

# Machine Learning

## Neural Networks

School of Science and Engineering

[https://www.zubairkhalid.org/ee514\\_2025.html](https://www.zubairkhalid.org/ee514_2025.html)

# Outline

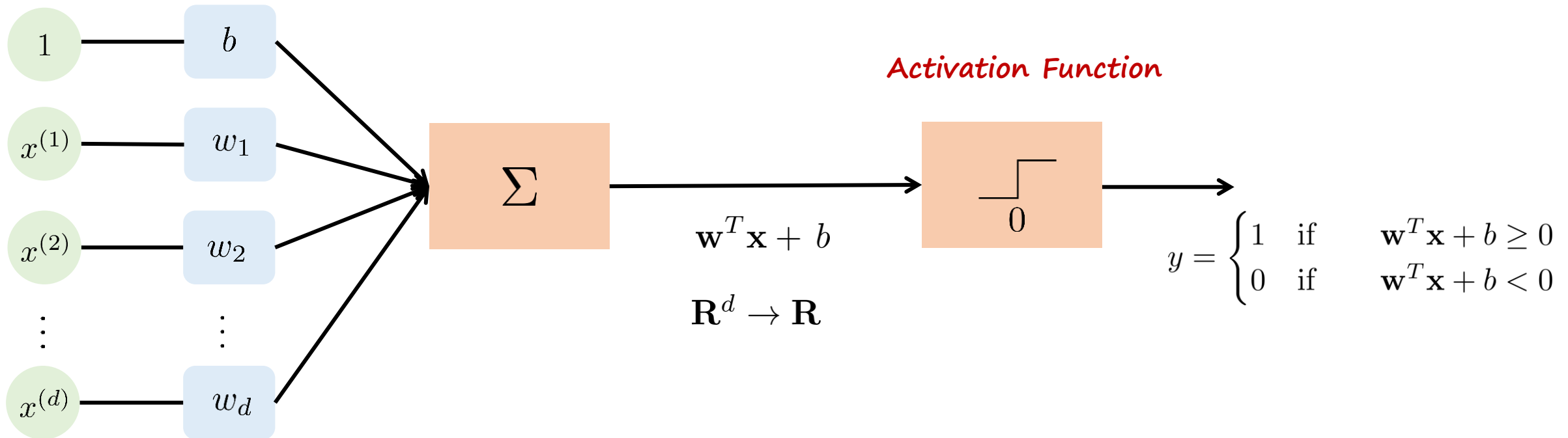
- Neural networks connection with perceptron and logistic regression
- Neural networks notation
- Neural networks 'Forward Pass'
- Activation functions
- Learning neural network parameters
  - Back Propagation.

# Neural Networks

## Connection with Logistic Regression and Perceptron:

- $d$  number of real-valued inputs  $x^{(1)}, x^{(2)}, \dots, x^{(d)} \in \mathbf{R}$ .
- Boolean output,  $y \in \{0, 1\}$ .

### Perceptron Model:

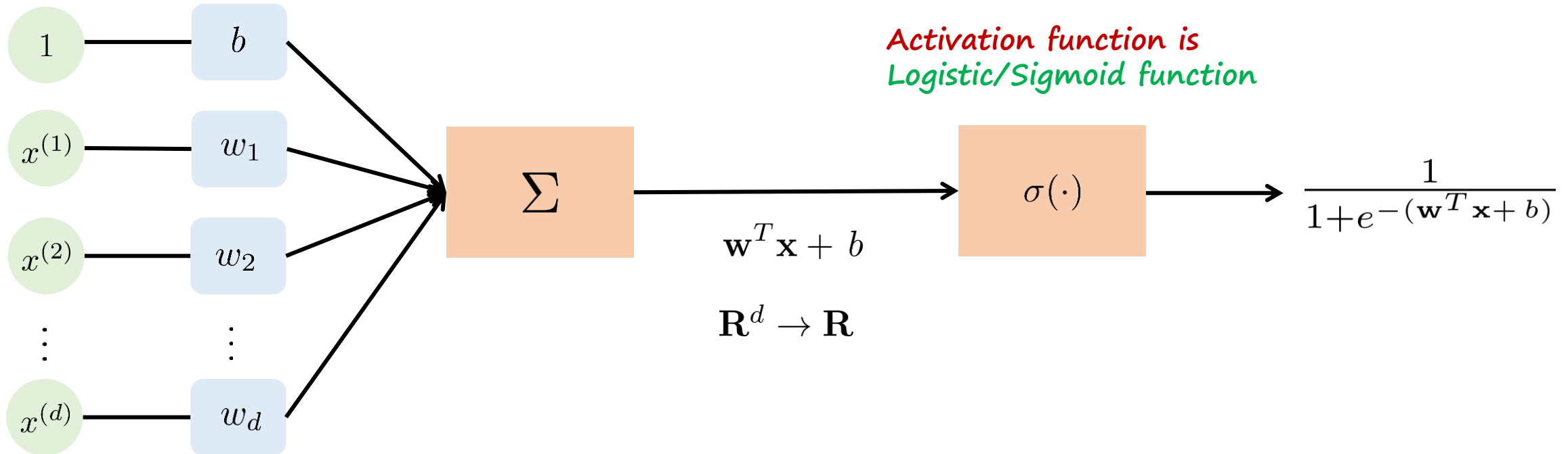


# Neural Networks

## Connection with Logistic Regression and Perceptron:

- $d$  number of real-valued inputs  $x^{(1)}, x^{(2)}, \dots, x^{(d)} \in \mathbf{R}$ .
- Boolean output,  $y \in \{0, 1\}$ .

### Logistic Regression Model:

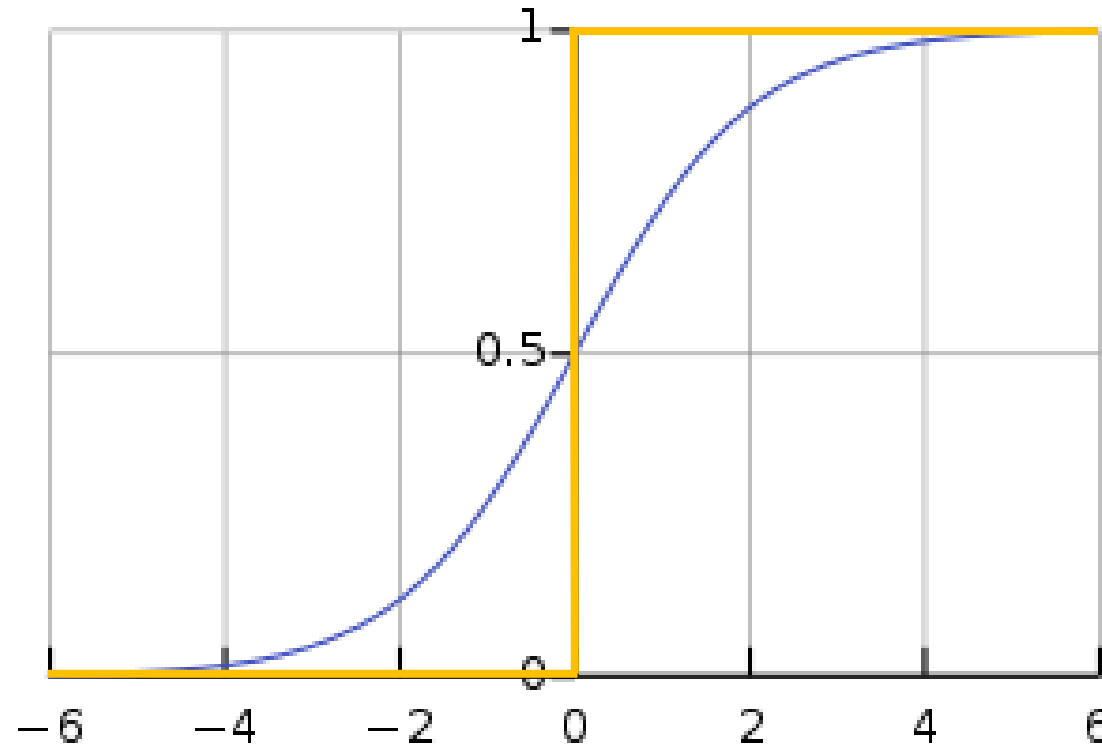


Logistic Regression Model, aka Sigmoid Neuron

# Neural Networks

## Connection with Logistic Regression and Perceptron:

Activation Function  
Perceptron vs Sigmoid Neuron



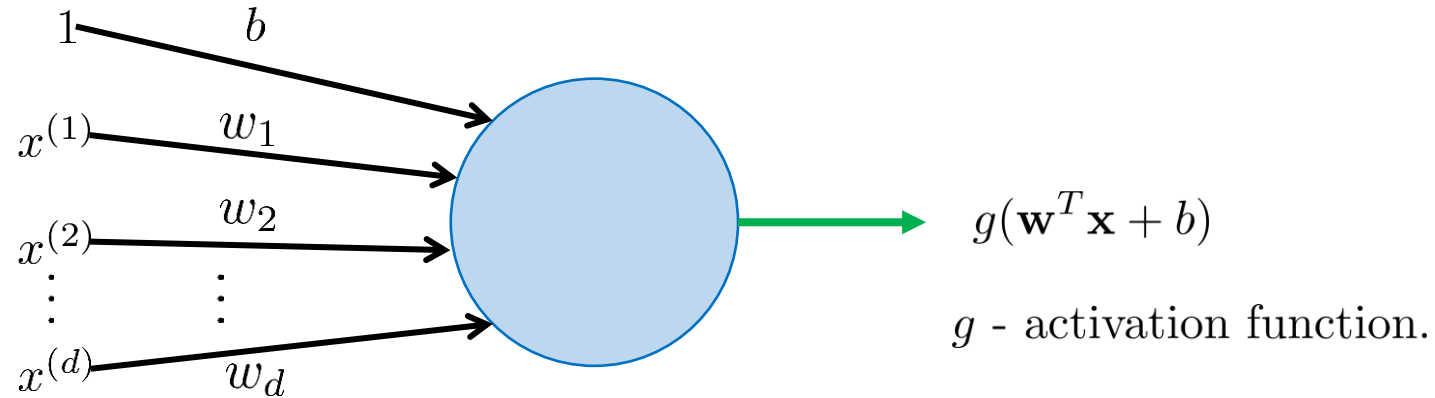
Weighted sum of inputs + bias

$$w^T \mathbf{x} + b$$

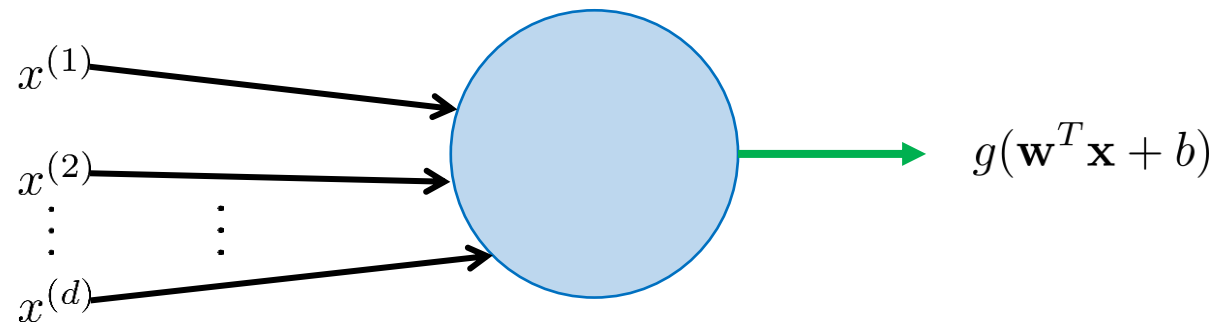
# Neural Networks

## Neuron Model:

### Compact Representation:



### More Compact Representation:



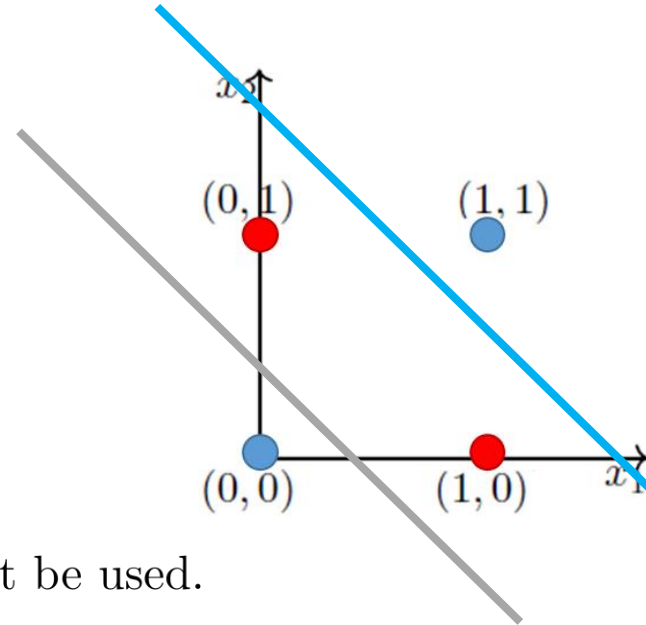
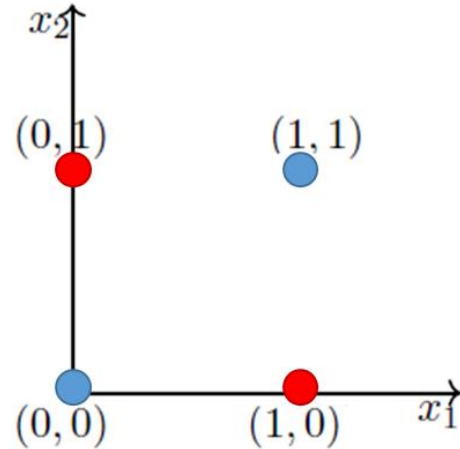
- Neuron model: Characterized by weights, bias and activation function.
- Weights  $\mathbf{w}$ , bias  $b$  - model parameters
- Activation function  $g$  - hyperparameter

# Neural Networks

## Neural Networks - Infamous XOR Problem:

- (1969) Minsky and Papert showed that a perceptron cannot classify XOR output.

