Lecture 13 : Neural Networks

- We already looked at two basic parametric models → linear regression
  → logistic regression

- A neural network, in some sense, extends these basic models by stacking multiple copies of these models to construct a hierarchical model

- This hierarchical model can describe more complicated relationships between inputs and outputs than a linear or logistic regression

- Deep learning is a subfield of machine learning that deals with such hierarchical machine learning models

Neural Network Model

- Earlier we introduced the concept of non-linear parametric functions for modelling the relationship between input variables $x_1, \dots, x_p$ and output y

$$\hat{y} = f_{\underline{\theta}}(\underline{x}) = f_{\underline{\theta}}(x_1, x_2, \dots, x_p)$$

the function f is "parameterized" by $\underline{\theta}$

- Such a non-linear function $f_{\underline{\theta}}(\cdot)$ can be created in many ways

- In neural network, the strategy is to use several layers of linear regression models and non-linear activation functions

- In linear regression, we wrote the model prediction as:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_p x_p$$

$$\underline{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$$

input vector

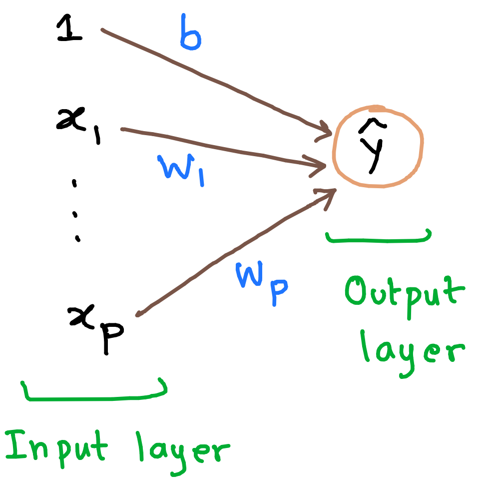
- We start the description of a neural network model with linear regression model

$$\hat{y} = \overset{\theta_0}{b} + \overset{\theta_1}{w_1} x_1 + \overset{\theta_2}{w_2} x_2 + \cdots + \overset{\theta_p}{w_p} x_p$$

$w_1, w_2, \ldots, w_p$ are called the weights and b is the offset

we use this notation because it is more popular in neural networks

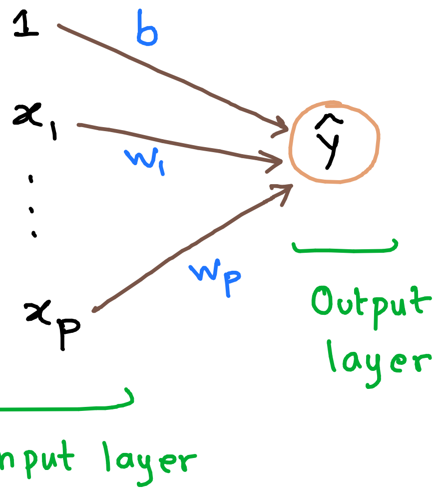
- Graphical representation



$\xrightarrow{w_j}$ – Each link is associated with a parameter

$\hat{y}$ – Output $\hat{y}$ is described as sum of all terms

$$\hat{y} = w_1 x_1 + \cdots + w_p x_p + b$$



$$\underbrace{\hat{y} = w_1 x_1 + \cdots + w_p x_p + b}_{\text{Describes a linear relationship}}$$

- How to describe a non-linear relationship between $\underline{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$ and $\hat{y}$?

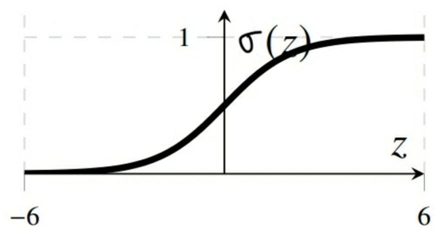
- Use activation function $h : \mathbb{R} \to \mathbb{R}$

$$\underbrace{\hat{y} = \sigma(w_1 x_1 + \cdots + w_p x_p + b)}_{\text{This now becomes a generalized linear model}}$$

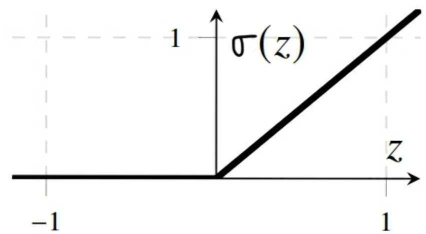
- Common choices of activation functions are:

Logistic (Sigmoid) : $\sigma(z) = \dfrac{1}{1 + e^{z}}$

ReLU : $\sigma(z) = \max(0, z)$ (standard choice)



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



ReLU: $\sigma(z) = \max(0, z)$

$$\underbrace{\hat{y} = \sigma(w_1 x_1 + \cdots + w_p x_p + b)}$$

This now becomes a generalized linear model

- This generalized linear model is very simple
- Cannot describe very complicated relationships between input $\underline{x}$ and output $\hat{y}$

- How can we extend this simple model to increase the flexibility?

