Topics For Today

1. Recap of multiclass classification
2. Deep Neural Networks (Back prop)
   - Scalar chain
   - Back prop for a MLP → Multi Layer Perceptron
   - Activation functions

   - Demo
   - Dropout
3. Variants of Gradient Descent + Demo
   - "Vanilla" Gradient Descent

Multiclass classification

n → no. of features

m → No of examples/data pts

$K = 2$ Binary Classification

$K = 3$ Multiclass Classification

- K : Number of classes.
  - K =2 Binary classification
  - K > 2 Multiclass (Multinomial) classification
- ML Recipe – Requires the following
  - Data Representation – How do we represent the output (it cannot be 0 or 1)
  - Model – What is the nonlinearity instead of sigmoid? (That was used to lie between 0 and 1)
  - Loss – What is the Loss function?
  - Optimization – Gradient Descent but What is the gradient for the new model?

Let us look at these issues one by one

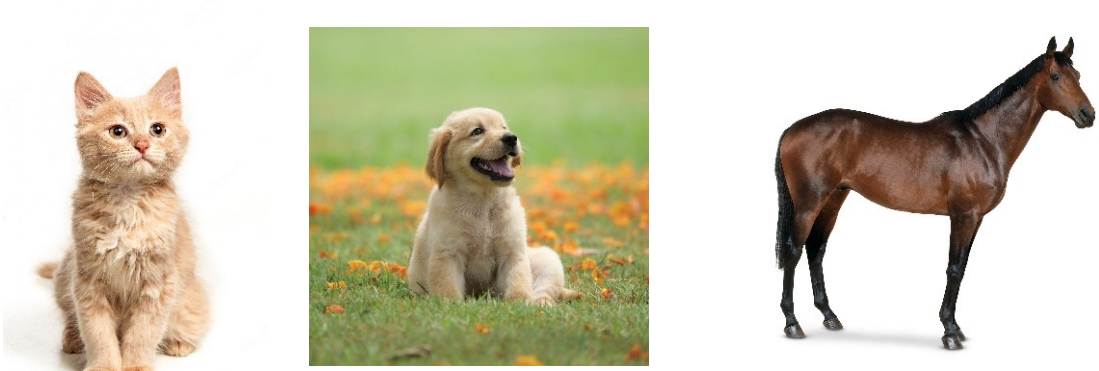
Data Representation – One Hot Vector

$K = 2$ Binary Classification

$y = 0$ $y = 1$

Binary Representation

$K = 3$ Multiclass Classification



1	0	0

Ground Truth y

0	**1**	0

0	0	**1**

One Hot Representation

- "One-Hot Representation" – Represent the output as a vector instead of a scalar
- Each output class is represented by a 1 in one location and 0 in others – One Hot
- We can interpret the entries in each box as the probability of the class

0.4	**0.5**	**0.1**
p(Cat)	p(Dog)	p(Horse)

Model Output $\hat{y}$ representation

$\sum_{i=1} \hat{y}_i = 1$

For both y and $\hat{y}$, every entry represents $p(y_i = 1|x)$

Model Nonlinearity -- Softmax

0.4	0.5	0.1
p(Cat)	p(Dog)	p(Horse)

$$\hat{y} = [0.4 \quad 0.5 \quad 0.1]$$

$$= [\hat{y_1} \quad \hat{y_2} \quad \hat{y_3}]$$

	z	$\sigma(z)$	Softmax(z)
1	1	0.7311	0.1191
2	3	0.9526	0.8801
3	-4	0.0180	0.0008
SUM Σ		**1.7016**	**1.0000**

- We can define $z = \Sigma w_i x_i$ as before
- However, we cannot use $\hat{y} = \sigma(z)$ since we would not have $\Sigma \hat{y_i} = 1$
- To guarantee that the output can be interpreted as a probability we introduce softmax.
- This is defined as

$$sf(z_2) = \frac{e^{z_2}}{D}, \quad sf(z_3) = \frac{e^{z_3}}{D} \Rightarrow sf(z_1) + sf(z_2) + sf(z_3) = 1$$

$$\text{softmax}(z_1) = \frac{\exp(z_1)}{(\exp(z_1) + \exp(z_2) + \exp(z_3)}$$

$sf(z_1)$; D

- This has all outputs between 0 and 1 as well as $\Sigma \hat{y_i} = 1$

Categorical Cross-Entropy (CCE) Loss Function

- Recall the binary cross entropy loss function

$$J^{BCE}(y, \hat{y}) = -y \, ln \, \hat{y} - (1 - y) \, ln \, (1 - \hat{y})$$

- We use a generalization of this for the multiclass case

$$J^{CCE}(y, \hat{y}) = -\Sigma_{i=1}^{K}(y_i \ln \hat{y}_i)$$

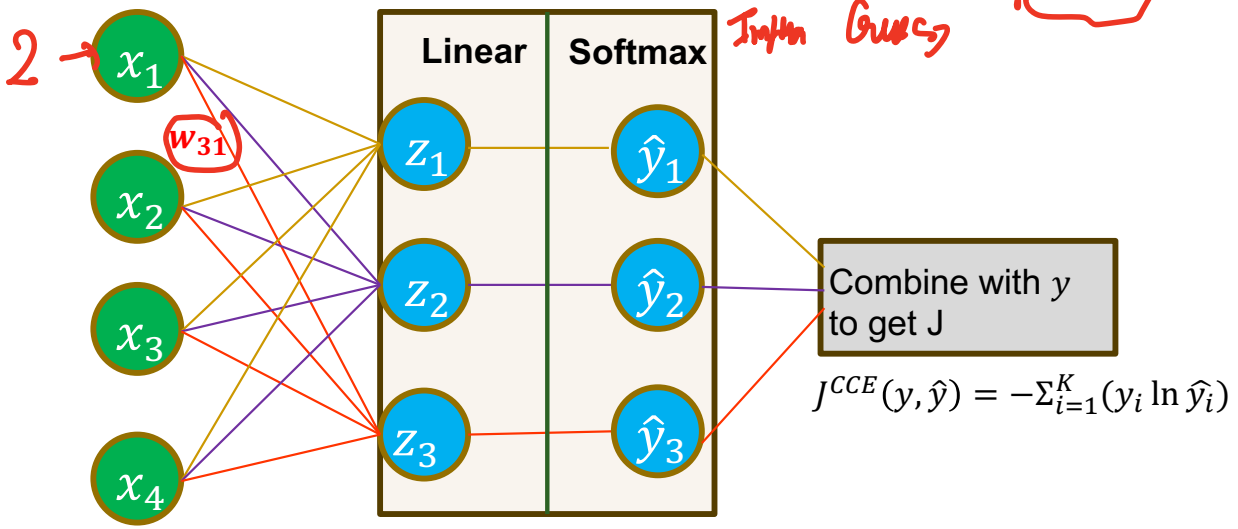
- What happens if apply CCE to the binary case with one-hot?

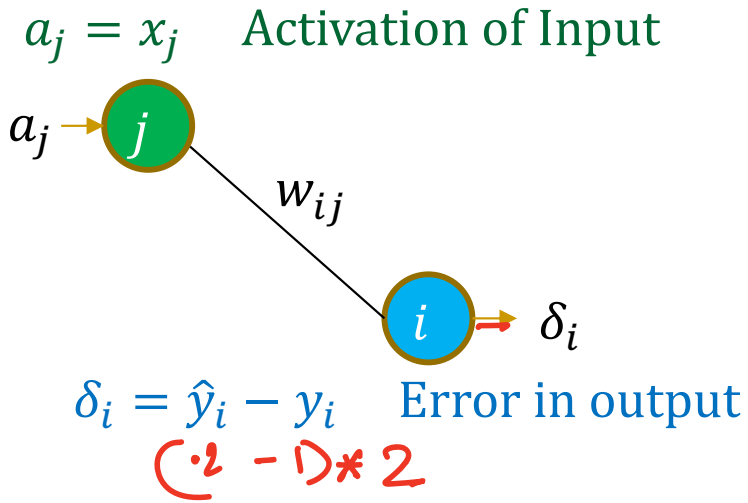
- $J^{CCE}(y, \hat{y}) = -\Sigma_{i=1}^{K}(y_i \ln \hat{y}_i) = -y_1 ln\hat{y_1} - y_2 ln\hat{y_2}$
- In the binary case $\hat{y_2} = (1 - \hat{y_1})$ and $y_2 = (1 - y_1)$

  $\Rightarrow J^{CCE}(y, \hat{y}) = -y_1 ln\hat{y_1} - (1 - y_1) \ln(1 - \hat{y_1})$
- This is identical to the binary cross entropy loss!

1		Binary		0	
1	0	One-hot	0	1	

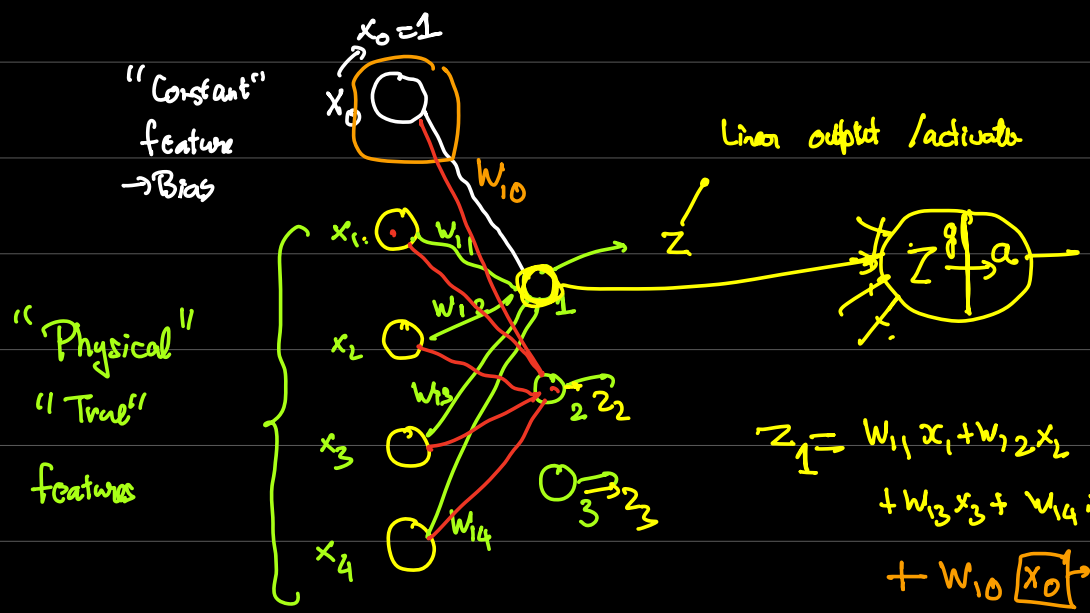
Optimization – Gradients in the Multiclass case





- It can be shown that $\frac{\partial J}{\partial z_1} = \hat{y}_1 - y_1$. Similarly, $\frac{\partial J}{\partial z_i} = \hat{y}_i - y_i$
  - Same $\frac{\partial J}{\partial z}$ expression as linear and logistic
- $\frac{\partial J}{\partial w_{31}}$ = Error in Output 3×Activation of Input 1=$\delta_3 \times a_1 = (\hat{y}_3 - y_3) \times x_1$

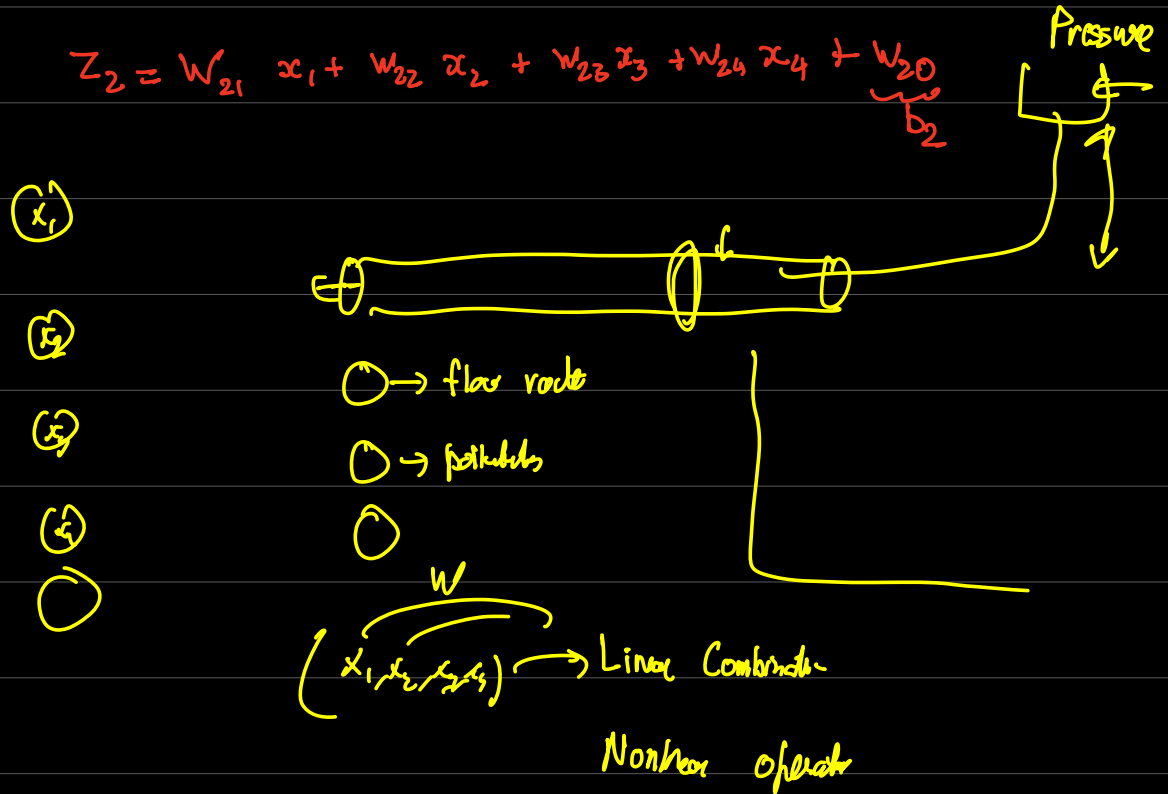
General Delta Rule $\frac{\partial J}{\partial w_{ij}} = \delta_i a_j$



$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + w_{10}\boxed{x_0 \to 1}$$

$$z_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + w_{14}x_4 + b_1$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + w_{24}x_4 + \underbrace{w_{20}}_{b_2}$$