$x_1$	$x_2$	XOR
0	0	0
1	0	1
0	1	1
1	1	0

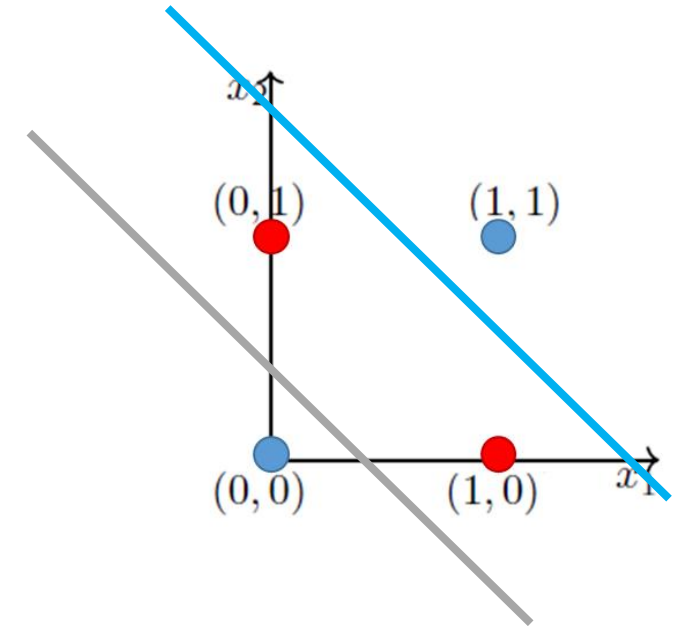
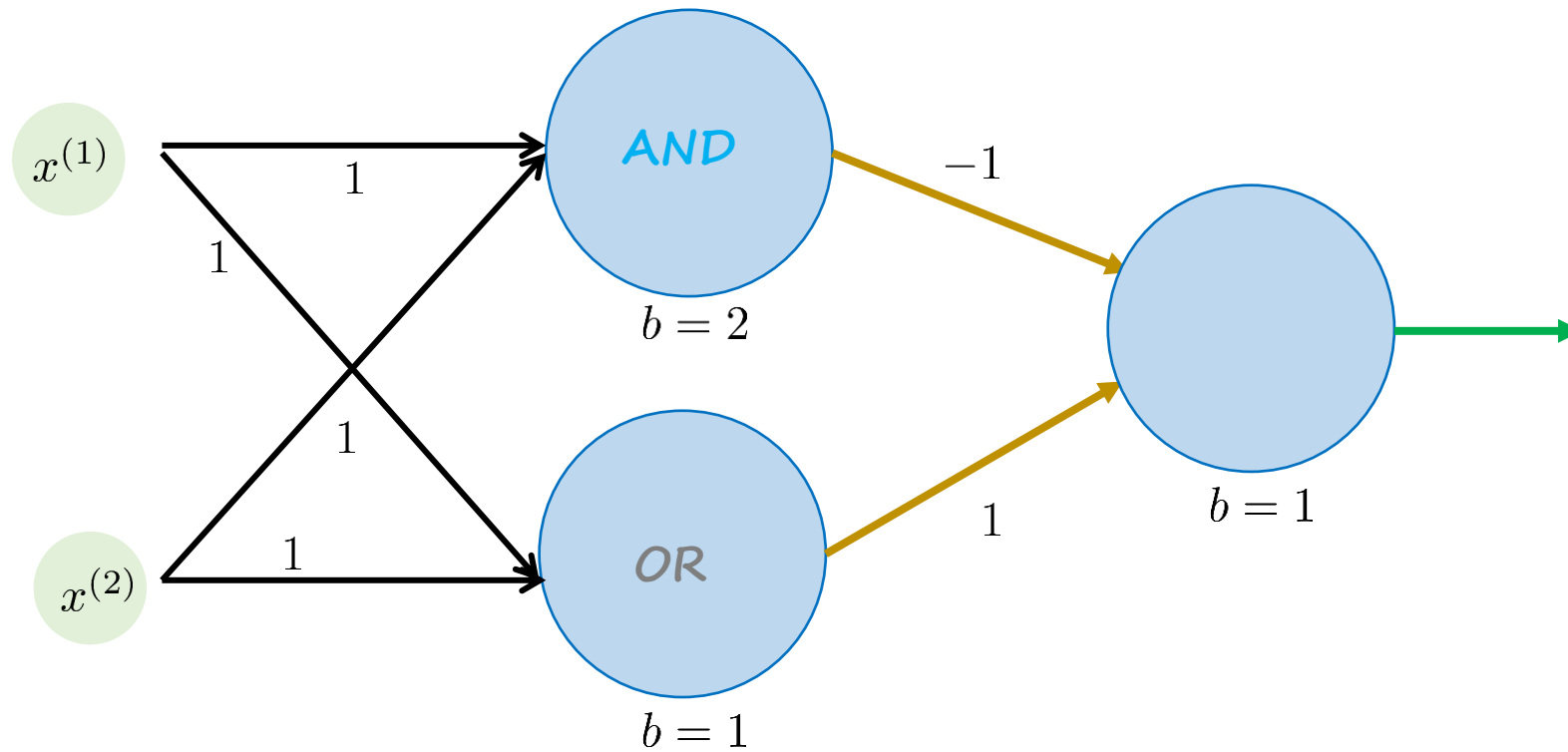


*Idea:*  
Learn AND and OR  
boundaries.

- Classes are not linearly separable: linear classifier cannot be used.
- We can either transform features or project the data to higher dimensional space.
- We can however build a network of linear classifiers.

# Neural Networks

## Neural Networks - Infamous XOR Problem:



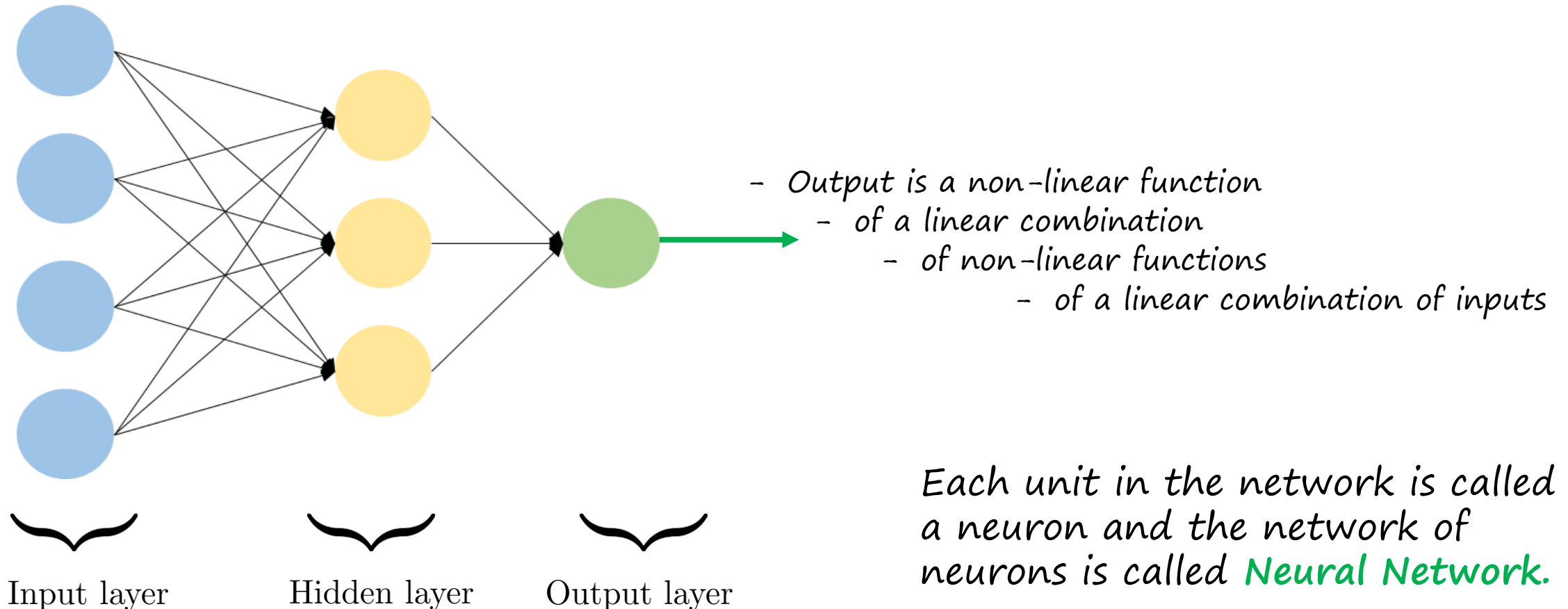
- This is a neural network; a network of perceptrons, aka multi-layer perceptron (MLP).



# Neural Networks

## Neural Networks

- A neural network is a set of neurons organized in layers.

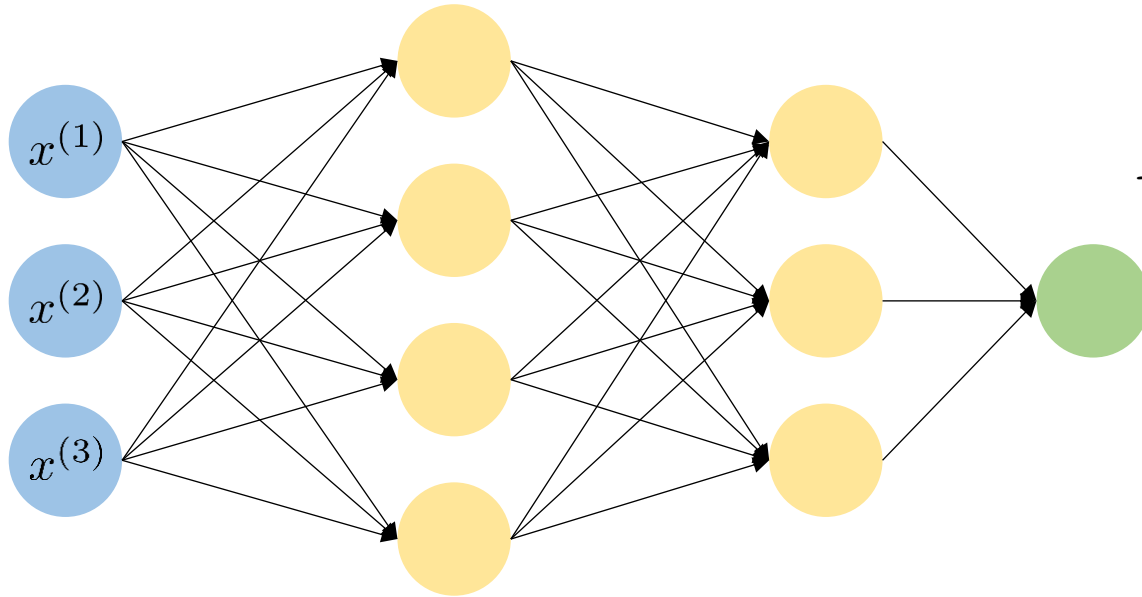


- Given the input and parameters of the neurons, we can determine the output by traversing layers from input to output. This is referred to as **Forward Pass**.

# Neural Networks

## Neural Networks:

### Example: 3-layer network, 2 hidden layers



- Output is a non-linear function
  - of a linear combination
    - of non-linear functions
    - of linear combinations
    - of non-linear functions
    - of linear combinations of inputs

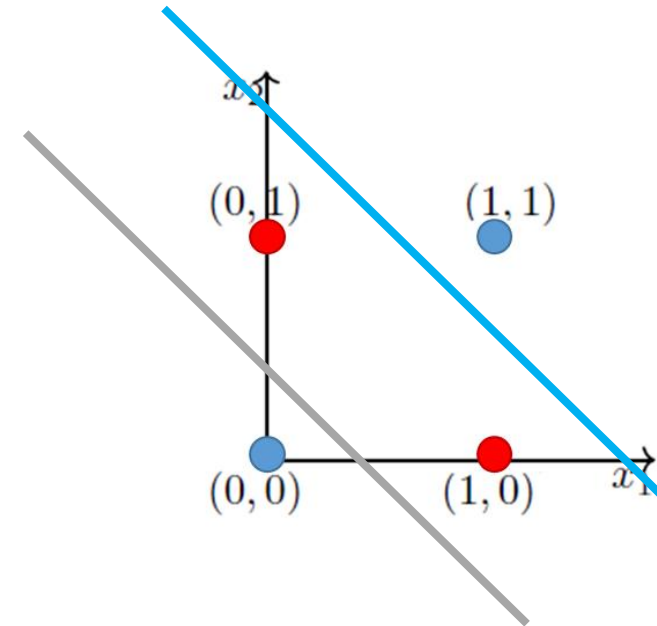
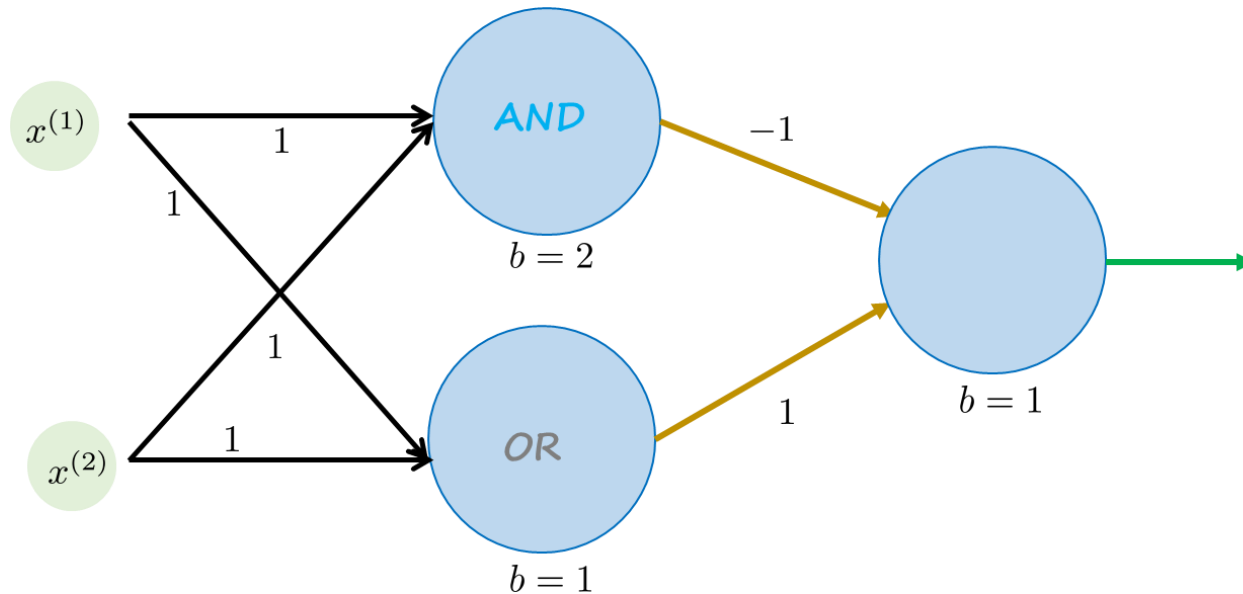
- We do not count the input layer because there are no parameters associated with it.
- Neural networks with neurons are also referred to as MLPs but we will refer to the network as MLP only when it is constructed using perceptrons.

