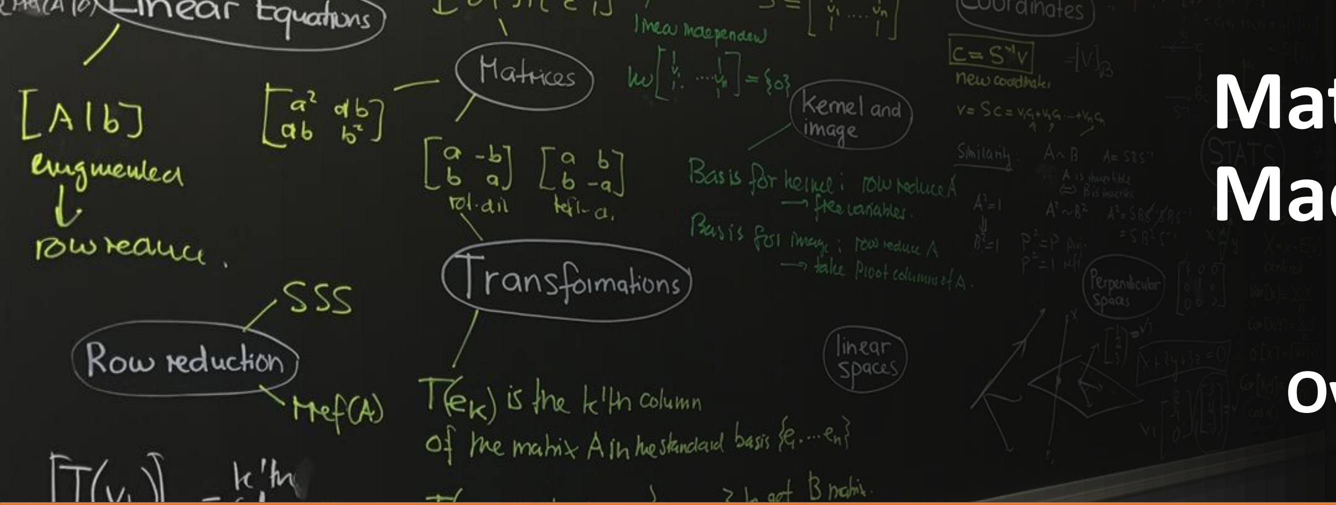


Mathematical Foundations for Machine Learning and Data Science

Overview of Perceptron Classifier, Logistic Regression and Neural Networks

Dr. Zubair Khalid

Department of Electrical Engineering
School of Science and Engineering
Lahore University of Management Sciences

https://www.zubairkhalid.org/ee212_2021.html

LUMS
A Not-for-Profit University

Outline

- Perceptron and Perceptron Classifier

Classification

Recap:

- We assume we have training data D given by

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

Binary or Binomial Classification:

- $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, 1\}$

- *Disease detection, spam email detection, fraudulent transaction, win/loss prediction, etc.*

Multi-class (Multinomial) Classification:

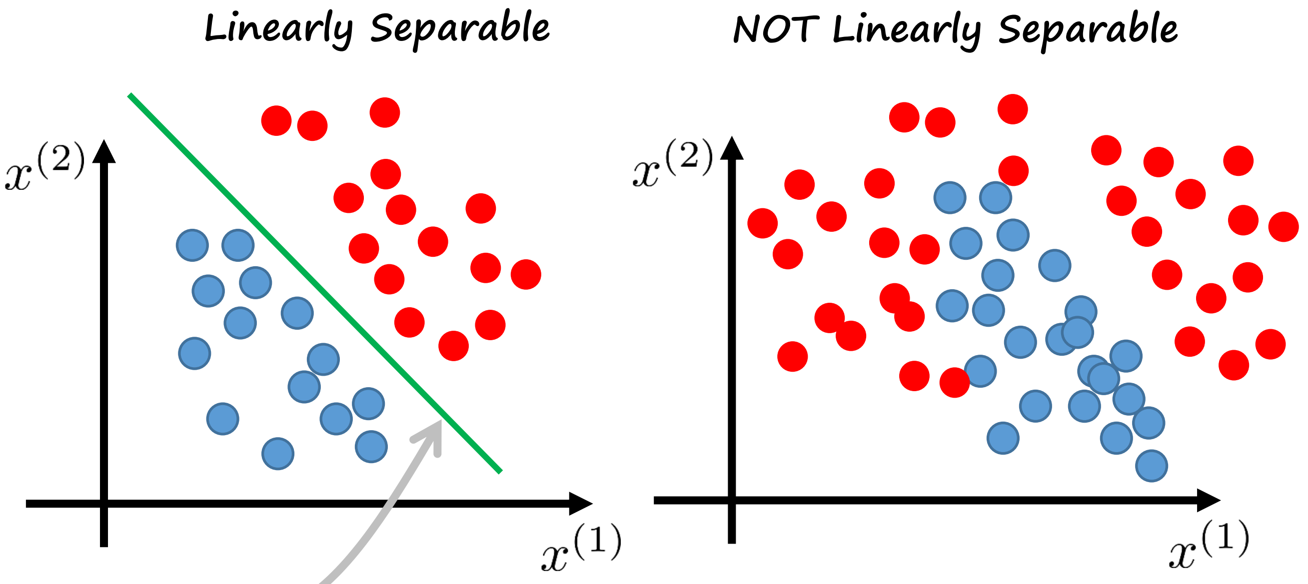
- $\mathcal{Y} = \{1, 2, \ldots, M\}$ (M-class classification)

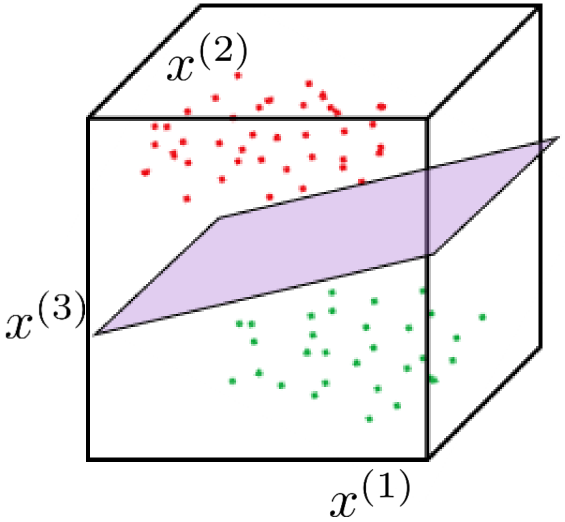
- *Emotion Detection.*
- *Vehicle Type, Make, model, of the vehicle from the images streamed by road cameras.*
- *Speaker Identification from Speech Signal.*
- *Sentiment Analysis (Categories: Positive, Negative, Neutral), Text Analysis.*
- *Take an image of the sky and determine the pollution level (healthy, moderate, hazard).*

Linear Classifiers

Overview:

- *Linear Separability*





- *Linear Classifiers*

$$h(\mathbf{x}) = \boldsymbol{\theta}^T\mathbf{x} + \theta_0$$

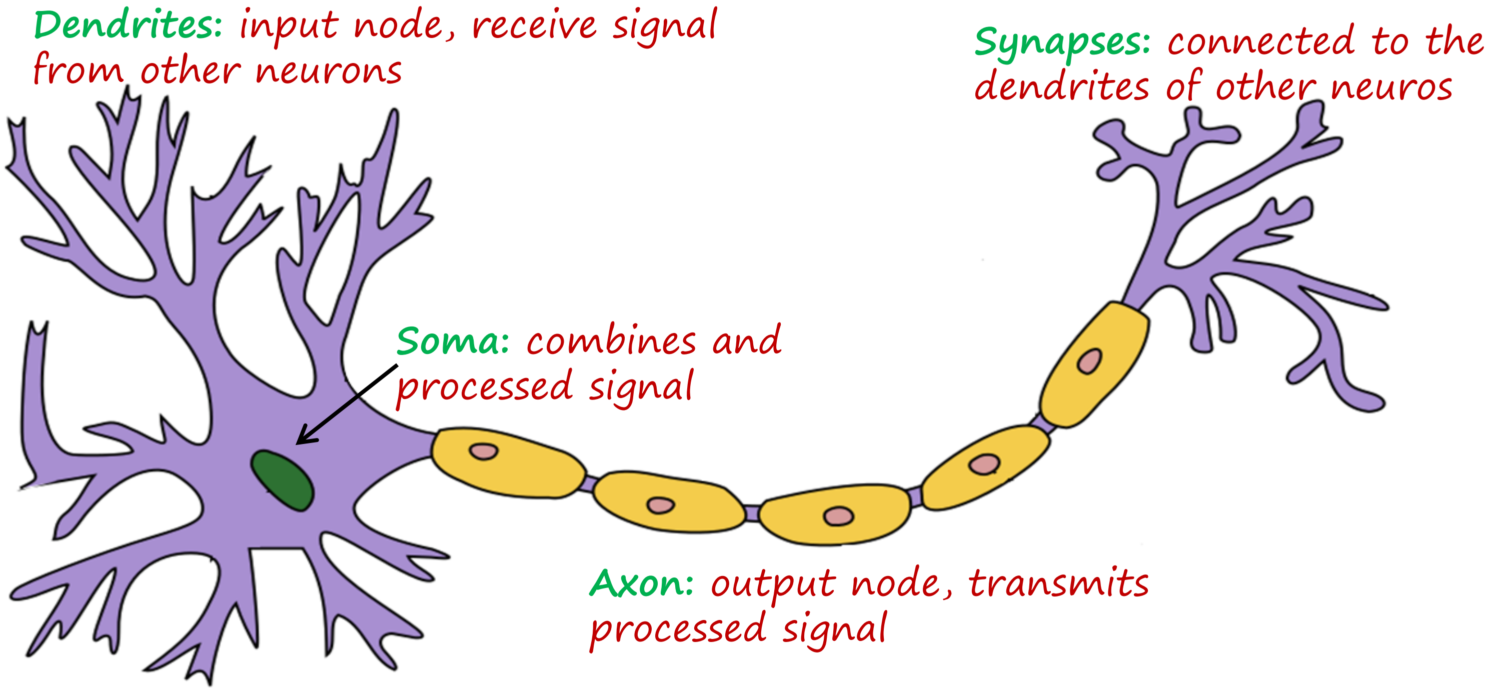
- line in 2D, plane in 3D, hyper-plane in higher dimensions.

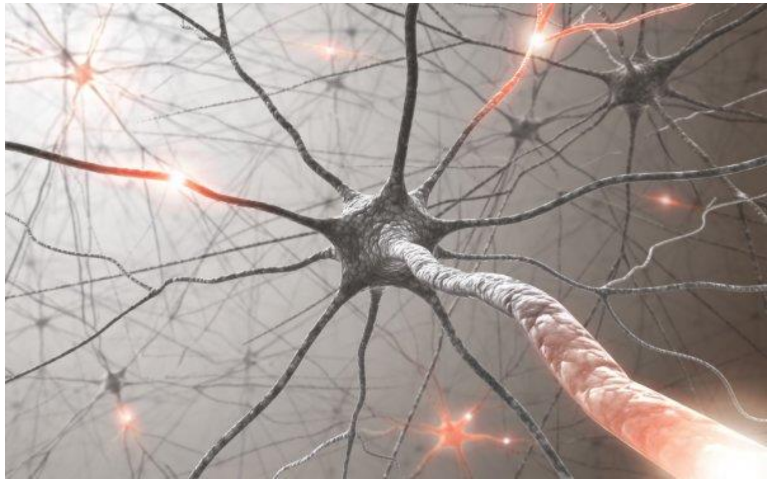
Perceptron Classifier

McCulloch-Pitts (MP) Neuron:

- McCulloch (neuroscientist) and Pitts (logician) proposed a computational model of the biological neuron in 1943.

Biological Neuron (Simplified illustration):





- *Neuron is fired or transmits the signal when it is activated by the combination of input signals.*

Source: McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron | by Akshay L Chandra | Towards Data Science

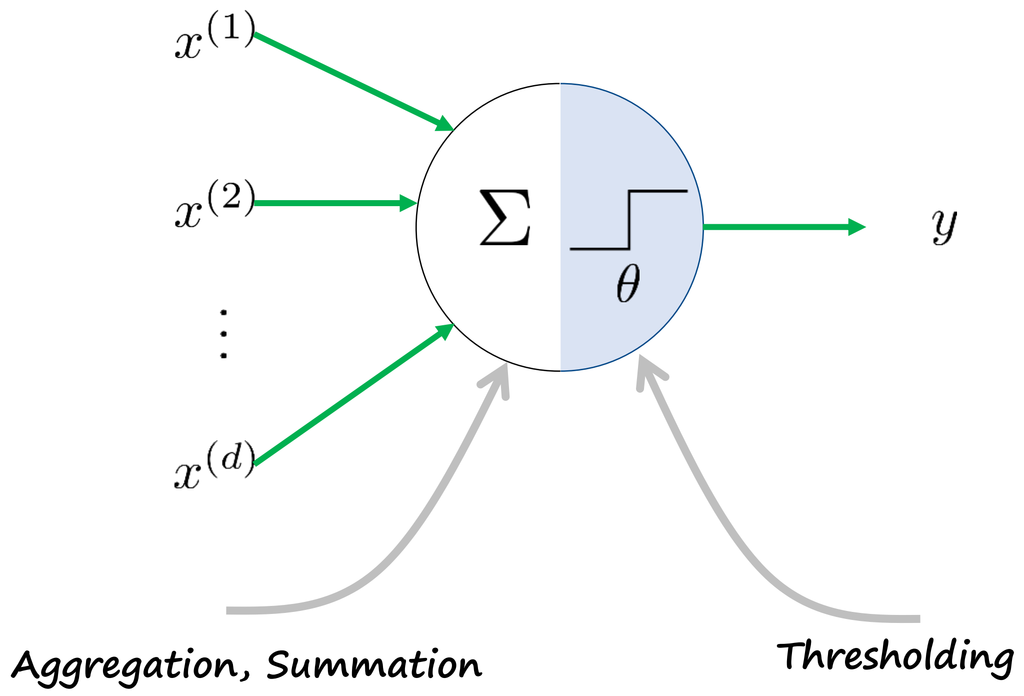
Perceptron Classifier

McCulloch-Pitts (MP) Neuron:

- d number of boolean inputs $x^{(1)}, x^{(2)}, \ldots, x^{(d)} \in \{0, 1\}$.
- Boolean output, $y \in \{0, 1\}$.
- If sum of inputs is less than θ, the output is zero and one otherwise.
- θ is a thresholding parameter that characterizes the neuron.
- Mathematically;

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{d} x^{(i)} \geq \theta \\ 0 & \text{if} \quad \sum_{i=1}^{d} x^{(i)} < \theta \end{cases}$$

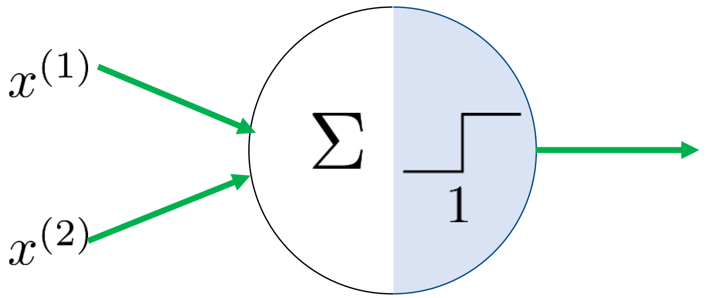
- Idea: Fire the nueron if at least θ number of inputs are active.