General Supervised Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using a hypothesis Model
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

$$\mathbf{x}^{(1)} = \left(x_1^{(1)}, x_2^{(1)}, .., x_n^{(1)}\right)$$

$$\mathbf{y}^{(1)} = \left(y_1^{(1)}, y_2^{(1)}, .., y_K^{(1)}\right)$$

$$\hat{\mathbf{y}}^{(1)} = \left(\hat{y}_1^{(1)}, \hat{y}_2^{(1)}, .., \hat{y}_K^{(1)}\right)$$

x	y	$\hat{y}$	J	
$\mathbf{x}^{(1)}$	$\mathbf{y}^{(1)}$	$\hat{\mathbf{y}}^{(1)}$	$J^{(1)}$	
$\mathbf{x}^{(2)}$	$\mathbf{y}^{(2)}$	$\hat{\mathbf{y}}^{(2)}$	$J^{(2)}$	
$\mathbf{x}^{(3)}$	$\mathbf{y}^{(3)}$	$\hat{\mathbf{y}}^{(3)}$	$J^{(3)}$	
$\mathbf{x}^{(4)}$	$\mathbf{y}^{(4)}$	$\hat{\mathbf{y}}^{(4)}$	$J^{(4)}$	
$\mathbf{x}^{(5)}$	$\mathbf{y}^{(5)}$	$\hat{\mathbf{y}}^{(5)}$	$J^{(5)}$	
…	…	…	…	
$\mathbf{x}^{(m)}$	$\mathbf{y}^{(m)}$	$\hat{\mathbf{y}}^{(m)}$	$J^{(m)}$	$\mathrm{J} = \Sigma J^{(i)}$

Linear Regression – Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using
  - The Linear Model $y^{(j)} = \Sigma_{i=0}^{n} w_i x_i^{(j)}$
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
  - Gradient $\frac{\partial J}{\partial w_j} = \Sigma_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

x	y	$\hat{y}$	J
$x^{(1)}$	$y^{(1)}$	$\hat{y}^{(1)}$	$J^{(1)}$
$x^{(2)}$	$y^{(2)}$	$\hat{y}^{(2)}$	$J^{(2)}$
$x^{(3)}$	$y^{(3)}$	$\hat{y}^{(3)}$	$J^{(3)}$
$x^{(4)}$	$y^{(4)}$	$\hat{y}^{(4)}$	$J^{(4)}$
$x^{(5)}$	$y^{(5)}$	$\hat{y}^{(5)}$	$J^{(5)}$
…	…	…	…
$x^{(m)}$	$y^{(m)}$	$\hat{y}^{(m)}$	$J^{(m)}$

$J = \Sigma J^{(i)}$

Binary Classification – Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using
  - The Logistic Model $\hat{y}^{(j)} = \sigma(\Sigma_{i=0}^{n} w_i x_i^{(j)})$
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
  - Gradient $\frac{\partial J}{\partial w_j} = \Sigma_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

x	y	$\hat{y}$	J
$x^{(1)}$	$y^{(1)}$	$\hat{y}^{(1)}$	$J^{(1)}$
$x^{(2)}$	$y^{(2)}$	$\hat{y}^{(2)}$	$J^{(2)}$
$x^{(3)}$	$y^{(3)}$	$\hat{y}^{(3)}$	$J^{(3)}$
$x^{(4)}$	$y^{(4)}$	$\hat{y}^{(4)}$	$J^{(4)}$
$x^{(5)}$	$y^{(5)}$	$\hat{y}^{(5)}$	$J^{(5)}$
…	…	…	…
$x^{(m)}$	$y^{(m)}$	$\hat{y}^{(m)}$	$J^{(m)}$

$J = \Sigma J^{(i)}$

Multinomial Classification – Learning Algorithm

- Step 1 : Data Collection – Collect Data in (x, y) pairs
- Step 2 : Parameter Initialization – Guess for w
- Step 3 : Forward Pass – For all data points in the dataset find $\hat{y}$ using
  - The Softmax Model $\hat{y}^{(j)} = \text{softmax}\,(\Sigma_{i=0}^{n} w_i x_i^{(j)})$
- Step 4 : Gradient Descent – Improve w using gradient descent. $w_j = w_j - \alpha \frac{\partial J}{\partial w_j}$
  - Gradient $\frac{\partial J}{\partial w_j} = \Sigma_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)}) x_j^{(i)}$
- Step 5 : Check exit criterion.
  - If not satisfied, go to Step 3 with the updated parameters

x	y	$\hat{y}$	J
$x^{(1)}$	$y^{(1)}$	$\hat{y}^{(1)}$	$J^{(1)}$
$x^{(2)}$	$y^{(2)}$	$\hat{y}^{(2)}$	$J^{(2)}$
$x^{(3)}$	$y^{(3)}$	$\hat{y}^{(3)}$	$J^{(3)}$
$x^{(4)}$	$y^{(4)}$	$\hat{y}^{(4)}$	$J^{(4)}$
$x^{(5)}$	$y^{(5)}$	$\hat{y}^{(5)}$	$J^{(5)}$
…	…	…	…
$x^{(m)}$	$y^{(m)}$	$\hat{y}^{(m)}$	$J^{(m)}$

$J = \Sigma J^{(i)}$

Practical Consideration – Batching Gradient Descent

x	y	$\hat{y}$	J
$\mathbf{x}^{(1)}$	$\mathbf{y}^{(1)}$	$\hat{\mathbf{y}}^{(1)}$	$J^{(1)}$
$\mathbf{x}^{(2)}$	$\mathbf{y}^{(2)}$	$\hat{\mathbf{y}}^{(2)}$	$J^{(2)}$
$\mathbf{x}^{(3)}$	$\mathbf{y}^{(3)}$	$\hat{\mathbf{y}}^{(3)}$	$J^{(3)}$
$\mathbf{x}^{(4)}$	$\mathbf{y}^{(4)}$	$\hat{\mathbf{y}}^{(4)}$	$J^{(4)}$
$\mathbf{x}^{(5)}$	$\mathbf{y}^{(5)}$	$\hat{\mathbf{y}}^{(5)}$	$J^{(5)}$
…	…	…	…
$\mathbf{x}^{(m)}$	$\mathbf{y}^{(m)}$	$\hat{\mathbf{y}}^{(m)}$	$J^{(m)}$

Gradient Update

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

$$\frac{\partial \mathrm{J}}{\partial w_j} = \frac{1}{\mathrm{b}} \frac{\Sigma \partial J^{(i)}}{\partial w_j}$$

b is the size of the batch

- Default option – Each update in gradient descent requires to see the whole dataset
  - That is, one update requires us to average the updates of every point in the whole dataset
  - This is called **Batch Gradient Descent** where b = m. One epoch has one update
  - Problems : (i) It is slow (ii) It is expensive for each update (iii) Data cannot be dynamic
- Stochastic Gradient Descent
  - $b = 1$. One epoch has m updates. Many more updates per epoch
  - Can handle dynamic incoming data but loss convergence is more erratic
- Mini-Batch Gradient Descent
  - $1 < b < m$. One epoch has $\frac{m}{b}$ updates. More than 1 update per epoch
  - Loss convergence more erratic than Batch and less erratic than stochastic

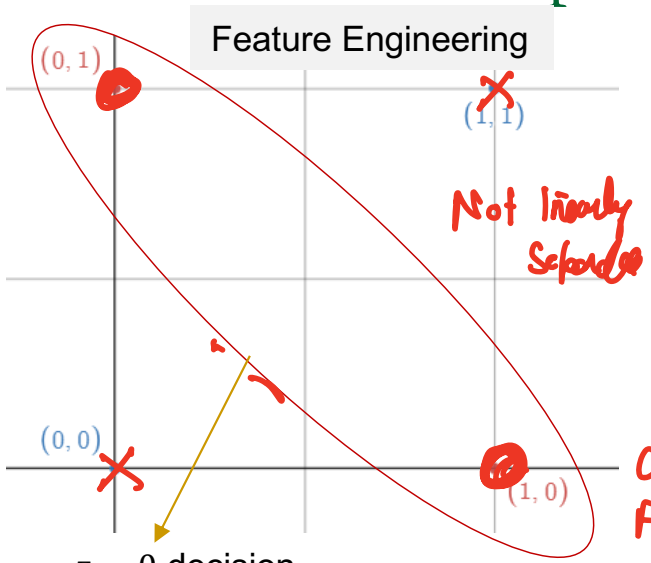
The XOR problem – Need for deep networks

	x_1	x_2	y
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

$\hat{y} = x_1 + x_2 - 2x_1x_2$

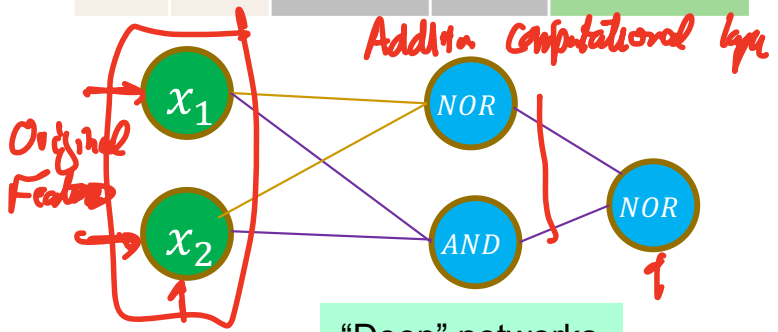
Hand-picked

Additional feature → Feature learning



$z = 0$ decision boundary is not a line

x_1	x_2	NOR	AND	NOR(NOR,AND)
0	0	1	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0

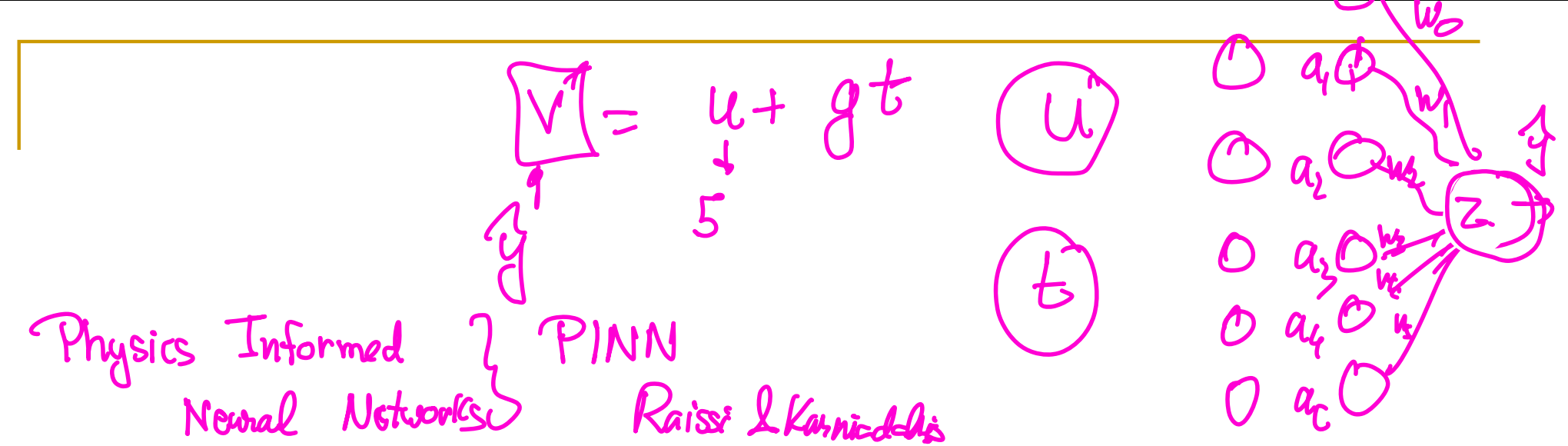


"Deep" networks

- In all cases so far, we map the input to $z = w_0 + w_1x_1 + \cdots$
- The decision boundary, $z = 0$ will, therefore, always be a line
  - If we directly map $x \rightarrow z$ like so far, it only works for **linearly separable data**
- Two solutions
  - Feature engineering -- Use nonlinear features. Example -- $\hat{y} = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2$
  - Use "deep" networks – Add extra hidden layers to create non-interpretable nonlinear features

$x_3 = x_1x_2$ Nonlinear feature

- We can use $XOR(x_1, x_2) = NOR\big(NOR(x_1, x_2), AND(x_1, x_2)\big)$ for example
- But, can we automatically come up with this representation via learning?



DEEP NEURAL NETWORKS

(Multi Layer Perceptrons – MLPs)

Regression tasks → $g_0(z) = z$ $\quad z = w_1 a_1 + w_2 a_2 + w_3 a_3 + w_4 a_4 + w_5 a_5 + w_0$

Linear activation

$\hat{y} = g_0(z)$ Nonlinearity at the output

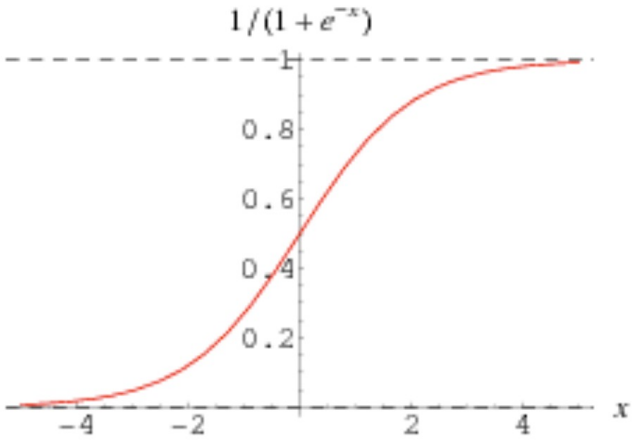
g_0 Can be different from g inside the network

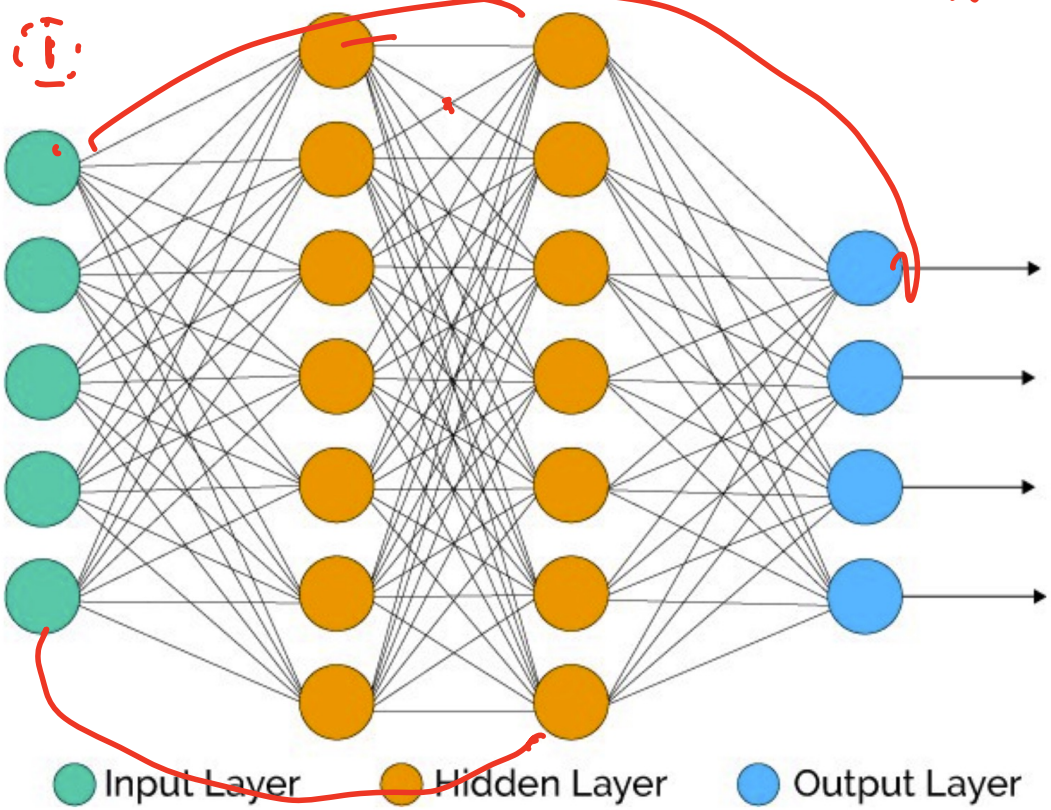
Recall : Neural Networks

Neural Network Model

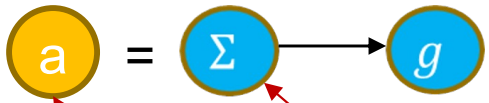
Common Activation function

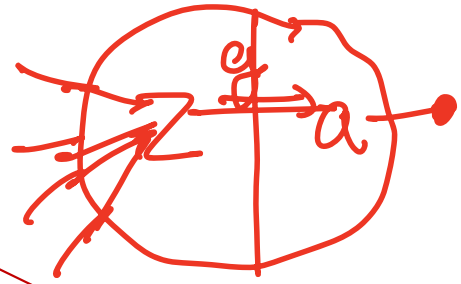
$$\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$$