**Feedforward Neural Network:** Output from one layer is an input to the next layer.

# Neural Networks

## What kind of functions can be modeled by a neural network?

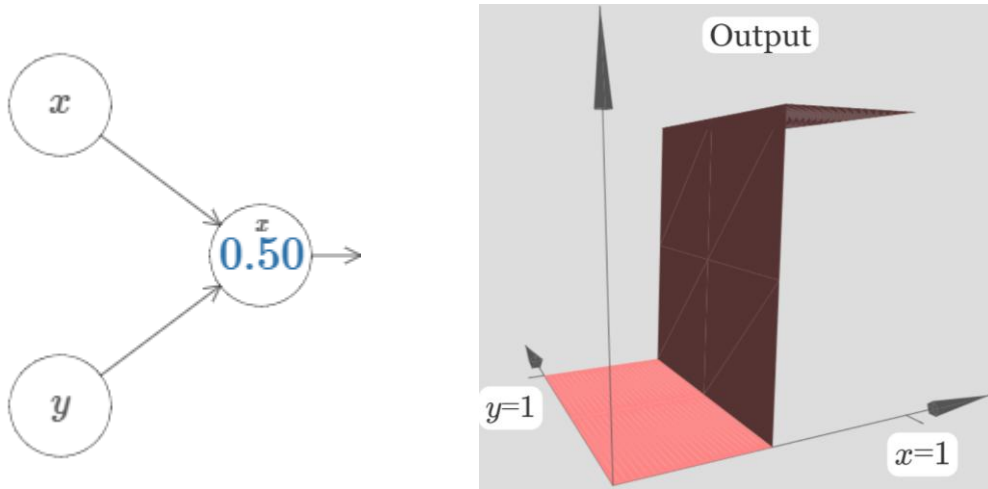
Intuition: XOR example



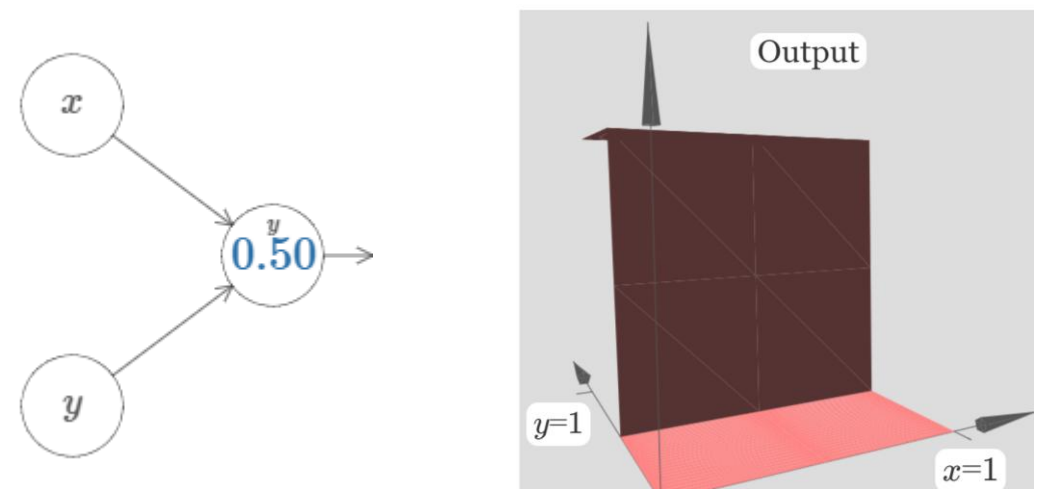
# Neural Networks

## What kind of functions can be modeled by a neural network?

Intuition: Example (Sigmoid neuron)



- bias=0.5 indicated.
- weight for  $x$  is very large.
- weight for  $y$  is zero.

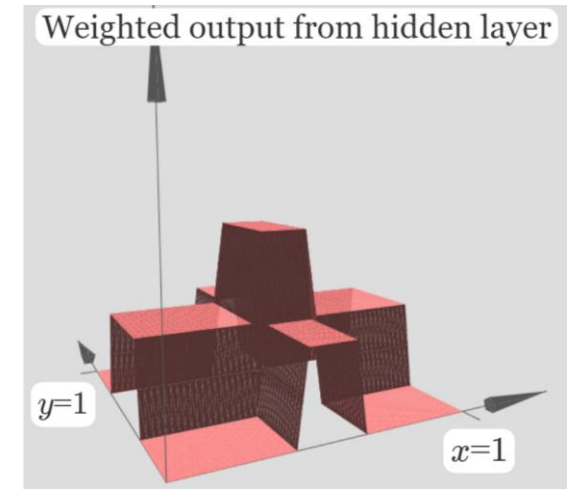
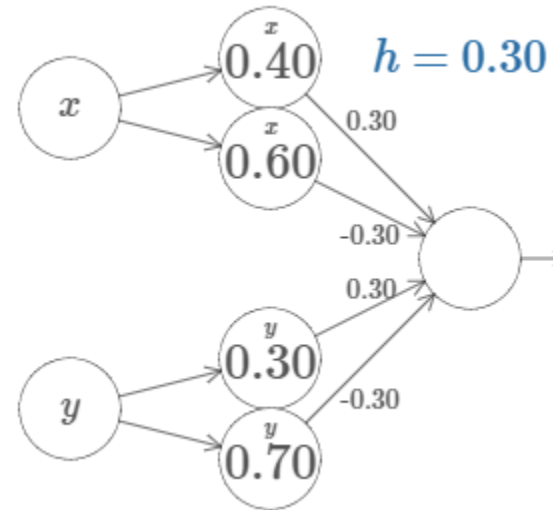
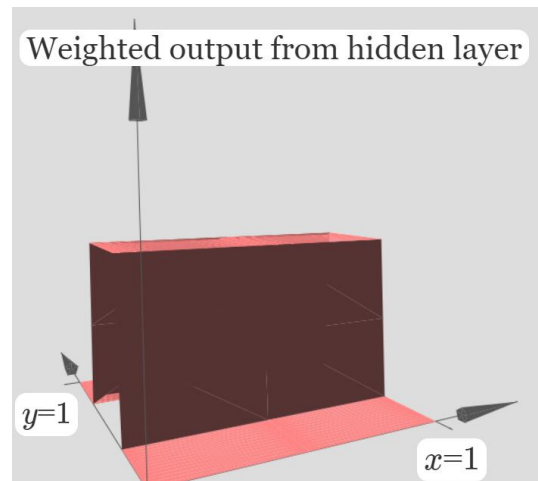
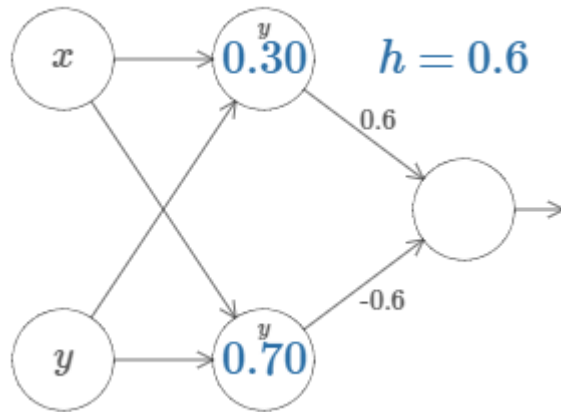
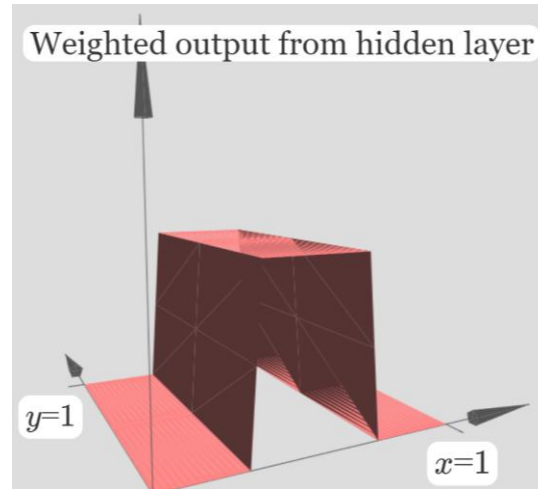
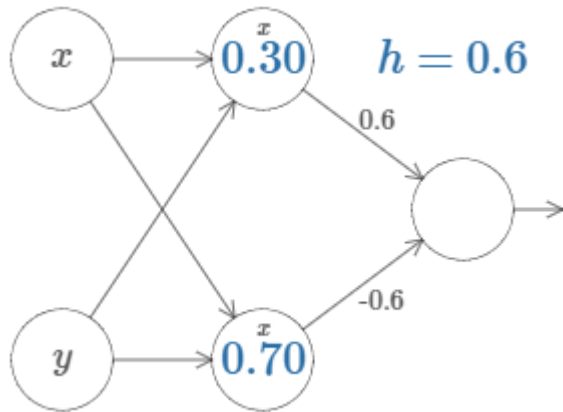


- bias=0.5 indicated.
- weight for  $y$  is very large.
- weight for  $x$  is zero.

# Neural Networks

## What kind of functions can be modeled by a neural network?

### Intuition: Example (Multi layer)



# Neural Networks

## What kind of functions can be modeled by a neural network?

### Universal Approximation Theorem (Hornik 1991):

“A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, *given enough hidden units.*”

- The theorem results demonstrates the capability of neural network, but this does not mean there is a learning algorithm that can find the necessary parameter values.
- Since each neuron represents non-linearity, we can keep on increasing the number of neurons in the hidden layer to model the function. But this will also increase the number of parameters defining the model.
- Instead of adding more neurons in the same layer, we prefer to add more hidden layers because non-linear projections of a non-linear projection can model complex functions relatively easy.

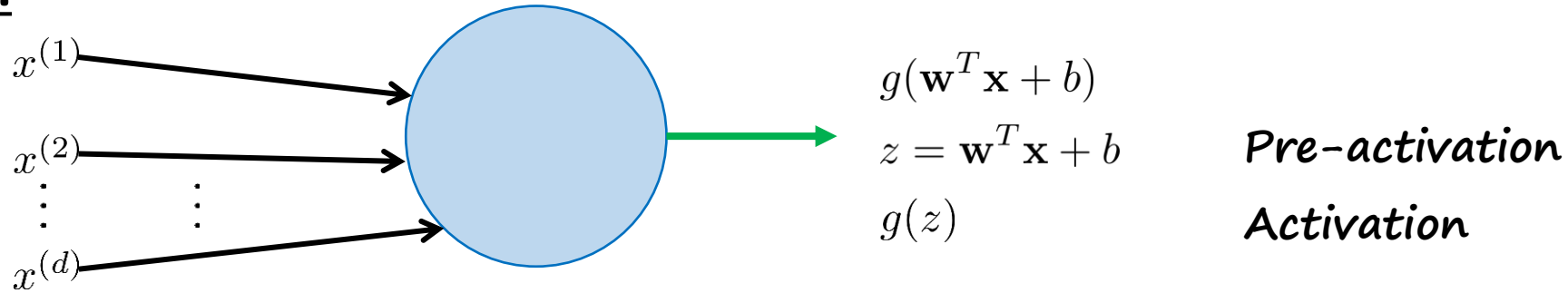
# Outline

- Neural networks connection with perceptron and logistic regression
- *Neural networks notation*
- *Neural networks 'Forward Pass'*
- Activation functions
- Learning neural network parameters
  - Back Propagation.

# Neural Networks

## Neural Networks – Notation:

### Single Neuron:



- If we stack  $n$  training data in a matrix  $\mathbf{X}$  of size  $d \times n$ , that is,  $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]$
- Using  $\mathbf{X}$  and by defining  $\mathbf{1}$  a row vector of ones of length  $n$ , we can define ‘pre-activation’ operation  $\mathbf{w}^T \mathbf{x} + b$  for all inputs compactly, denoted by  $\mathbf{z}$  as

$$\mathbf{z} = \mathbf{w}^T \mathbf{X} + b\mathbf{1}$$

$(1 \times n) \quad (1 \times d)(d \times n) + (1 \times n)$

*Pre-activation  
(Aggregation)*

*Linear transformation*

- Using activation function  $g$ , we obtain

$$\mathbf{a} = g(\mathbf{z})$$

*Activation*

*Non-linear transformation*

- Activation function is operating on each entry of  $\mathbf{z}$ .
- $\mathbf{a}$  - a row vector of length  $n$ ;  $i$ -th entry represents an output for  $i$ -th input.



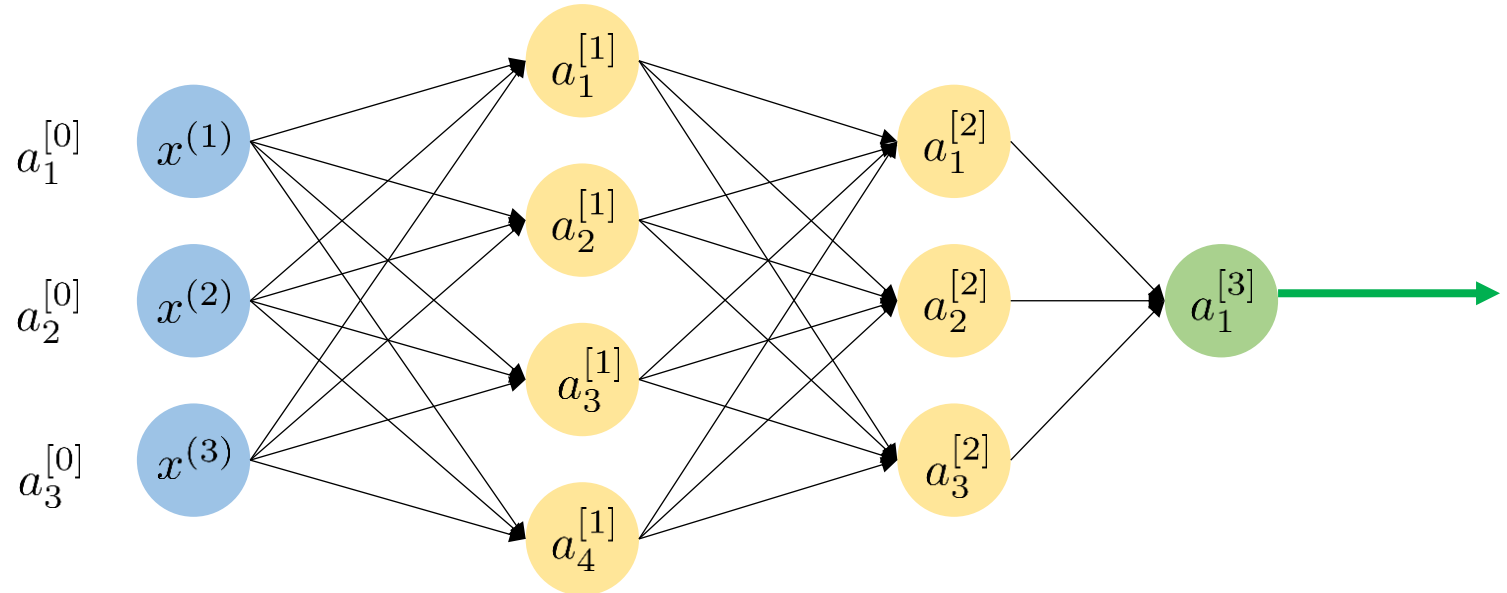
# Neural Networks

## Neural Networks – Notation:

- $L$  - number of layers.
- Number of nodes in the  $\ell$ -th layer,  $m^{[\ell]}$
- $a_i^{[\ell]}$  denotes the output of  $i$ -th node in the  $\ell$ -th layer. •  $\mathbf{a}^{[\ell]}$  - vector of outputs of  $\ell$ -th node.
- $\mathbf{a}^{[0]} = \mathbf{x}$  input layer.
- $\mathbf{a}^{[L]} = y$  output layer.