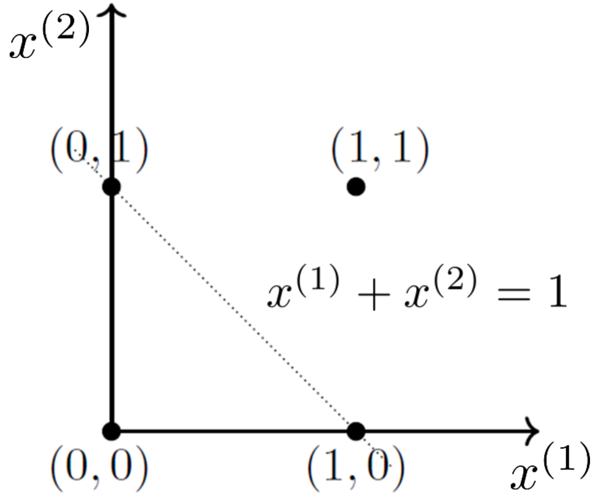
Perceptron Classifier

McCulloch-Pitts Neuron (MP) - Examples:

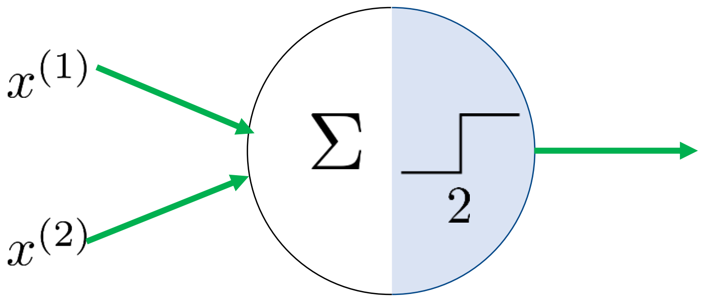
- OR of two inputs.



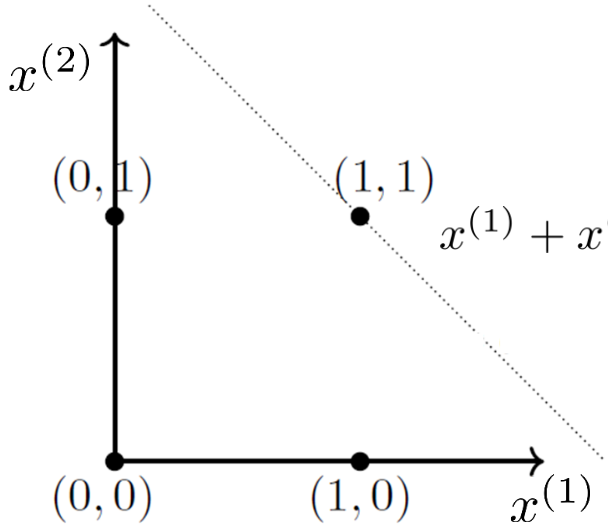
$$y = \begin{cases} 1 & \text{if} \quad x^{(1)} + x^{(2)} \geq 1 \\ 0 & \text{if} \quad x^{(1)} + x^{(2)} < 1 \end{cases}$$



- AND of two inputs.



$$y = \begin{cases} 1 & \text{if} \quad x^{(1)} + x^{(2)} \geq 2 \\ 0 & \text{if} \quad x^{(1)} + x^{(2)} < 2 \end{cases}$$

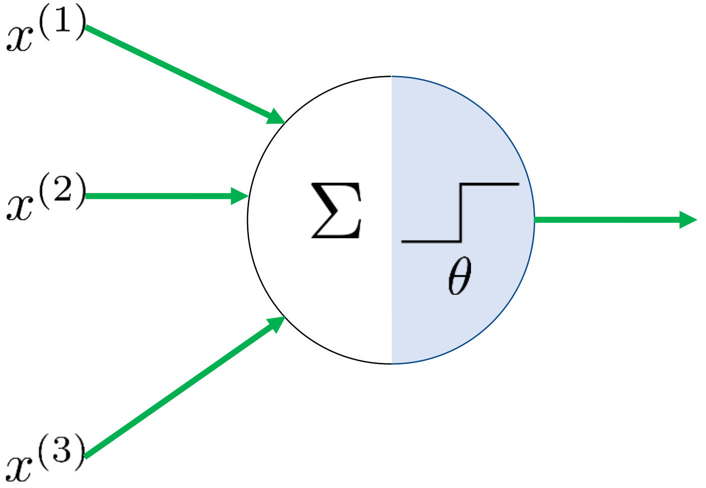


Source: McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron | by Akshay L Chandra | Towards Data Science

Perceptron Classifier

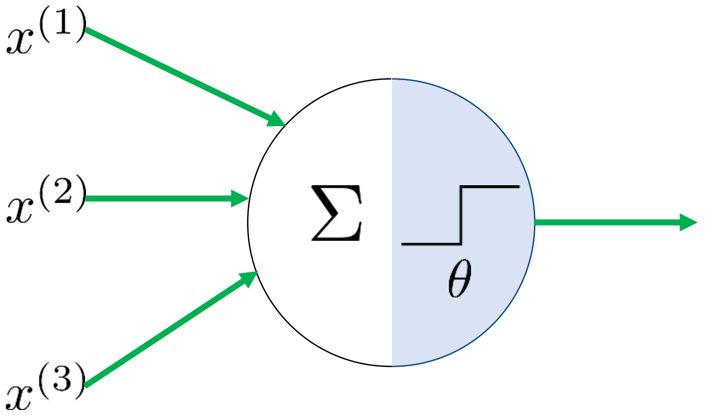
McCulloch-Pitts Neuron (MP) - Examples:

- OR of three inputs.



$$y = \begin{cases} 1 & \text{if} \quad x^{(1)} + x^{(2)} + x^{(3)} \geq 1 \\ 0 & \text{if} \quad x^{(1)} + x^{(2)} + x^{(3)} < 1 \end{cases}$$

- AND of three inputs.



$$y = \begin{cases} 1 & \text{if} \quad x^{(1)} + x^{(2)} + x^{(3)} \geq 3 \\ 0 & \text{if} \quad x^{(1)} + x^{(2)} + x^{(3)} < 3 \end{cases}$$

Source: McCulloch-Pitts Neuron — Mankind's First Mathematical Model Of A Biological Neuron | by Akshay L Chandra | Towards Data Science

Perceptron Classifier

McCulloch-Pitts (MP) Neuron – Limitations:

- Can classify if inputs are linearly separable with respect to the output.
  - How to handle the functions/mappings that are not linearly separable e.g., XOR?
- Can handle only boolean inputs.
  - Gives equal or no weightage to the inputs
  - How can we assign different weights to different inputs?
- We hand-code threshold parameter
  - Can we automate the learning process of the parameter?


NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

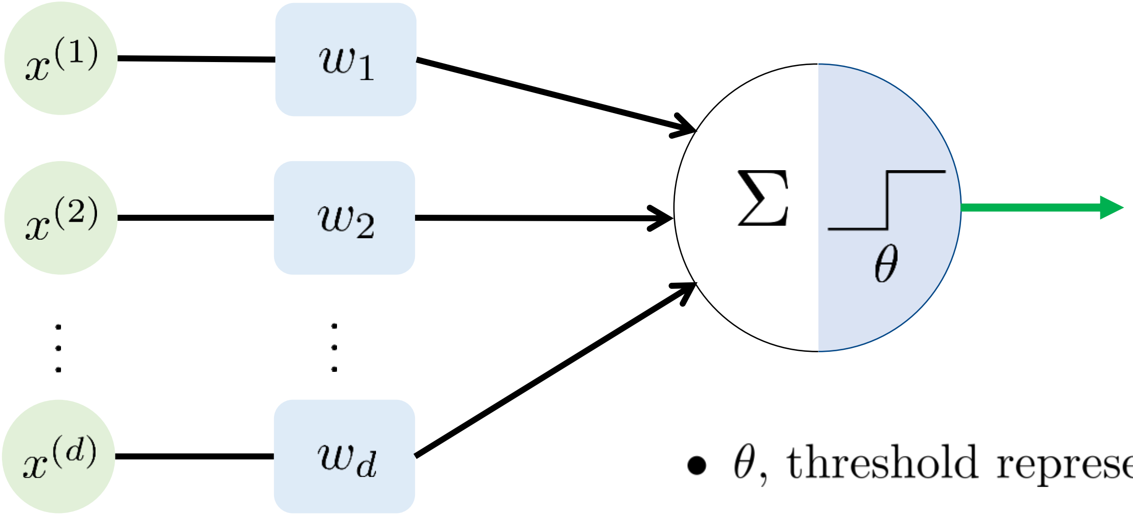
The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be

- To overcome these limitations, another model, known as perception model or perceptron, was proposed by Frank Rosenblatt (1958) and analysed by Minsky and Papert (1969).
  - Inputs real valued, weights used in aggregation
  - Learning of weights and threshold is possible.

Perceptron Classifier

Perceptron:

- d number of real-valued inputs $x^{(1)}, x^{(2)}, \dots, x^{(d)} \in \mathbf{R}$. *(Difference from MP Neuron)*
- Boolean output, $y \in \{0, 1\}$.
- If sum of inputs is less than θ, the output is zero and one otherwise.
- Threshold θ and weights $w_1, w_2, \dots, w_d$ are model parameters. *(Difference from MP Neuron)*



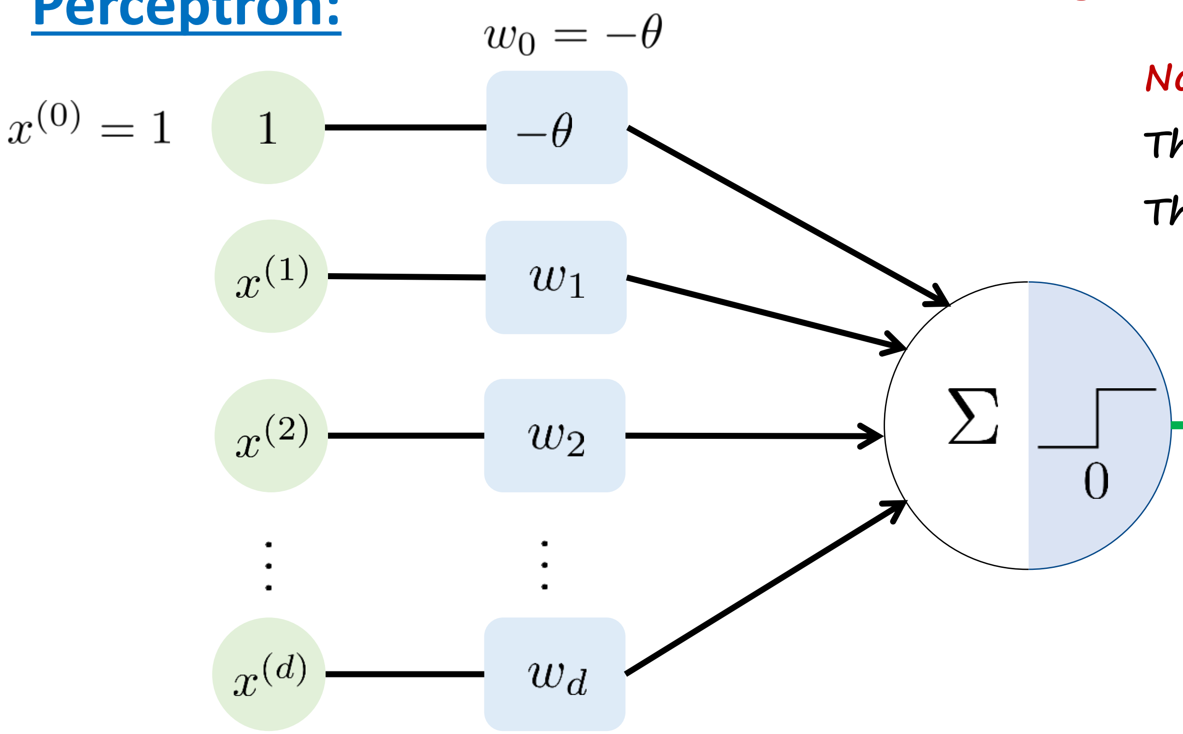
$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} \geq \theta \\ 0 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} < \theta \end{cases} = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} - \theta \geq 0 \\ 0 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} - \theta < 0 \end{cases}$$

- θ, threshold represents a bias here.
- θ can be considered or absorbed as a weight.
- This will make aggregation/thresholding independent of any parameters.

