

# Neural Networks

(Course: Introduction to Data Science)

Tirtharaj Dash

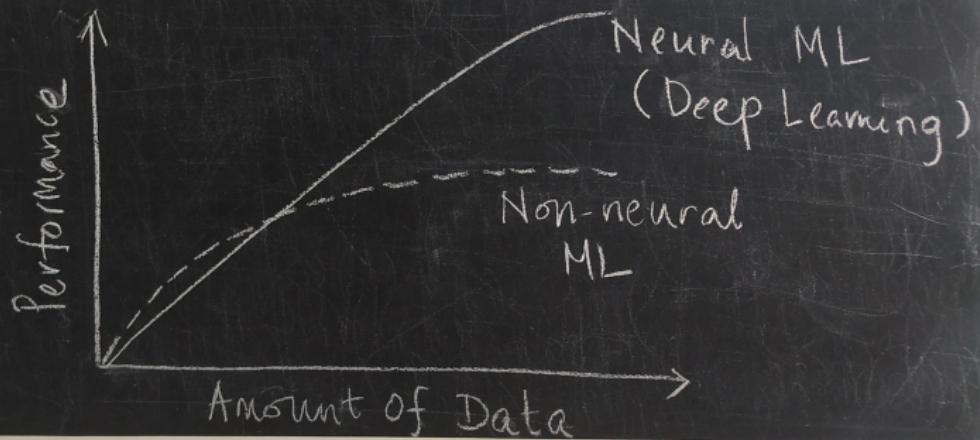
BITS Pilani, K.K. Birla Goa Campus

*tirtharaj@goa.bits-pilani.ac.in*

October 30, 2020

# Why study NNs?

Why should you study/use Neural Nets :-



# In this lecture:

- Logistic Regression as a NN (Perceptron)
- Issues with perceptron
- Layered NNs (Multilayer Perceptron)
- Activation functions

# Logistic Regression as NN I

Logistic Regression :-

$$\hat{y} = \sigma(\underline{w}^T \underline{x} + b) \quad ; \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

We interpret  $\hat{y} = P(y=1 | \underline{x})$

## Logistic Regression as NN II

In other words:-

If  $y = 1$ ,  $P(y|x) = \hat{y}$

If  $y = 0$ ,  $P(y|x) = 1 - \hat{y}$

## Logistic Regression as NN III

Cost function (binary clf.):

$$P(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$

Linear regression

$$\hat{y} = w^T x + b$$

$$L(y, \hat{y})$$

If  $y=1$ ;  $P(y|x) = \hat{y}^1 (1-\hat{y})^0 = \hat{y}$  or  $\frac{1}{2} (y - \hat{y})^2$

If  $y=0$ ,  $P(y|x) = \hat{y}^0 (1-\hat{y})^1 = 1-\hat{y}$  or  $\frac{1}{2m} \sum_m (y - \hat{y})^2$

# Logistic Regression as NN IV

Goal: Maximise  $P(y|x)$

$\Rightarrow$  maximise  $\log P(y|x)$

$$\log P(y|x) = \log \hat{y} (1-\hat{y})^{1-y}$$

$$= y \log \hat{y} + (1-y) \log (1-\hat{y})$$

$\Rightarrow$  minimise  $-\log P(y|x)$

Linear Regression

$$\hat{y} = w^T x + b$$

$$L(y, \hat{y})$$

$$= \frac{1}{2} (y - \hat{y})^2$$

$$\text{or } \frac{1}{2m} \sum_m (y - \hat{y})^2$$

The cost function:  $-\{y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\}$  is known as **cross-entropy loss function**.

# Logistic Regression as NN V

- Cross-entropy function computes the entropy between two different probability distribution.
- As  $\hat{y} \in [0, 1]$ , we can see that  $(\hat{y}, 1 - \hat{y})$  is the probability distribution obtained from the logistic model, given any example  $x$  or  $\underline{x}$ .
- *Where is the other probability distribution?* We can treat the labels as a probability distribution. Any example  $x$  is associated with a label 1 or 0 (1 or 2, or whatever). When we convert these to one-hot encoded vector we get the labelling class 1: [1,0], class 2:[0,1]. This itself is a probability distribution. Therefore, each example is associated with a true probability distribution that the logistic model tries to reach.