### Example: 3-layer network, 2 hidden layers

- $L = 3$
- $m^{[1]} = 4, m^{[2]} = 3, m^{[3]} = 1$

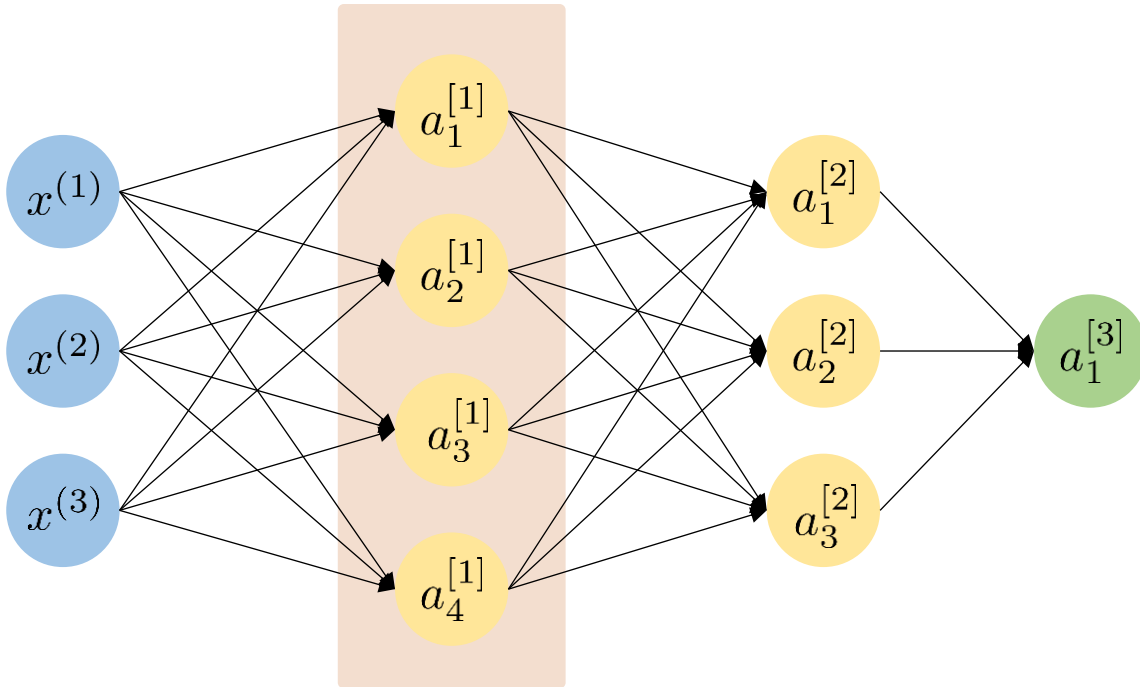


# Neural Networks

## Neural Networks – Notation:

- $\mathbf{w}_i^{[\ell]}$  and  $b_i^{[\ell]}$  denote the weight and bias associated with the  $i$ -th node in the  $\ell$ -th layer respectively.
- $w_{i,j}^{[\ell]}$  denote the weight and bias associated with the  $j$  – th input of the  $i$ -th node in the  $\ell$ -th layer respectively.

### Example: 3-layer network, 2 hidden layers



Layer 1 output

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T} \mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]}$$

$$a_3^{[1]} = g(z_3^{[1]}), \quad z_3^{[1]} = \mathbf{w}_3^{[1]T} \mathbf{x} + b_3^{[1]}$$

$$a_4^{[1]} = g(z_4^{[1]}), \quad z_4^{[1]} = \mathbf{w}_4^{[1]T} \mathbf{x} + b_4^{[1]}$$

# Neural Networks

## Neural Networks – Notation:

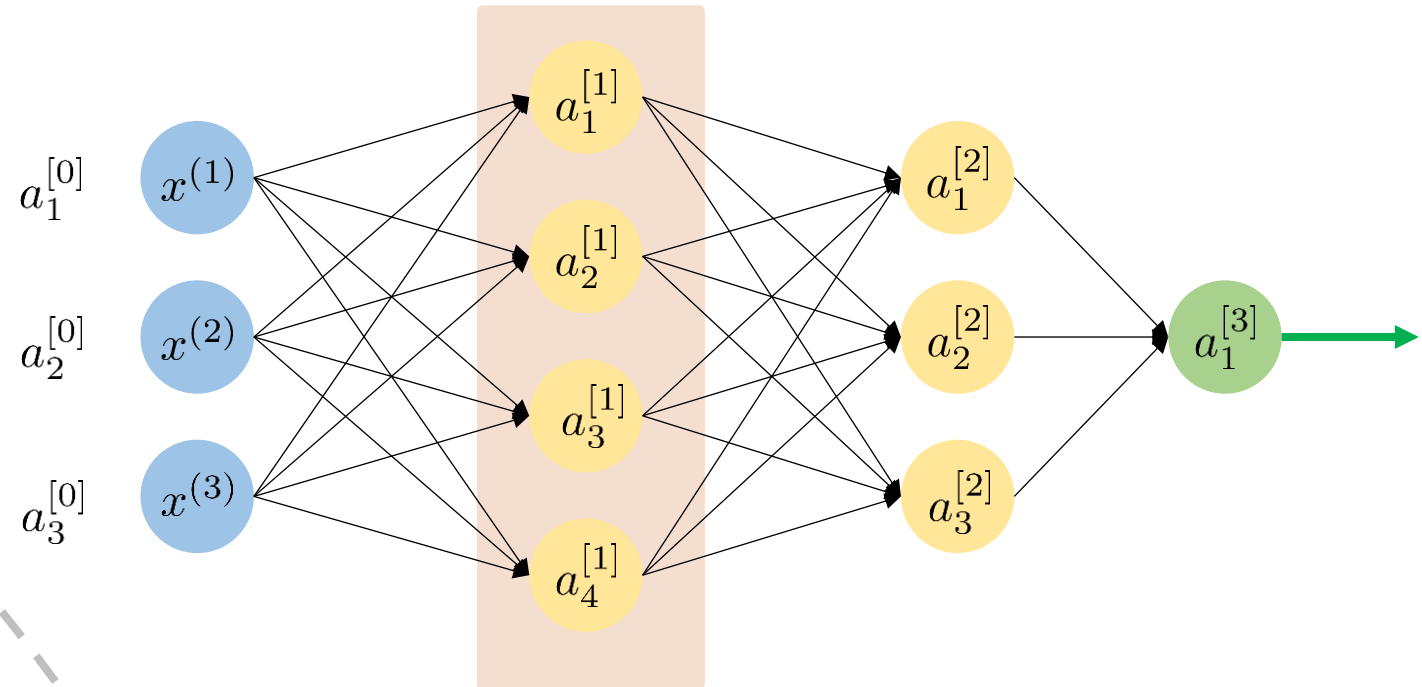
Layer 1 output

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T} \mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]}$$

$$a_3^{[1]} = g(z_3^{[1]}), \quad z_3^{[1]} = \mathbf{w}_3^{[1]T} \mathbf{x} + b_3^{[1]}$$

$$a_4^{[1]} = g(z_4^{[1]}), \quad z_4^{[1]} = \mathbf{w}_4^{[1]T} \mathbf{x} + b_4^{[1]}$$



$$\mathbf{W}^{[1]} = \begin{bmatrix} \mathbf{w}_1^{[1]T} \\ \mathbf{w}_2^{[1]T} \\ \mathbf{w}_3^{[1]T} \\ \mathbf{w}_4^{[1]T} \end{bmatrix}$$

$$\mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \\ b_4^{[1]T} \end{bmatrix}$$

$$\mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]T} \\ z_2^{[1]T} \\ z_3^{[1]T} \\ z_4^{[1]T} \end{bmatrix}$$

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]} \mathbf{x} + \mathbf{b}^{[1]}$$
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]} \mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

- $\mathbf{W}^{[1]}$  and  $\mathbf{b}^{[1]}$  are the parameters of the first layer.

# Neural Networks

## Neural Networks – Forward Pass:

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

- Q. What is the size of  $\mathbf{W}^{[1]}$ ?

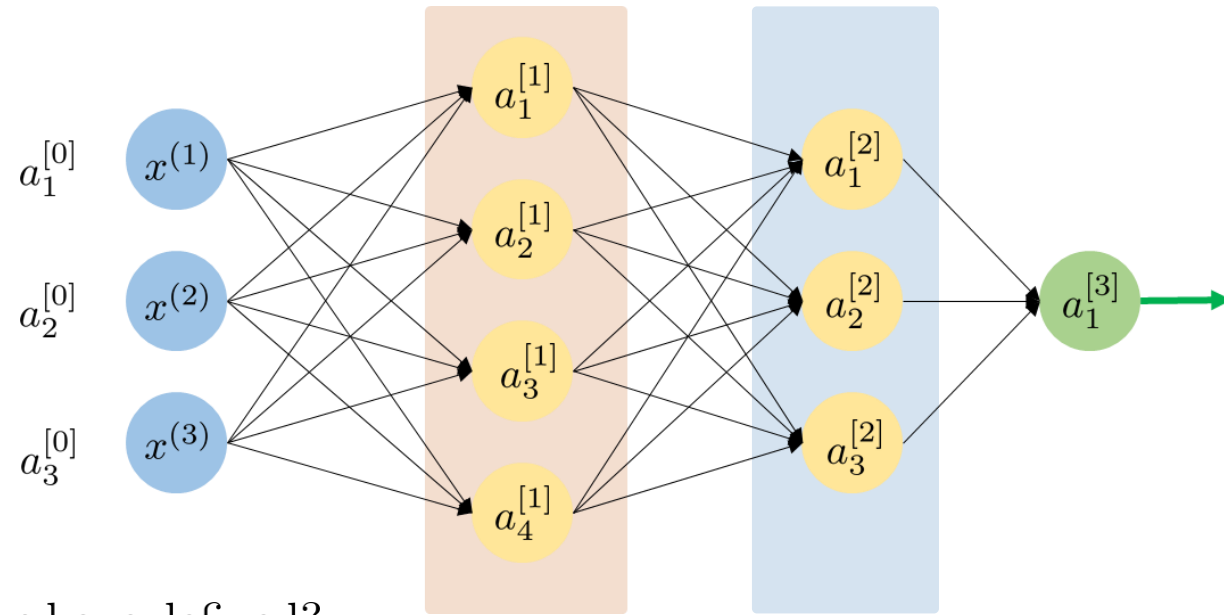
A. No. of nodes  $\times$  No. of inputs.  $4 \times 3$

No. of nodes  $\times$  No. nodes in the previous layer.

- Q. Can we write output of second layer using the notation we have defined?

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$



- Q. What is the size of  $\mathbf{W}^{[2]}$ ?  $3 \times 4$

- Q. What is the size of  $\mathbf{W}^{[3]}$ ?  $1 \times 3$

- $\mathbf{W}^{[\ell]}$  and  $\mathbf{b}^{[\ell]}$  are the parameters of the  $\ell$ -th layer.

- Using these equations, we can determine the output given input and parameters of layers (**Forward Pass**).

# Neural Networks

## Neural Networks – Forward Pass Summary:

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

$$(4 \times 1) = (4 \times 3)(3 \times 1) + (4 \times 1)$$

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$(3 \times 1) = (3 \times 4)(4 \times 1) + (3 \times 1)$$

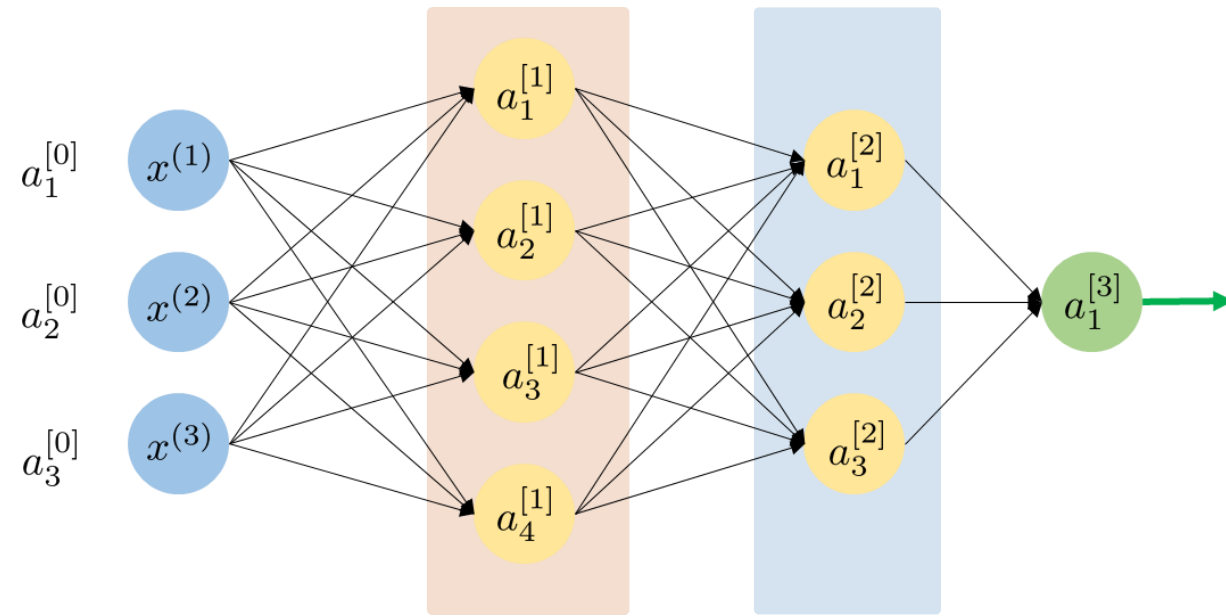
$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$

$$(1 \times 1) = (1 \times 3)(3 \times 1) + (1 \times 1)$$

- In general, we have

$$\mathbf{a}^{[\ell]} = g(\mathbf{z}^{[\ell]}), \quad \mathbf{z}^{[\ell]} = \mathbf{W}^{[\ell]}\mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]}$$

for  $\ell = 1, 2, \dots, L$ , where  $\mathbf{a}^{[0]} = \mathbf{x}$ .



- How many parameters do we have by the way?
- This formulation is for one input  $\mathbf{x}$ .
- How can we extend this formulation  $n$  inputs?

# Neural Networks

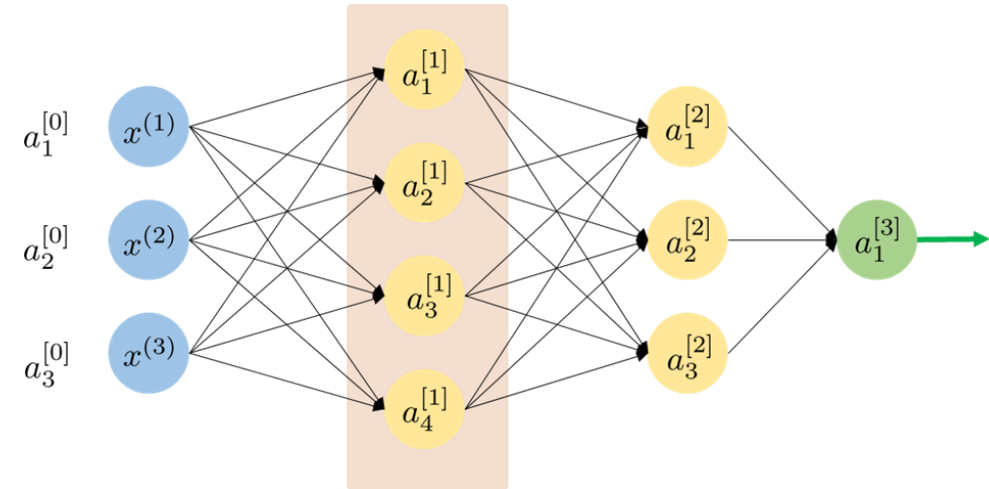
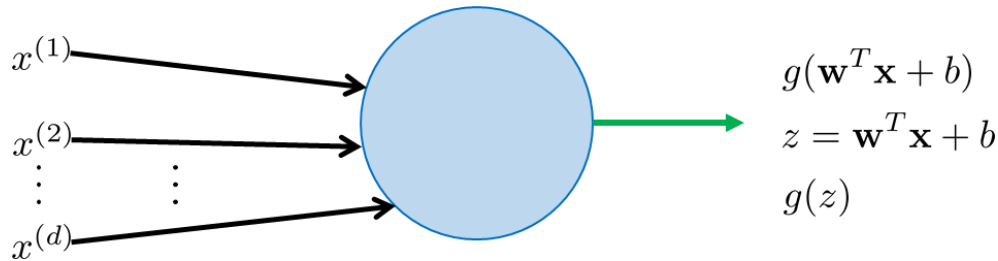
## Neural Networks – Forward Pass – Incorporating all Inputs:

$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

$$(4 \times 1) = (4 \times 3)(3 \times 1) + (4 \times 1)$$

### Recall:



$$\mathbf{A}^{[1]} = g(\mathbf{Z}^{[1]}), \quad \mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}$$

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{A}^{[0]} + \mathbf{b}^{[1]}$$

$$(4 \times n) = (4 \times 3)(3 \times n) + (4 \times n)$$

- For single neuron, we developed the following formulation incorporating all inputs simultaneously.

$$\mathbf{z} = \mathbf{w}^T \mathbf{X} + b\mathbf{1}$$

$$\mathbf{a} = g(\mathbf{z})$$

- $\mathbf{a}$  - a row vector of length  $n$ ;  
 $i$ -th entry represents an output for  $i$ -th input.

