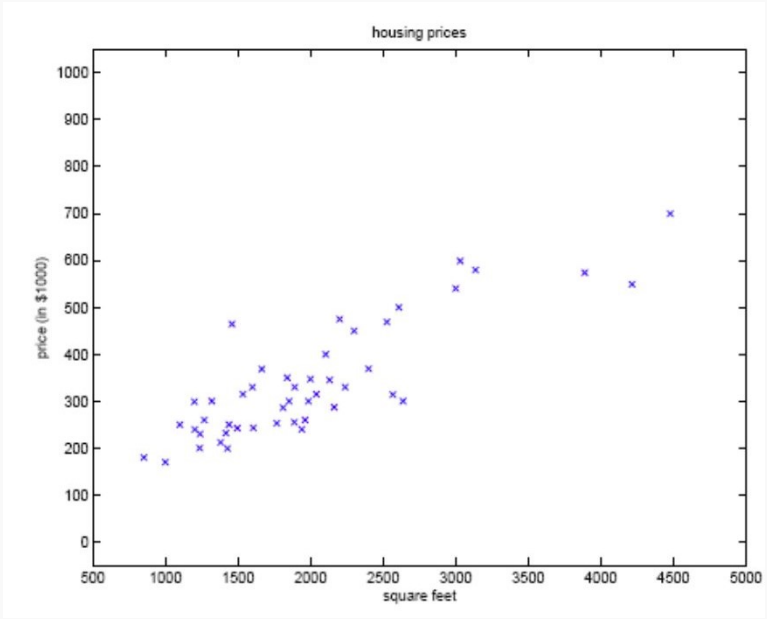
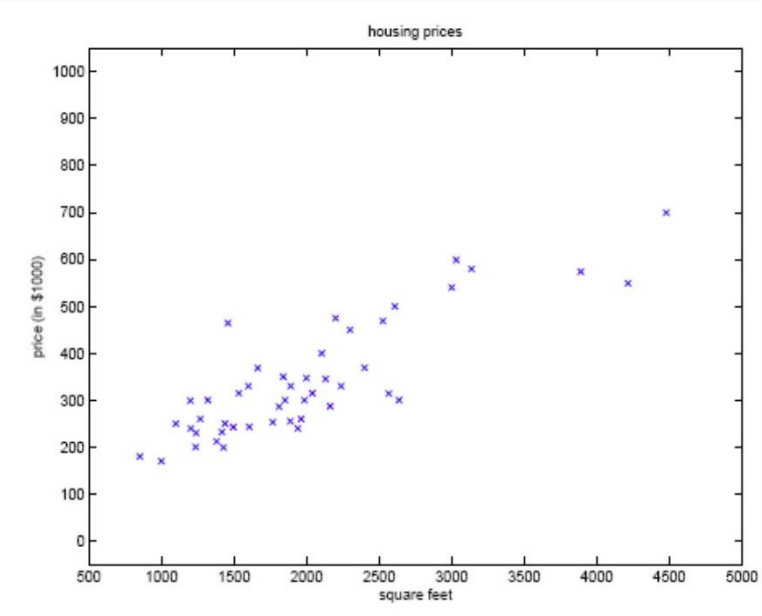
Applied Deep Learning

Lecture 2

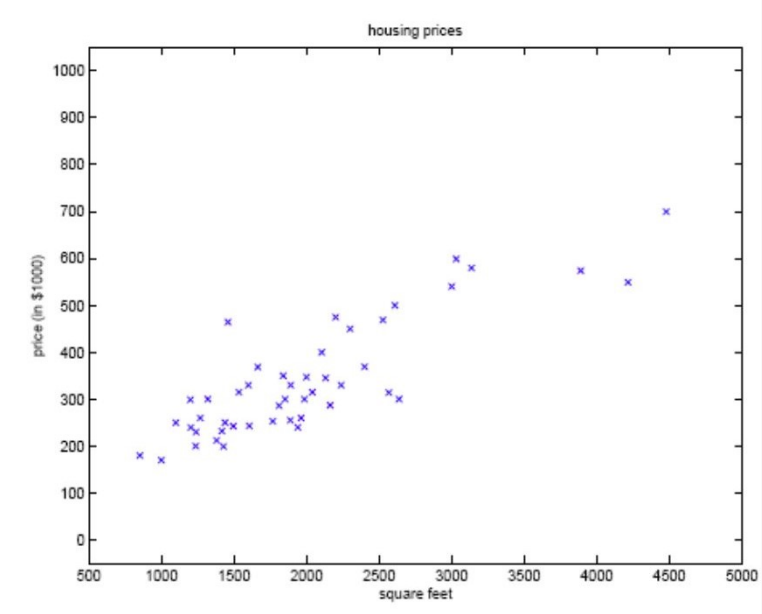
Ganapathy Krishnamurthi

IIT-Madras

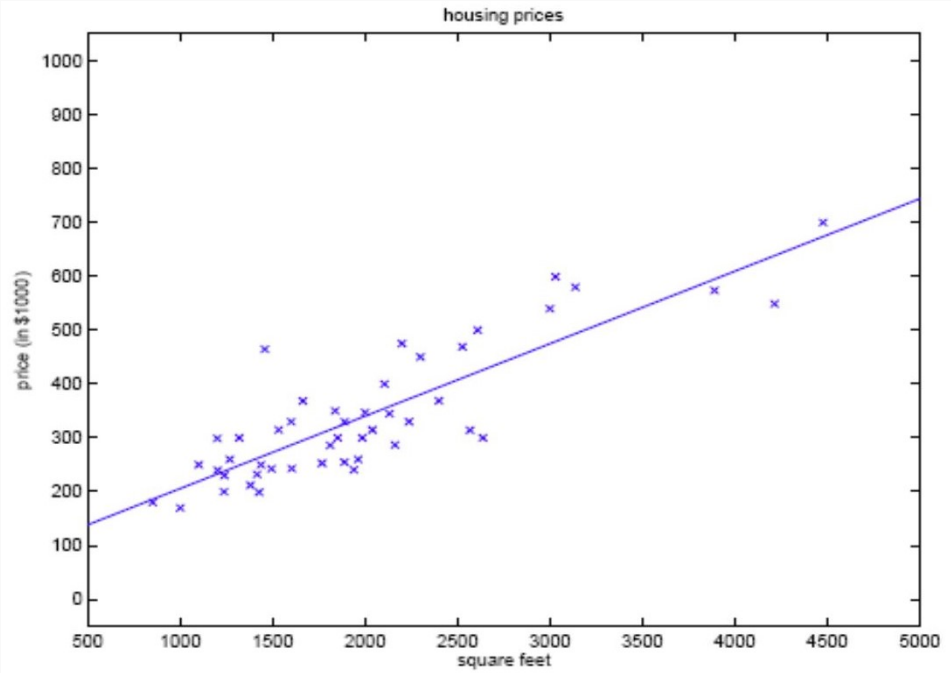

housing prices
price (in $1000)
square feet
0
100
200
300
400
500
600
700
800
900
1000
500
1000
1500
2000
2500
3000
3500
4000
4500
5000



Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
$\vdots$	$\vdots$

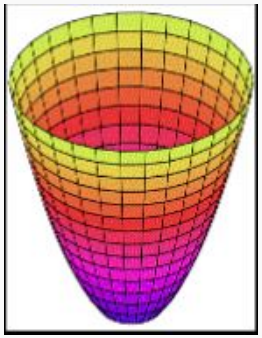
Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

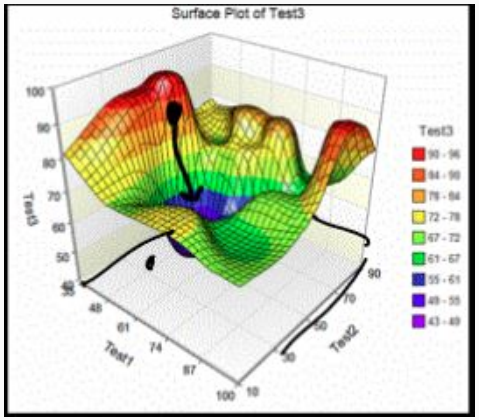
$$J = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2$$

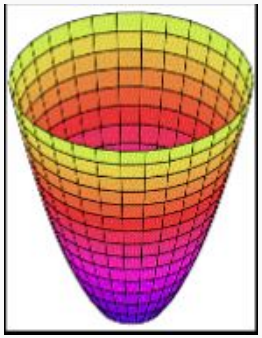
$$= \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - X_w)^2$$

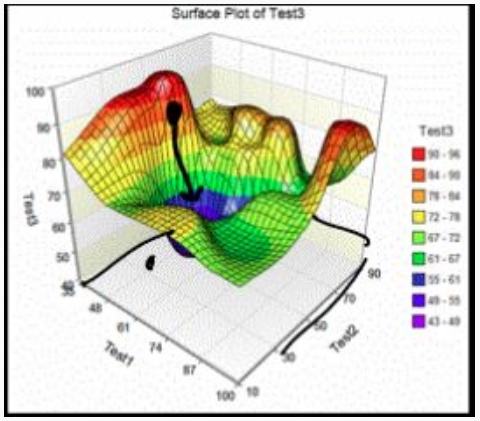
X_w is the input matrix.

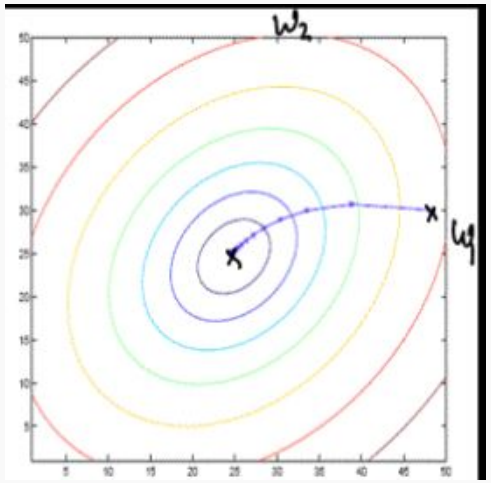
Solve for $\nabla_w J = 0 \rightarrow$ Gives linear system of equations for w, which is normal equation.

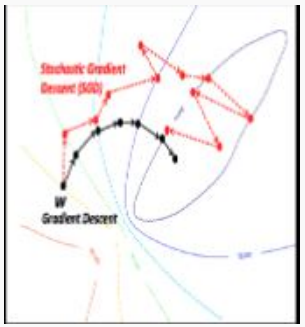


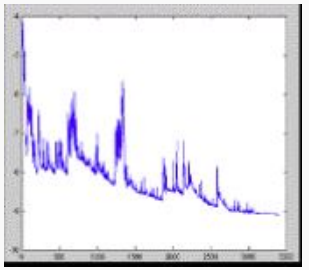

Surface Plot of Test3
Test3
100
90
80
70
60
50
Test1
48
61
74
87
100
Test2
10
30
50
70
90
Test3
90 - 96
84 - 90
78 - 84
72 - 78
67 - 72
61 - 67
55 - 61
49 - 55
43 - 49

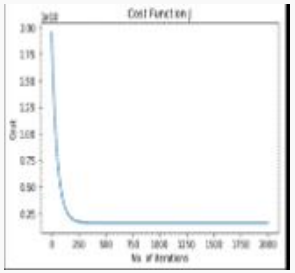



Surface Plot of Test3
Test3
Test1
Test2


w_2
w_1


Stochastic Gradient Descent (SGD)
W
Gradient Descent




Cost Function J
Cost
No. of Iterations

Recap - Linear Regression

kaggle → housing prices dataset

$\vec{x} = [1710, 3]$

$= [x_1^{(i)}, x_2^{(i)}]$ → index of the sample

↳ component index / feature index

output vector

# GrLivArea	# BedroomA...	# SalePrice
1710	3	208500
1262	3	181500
1786	3	223500
1717	3	140000
2198	4	250000
1362	1	143000
1694	3	307000
2090	3	200000
1774	2	129900
1077	2	118000

vector

m → samples or no of data points

n → number of feature

Living area and bedroom in table Input $\vec{x}$ and Price is the output (y)

$$\vec{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}) \rightarrow y^{(i)}$$

feature vector

Target

Recap - Linear Regression

# GrLivArea	# BedroomA...	# SalePrice
1710	3	208500
1262	3	181500
1786	3	223500
1717	3	140000
2198	4	250000
1362	1	143000
1694	3	307000
2090	3	200000
1774	2	129900
1077	2	118000

Living area and bedroom in table Input $\overrightarrow{x}$ and Price is the output (y)

Input	Output	Model
$1, x_1^{(1)}, x_2^{(1)}, x_3^{(1)}, \ldots x_n^{(1)}$	$y^{(1)}$	$\hat{y}^{(1)}$
.	.	.
.	.	.
.	.	.
$1, x_1^{(m)}, x_2^{(m)}, x_3^{(m)}, \ldots x_n^{(m)}$	$y^{(m)}$	$\hat{y}^{(m)}$
$X : m \times n$	Ground Truth ($\overrightarrow{y}$)	($\overrightarrow{\hat{y}}$)

Where m is the examples and n is the features

$$\overrightarrow{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}) \rightarrow y^{(i)}$$

input → target output

Example $\rightarrow [\vec{x}^{(i)}, y^{(i)}]$

$\hat{y} = b + wx \rightarrow$ Single feature

$b \rightarrow w_0$, Which is bias unit

$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} \ldots + w_n x_n^{(i)} \rightarrow$ General case n features

$= w^T x$

$w_0 x_0$

$x_0 = 1$

$$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ . \\ . \\ . \\ w_n \end{bmatrix}$$

$$\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \ldots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \ldots & x_n^{(2)} \\ \vdots & & & & \\ 1 & x_1^{(m)} & x_2^{(m)} & \ldots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

m rows

$n+1$ columns

$(n+1) \times 1$

$$\underset{m \times 1}{\vec{\hat{y}}} = \underset{m \times (n+1)}{X} \; \underset{(n+1) \times 1}{w}$$

The cost function $J = \frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$. This is mean square error (MSE).

Find w that minimizes J

↳ Least squares

Set $\nabla_w J = 0$

$$\frac{\partial}{\partial w}\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \hat{y}^{(i)})^2 = 0$$

$$\frac{\partial}{\partial w}\left[\frac{1}{m}\left\|\vec{y} - \vec{\hat{y}}\right\|_2^2\right] = 0$$

$$\frac{\partial}{\partial w}\left[[y - X_w]^T[y - X_w]\right] = 0$$

$$\frac{\partial}{\partial w}\left[y^T y - w^T x^T y - y^T X_w + w^T X^T X_w\right] = 0$$

$$\boxed{-2X^T y + 2X^T X_w = 0}$$

$$\boxed{w = (X^T X)^{-1} X^T y}$$

Normal equation

$$\frac{\partial (V^T a)}{\partial \vec{V}} = \vec{a}$$

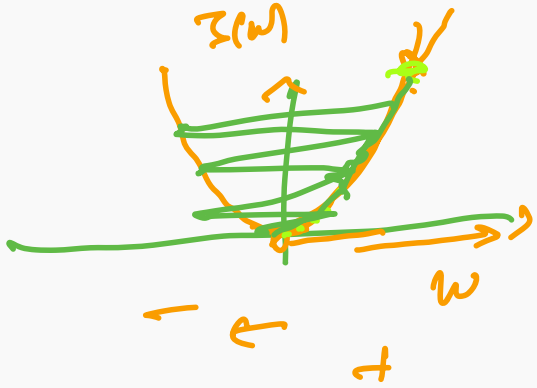
$$\frac{\partial (V^T A_v)}{\partial \vec{V}} = 2A_v$$

Matrix Calculus

Better to use Gradient Descent

Algorithm

1. Initialize (guess) $\vec{w}$, set α, Which is learning rate.
2. Iterate $\vec{W} = \vec{W} - \alpha \nabla_w J$
3. Check for convergence / stopping criteria.
   If not met, go to 2



For $\nabla_w J$

$$w_j := w_j - \alpha \frac{\partial J}{\partial w_j}$$

hyper parameter optimization

$\alpha \to \frac{\alpha}{\text{iter}}$

every N iterations

$$J = \frac{1}{m}\sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

$$\frac{\partial J}{\partial w} = \frac{1}{m}\sum_i -2(y^{(i)} - \hat{y}^{(i)})\frac{\partial \hat{y}^{(i)}}{\partial w}$$

$$\left(\frac{\partial J}{\partial w}\right)_j = \frac{1}{m}\sum_{i=1}^{m} -2(y^{(i)} - \hat{y}^{(i)})\frac{\partial \hat{y}^{(i)}}{\partial w_j}$$

$$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + \ldots + w_n x_n^{(i)}$$

$$\frac{\partial \hat{y}^{(i)}}{\partial w_j} = x_j^{(i)} \rightarrow j^{th} \text{ feature of } i^{th} \text{ example}$$

$$\frac{\partial J}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m} -2(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$

$$\delta_i = Error^{(i)}\, Input$$

$$\frac{\partial J}{\partial \vec{w}} = \frac{1}{m}\sum_{i=1}^{m} -2\delta^{(i)}\vec{x}^{(i)}$$

$$= \frac{1}{m} Sum\left[-2Error^{(i)} * Feature^{(i)}\right]$$

$\frac{\partial J}{\partial \hat{y}}\,\frac{\partial \hat{y}}{\partial w}$

$x^{(1)} \rightarrow$

$x^{(2)}$

$x^{(m)}$

$$\frac{\partial J}{\partial w} = \begin{bmatrix} \frac{\partial J}{\partial w_0} \\ \frac{\partial J}{\partial w_1} \\ \vdots \\ \frac{\partial J}{\partial w_n} \end{bmatrix}$$

SGD → 1 pt

Batch → All

Mini-Batch → subset

This is the batch- gradient descent.

Mini-Batch Gradient Descent

$\sum_{mini-batch}[-2Error * Feature]$

Stochastic Gradient Descent (SGD) $\rightarrow$ Mini-batch size $= 1$.

General Algorithm

1. Initialize $\vec{w}, \alpha$
2. Split data into mini-batches, Randomly shuffle.
3. $\vec{w} = \vec{w} - \sum_{mini-batch}(\frac{2Error * \vec{Feature}}{m})$ - Once you see the whole data $\rightarrow$ epoch.
4. Until stopping

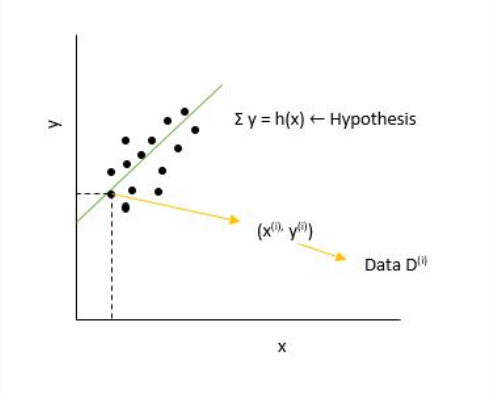
Topics for today

1. Bias/variance Trade off
2. Regularization
3. Probabilistic Interpretation $\rightarrow$ MAP/MLE
4. Logistic Regression

$\hat{y} = h(x) to$ Hypothesis function.

$\hat{y} = w^T x \rightarrow$ Linear regression.

Least Square Cost function: $\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} - h(x^{(i)}))^2 = J$



$$y^{(i)} = h(x^{(i)}) + \delta^{(i)}$$

Where $h(x^{(i)})$ is $\hat{y}$ and $\delta^{(i)}$ is $P(y|x)$.

Probabilistic Interpretation

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where $P(h|D)$ is the Posterior Probability which is the probability of hypothesis h Given data is D.
$P(D|h)$ is the likelihood (How likely is Data D for given hypothesis h).
$P(h)$ is the prior probability (Before we know anything about the data)

MAP

MAP: Maximum a posteriori probability ($h \epsilon H$ (Space/ set of all hypothesis)) says the best hypothesis is the one which maximizes $P(h|D)$ mathematically,

$$h_{MAP} = \underset{h \epsilon H}{argmax} P(h|D)$$

From Bayes' Theorem, $h_{MAP} = \underset{h \epsilon H}{argmax} \frac{P(D|h)P(h)}{P(D)}$.

$P(D)$ is independent of h.

If $P(h_i)$ are all equal (that is, equally likely priors), then

$$h_{MAP} = \underset{h \epsilon H}{argmax} P(D|h)$$

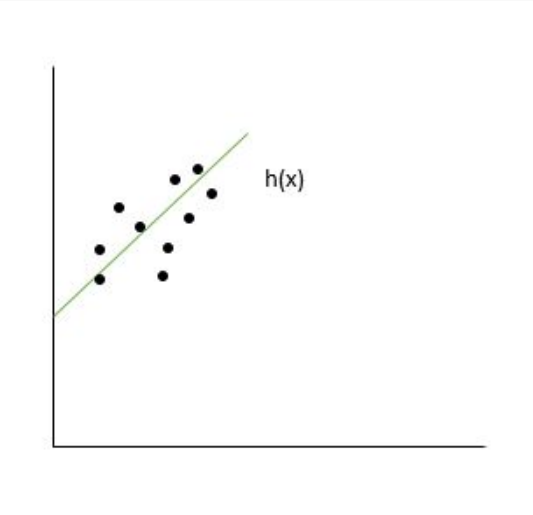
i.e., For uniform priors,

MAP

$$h_{MAP} = h_{max-likelihood}$$

Now, consider a general regression problem:

$$\underset{Data}{y^{(i)}} = \underset{Hypothesis}{h(x^{(i)})} + \underset{Error}{\delta^{(i)}}$$



MAP

Assume:

- Independent $\delta^{(i)}$. That is, all training examples, are independent.
- Normally distributed $\delta^{(i)}$: $\delta \sim N(0, \sigma^2)$ (Mean $= 0$, Variance $= \sigma^2$)

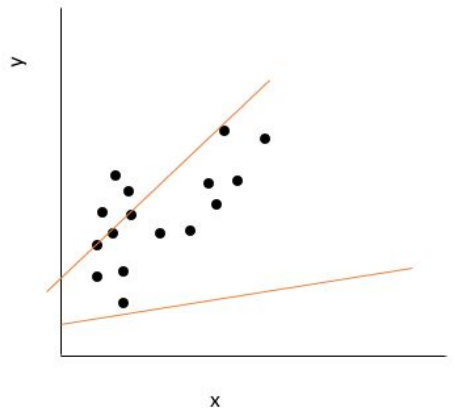
$$P(\delta^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}} exp\{\frac{-\delta^2}{2\sigma^2}\}$$

Now, the regression problem

Hypothesis Space (H) - Set of functions that the learning algorithm is allowed to choose from for a solution.
Ex: Linear Regression has H = Set of functions linear in input feaures.

$$h \epsilon H : S \cdot t \ \underset{h \epsilon H}{argmin} \underbrace{\|y - \hat{y}\|^2}_{Least\ square\ formulation}$$

Least squares from Max-Likelihood



$$y = h(x) + \delta$$

Assumptions:

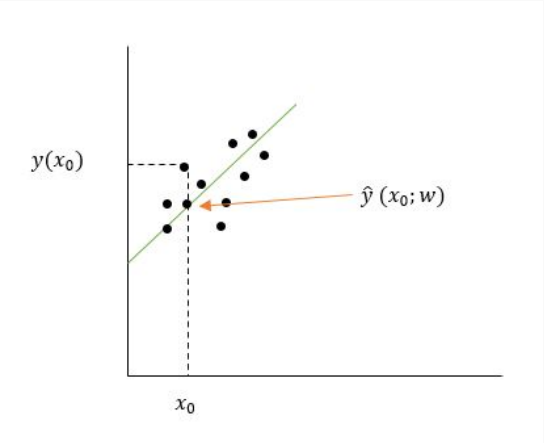
- I.I.D δ
- δ is normally distributed. $\delta \sim N(0, \sigma^2)$

Question: Given a particular fit, how likely is the data?

$$P(\overrightarrow{y}|\overrightarrow{x}; w, \sigma^2)$$

$$= P(y^{(1)}, y^{(2)}, ...y^{(m)}|\overrightarrow{x}; w)$$

$$\Pi_{i=1}^{m} P(y^{(i)}|x^{(i)}; w, \sigma^2)$$



$$\therefore Likelihood = \Pi_{i=1}^{m} P(y^{(i)}|x^{(i)}; w, \sigma^2)$$

$$\underset{\log(P(D|h))}{\log Likelihood} = \sum_{i=1}^{m} \log(P(y^{(i)}|x^{(i)}; w))$$

$$N(y^{(i)};\hat{y}(x^{(i)}),\sigma^2)$$

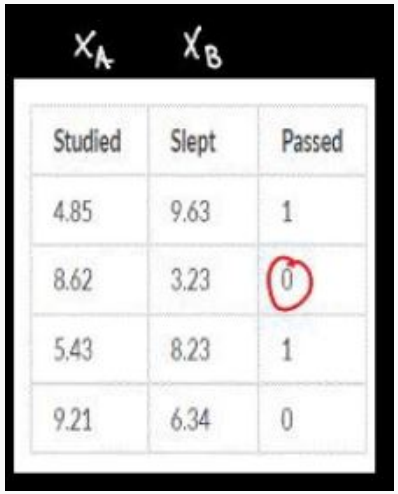
$$= \frac{1}{\sqrt{2\pi\sigma^2}} exp\left[\frac{-(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}\right]$$

$$= \frac{-m}{2}\log 2\pi - m\log\sigma - \sum_{i=1}^{m}\frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$$

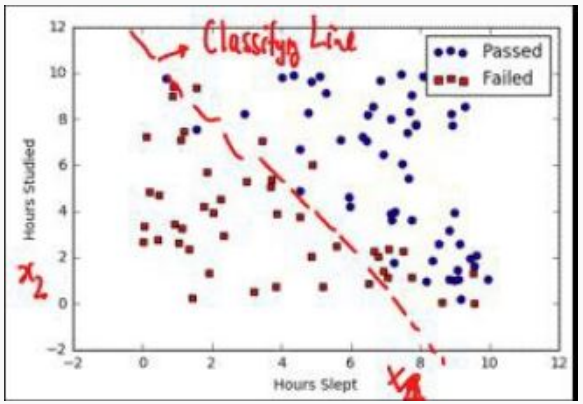
$$\therefore argmax \log(P(D|h)) = \underset{w}{argmin} \sum_{i=1}^{m}\frac{(y^{(i)} - \hat{y}^{(i)})^2}{2\sigma^2}$$

$$W_{MLE} = W_{Least-square}$$

We can also obtain $W_{Regularized\ least\ squares}$ by finding W_{MAP} by using an appropriate prior.



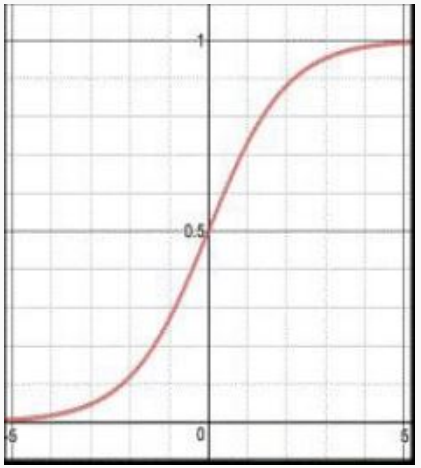
Studied	Slept	Passed
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0



Classify Haplers in two steps.

1. Finding classifying line (boundary)
2. Assigning a label (value) to each point

Sigmoid function



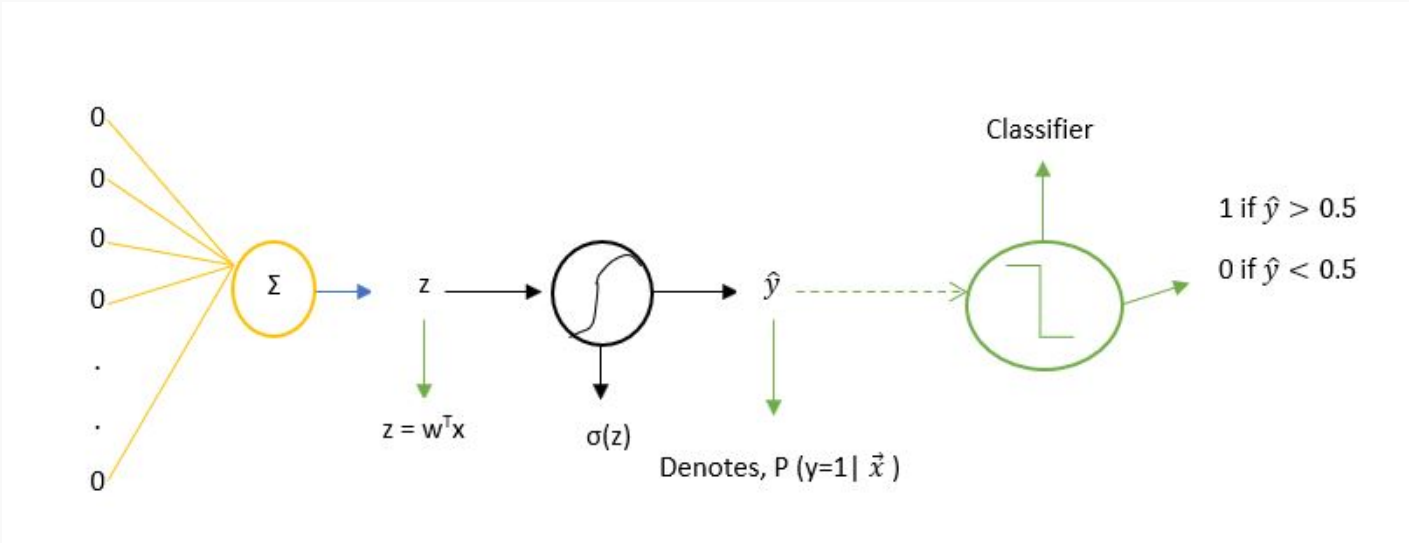
$$\sigma(z) = \frac{1}{1 + exp(-z)}$$

$$z \to -\infty,\ \sigma(z) \to 0$$

$$z \to \infty,\ \sigma(z) \to 1$$

$$z = 0,\ \sigma(z) = 0.5$$

Linear regression model



Summarize Linear Regression

$$\underset{\text{Input}}{\overrightarrow{X}} \rightarrow \underset{\text{features}}{(x_1^{(i)}, x_2.., x_n)}$$

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

Sum that

Find $\hat{y}$ $S \cdot t \frac{1}{m} || \overrightarrow{y} - \overrightarrow{\hat{y}} ||_2^2$ is minimized.