## Logistic Regression as NN VI

cost on m. examples :-

$$P(\text{labels in training set}) = \prod_{i=1}^m P(y^{(i)} | \underline{x}^{(i)})$$

$$\log P(\dots) = \sum_{i=1}^m \log P(y^{(i)} | \underline{x}^{(i)})$$

minimise the loss as

$$-\frac{1}{m} \sum_{i=1}^m \log P(y^{(i)} | \underline{x}^{(i)})$$

## Logistic Regression as NN VII

For convenience, let's stay with the loss for a single example:

So, the loss/cost function is [2]

$$L(y, \hat{y}) = -\left( y \log \hat{y} + (1-y) \log (1-\hat{y}) \right)$$

$$\underline{z} = \underline{w}^T \underline{x} + b$$

$$= -\left( y \log \sigma(z) + (1-y) \log (1-\sigma(z)) \right)$$

# Logistic Regression as NN VIII

What do these two terms in the loss try to achieve?

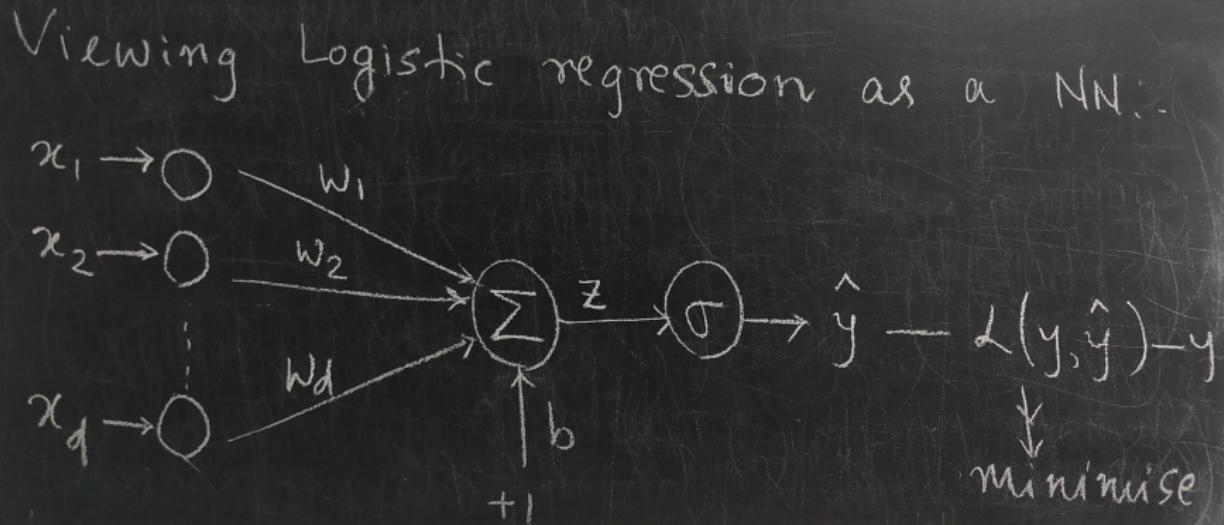
Let's re-look at the cost function.

$$-\hat{y} \log \hat{y}$$

$$-(1-\hat{y}) \log(1-\hat{y})$$



# Logistic Regression as NN IX



# Logistic Regression as NN X

Goal is to minimise  $L(y, \hat{y})$

i.e.

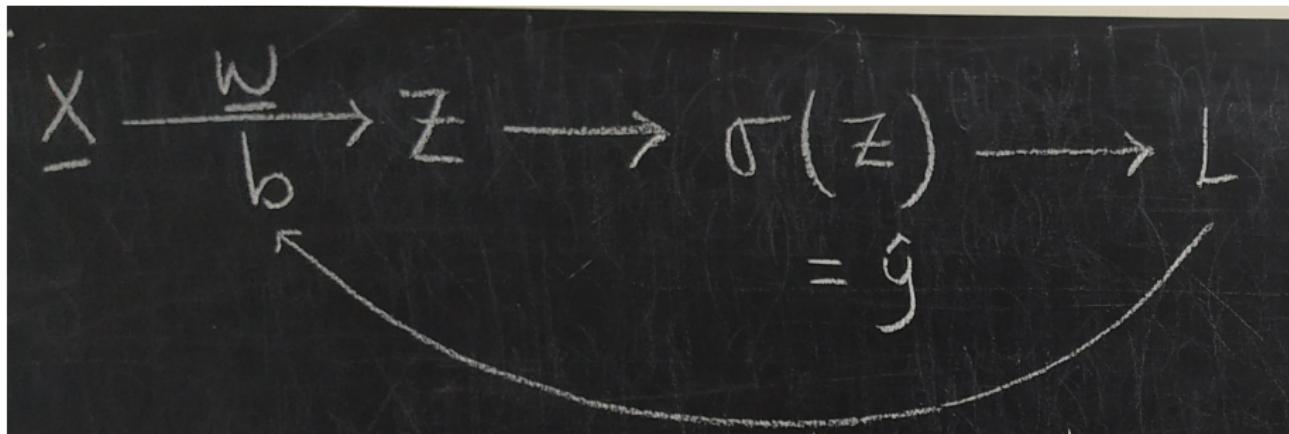
$$\min_{\underline{w}, b} L(y, \hat{y})$$