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

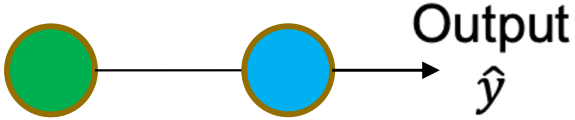




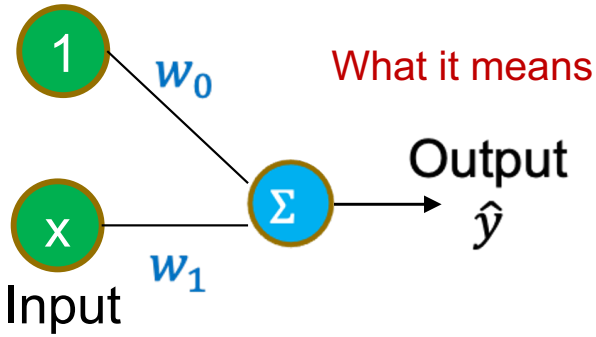
A Neuron has a linear and a nonlinear operation

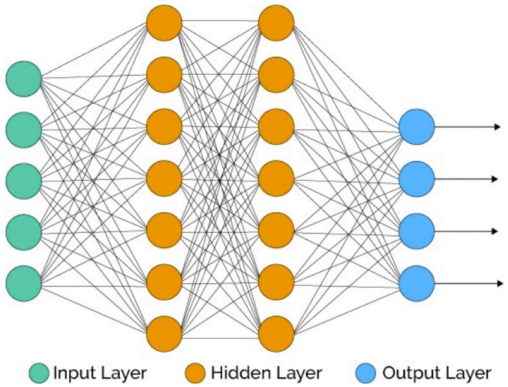
Weights and biases

What is shown

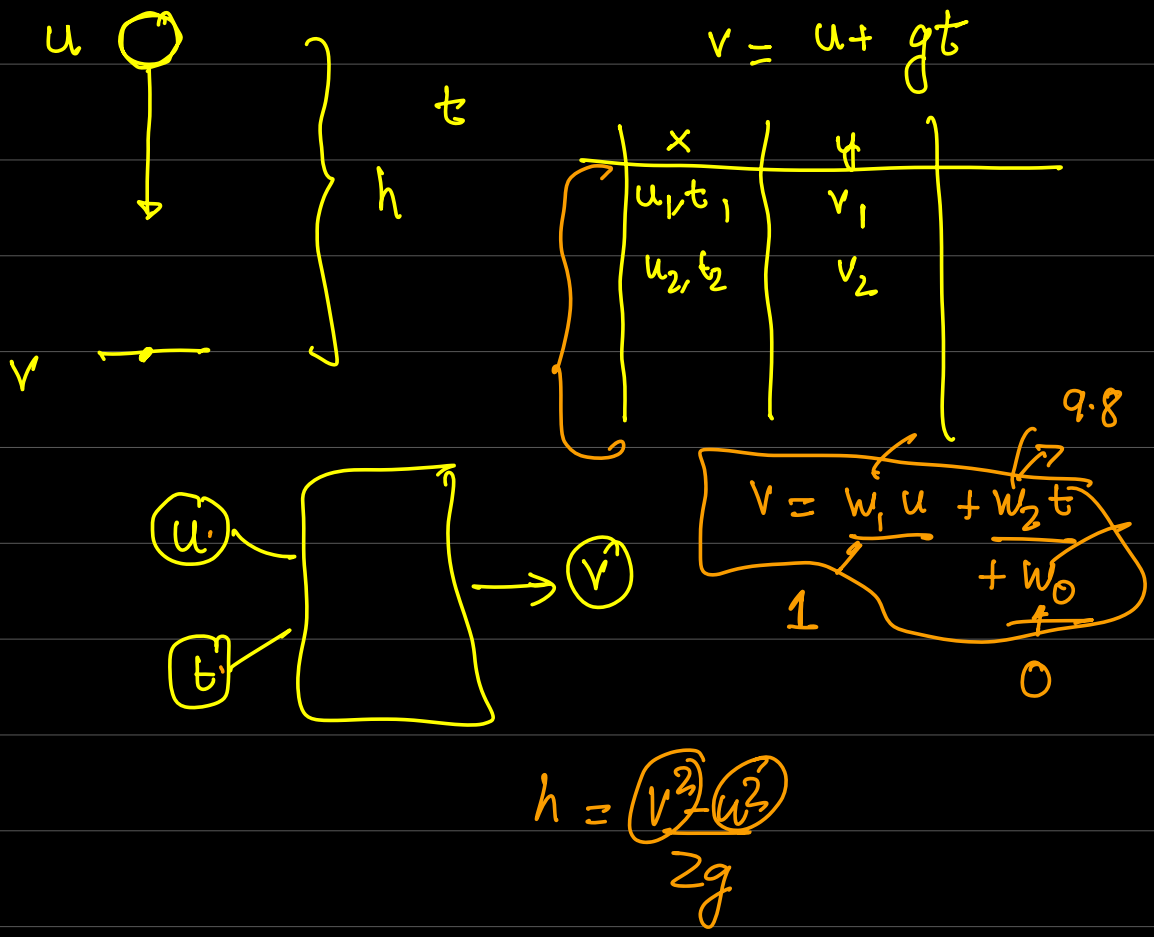


What it means



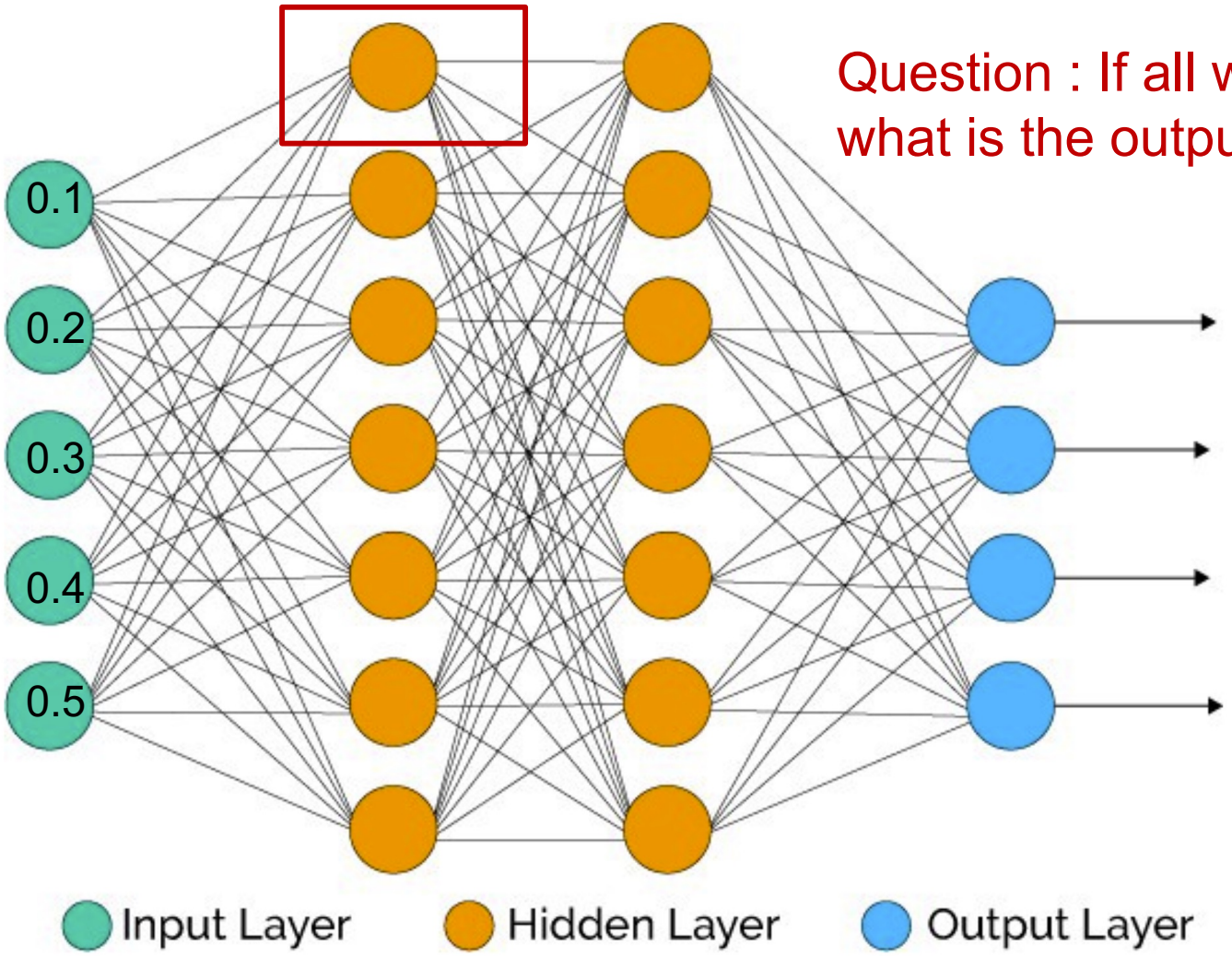


- Constant neurons representing a “feature” with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a “bias unit”
- The other parameters are called “weights”

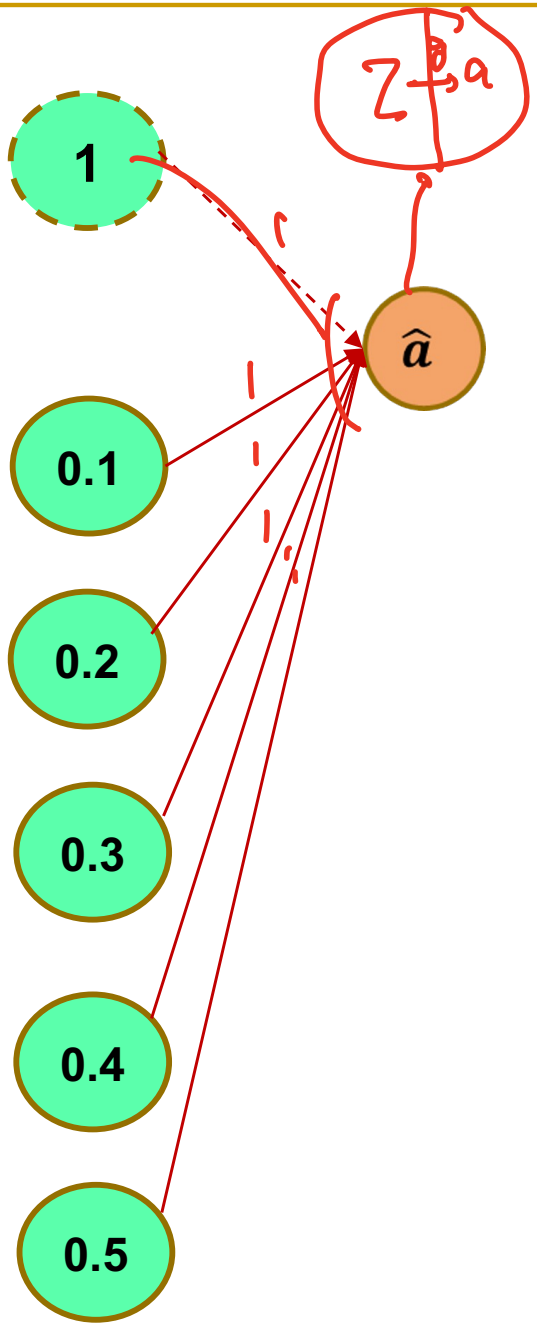


$$h = \frac{v^2 - w^2}{2g}$$

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

Ans: The output has two parts

Linear -- $z = \Sigma w_i x_i$

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

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

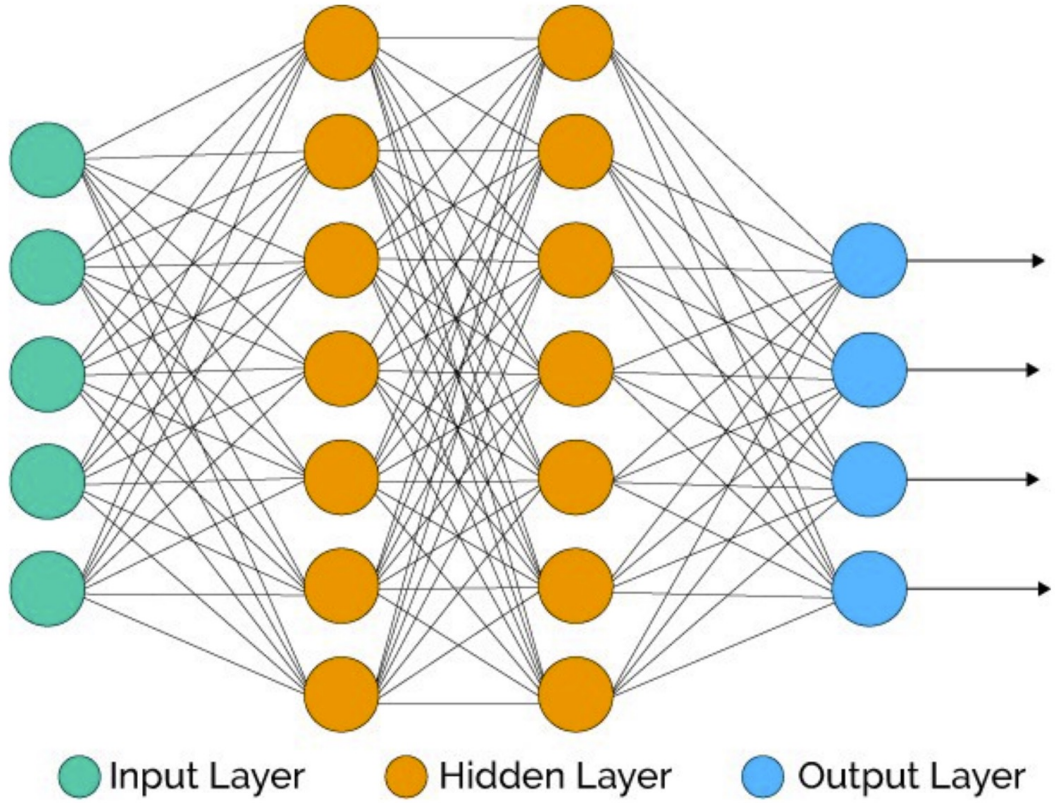
Nonlinear Activation -- $\hat{a} = g(z)$

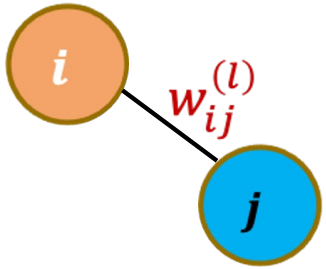
$$= \frac{1}{(1 + \exp(-2.5))}$$

$$= 0.9241$$

Don't forget the bias unit!