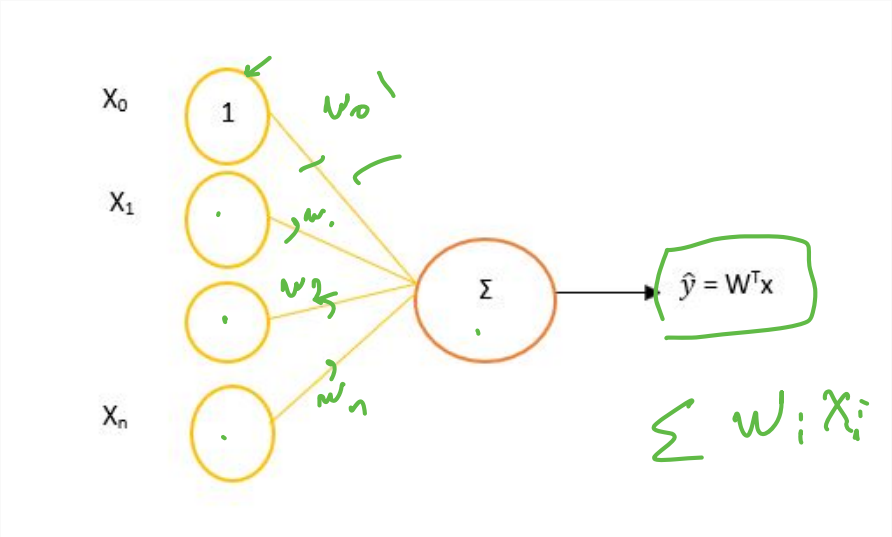
$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} -2 (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$$

$$w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$$

$(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$

$y^{(i)}(w)$

$y(w)$

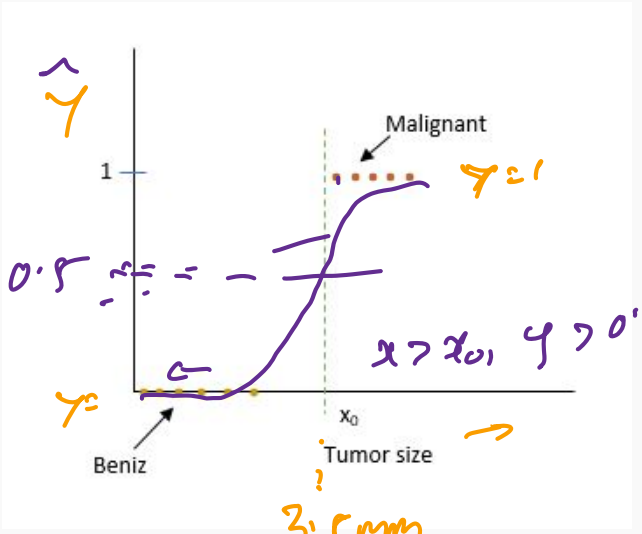


$\sum w_i x_i$

n- features

Classification:

one feature



Patient Data

Tumor size x (mm)	(y) Benign/ Malignant
2.0	0
2.5	0
3.0	0
3.2	0
3.5	1
3.9	1
4.0	1
5.0	1

$$y = \begin{cases} 1 & x > x_0 \\ 0 & x < x_0 \end{cases}$$

Recap

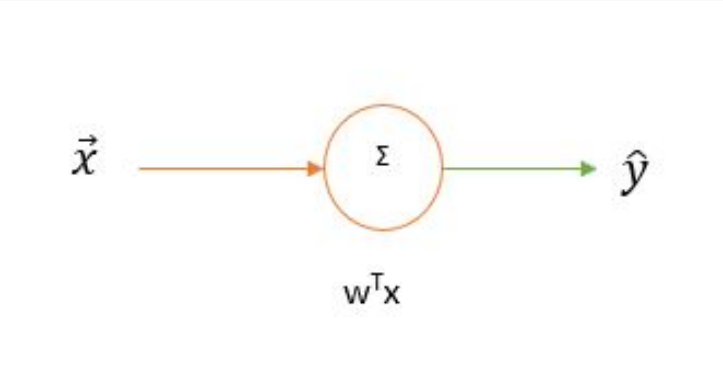
Linear Regression $\rightarrow$ For continuous data

Maps: $\overrightarrow{x} \rightarrow y$, where y is continuous.

Recap

Linear Regression $\rightarrow$ For continuous data

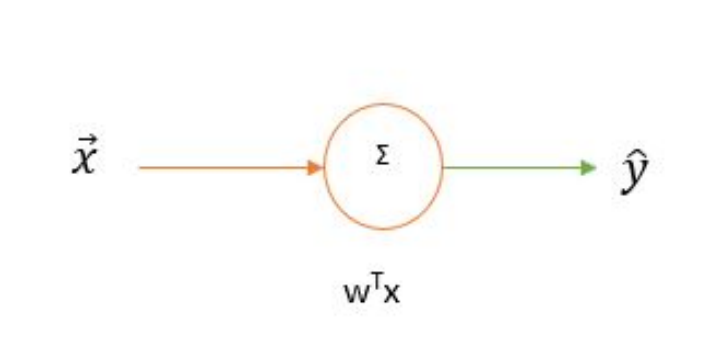
Maps: $\vec{x} \rightarrow y$, where y is continuous.



Recap

Linear Regression $\rightarrow$ For continuous data

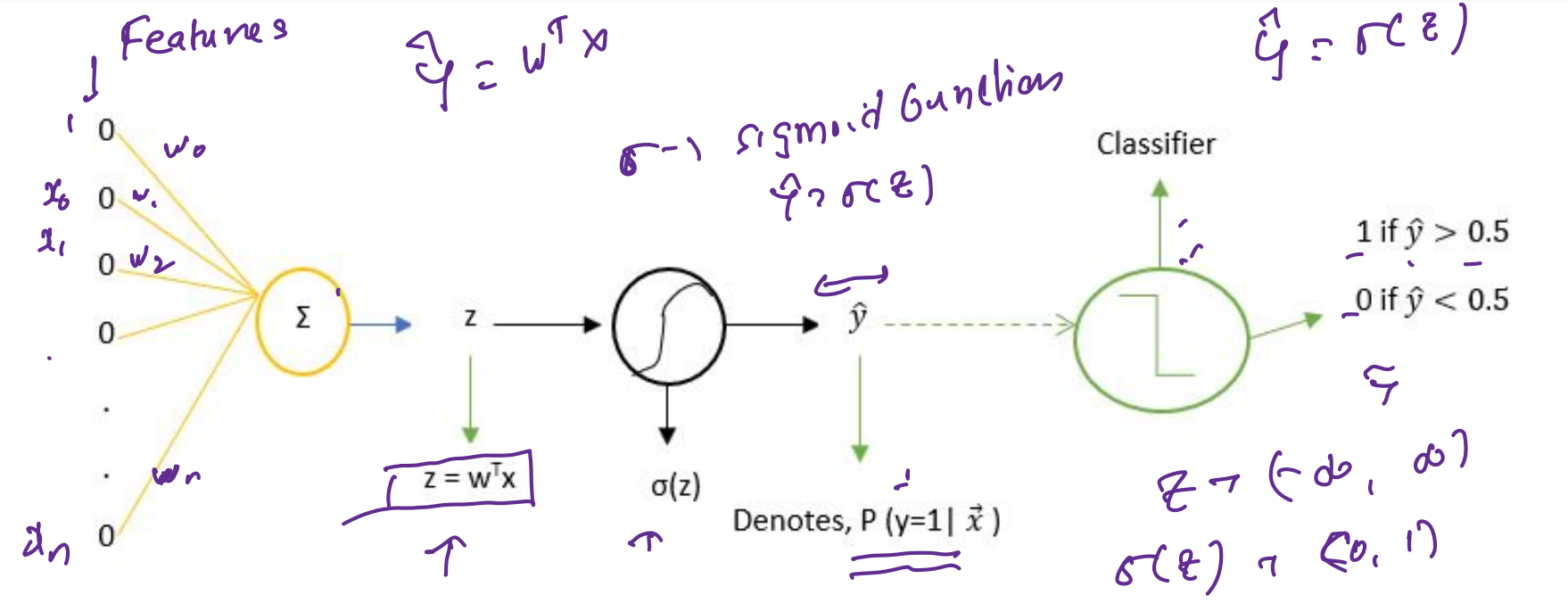
Maps: $\vec{x} \rightarrow y$, where y is continuous.



$$L = \sum_{i=1}^{m} (y^{(i)}_{True} - \hat{y}^{(i)}_{Predicted})^2$$

Logistic Regression → For discrete data (Classification)

Maps: $\vec{x} \rightarrow y$, where y is discrete which is either 0 or 1.

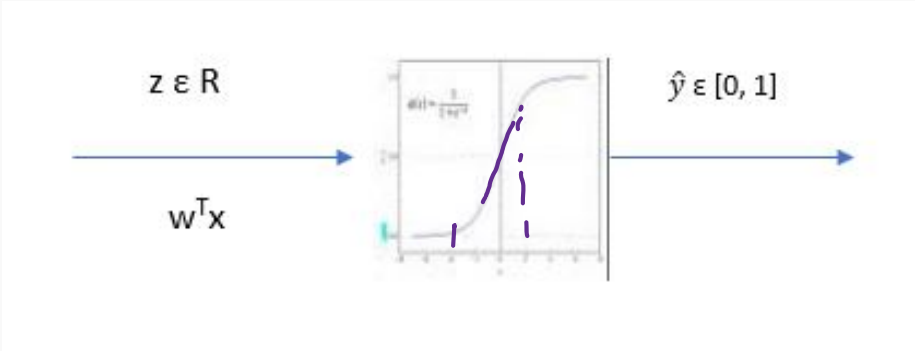


This z is continuous and unbounded. Sigmoid function is a maps to a continuous number between 0 and 1. This represents probability that ~~P(y=1—x)~~. $P(y=1|x)$

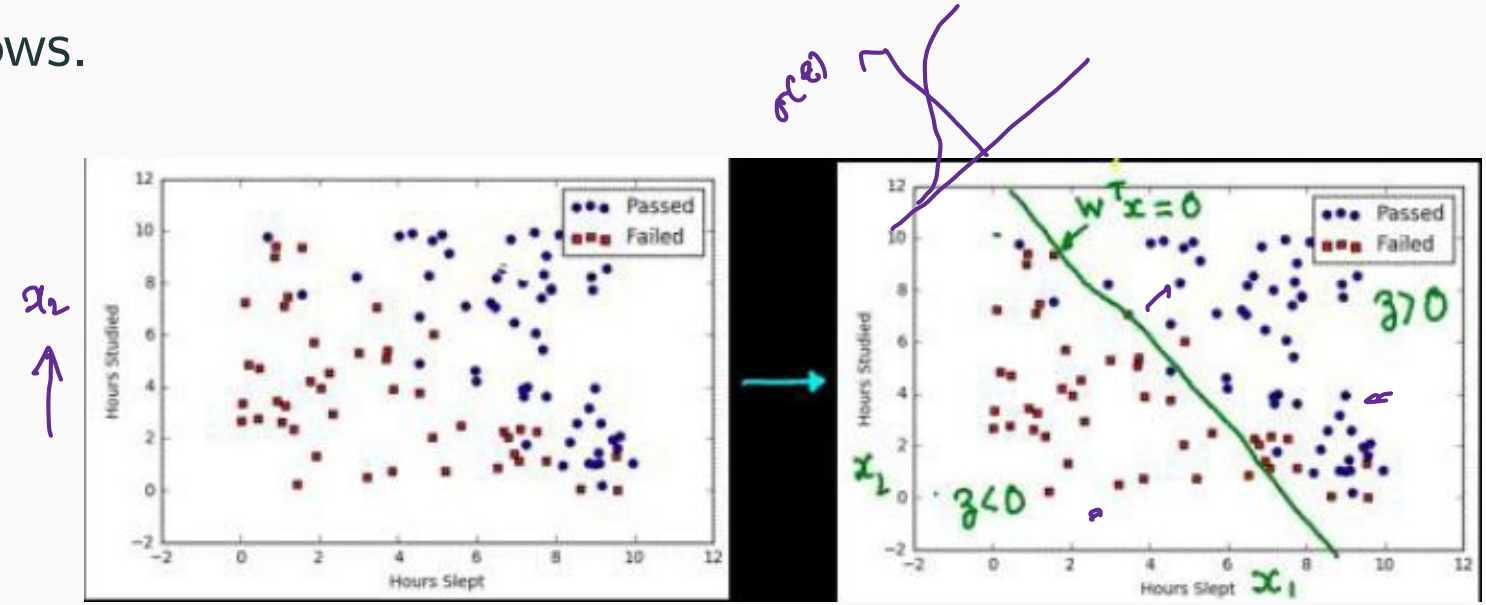
Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$

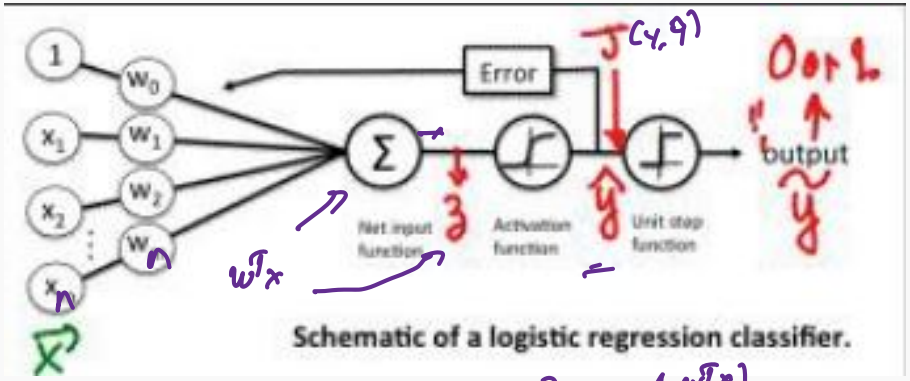
$z \to -\infty \Rightarrow \sigma(z) \to 0$

$z \to \infty \Rightarrow \sigma(z) \to 1$



Geometrically, the net effect of the logistic regression process is as follows.

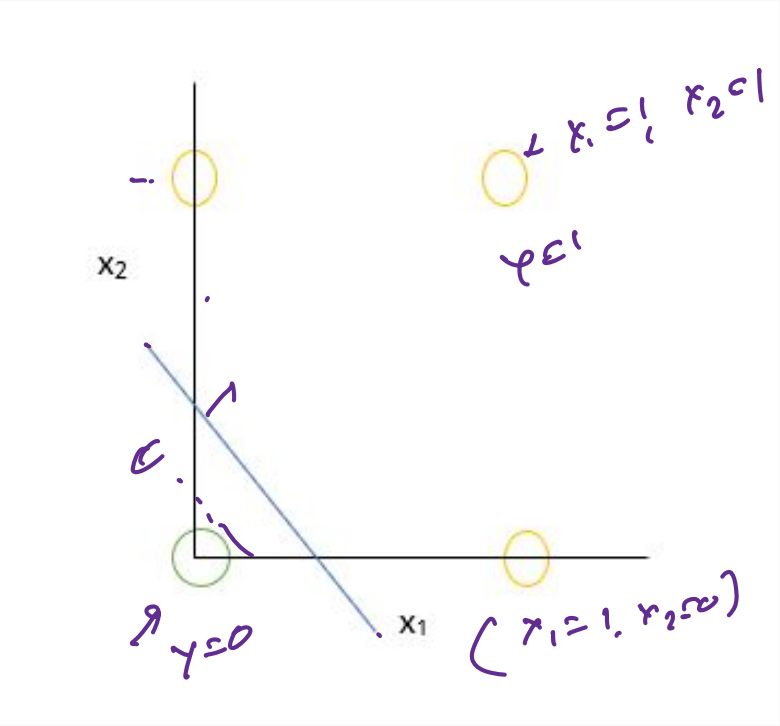




Example: OR Gate

x_1	x_2	y	z
0	0	0	< 0 (Need bias)
0	1	1	> 0
1	0	1	> 0
1	1	1	> 0

$$z = w_0 + w_1 x_1 + w_2 x_2$$



Find $\vec{w}$ $S \cdot t$ we get the OR gate.

$$w_0 = -1,\ w_1 = 2, w_2 = 2$$
$$\Rightarrow -1 + 2x_1 + 2x_2 = 0$$
$$\Rightarrow x_1 + x_2 = 1/2$$

Loss Function? What about LSE?

$$L = \sum (y^{(i)} - \hat{y}^{(i)})^2$$

y	$\hat{y}$	LSE
0	0	0
1	1	0
0	1	1
1	0	1

Cross- Entropy cost function

$$J^{(i)} = -\{ \underbrace{y^{(i)} \log \hat{y}^{(i)}}_{\text{Cost for } P(y=1|x)} + \underbrace{(1 - y^{(i)}) \log(1 - \hat{y}^{(i)})}_{\text{Cost for } P(y=0|x)} \}$$

Partial cost function

$$J = \sum_{i=1}^{m} J^{(i)}$$

Gradient Descent: $w = w - \alpha \nabla_w J$

For cross-entropy $J \rightarrow \frac{\partial J^{(i)}}{\partial w_j} = -(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$

$-(y^{(i)} - \hat{y}^{(i)}) x_j^{(i)}$ Similar to the least squares $\nabla_w J$

$\hat{y} = \sigma(z)$ z-ven layer number

$$-\left[y \log \hat{y} + (1-y) \log(1-\hat{y}) \right]$$

$\rightarrow y = 1, \quad \hat{y} \simeq 1$

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

$$1 \cdot \log 1 + 0 \cdot \log 0$$

$$0 + \div 0$$

$y = 1, \quad \hat{y} = 0$

$$-\left[-\infty + 0 \right] \Rightarrow +\infty$$

$y = 0$

$\hat{y} = 1$

$\hat{y} = 0$

Revisiting OR gate : Gradient Descent

Example	x_0	x_1	x_2	y	$\tilde{y}$
(1)	1	0	0	0	1
(2)	1	0	1	1	1
(3)	1	1	0	1	1
(4)	1	1	1	1	1

Do not do this $\tilde{y}$. In practice use $\hat{y}$.

$\alpha = 1$

$Say\ w = [2\ 2\ 2]^T$

$$\nabla_w J = \sum_{i=1}^{m} -(y^{(i)} - \hat{y}^{(i)})\vec{x}^{(i)}$$

$$= (1)\begin{bmatrix}1\\0\\0\end{bmatrix} + (0) + (0) + (0)$$

$$= [1\ 0\ 0]$$

$$\frac{\partial J}{\partial w_0} = (0-1)\cdot 1 \quad \begin{matrix} m_{21} \\ \uparrow \\ x_0 \end{matrix}$$

$$\frac{\partial J^{(i)}}{\partial w_j} = -\left(y^{(i)} - \hat{y}^{(i)}\right) x_j^{(i)}$$

$$\begin{pmatrix} \frac{\partial J^{(1)}}{\partial w_0} \\ \frac{\partial J^{(1)}}{\partial w_1} \\ \frac{\partial J^{(1)}}{\partial w_2} \end{pmatrix} = \begin{matrix} -(0-1)\, x_0^{(1)} = 1 \\ -(0-1)\, x_1^{(1)} = 0 \\ -(0-1)\, x_2^{(1)} = 0 \end{matrix}$$

$$w \leftarrow w - 1 \cdot \frac{\partial J}{\partial w}$$

$$\begin{bmatrix} \quad \end{bmatrix} \leftarrow \begin{bmatrix} 2 \\ 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$, y

$J = (y - \hat{y})^2$ ← one data point

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1}$$

$$= 2(y - \hat{y}) \cdot \frac{\partial \hat{y}}{\partial w_1}$$

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2$$

$$\frac{\partial \hat{y}}{\partial w_1} = 0 + x_1 + 0 = x_1$$

$$\left.\begin{aligned} \frac{\partial J}{\partial w_1} &= 2(y - \hat{y}) \cdot x_1 \\ \frac{\partial J}{\partial w_2} &= 2(y - \hat{y}) \cdot x_2 \end{aligned}\right\}$$ For one Data point

$$\frac{\partial J}{\partial w_1} = 2(y^{(i)} - \hat{y}^{(i)}) \cdot x_1$$

$$\frac{\partial J}{\partial w_j} = 2(y^{(i)} - \hat{y}^{(i)}) \cdot x_j$$ ← gradient contribution from one data point (i)

m data points

$$\frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} 2(y^{(i)} - \hat{y}^{(i)}) \cdot x_j^{(i)}$$

↳ Do this for all j

$$w_j \leftarrow \left[w_j - \alpha \frac{\partial J}{\partial w_j} \right]$$ ← Linear Regression, Logistic Regression

$$\Rightarrow w = w - [1\ 0\ 0] = [1\ 2\ 2]$$

$$\downarrow$$

$$[0\ 2\ 2]$$

$$\downarrow$$

$$[-1\ 2\ 2]$$

$$\tilde{y} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} = y$$

$$\nabla_w J = 0$$

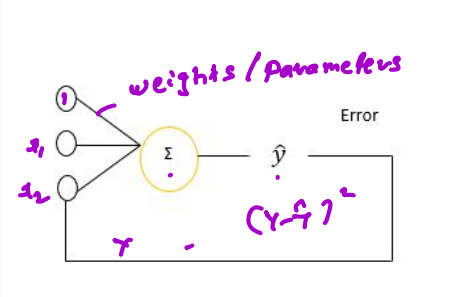
When number of classes > 2, k=2 Binary classification

1. Some representation for y.
2. Loss function.
3. Non-linearity

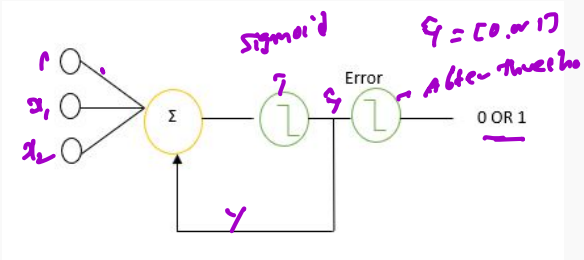
Linear Regression

Logistic Regression

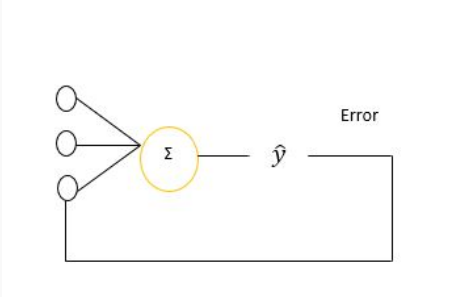
Linear Regression



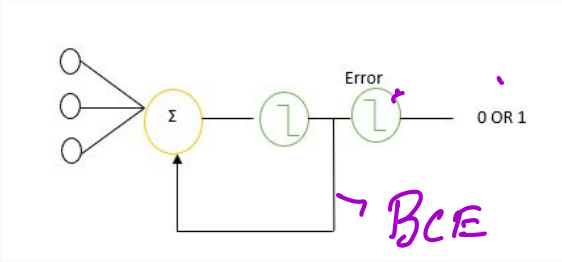
Logistic Regression



Linear Regression



Logistic Regression



Feed forward process: i.e. Finding output: Given (Inputs, Parameters)

Loss fn => $L(y, \hat{y}) = (y - \hat{y})^2$

y → Actual price

$\hat{y}$ → Estimated price

Logistic y - 0, or 1

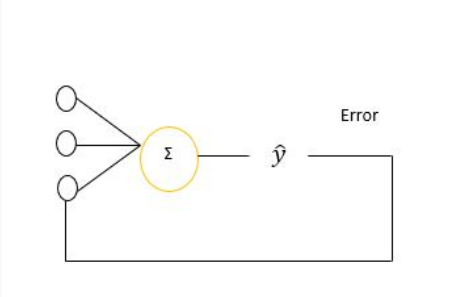
$\hat{y}$ - prob $(y = 1 | x)$

↳ $-(y \log \hat{y}$

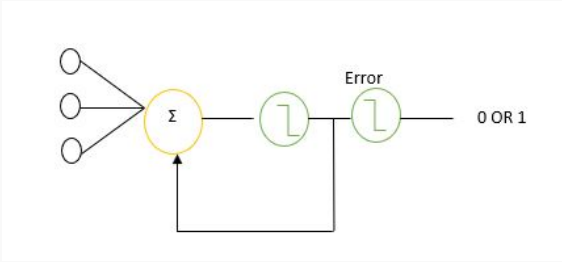
y: 1

$\hat{y}$: 0

Linear Regression



Logistic Regression



Feed forward process: i.e. Finding output: Given (Inputs, Parameters)

$$\hat{y} = w^T \vec{x} = \vec{w} \cdot \vec{x}$$

i.e. $\hat{y}^{(i)} = \sum_{j=0}^{n} w_j x_j^{(i)}; \; x_0^{(i)} = 1$

$$\hat{y} = \frac{1}{1 + e^{wx}} = sigmoid(\vec{w} \cdot \vec{x})$$

A quantized classifier is applied after this.

Loss function: $L = \sum_{i=1}^{m} L^{(i)}$

L over the samples

or

Training Examples

Loss function: $L = \sum_{i=1}^{m} L^{(i)}$

$$L^{(i)} = \frac{1}{2}\left[\underbrace{y^{(i)}}_{Groundtruth} - \underbrace{\hat{y}^{(i)}}_{Prediction}\right]^2$$

$$L^{(i)} = -\left\{\underbrace{y^{(i)} \ln \hat{y}^{(i)}}_{Cross-entropy} + \underbrace{(1 - y^{(i)}) \ln(1 - \hat{y}^{(i)})}_{Costfunction}\right\}$$

$$= -\sum_{k=0}^{1} y_k^{(i)} \ln \hat{y}_k^{(i)}$$

↳ over the categories

Loss function: $L = \sum_{i=1}^{m} L^{(i)}$

$$L^{(i)} = \frac{1}{2}\left[\underbrace{y^{(i)}}_{Groundtruth} - \underbrace{\hat{y}^{(i)}}_{Prediction}\right]^2$$

$$L^{(i)} = -\left\{\underbrace{y^{(i)} \ln \hat{y}^{(i)}}_{Cross-entropy} + \underbrace{(1 - y^{(i)}) \ln(1 - \hat{y}^{(i)})}_{Costfunction}\right\}$$

$$= -\sum_{k=0}^{1} y_k^{(i)} \ln \hat{y}_k^{(i)}$$

Gradient:

Loss function: $L = \frac{1}{m}\sum_{i=1}^{m} L^{(i)}$ → EXPECTATION

$$L^{(i)} = \frac{1}{2}\left[\underbrace{y^{(i)}}_{Groundtruth} - \underbrace{\hat{y}^{(i)}}_{Prediction}\right]^2$$

$$L^{(i)} = -\left\{\underbrace{y^{(i)} \ln \hat{y}^{(i)}}_{Cross-entropy} + (1 - y^{(i)}) \ln(1 - \hat{y}^{(i)})\right\}$$

$$= -\sum_{k=0}^{1} y_k^{(i)} \ln \hat{y}_k^{(i)}$$

Gradient:

$$\frac{\partial L^{(i)}}{\partial w_j} = \{\hat{y}^{(i)} - y^{(i)}\} x_j^{(i)}$$

n - weights / parameter

$$\frac{\partial L^{(i)}}{\partial w_j} = \{\hat{y}_1^{(i)} - y_1^{(i)}\} x_j^{(i)}$$

$$w_j = w_j - \alpha \frac{\partial L^{(i)}}{\partial w_j}$$

General Algorithm for finding weights for the feedforward network

1. Initialize weights via some guess $\overrightarrow{w_{init}}$
2. Use feedforward network to find prediction $\hat{y}$. Typically this will be different from ground truth y.
3. Improve weights through gradient descent.

$$w = w - \alpha \frac{\partial L}{\partial w}$$

Repeat step 2 and 3 til convergence.

Multinomial classification of $K > 2$ classes requires

- Representation for y
- Nonlinearity
- Cost/ Loss function

Representation for y

One-hot vector: say 3 classes

One hot vector representation

1 of k

$$\begin{bmatrix}1\\0\\0\end{bmatrix}_{Class1} \text{ OR } \begin{bmatrix}0\\1\\0\end{bmatrix}_{Class2} \text{ OR } \begin{bmatrix}0\\0\\1\end{bmatrix}_{Class3}$$

Bad predictions

$$\begin{bmatrix}0.6\\0.1\\0.05\\\vdots\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} \rightarrow \hat{y} = \begin{bmatrix}\hat{y}_1\\\hat{y}_2\\\hat{y}_3\end{bmatrix}$$

Good predictions

$$\begin{bmatrix}0.01\\0.2\\0.9\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\hat{y}_k = P(y_k = 1|x)$$

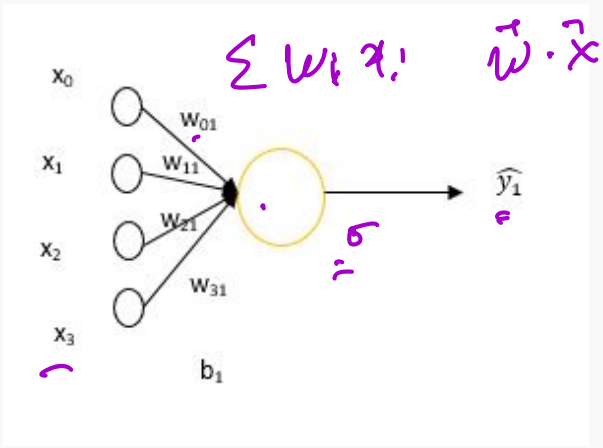
In general, $\hat{y} = \begin{bmatrix} P(y_1 = 1|x) \\ P(y_2 = 1|x) \\ . \\ . \\ . \\ P(y_k = 1|x) \end{bmatrix}$

1 of k classes

Schematic: K= 3 classes

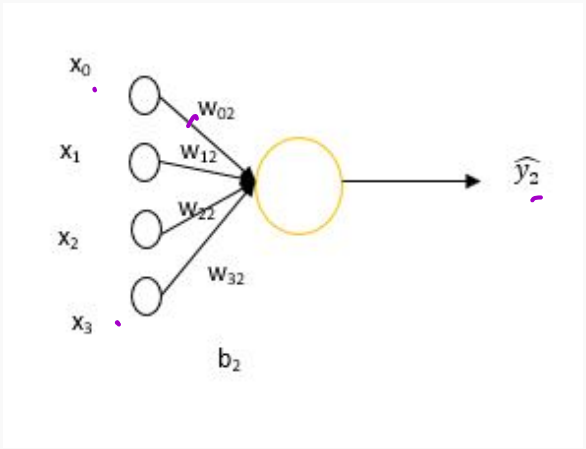
3. 4

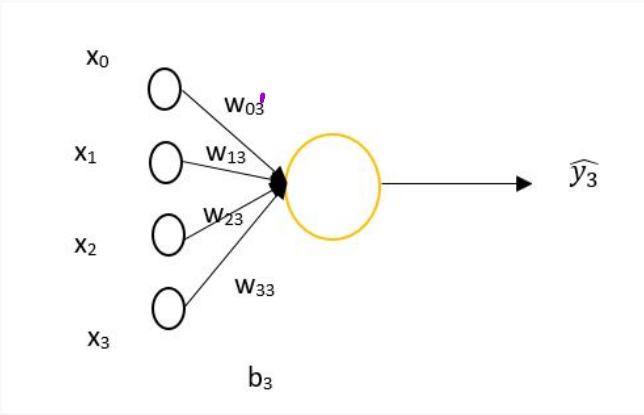
$(\hat{y}, 1-\hat{y})$



$\sum w_i x_i$ $\vec{w} \cdot \vec{x}$

$= 5$





Separate $\hat{y}$ for each class,
using k different (k=3) sets of weights/params

$$\hat{y}_1 = g(w_{01}x_0 + w_{11}x_1 + w_{21}x_2 + w_{31}x_3)$$

W_{ij}, where i is the feature and j is the class

$$\vec{\hat{y}} = g(W^T x + \vec{b})$$

What is g?