# Neural Networks

## Neural Networks – Forward Pass Summary – All Inputs:

$$\mathbf{A}^{[1]} = g(\mathbf{Z}^{[1]}), \quad \mathbf{Z}^{[1]} = \mathbf{W}^{[1]} \mathbf{X} + \mathbf{b}^{[1]}$$

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]} \mathbf{A}^{[0]} + \mathbf{b}^{[1]}$$

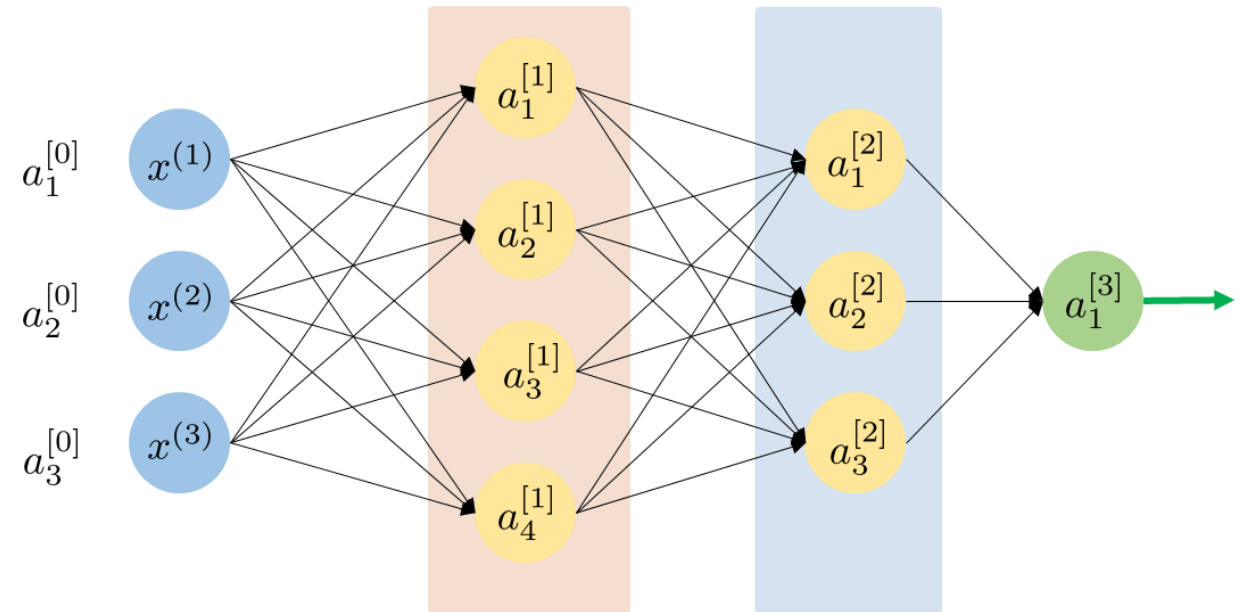
$$(4 \times n) = (4 \times 3)(3 \times n) + (4 \times n)$$

$$\mathbf{A}^{[2]} = g(\mathbf{Z}^{[2]}), \quad \mathbf{Z}^{[2]} = \mathbf{W}^{[2]} \mathbf{A}^{[1]} + \mathbf{b}^{[2]}$$

$$(3 \times n) = (3 \times 4)(4 \times n) + (3 \times n)$$

$$\mathbf{A}^{[3]} = g(\mathbf{Z}^{[3]}), \quad \mathbf{Z}^{[3]} = \mathbf{W}^{[3]} \mathbf{A}^{[2]} + \mathbf{b}^{[3]}$$

$$(1 \times n) = (1 \times 3)(3 \times n) + (1 \times n)$$



- In general, we have

$$\mathbf{A}^{[\ell]} = g(\mathbf{Z}^{[\ell]}), \quad \mathbf{Z}^{[\ell]} = \mathbf{W}^{[\ell]} \mathbf{A}^{[\ell-1]} + \mathbf{b}^{[\ell]} \quad \ell = 1, 2, \dots, L$$

# Outline

- Neural networks connection with perceptron and logistic regression
- Neural networks notation
- Neural networks 'Forward Pass'
- *Activation functions*
- Learning neural network parameters
  - Back Propagation.



# Neural Networks

## Activation Function:

- Single neuron is characterized by weights, bias and activation function  $g$ .

- We require  $g$  to be some non-linear function.

- Why?

- If  $g$  is a linear function, e.g., identity or  $g(z) = \alpha z + \beta$

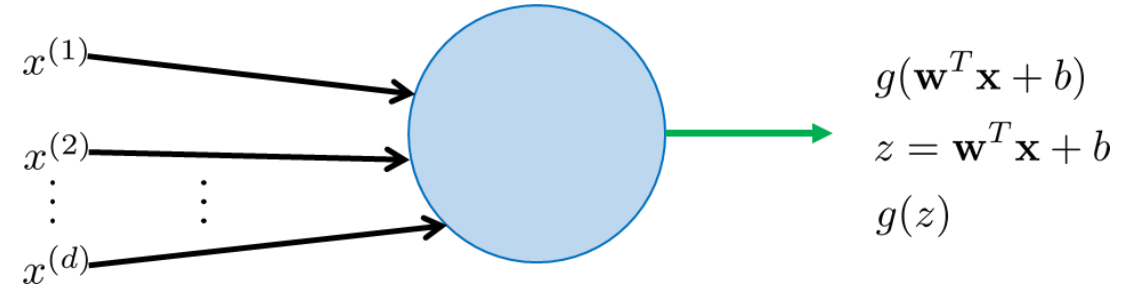
$$g(\mathbf{w}^T \mathbf{x} + b) = \alpha(\mathbf{w}^T \mathbf{x} + b) + \beta \equiv \tilde{\mathbf{w}}^T \mathbf{x} + \tilde{b}$$

- Consequently, different layers of the network can be equivalently represented by a single linear transformation.

- We require active function  $g$  to be differentiable if we want to use gradient descent.

- Some standard activation functions:

- identity (linear), step, rectified linear, leaky rectified linear sigmoid, tanh



# Neural Networks

## Step:

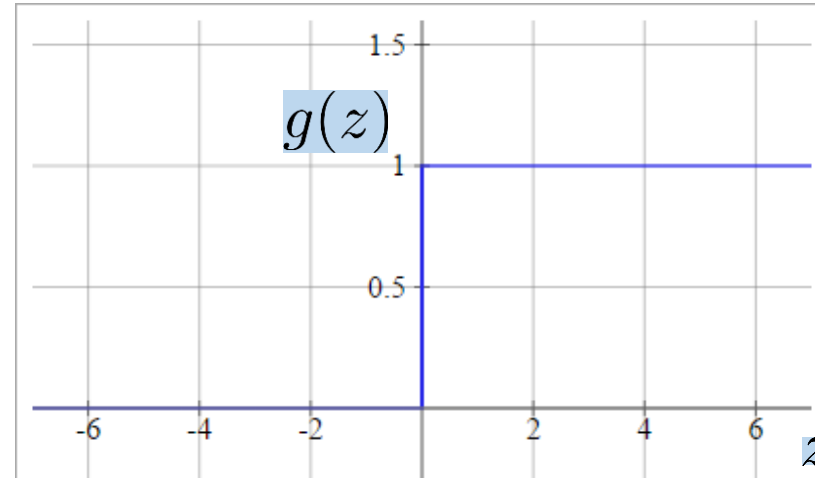
- Step function is defined as

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

- We have used this activation function before.

Perceptron

- Issues: Non-differentiable and only supports binary classification.



## Linear:

$$g(z) = \alpha z + \beta$$

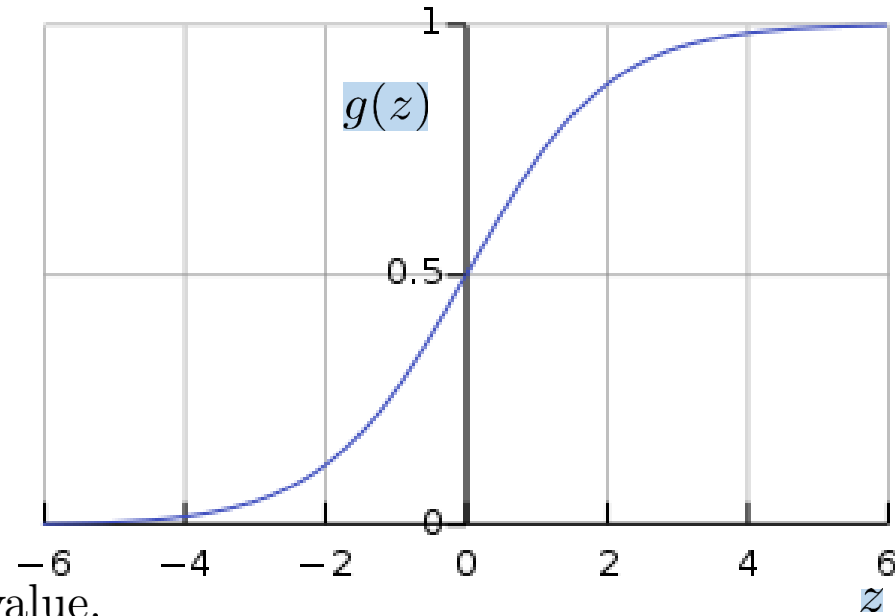
- Constant derivative,  $g'(z) = \alpha$ ; cannot be used for backpropagation.
- Used for simple linear regression model.
- Does not capture non-linearities irrespective of the depth of the network.

# Neural Networks

## Sigmoid:

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (\text{Sigmoid: because of S shaped curve})$$

- Squishes values in  $(-\infty, \infty)$  to  $(0, 1)$ , bounded, strictly increasing.
- Suitable for output neurons of neural networks used for classification.
- We have used this activation function before. Logistic regression.
- It is differentiable.  $g'(z) = g(z)(1 - g(z))$



## Issues:

- Saturation Problem or Vanishing Gradient:

Neuron is saturated considered when it reaches its maximum or minimum value.

$g(z) = 0$  or  $g(z) = 1$ . Consequence:  $g'(z) = g(z)(1 - g(z)) = 0$  (poor learning for deep networks)

- Sigmoid outputs are not zero-centered:

Due to this, the gradient of all the weights for a neuron is either positive or negative. Consequently, the weights move in one direction. This issue is less severe as gradients are added across the batch and mitigate this.

# Neural Networks

## tanh (Hyperbolic tangent):

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

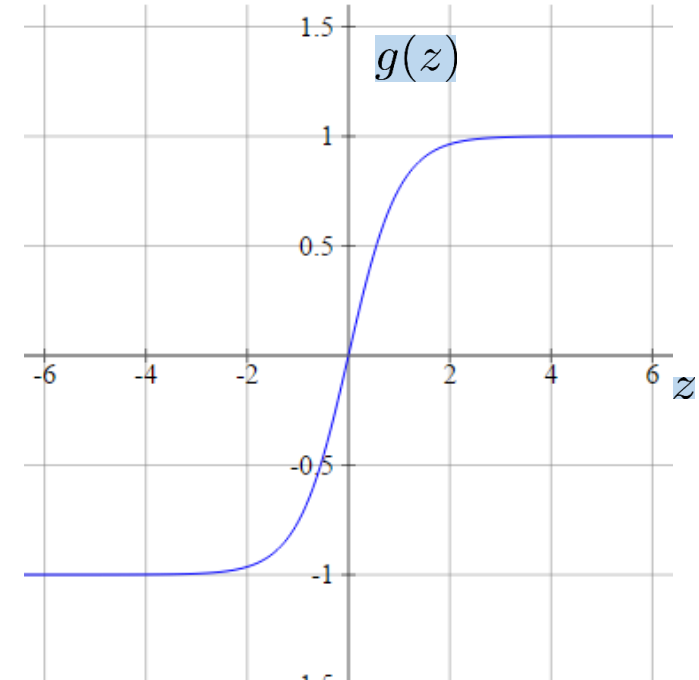
- Scaled version of sigmoid.
- Squishes values in  $(-\infty, \infty)$  to  $(-1, 1)$ , bounded, strictly increasing.
- Unlike Sigmoid, tanh is a zero-centered function and resolve the issue associated with the sigmoid.
- It is differentiable.  $g'(z) = 1 - (g(z))^2$

### Issues:

- Saturation Problem or Vanishing Gradient:

Neuron is saturated considered when it reaches its maximum or minimum value.

$g(z) = -1$  or  $g(z) = 1$ . Consequence:  $g'(z) = 1 - (g(z))^2 = 0$  (poor learning for deep networks)



# Neural Networks

## Rectifier – Rectified Linear Unit (ReLU):

$$g(z) = \max(0, z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases}$$

- Super simple.

- It is differentiable.  $g'(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$

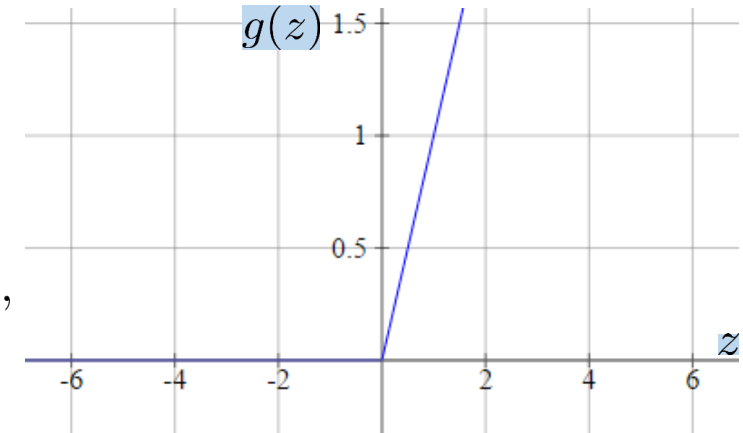
- It is the most used activation function.

### Why?

- Unlike Sigmoid and tanh that required the computation of exponents, easier to compute.
- It only activates for which the output is non-negative. It deactivates the neurons if the pre-activation output is less than 0. This makes ReLU computationally efficient relative to sigmoid and tanh.

### Issues:

- ReLU is not zero centered.  
As indicated earlier, this is not a major issue and training the network longer can resolve this.
- ReLU does not suffer from saturation problem. Vanishing gradient however occurs for negative values; dead neuron.