Stack several generalized linear models in a sequential construction

Ex:

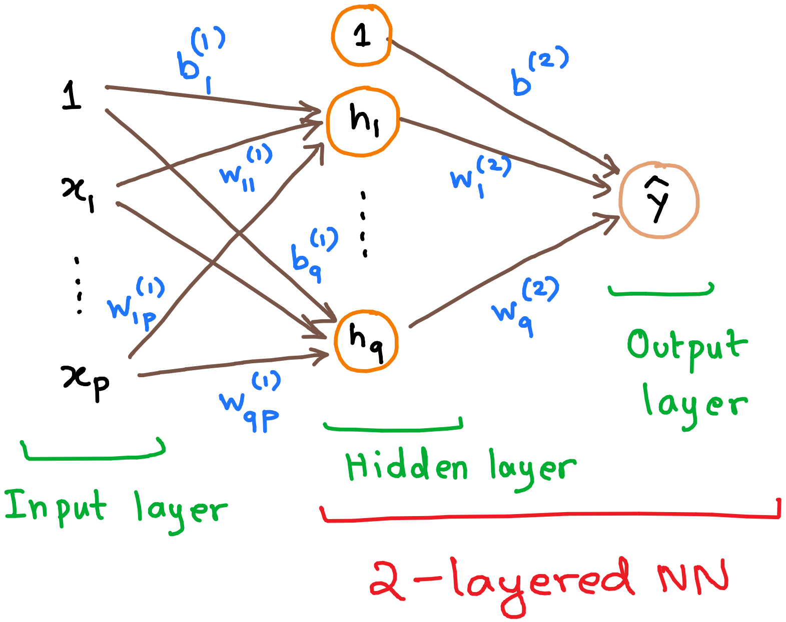
One-layer of stacking (hidden layer)

$$\begin{cases} h_1 = \sigma(w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + \cdots + w_{1p}^{(1)} x_p + b_1^{(1)}) \\ h_2 = \sigma(w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + \cdots + w_{2p}^{(1)} x_p + b_2^{(1)}) \\ \vdots \\ h_q = \sigma(w_{q1}^{(1)} x_1 + w_{q2}^{(1)} x_2 + \cdots + w_{qp}^{(1)} x_p + b_q^{(1)}) \end{cases}$$

Output layer

$$\hat{y} = w_1^{(2)} h_1 + w_2^{(2)} h_2 + \cdots + w_q^{(2)} h_q + b^{(2)}$$

Two-layer Neural Network



- Each link (arrow) is associated with a weight

$$\underset{\text{output node \#}}{w}{}_{i\,j}^{\overset{\text{layer \#}}{(1)}} \quad (j: \text{input node \#})$$

$$x_j \xrightarrow{w_{ij}^{(1)}} h_i$$

- Each circular node is associated with its own bias term b

$$b_i^{(1)} \quad (\text{layer \#}: (1),\ \text{output node \#}: i)$$

$$1 \xrightarrow{b_i^{(1)}} h_i$$

$$h_1 = \sigma\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + \cdots + w_{1p}^{(1)}x_p + b_1^{(1)}\right)$$

$$h_2 = \sigma\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + \cdots + w_{2p}^{(1)}x_p + b_2^{(1)}\right)$$

$$\vdots \qquad \vdots$$

$$h_q = \sigma\left(w_{q1}^{(1)}x_1 + w_{q2}^{(1)}x_2 + \cdots + w_{qp}^{(1)}x_p + b_q^{(1)}\right)$$

$$\hat{y} = w_1^{(2)}h_1 + w_2^{(2)}h_2 + \cdots + w_q^{(2)}h_q + b^{(2)}$$

Vectorized Representation

- The 2-layered neural network can be more compactly written using matrix notation:

$$\underline{\underline{W}}^{(1)} = \begin{bmatrix} w_{11}^{(1)} & \cdots & w_{1p}^{(1)} \\ \vdots & & \vdots \\ w_{q1}^{(1)} & \cdots & w_{qp}^{(1)} \end{bmatrix}_{q\times p}, \quad \underline{b}^{(1)} = \begin{bmatrix} b_1^{(1)} \\ \vdots \\ b_q^{(1)} \end{bmatrix}_{q\times 1}, \quad \underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}_{p\times 1}$$

$$\underline{\underline{W}}^{(2)} = \begin{bmatrix} w_1^{(2)} & \cdots & w_q^{(2)} \end{bmatrix}_{1\times q}, \quad \underline{b}^{(2)} = \begin{bmatrix} b^{(2)} \end{bmatrix}_{1\times 1}, \quad \underline{h} = \begin{bmatrix} h_1 \\ \vdots \\ h_q \end{bmatrix}_{q\times 1}$$

Compact representation

$$\underset{q\times 1}{\underline{h}} = \sigma\big(\underbrace{\underline{\underline{W}}^{(1)}\underline{x} + \underline{b}^{(1)}}_{q\times 1}\big)$$

$$\underset{1\times 1}{\hat{y}} = \underset{1\times q}{\underline{\underline{W}}^{(2)}}\,\underset{q\times 1}{\underline{h}} + \underset{1\times 1}{\underline{b}^{(2)}}$$

$$\underline{\Theta} = \begin{bmatrix} \mathrm{vec}(\underline{\underline{W}}^{(1)}) \\ \underline{b}_1^{(1)} \\ \mathrm{vec}(\underline{\underline{W}}^{(2)}) \\ b^{(2)} \end{bmatrix}$$

$$\hat{y} = f_{\underline{\Theta}}(\underline{x})$$

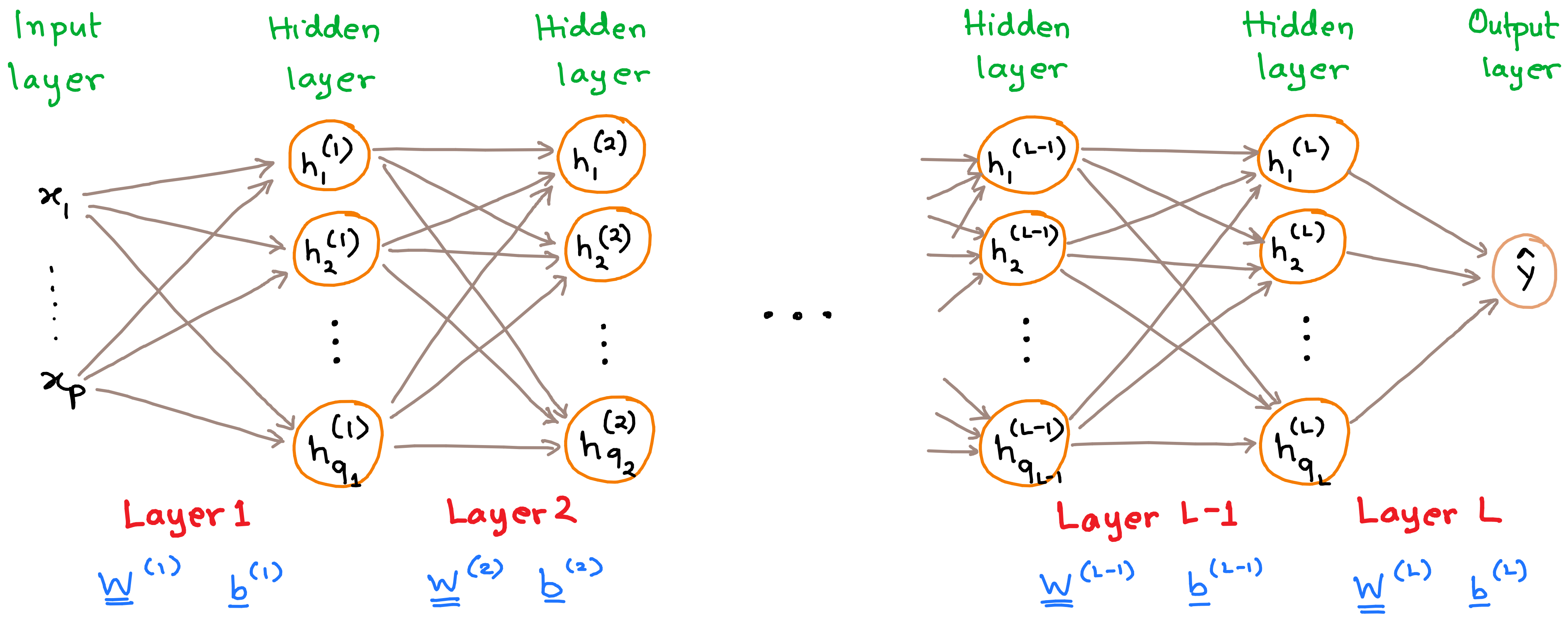
Note: The activation function $\sigma(\cdot)$ operates element-wise