## Logistic Regression as NN XI

To minimise  $L(y, \hat{y})$  using gradient descent (GD), we need the following gradients:

$$\nabla_i \frac{\partial L}{\partial w_i} \quad \text{and} \quad \frac{\partial L}{\partial b}$$

## Logistic Regression as NN XII



# Logistic Regression as NN XIII



↓ by chain rule

$$\frac{\partial L}{\partial \underline{w}} = \boxed{\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z}} \frac{\partial Z}{\partial \underline{w}}$$

and

$$\frac{\partial L}{\partial b} = \boxed{\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial Z}} \frac{\partial Z}{\partial b}$$

## Logistic Regression as NN XIV

$$L = -y \log \hat{y} - (1-y) \log (1-\hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y}) \quad \left| \text{as } \frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z)) \right.$$

# Logistic Regression as NN XV

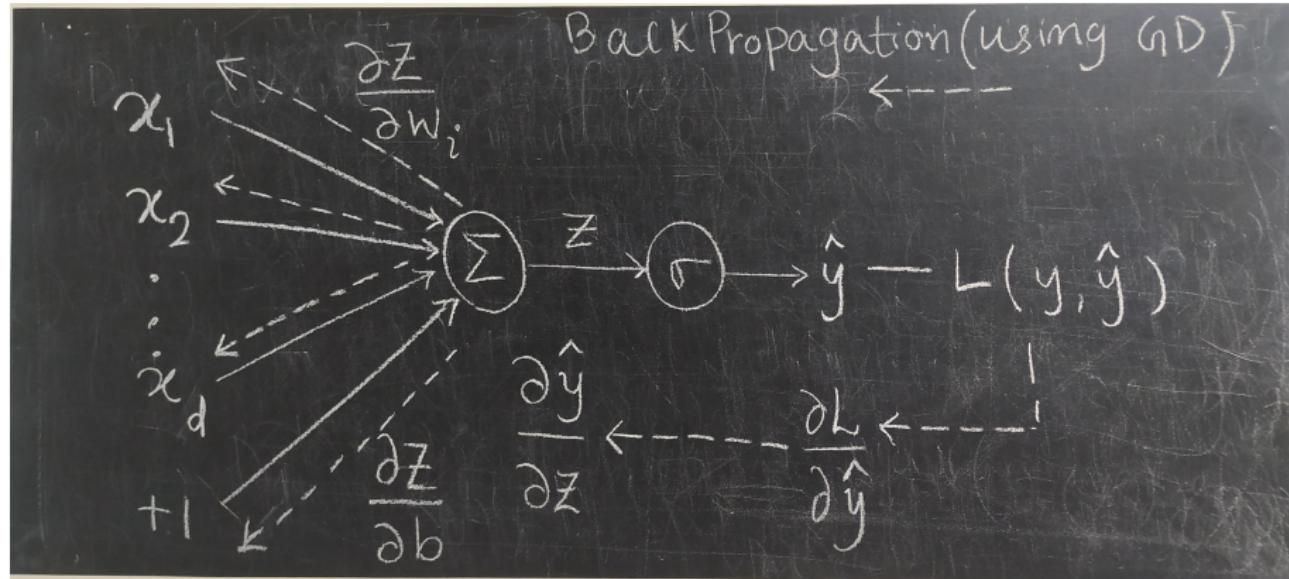
$$\text{So } \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \hat{y}(1-\hat{y})$$
$$= \hat{y} - y$$

# Logistic Regression as NN XVI

$$\frac{\partial L}{\partial w_i} = (\hat{y} - y) \frac{\partial z}{\partial w_i} = (\hat{y} - y) x_i$$

$$\frac{\partial L}{\partial b} = (\hat{y} - y) \frac{\partial z}{\partial b} = (\hat{y} - y) . 1$$

# Logistic Regression as NN XVII



# Logistic Regression as NN XVIII

To update the parameters:

$$w_i = w_i - \alpha \frac{\partial \mathcal{L}}{\partial w_i}, \quad \forall i \in \{1, \dots, d\}$$

$$b = b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$

where,  $\alpha \in (0, 1]$  is the learning rate.