*Different variants of ReLU have been proposed to overcome these issues.*

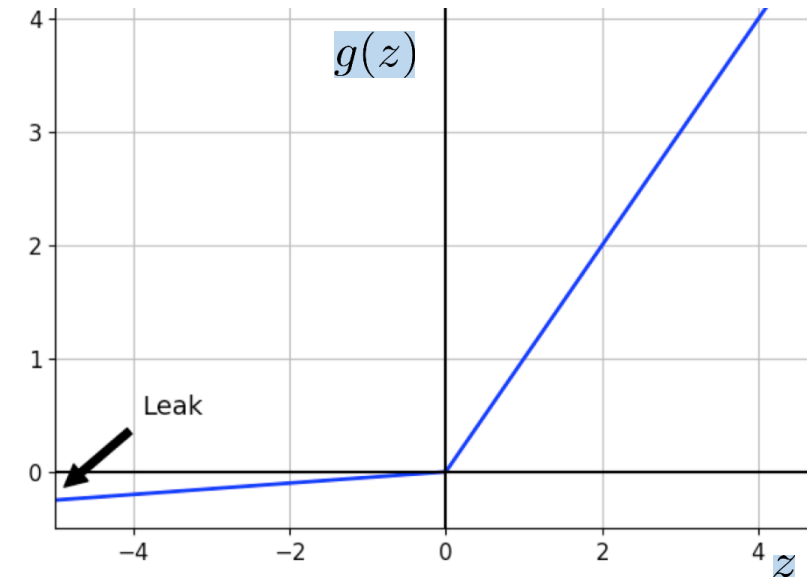
# Neural Networks

## Leaky Rectified Linear Unit (Leaky ReLu):

$$g(z) = \begin{cases} z & z \geq 0 \\ \alpha z & z < 0 \end{cases}$$

- This resolves the issues associated with ReLu to an extent.
  - Due to non-zero output for negative values, it keeps neurons alive.
  - It is not zero centered but close to it.
- If  $\alpha$  is any value less than 0.01, randomized ReLu.
- To be precise,  $\alpha = 0.01$  for leakly ReLu.

- It is differentiable.  $g'(z) = \begin{cases} 1 & z \geq 0 \\ \alpha & z < 0 \end{cases}$



*We use rectified linear, leaky rectified linear sigmoid and tanh for hidden layer.*

# Neural Networks

## Activation Function for output layer:

- Identity function if the problem is regression.
- Sigmoid function if output needs to be between 0 and 1.
  - e.g., probability value.
  - classification problem.
- Softmax if output needs to be probability distribution.
  - multi-class classification problem.

# Neural Networks

## Choice of the Activation Function:

- For classification tasks;
  - we prefer to use sigmoid, tanh functions and their combinations.
- Due to the saturation problem, sigmoids and tanh functions are sometimes avoided.
- As indicated earlier, ReLU function is mostly used (computationally fast).
  - ReLU variants are used to resolve a dead neuron issue (e.g., Leaky ReLU).
- It must be noted that ReLU function is only used in the hidden layers.
- Start with ReLU or leaky/randomized ReLU and if the results are not satisfactory, you may try other activation functions.



# Outline

- Neural networks connection with perceptron and logistic regression
- Neural networks notation
- Neural networks 'Forward Pass'
- Activation functions
- Learning neural network parameters
  - Back Propagation

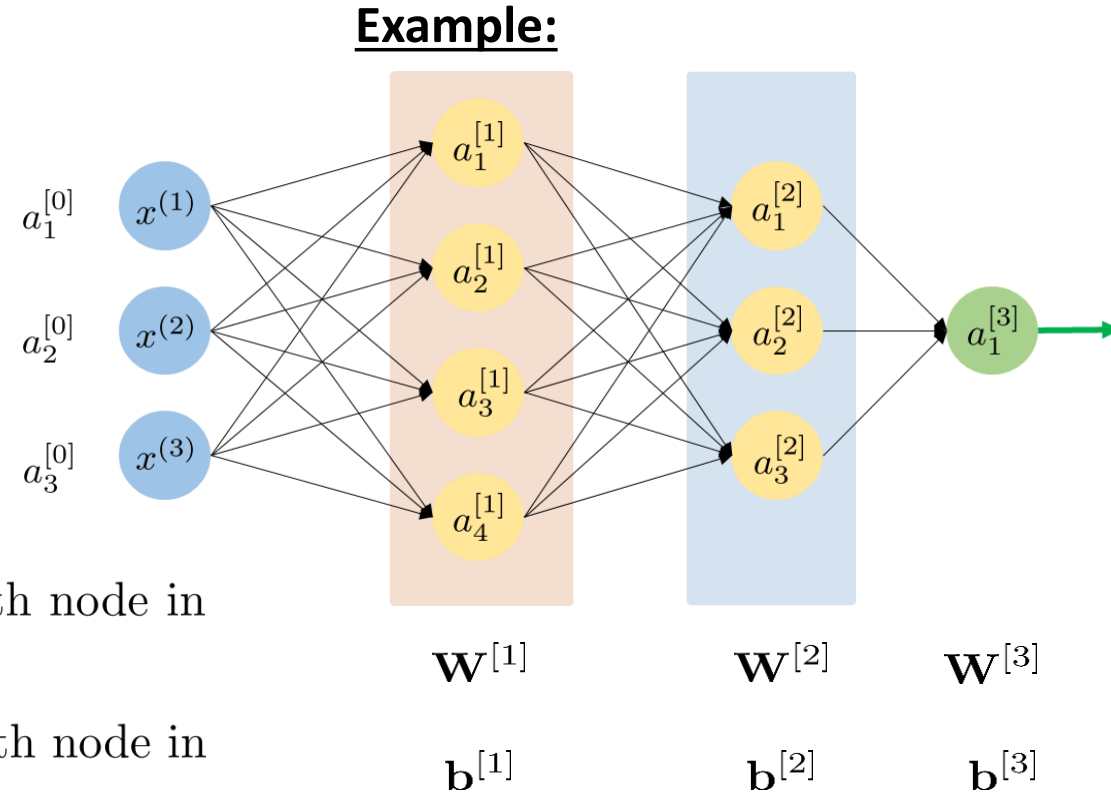
# Neural Networks

## Learning Weights:

- Given the training data, we want to learn the weights (weight matrices+bias vectors) for hidden layers and output layer.

## Notation revisit:

- $L$  - number of layers.
- Number of nodes in the  $\ell$ -th layer,  $m^{[\ell]}$
- $a_i^{[\ell]}$  denotes the output of  $i$ -th node in the  $\ell$ -th layer.
- $\mathbf{a}^{[\ell]}$  output of  $\ell$ -th layer,  $\mathbf{a}^{[0]} = \mathbf{x}$ .
- $\mathbf{a}^{[L]} = y$  output layer.
- $\mathbf{w}_i^{[\ell]}$  and  $b_i^{[\ell]}$  denote the weight and bias associated with the  $i$ -th node in the  $\ell$ -th layer respectively.
- $w_{i,j}^{[\ell]}$  denotes the weight associated with the  $j$ -th input of the  $i$ -th node in the  $\ell$ -th layer.



*Parameters we need to learn!*

# Neural Networks

## Learning Weights:

- We assume we have training data  $D$  given by

$$D = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- Consider a network with  $d$  nodes (features) at the input layer, 1 output node and any number of hidden layers.
- Define the loss function (for regression problem):

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (\tilde{y}_i - y_i)^2$$

where  $\tilde{y}_i$  denotes the output of the neural network for  $i$ -th input.

- We can use gradient descent to learn the weight matrices and bias vectors.
- Given our prior knowledge, output  $y$  is a composite function of input  $\mathbf{x}$ . Therefore, it is continuous and differentiable and we can use chain rule to compute the gradient.

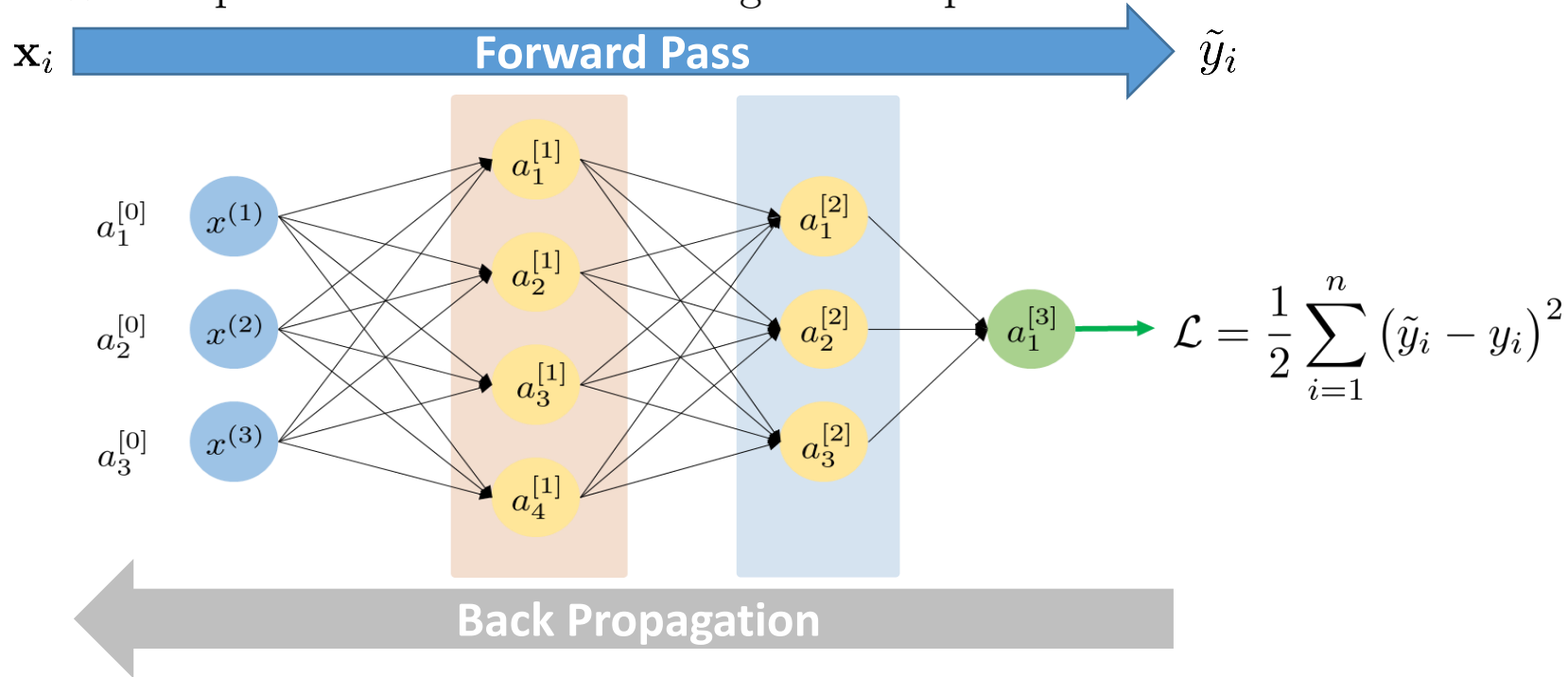
*We use log loss here if we have a classification problem and output represents probability.*

*We use a method called 'Back Propagation' to implement the chain rule for the computation the gradient.*

# Neural Networks

## Back Propagation – Key Idea:

- We compute the loss function using forward pass.



*The weights are the only parameters that can be modified to make the loss function as low as possible.*

- Gradient descent:  $w_{i,j}^{[\ell]} = w_{i,j}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial w_{i,j}^{[\ell]}}$

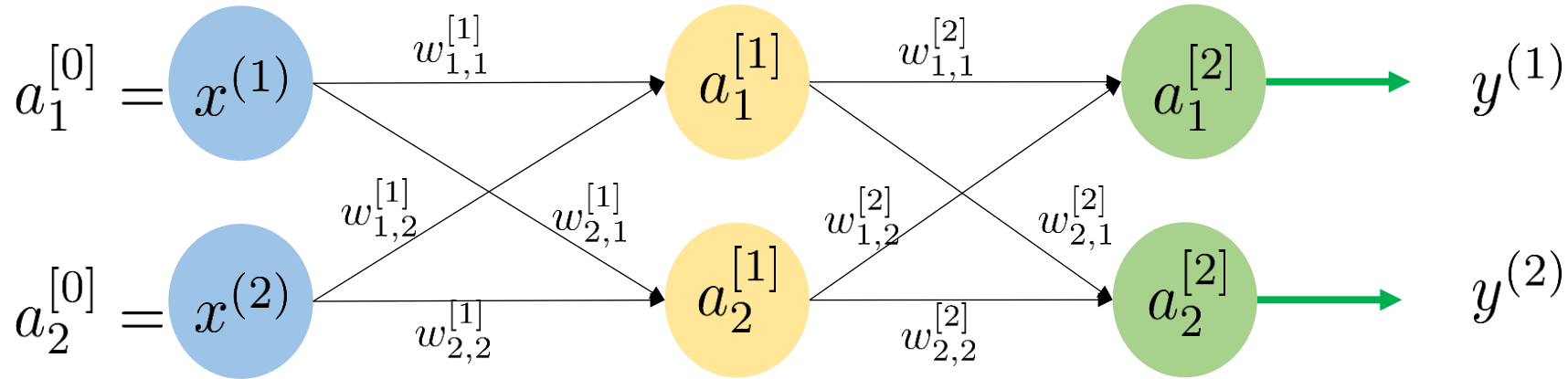
*Learning problem reduces to the question of calculating gradient (partial derivatives) of loss function.*

- We compute the derivative by propagating the total loss at the output node back into the neural network to determine the contribution of every node in the loss. (**Back Propagation**)

# Neural Networks

## Back Propagation – Example:

- 2 layer with 2 neurons in the hidden layer, 2 inputs, 2 outputs network.
- Assuming sigmoid as activation function, that is,  $g(z) = \sigma(z)$ .