Deep neural network

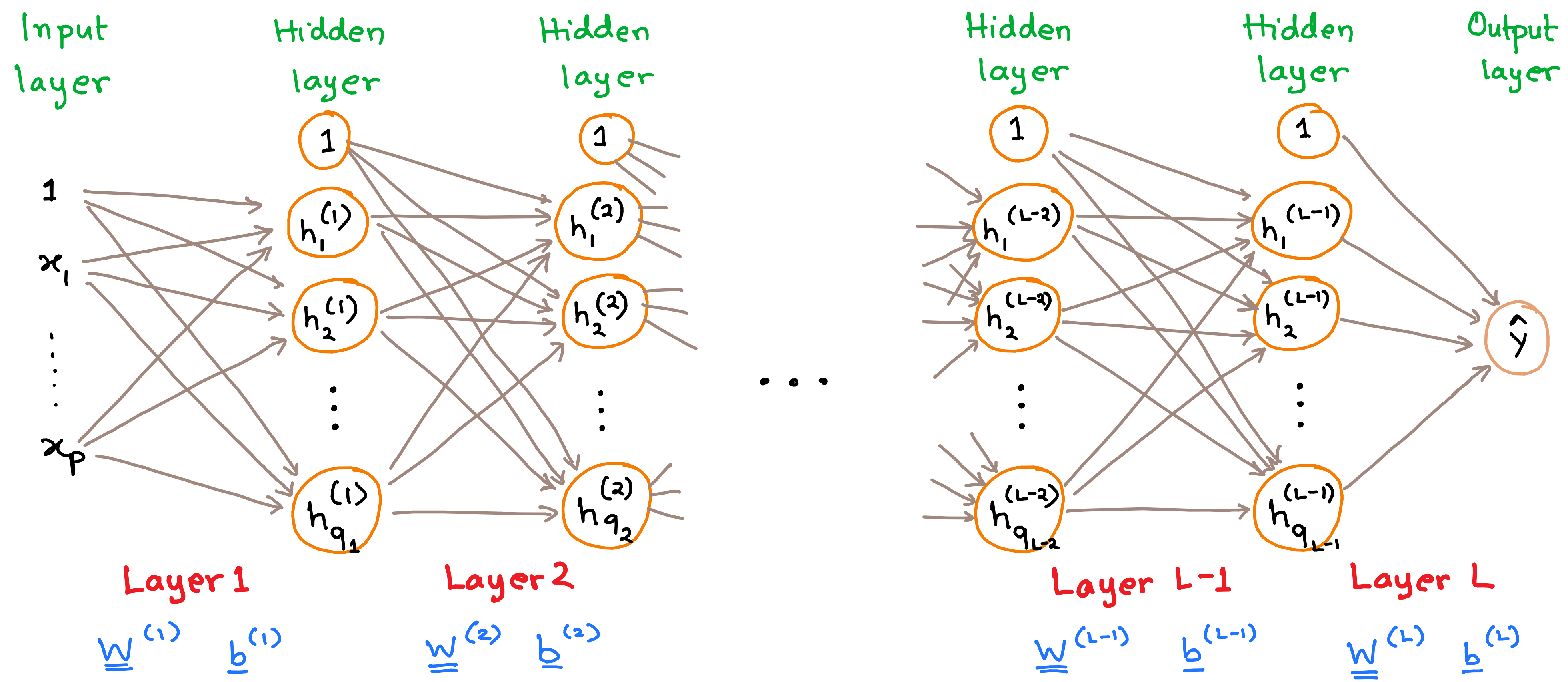
- The 2-layered neural network is called shallow NN (because it has one hidden layer)

- The real flexibility of neural network comes when we have more than one hidden layer.