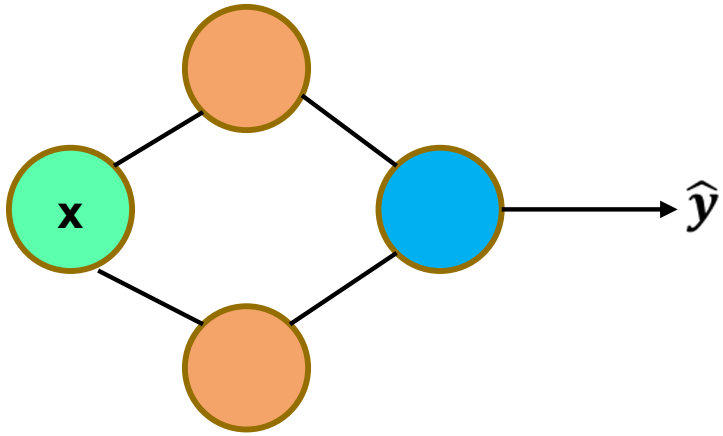
Weights notation (temporarily, only for this example)



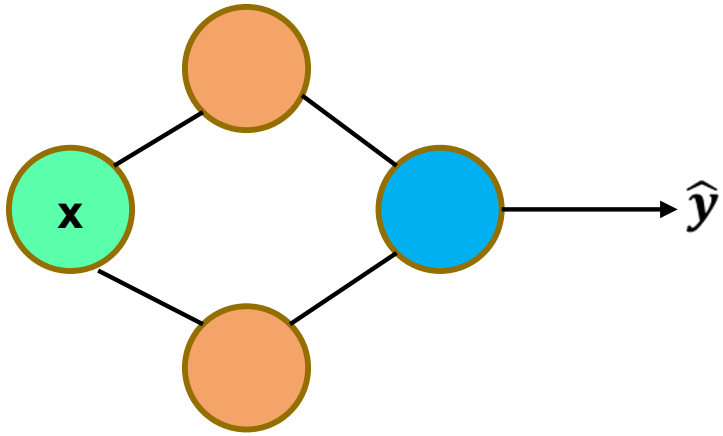


- $w_{ij}^{(l)}$ is the weight connecting Neuron i of Layer l to Neuron j of Layer $l + 1$
- Bias units are the 0^{th} neurons

A simple forward pass calculation

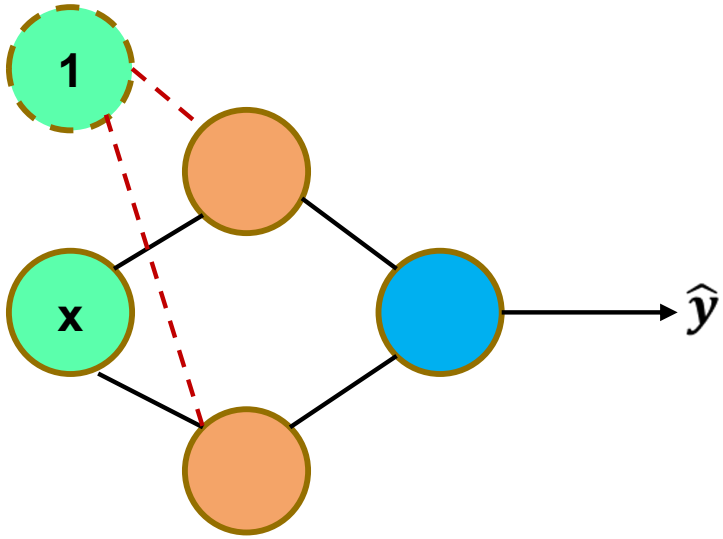


Question: Find the output of the given network for the following



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

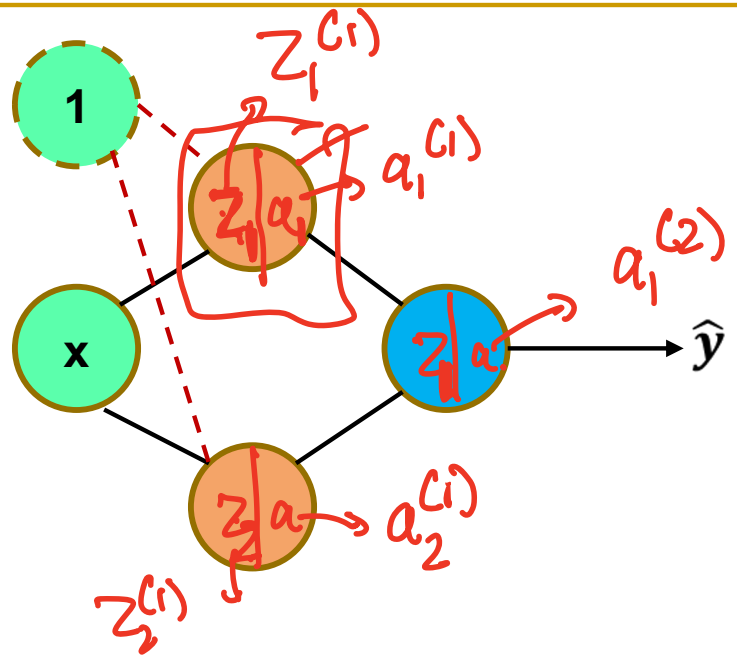
$$x = 0.5$$



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

$$x = 0.5$$

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



$a_1^{(1)} = ?$

$a_1^{(1)} = ?$

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

$$x = 0.5$$

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

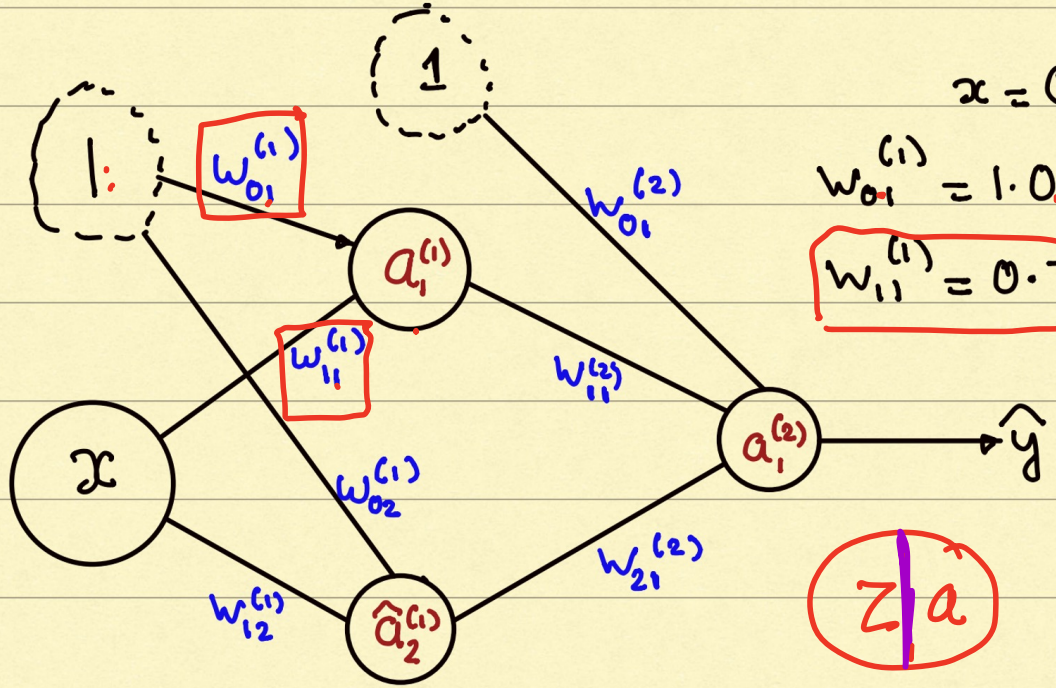
$$w_{11}^{(1)} = 0.7 \quad w_{12}^{(1)} = 0.9$$

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

$$w_{11}^{(2)} = 0.7 \quad w_{21}^{(2)} = 0.9$$

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

SOLUTION



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

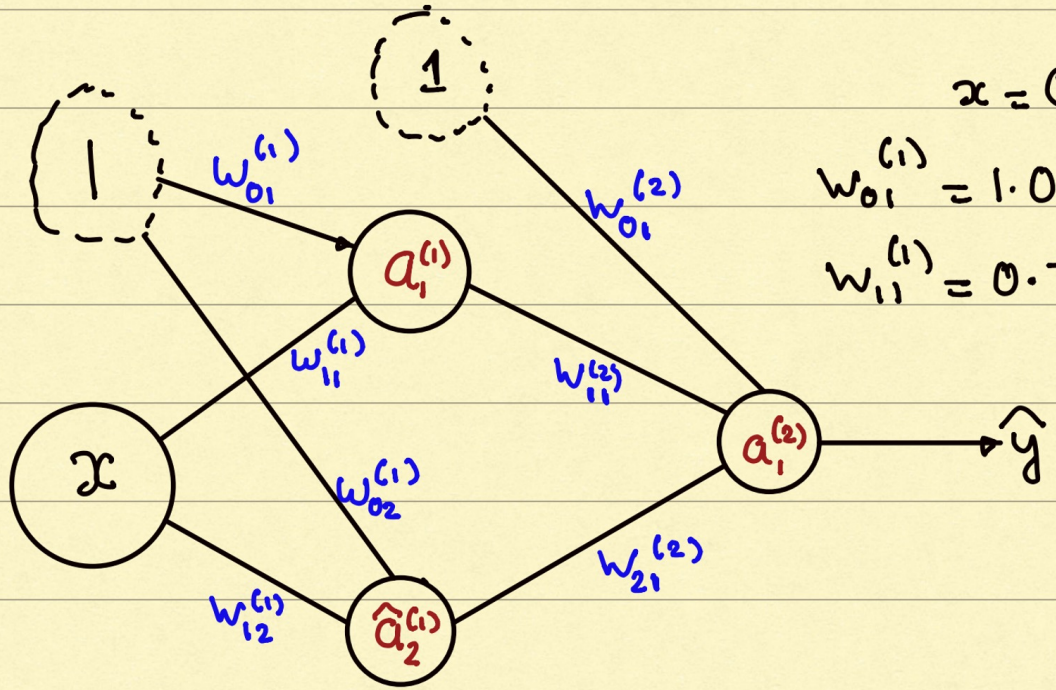
Linear $\rightarrow$ $z_1^{(1)} = w_{01}^{(1)} * 1 + w_{11}^{(1)} * x$

$= 1 * 1 + 0.7 * 0.5$

$= 1.35$

$a_1^{(1)} = g(z_1^{(1)})$

$= \sigma(1.35)$



$x = 0.5$

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

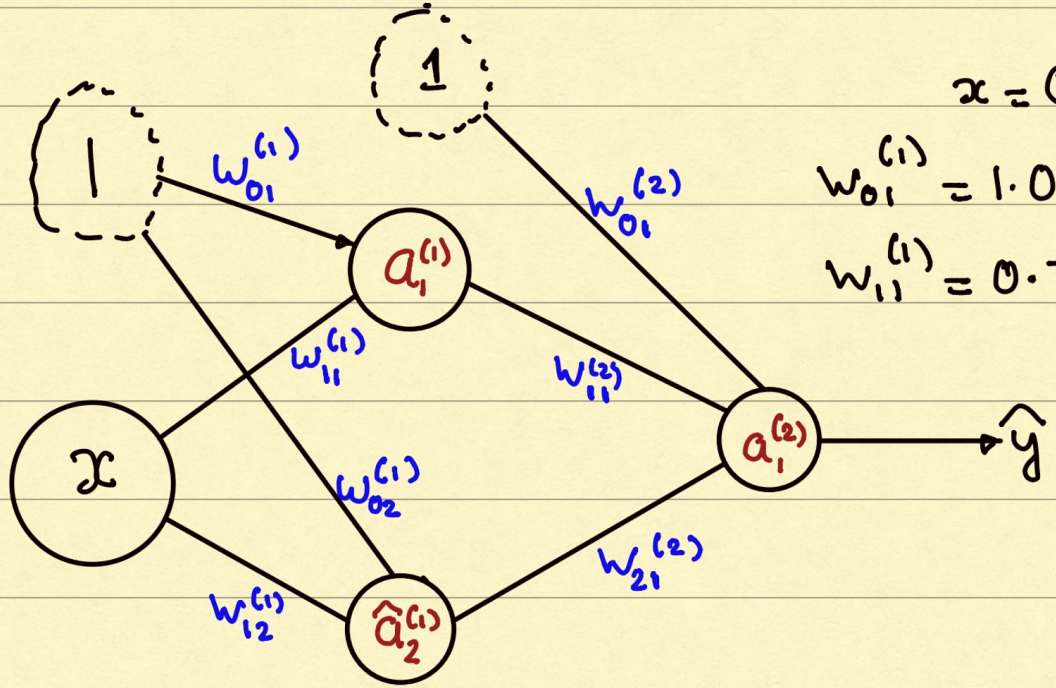
$w_{11}^{(1)} = 0.7 \quad w_{12}^{(1)} = 0.9$