Suppose we chose g = Sigmoid function

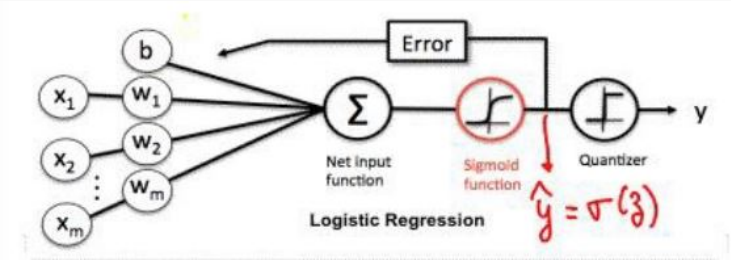
$$\sigma(z) = \frac{1}{1 + exp(-z)} = \frac{exp(z)}{1 + exp(z)}$$

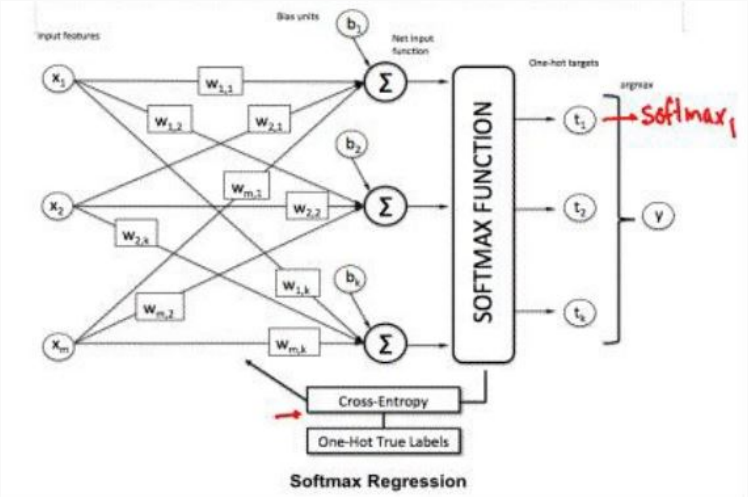
$z_1 \rightarrow \circ \rightarrow \hat{y}_1 = g(z_1)$

$z_2 \rightarrow \circ \rightarrow \hat{y}_2 = g(z_2)$

$z_3 \rightarrow \circ \rightarrow \hat{y}_3 = g(z_3)$

In general, if we set g= sigmoid, $\sum_{k=1}^{k} \hat{y}_k \neq 1$. So, we use softmax. $softmax_j = \frac{exp(z_j)}{\sum_{k=1}^{k} exp(z_k)}$





Cost function (Cross-Entropy) :

$$L^{(i)} = -\sum_{k=1}^{K} y_k^{(i)} \ln \hat{y}_k^{(i)}$$

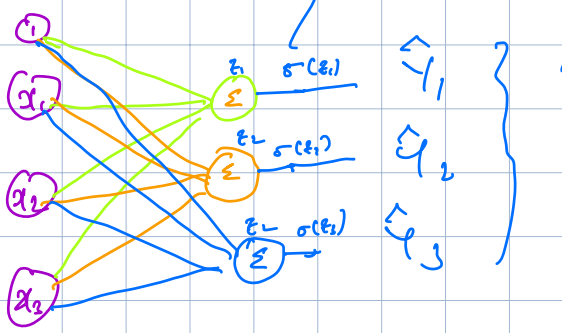
y_1 y_2 y_3

$[0\ 1\ 0]$

$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$

$[0.05\ 0.8\ 0.15]$

4 for one data point



$$\hat{y}_1 = p_v(y_1|x)$$
$$\hat{y}_2 = p_v(y_2|x)$$
$$\hat{y}_n = p_v(y_3|x)$$

Independent

$y_1 + y_2 + y_3 \neq 1$

Softmax output

$$\hat{y}_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\hat{y}_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

$$\hat{y}_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

Softmax output

Compare this with binary cross-entropy, which is $-\sum_{k=1}^{2} y_k^{(i)} \ln \hat{y}_k^{(i)}$

$$L = \sum_{i=1}^{m} L^{(i)} = -\sum_{i=1}^{m} \left[\sum_{k=1}^{K} y_k^{(i)} \ln \hat{y}_k^{(i)} \right]$$

G.T

probability output

Summary: Binary vs Multinomial Classification

Binary	Multinomial
If there are two classes	For more than 2 classes
$\hat{y} = \sigma(\vec{w}.\vec{x})$	$\hat{y}_j = \frac{e^{w_j \cdot x}}{\sum_{i=1}^{k} e^{w_j \cdot x}}$
$\hat{y}$: Scalar $\epsilon[0,1]$	$\hat{y}$: One hot vector
w is a vector: $(n+1) \times 1$	w is a matrix: $(n+1) \times K$
$L^{(i)} = -\sum_{k=1}^{2} y_k^{(i)} \ln \hat{y}_k^{(i)}$	$L^{(i)} = -\sum_{k=1}^{K} y_k^{(i)} \ln \hat{y}_k^{(i)}$

j - class index

n - No of feature

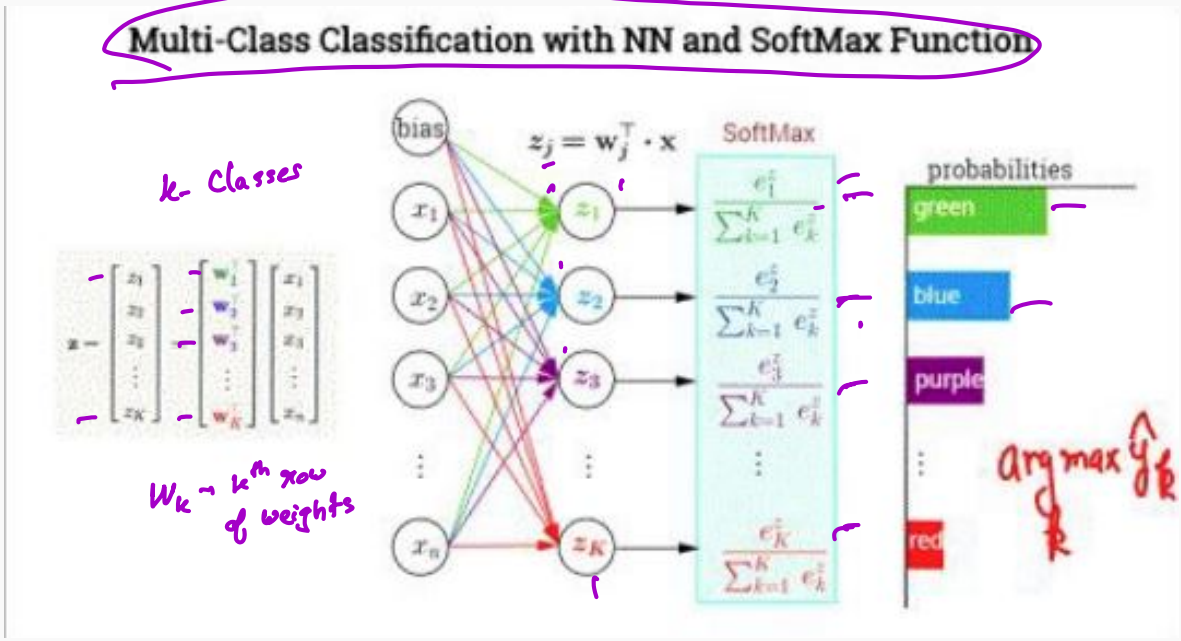
+1 - Bias unit

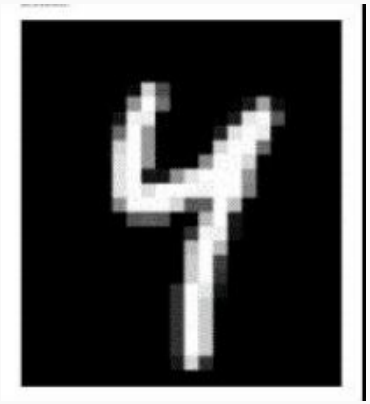
k=1, $\hat{y}$

k=2, $1-\hat{y}$

$$\begin{pmatrix} w_{01} & w_{02} & \cdots w_{0K} \\ \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & w_{nK} \end{pmatrix}$$

Examples





$$\sum_k y_k \ln \hat{y}_k$$

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

$$\ln(\hat{y}_i)$$

$$z_i - \ln\left(\sum_k e^{z_k}\right)$$

$$\hat{y}_1 = \frac{e^{z_1}}{\sum_{k=1}^{K} e^{z_k}}$$

$$\ln \hat{y}_1 = \ln(e^{z_1}) - \ln\left(\sum_{k=1}^{K} e^{z_k}\right)$$

$$= z_1 - \ln\left(\sum_{k=1}^{K} e^{z_k}\right)$$

$$z_1 = w_1^T x$$

↳ class index

$$L = \sum_{k=1}^{K} y_k \ln \hat{y}_k$$

$\frac{dL}{dw}$

features, $n = 3$

$k=1$ $k=2$ $k=3$

$$\begin{pmatrix} w_0 & w_0 & w_0 \\ w_1 & w_1 & w_1 \\ w_2 & w_2 & w_2 \\ w_3 & w_3 & w_3 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$$

$(w^T x)$

one hot encoding

softmax

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \hat{y}_3 \end{pmatrix} \quad \hat{y}$$

$$\hat{q} = \begin{pmatrix} \hat{q}_1 \\ \hat{q}_2 \\ \hat{q}_3 \end{pmatrix} \qquad \text{G.T } q_2 \begin{pmatrix} 0 \to q_1 \\ 1 \to q_2 \\ 0 \to q_3 \end{pmatrix}$$

$$\begin{pmatrix} 0 \\ \hat{q} \end{pmatrix}$$

$$\text{G.T}: q = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

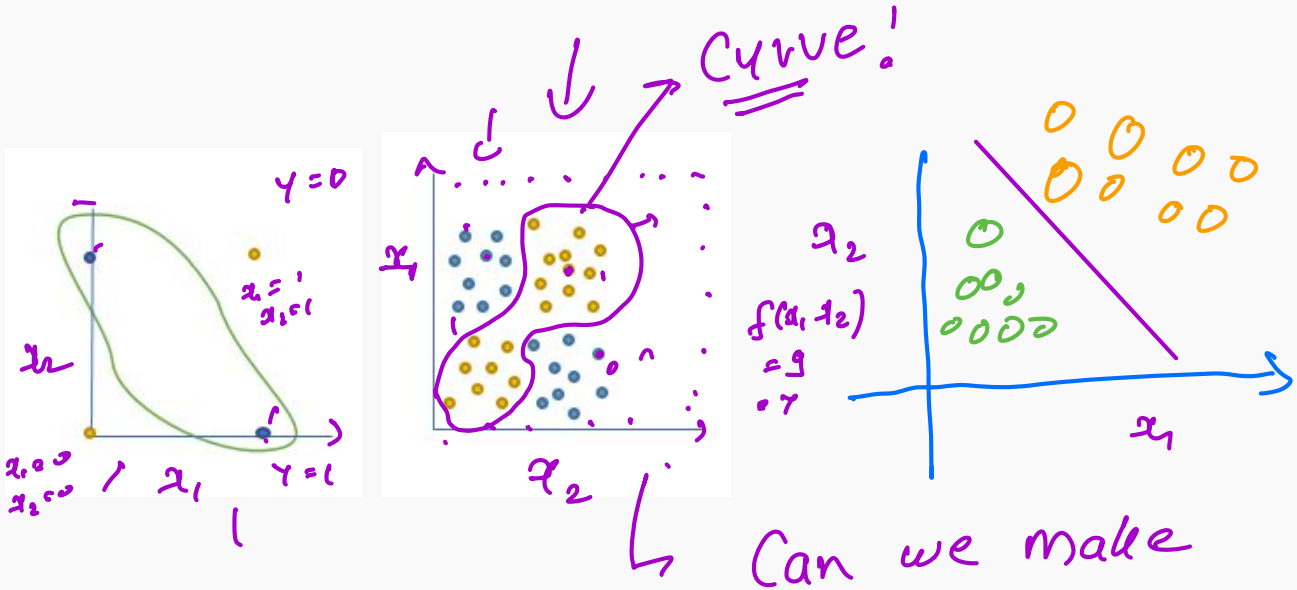
one data point

$$\ln \hat{q} = \begin{pmatrix} \ln \hat{q}_1 \\ \ln \hat{q}_2 \\ \ln \hat{q}_3 \end{pmatrix} \qquad \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix}$$

$$\sum_{k=1}^{3} q_k \ln \hat{q}_k = (q_1\ q_2\ q_3) \begin{pmatrix} \ln \hat{q}_1 \\ \ln \hat{q}_2 \\ \ln \hat{q}_3 \end{pmatrix}$$

$1 \times 3 \qquad 3 \times 1$

Why Deep networks?



Not linearly separable: XOR

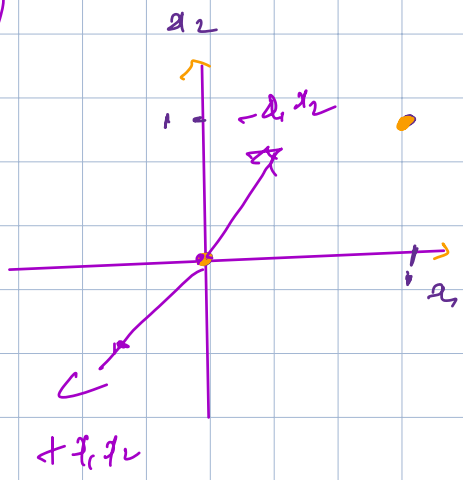
$x_1 \in [0,1]$
$x_2 \in [0,1]$

x_1	x_2	y
A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

→ A classifier that will mimic this logic gate. ?

$$y = x_1 + x_2 - 2x_1x_2$$

x_1	x_2	y
0	0	0
0	1	+1
1	0	+1
1	1	0



$y = x_1 + x_2 - 2x_1x_2$ ← logic gate

↳ non-linear term

$x_1x_2 = x_3$

$y = [x_1 + x_2 - 2x_3]$

↳ hyperplane

Why Deep networks?

Solution:

Use nonlinear features (Kernel Trick) can lead to combinational explosion (cures of dimensionality) $\rightarrow X_1^2, X_1X_2, X_2^2$.

Trade dimension for complexity

$\downarrow$

Neural Networks with "hidden" layers.

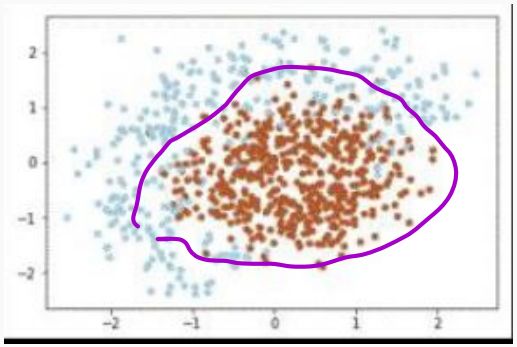
$\downarrow$

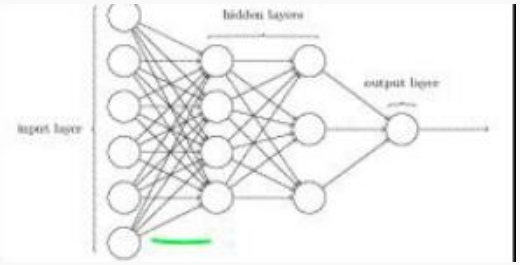
Universal Approximation Theorem

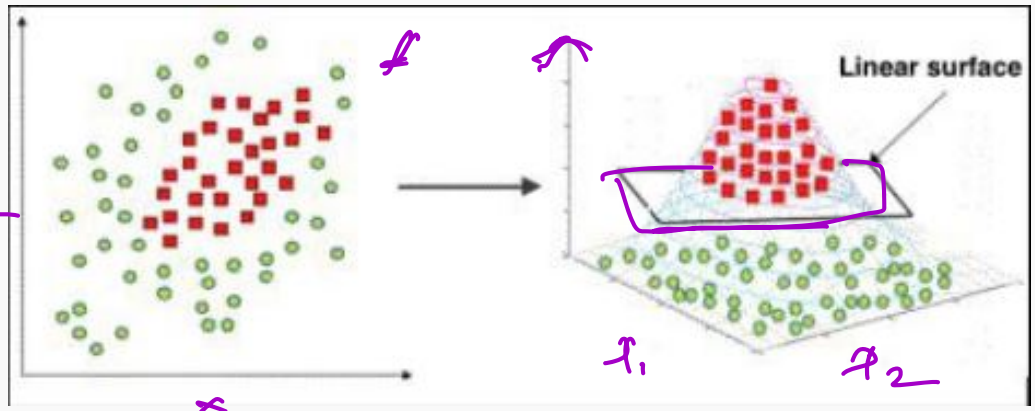
$\downarrow$

Any function can be approximated arbitrarily close by a Neural Network with a single hidden layer.

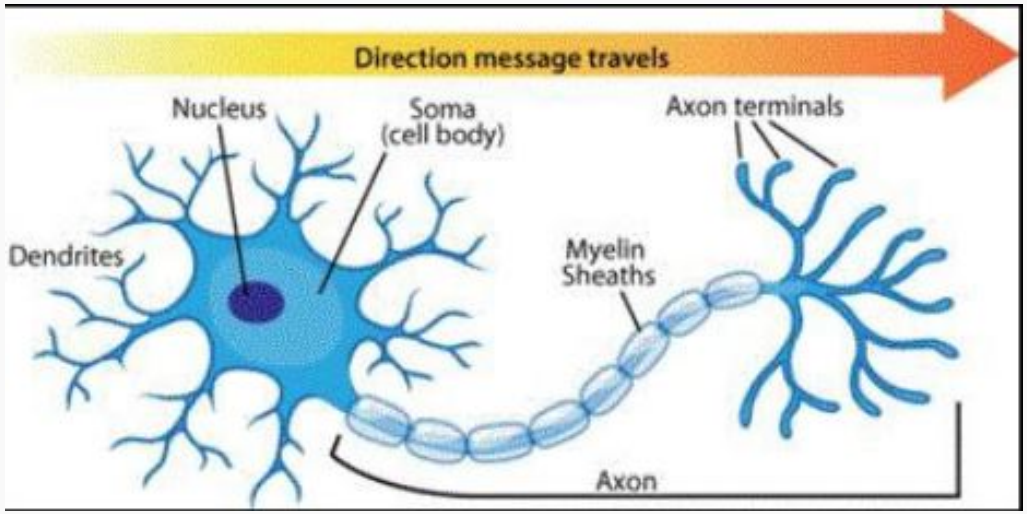
Deep Networks

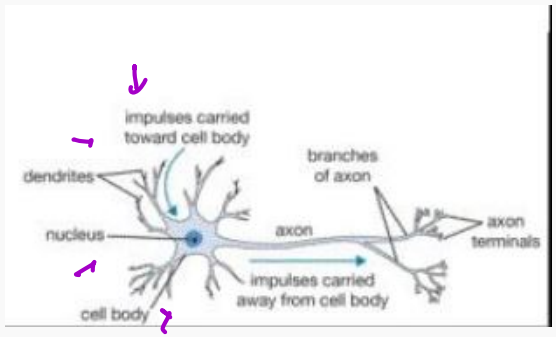


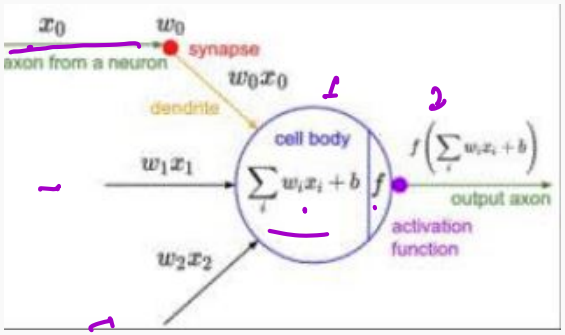




Biological Analogy







How does our Brain Work?

- A neuron is connected to other neurons via its input and output links
- Each incoming neuron has an activation value and each connection has a weight associated with it
- The neuron sums the incoming weighted values and this value is input to an activation function
- The output of the activation function is the output from the neuron

Neural Networks

Neural Networks

Linear Model:

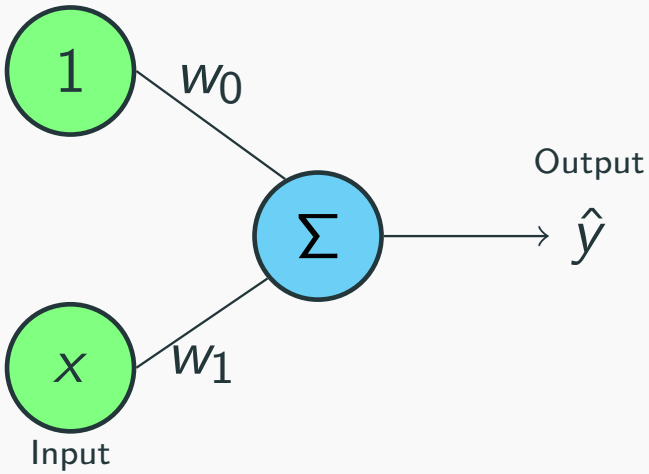
Neural Networks

Linear Model:

$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural Networks

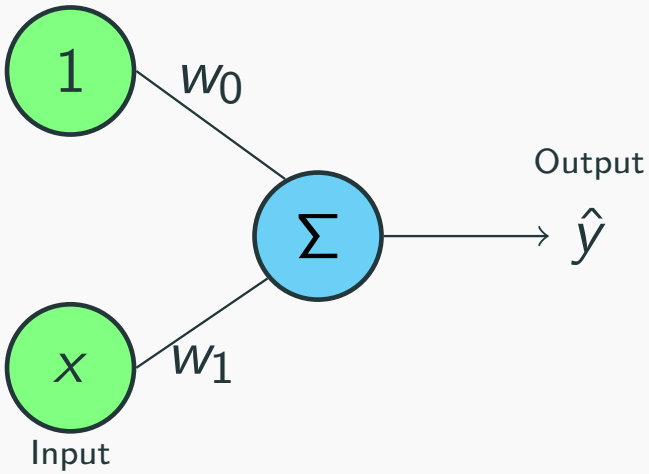
Linear Model:



$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural Networks

Linear Model:

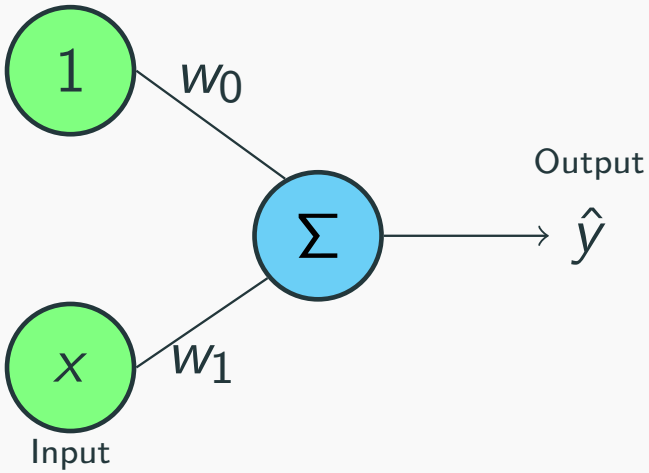


Neural network model:

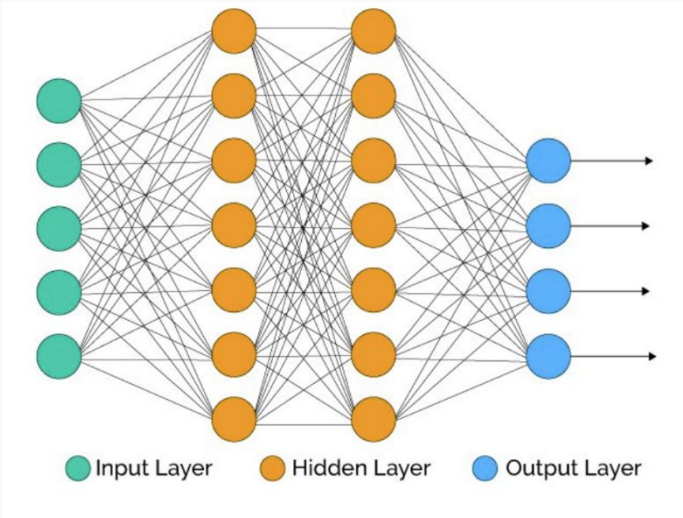
$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural Networks

Linear Model:



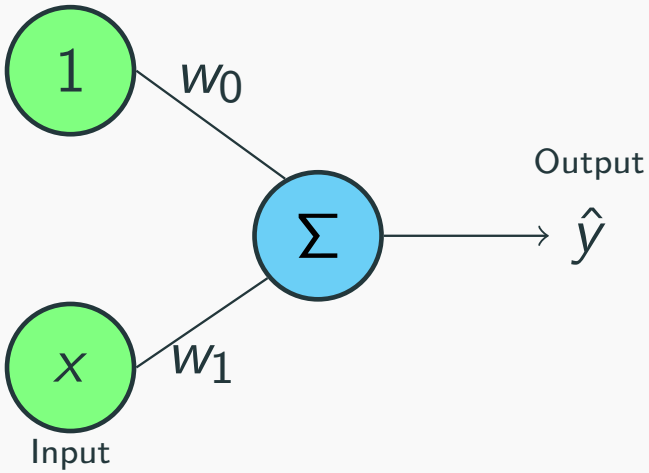
Neural network model:



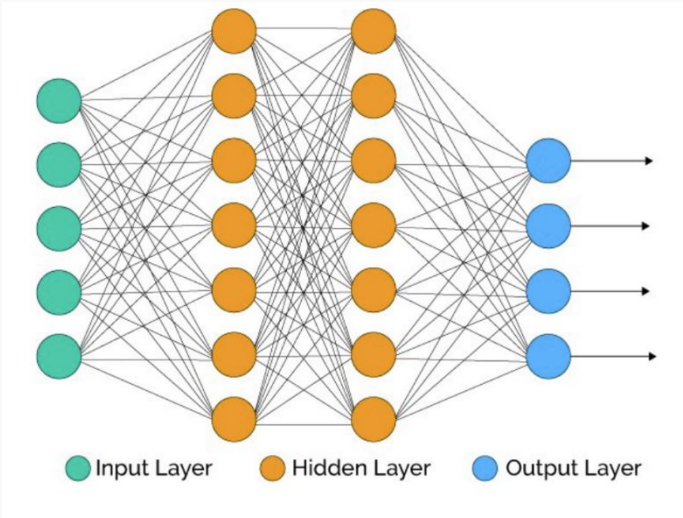
$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural Networks

Linear Model:



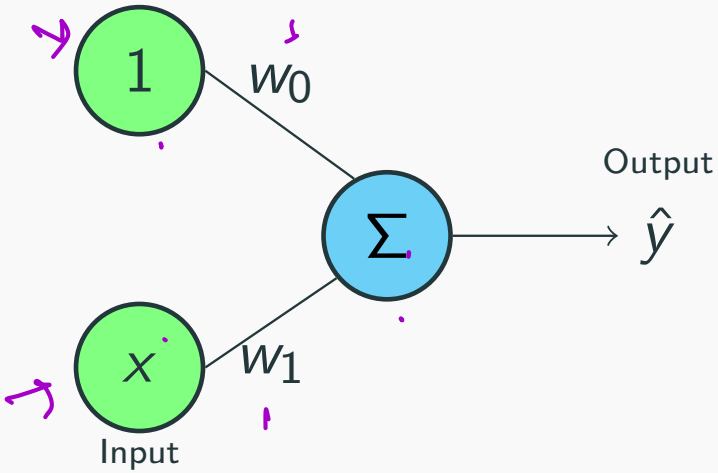
Neural network model:



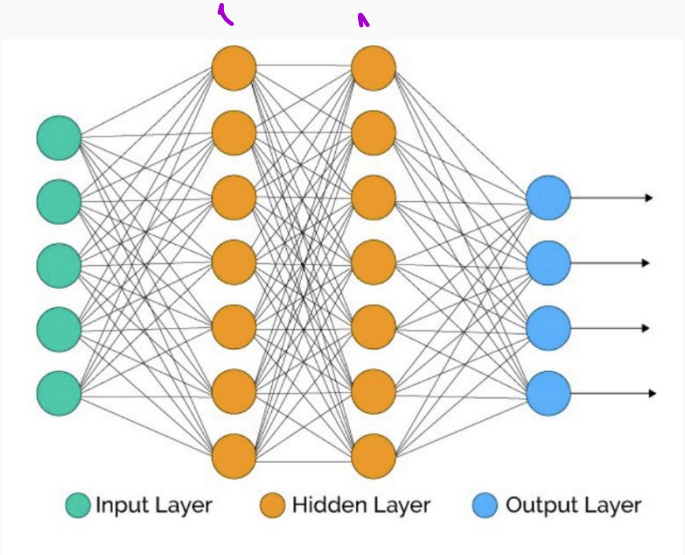
$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural Networks

Linear Model:

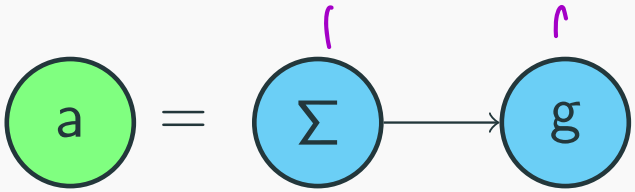


Neural network model:



$$a = g(\sum w_i x_i)$$

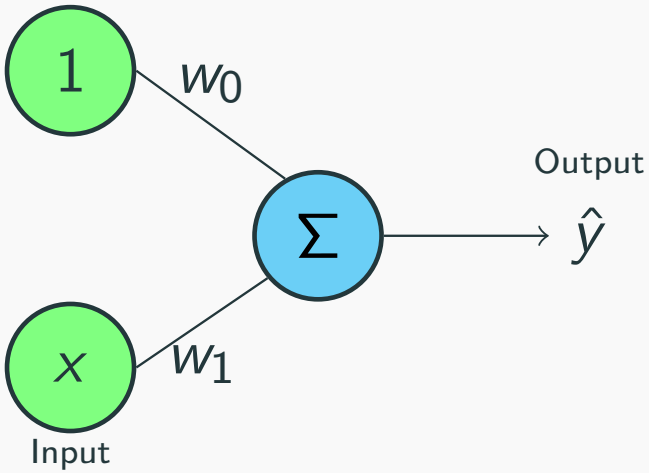
$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$



A Neuron has a linear and a nonlinear operation.

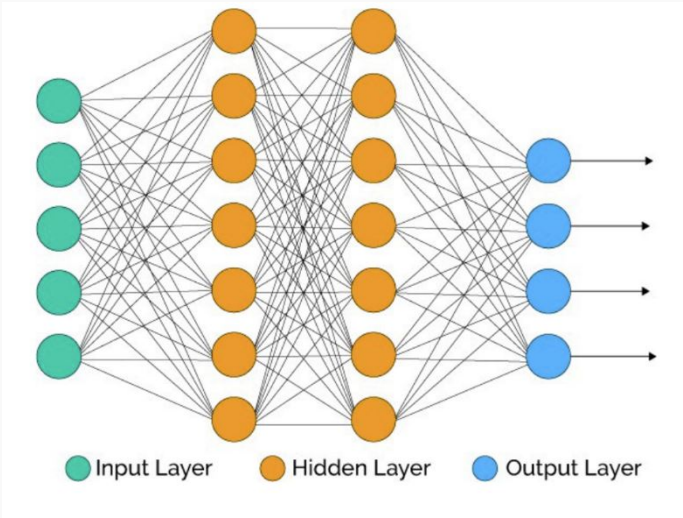
Neural Networks

Linear Model:



$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

Neural network model:



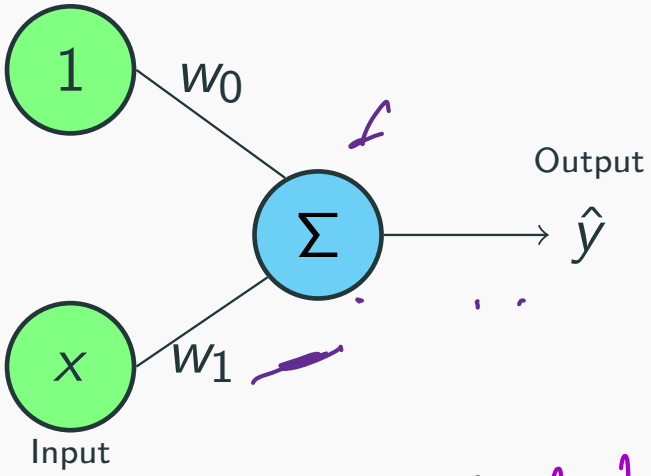
$$a = g(\sum w_i x_i)$$

A Neuron has a linear and a nonlinear operation.