- Stacking multiple hidden layers leads to a DEEP neural network



Deep Neural Network (Feed-forward Neural Network – FNN)



Mathematical representation of FNN:

$$\underline{h}^{(1)} = \sigma\left(\underline{\underline{W}}^{(1)} \underline{x} + \underline{b}^{(1)}\right)$$

$$\underline{h}^{(2)} = \sigma\left(\underline{\underline{W}}^{(2)} \underline{h}^{(1)} + \underline{b}^{(2)}\right)$$

$$\vdots$$

$$\underline{h}^{(L)} = \sigma\left(\underline{\underline{W}}^{(L-1)} \underline{h}^{(L-1)} + \underline{b}^{(L-1)}\right)$$

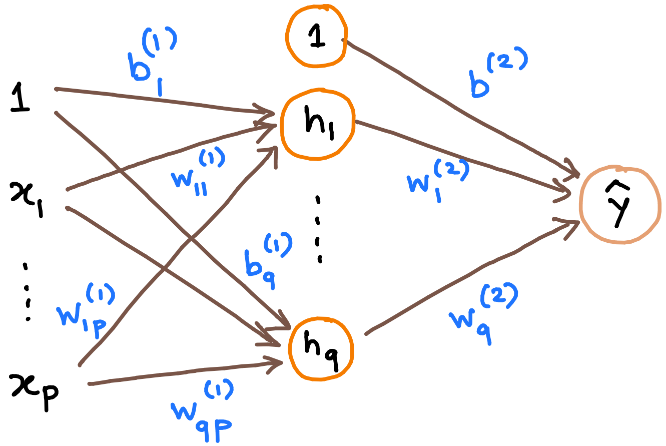
$$\hat{y} = \underline{\underline{W}}^{(L)} \underline{h}^{(L-1)} + \underline{b}^{(L)}$$

Vectorization over datapoints

– During training, the neural network model is used to compute the predicted output for several input points $\{\underline{x}_i\}_{i=1}^{N}$

Superscript (ℓ) → will denote layer #

– The 2-layer neural network



column vector

$$\underset{q\times 1}{\underline{h}} = \sigma\big(\underbrace{\underline{\underline{W}}^{(1)}\underline{x} + \underline{b}^{(1)}}_{q\times 1}\big)$$

$$\underset{1\times 1}{\hat{y}} = \underset{1\times q}{\underline{\underline{W}}^{(2)}}\ \underset{q\times 1}{\underline{h}} + \underset{1\times 1}{\underline{b}^{(2)}}$$

→ row vector

$$\underset{1\times q}{\underline{h}_i^T} = \sigma\Big(\underset{1\times p}{\underline{x}_i^T}\ \underset{p\times q}{\underline{\underline{W}}^{(1)T}} + \underset{1\times q}{\underline{b}^{(1)T}}\Big)$$

$$\hat{y}_i = \underset{1\times q}{\underline{h}_i^T}\ \underset{q\times 1}{\underline{\underline{W}}^{(2)T}} + \underset{1\times 1}{\underline{b}^{(2)T}}$$

$i = 1, 2, \dots, n$

Transpose

Vectorization over datapoints

- During training, the neural network model is used to compute the predicted output for several input points $\{ y_i , \underline{x}_i \}_{i=1}^{N}$

- The 2-layer neural network

column vector

$$\underset{q\times 1}{\underline{h}} = \sigma\big(\underbrace{\underline{\underline{W}}^{(1)} \underline{x} + \underline{b}^{(1)}}_{q\times 1}\big)$$

$$\underset{1\times 1}{\hat{y}} = \underset{1\times q}{\underline{\underline{W}}^{(2)}} \underset{q\times 1}{\underline{h}} + \underset{1\times 1}{b^{(2)}}$$

$\longrightarrow$

row vector

$$\underset{1\times q}{\underline{h}_i^T} = \sigma\left(\underset{1\times p}{\underline{x}_i^T} \underset{p\times q}{\underline{\underline{W}}^{(1)^T}} + \underset{1\times q}{\underline{b}^{(1)^T}} \right)$$

$$\underset{1\times 1}{\hat{y}_i} = \underset{1\times q}{\underline{h}_i^T} \underset{q\times 1}{\underline{\underline{W}}^{(2)^T}} + \underset{1\times 1}{\underline{b}^{(2)^T}}$$