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

$$\hat{a}_1^{(1)} = g(z_1^{(1)}) =$$



$x = 0.5$

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

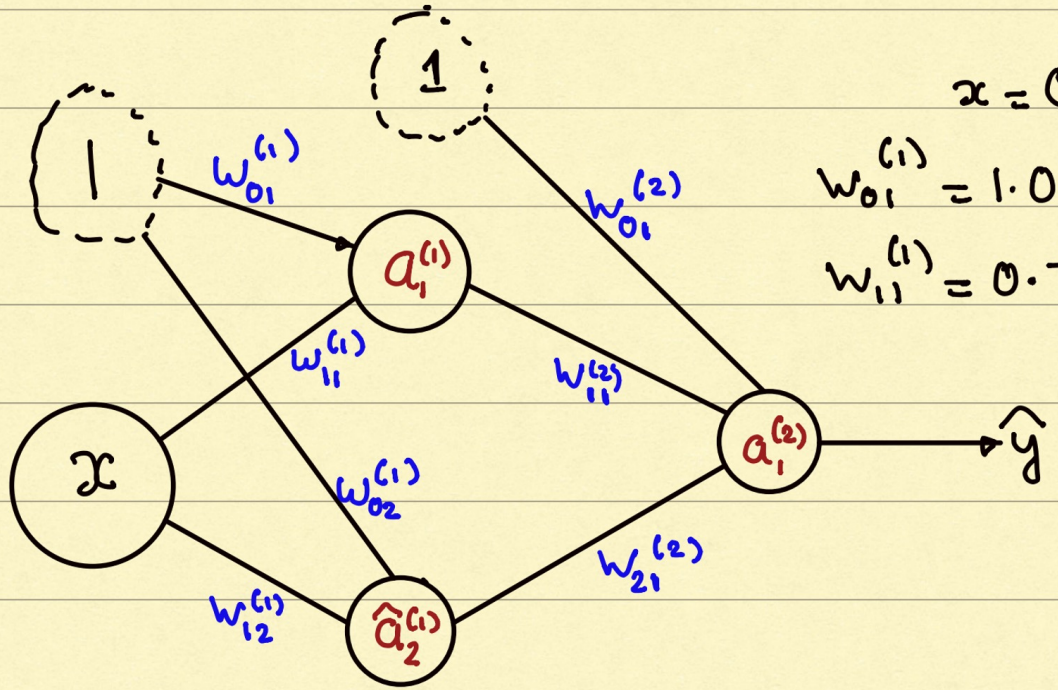
$w_{11}^{(1)} = 0.7 \quad w_{12}^{(1)} = 0.9$

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

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



$x = 0.5$

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

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

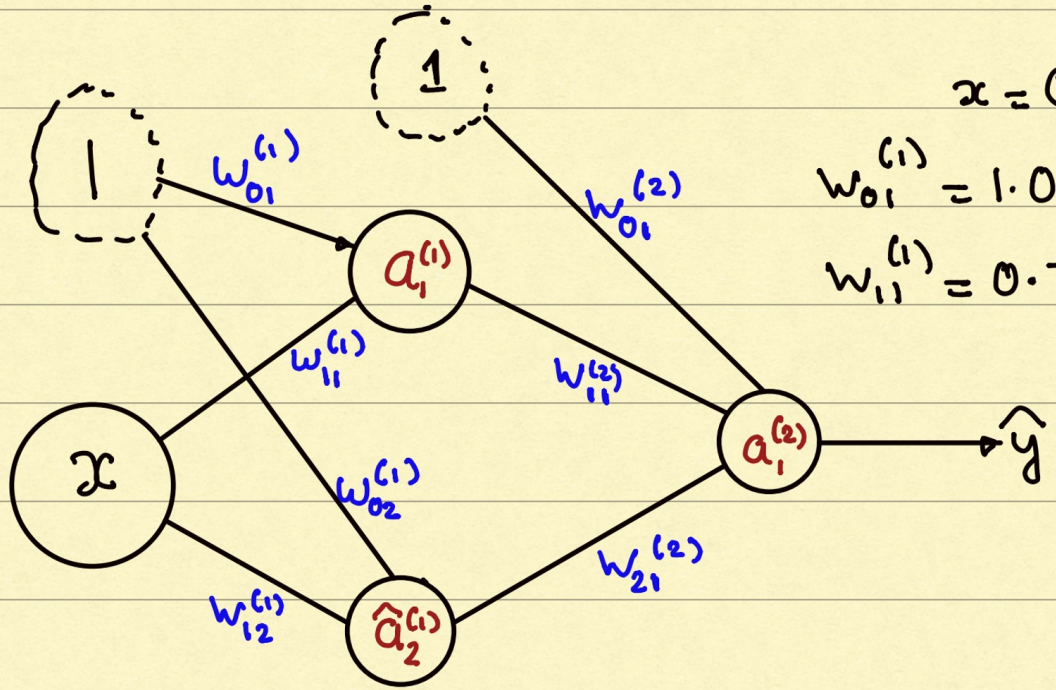
$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

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

$$= g(1 * 1 + 0.7 * 0.5) = g(1.35)$$



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

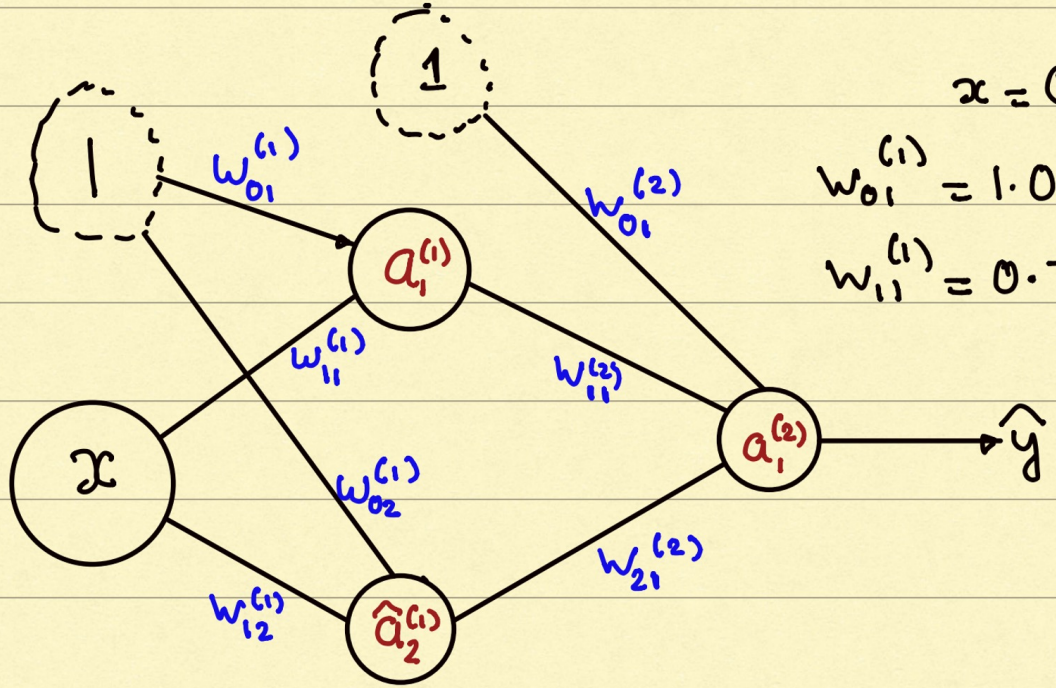
$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

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

$$= g(1 * 1 + 0.7 * 0.5) = g(1.35)$$

($z_1^{(1)}$)



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

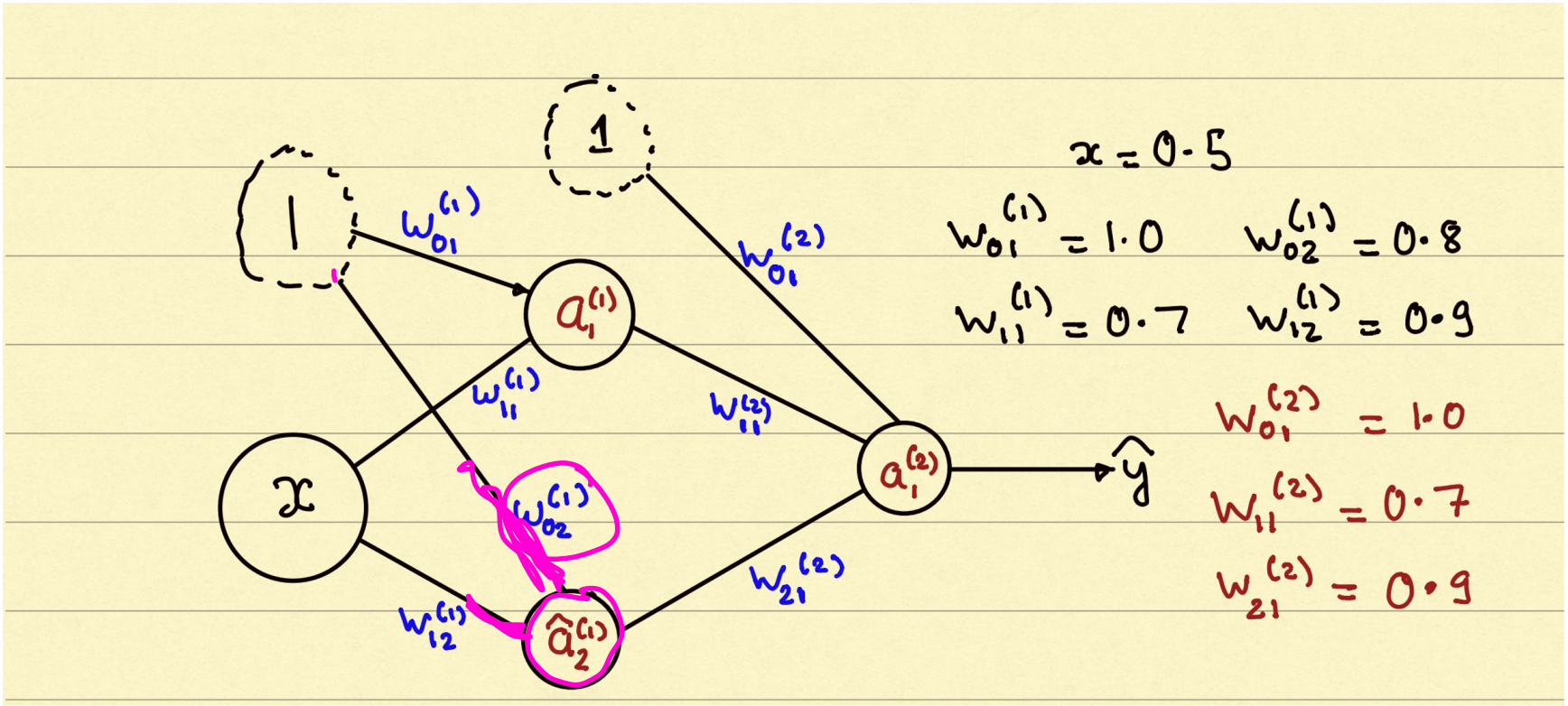
$w_{21}^{(2)} = 0.9$

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

$$= g(1 * 1 + 0.7 * 0.5) = g(1.35)$$

$z_1^{(1)}$

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



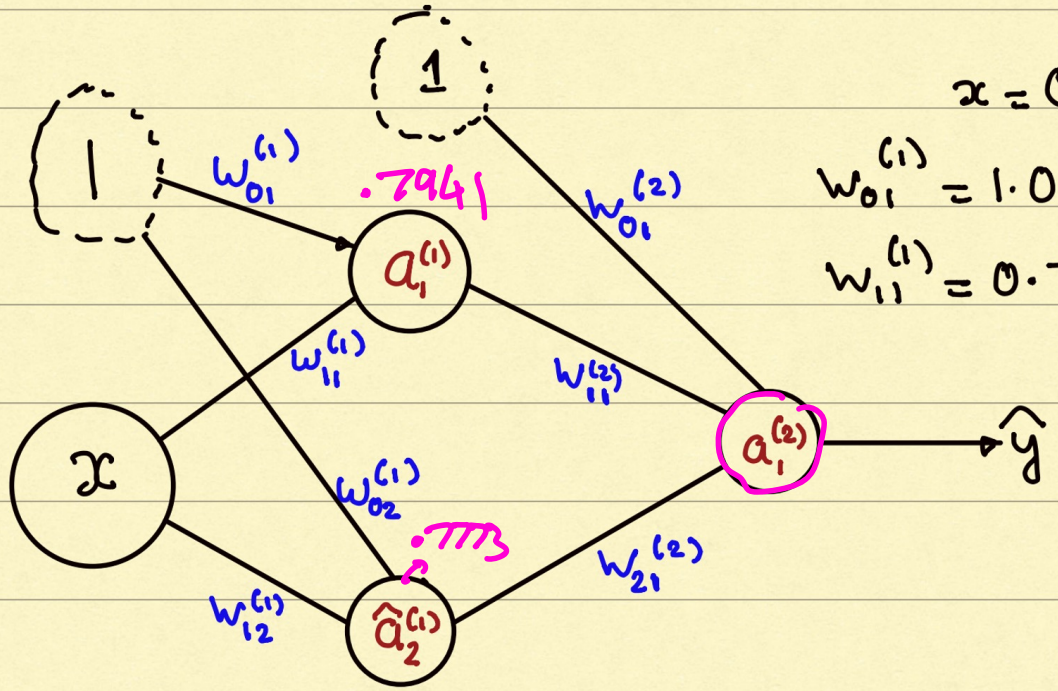
$a_2^{(1)} \rightarrow$ $z_2 \mid a_2^{(1)}$

$z_2^{(1)} = w_{02}^{(1)} * 1 + w_{12}^{(1)} * 0.5$

$0.8 * 1 + 0.9 * .5$

$= .8 + .45 \quad = 1.25$

$a_2^{(1)} = \sigma(1.25)$



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

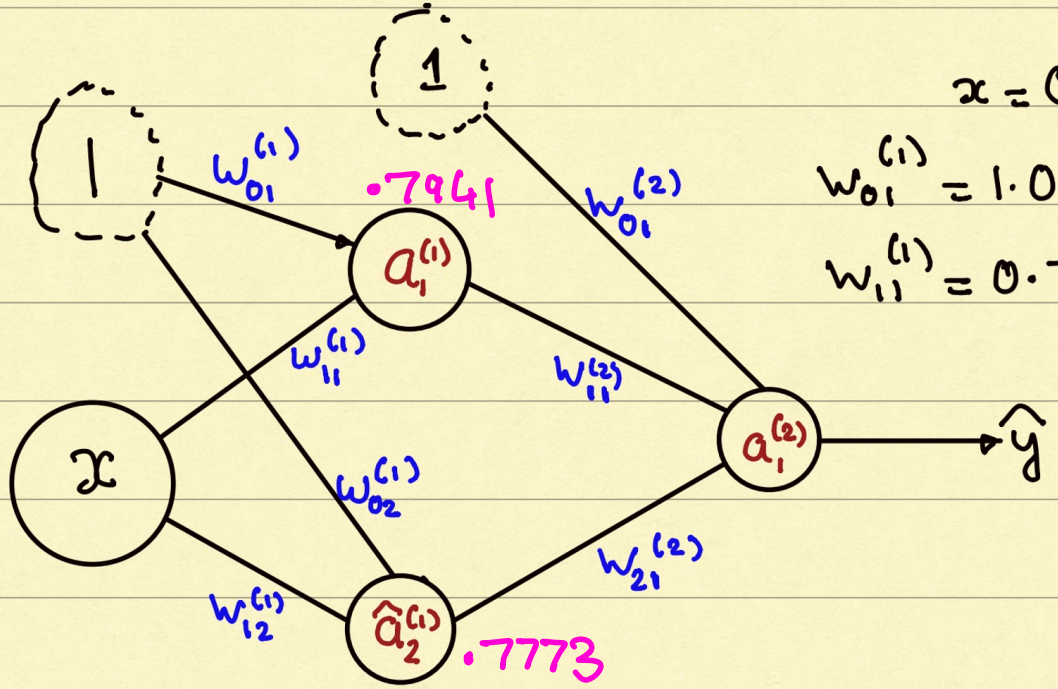
$w_{21}^{(2)} = 0.9$

$$\hat{a}_2^{(1)} = g(z_2^{(1)}); \quad z_2^{(1)} = w_{02}^{(1)} x_0 + w_{12}^{(1)} x_1 = 0.8 + 0.9 * 0.5$$