Source: [Perceptron: The Artificial Neuron (An Essential Upgrade To The McCulloch-Pitts Neuron) | by Akshay L Chandra | Towards Data Science]

Perceptron Classifier

Perceptron:



Note:

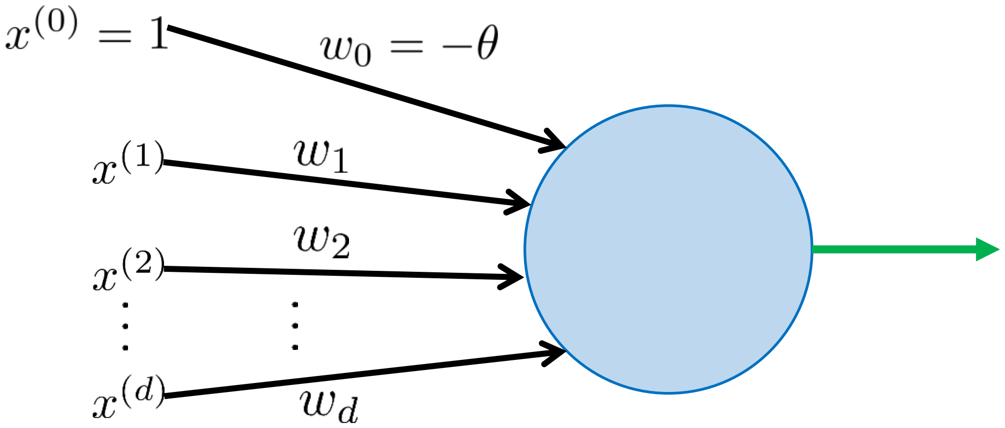
Threshold is zero here.

Threshold has been absorbed as a model weight.

$$y = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} - \theta \geq 0 \text{ or } \sum_{i=0}^{d} w_i\, x^{(i)} \geq 0 \\ 0 & \text{if} \quad \sum_{i=1}^{d} w_i\, x^{(i)} - \theta < 0 \text{ or } \sum_{i=0}^{d} w_i\, x^{(i)} < 0 \end{cases}$$

Alternative (Compact) Representation:

- $\mathbf{x} = [x^{(0)}, x^{(1)}, \ldots, x^{(d)}]$
- $\mathbf{w} = [w_0, w_1, \ldots, w_d]$



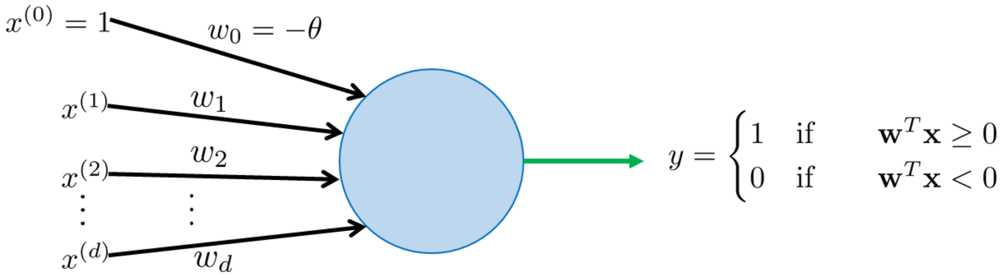
$$y = \begin{cases} 1 & \text{if} \quad \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{if} \quad \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

Perceptron Classifier

Classification using Perceptron:

- Since $\mathbf{w}^T\mathbf{x} = 0$ represents a hyper-plane in the d-dimensional space, we can use perceptron as a binary classifier if the classes are linearly separable.

- How is this different from MP neuron?
  - Inputs are real-valued.
  - We have real-valued weights in the process of aggregation.
  - We can learn the weights.



- Remark:

  If classes are labeled as 1 and −1

$$y = \begin{cases} 1 & \text{if} \quad \mathbf{w}^T\mathbf{x} \geq 0 \\ -1 & \text{if} \quad \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

  We often write output as

$$y = \text{sign}(\mathbf{w}^T\mathbf{x})$$

  sign(.) returns sign of the argument.

Outline

- Perceptron and Perceptron Classifier
- Logistic Regression Classifier

Logistic Regression

Overview:

- kNN: Instance based Classifier
- **Logistic Regression:** Discriminative Classifier
  - Estimate $P(y|x)$ directly from the data

- **'Logistic regression'** is an algorithm to carry out classification.
  - Name is misleading; the word 'regression' is due to the fact that the method attempts to fit a linear model in the feature space.
- Instead of predicting class, we compute the probability of instance being that class.

- Mathematically, model is characterized by variables $\boldsymbol{\theta}$.

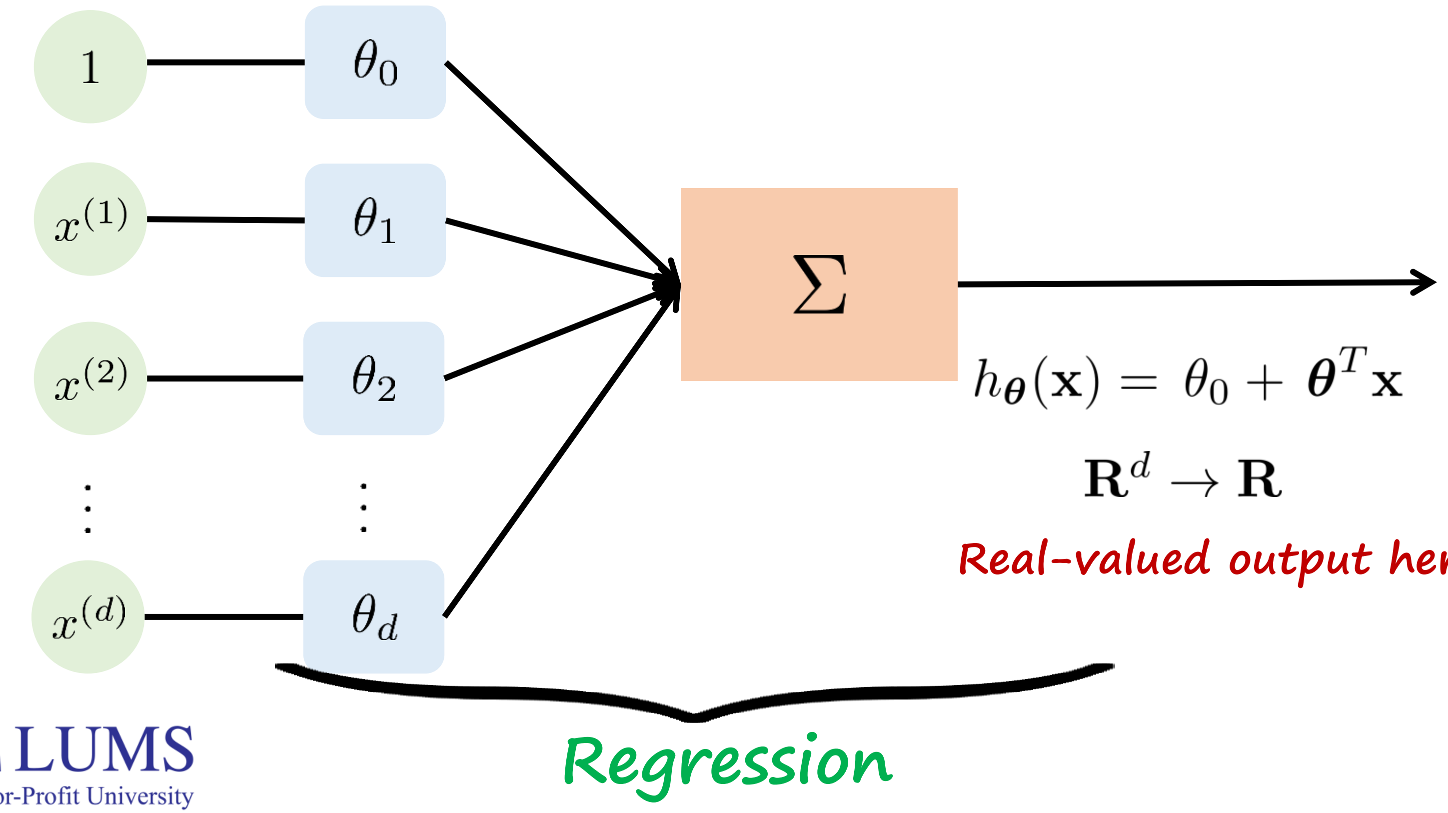
$$h_{\boldsymbol{\theta}}(\mathbf{x}) = P(y|\mathbf{x})$$ Posterior probability

- A simple form of a neural network.

Logistic Regression

Model:

- *Consider a binary classification problem.*
- *We have a multi-dimensional feature space (d features).*
- *Features can be categorical (e.g., gender, ethnicity) or continuous (e.g., height, temperature).*
- ***Logistic regression model:***



$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \boldsymbol{\theta}^T \mathbf{x}$$

$$\mathbf{R}^d \rightarrow \mathbf{R}$$

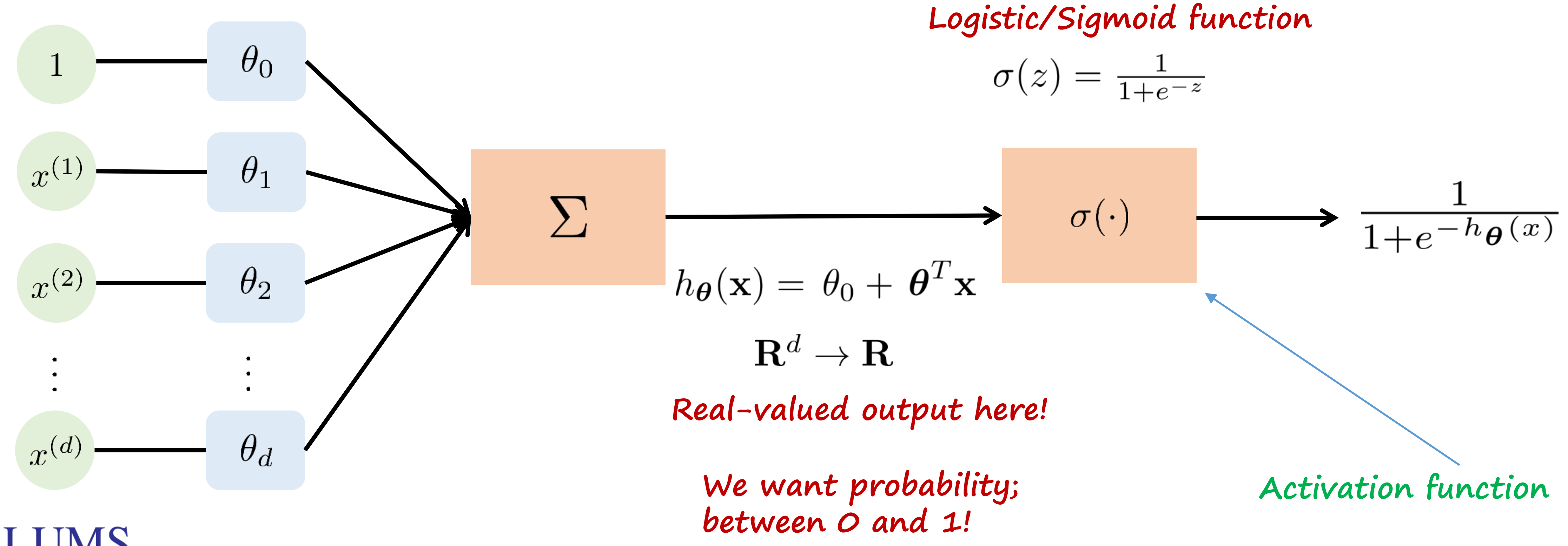
Real-valued output here!

$$\begin{cases} h_{\boldsymbol{\theta}}(\mathbf{x}) > 0 & \text{class 1} \\ h_{\boldsymbol{\theta}}(\mathbf{x}) \leq 0 & \text{class 0} \end{cases}$$

Logistic Regression

Model:

- Consider a binary classification problem.
- We have a multi-dimensional feature space (d features).
- Features can be categorical (e.g., gender, ethnicity) or continuous (e.g., height, temperature).
- <u>Logistic regression model:</u>