- Given training data

$$x^{(1)} = 0.05, \quad x^{(2)} = 0.1, \quad y^{(1)} = 0.01, \quad y^{(2)} = 0.99$$

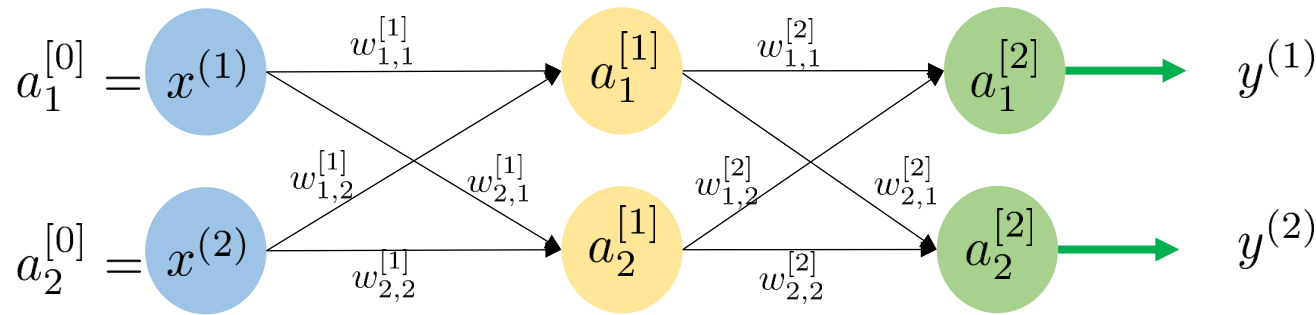
- Initial values of weights and biases:

$$w_{1,1}^{[1]} = 0.15, \quad w_{1,2}^{[1]} = 0.2, \quad w_{2,1}^{[1]} = 0.25, \quad w_{2,2}^{[1]} = 0.3, \quad b_1^{[1]} = 0.35, \quad b_2^{[1]} = 0.35.$$

$$w_{1,1}^{[2]} = 0.4, \quad w_{1,2}^{[2]} = 0.45, \quad w_{2,1}^{[2]} = 0.5, \quad w_{2,2}^{[2]} = 0.55, \quad b_1^{[2]} = 0.6, \quad b_2^{[2]} = 0.6.$$

# Neural Networks

## Back Propagation – Example:



- Loss function

(noting output is a vector):

$$\mathcal{L} = \frac{1}{2} \|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

$$\mathcal{L} = \frac{1}{2} \|(0.01, 0.99) - (0.7514, 0.7729)\|^2 = 0.2984$$

### Forward Pass

$$a_1^{[1]} = g(z_1^{[1]}), \quad z_1^{[1]} = \mathbf{w}_1^{[1]T} \mathbf{x} + b_1^{[1]}$$

$$a_2^{[1]} = g(z_2^{[1]}), \quad z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]}$$

$$a_1^{[2]} = g(z_1^{[2]}), \quad z_1^{[2]} = \mathbf{w}_1^{[2]T} \mathbf{x} + b_1^{[2]}$$

$$a_2^{[2]} = g(z_2^{[2]}), \quad z_2^{[2]} = \mathbf{w}_2^{[2]T} \mathbf{x} + b_2^{[2]}$$

$$z_1^{[1]} = w_{1,1}^{[1]}x^{(1)} + w_{1,2}^{[1]}x^{(2)} + b_1^{[1]} = 0.3775, \quad a_1^{[1]} = g(0.3775) = 0.5933$$

$$z_2^{[1]} = \mathbf{w}_2^{[1]T} \mathbf{x} + b_2^{[1]} = 0.3925, \quad a_2^{[1]} = g(0.3925) = 0.5969$$

$$z_1^{[2]} = \mathbf{w}_1^{[2]T} \mathbf{x} + b_1^{[2]} = 1.106, \quad a_1^{[2]} = g(1.106) = 0.7514 = \tilde{y}^{(1)}$$

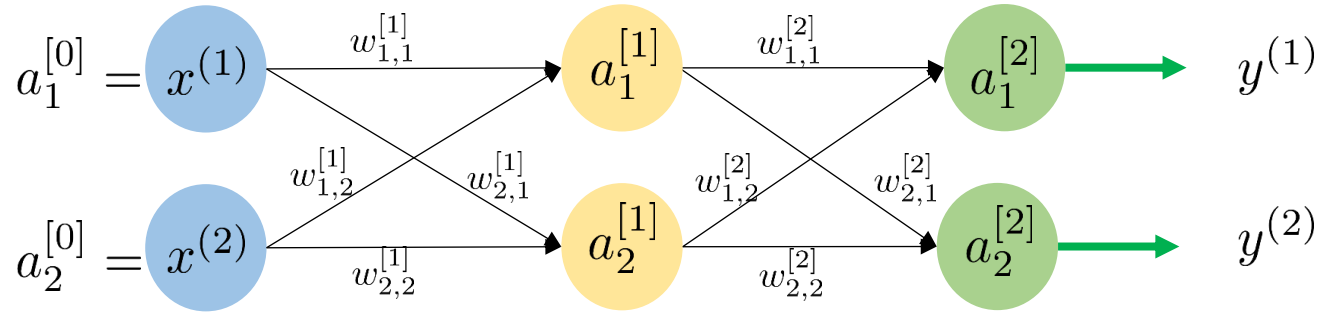
$$z_2^{[2]} = \mathbf{w}_2^{[2]T} \mathbf{x} + b_2^{[2]} = 1.225, \quad a_2^{[2]} = g(1.225) = 0.7729 = \tilde{y}^{(2)}$$

Nothing *fancy* so far, we have computed the output and loss by traversing neural network.

Let's compute the contribution of loss by each node; back propagate the loss.

# Neural Networks

## Back Propagation – Example:



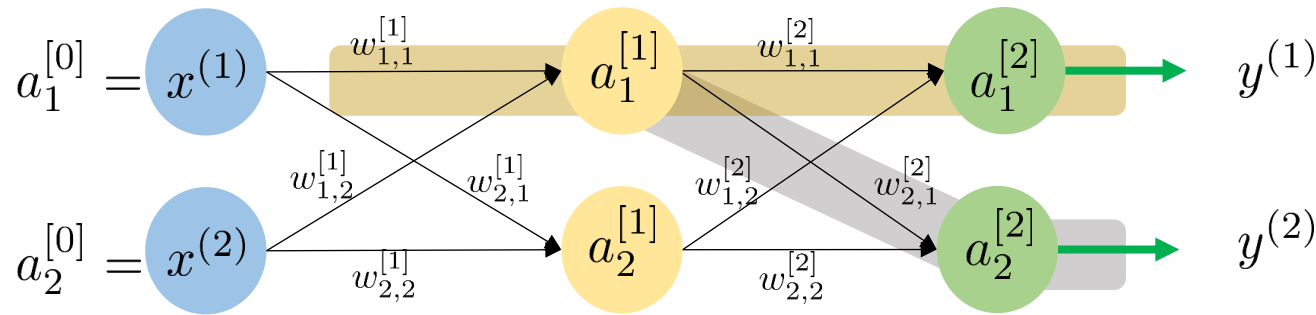
- Consider a case when we want to compute  $\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}}$
- Traverse the path from the loss function back to the weight  $w_{1,1}^{[2]}$ :

$$\left. \begin{aligned} \mathcal{L} &= \frac{1}{2} \|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2 \\ \tilde{y}^{(1)} &= \sigma(z_1^{[2]}) \\ z_1^{[2]} &= w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]} \end{aligned} \right\} \frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} \left\{ \begin{aligned} \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} &= \tilde{y}^{(1)} - y^{(1)} = 0.7414 \\ \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} &= \sigma(\partial z_1^{[2]}) \left(1 - \sigma(\partial z_1^{[2]})\right) = 0.1868 \\ \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} &= a_1^{[a]} = 0.5933 \end{aligned} \right.$$

$$= 0.0821$$

# Neural Networks

## Back Propagation – Example:



$$\mathcal{L} = \frac{1}{2} \|(\tilde{y}^{(1)} - y^{(1)})^2 - (\tilde{y}^{(2)} - y^{(2)})^2\|^2$$

- Consider a case when we want to compute  $\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}}$
- Traverse the path from the loss function back to the weight  $w_{1,1}^{[1]}$ . There are two paths from the output to the weight  $w_{1,1}^{[1]}$ . In other words,  $w_{1,1}^{[1]}$  is contributing to both the outputs.

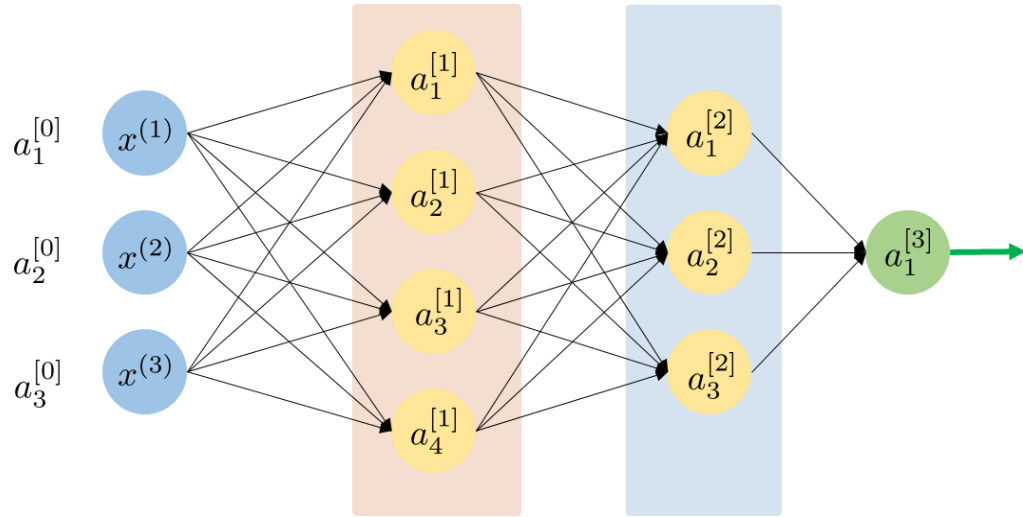
$$\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}} + \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(2)}} \frac{\partial \tilde{y}^{(2)}}{\partial z_2^{[2]}} \frac{\partial z_2^{[2]}}{\partial a_1^{[1]}} \frac{\partial a_1^{[1]}}{\partial z_1^{[1]}} \frac{\partial z_1^{[1]}}{\partial w_{1,1}^{[1]}}$$

- Looking tedious but the concept is very straightforward. I encourage you to write one partial derivative using the same approach to strengthen the concept.



# Neural Networks

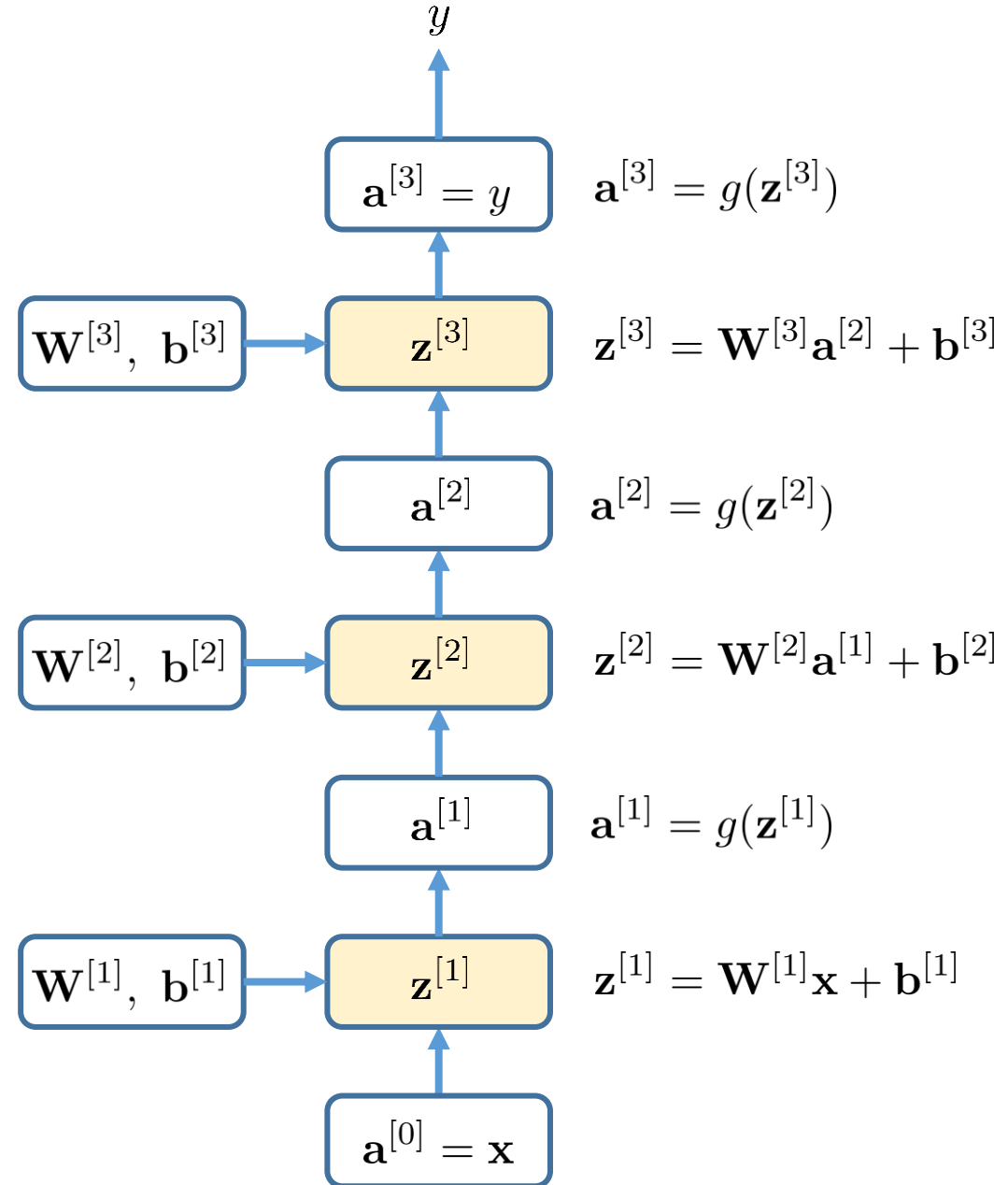
## Back Propagation – Vectorization:



$$\mathbf{a}^{[1]} = g(\mathbf{z}^{[1]}), \quad \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[2]} = g(\mathbf{z}^{[2]}), \quad \mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\mathbf{a}^{[3]} = g(\mathbf{z}^{[3]}), \quad \mathbf{z}^{[3]} = \mathbf{W}^{[3]}\mathbf{a}^{[2]} + \mathbf{b}^{[3]}$$



# Neural Networks

## Back Propagation – Vectorization:

- We compute loss function  $\mathcal{L}$  using forward pass.
- We update  $\mathbf{W}^{[\ell]}$  and  $\mathbf{b}^{[\ell]}$  using gradient descent as:

$$\mathbf{W}^{[\ell]} = \mathbf{W}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} \quad \mathbf{b}^{[\ell]} = \mathbf{b}^{[\ell]} - \alpha \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}}$$

## Partial Derivatives:

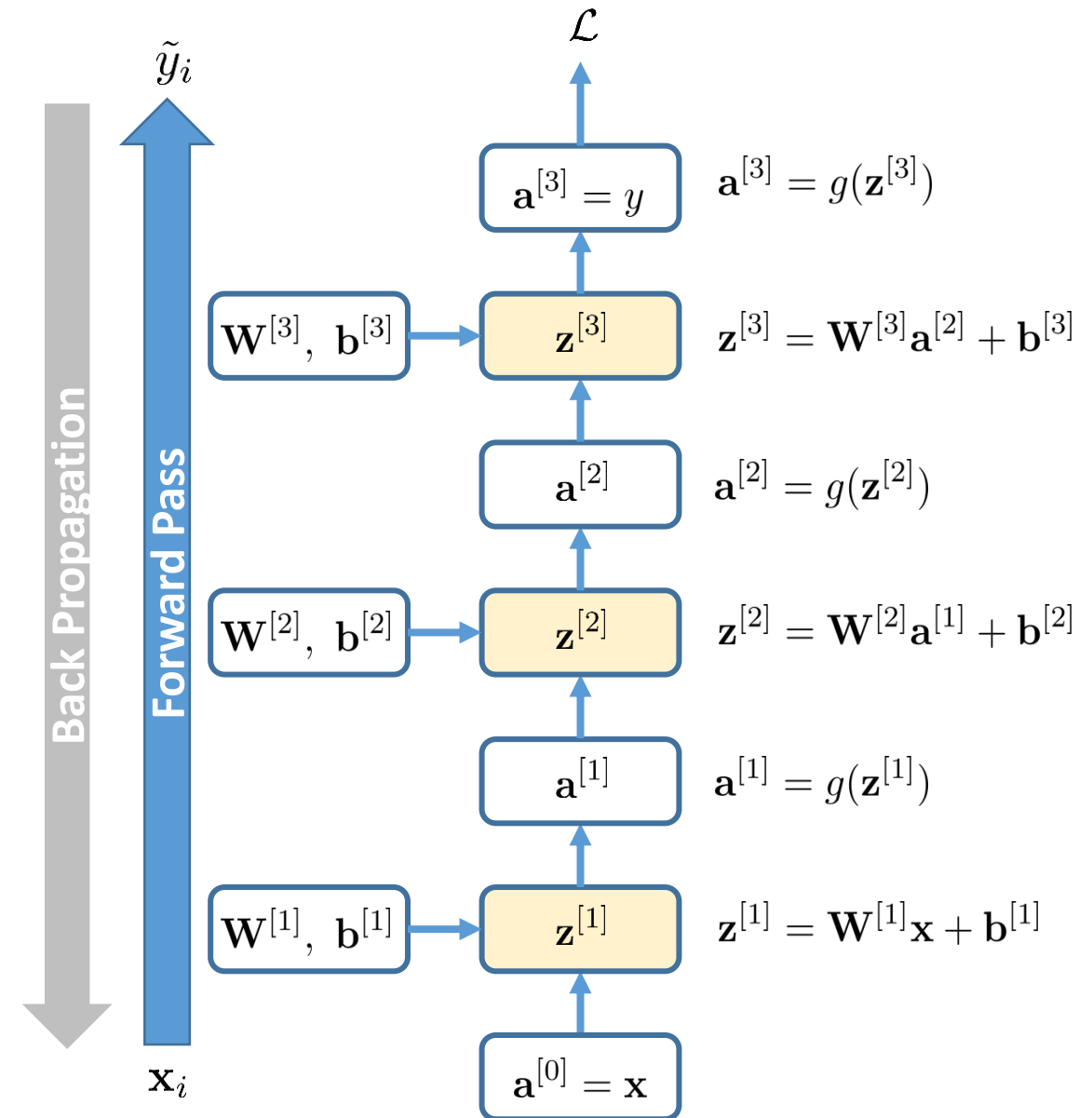
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$



# Neural Networks

## Back Propagation – Vectorization:

### Partial Derivatives:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{W}^{[3]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{b}^{[3]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}}$$

- We need to develop capability to compute  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[l]}}$ ,  $\frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{W}^{[l]}}$  and  $\frac{\partial \mathbf{z}^{[l]}}{\partial \mathbf{b}^{[l]}}$ .