Example of non-linear function is $Sigmoid(x) = \frac{1}{1+exp(-x)}$

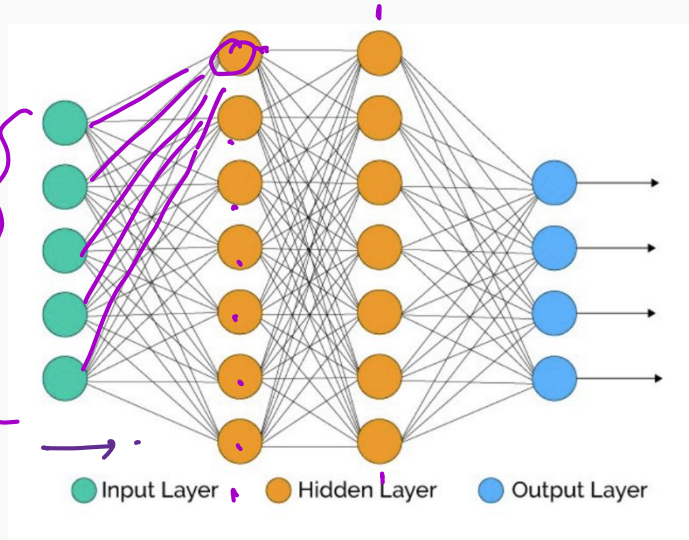
Neural Networks

Linear Model:



$$\hat{y} = w_0 + w_1 x = \sum w_i x_i$$

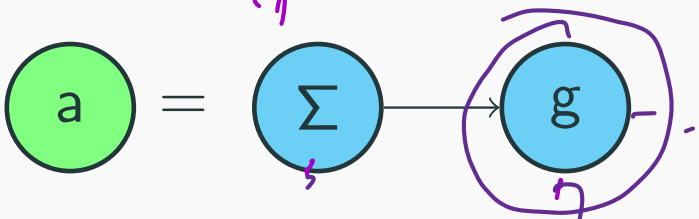
Neural network model:

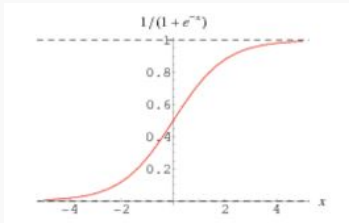


At least A hidden layer

$$a = g(\sum w_i x_i)$$

weighted sum → non-linearity



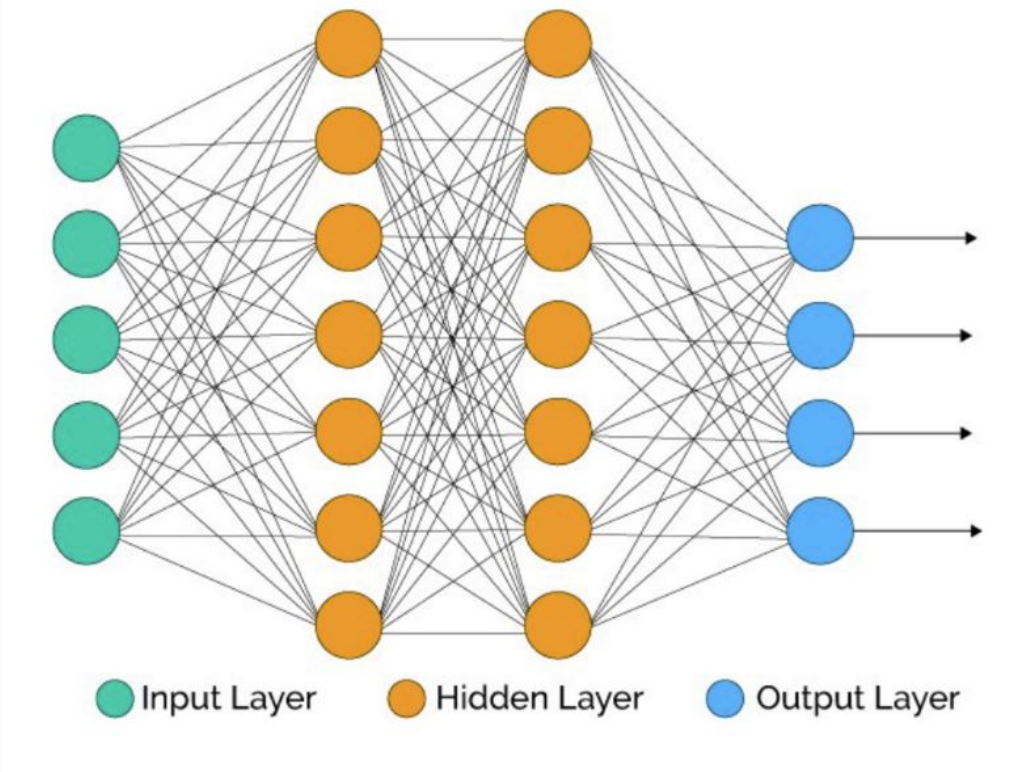


A Neuron has a linear and a nonlinear operation.

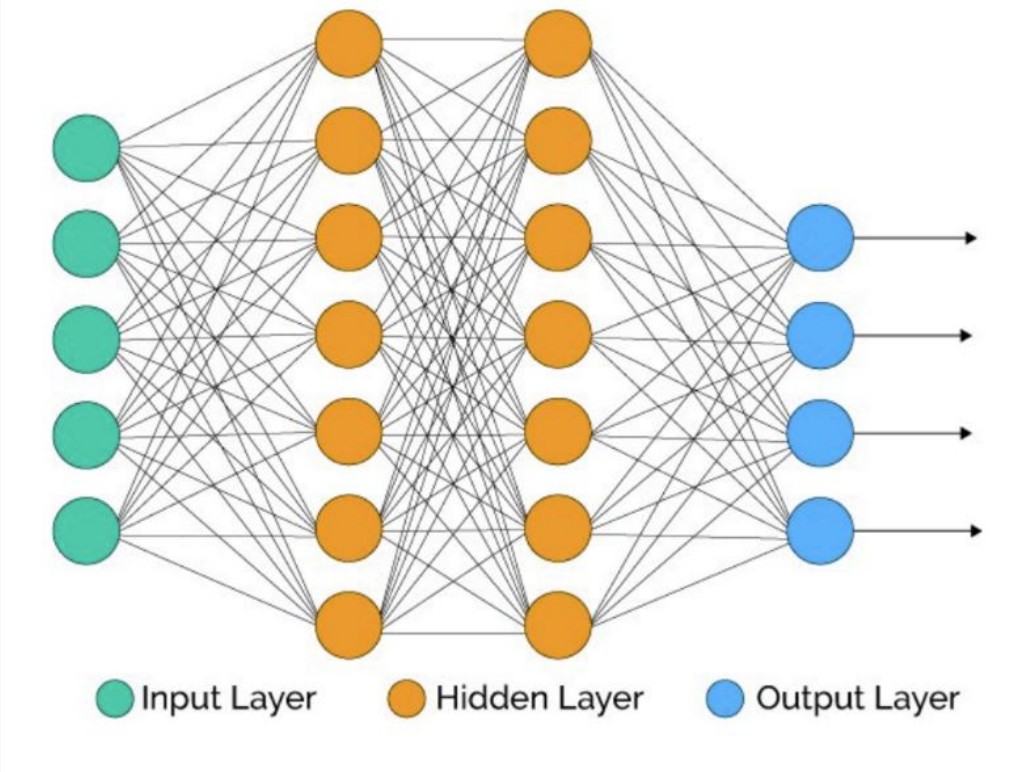
creates non-linear features

Example of non-linear function is $Sigmoid(x) = \frac{1}{1+exp(-x)}$

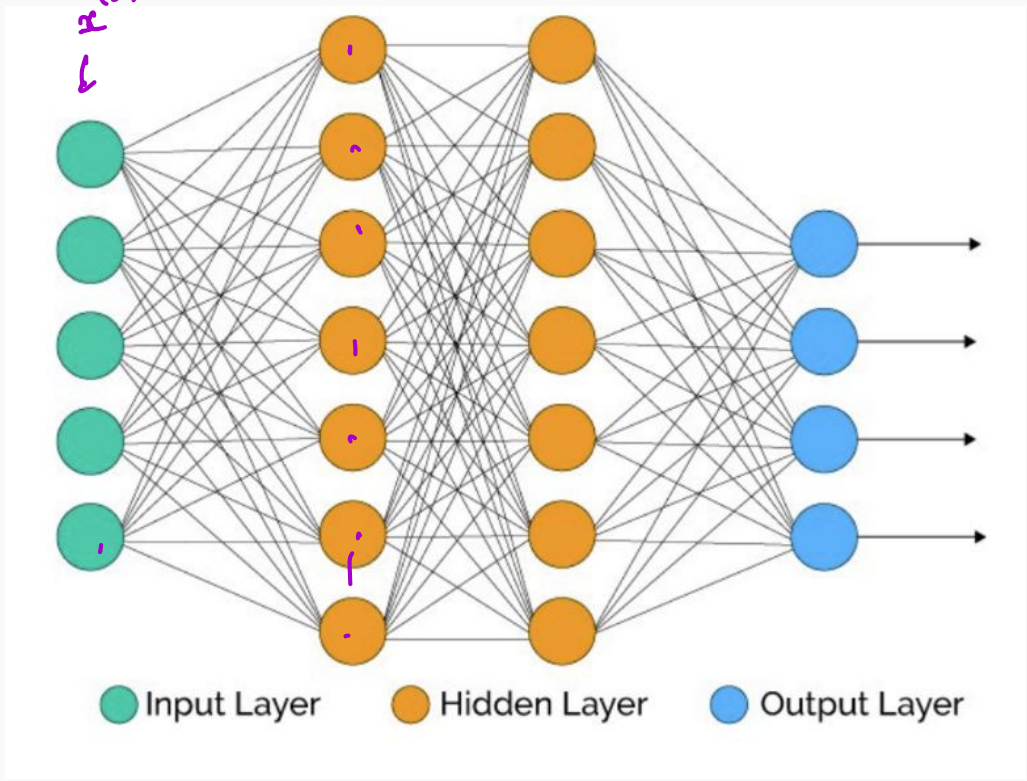
Universal Approximation Theorem



Universal Approximation Theorem



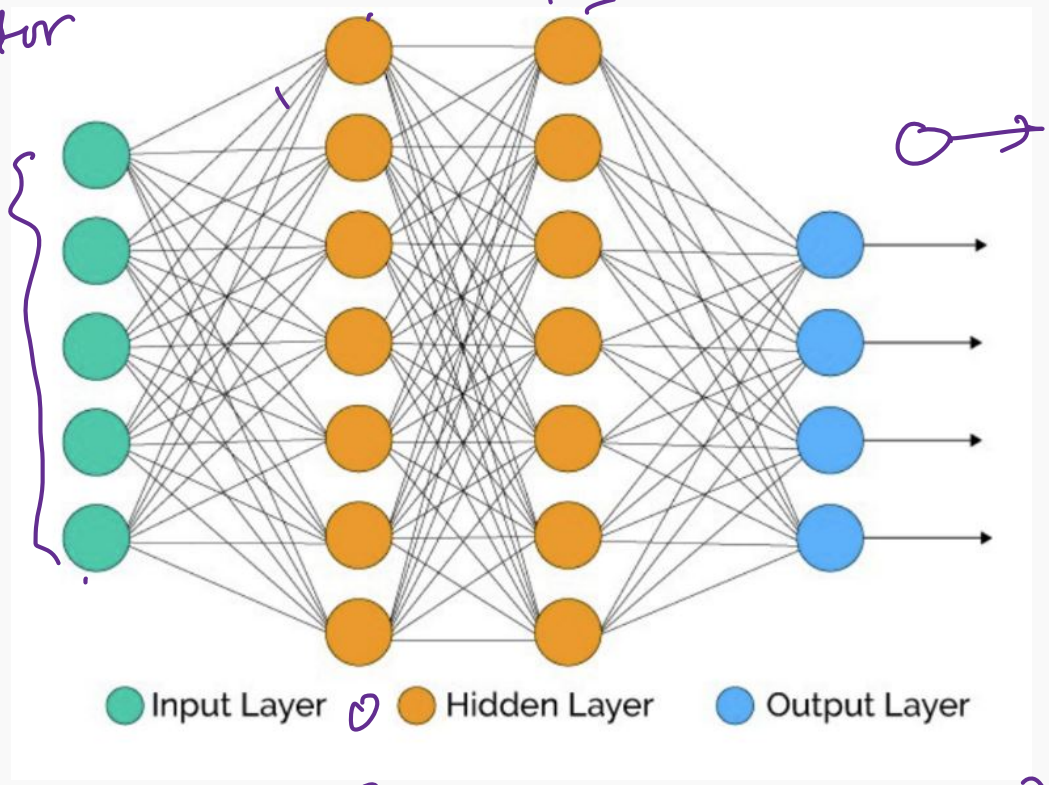
Universal Approximation Theorem



Theorem

Given sufficient data and neurons, a Neural Network can approximate any function to any desired accuracy.

Universal Approximation Theorem

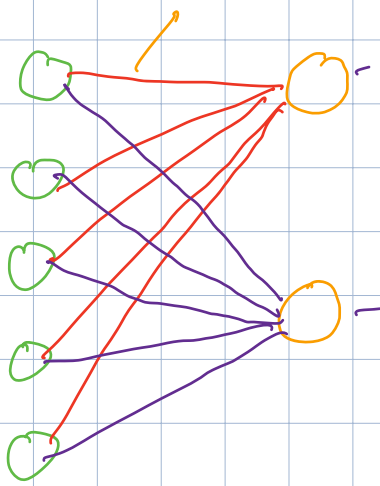


Theorem

Given sufficient data and neurons, a Neural Network can approximate any function to any desired accuracy.

---

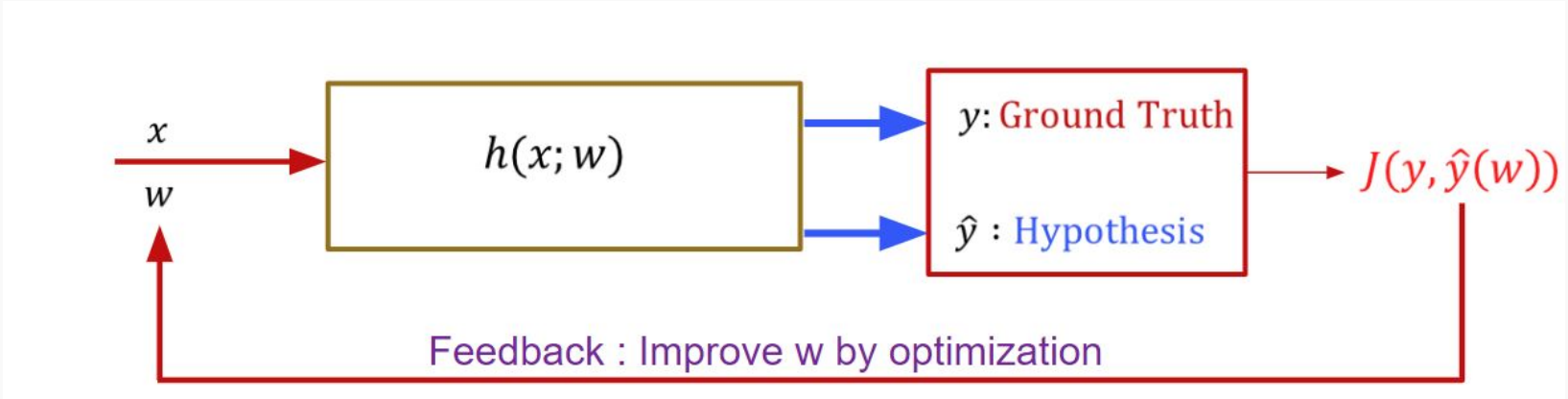
one set of weights → different values



5 + 5 = 10 unique weights

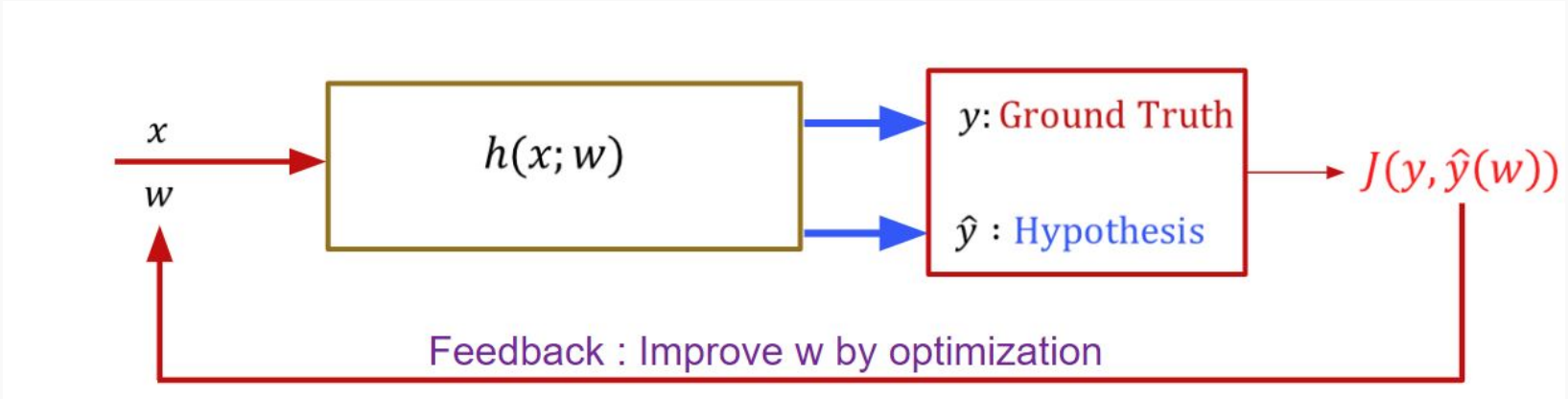
output will also be unique

Recall : Learning the parameters via feedback



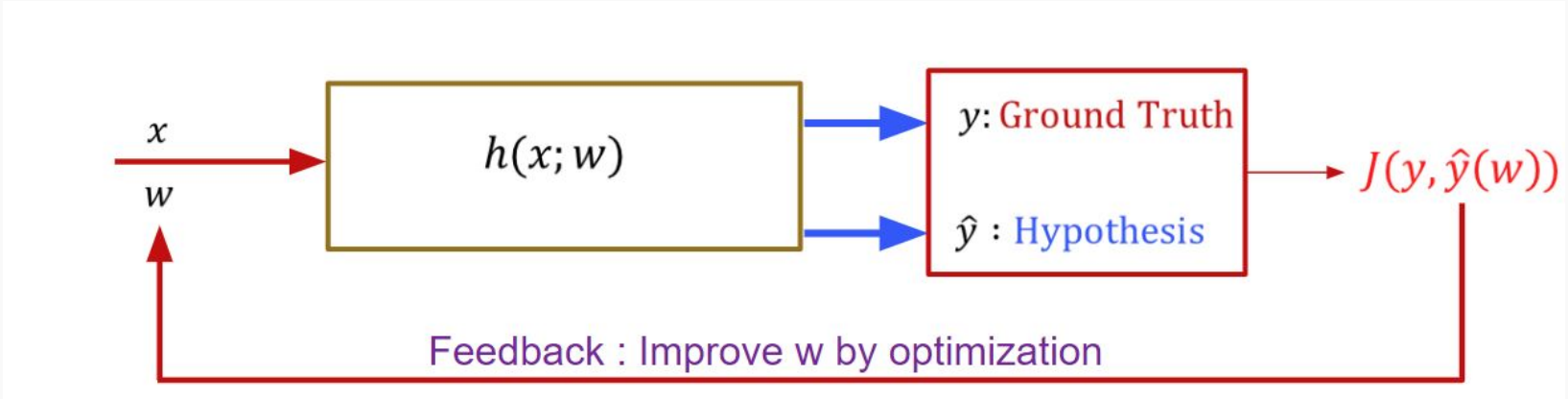
- To learn the parameters, we follow this paradigm
- Guess for the form of the hypothesis function h(x; w)
- For an arbitrary guess for w
  - We will get some $\hat{y} = h(x; w)$ which will not match the ground truth y.
  - Define a cost function $J(y, \hat{y}(w))$ depending on the difference

Recall : Learning the parameters via feedback



- To learn the parameters, we follow this paradigm
- Guess for the form of the hypothesis function h(x; w)
- For an arbitrary guess for w
  - We will get some $\hat{y} = h(x; w)$ which will not match the ground truth y.
  - Define a cost function $J(y, \hat{y}(w))$ depending on the difference
- Find optimal w by minimizing $J(w) \rightarrow$ Feedback Process
  - Requires Backpropagation for $\frac{\partial J}{\partial w}$

Recall : Learning the parameters via feedback



- To learn the parameters, we follow this paradigm
- Guess for the form of the hypothesis function h(x; w)
- For an arbitrary guess for w
  - We will get some $\hat{y} = h(x; w)$ which will not match the ground truth y.
  - Define a cost function $J(y, \hat{y}(w))$ depending on the difference
- Find optimal w by minimizing $J(w) \rightarrow$ Feedback Process
  - Requires Backpropagation for $\frac{\partial J}{\partial w}$

Even complicated Neural Networks work in exactly the same way!

Calculation in a single neuron

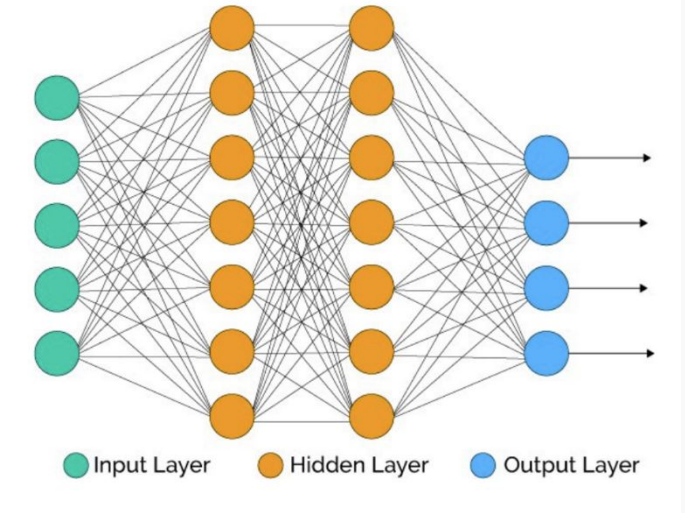
Recall : Neural Networks

Recall : Neural Networks

Neural Network model:

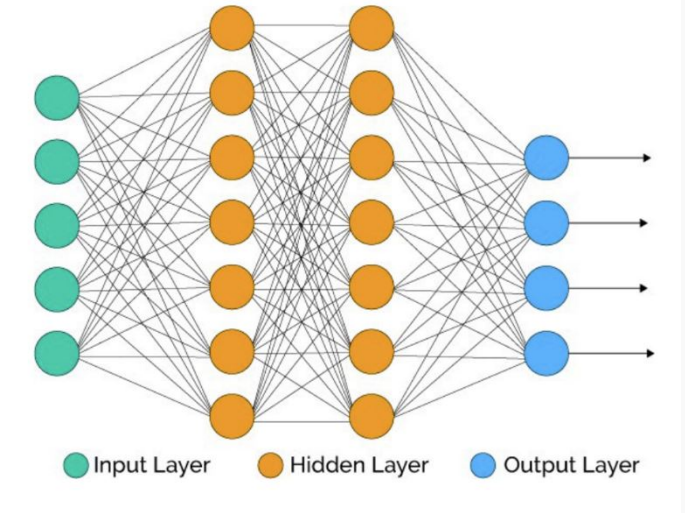
Recall : Neural Networks

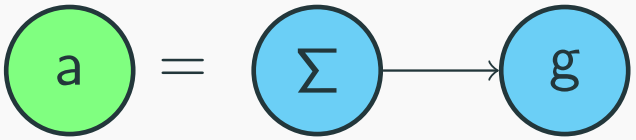
Neural Network model:



Recall : Neural Networks

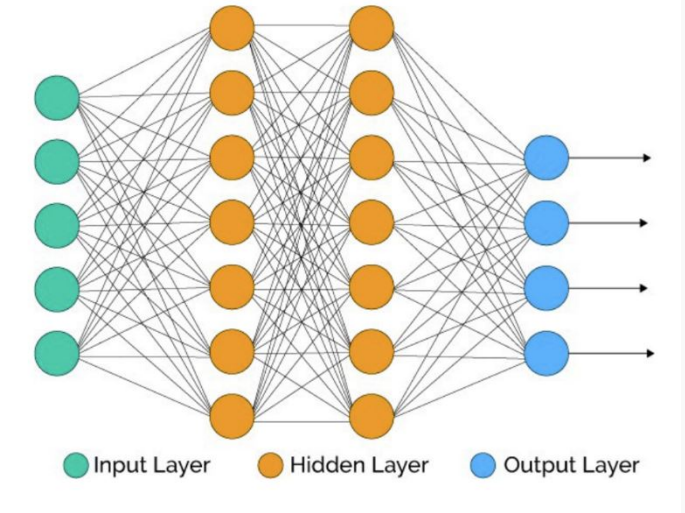
Neural Network model:



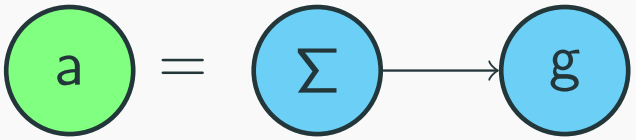


Recall : Neural Networks

Neural Network model:

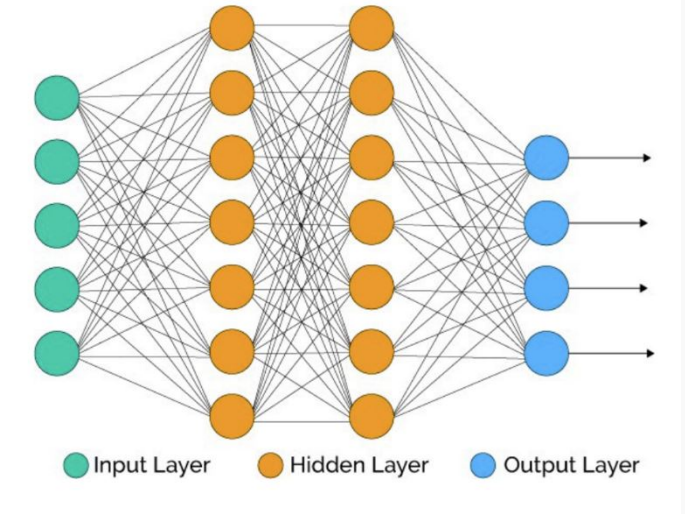


$$a = g(\sum w_i x_i)$$

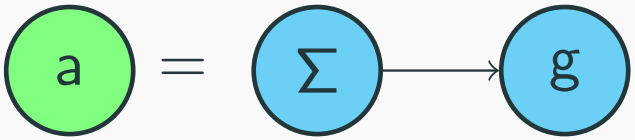


Recall : Neural Networks

Neural Network model:



$$a = g(\sum w_i x_i)$$



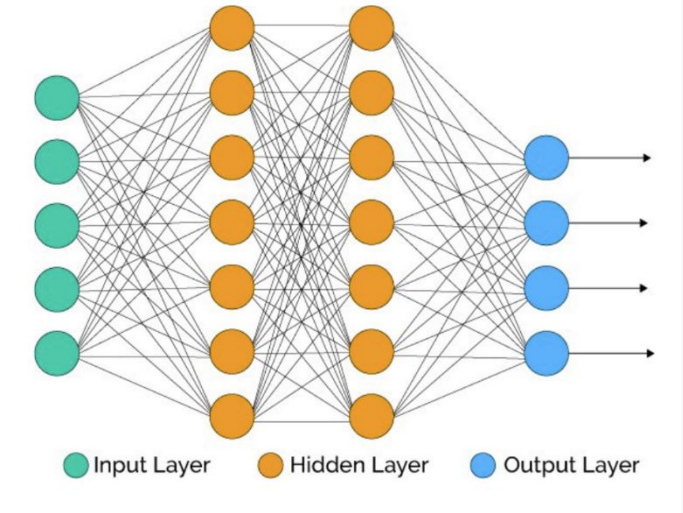
A Neuron has a linear and a nonlinear operation.

Recall : Neural Networks

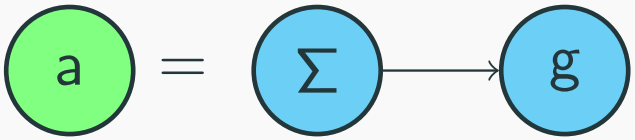
Neural Network model:

Common activation Function:

$$Sigmoid(x) = \frac{1}{1 + exp(-x)}$$



$$a = g(\sum w_i x_i)$$



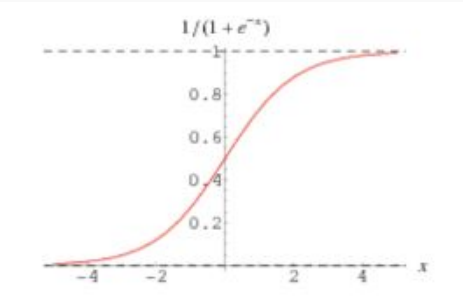
A Neuron has a linear and a nonlinear operation.

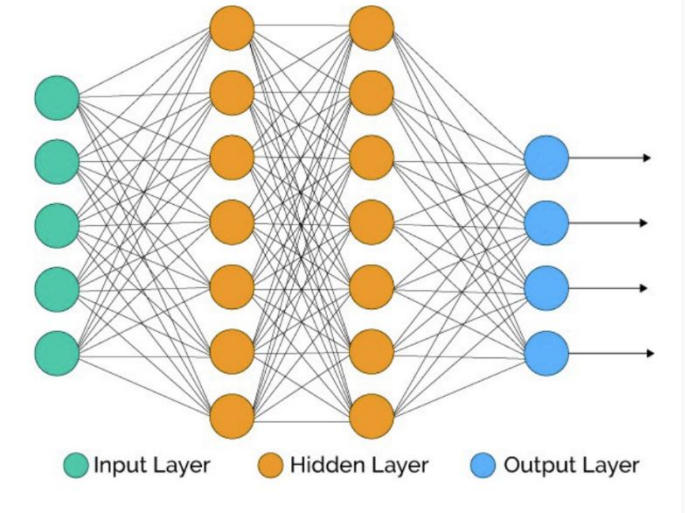
Recall : Neural Networks

Neural Network model:

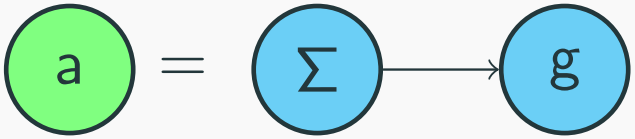
Common activation Function:

$$Sigmoid(x) = \frac{1}{1 + exp(-x)}$$





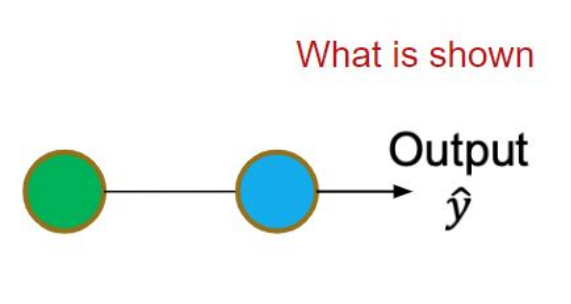
$$a = g(\sum w_i x_i)$$



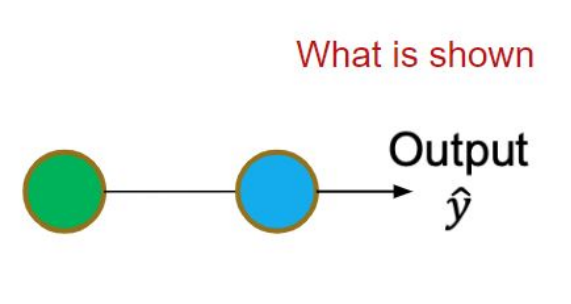
A Neuron has a linear and a nonlinear operation.