Logistic Regression

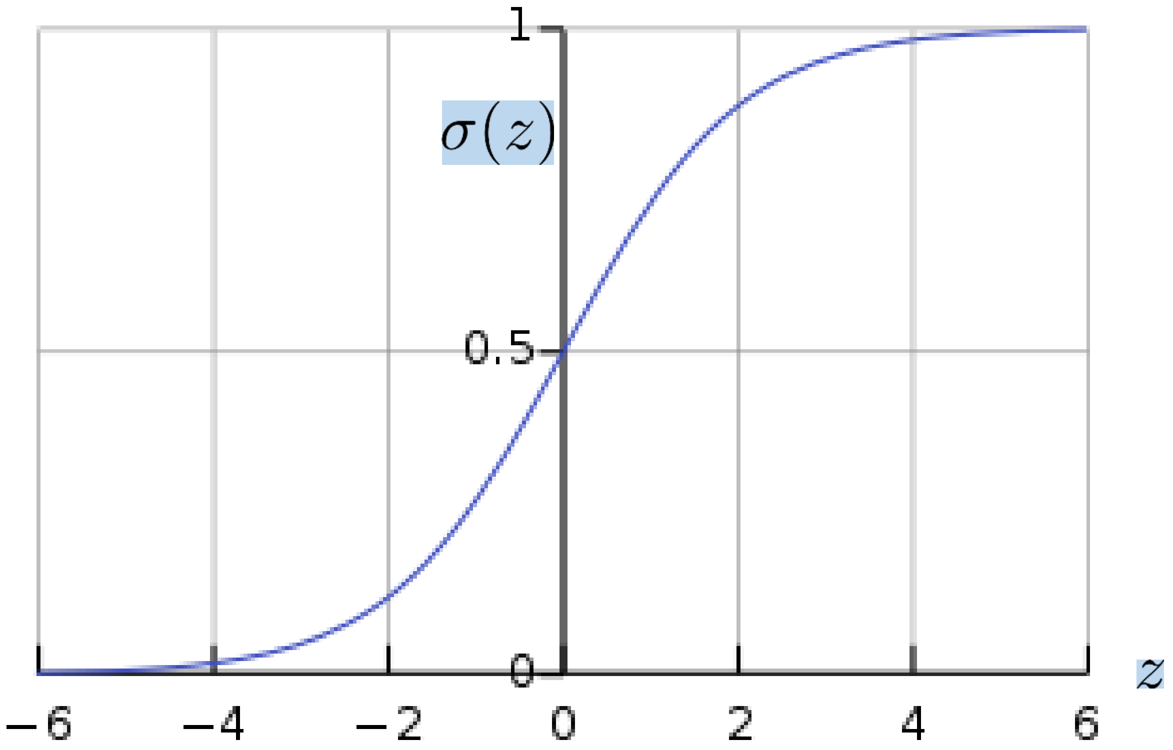
Logistic (Sigmoid) Function

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

- Interpretation: maps $(-\infty, \infty)$ to $(0,1)$
- Squishes values in $(-\infty, \infty)$ to $(0,1)$
- It is differentiable.
- Generalized logistic function:

$$\sigma(z) = \frac{L}{1+e^{-k(z-z_0)}}$$

- Sigmoid: because of S shaped curve

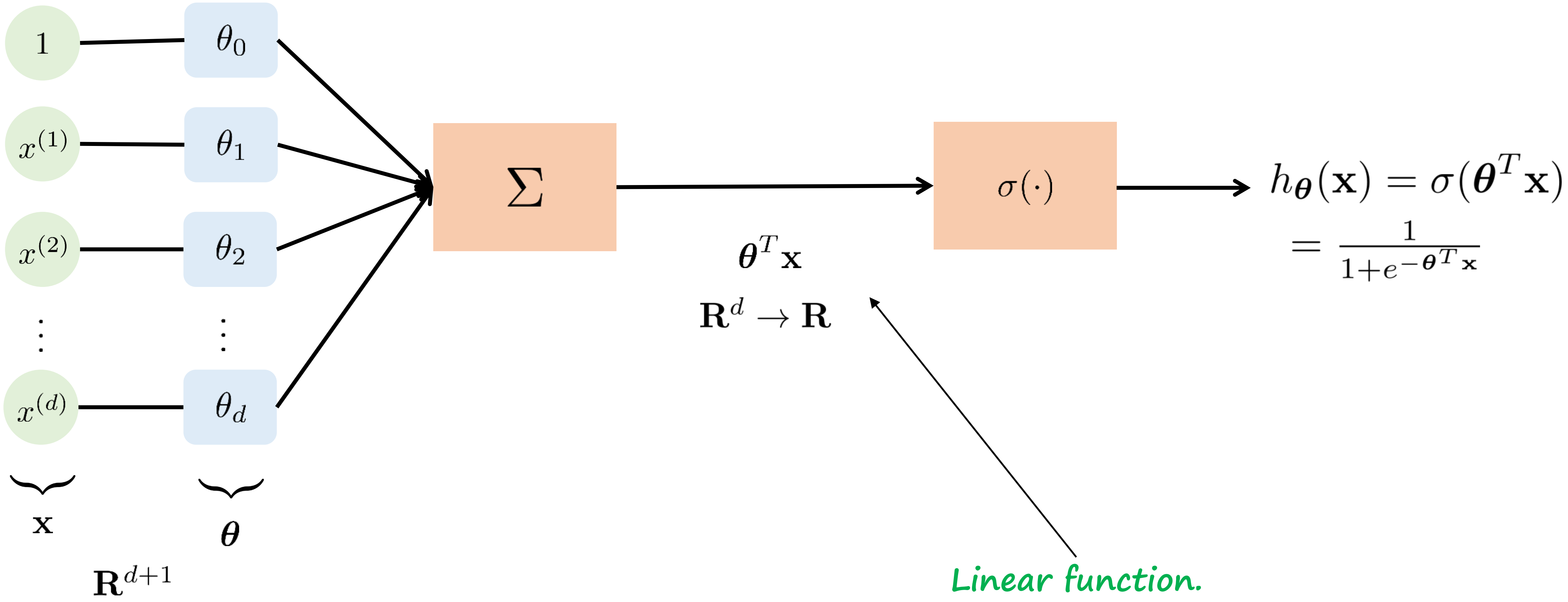


LUMS
A Not-for-Profit University

Logistic Regression

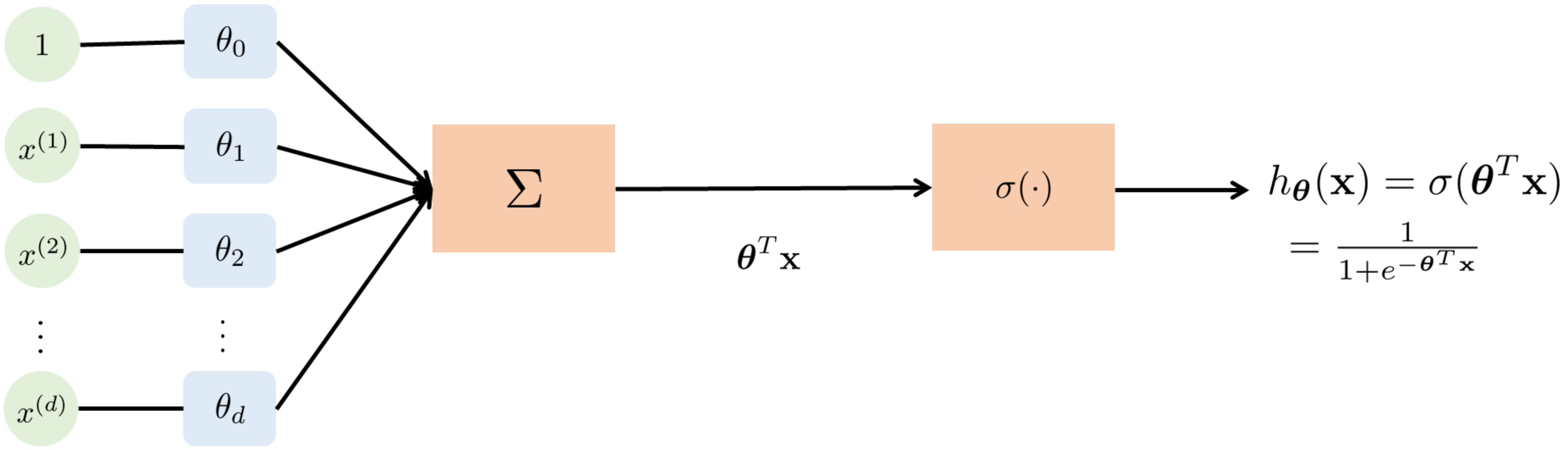
Change in notation:

- *Treat bias term as an input feature for notational convenience.*



Logistic Regression

Classification:



- $h_{\boldsymbol{\theta}}(\mathbf{x}) = P(y = 1|\mathbf{x})$ represents the probability of class membership.
- Assign class by applying threshold as

$$\hat{y} = \begin{cases} \text{Class 1} & \sigma(\boldsymbol{\theta}^T\mathbf{x}) > 0.5 \\ \text{Class 0} & \text{otherwise} \end{cases}$$

- 0.5 is the threshold defining decision boundary.
- We can also use values other than 0.5 as threshold.

Outline

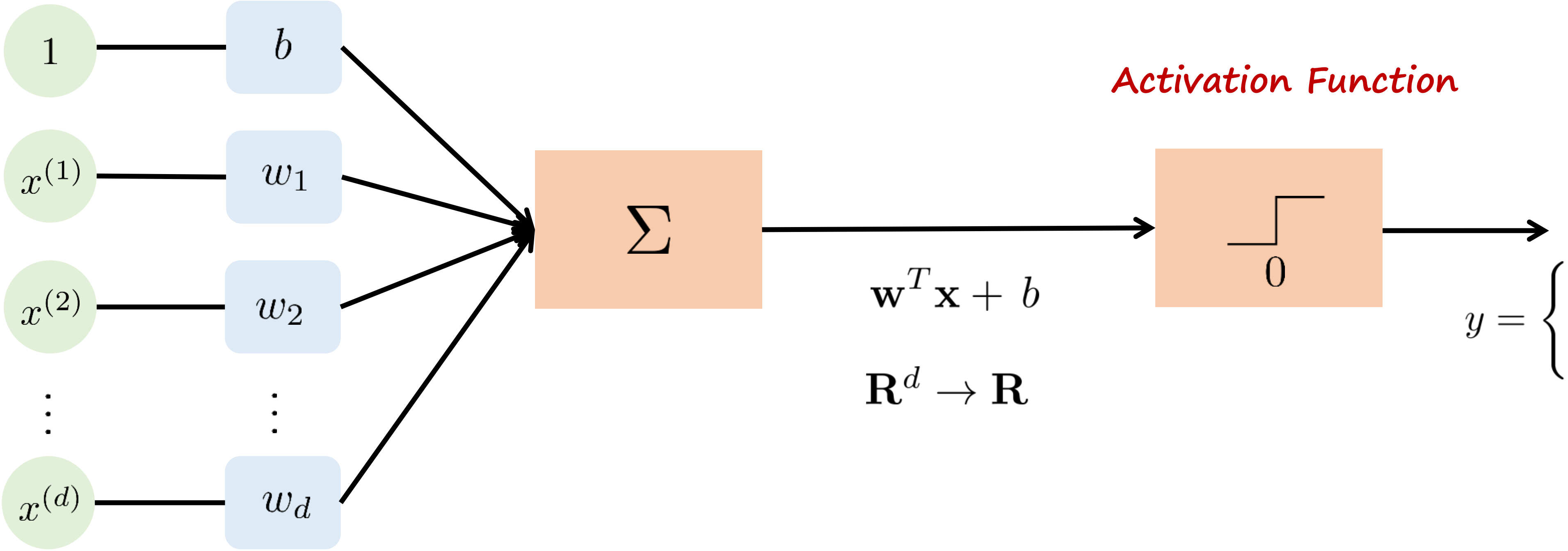
- Perceptron and Perceptron Classifier
- Logistic Regression Classifier
- Neural Networks
  - Neural networks connection with perceptron and logistic regression

Neural Networks

Connection with Logistic Regression and Perceptron:

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

Perceptron Model:



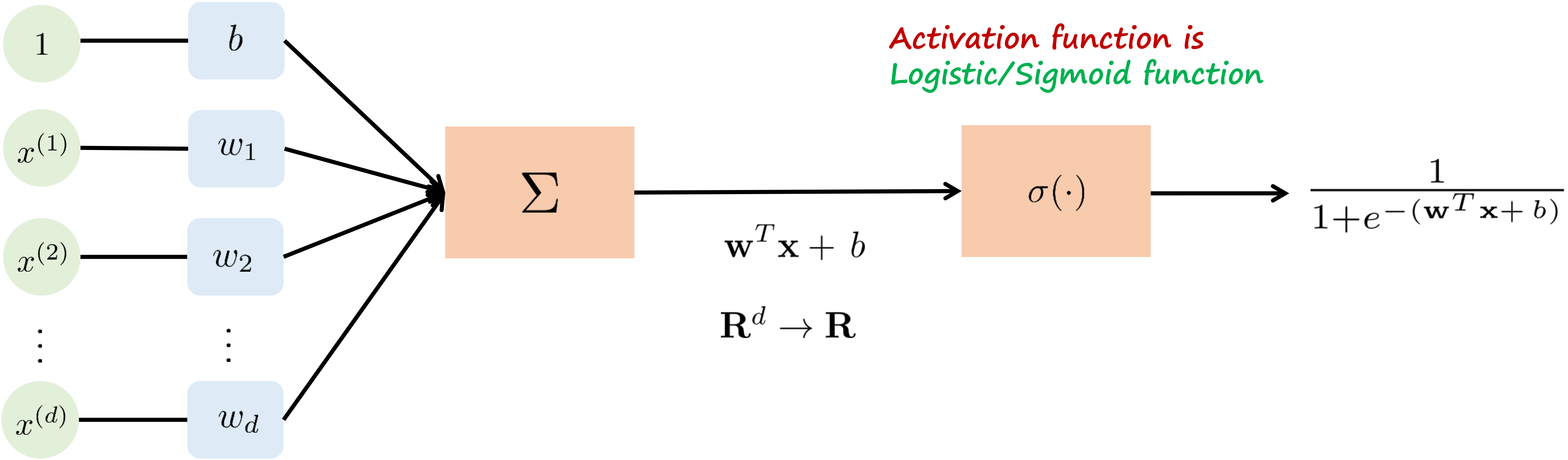
$$y = \begin{cases} 1 & \text{if} \quad \mathbf{w}^T\mathbf{x} + b \geq 0 \\ 0 & \text{if} \quad \mathbf{w}^T\mathbf{x} + b < 0 \end{cases}$$