$$\Rightarrow z_2^{(1)} = 1.25 \Rightarrow \hat{a}_2 = g(z_2^{(1)})$$

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

$$\sigma(z) = \frac{1}{1+\exp(-1.25)}$$



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

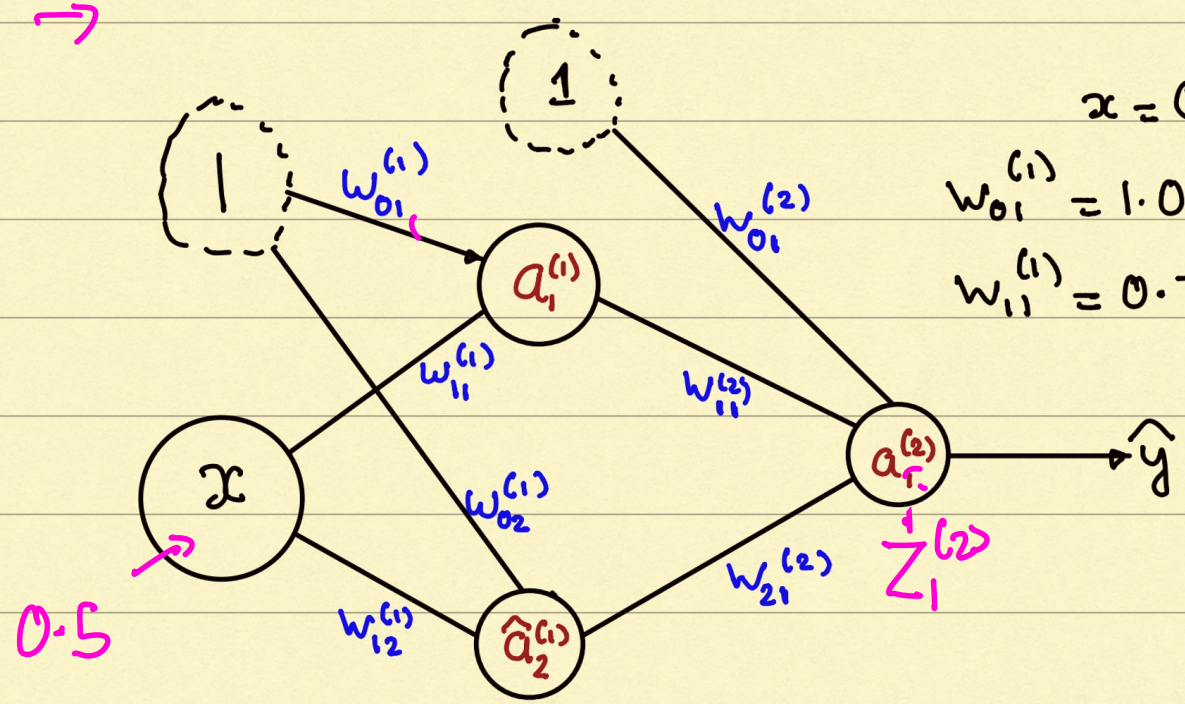
500

Traing

Testy

Deployment (Inference)

$w \rightarrow w - \alpha \frac{\partial J}{\partial w_1} \rightarrow (\hat{y} - y) \cdot x.$



$x = 0.5$

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

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

$w_{01}^{(2)} = 1.0$

$w_{11}^{(2)} = 0.7$

$w_{21}^{(2)} = 0.9$

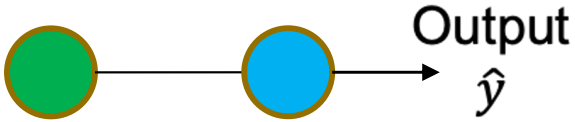
$$\hat{a}_1^{(2)} = g(z_1^{(2)}); \quad z_1^{(2)} = w_{01}^{(2)} x_0 + w_{11}^{(2)} a_1^{(1)} + w_{21}^{(2)} a_2^{(1)}$$

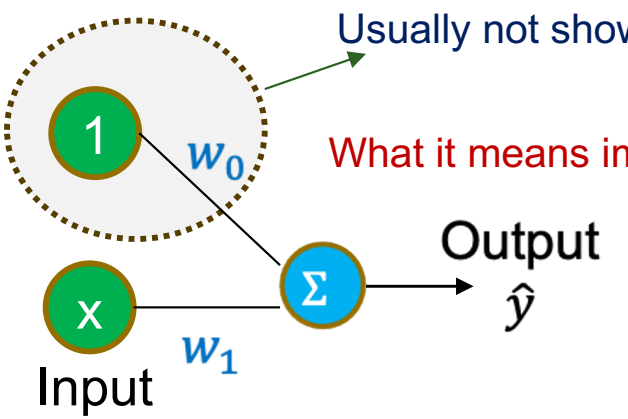
$$\Rightarrow z_1^{(2)} = 1*1 + 0.7*0.7941 + 0.9*0.7773$$

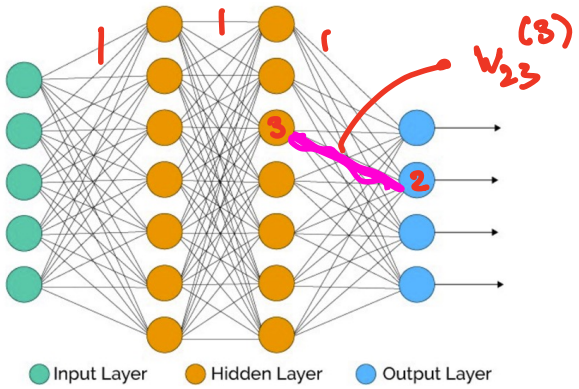
$$= 2.255 \Rightarrow \boxed{a_1^{(2)} = 0.9051} = \hat{y}$$

Weights notation (Actual Notation)

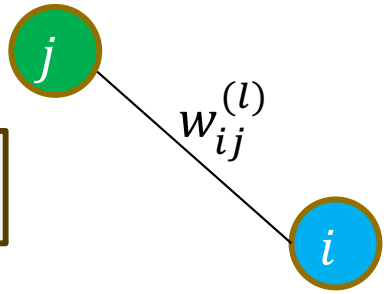
What is shown explicitly







$$w_{ij}^{(l)} \rightarrow w_{\text{to,from}}^{(layer)}$$

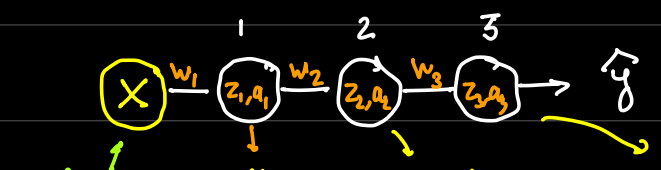


- Constant neurons representing a “feature” with 1 are usually not depicted in the pictorial representation.
- The parameters going from the constant neuron is called a bias and the neuron is called a “bias unit”
- The other parameters are called “weights”

Back prop in a scalar chain

No bias

$a^{(l)} = g(z^{(l)})$



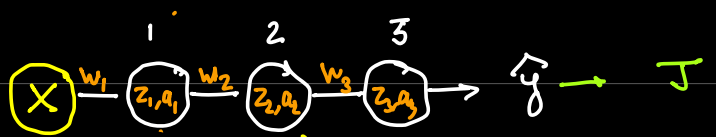
$a^0 = x$ $\quad z^{(1)}, a^{(1)} \quad z^{(2)}, a^{(2)} \quad z^{(3)}, a^{(3)}$

$\boxed{a^{(0)} = x}$

Forward Pass →

$$z^{(1)} = W_1 a^{(0)} \qquad a^{(1)} = g(z^{(1)})$$
$$z^{(2)} = W_2 a^{(1)} \qquad a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = W_3 a^{(2)} \qquad a^{(3)} = g(z^{(3)})$$

$\hat{y} = a^{(3)}$



Nomenclature

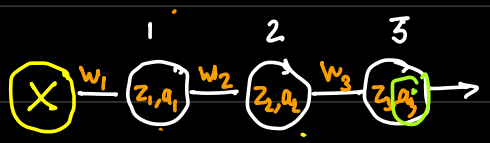
$\frac{\partial J}{\partial w_1}$	$\frac{\partial J}{\partial z_1} = \delta_1$; $\frac{\partial J}{\partial a_1} = e_1$
$\frac{\partial J}{\partial w_2}$	$\frac{\partial J}{\partial z_2} = \delta_2$; $\frac{\partial J}{\partial a_2} = e_2$
$\frac{\partial J}{\partial w_3}$	$\frac{\partial J}{\partial z_3} = \delta_3$; $\frac{\partial J}{\partial a_3} = e_3$

DELTA RULE

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial w_i}$$
$$= \delta^{(l+1)} \, a^{(l)}$$

$$z^{(l+1)} = W_i \, a^{(l)}$$

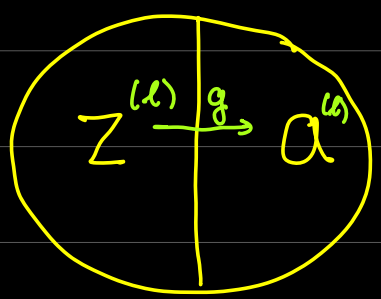
$$\boxed{\frac{\partial J}{\partial w_i} = \delta^{(l+1)} \, a^{(l)}}$$



$$\frac{\partial a}{\partial z} = g'(z)$$

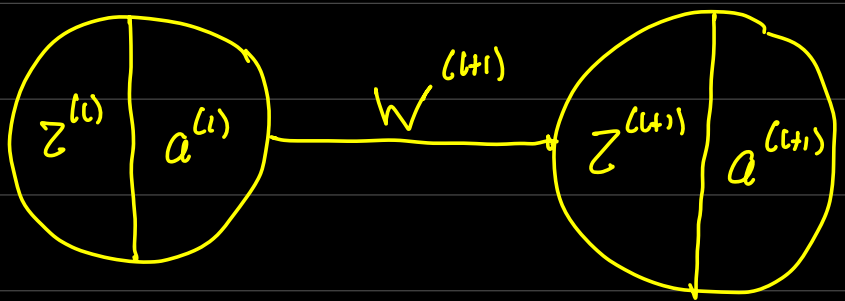
$$a = g(z)$$

$$\frac{\partial J}{\partial z^{(l)}}$$



$$\frac{\partial J}{\partial z^{(l)}} = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}}$$

$$\delta^{(l)} = e^{(l)} \cdot g'(z^{(l)})$$



$$\frac{\partial J}{\partial a^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}} \cdot \frac{\partial z^{(l+1)}}{\partial a^{(l)}}$$

$$e^{(l)} = \delta^{(l+1)} \cdot W^{(l+1)}$$

$$z^{(l+1)} = W^{(l+1)} a^{(l)}$$

$$g(z) = \sigma(z)$$

$$g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = a*(1-a) = \sigma*(1-\sigma)$$

$$\frac{\partial J}{\partial W^{(l)}} = \delta^{(l)} a^{(l-1)} \quad ; \quad \delta^{(l)} = e^{(l)} g'(z^{(l)})$$

$$e^{(l)} = \delta^{(l+1)} W^{(l+1)}$$

$$\sigma(z)*(1-\sigma(z)) \rightarrow a*(1-a)$$

Start with $\frac{\partial J}{\partial a_3} = \frac{\partial J}{\partial \hat{y}}$; $J = \frac{1}{2}(\hat{y}-y)^2$

$$\Rightarrow \frac{\partial J}{\partial \hat{y}} = \hat{y} - y$$

We know e_3

$$1-a = 1 - \frac{1}{1+e^{-z}}$$

$\Rightarrow$

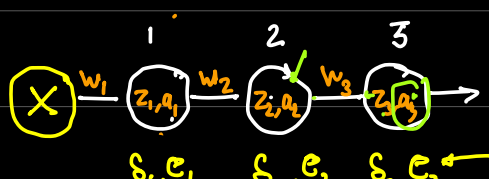
$$a = g(z) = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{1+e^{-z}-1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}}$$

$$g'(z) = \frac{dg}{dz} = \frac{-1}{(1+e^{-z})^2} * -e^{-z}$$

$$g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = \frac{e^{-z}}{1+e^{-z}} * \frac{1}{1+e^{-z}}$$

$$= \frac{e^{-z}}{1+e^{-z}} \cdot a$$

$$g'(z) = a(1-a) = \sigma(z)[1-\sigma(z)]$$



$x \xrightarrow{w} \hat{y}$ Forward Pass

$$z^{(l)} = w^{(l)} a^{(l-1)}; \quad a^{(l)} = g(z^{(l)})$$

$$\frac{\partial J}{\partial a_3} = \hat{y} - y$$

$$\delta^{(l)} = e^{(l)} \cdot g'(z^{(l)});$$

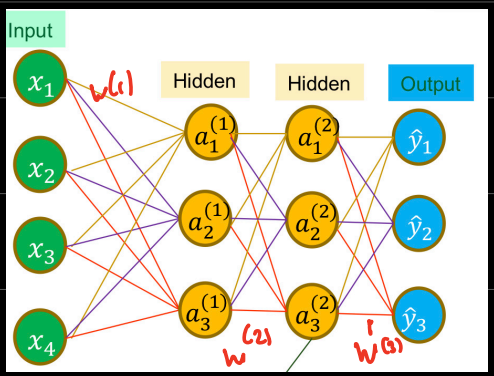
$$e^{(l)} = w^{(l+1)} \delta^{(l+1)}$$

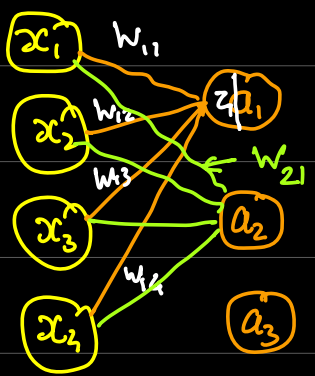
$$\frac{\partial J}{\partial w} \leftarrow J$$

$$\boxed{\frac{\partial J}{\partial w^{(l)}} = \delta^{(l)} a^{(l-1)}}$$

MLP - Back prop.