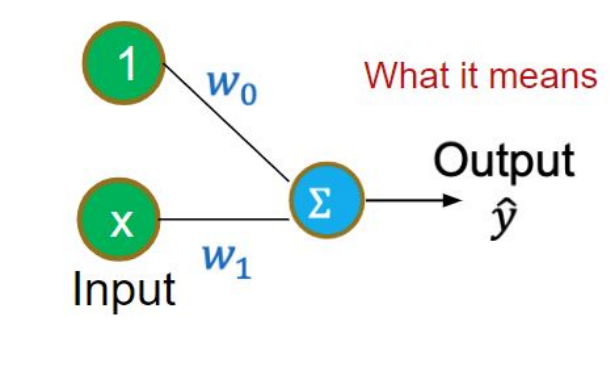
Weights and biases

Weights and biases

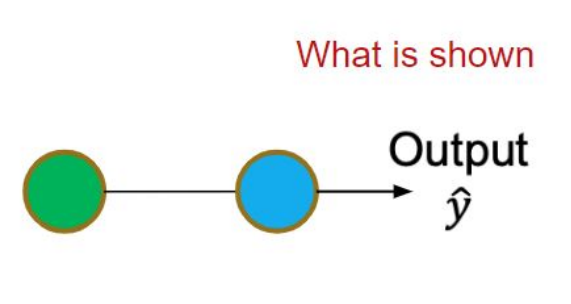


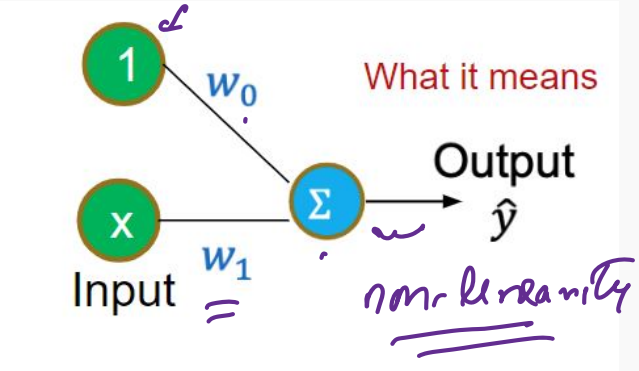
Weights and biases





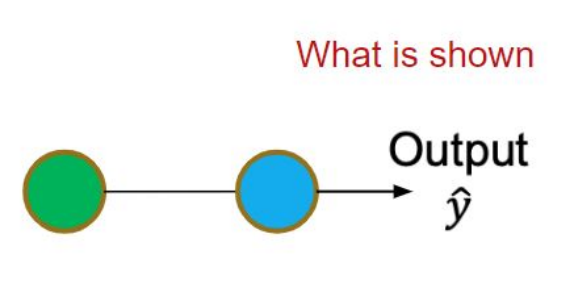
Weights and biases

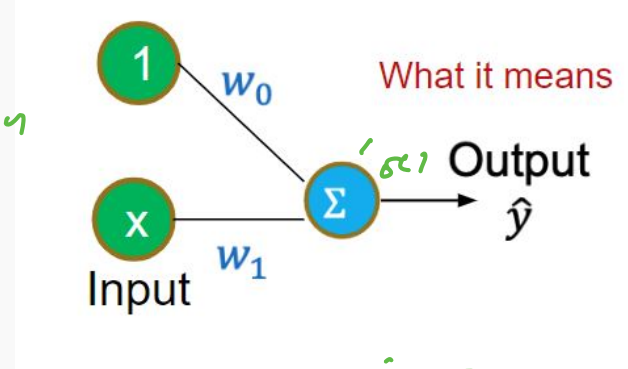


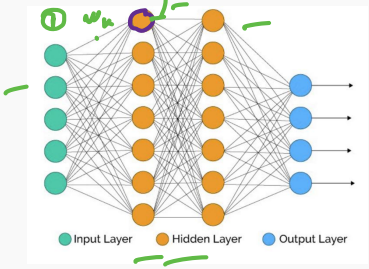


- Constant neurons representing a "feature" with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a "bias unit"
- The other parameters are called "weights"

Weights and biases



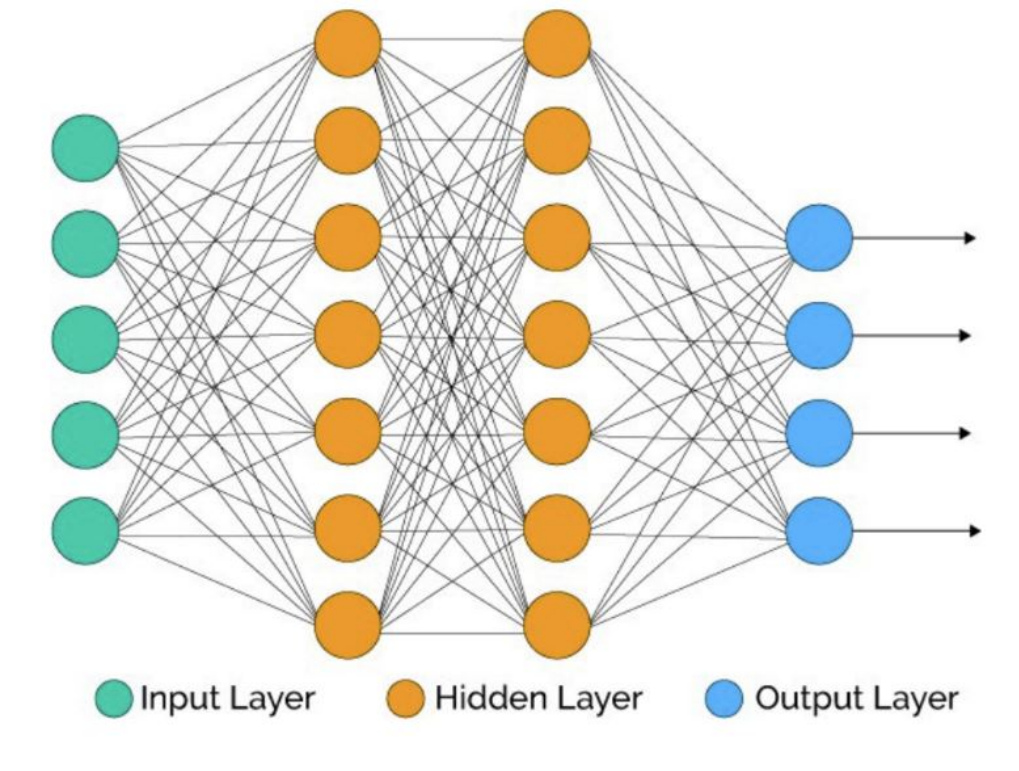




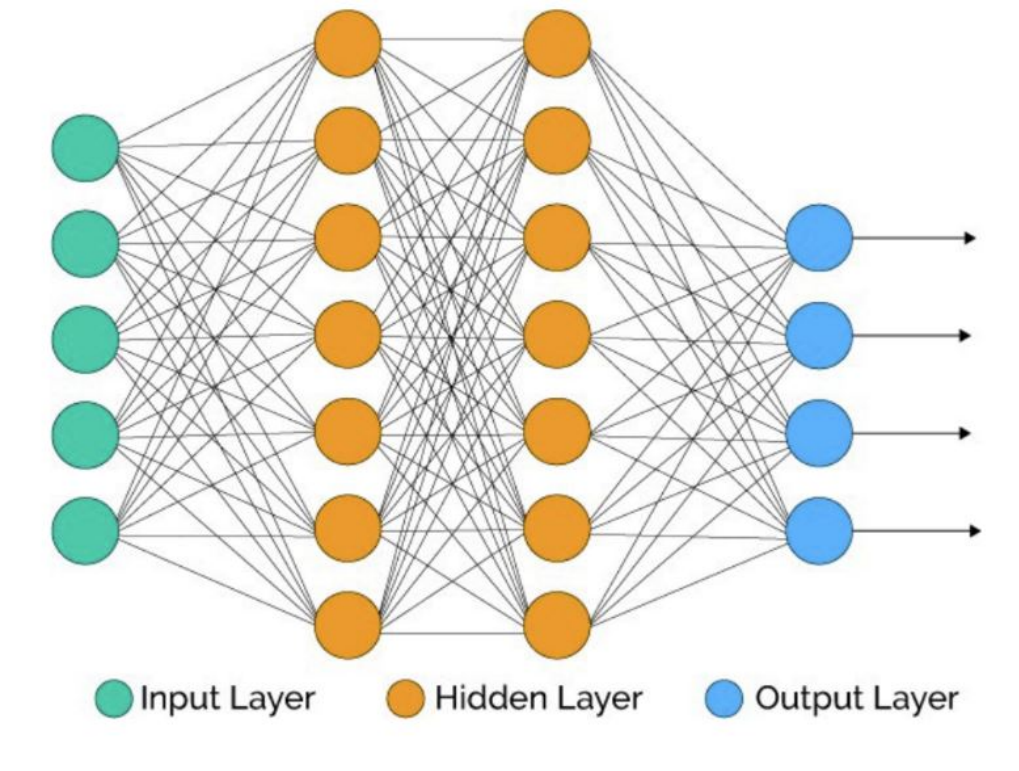
- Constant neurons representing a "feature" with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a "bias unit"
- The other parameters are called "weights"

The output of a single neuron

The output of a single neuron

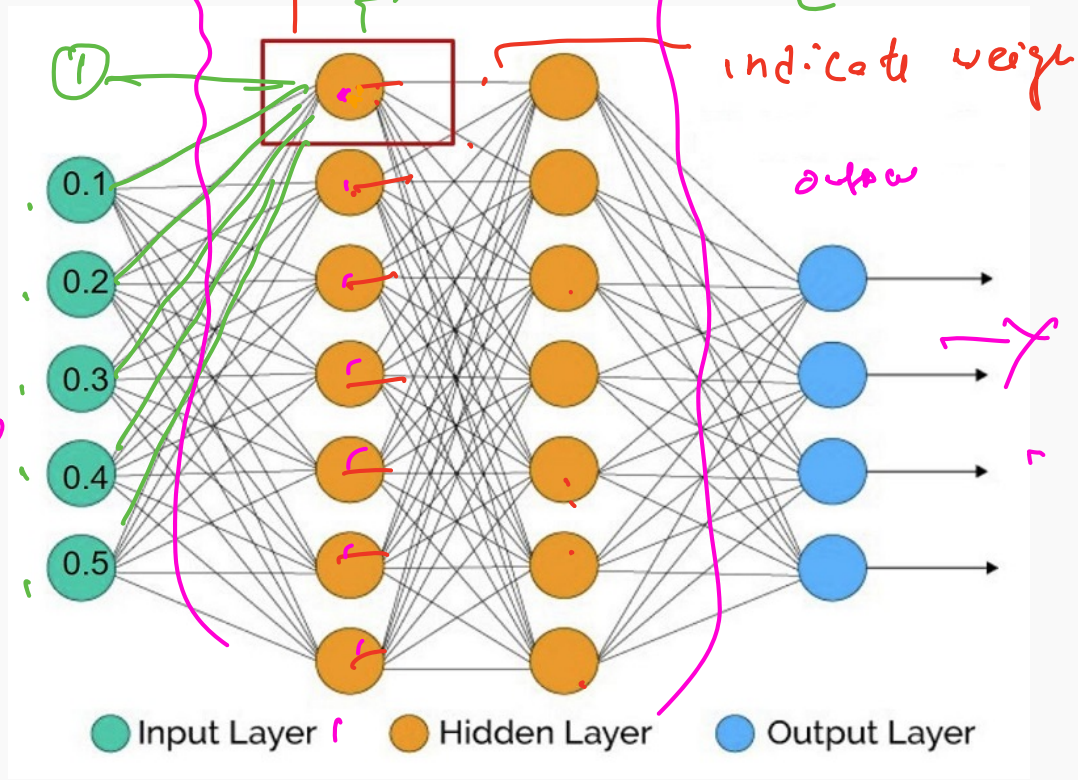


The output of a single neuron

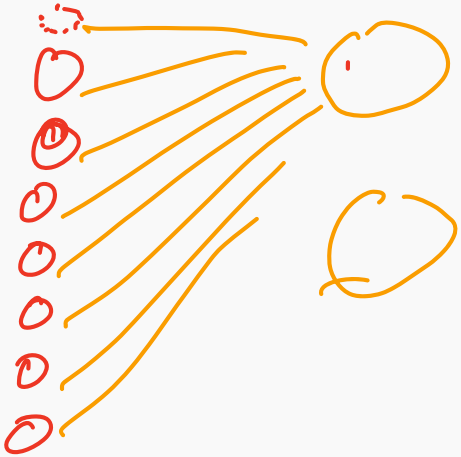


Question : If all weights are 1, then what is the output of the shown neuron?

The output of a single neuron

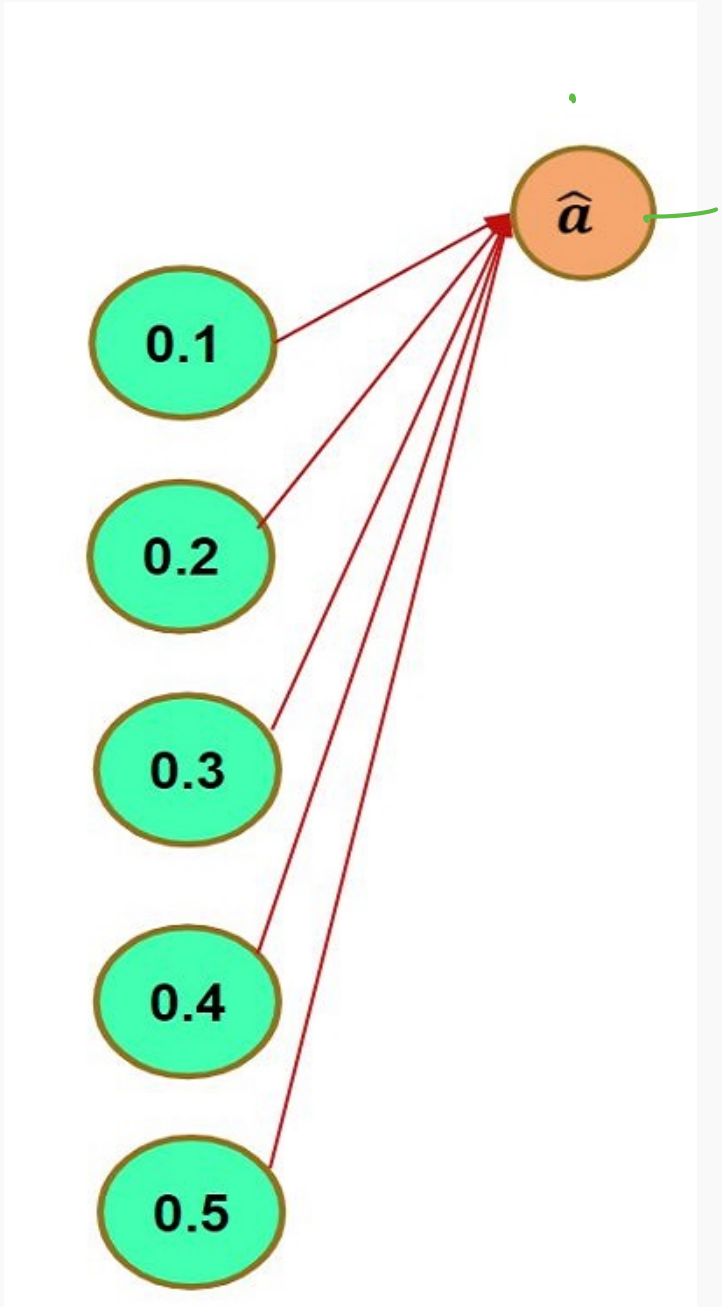


Question : If all weights are 1, then what is the output of the shown neuron?



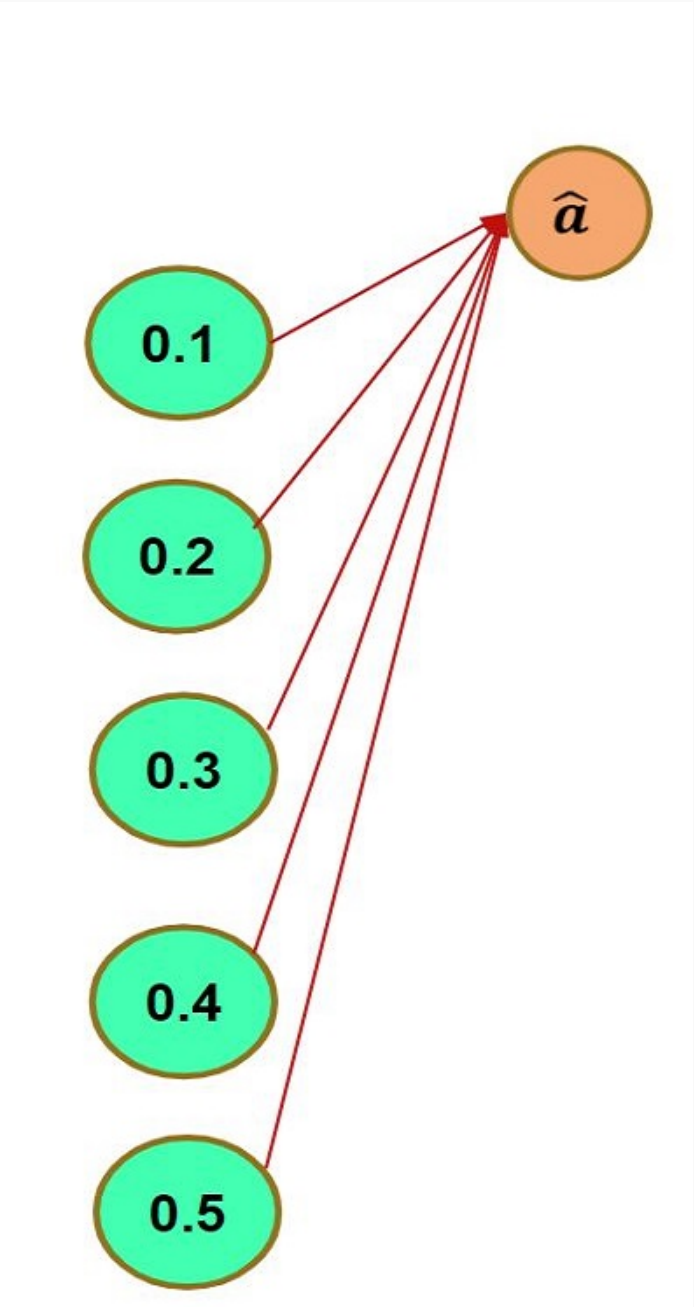
Find the output of the neuron $\hat{a}$

All weights are equal to 1



Find the output of the neuron $\hat{a}$

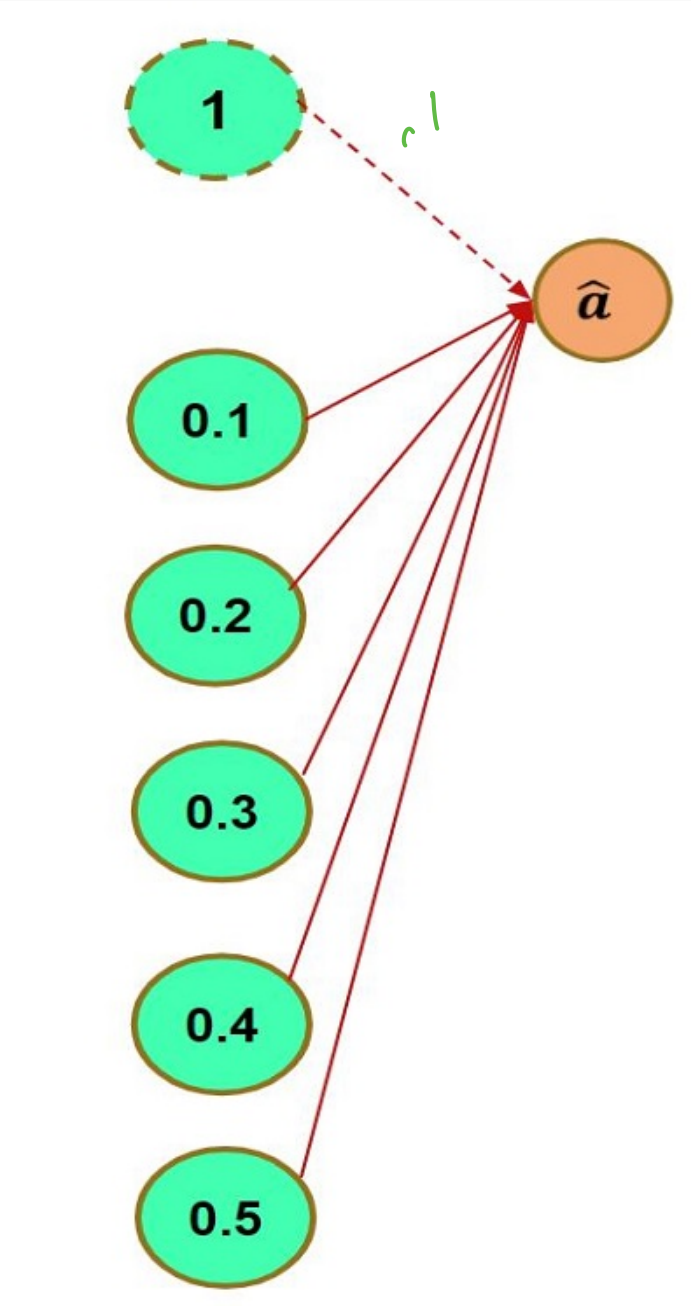
All weights are equal to 1



Find the output of the neuron $\hat{a}$

All weights are equal to 1

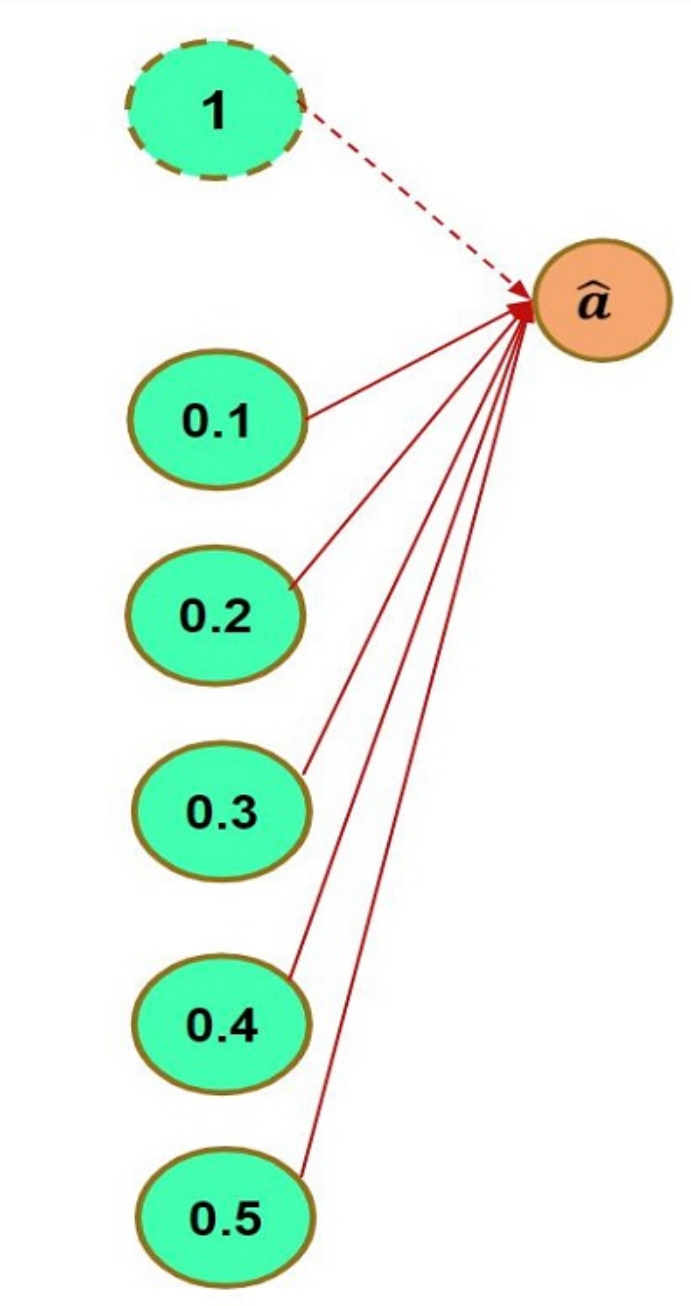
Don't forget the bias unit!



Find the output of the neuron $\hat{a}$

All weights are equal to 1

Don't forget the bias unit!

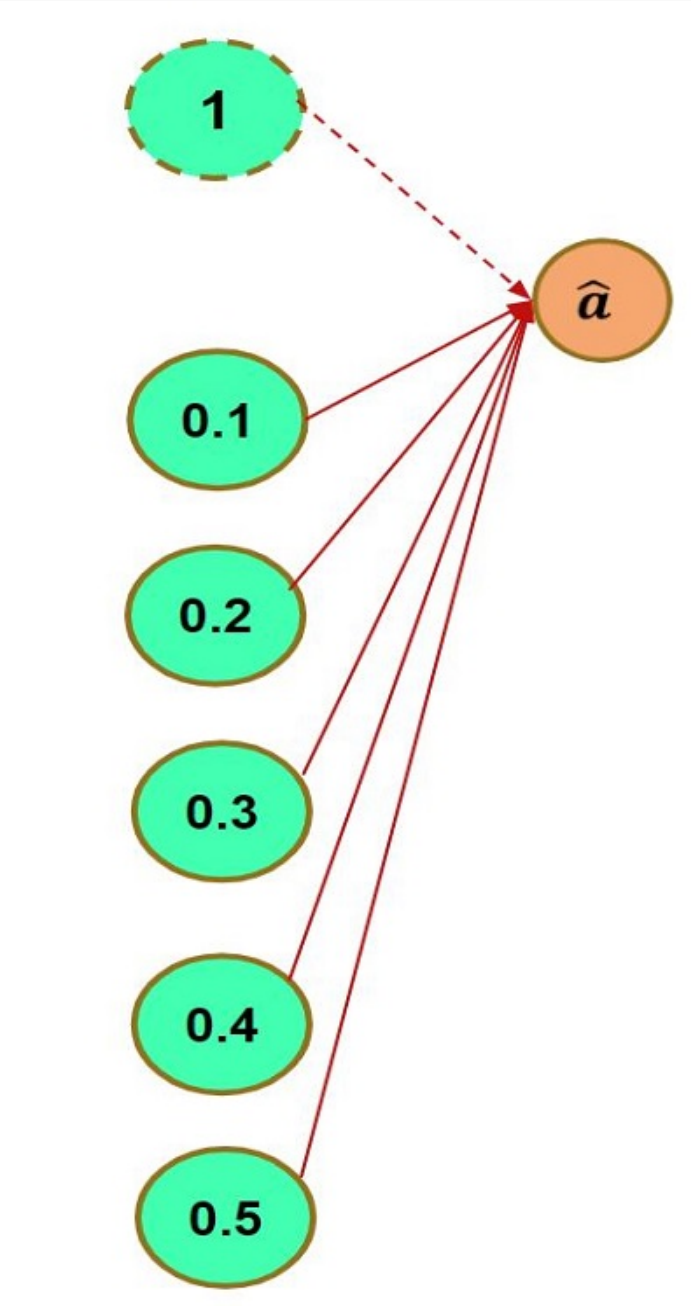


Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

Don't forget the bias unit!



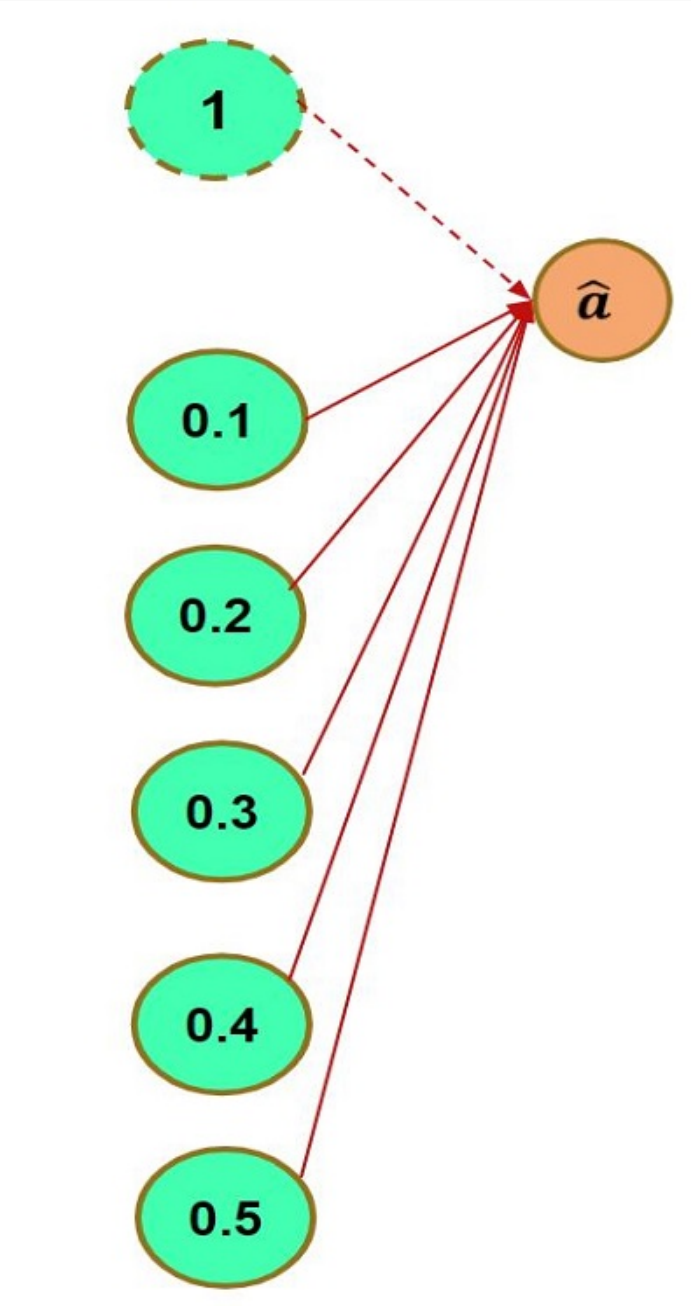
Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

Linear: $z = \sum w_i x_i$

Don't forget the bias unit!



Find the output of the neuron $\hat{a}$

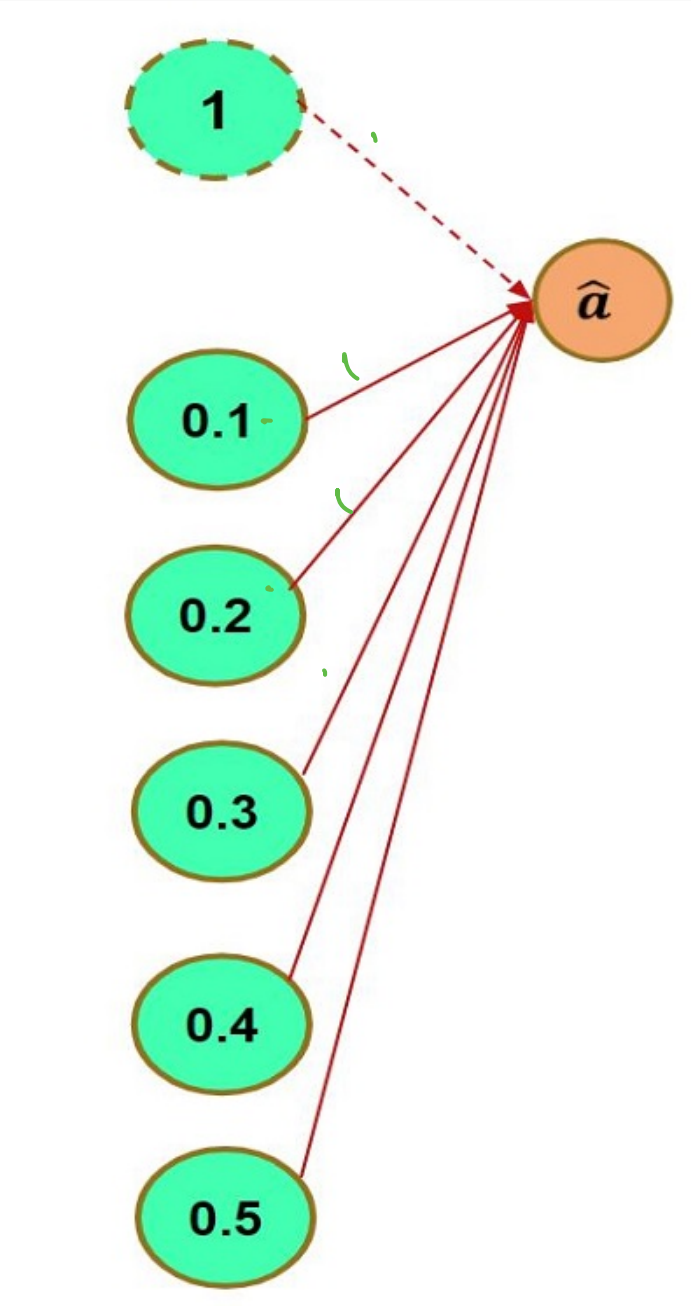
All weights are equal to 1

Ans: The output has two parts

Linear: $z = \sum w_i x_i$

$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

Don't forget the bias unit!