Neural Networks

Connection with Logistic Regression and Perceptron:

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

Logistic Regression Model:



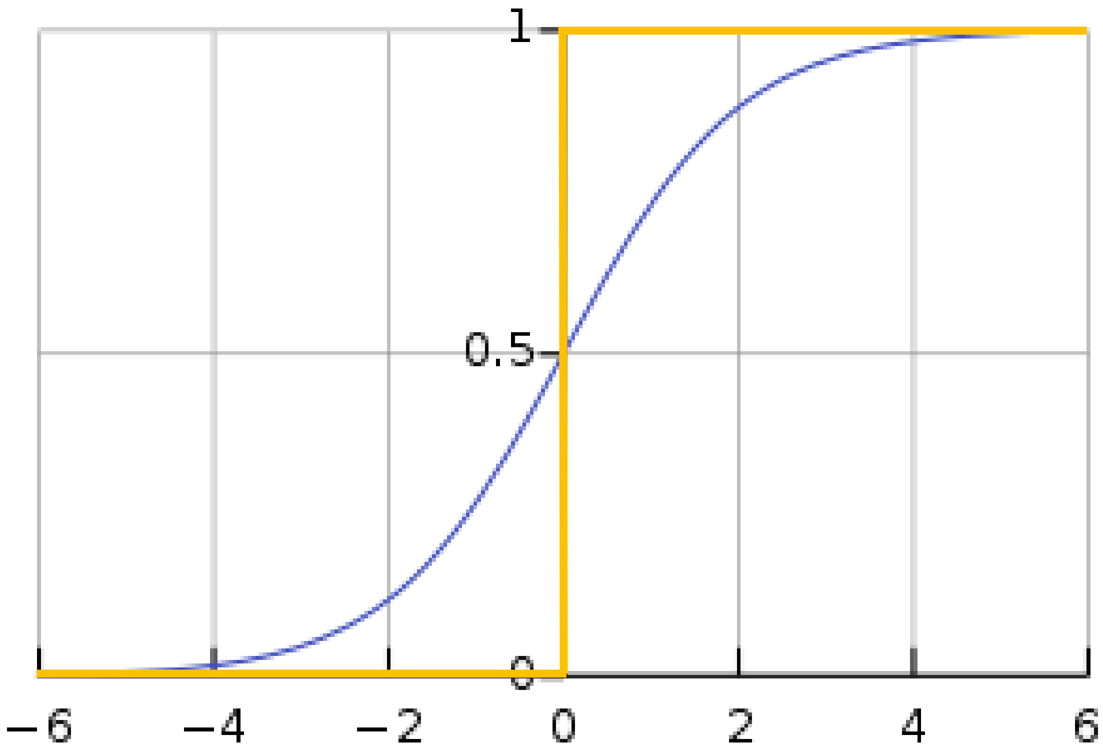
Logistic Regression Model, aka Sigmoid Neuron

Neural Networks

Connection with Logistic Regression and Perceptron:

Activation Function

Perceptron vs Sigmoid Neuron



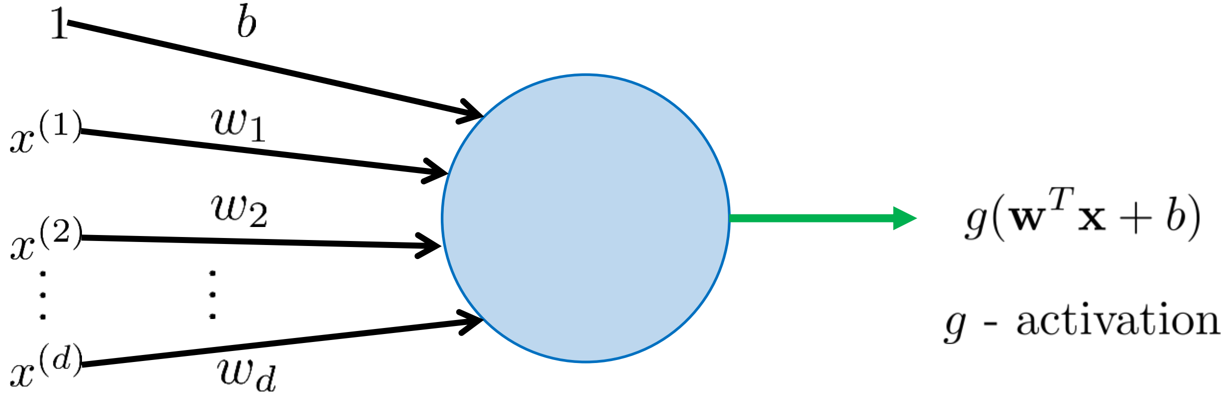
Weighted sum of inputs + bias

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

Neural Networks

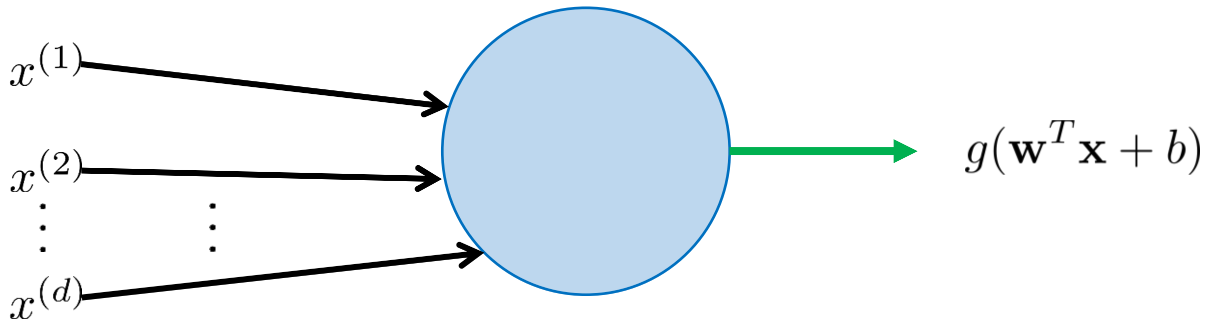
Neuron Model:

Compact Representation:



g - activation function.

More Compact Representation:

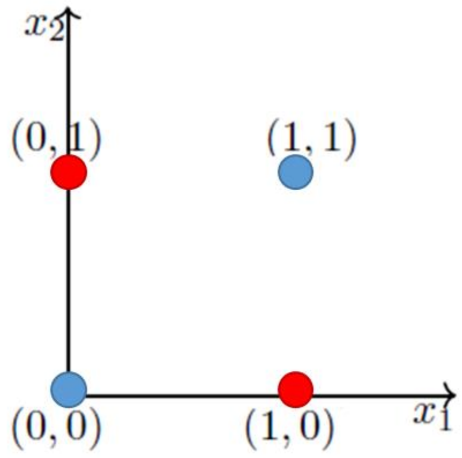


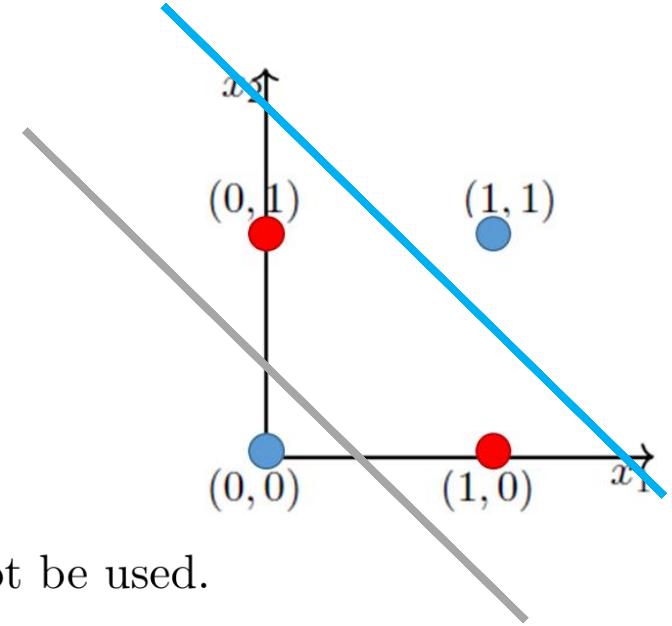
- Neuron model: Characterized by weights, bias and activation function.
  - Weights $\mathbf{w}$, bias b - model parameters
  - Activation function g - hyperparameter

Neural Networks - 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



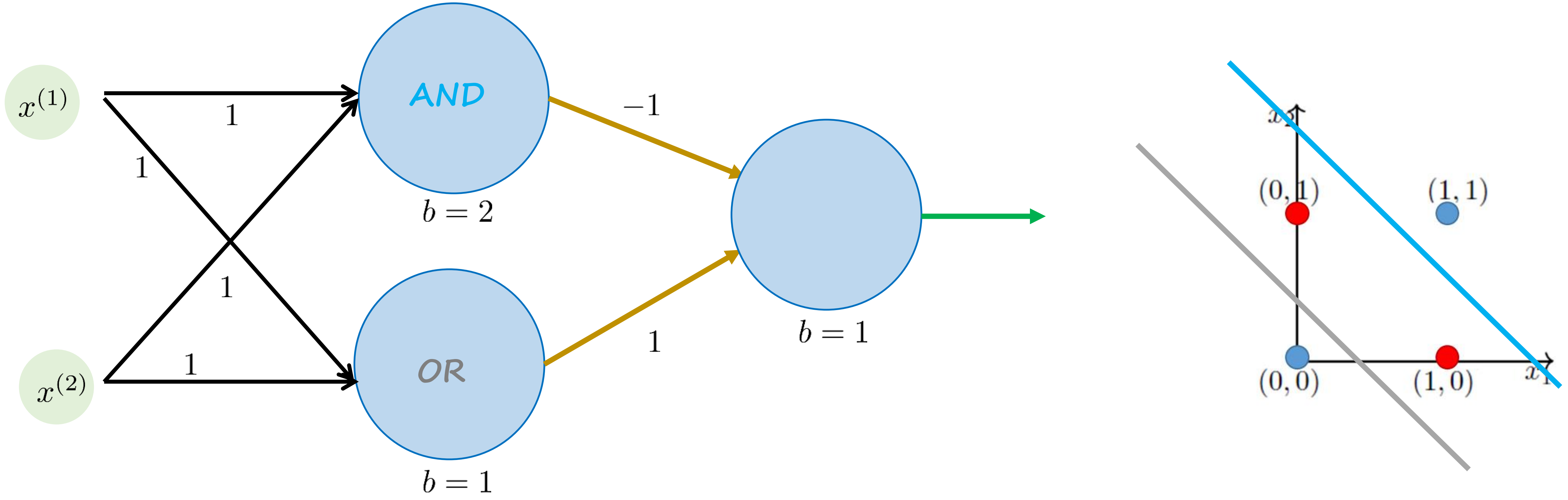


Idea:
Learn AND and OR boundaries.

- Classes are not linearly separable: linear classifier cannot be used.
- We can either transfrom 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:

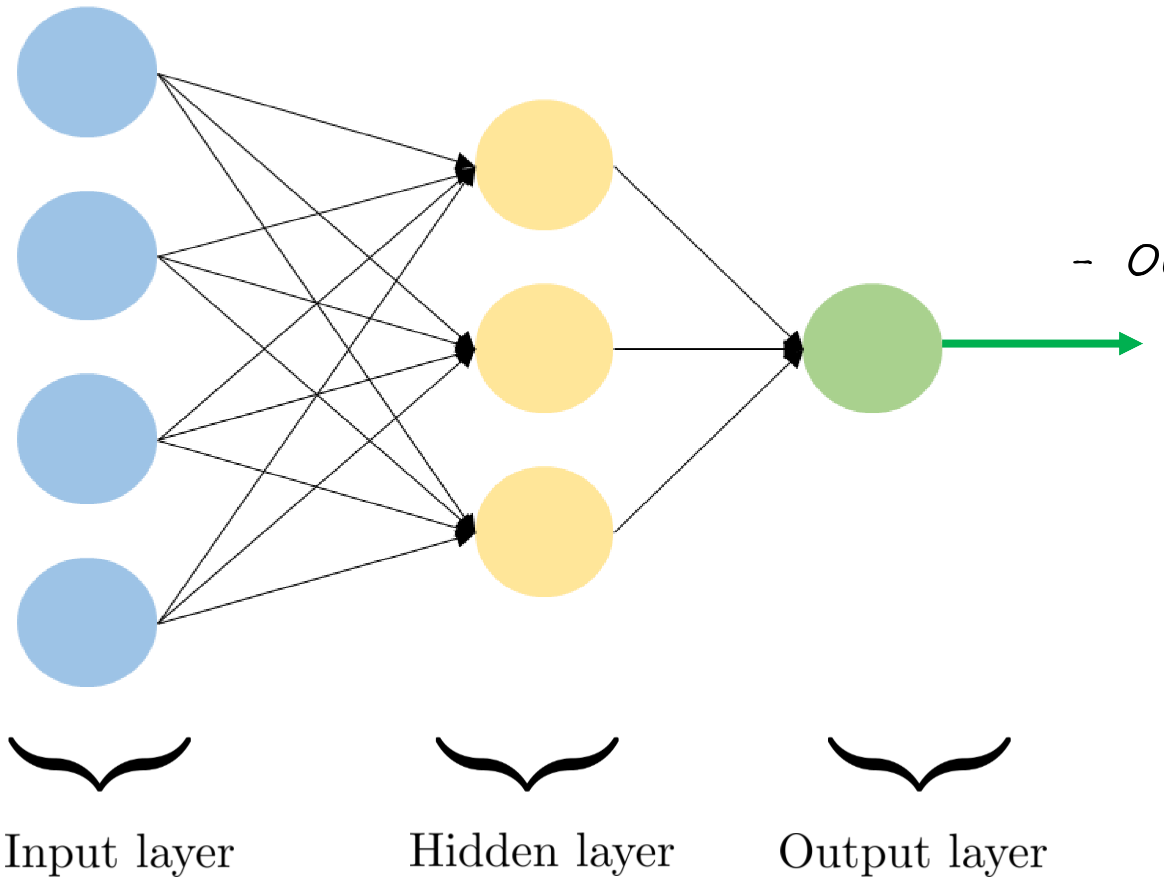


- This is a neural network; a network of perceptrons, aka multi-layer perceptron (MLP).

