuable information may be lost by discarding the results of less-successful classifiers E.g., the discarded classifiers may correctly classify some samples

A single classifier leads to a model which is not very Accurate, Whereas Ensemble learning employs multiple learners and combines their predictions. It gives the aggregated output from various classifiers which includes the results of many other features and parameters. This leads to an improvement in predictive accuracy. Hence, Ensemble methods perform better than the single classifier.

Ensemble Methods:

Boosting: Learns sequentially and adaptively to improve model predictions of a learning algorithm.

Bagging: Learns from each other independently in parallel and combines them for determining the model average.

- Bagging is only effective only when we are using unstable nonlinear models.It decreases the variance and helps to avoid overfitting. It is usually applied to decision tree methods.
- Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers.
- Boosting is used to learn complementary classifiers so that instance classification is realized by taking the weighted sum of the classifiers
- Bagging always uses re-sampling rather than re-weighting
- Bagging does not modify the distribution over examples or mislabels, but instead always uses the uniform distribution.
- In forming the final hypothesis, bagging gives equal weight to each of the weak hypotheses

Q3)

<u>**Recurrent Neural Networks:**</u>

- A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed or undirected graph along a temporal sequence.

- Hence, RNNs are quite popular in terms of Natural Language processing tasks where words in a sentence have an inherent directed graphical relations amongst themselves.

- RNNs typically process words in a sequence wise fashion where each word is processed and unit moves to the next word and gradient descent happens in the opposite direction of output computation.

However there are some problems with respect to vanilla RNN approaches. They are:

1. RNNs are not very good at capturing long-term dependencies (Vanishing Gradient Problem):
   a. When weights are initialised too small, the resultant gradients shrink exponentially and there cant be a proper gradient descent, hence no learning.
2. RNNs could potentially result in highly variant outputs (Exploding Gradient Problem)
   . a. When weights are initialised too large, the outputs from each step will vary highly and gradients grow exponentially, resulting in fluctuating weights.

We typically use variations of Vanilla RNNs like GRU, LSTM to deal with these problems.

<u>Parallel Training in RNNs</u>

The property of RNN limits it to be sequent in nature. Therefore, the model has to wait for the past words to be processed before the processing of the current word. However, there is potential to optimise this approach.

This could be done via parallel training using data paralleling where forward propagation produces outputs for multiple inputs at a time, and the backward-propagation uses all these outputs to propagate gradients from time-step T to 1. This is called mini-batch training of neural networks.

<u>Expressibility of RNN is higher than HMM:</u>

Although HMMs are good models for sequential learning tasks, they suffer with the major limitation that it can remember only log(N) amount of bits about the past, given that the number of hidden states are N. This issue is taken care of in RNNs.
Also, HMMs are inherently much simpler than RNNs in design and rely on strong assumptions which may not always be true. An RNN may perform better if we have a very large dataset, since the extra complexity can take better advantage of the information in our data. The major loophole is that the state transitions in HMM depend only on the current state and not on the previous states. This assumption is not always true. In many real world examples the Markov assumption cannot hold true. In such cases RNNs perform better than HMMs are hence preferred.

LSTM and GRU appealing characteristics:
The architectural differences in the construction of LSTM and GRUs, make them suited for different applications. Both LSTM and GRUs were proposed with the aim of improving the problems seen in vanilla RNNs. The key architectural differences between LSTM and GRUs are: -

- The GRU has two gates, LSTM has three gates
- GRU does not possess any internal memory, they don’t have an output gate that is present in LSTM

- In LSTM the input gate and target gate are coupled by an update gate and in GRU reset gate is applied directly to the previous hidden state. In LSTM the responsibility of reset gate is taken by the two gates i.e., input and target.

We can see experimentally that GRU takes lesser time to train than LSTM, whereas LSTMs are more accurate in general. We can go with LSTM when we are dealing with large data and accuracy is important.