## Logistic Regression as NN XIX

The simplest kind of neural network, which is analogous to the Logistic Regression is called "Perceptron".

In the NN world, these parameters  $w$  and  $b$  are called **synaptic weights** or "simply" **weights**;  $z$  is called the **net input**. The function that operates on the net input in any neuron is called **activation function**.

# Issues with Perceptron I

Problems that can be  
Solved by a perceptron:



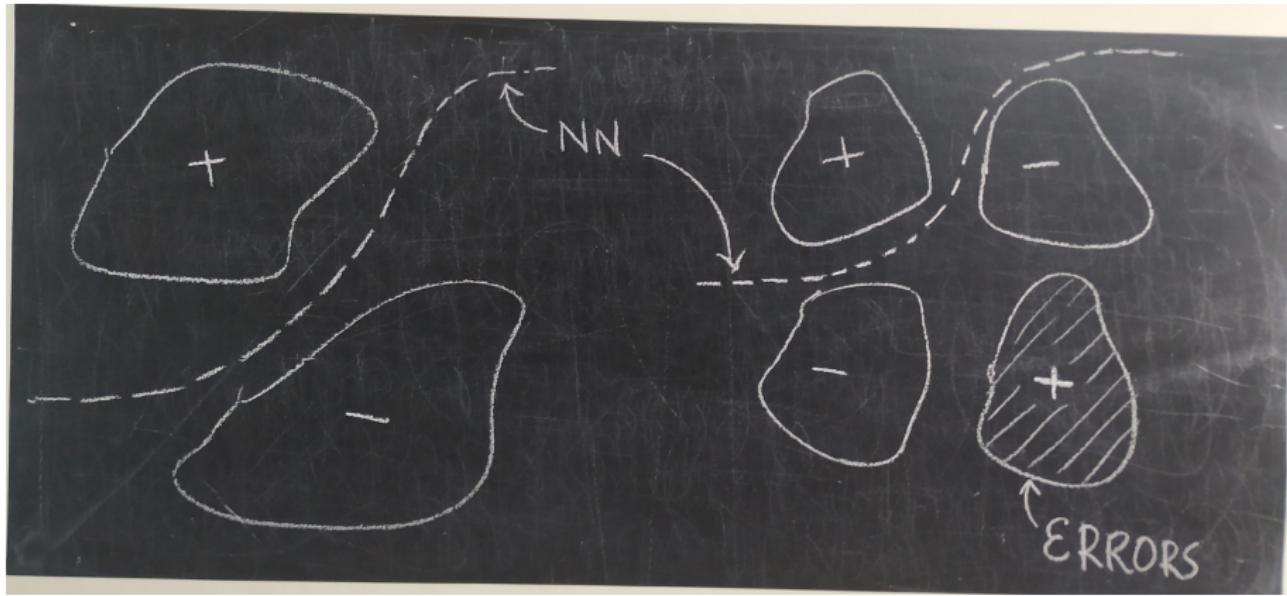
e.g. AND  
OR

cannot be solved:-



e.g.  
XOR

## Issues with Perceptron II



- Many real-world problems are of second Kind.