Neural Networks

Neural Networks

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



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

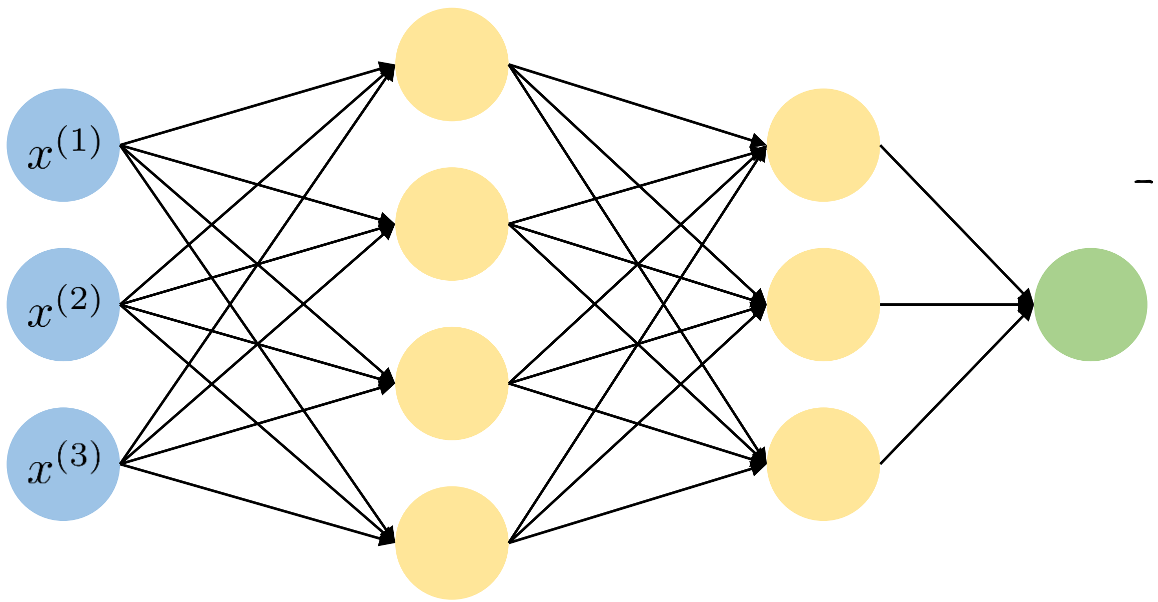
Each unit in the network is called a neuron and the network of neurons is called Neural Network.

- 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*

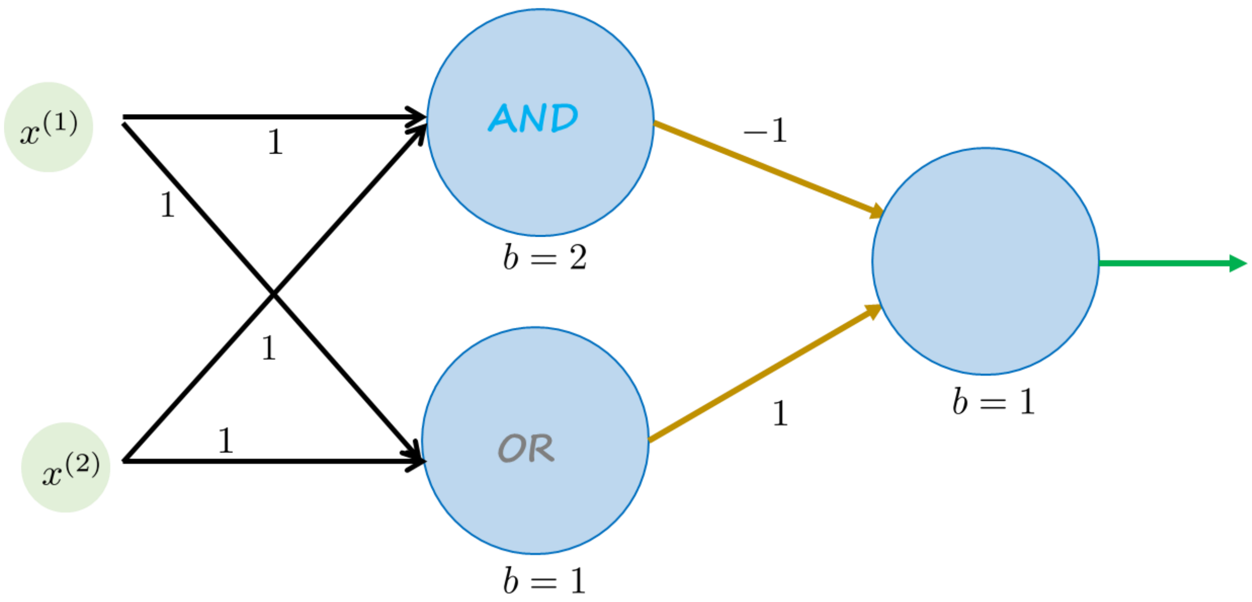
- We do not count the input layer because there are no parameters assoictaed with it.
- Neural networks with neurons are also referred to as MLPs but we will refer to the network as MLP only when it is constructed using perceptrons.

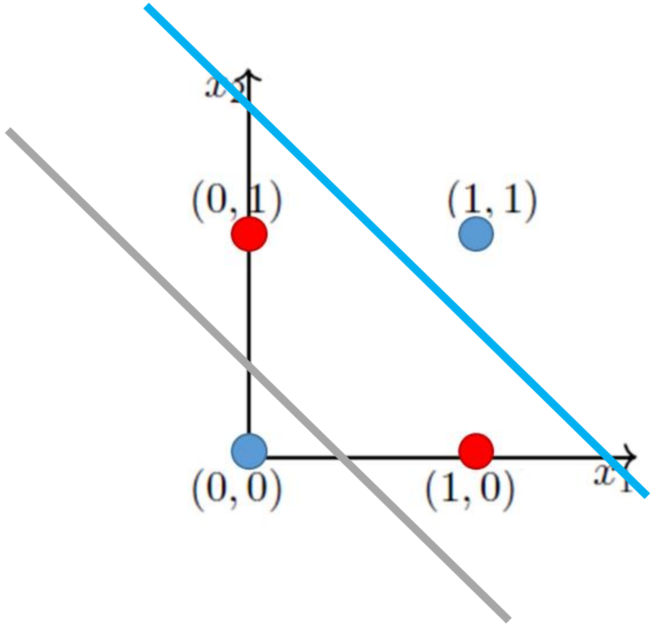
Feedforward Neural Network: ***Output from one layer is an input to the next layer.***

Neural Networks

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

Intuition: *XOR example*

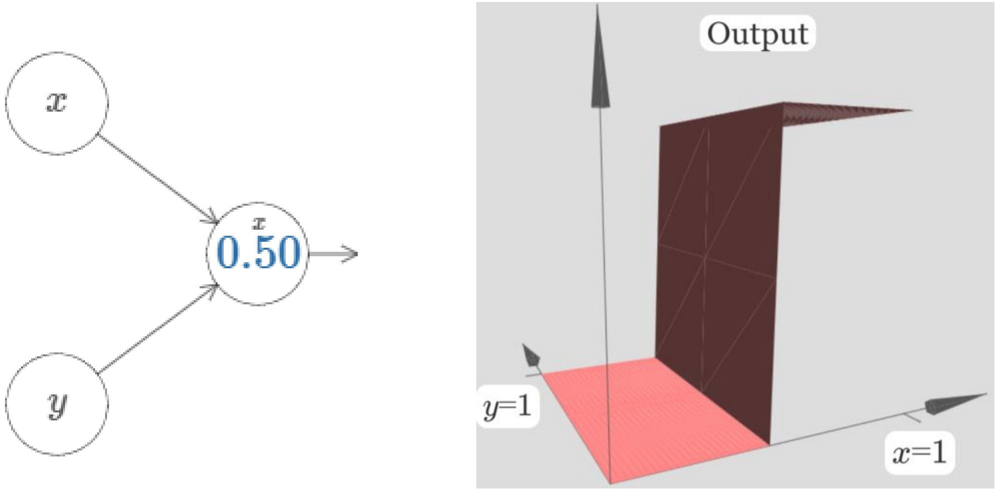




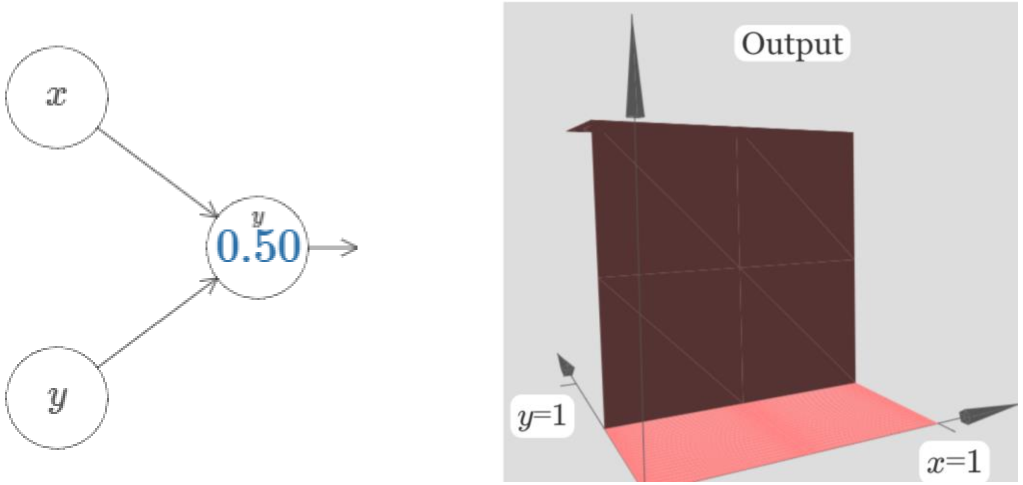
Neural Networks

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.



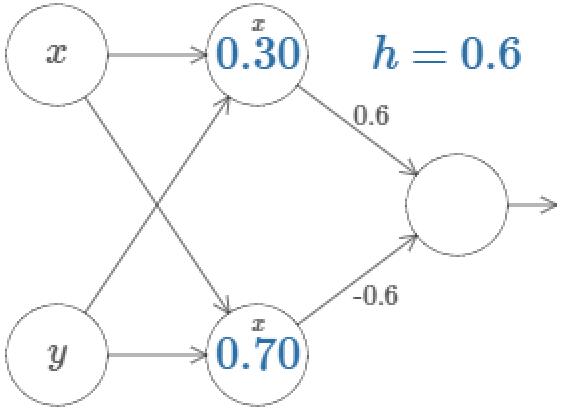
- bias=0.5 indicated.
- weight for y is very large.
- weight for x is zero.

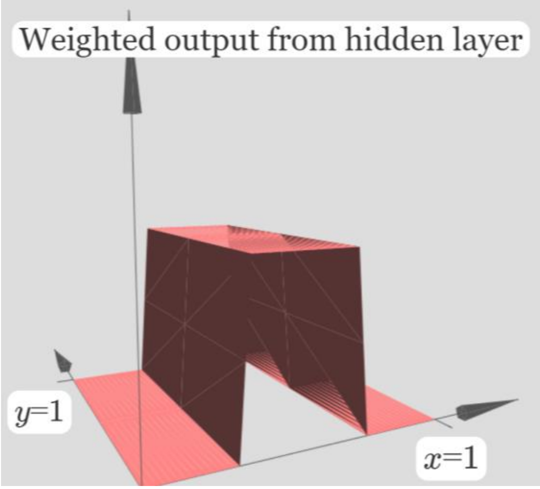
Source: http://neuralnetworksanddeeplearning.com/chap4.html

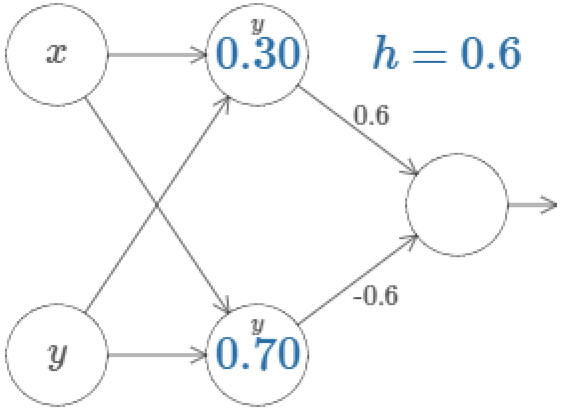
Neural Networks

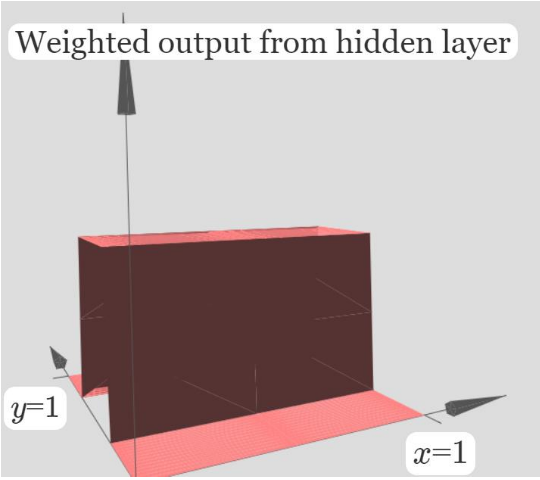
What kind of functions can be modeled by a neural network?