# Neural Networks

## Back Propagation – Vectorization:

- We first compute  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$  for  $\ell = L$  (last layer), that is,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} \frac{\partial \mathbf{a}^{[L]}}{\partial \mathbf{z}^{[L]}}$$

In literature, we often denote  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$  with  $\delta^\ell$ .

- Derivative of loss function with respect to activation function output of the last layer:

- It depends on the definition of the loss function.

- Regression:

Squared error:  $\mathcal{L} = \frac{1}{2} (\mathbf{a}^{[L]} - y)^2 = \frac{1}{2} \|\mathbf{a}^{[L]} - y\|^2$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} = \mathbf{a}^{[L]} - y$$

- Classification:

Log-loss (cross entropy):  $\mathcal{L} = -y \log \mathbf{a}^{[L]} + (1 - y) \log (1 - \mathbf{a}^{[L]})$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} = - \left( \frac{y}{\mathbf{a}^{[L]}} - \frac{1 - y}{1 - \mathbf{a}^{[L]}} \right)$$

# Neural Networks

## Back Propagation – Vectorization:

- Derivative of activation function with respect to pre-activation output:

$$\mathbf{a}^{[\ell]} = g(\mathbf{z}^{[\ell]})$$

$$\frac{\partial \mathbf{a}^{[\ell]}}{\partial \mathbf{z}^{[\ell]}} = g'(\mathbf{z}^{[\ell]}) = \sigma(\mathbf{z}^{[\ell]}) \odot \left(1 - \sigma(\mathbf{z}^{[\ell]})\right)$$

if sigmoid is used as an activation function.

- $\odot$  represents the element wise multiplication of matrices or vectors.
- Combining these two, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} \odot \frac{\partial \mathbf{a}^{[L]}}{\partial \mathbf{z}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} \odot g'(\mathbf{z}^{[L]})$$

- We have computed  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$  for  $\ell = L$  (last layer).

# Neural Networks

## Back Propagation – Vectorization:

- Observing this,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \frac{\partial \mathbf{a}^{[3]}}{\partial \mathbf{z}^{[3]}} \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{a}^{[1]}} \frac{\partial \mathbf{a}^{[1]}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

we can write

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell+1]}} \frac{\partial \mathbf{z}^{[\ell+1]}}{\partial \mathbf{a}^{[\ell]}} \frac{\partial \mathbf{a}^{[\ell]}}{\partial \mathbf{z}^{[\ell]}}$$

- Derivative of pre-activation output of the layer with respect to the input of the current layer or output of the previous layer:

$$\mathbf{z}^{[\ell+1]} = \mathbf{W}^{[\ell+1]} \mathbf{a}^{[\ell]} + \mathbf{b}^{[\ell+1]} \quad \frac{\partial \mathbf{z}^{[\ell+1]}}{\partial \mathbf{a}^{[\ell]}} = \mathbf{W}^{[\ell+1]}$$

- Using these results, we can write

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} = \left( \mathbf{W}^{[\ell+1]T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell+1]}} \right) \odot g'(\mathbf{z}^{[\ell]})$$

changed order to keep the consistency of the multiplication of matrices.

# Neural Networks

## Back Propagation – Vectorization:

- Derivative of pre-activation output of the layer with respect to weight or bias:
- We know that

$$\mathbf{z}^{[\ell]} = \mathbf{W}^{[\ell]} \mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]} \quad \frac{\partial \mathbf{z}^{[\ell]}}{\partial \mathbf{W}^{[\ell]}} = \mathbf{a}^{[\ell-1]} \quad \frac{\partial \mathbf{z}^{[\ell]}}{\partial \mathbf{b}^{[\ell]}} = \mathbf{1}$$

- Finally, we obtain

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \frac{\partial \mathbf{z}^{[\ell]}}{\partial \mathbf{W}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \mathbf{a}^{[\ell-1]T}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \frac{\partial \mathbf{z}^{[\ell]}}{\partial \mathbf{b}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$$

# Neural Networks

## Back Propagation – Vectorization:

### Key Equations:

- Gradients with respect to weight and bias:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \mathbf{a}^{[\ell-1]T} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$$

- We compute  $\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}}$  as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} = \left( \mathbf{W}^{[\ell+1]T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell+1]}} \right) \odot g'(\mathbf{z}^{[\ell]})$$

- Base case:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[L]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[L]}} \odot g'(\mathbf{z}^{[L]})$$

- For sigmoid activation function:

$$g'(\mathbf{z}^{[\ell]}) = \sigma(\mathbf{z}^{[\ell]}) \odot \left( 1 - \sigma(\mathbf{z}^{[\ell]}) \right)$$

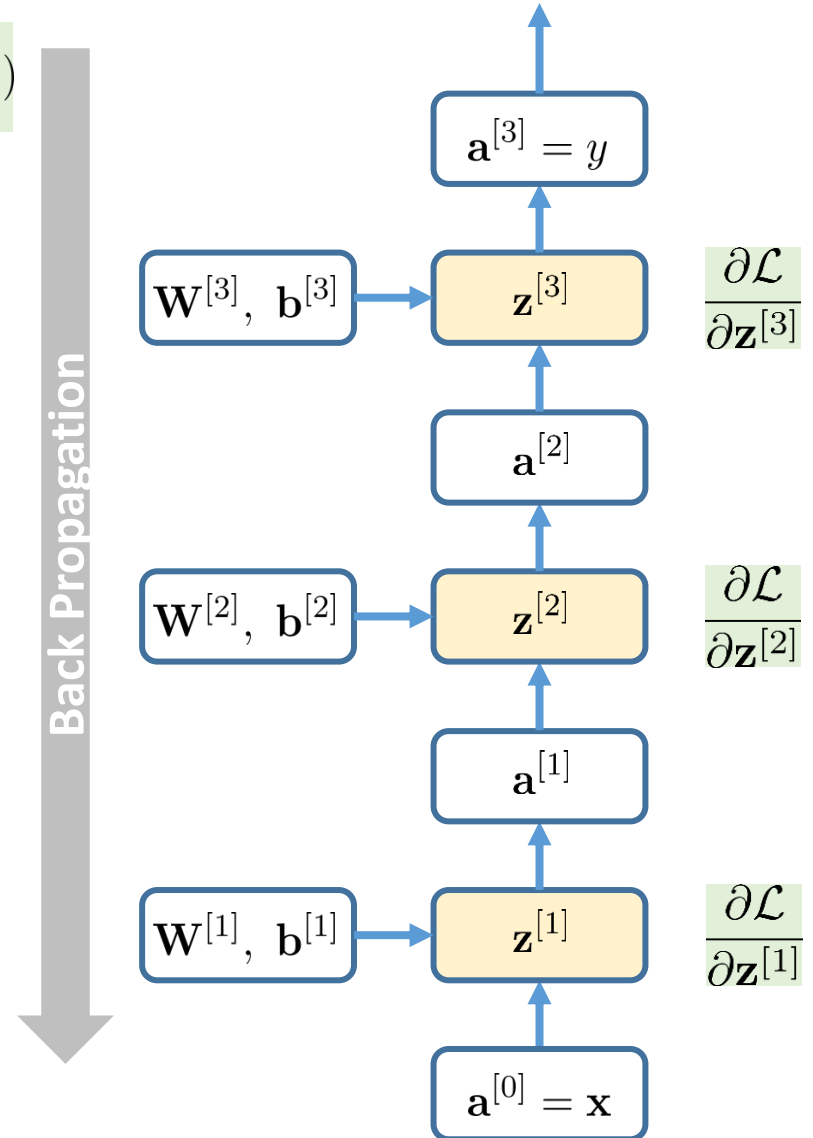
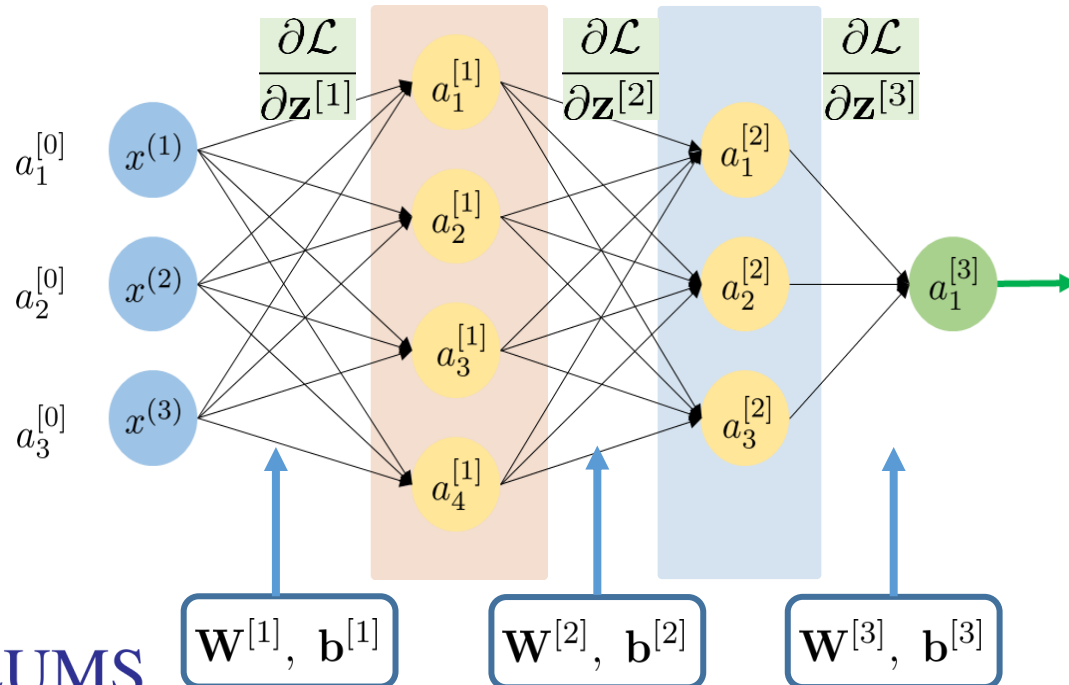


# Neural Networks

## Back Propagation – Vectorization:

### Interpretation of Key Equations:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \mathbf{a}^{[\ell-1]T} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{b}^{[\ell]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} \quad \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell]}} = \left( \mathbf{W}^{[\ell+1]T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[\ell+1]}} \right) \odot g'(\mathbf{z}^{[\ell]})$$



# Neural Networks

## Back Propagation – Vectorization:

- Assuming squared error and sigmoid activation, let's see if it all makes sense. We want to compute:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}}$$

- Base case:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}^{[3]}} \odot g'(\mathbf{z}^{[3]}) = (\mathbf{a}^{[L]} - y) \odot g'(\mathbf{z}^{[3]}) \quad 1 \times 1$$

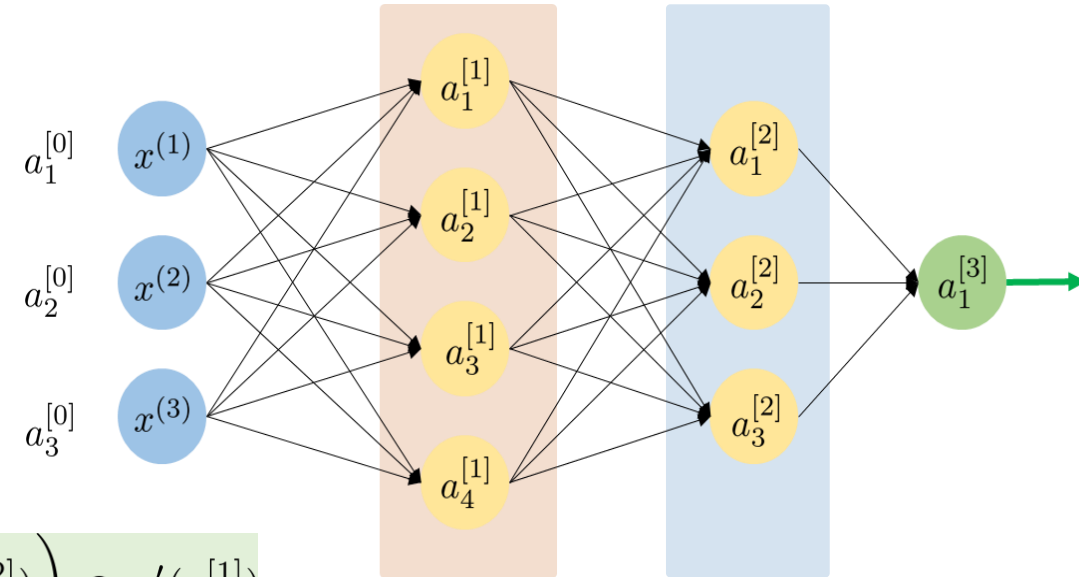
$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} = \left( \mathbf{W}^{[2]T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[2]}} \right) \odot g'(\mathbf{z}^{[1]}) = \left( \mathbf{W}^{[2]T} \left( \mathbf{W}^{[3]T} \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[3]}} \right) \odot g'(\mathbf{z}^{[2]}) \right) \odot g'(\mathbf{z}^{[1]})$$

$4 \times 3 \quad 3 \times 1 \quad 3 \times 1 \quad 4 \times 1$

- Finally:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{[1]}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}^{[1]}} \mathbf{a}^{[0]T}$$

$4 \times 3 \quad 4 \times 1 \quad 1 \times 3$



# Neural Networks

## References:

- TM – Chapter 4
- CB – Chapter 5 (See Section 5.3 for back propagation)
- *'Neural Networks: A Comprehensive Foundation'* and *'Neural Networks and Learning Machines'* by Simon Haykin
- <http://neuralnetworksanddeeplearning.com/>