Skippy bias





$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}_{3\times 1} = \begin{aligned} &W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + W_{14}x_4 \\ &W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + W_{24}x_4 \\ &W_{31}x_1 + W_{32}x_2 + W_{33}x_3 + W_{34}x_4 \end{aligned}$$

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \\ z_3^{(1)} \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

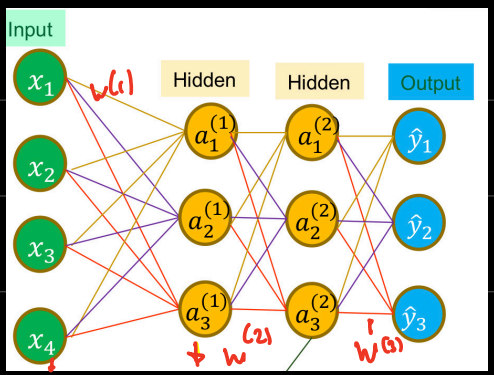
$$\underset{3\times 1}{Z^{(1)}} = \underset{3\times 4}{W^{(1)}}\ \underset{4\times 1}{x}$$

$$a^{(1)} = g\begin{bmatrix} (z_1^{(1)}) \\ (z_2^{(1)}) \\ (z_3^{(1)}) \end{bmatrix}$$

$$a^{(1)} = g(z^{(1)})$$

↳ for every component

$$a^{(0)} = x$$



$$z^{(1)} = \underset{3\times 4}{W^{(1)}} a^{(0)} \qquad a^{(1)} = g(z^{(1)})$$

$$\underset{3\times 1}{z^{(2)}} = \underset{3\times 3}{W^{(2)}} \underset{3\times 1}{a^{(1)}} \qquad a^{(2)} = g(z^{(2)})$$

$$z^{(3)} = W^{(3)} a^{(2)} \qquad a^{(3)} = g(z^{(3)})$$

$$\hat{y} = a^{(3)}$$

$a_1^{(2)}$ → $\hat{y}_1(a_1^{(3)})$

$a_2^{(2)}$ → $\hat{y}_2(a_2^{(3)})$

$a_3^{(2)}$ → $\hat{y}_3(a_3^{(3)})$

$\} \rightarrow J$

$$\frac{\partial J}{\partial z} = \delta \qquad \frac{\partial J}{\partial a} = e$$

$$\underset{3\times 1}{e^{(3)}} = \frac{\partial J}{\partial a^{(3)}} = \frac{\partial J}{\partial \hat{y}}$$

↳ Can be calculated

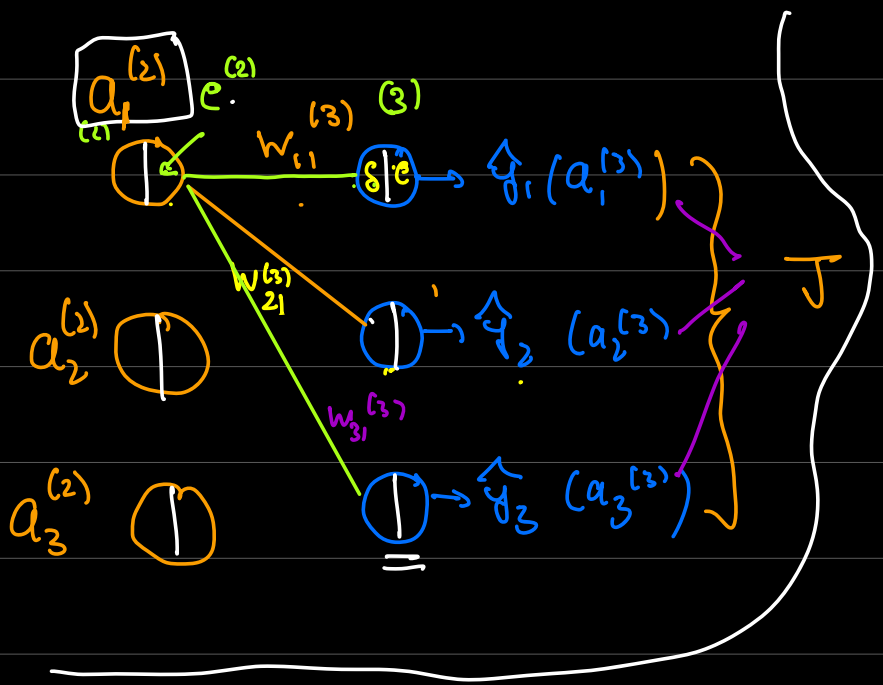
e.g MSE $\Rightarrow e = \hat{y} - y$

$$a^{(3)} = g(z^{(3)}) \Rightarrow \frac{\partial a^{(3)}}{\partial z^{(3)}} = g'(z^{(3)})$$

$$\frac{\partial J}{\partial z^{(3)}} = \frac{\partial J}{\partial a^{(3)}} \frac{\partial a^{(3)}}{\partial z^{(3)}}$$

$$\underset{3\times 1}{\delta^{(3)}} = \underset{3\times 1}{e^{(3)}} \odot \underset{3\times 1}{g'(z^{(3)})}$$

$\odot$ ↓ Element-wise



$$\frac{\partial J}{\partial a_1^{(2)}} = \frac{\partial J}{\partial z_1^{(3)}} \cdot \frac{\partial z_1^{(3)}}{\partial a_1^{(2)}} + \frac{\partial J}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial a_1^{(2)}} + \frac{\partial J}{\partial z_3^{(3)}} \cdot \frac{\partial z_3^{(3)}}{\partial a_1^{(2)}}$$

$$e_1^{(2)} = \delta_1^{(3)} W_{11}^{(3)} + \delta_2^{(3)} W_{21}^{(3)} + \delta_3^{(3)} W_{31}^{(3)}$$

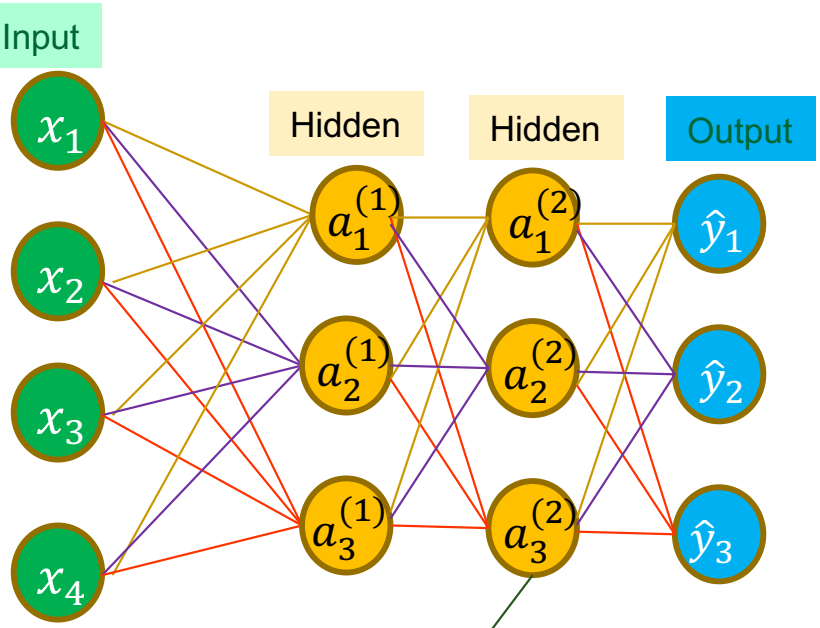
$$e_2^{(2)} = \delta_1 W_{12} + \delta_2 W_{22} + \delta_3 W_{32}$$

$$e_3^{(2)} = \delta_1 W_{13} + \delta_2 W_{23} + \delta_3 W_{33}$$

$$\Rightarrow e^{(2)} = \underbrace{\begin{bmatrix} W_{11} & W_{21} & W_{31} \\ W_{12} & W_{22} & W_{32} \\ W_{13} & W_{23} & W_{33} \end{bmatrix}}_{W^T} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}$$

$$\Rightarrow \boxed{e^{(2)} = W^T \delta^{(3)}}$$

Forward pass through a deep network



Main Ideas

Every neuron has a linear and a nonlinear operation

The linear intermediate output is denoted by $z_i^{(l)}$

The nonlinear final output is denoted by $a_i^{(l)}$

$$z_i^{(l)} = \Sigma w_{ij} a_i^{(l-1)}$$
$$a_i^{(l)} = g\left(z_i^{(l)}\right)$$

} Scalar

Linear

$$z_1^{(2)} = W_{11}a_1^{(1)} + W_{12}a_2^{(1)} + W_{13}a_3^{(1)}$$
$$z_2^{(2)} = W_{21}a_1^{(1)} + W_{22}a_2^{(1)} + W_{23}a_3^{(1)}$$
$$z_3^{(2)} = W_{31}a_1^{(1)} + W_{32}a_2^{(1)} + W_{33}a_3^{(1)}$$

$$\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & W_{13} \\ W_{21} & W_{22} & W_{23} \\ W_{31} & W_{32} & W_{33} \end{bmatrix} \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_3^{(1)} \end{bmatrix}$$

$$\boldsymbol{z^{(2)} = W a^{(1)}}$$

Nonlinear

$$a_1^{(2)} = g(z_1^{(2)})$$
$$a_2^{(2)} = g(z_2^{(2)})$$
$$a_3^{(2)} = g(z_3^{(2)})$$

$$\begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} = g\left(\begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}\right)$$

$$\boldsymbol{a^{(2)} = g(z^{(2)})}$$

Many choices for the nonlinear activation function g are possible in the intermediate layers

In the output layer g depends on the task.

Forward Pass

Step 1. $\boldsymbol{z^{(1)} = Wx}$ $\quad \boldsymbol{a^{(1)} = g(z^{(1)})}$

Step 2. $\boldsymbol{z^{(2)} = Wa^{(1)}}$ $\quad \boldsymbol{a^{(2)} = g(z^{(2)})}$

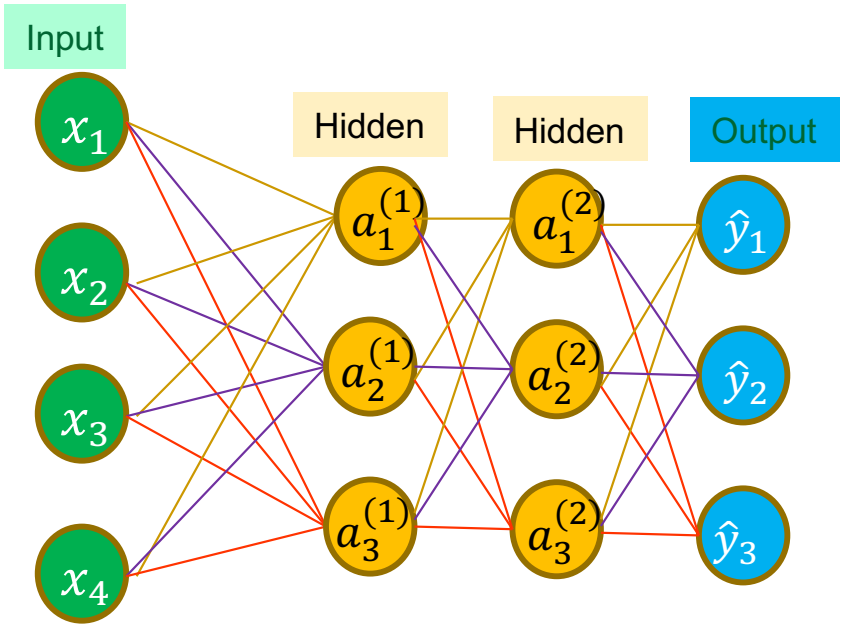
Step 3. $\boldsymbol{z^{(3)} = Wa^{(2)}}$ $\quad \boldsymbol{a^{(3)} = g(z^{(3)})}$

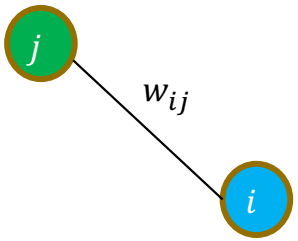
$\boldsymbol{a^{(0)} = x}$

$\hat{y} = \boldsymbol{a^{(3)}}$

For N_w weights, number of multiplications in forward pass $\boldsymbol{F \approx O(N_w)}$

FDM or the need for backprop





Finite Difference

$$\frac{\partial J}{\partial w_{ij}} = \frac{\left(J(w_{11}, \dots, \boldsymbol{w_{ij} + \Delta w_{ij}}, \dots) - J(w_{11}, \dots, \boldsymbol{w_{ij}} \dots)\right)}{\boldsymbol{\Delta w_{ij}}}$$

- The forward pass gave us $\hat{y}$ as well as all the $\hat{a}$ (in the hidden layers)
  - Linear calculations : $\boldsymbol{z = Wa}$ which were matrix multiplications
  - Nonlinear calculations : $\boldsymbol{a = g(z)}$ which are element wise calculations

Forward Pass

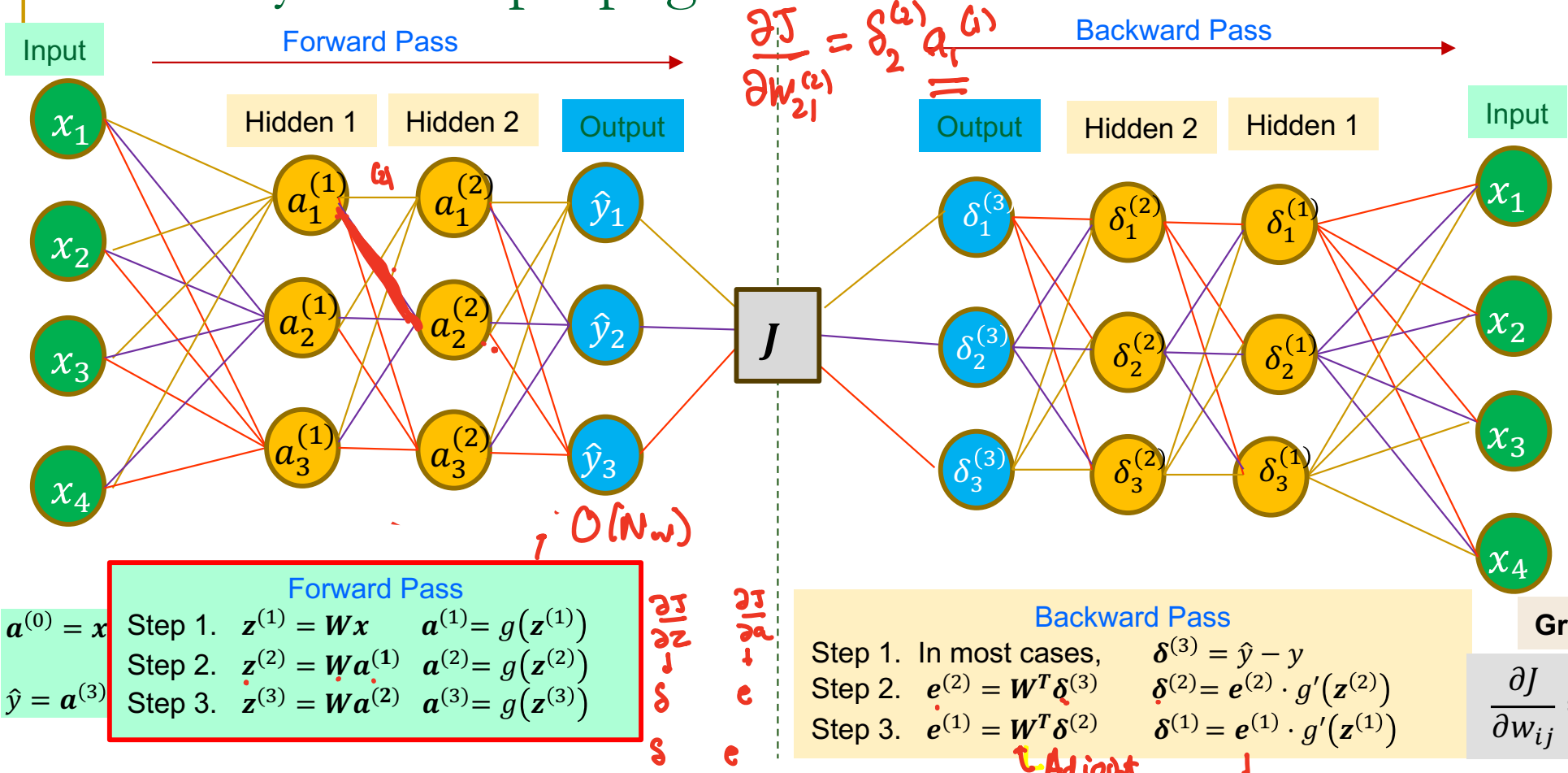
Step 1. $\boldsymbol{z^{(1)} = Wx}$ $\quad \boldsymbol{a^{(1)}} = g(\boldsymbol{z^{(1)}})$