Find the output of the neuron $\hat{a}$

All weights are equal to 1

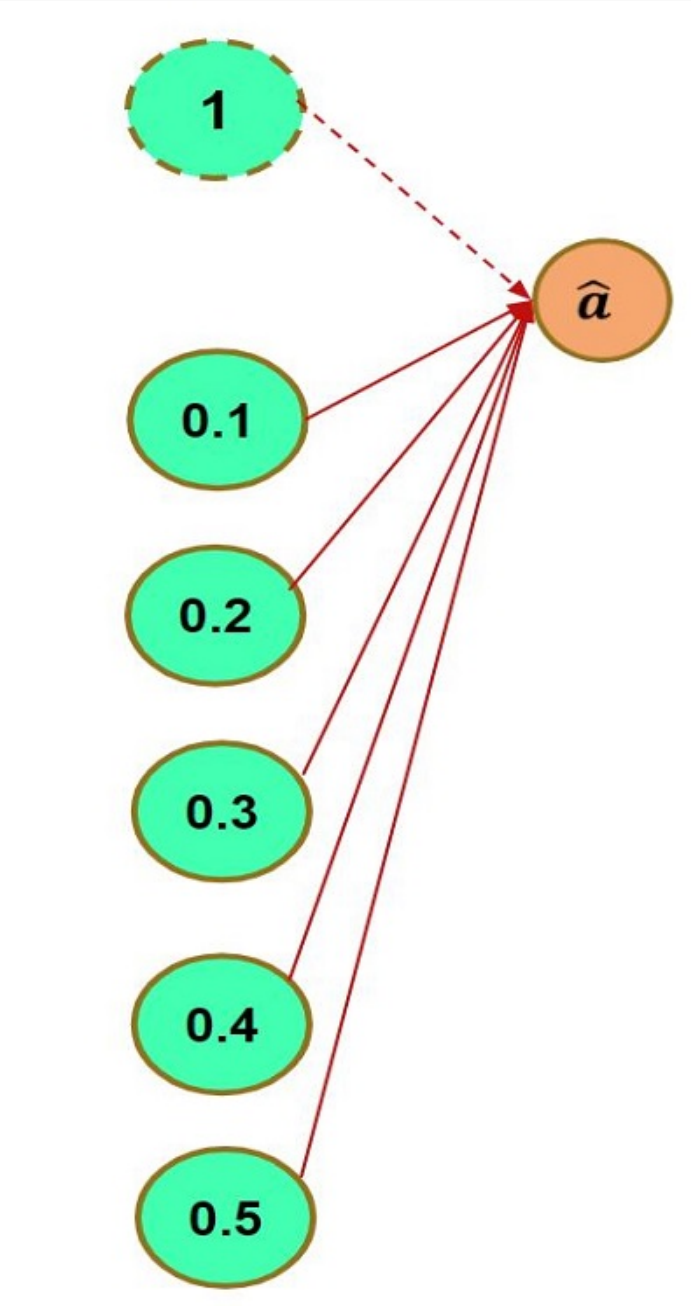
Ans: The output has two parts

Linear: $z = \sum w_i x_i$

$$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$$

$$z = 1 + 0.1 + 0.2 + 0.3 + 0.4 + 0.5 = 2.5$$

Don't forget the bias unit!



Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

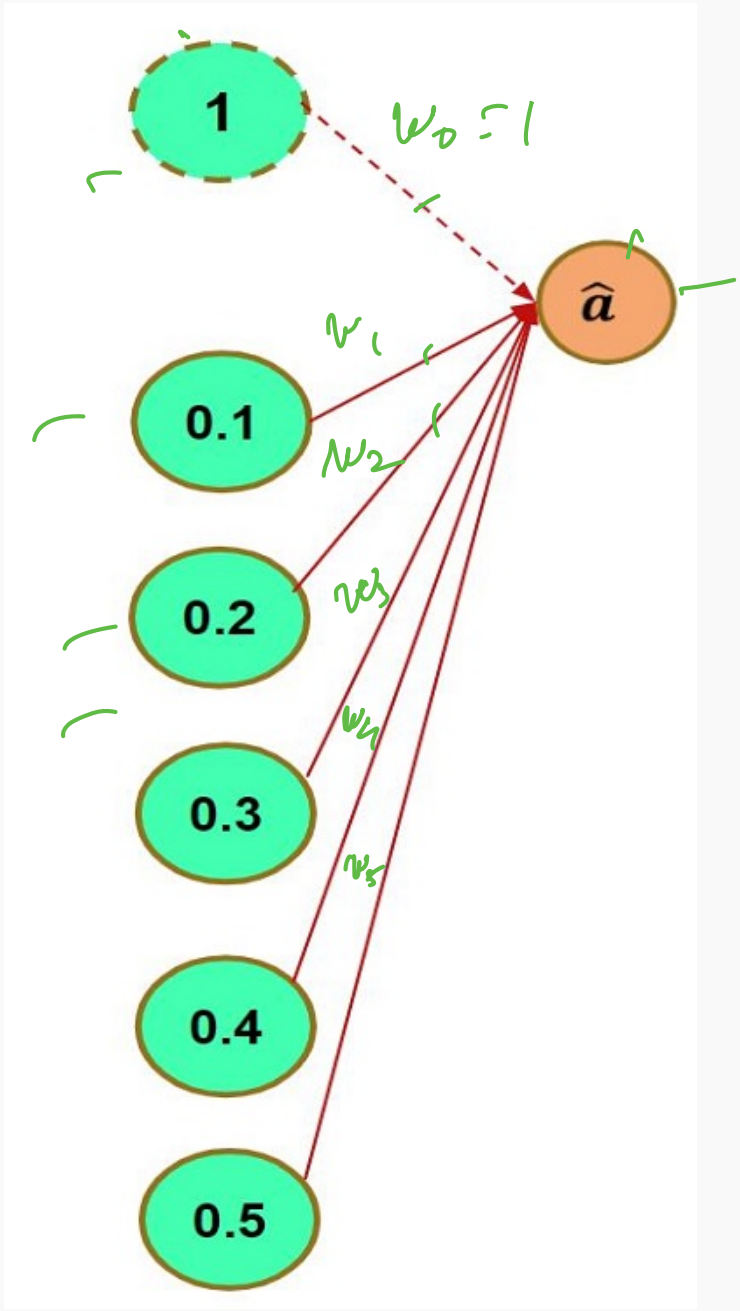
Linear: $z = \sum w_i x_i$

$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

$z = 1 + 0.1 + 0.2 + 0.3 + 0.4 + 0.5 = 2.5$

Non-linear Activation: $\hat{a} = g(z)$

Don't forget the bias unit!



$\sum wx + b'$ x_0

$x_0 \to 1$

Forward Pass

Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

Linear: $z = \sum w_i x_i$

$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

$z = 1 + 0.1 + 0.2 + 0.3 + 0.4 + 0.5 = 2.5$

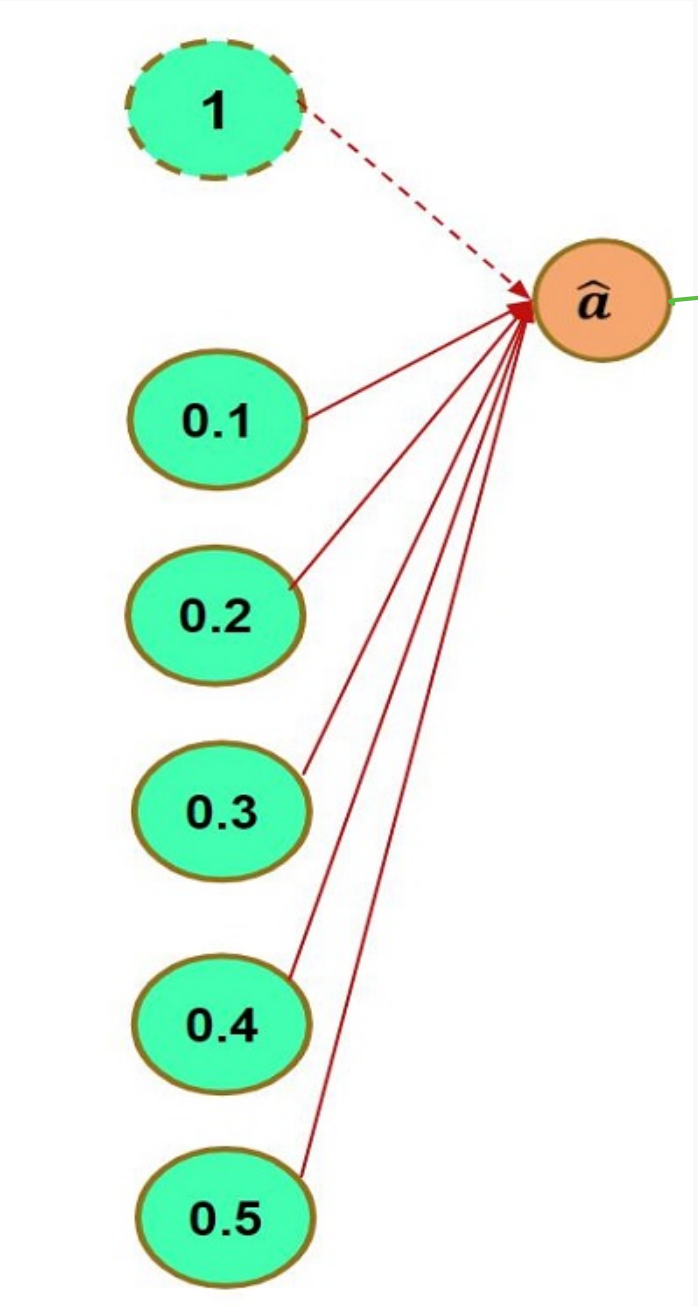
Non-linear Activation: $\hat{a} = g(z)$

$\hat{a} = \frac{1}{1+exp(-2.5)}$

$= \sum_{i=1}^{n} w_i x_i + w_0$

$= \sum_{i=0}^{n} w_i x_i \quad x_0 = 1$

Don't forget the bias unit!



Find the output of the neuron $\hat{a}$

All weights are equal to 1

Ans: The output has two parts

Linear: $z = \sum w_i x_i$ → output

$z = w_0 x_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$

$z = 1 + 0.1 + 0.2 + 0.3 + 0.4 + 0.5 = 2.5$

Non-linear Activation: $\hat{a} = g(z)$

$\hat{a} = \frac{1}{1+exp(-2.5)}$ → output

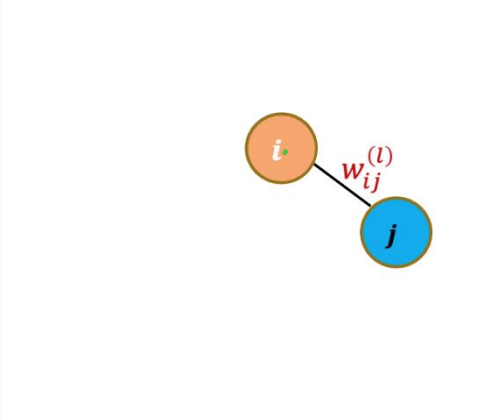
$\hat{a} = 0.9241$

= hidden layer

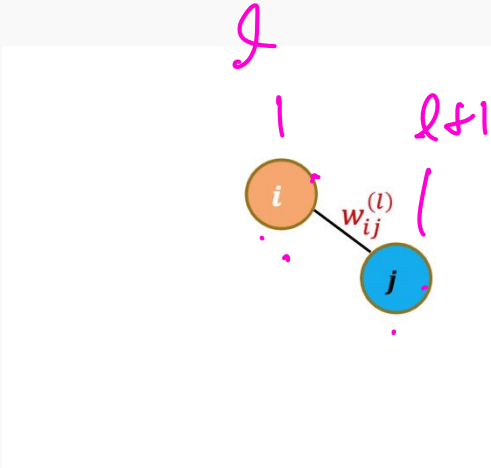
Don't forget the bias unit!

Weights notation

Weights notation

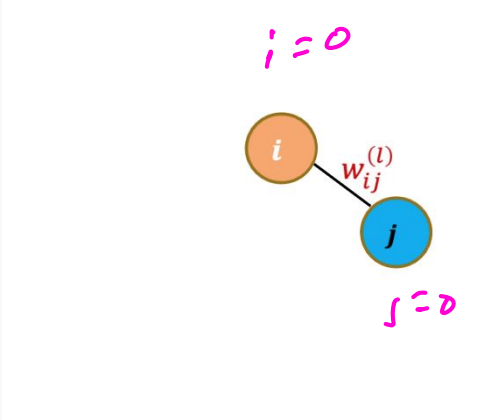


Weights notation



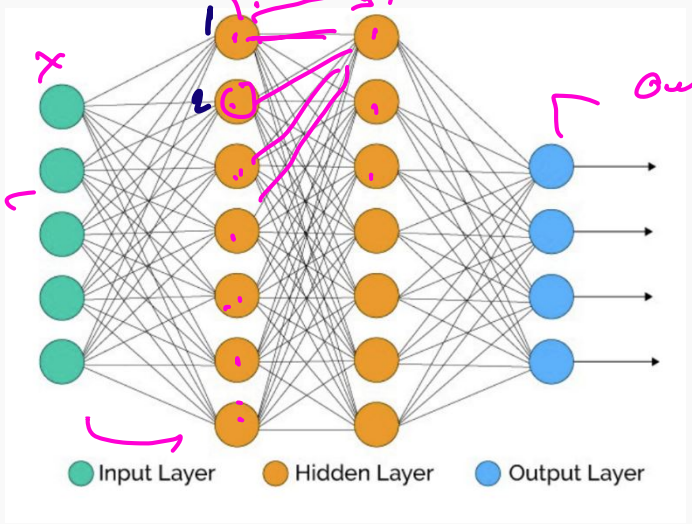
- $w_{ij}^{(l)}$ is the weight connecting Neuron i of Layer l Neuron j of Layer $l+1$.

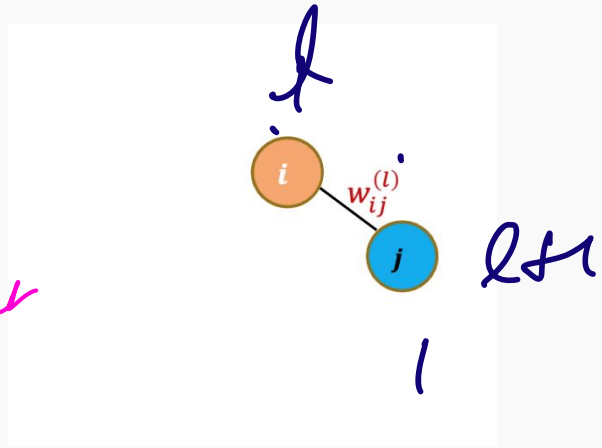
Weights notation



- $w_{ij}^{(l)}$ is the weight connecting Neuron i of Layer l Neuron j of Layer $l+1$.
- bias units are the 0^{th} neurons.

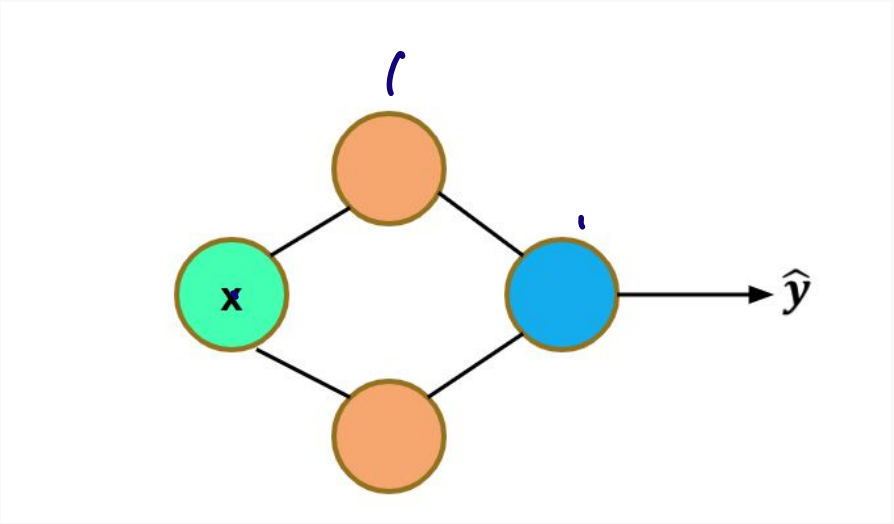
Weights notation





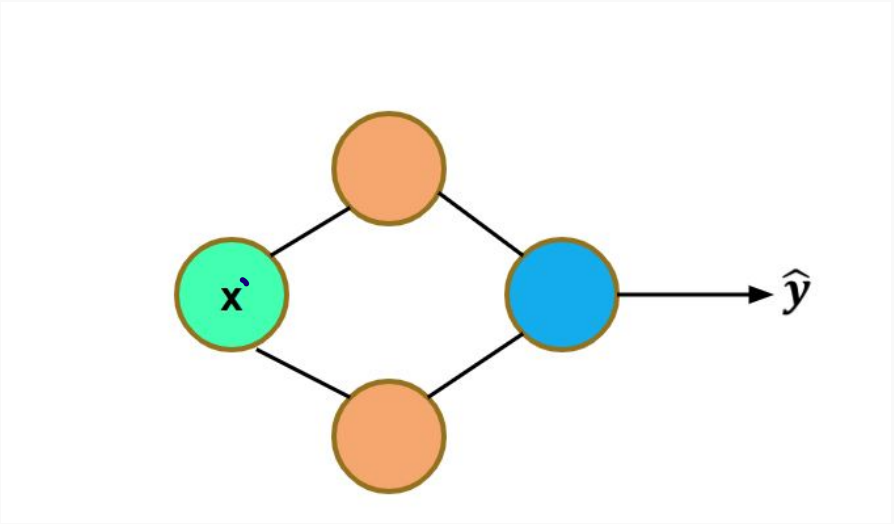
- $w_{ij}^{(l)}$ is the weight connecting Neuron i of Layer l Neuron j of Layer $l+1$.
- bias units are the 0^{th} neurons.

A simple forward pass calculation



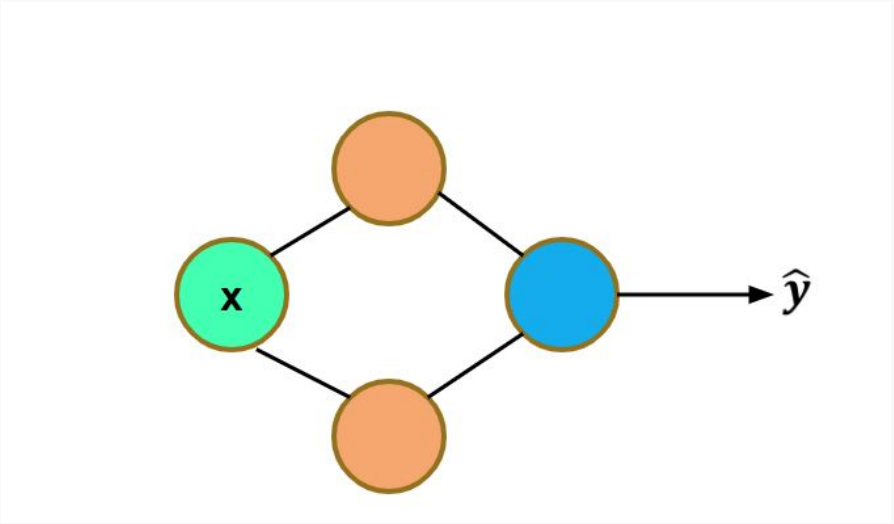
Question: Find the output of the given network for the following.

Univariate input
1 – hidden layer
1 – output



Question: Find the output of the given network for the following.

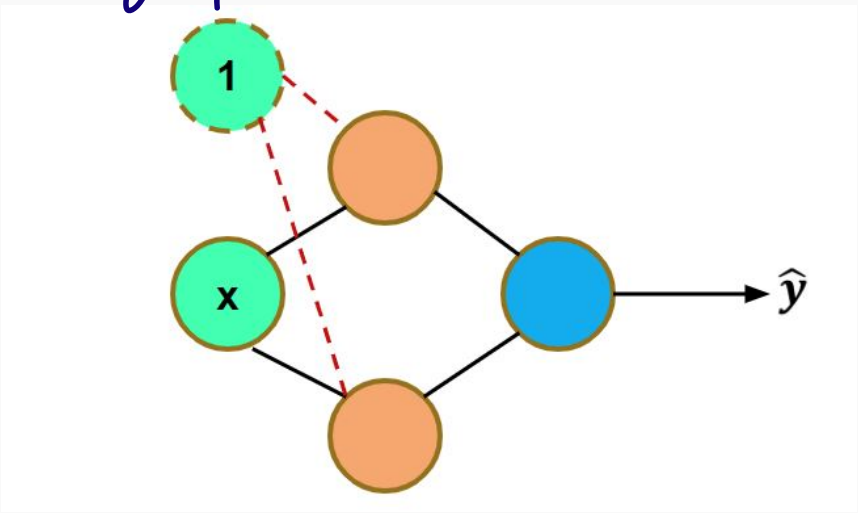
$$x = 0.5$$



Question: Find the output of the given network for the following.

$$x = 0.5$$

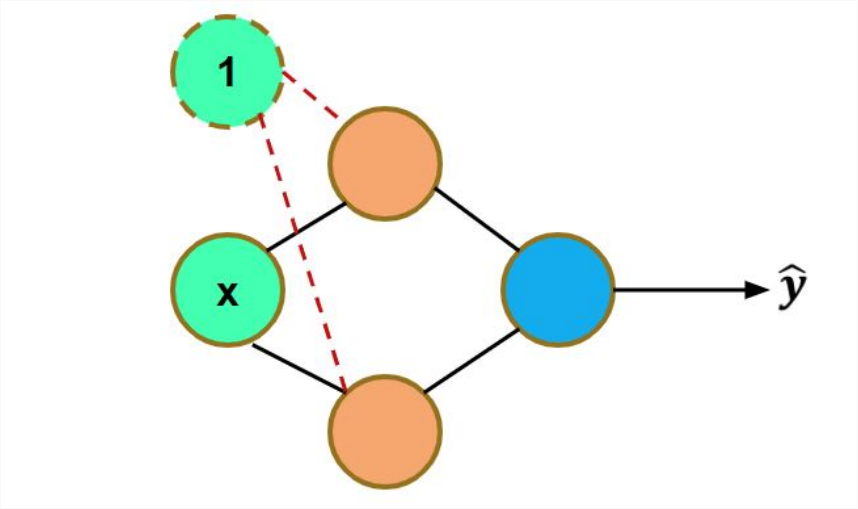
$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$



Question: Find the output of the given network for the following.

$$x = 0.5$$

$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$

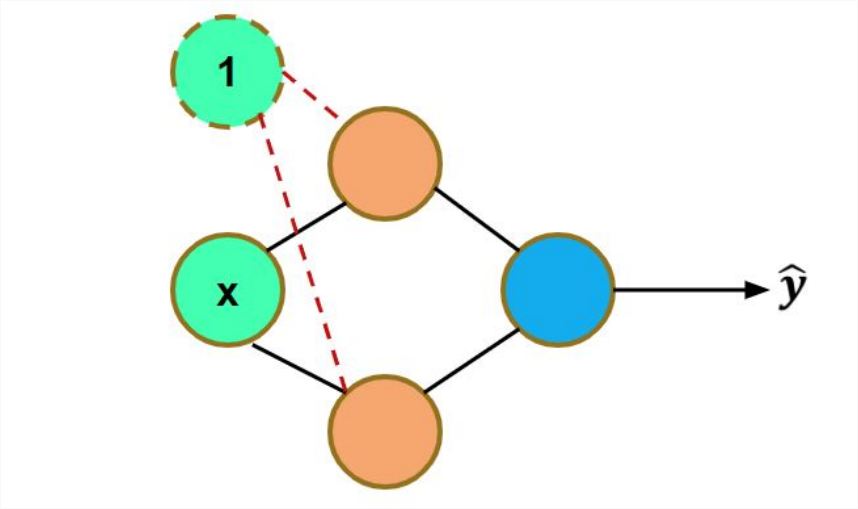


Question: Find the output of the given network for the following.

$$x = 0.5$$

$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$

$$w_{11}^{(1)} = 0.7 \; w_{12}^{(1)} = 0.9$$



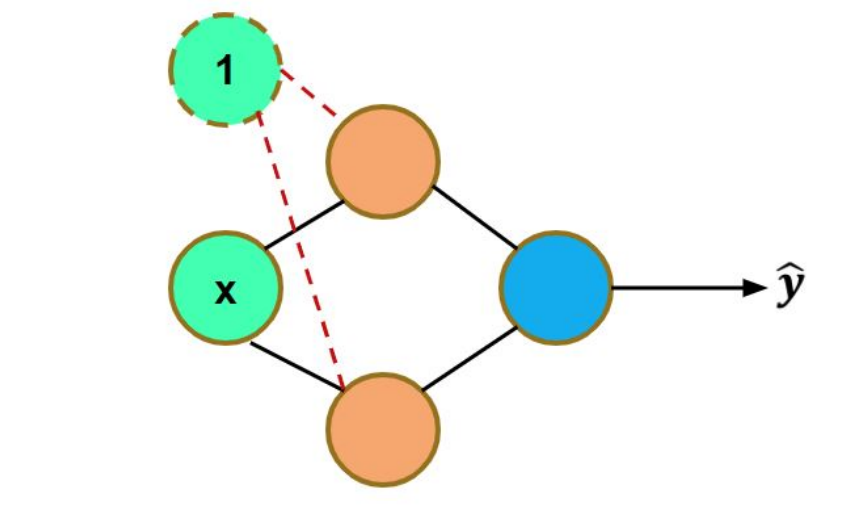
Question: Find the output of the given network for the following.

$$x = 0.5$$

$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$

$$w_{11}^{(1)} = 0.7 \; w_{12}^{(1)} = 0.9$$

$$w_{01}^{(2)} = 1$$



Question: Find the output of the given network for the following.