$i = 1, 2, \ldots, n$

Transposed

- Similar to linear regression, we stack all data points in matrices

$$\underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}_{N\times 1}, \quad \underline{\underline{X}} = \begin{bmatrix} \underline{x}_1^T \\ \vdots \\ x_N^T \end{bmatrix}_{N\times p}, \quad \hat{\underline{y}} = \begin{bmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_N \end{bmatrix}_{N\times 1}, \quad \underline{\underline{H}} = \begin{bmatrix} \underline{h}_1^T \\ \vdots \\ h_N^T \end{bmatrix}_{N\times q}$$

each row represents a training data point

$$\underset{N\times q}{\underline{\underline{H}}} = \sigma\left(\underset{N\times p}{\underline{\underline{X}}} \underset{p\times q}{\underline{\underline{W}}^{(1)^T}} + \underset{1\times q}{\underline{b}^{(1)^T}} \right)$$

$$\underset{N\times 1}{\hat{\underline{y}}} = \underset{N\times q}{\underline{\underline{H}}} \underset{q\times 1}{\underline{\underline{W}}^{(2)^T}} + \underset{1\times 1}{\underline{b}^{(2)^T}}$$

added to each row

$$\boxed{\begin{aligned} \underline{\underline{H}} &= \sigma\left(\underline{\underline{X}}\ \underline{\underline{W}}^{(1)^T} + \underline{b}^{(1)^T} \right) \\ \hat{\underline{y}} &= \underline{\underline{H}}\ \underline{\underline{W}}^{(2)^T} + \underline{b}^{(2)^T} \end{aligned}}$$

- These vectorized equations would be used in implementation in PyTorch, Tensorflow, etc.

$$\Downarrow$$

$$\underline{\underline{H}} = \sigma\left(\underline{\underline{X}}\ \tilde{\underline{\underline{W}}}^{(1)} + \tilde{\underline{b}}^{(1)} \right)$$

$$\hat{\underline{y}} = \underline{\underline{H}}\ \tilde{\underline{\underline{W}}}^{(2)} + \tilde{\underline{b}}^{(2)}$$

- During programming, you may consider using the transposed versions of $\underline{\underline{W}}$ and $\underline{b}$ as the weight matrix and bias vectors to avoid transposing them in each layer

$$\tilde{\underline{\underline{W}}}^{(1)} \longleftarrow \underline{\underline{W}}^{(1)^T} \qquad \tilde{\underline{b}}^{(1)} \longleftarrow \underline{b}^{(1)^T}$$

$$\tilde{\underline{\underline{W}}}^{(2)} \longleftarrow \underline{\underline{W}}^{(2)^T} \qquad \tilde{\underline{b}}^{(2)} \longleftarrow \underline{b}^{(2)^T}$$

Neural Networks for Classification

- How did we extend linear regression to logistic regression?
  - By applying logistic (or sigmoid) function to the output of linear regression in case of binary classification
  - For multi-class classification (with $y \in \{1, 2, \ldots, M\}$ classes), we used the softmax function

$$\text{softmax}(\underline{z}) = \frac{1}{\sum_{j=1}^{M} e^{z_j}} \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \vdots \\ e^{z_M} \end{bmatrix}$$

- The softmax function now becomes an additional activation function, acting on the final layer of the neural network

$$\underline{h}^{(1)} = \sigma\left(\underline{\underline{W}}^{(1)} \underline{x} + \underline{b}^{(1)}\right)$$

$$\vdots$$

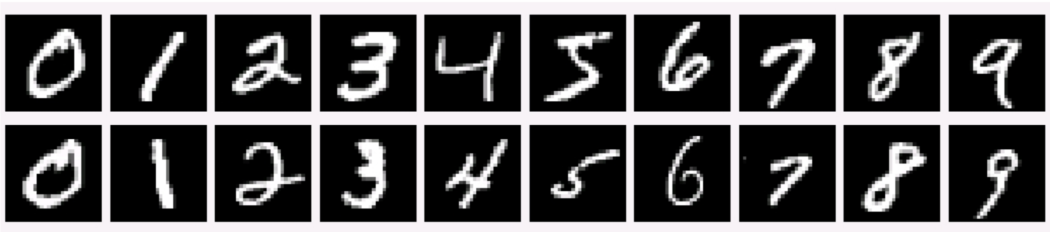
$$\underline{h}^{(L-1)} = \sigma\left(\underline{\underline{W}}^{(L-1)} \underline{h}^{(L-2)} + \underline{b}^{(L-1)}\right)$$

$$\underset{M \times 1}{\underline{z}} = \underline{\underline{W}}^{(L)} \underline{h}^{(L-1)} + \underline{b}^{(L)}$$

$$\underset{M \times 1}{\underline{g}} = \text{softmax}(\underline{z})$$

MNIST example : Classification of handwritten digits

- Dataset has 60000 training images
- " " 10000 test/validation images
- Each data point consists of a 28x28 pixel grayscale image of a handwritten digit
- Each image is also labelled with the digit 0, 1, 2, ..., 9 that it depicts
- Each pixel intensity has been normalized between [0, 1]