Step 2. $\boldsymbol{z^{(2)} = Wa^{(1)}}$ $\quad \boldsymbol{a^{(2)}} = g(\boldsymbol{z^{(2)}})$

Step 3. $\boldsymbol{z^{(3)} = Wa^{(2)}}$ $\quad \boldsymbol{a^{(3)}} = g(\boldsymbol{z^{(3)}})$

- What we need for gradient descent is $w = w - \alpha \frac{\partial J}{\partial w_{ij}}$ for every w_{ij} in every iteration
- **Finite Difference Method (FDM)** Inefficient way of computing $\frac{\partial J}{\partial w_{ij}}$
  - Remember that gradient descent takes many iterations (epochs) to converge.
  - FDM will take $(\boldsymbol{N_w + 1}) \times F$ calculations every iteration. (**1 base + N_w perturbed forward passes**)
  - N_w is the number of weights and F is the computational expense of one forward pass
  - $\boldsymbol{O(N_w^2)}$ Very expensive for big networks. Solution is backprop which is implemented in modern networks

Summary of backpropogation in MLPs



$a^{(0)} = x$

$\hat{y} = a^{(3)}$

Forward Pass

Step 1. $z^{(1)} = Wx$ $a^{(1)} = g(z^{(1)})$

Step 2. $z^{(2)} = Wa^{(1)}$ $a^{(2)} = g(z^{(2)})$

Step 3. $z^{(3)} = Wa^{(2)}$ $a^{(3)} = g(z^{(3)})$

Backward Pass

Step 1. In most cases, $\delta^{(3)} = \hat{y} - y$

Step 2. $e^{(2)} = W^T \delta^{(3)}$ $\delta^{(2)} = e^{(2)} \cdot g'(z^{(2)})$

Step 3. $e^{(1)} = W^T \delta^{(2)}$ $\delta^{(1)} = e^{(1)} \cdot g'(z^{(1)})$

Adjoint

$O(N_w)$

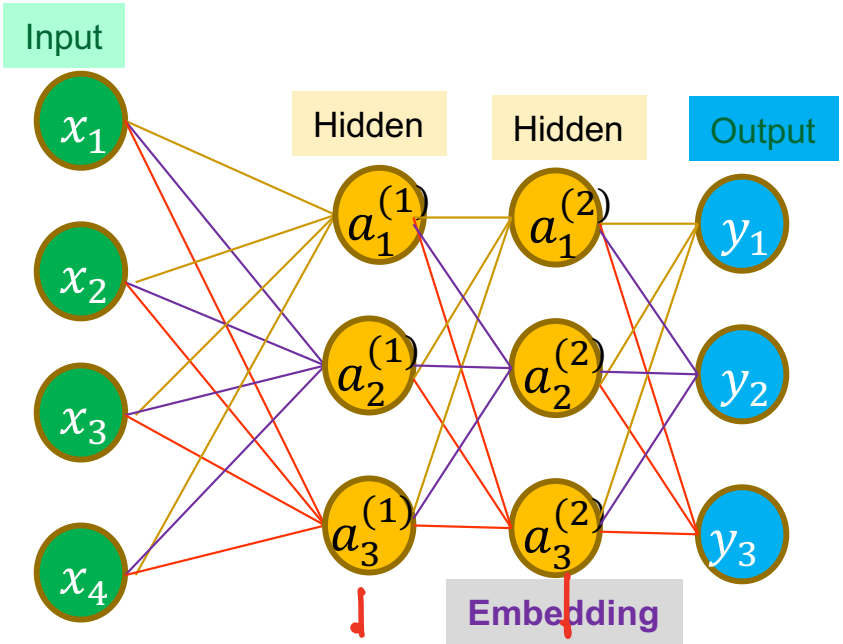
Gradient

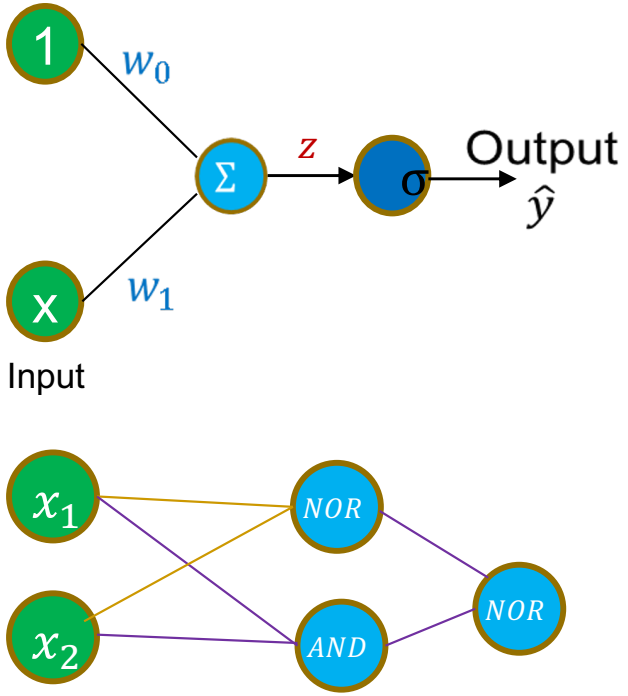
$$\frac{\partial J}{\partial w_{ij}} = \delta_i a_j$$

- The above expressions hold true for MLPs – Multi Layer Perceptrons.
- **MLP** – Fully connected network in multiple, sequential layers.
- Notice the symmetry between the forward pass and the backward pass.
- Both involve a sequence of Linear (Matrix multiplication) followed by a nonlinear step ($g(z)$ or $g'(z)$)
- Other Deep Neural Networks need not have such need formulae but the overall symmetry remains

For N_w weights, number of multiplications in backprop $\boldsymbol{F \approx O(N_w)}$

Embeddings – Or how deep neural networks work



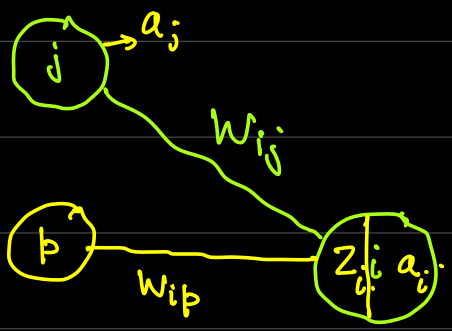


- Every "deep" network has, in essence, **a prefix** to a final linear, logistic, or softmax layer
  - That is, a linear regression or a linear classification problem
- The task of all the prefix (hidden) layers is to transform the problem into a representation which will work linearly.
- That is, deep networks learn representations that work simply with final layers
- The final representation before the output layer is called the **Embedding**
- **Embeddings** are like zipped, vector versions of the input
  - Of great practical importance. OpenAI "sells" embeddings for ChatGPT currently

$$z^{(3)} = W \; a^{(2)} \quad ; \quad a^{(2)} = W^{-1} \; z^{(3)}$$

$$\frac{\partial J}{\partial a^{(2)}} = W^{T} \; \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial z} \; , \; \frac{\partial J}{\partial a} \qquad \delta^{(2)} = W^{T} \; \delta^{(3)}$$



$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial z_i} \; \frac{\partial z_i}{\partial w_{ij}}$$

δ_i $\quad$ a_j

$$\boxed{\frac{\partial J}{\partial w_{ij}} = \delta_i \; a_j}$$

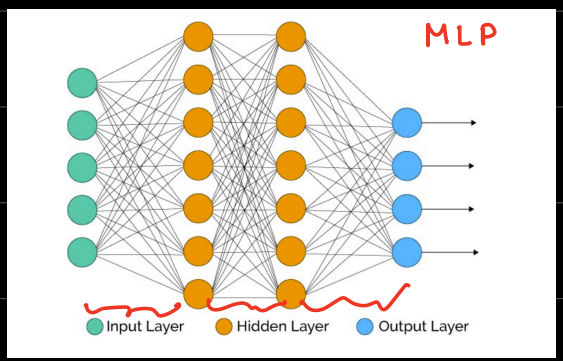
"DELTA" RULE

$$z_i = w_{ij} \; a_j + w_{ip} \; a_p$$

$$\frac{\partial z_i}{\partial w_{ij}} = a_j + 0$$

RECAP

Expressions in the general, MLP case



Final Layer : $e = \frac{\partial J}{\partial \hat{y}}$

Hidden Layers :

$$e^{(k)} = W^T \delta^{(k+1)}$$

$$\delta^{(k)} = g'(z) \odot e^{(k)}$$

$\frac{\partial J}{\partial w} = \Delta w$

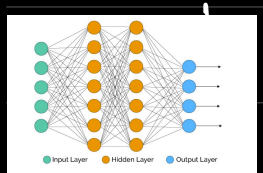
$\frac{\partial J}{\partial a} = e$

Back prop Gradient $\frac{\partial J}{\partial z} = \delta$

Full Algorithm for Backprop weight update

1. Initialize Weights Randomly
2. Perform Forward Prop till final layer → For every data pt

Linear: $z^{(k)} = W^{(k)} a^{(k-1)}$ Nonlinear: $a^{(k)} = g(z^{(k)})$

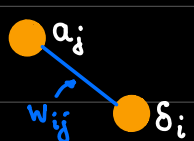


3. Calculate $e = \frac{\partial J}{\partial \hat{y}}$ at the output of the final layer
4. Perform Backprop till first layer

Nonlinear: $\delta^{(k)} = g'(z^{(k)}) \odot e^{(k)}$ ($\frac{\partial J}{\partial z}$, $\frac{\partial J}{\partial a}$)

Linear: $e^{(k)} = W^{T(k+1)} \delta^{(k+1)}$ ($\frac{\partial J}{\partial a}$, $\frac{\partial J}{\partial z}$)

5. Use the "Delta" Rule to calculate the weight update



$$\frac{\partial J}{\partial w_{ij}} = \delta_i a_j \equiv \Delta w_{ij}$$

6. Use Gradient Descent to update w_{ij}

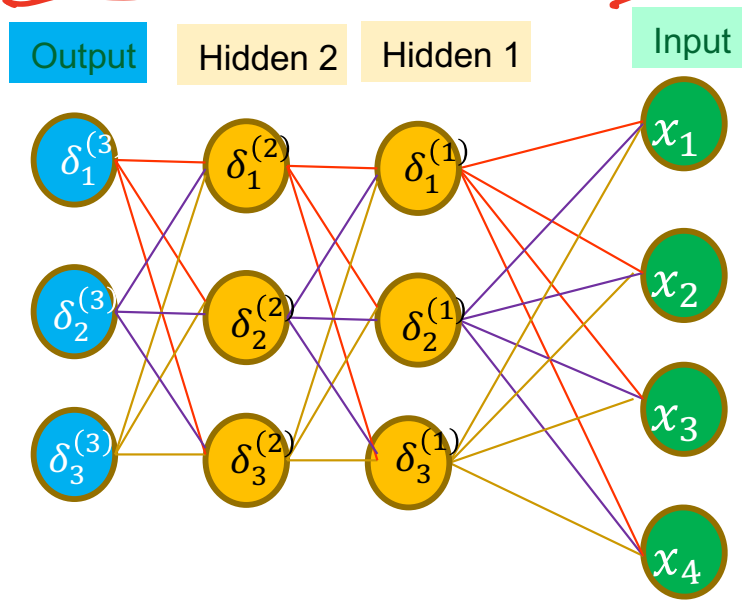
$$w_{ij} = w_{ij} - \alpha \Delta w_{ij}$$

7. Repeat Steps 2-4 for all data pts → One Epoch
8. Repeat Steps 2-5 till convergence → Multiple Epochs

Problems in Deep Networks : Vanishing Gradients (↓ Activation Function), Exploding Gradients (↓ RNNs), Overfitting (↓ Regularization ↓ Dropout)

Practical Consideration – Activation Functions, Nonlinearity

- When people tried to train deep networks with $g = \sigma$, the weights did not update
- Reason – In the backprop formula, we have $g'(z)$ used recursively
- This means that if $g'(z) < 1$, the gradient rapidly decays over the layers – Vanishing Gradient Problem
- Solution is to use other nonlinearities $g(z) = tanh$ and $g(z) = \text{ReLU(z)}$



Backward Pass

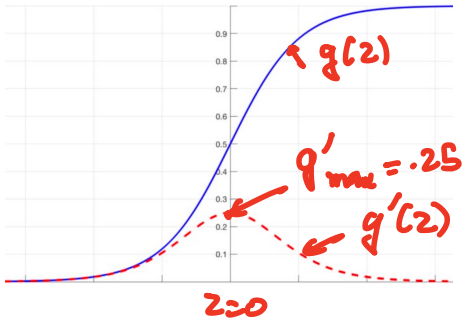
Step 1. In most cases, $\boldsymbol{\delta^{(3)} = \hat{y} - y}$

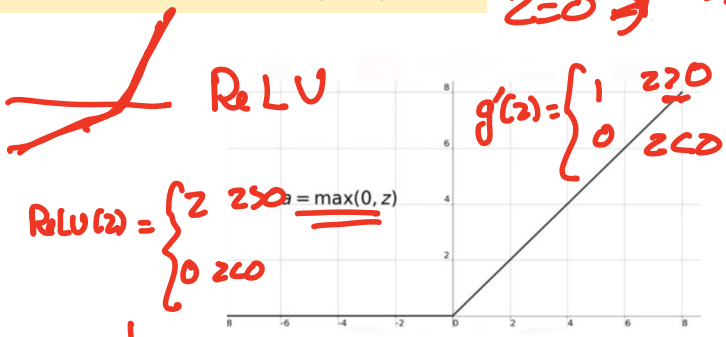
Step 2. $\boldsymbol{e^{(2)} = W^T \delta^{(3)}}$ $\quad \boldsymbol{\delta^{(2)} = e^{(2)} \cdot g'(z^{(2)})}$

Step 3. $\boldsymbol{e^{(1)} = W^T \delta^{(2)}}$ $\quad \boldsymbol{\delta^{(1)} = e^{(1)} \cdot g'(z^{(1)})}$

$g(z) = \tanh(z)$

$\frac{e^z + e^{-z}}{e^z - e^{-z}}$





$g(z) = \sigma$; $g'(z) = \sigma(1-\sigma)$

$.5 * (1-.5) = .25$

ReLU (CNN) > tanh (RNN) | σ → Only in the final layer