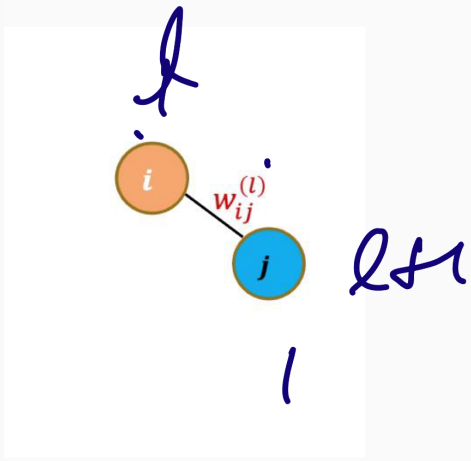
$$x = 0.5$$

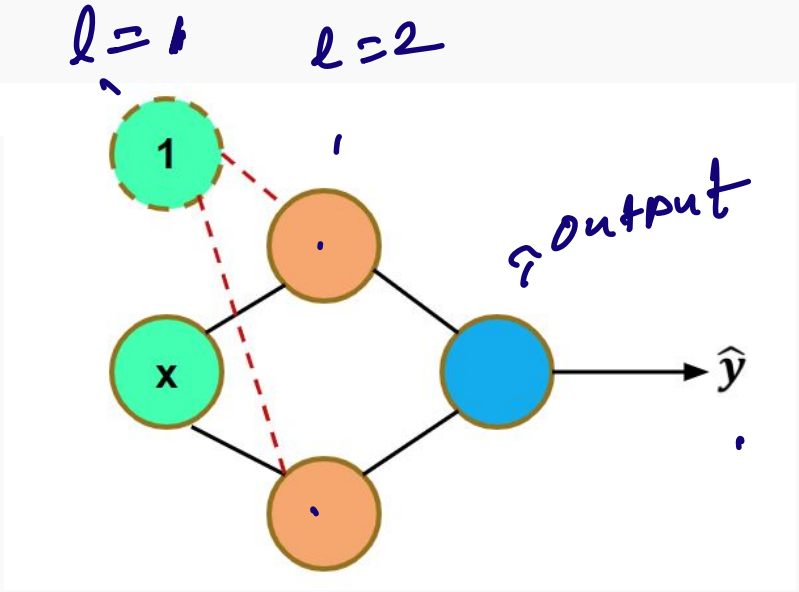
$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$

$$w_{11}^{(1)} = 0.7 \; w_{12}^{(1)} = 0.9$$

$$w_{01}^{(2)} = 1$$

$$w_{11}^{(2)} = 1 \; w_{21}^{(2)} = 0.9$$





Question: Find the output of the given network for the following.

$$x = 0.5$$

$$w_{01}^{(1)} = 1.0 \; w_{02}^{(1)} = 0.8$$

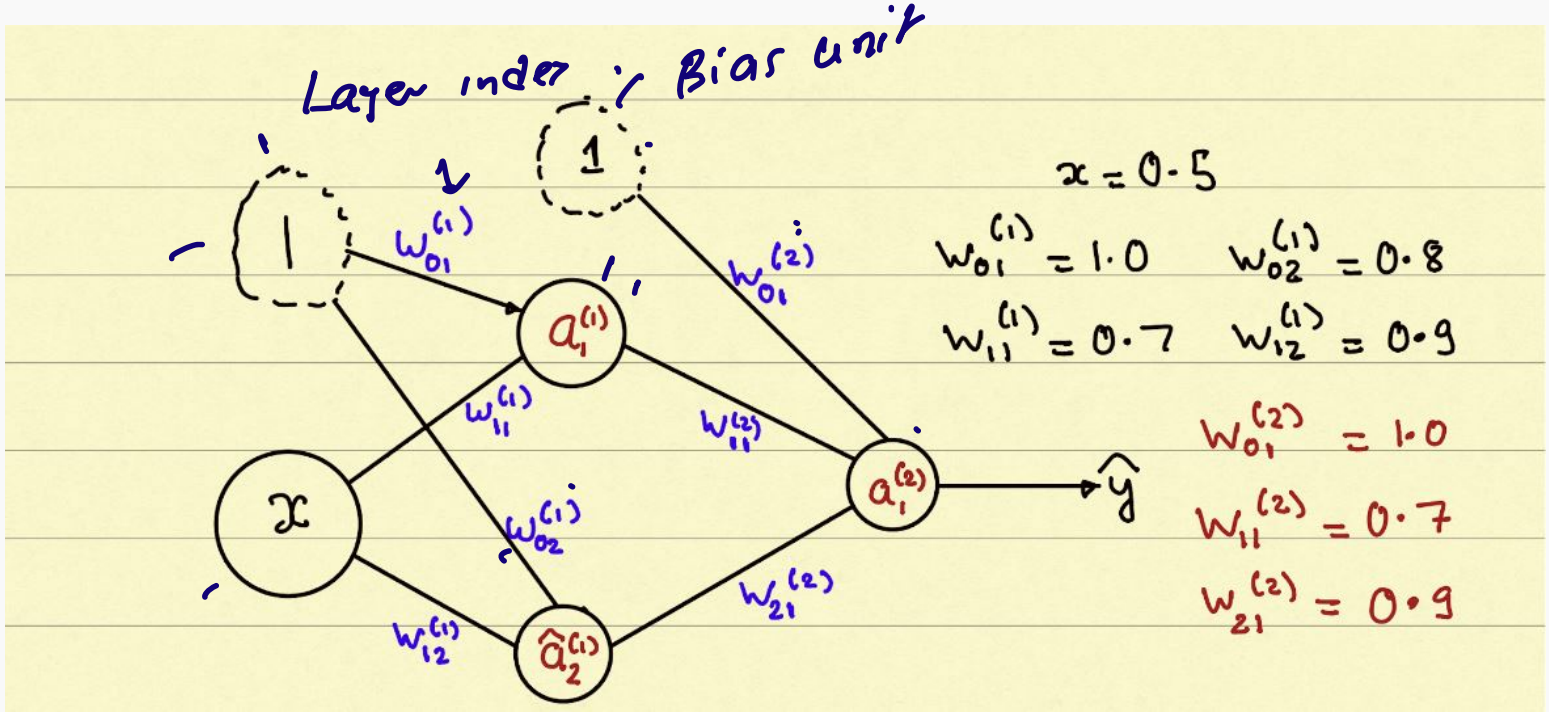
$$w_{11}^{(1)} = 0.7 \; w_{12}^{(1)} = 0.9$$

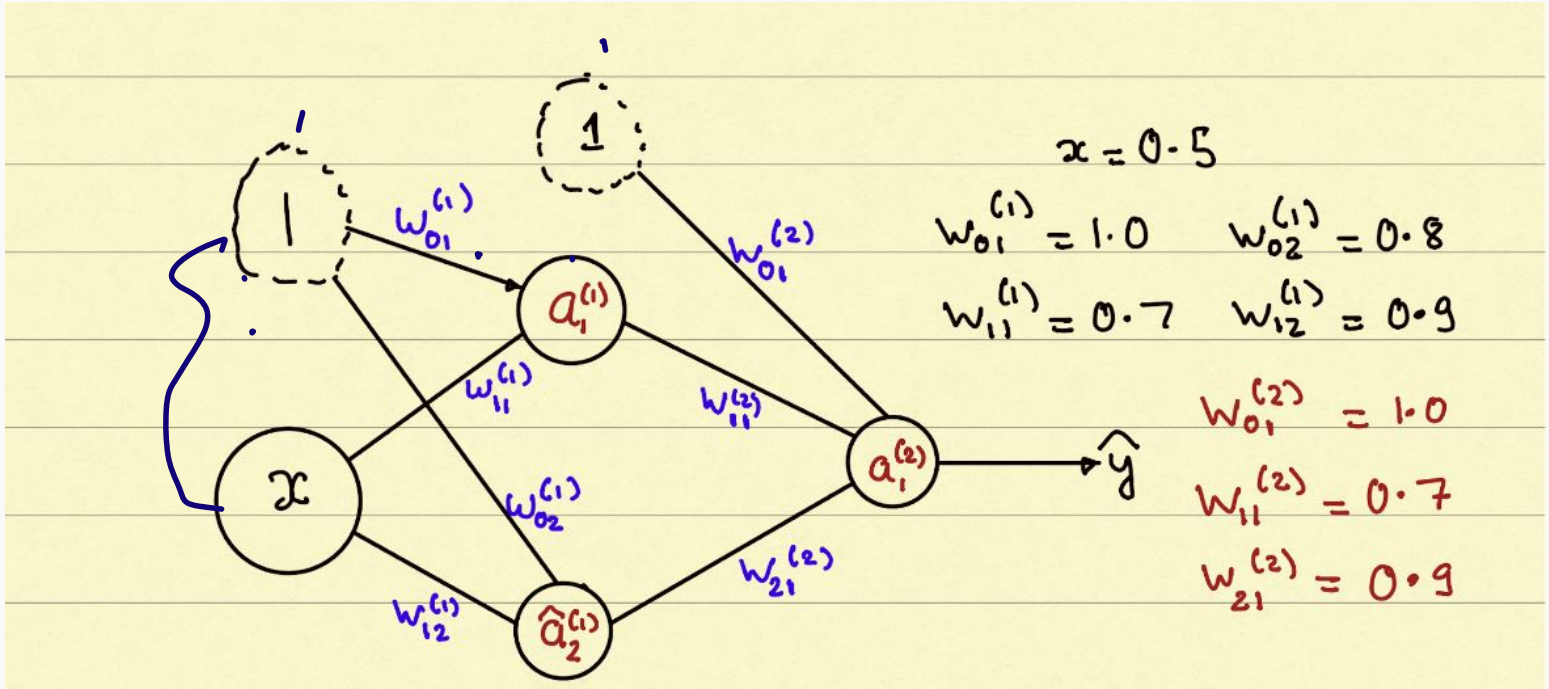
$$w_{01}^{(2)} = 1$$

$$w_{11}^{(2)} = 1 \; w_{21}^{(2)} = 0.9$$

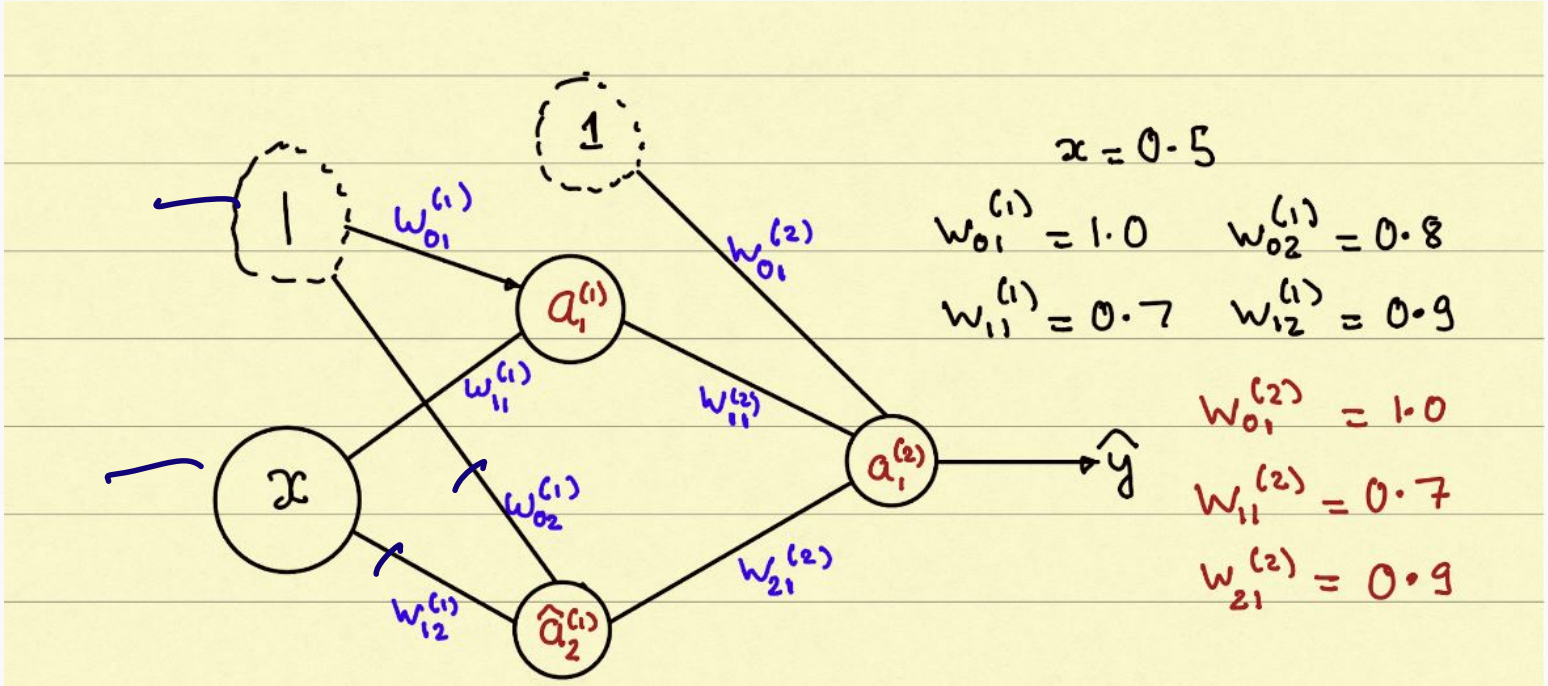
Assume that the activation function in the hidden layer and the output layer is the sigmoid

SOLUTION


Layer index
Bias unit
1
1
$w_{01}^{(1)}$
$w_{01}^{(2)}$
$a_1^{(1)}$
$w_{11}^{(1)}$
$w_{11}^{(2)}$
x
$a_1^{(2)}$
$\hat{y}$
$w_{02}^{(1)}$
$w_{21}^{(2)}$
$w_{12}^{(1)}$
$\hat{a}_2^{(1)}$
$x = 0.5$
$w_{01}^{(1)} = 1.0$
$w_{02}^{(1)} = 0.8$
$w_{11}^{(1)} = 0.7$
$w_{12}^{(1)} = 0.9$
$w_{01}^{(2)} = 1.0$
$w_{11}^{(2)} = 0.7$
$w_{21}^{(2)} = 0.9$

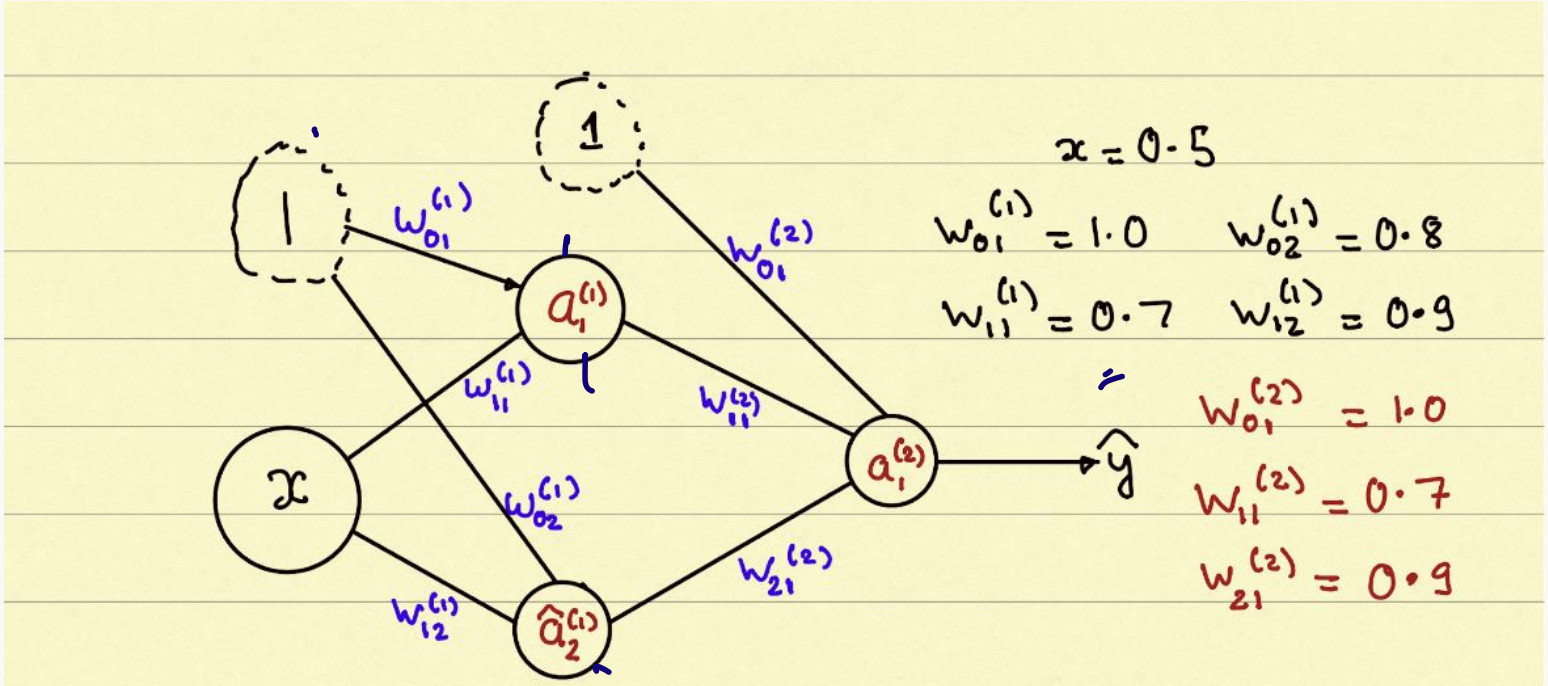


$$\hat{a}_1^{(1)} = g(z_1^{(1)}) = g(w_{01}^{(1)} x_0 + w_{11}^{(1)} x)$$



$$\hat{a}_1^{(1)} = g(z_1^{(1)}) = g(w_{01}^{(1)} x_0 + w_{11}^{(1)} x)$$

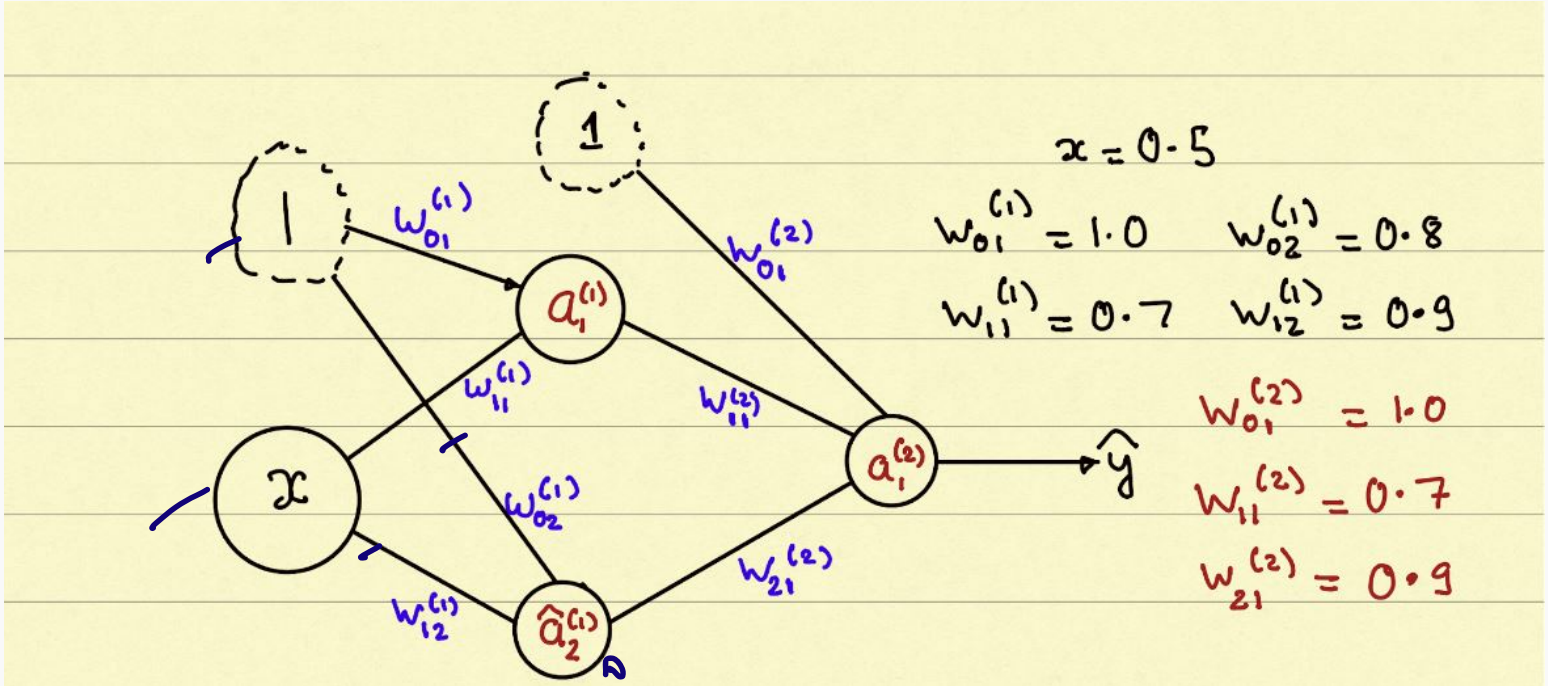
$$= g(1 \times 1 + 0.7 \times 0.5) = g(1.35)$$



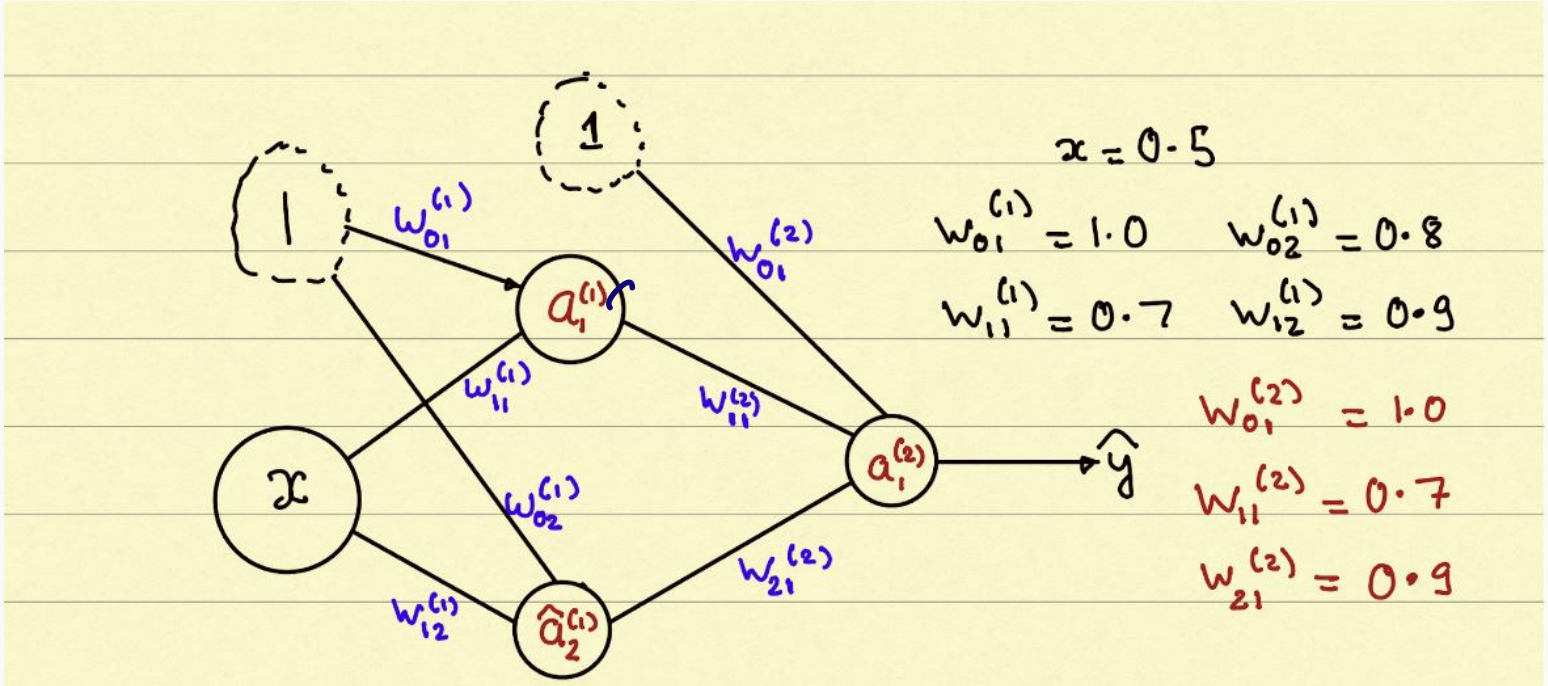
$$\hat{a}_1^{(1)} = g(z_1^{(1)}) = g(w_{01}^{(1)} x_0 + w_{11}^{(1)} x)$$

$$= g(1 \times 1 + 0.7 \times 0.5) = g(1.35)$$

$$\hat{a}_1^{(1)} = 0.7941$$

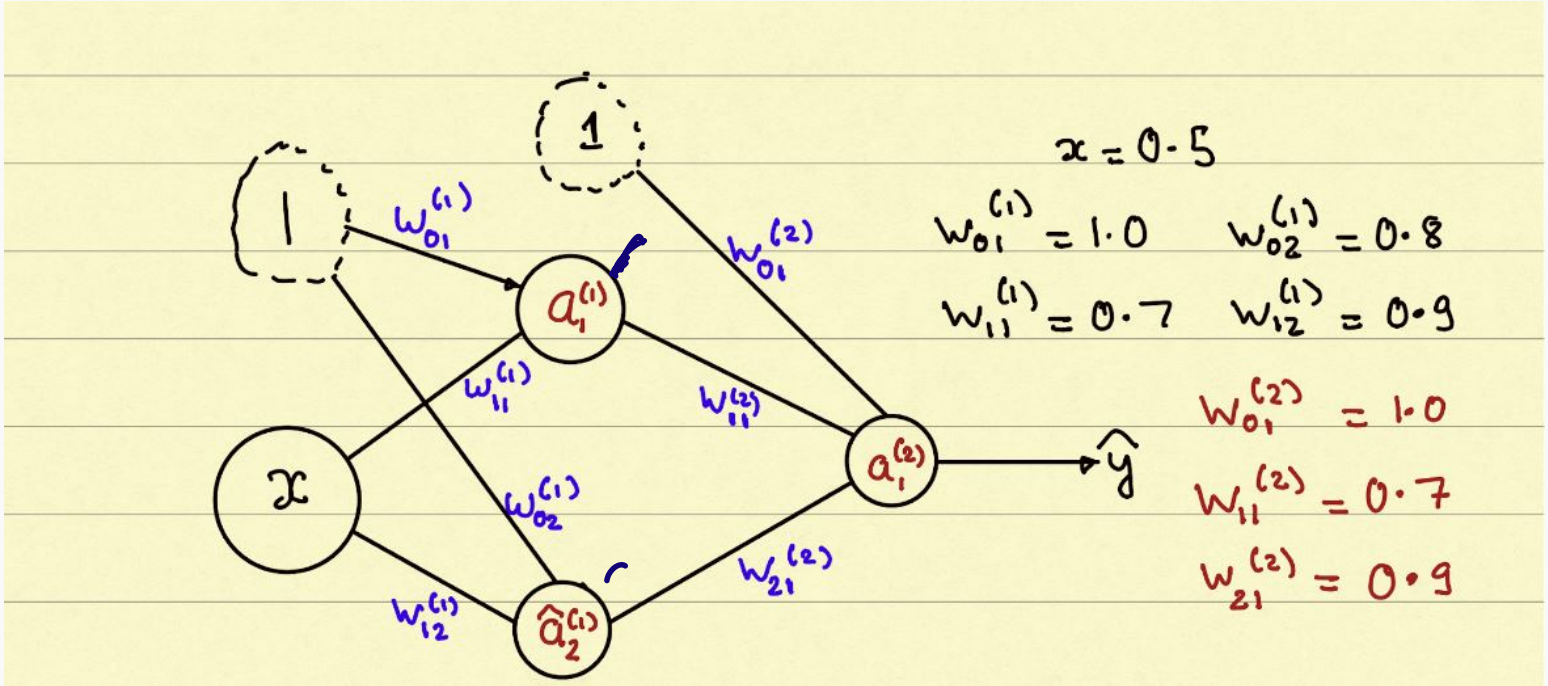


$$\hat{a}_2^{(1)} = g(z_2^{(1)}) = g(w_{02}^{(1)} x_0 + w_{12}^{(1)} x_1)$$



$$\hat{a}_2^{(1)} = g(z_2^{(1)}) = g(w_{02}^{(1)} x_0 + w_{12}^{(1)} x_1)$$

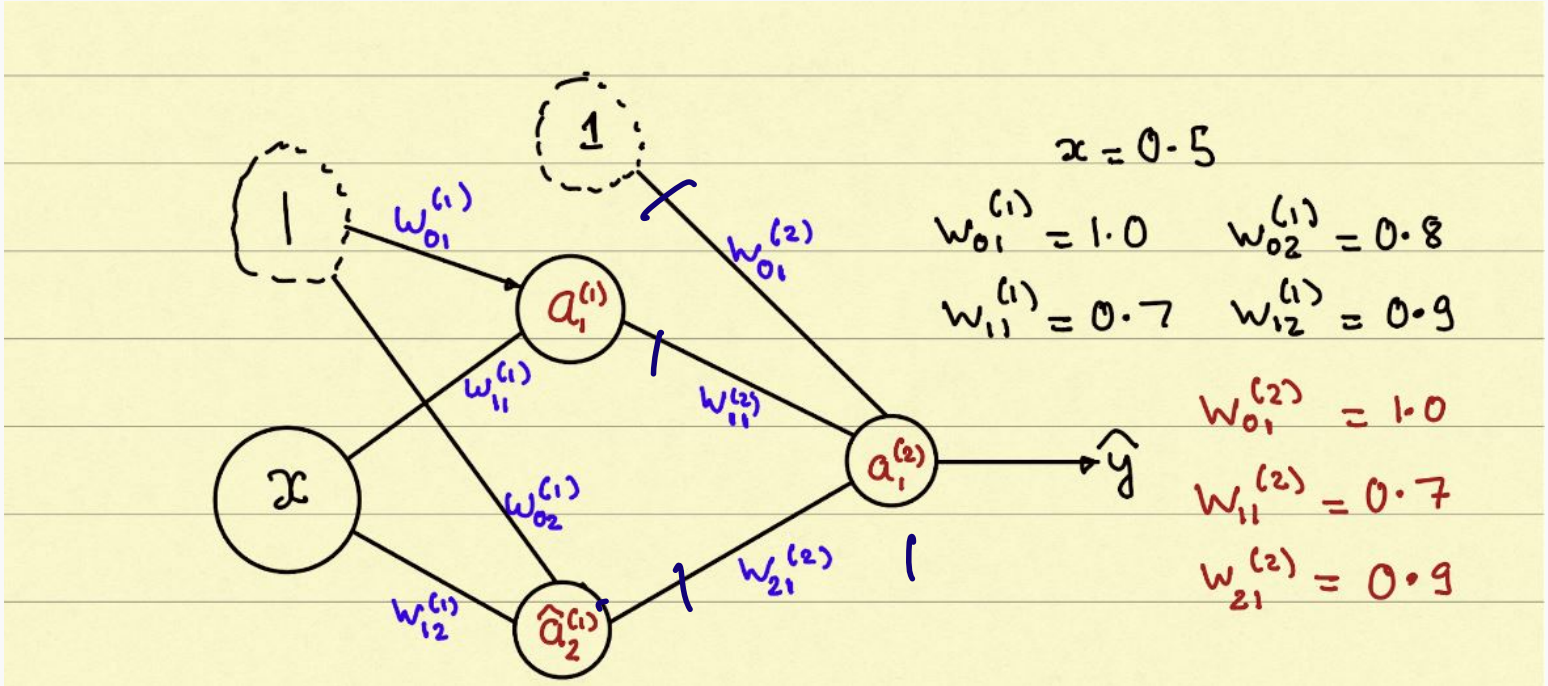
$$= g(0.8 + 0.9 \times 0.5) = g(1.25)$$



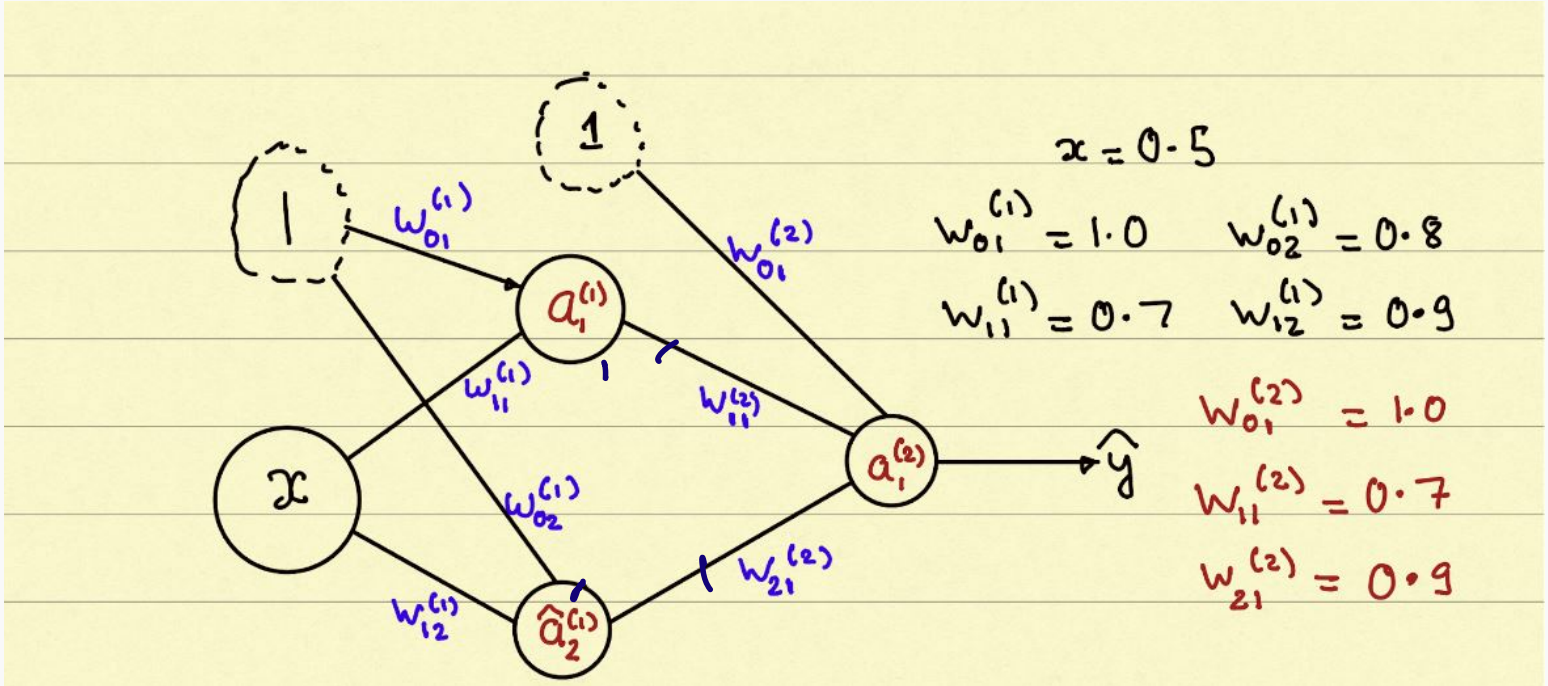
$$\hat{a}_2^{(1)} = g(z_2^{(1)}) = g(w_{02}^{(1)} x_0 + w_{12}^{(1)} x_1)$$