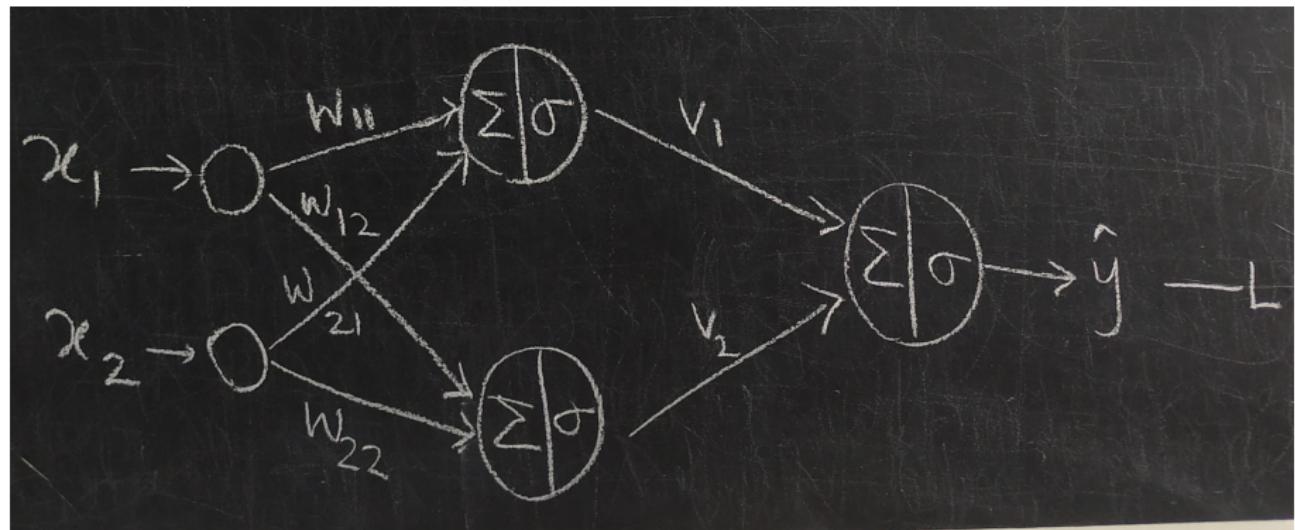
# Solution

Represent the features in a transformed feature space on which learning the model will become easier:

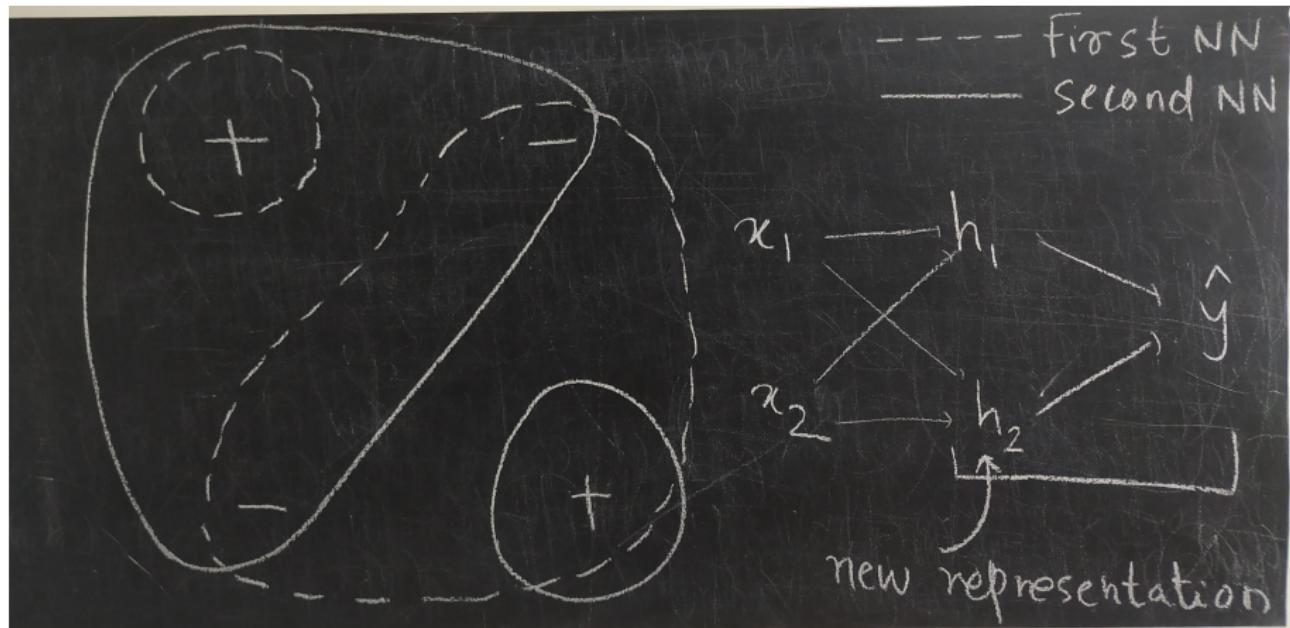
We will be restricting our study in this course to a multilayered neural network that achieves this: [Multilayer Perceptron](#)

# Multilayer NNs I

Multiple levels of transformation of features using layers:

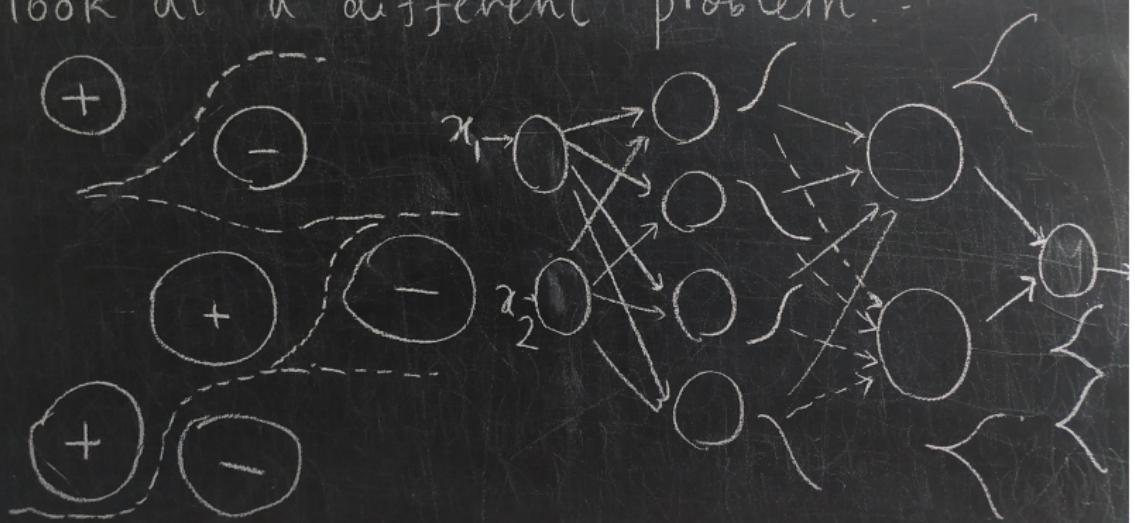


# Multilayer NNs II



# Multilayer NNs III

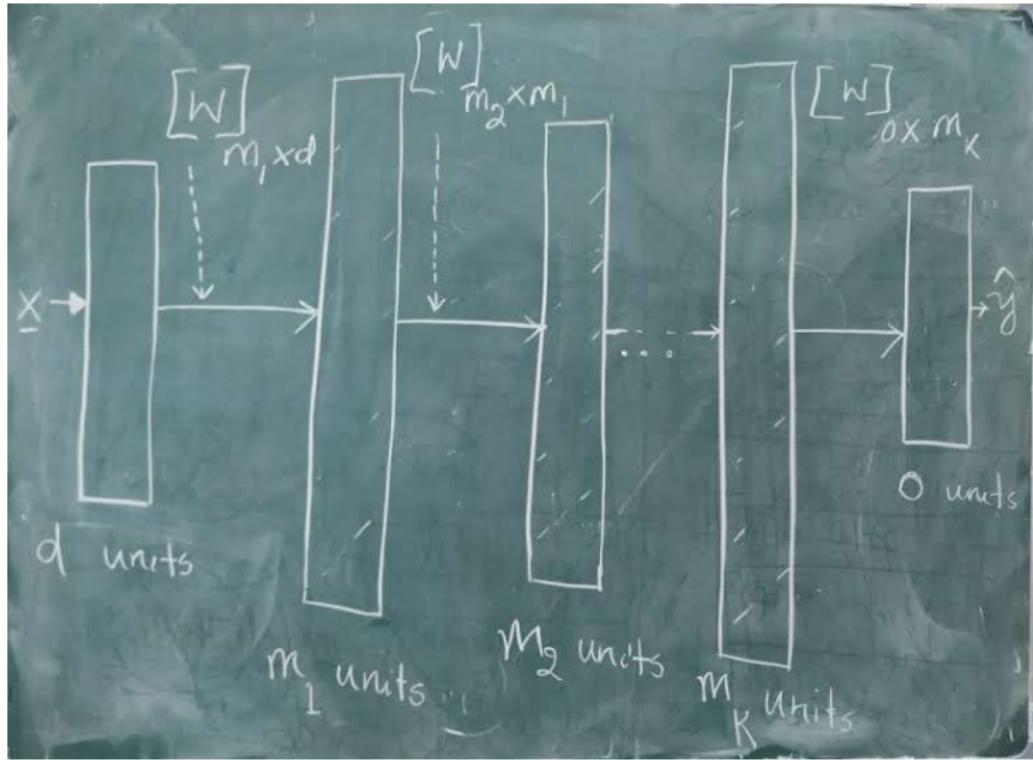
Let's look at a different problem:-



# Multilayer NNs IV

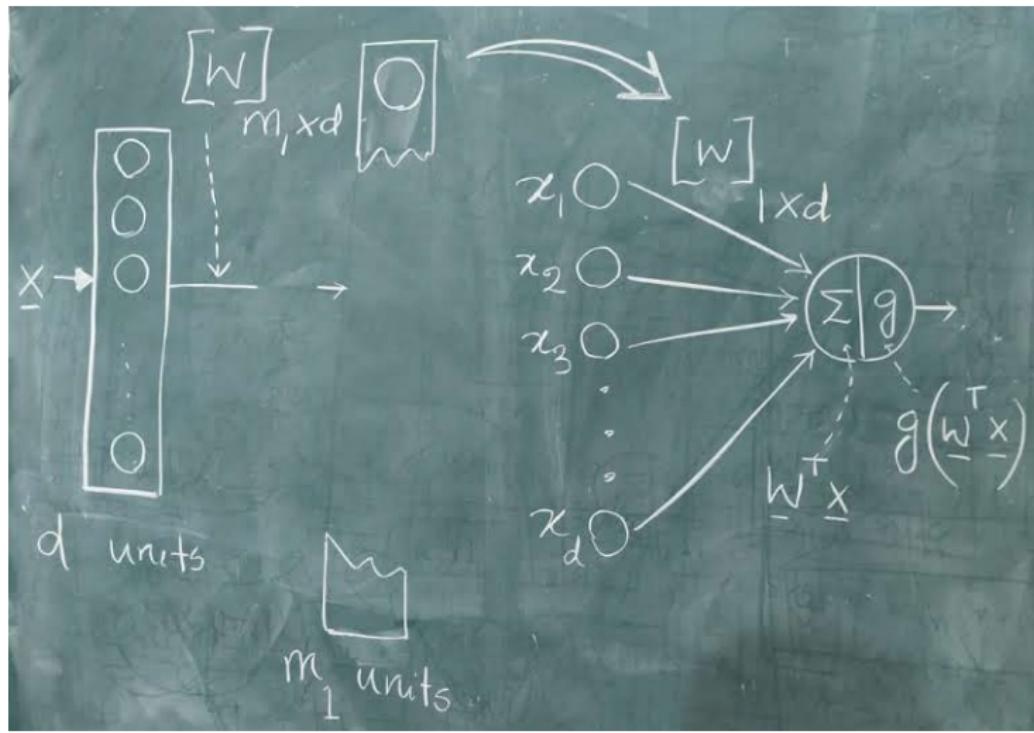
A neural network that is based on this idea is ‘Multilayer Perceptron (MLP)’.