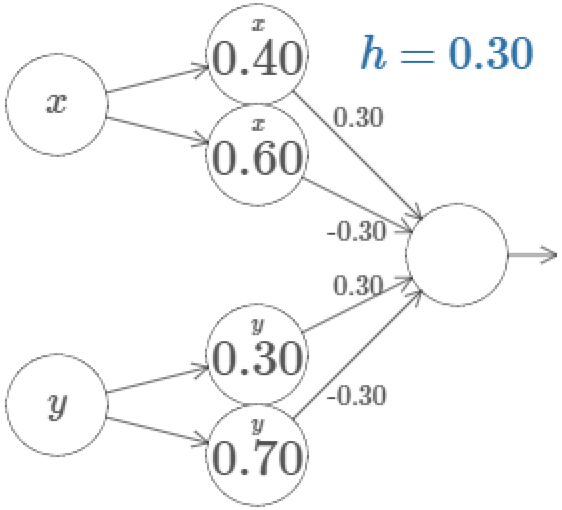
Intuition: *Example (Multi layer)*

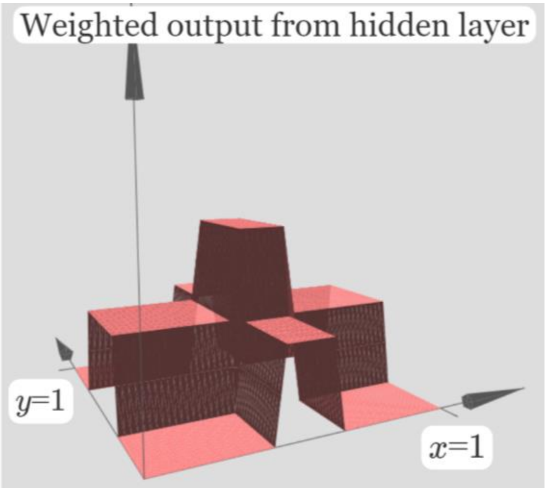












Source: http://neuralnetworksanddeeplearning.com/chap4.html

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.

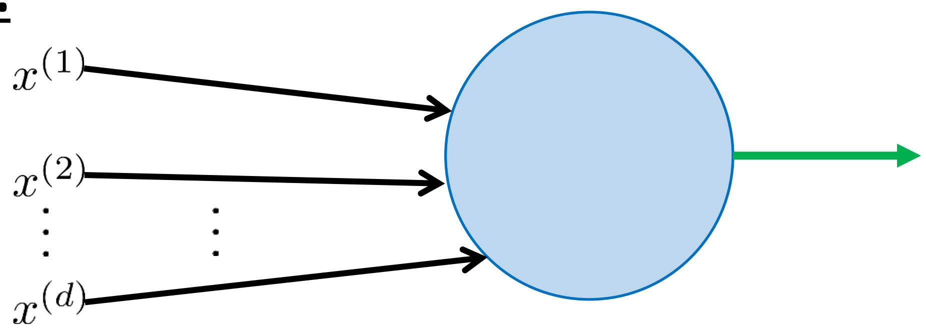
Outline

- Perceptron and Perceptron Classifier
- Logistic Regression Classifier
- Neural Networks
  - Neural networks connection with perceptron and logistic regression
- Neural networks 'Forward Pass'

Neural Networks

Neural Networks – Notation:

Single Neuron:



$g(\mathbf{w}^T\mathbf{x} + b)$

$z = \mathbf{w}^T\mathbf{x} + b$ Pre-activation

$g(z)$ Activation

- If we stack n training data in a matrix $\mathbf{X}$ of size $d \times n$, that is, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$
- Using $\mathbf{X}$ and by defining $\mathbf{1}$ a row vector of ones of length n, we can define 'pre-activatation' 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.

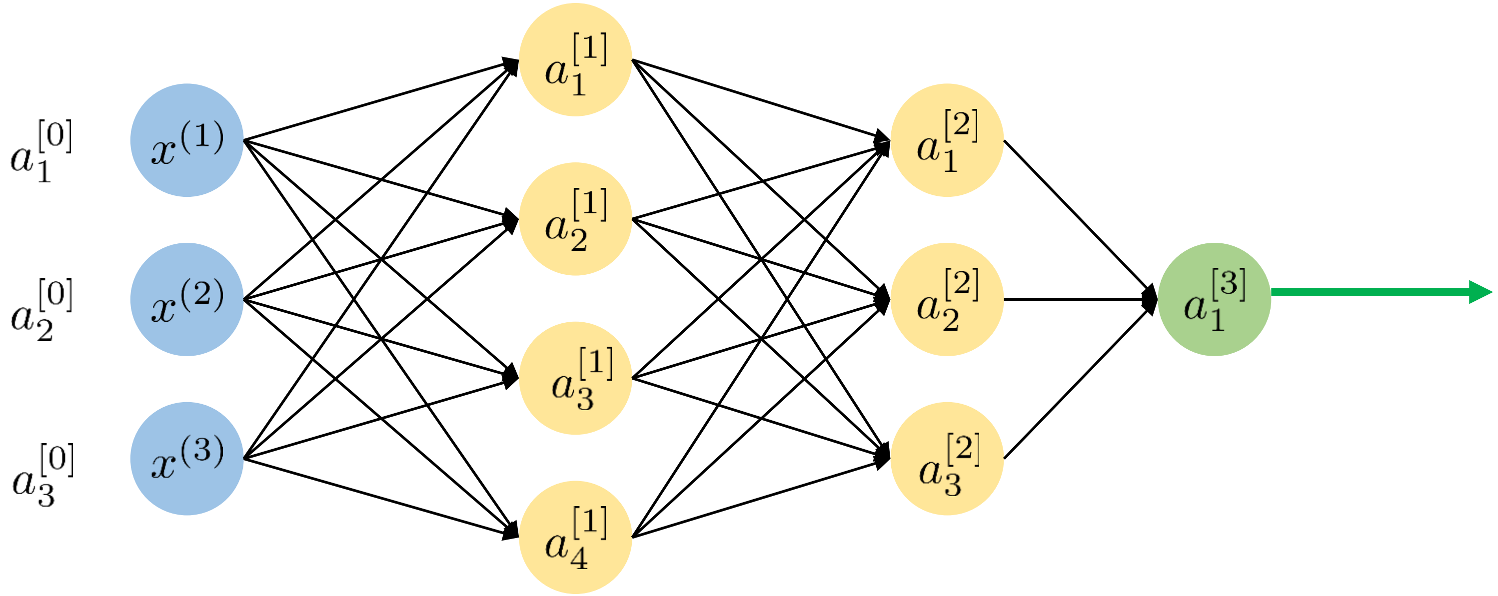
Neural Networks

Neural Networks – Notation:

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

Example: 3-layer network, 2 hidden layers

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

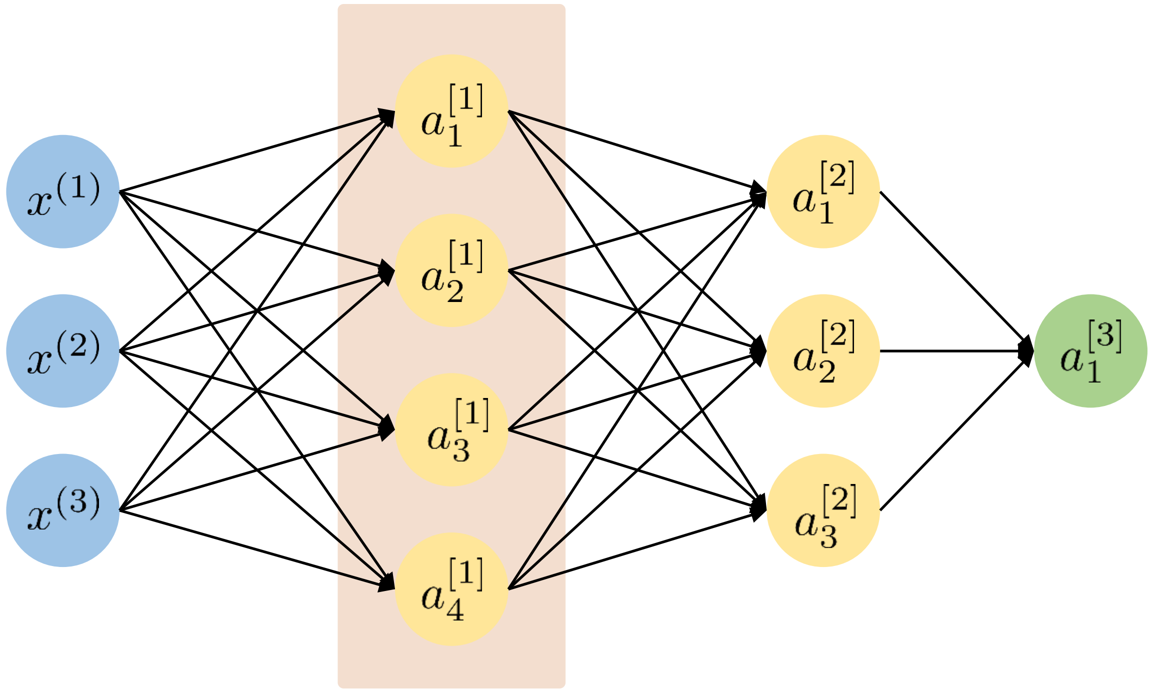


Neural Networks

Neural Networks – Notation:

- $\mathbf{w}_i^{[\ell]}$ and $b_i^{[\ell]}$ denote the weight and bias associated with the i-th node in the ℓ-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 ℓ-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:

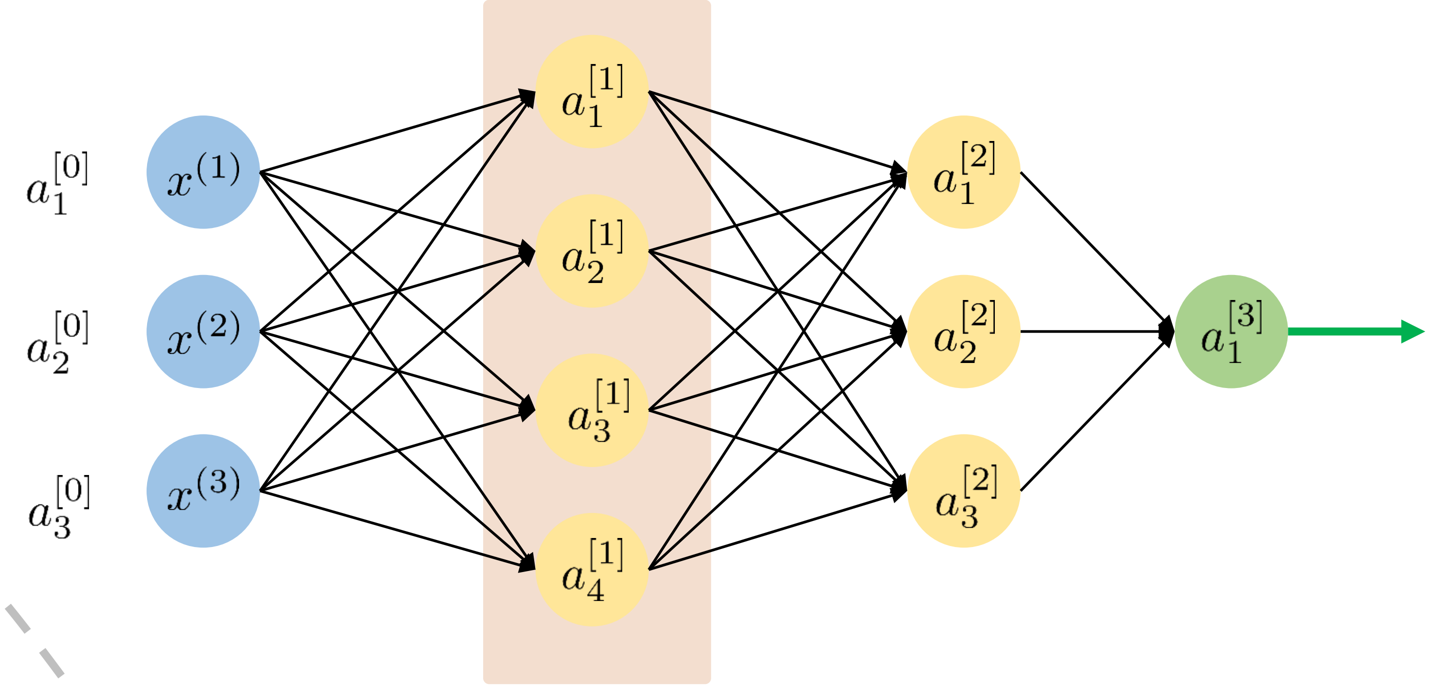
Layer 1 output

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

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

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

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



$$\mathbf{W}^{[1]} = \begin{bmatrix} \mathbf{w}_1^{[1]T} \\ \mathbf{w}_2^{[1]T} \\ \mathbf{w}_3^{[1]T} \\ \mathbf{w}_4^{[1]T} \end{bmatrix} \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \\ b_4^{[1]T} \end{bmatrix} \quad \mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]T} \\ z_2^{[1]T} \\ z_3^{[1]T} \\ z_4^{[1]T} \end{bmatrix}$$