$$= g(0.8 + 0.9 \times 0.5) = g(1.25)$$

$$\hat{a}_2^{(1)} = 0.7773$$

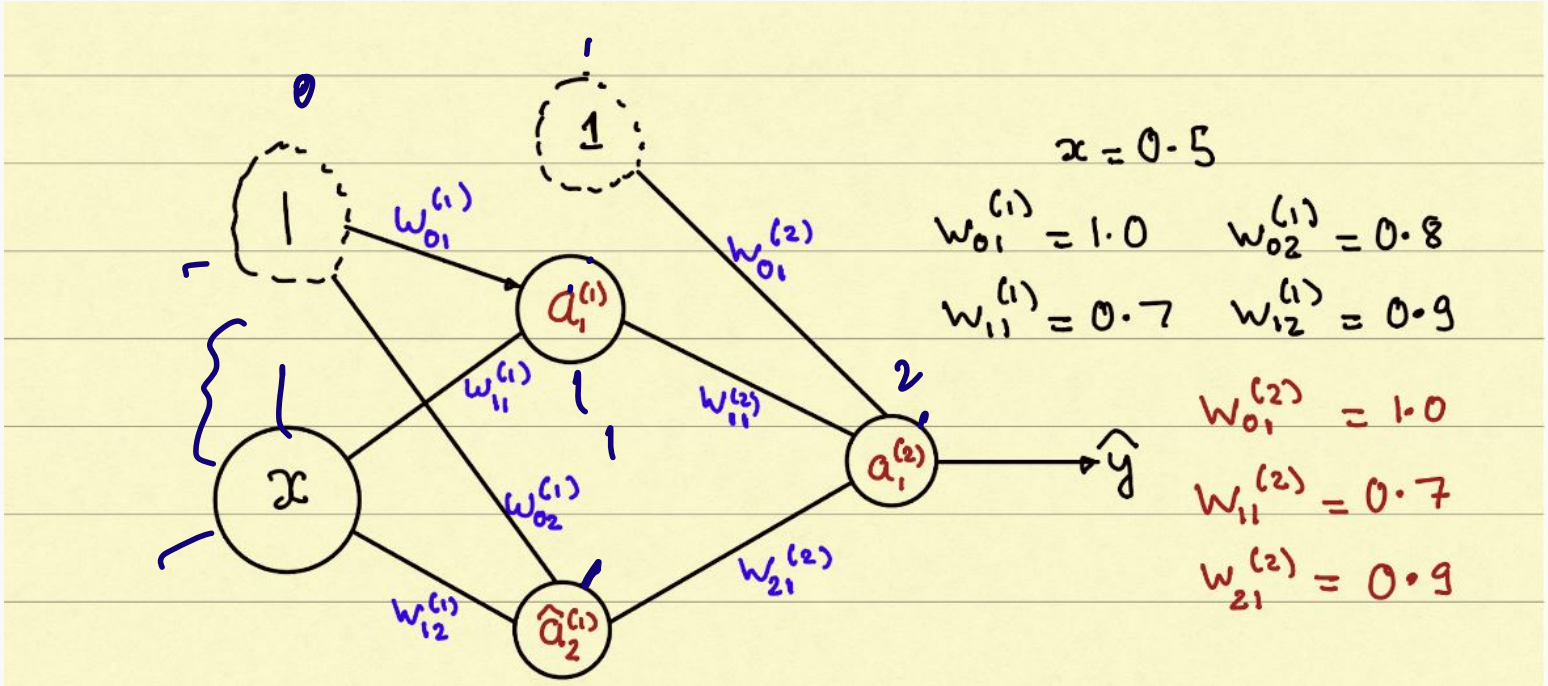


$$\hat{a}_1^{(2)} = g(z_1^{(2)}) = g(w_{01}^{(2)} x_0 + w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)})$$



$$\hat{a}_1^{(2)} = g(z_1^{(2)}) = g(w_{01}^{(2)} x_0 + w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)})$$

$$= g(1 \times 1 + 0.7 \times 0.7941 + 0.9 \times 0.7773) = g(2.255)$$

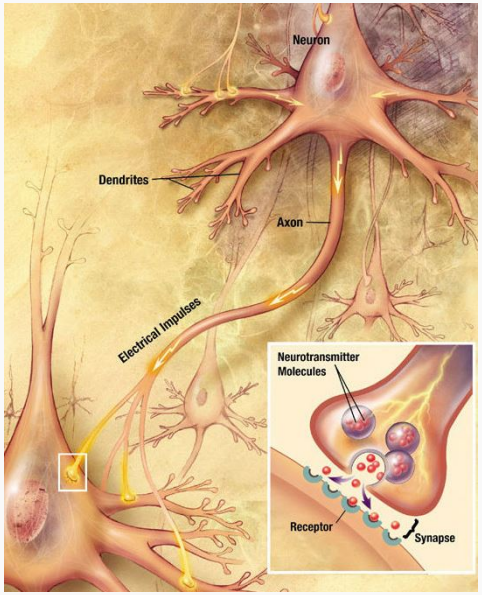


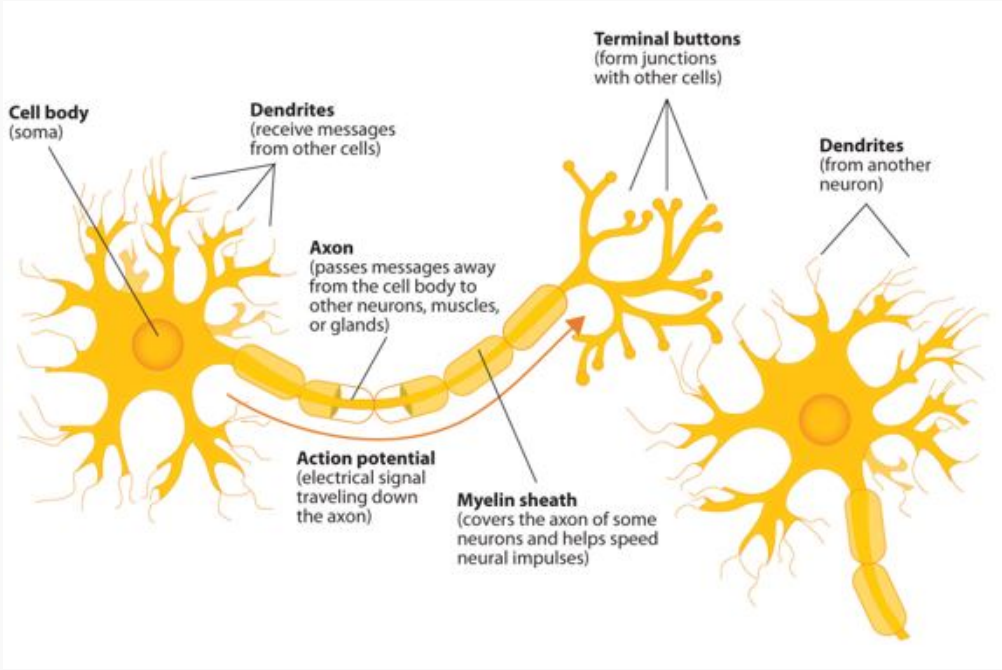
$$\hat{a}_1^{(2)} = g(z_1^{(2)}) = g(w_{01}^{(2)} x_0 + w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)})$$

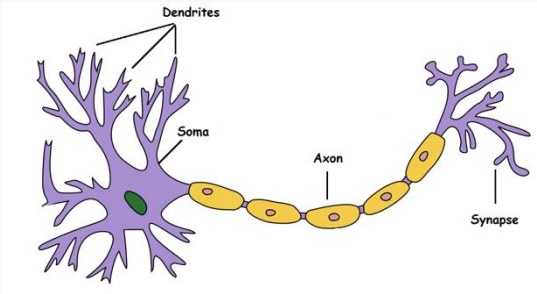
$$= g(1 \times 1 + 0.7 \times 0.7941 + 0.9 \times 0.7773) = g(2.255)$$

$$\hat{a}_1^{(2)} = 0.9051 = \hat{y}$$

From biology to computation


Neuron
Dendrites
Axon
Electrical Impulses
Neurotransmitter Molecules
Receptor
Synapse


Terminal buttons
(form junctions with other cells)
Cell body
(soma)
Dendrites
(receive messages from other cells)
Dendrites
(from another neuron)
Axon
(passes messages away from the cell body to other neurons, muscles, or glands)
Action potential
(electrical signal traveling down the axon)
Myelin sheath
(covers the axon of some neurons and helps speed neural impulses)


Dendrites
Soma
Axon
Synapse

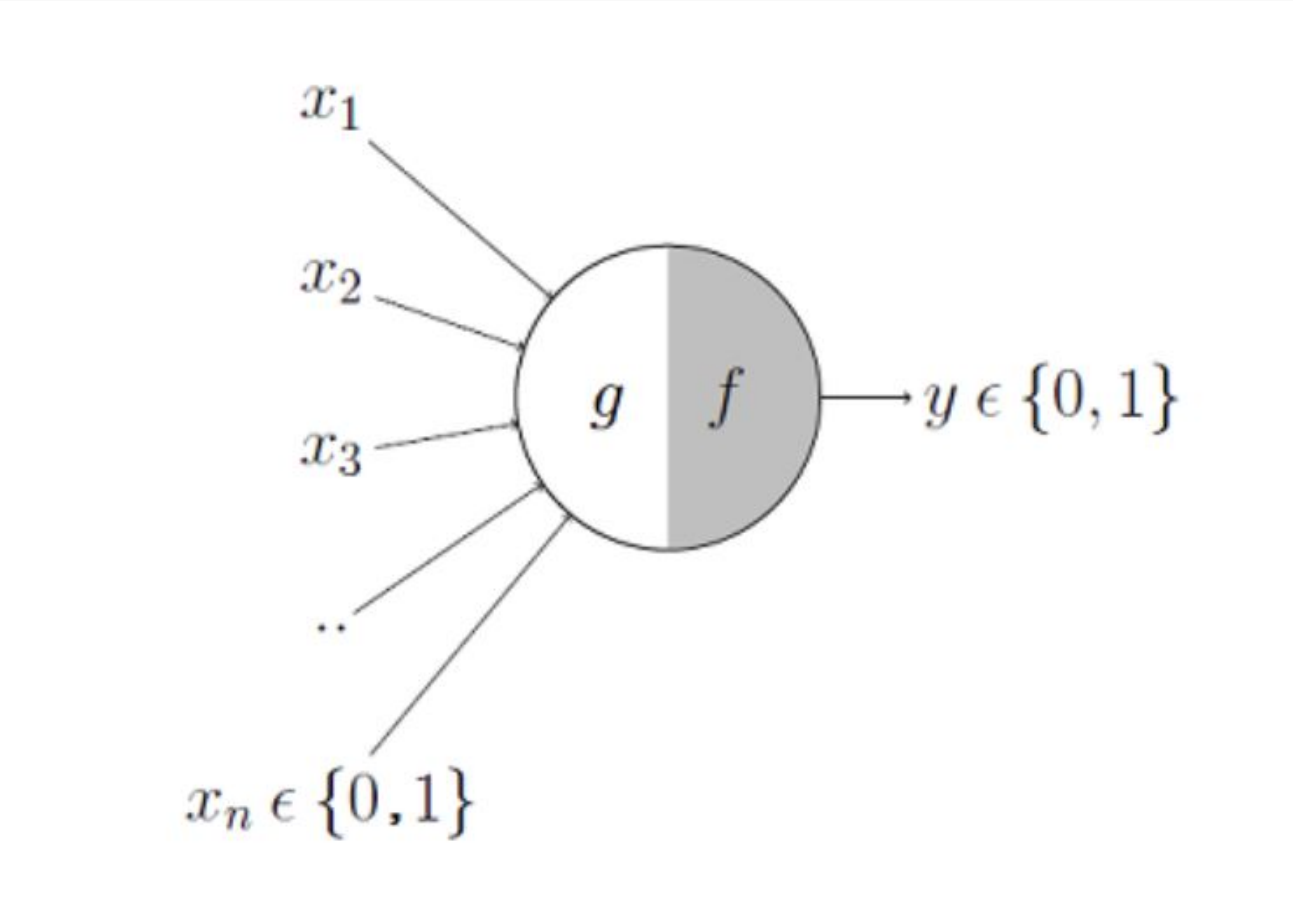
McCulloch-Pitts Neuron

McCulloch-Pitts Neuron

- First Mathematical model of Neuron (1943)

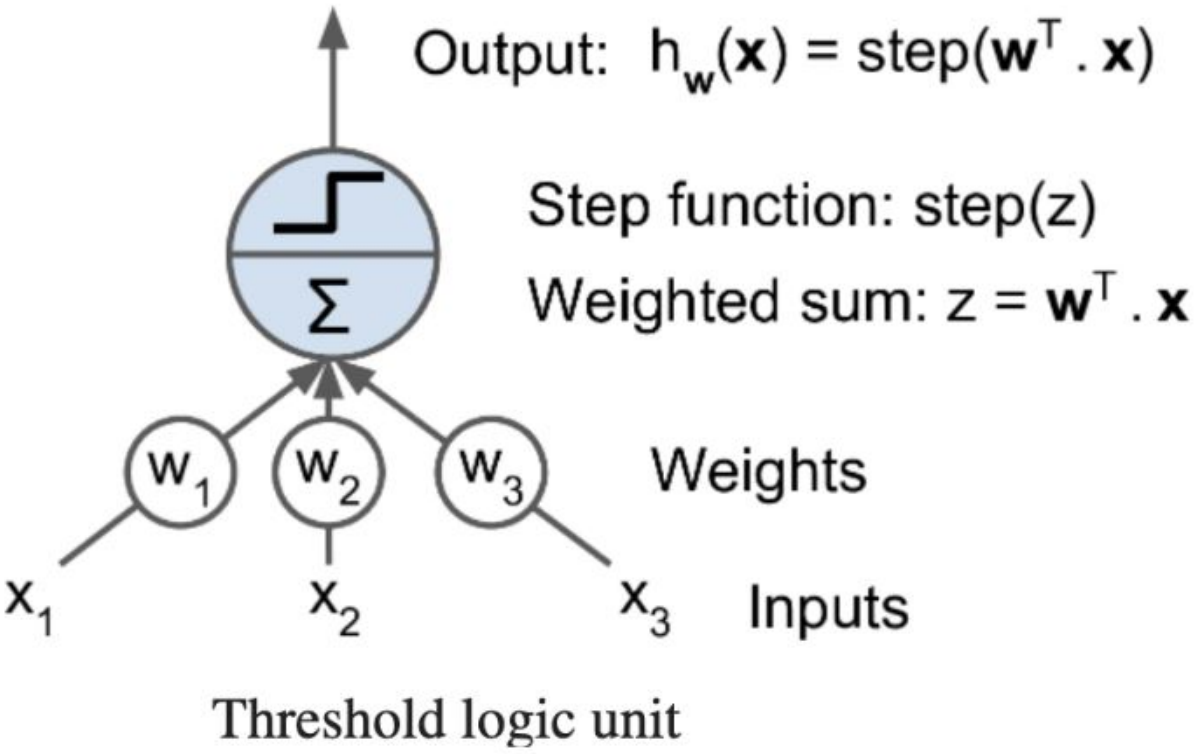
McCulloch-Pitts Neuron

- First Mathematical model of Neuron (1943)



Rosenblatt Perceptron (1957)

Rosenblatt Perceptron (1957)

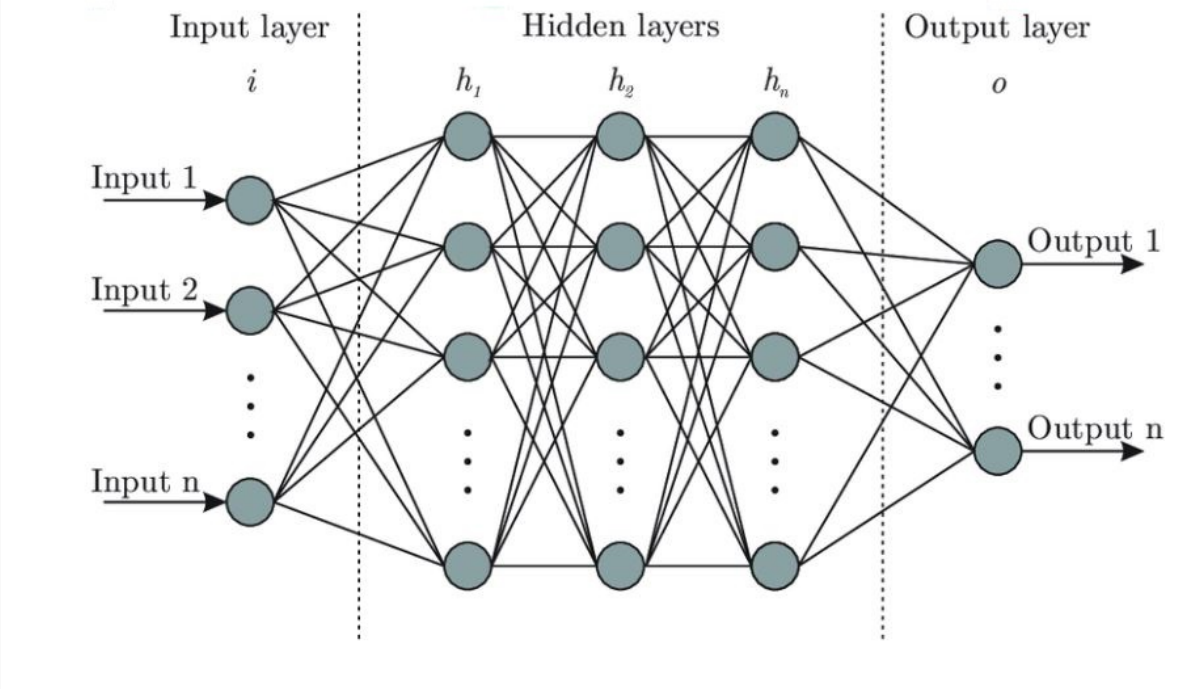


Threshold logic unit

Common non-linearities

Multilayer Perceptrons/Deep NNs

Multilayer Perceptrons/Deep NNs



Multilayer Perceptrons/Deep NNs

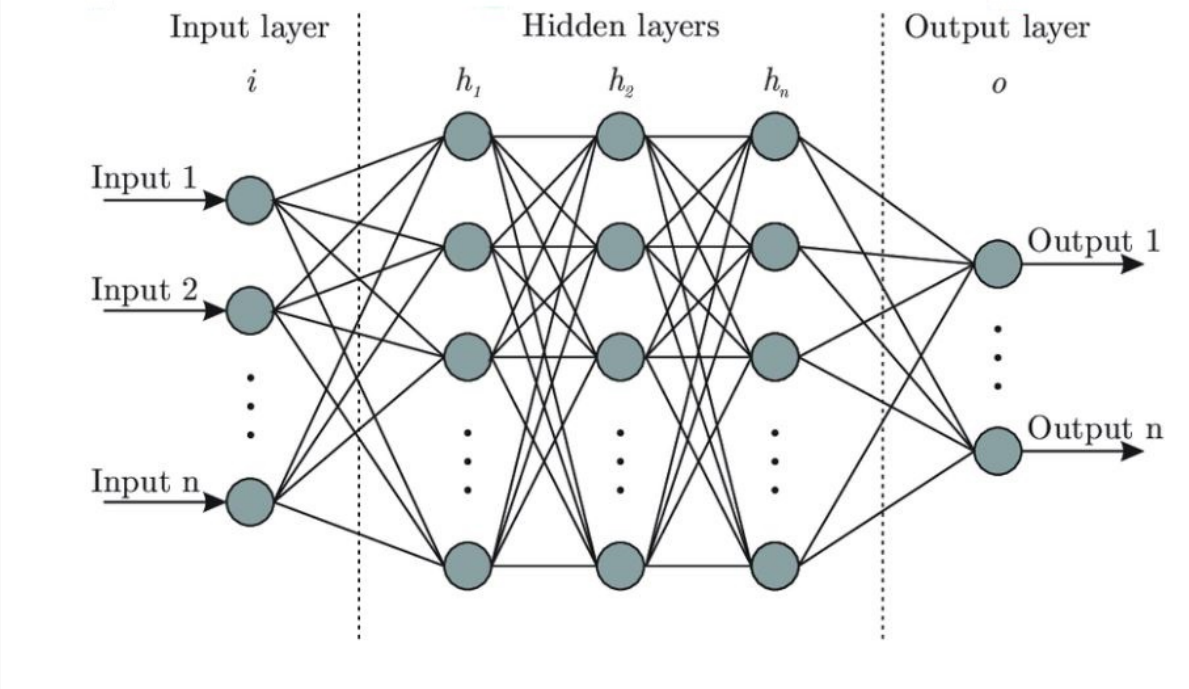
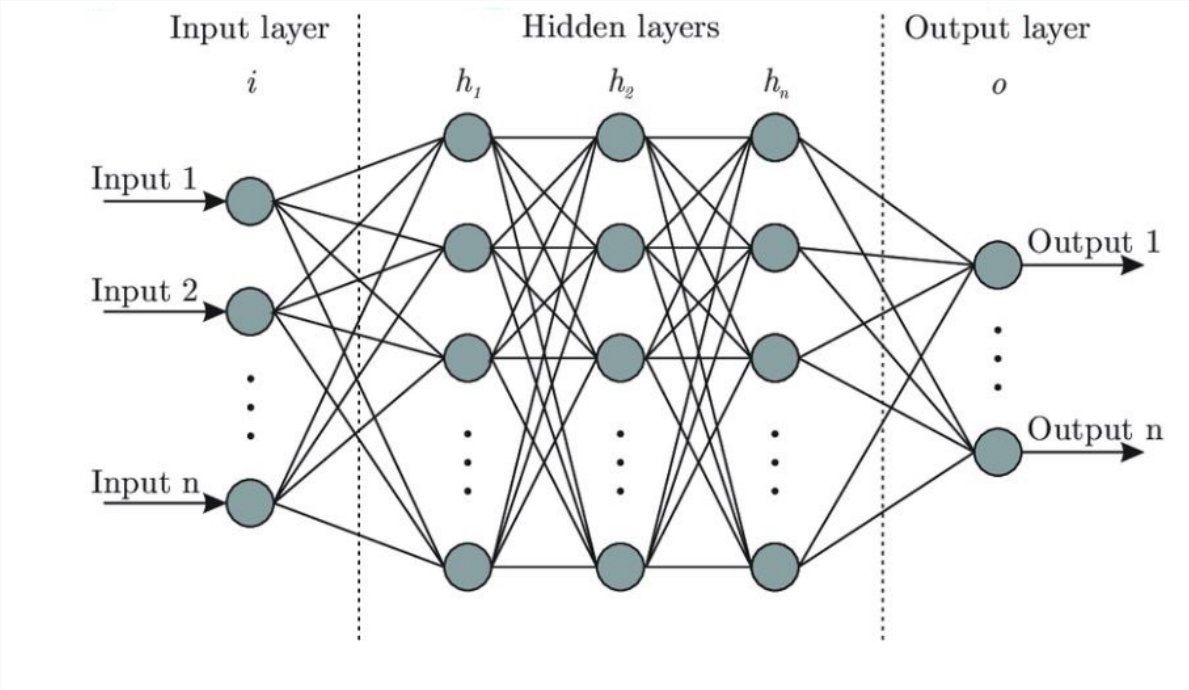


Image from kdnuggets.com

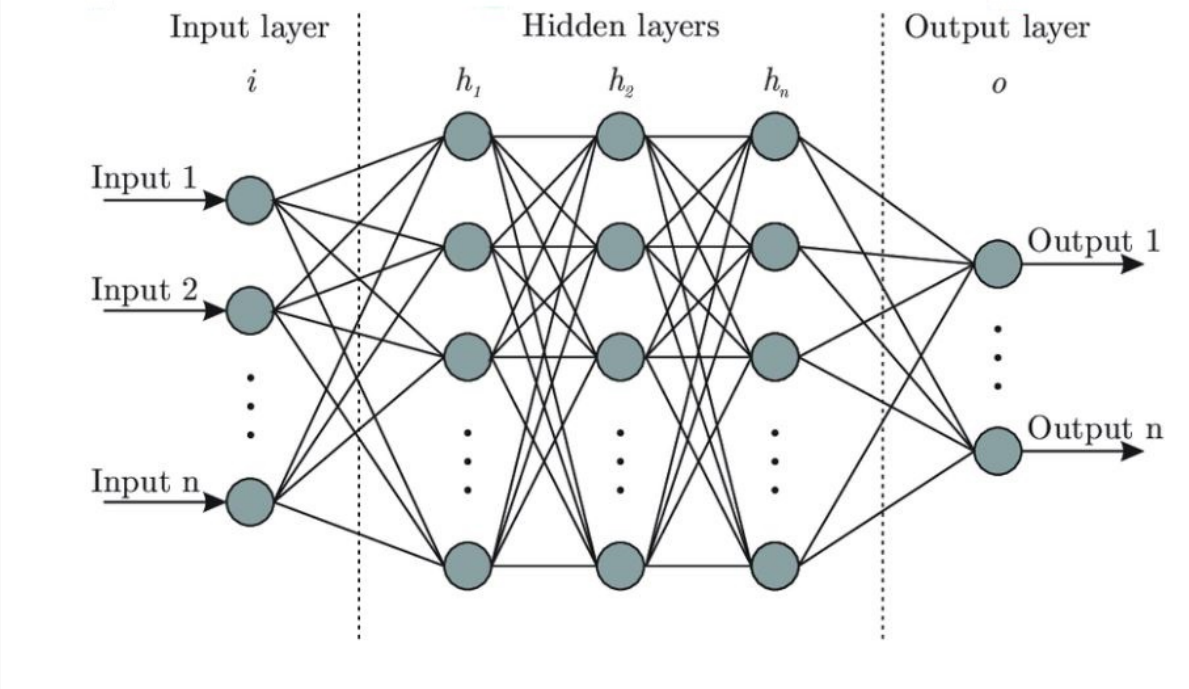
Multilayer Perceptrons/Deep NNs



- MLPs or Fully connected Networks are very common

Image from kdnuggets.com

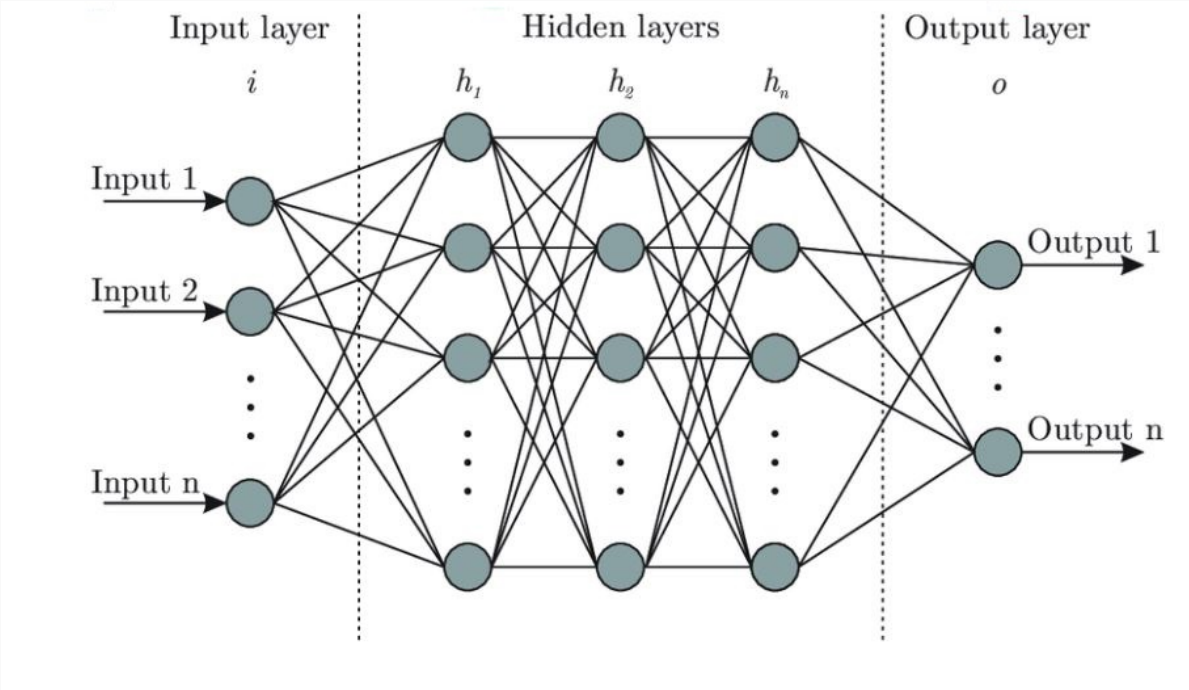
Multilayer Perceptrons/Deep NNs



- MLPs or Fully connected Networks are very common
  - At least as parts of other, more sparse networks

Image from kdnuggets.com

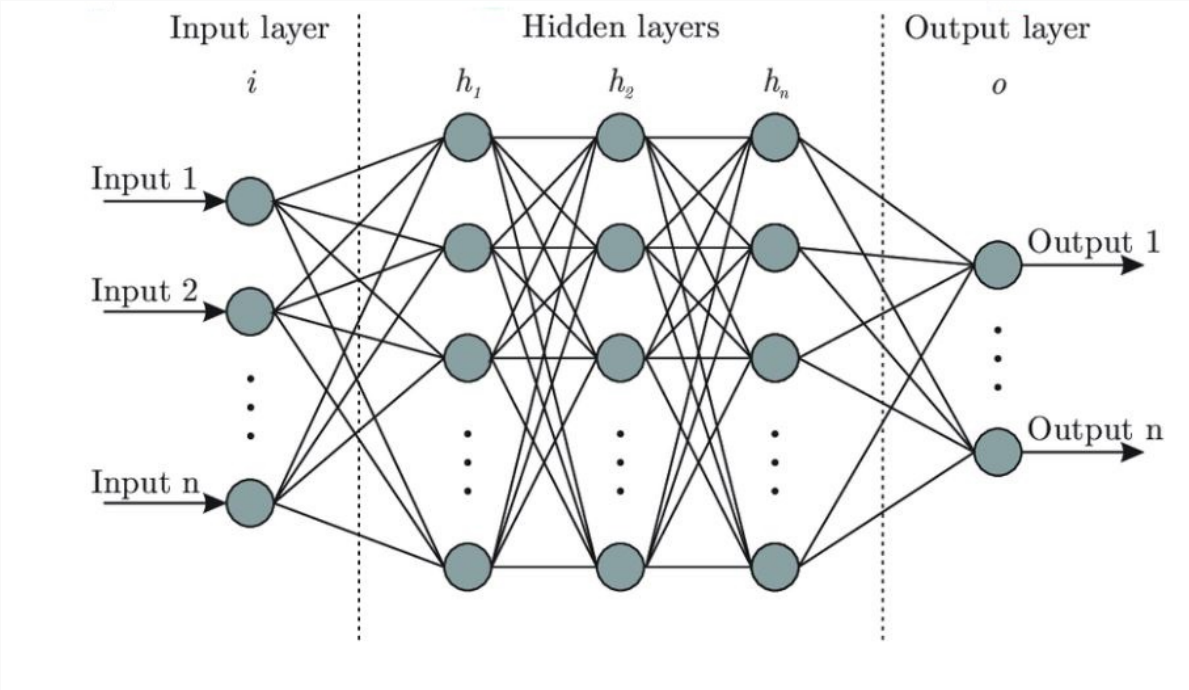
Multilayer Perceptrons/Deep NNs



- MLPs or Fully connected Networks are very common
  - At least as parts of other, more sparse networks
- Satisfy the Universal Approximation Theorem

Image from kdnuggets.com

Multilayer Perceptrons/Deep NNs



- MLPs or Fully connected Networks are very common
  - At least as parts of other, more sparse networks
- Satisfy the Universal Approximation Theorem
  - Can approximate any function to arbitrary accuracy with just one layer.

Image from kdnuggets.com

Types of Neural Networks

Types of Neural Networks

- Artificial Neural Networks

Types of Neural Networks

- Artificial Neural Networks
  - For general numerical data

Types of Neural Networks

- Artificial Neural Networks
  - For general numerical data
- Convolutional Neural Networks (CNNs)

Types of Neural Networks

- Artificial Neural Networks
  - For general numerical data
- Convolutional Neural Networks (CNNs)
  - For image based data

Types of Neural Networks

- Artificial Neural Networks
  - For general numerical data
- Convolutional Neural Networks (CNNs)
  - For image based data
- Recurrent Neural Networks (RNNs)

Types of Neural Networks

- Artificial Neural Networks
  - For general numerical data
- Convolutional Neural Networks (CNNs)
  - For image based data
- Recurrent Neural Networks (RNNs)
  - For sequential data

From linear models to DNNs

From linear models to DNNs