# Multilayer NNs V



# Multilayer NNs VI

Visualising a single neuron in a hidden layer:



# Multilayer NNs VII

Some difficulties concerning the structure of an MLP:

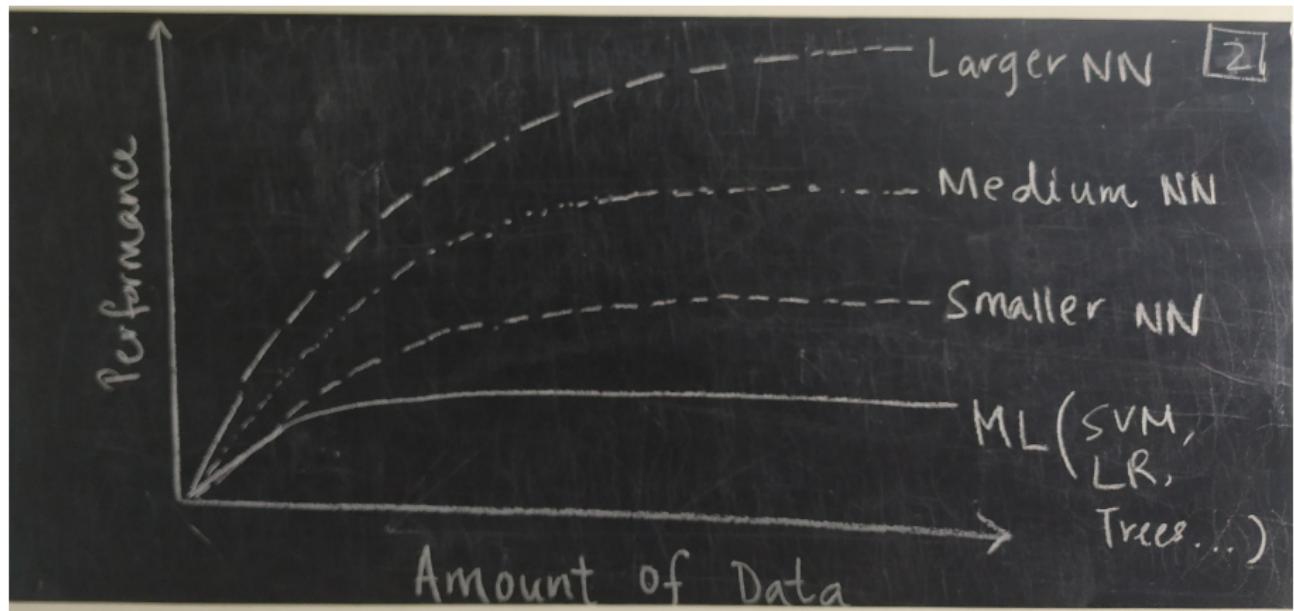
- ① How many hidden layers for a given problem?
- ② How many hidden units per layer?
- ③ What activation function to choose?