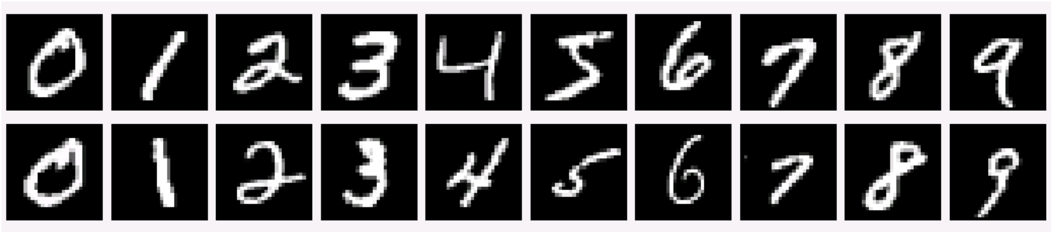


- Consider image as input $\underline{x} = [x_1 \; x_2 \; .... \; x_p]^T$
  - $p = 28 \times 28 = 784$ input variables (flattened out)
  - Each x_j corresponds to a pixel in the image and represents its intensity

$x_j = 0 \longrightarrow$ black pixel

$x_j = 1 \longrightarrow$ white pixel ,

Anything between 0 and 1 is grey pixel



- Consider image as input $\underline{x} = [x_1 \; x_2 \; .... \; x_p]^T$
  - $p = 28 \times 28 = 784$ input variables (flattened out)
  - Each x_j corresponds to a pixel in the image and represents its intensity

$x_j = 0 \longrightarrow$ black pixel

$x_j = 1 \longrightarrow$ white pixel,

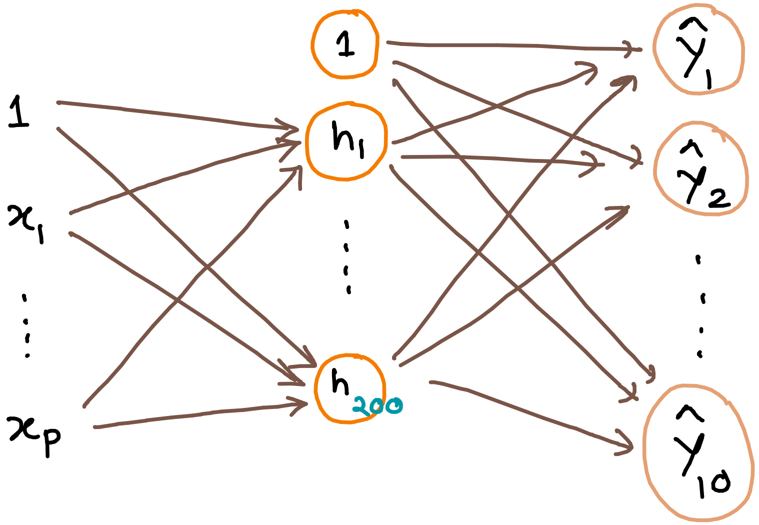
Anything between 0 and 1 is grey pixel

Using a 2-layer NN

Consider 200 hidden units

$$\underbrace{\underline{\underline{H}}}_{60000 \times 200} = \sigma\Big(\underbrace{\underline{\underline{X}}}_{60000 \times 784} \; \underbrace{\underline{\underline{W}}^{(1)^T}}_{784 \times 200} + \underbrace{\underline{b}^{(1)^T}}_{1 \times 200} \Big)$$

$$\underbrace{\hat{\underline{Y}}}_{60000 \times 10} = \underline{\underline{H}} \; \underbrace{\underline{\underline{W}}^{(2)^T}}_{200 \times 10} + \underbrace{\underline{b}^{(2)^T}}_{1 \times 10}$$



Total parameters $= 784 \times 200 + 200 + 200 \times 10 + 10 = 159010$!!