# Multilayer NNs VIII

Hidden layers and units:

- ① The hidden layers behave like feature detectors.
- ② Early layers detect low-level features, and later layers detect high-level features.
- ③ Both these parameters can be considered as hyperparameters and they can be tuned using cross-validation method.
- ④ These parameters influence the performance of the model.

# Multilayer NNs IX



# Multilayer NNs X

Activation function:

- No activation for inputs
- Output activation depends on the target domain:

Problem	# of output units	$g(\cdot)$
$y \in \mathbb{R}$	1	linear
$y \in \{0, 1\}$	1	sigmoid
$y \in \{0, 1, \dots, c\}$	$c$	softmax

- Hidden unit activations are non-linear:

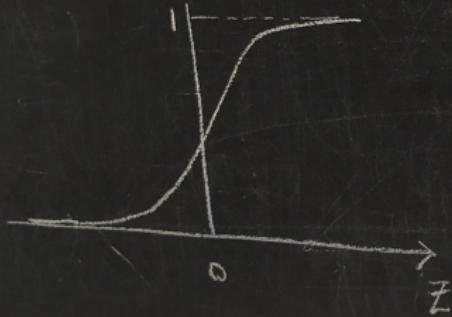
- sigmoid
- tanh
- ReLU and its variants

# Multilayer NNs XI

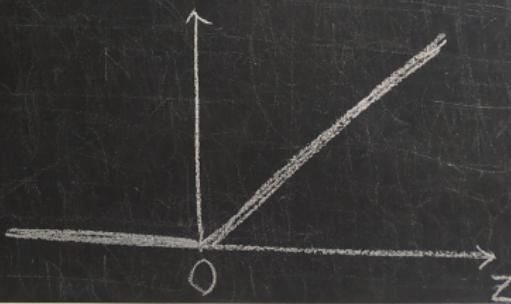
ReLU activation:

Choosing ReLU over Sigmoid.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Rectified Linear Unit  
 $\text{ReLU}(z) = \max(0, z)$



## Multilayer NNs XII

We know,  $Z = \underline{w}^T \underline{x} + b$

- ① as  $Z \rightarrow \infty$ ,  $\therefore \sigma^l(Z) \rightarrow 1(1-1) = 0$   
(Vanishing gradient)
- ②  $\max(0, Z)$  is computationally cheaper  
than exponential operation in  $\sigma(Z)$ .

# Multilayer NNs XIII

Disadvantages of using ReLU over  $\sigma$ :

- ① If many hidden neurons receive '0' in their net input, they don't fire.

"Dying ReLU problem"  $\leftarrow \text{max}(0, -w) = 0$

# Multilayer NNs XIV

Two variants of ReLU function:

- Leaky ReLU:

$$\text{LeakyReLU}(z) = \max(0.01z, z)$$

- Parametric ReLU:

$$\text{pReLU}(z) = \max(\alpha z, z)$$

The parameter  $\alpha$  is learned using an optimiser.

# Multilayer NNs XV

Weights:

The synaptic weights ( $ws$ ) for each layer of an MLP are optimised using gradient descent. The procedure is popularly known as 'Backpropagation'.

Mathematics behind 'BackPropagation' is beyond the scope of this course. However, we will do practical problem solving by implementing a neural network using any popular Python library such as Scikit-learn or Tensorflow-Keras or PyTorch.