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

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

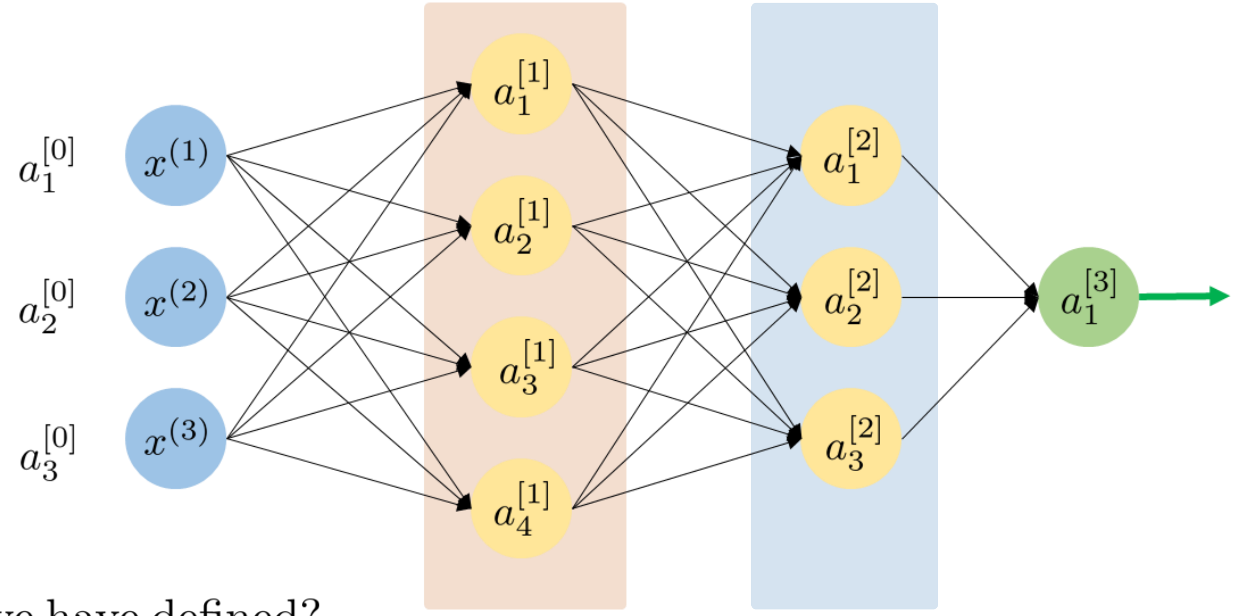
- $\mathbf{W}^{[1]}$ and $\mathbf{b}^{[1]}$ are the parameters of the first layer.

Neural Networks

Neural Networks – Forward Pass:

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

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



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

A. No. of nodes × No. of inputs. 4×3

No. of nodes × No. nodes in the previous layer.

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

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

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

- Q. What is the size of $\mathbf{W}^{[2]}$? 3×4
- Q. What is the size of $\mathbf{W}^{[3]}$? 1×3
- $\mathbf{W}^{[\ell]}$ and $\mathbf{b}^{[\ell]}$ are the parameters of the ℓ-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:

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

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

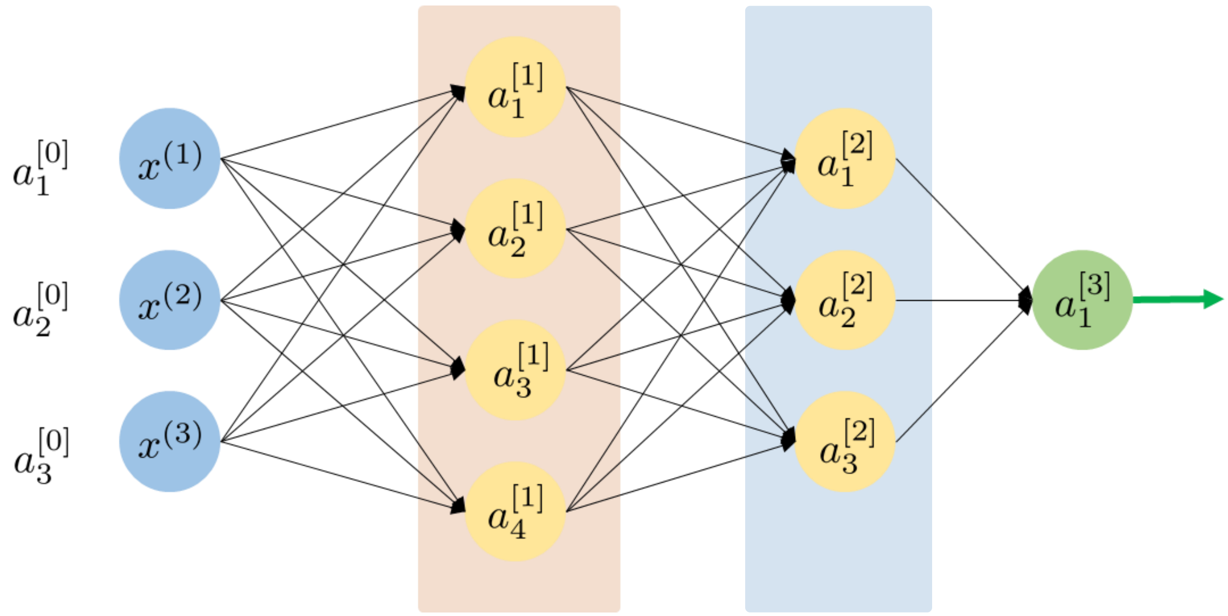
$$(4 \times 1) = (4 \times 3)(3 \times 1) + (4 \times 1)$$

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

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

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

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



- In general, we have

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

for $\ell = 1, 2, \ldots, L$, 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?

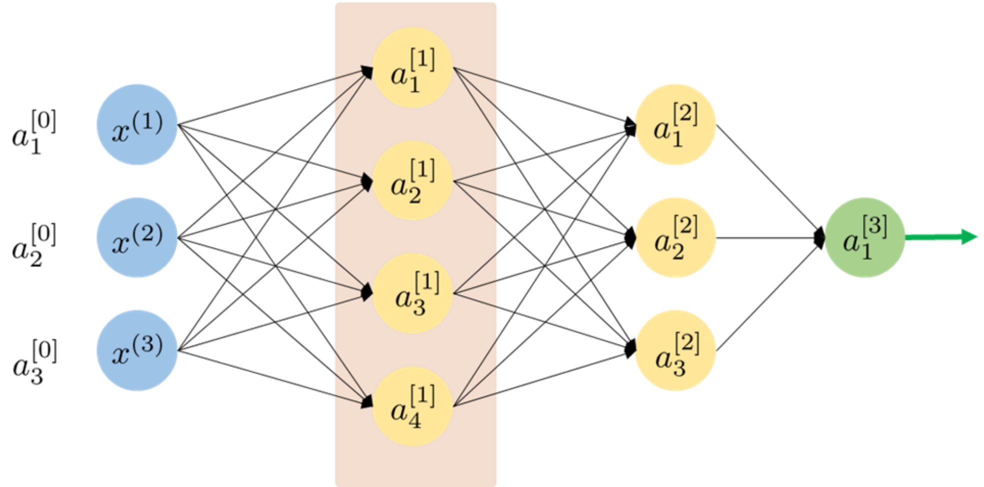
Neural Networks

Neural Networks – Forward Pass – Incorporating all Inputs:

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

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

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

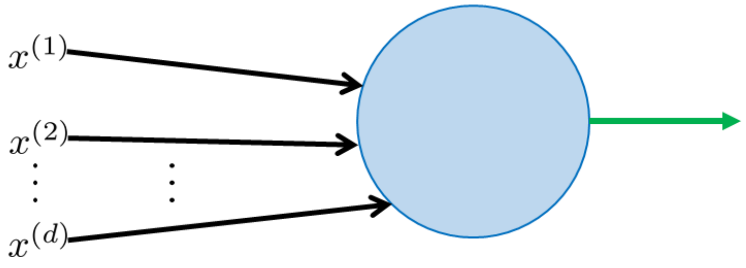


$$\mathbf{A}^{[1]} = g(\mathbf{Z}^{[1]}), \quad \mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]}$$

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

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

Recall:



$g(\mathbf{w}^T\mathbf{x} + b)$

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

$g(z)$

- 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:

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

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

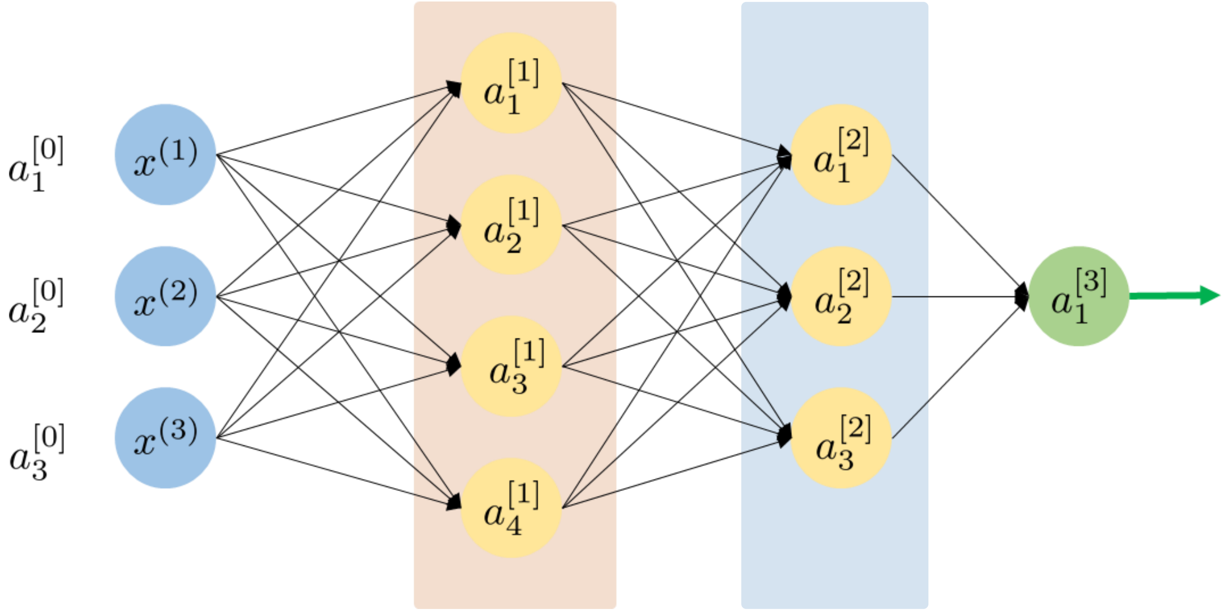
$$(4 \times n) = (4 \times 3)(3 \times n) + (4 \times n)$$

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

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

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

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



- In general, we have

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

Outline

- Perceptron and Perceptron Classifier
- Logistic Regression Classifier
- Neural Networks
  - Neural networks connection with perceptron and logistic regression
- Neural networks 'Forward Pass'
- Learning neural network parameters
  - Back Propagation.

Neural Networks

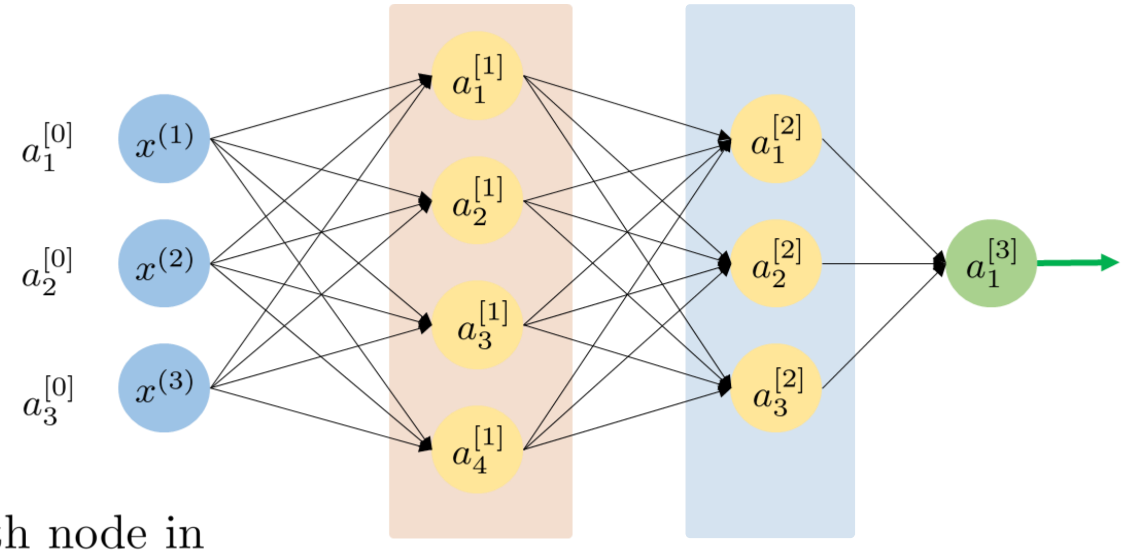
Learning Weights:

- 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 ℓ-th layer, $m^{[\ell]}$
- $a_i^{[\ell]}$ denotes the output of i-th node in the ℓ-th layer.
- $\mathbf{a}^{[\ell]}$ output of ℓ-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 ℓ-th layer respectively.
- $w_{i,j}^{[\ell]}$ denotes the weight associated with the j-th input of the i-th node in the ℓ-th layer.

Example:



$\mathbf{W}^{[1]}$ $\mathbf{W}^{[2]}$ $\mathbf{W}^{[3]}$

$\mathbf{b}^{[1]}$ $\mathbf{b}^{[2]}$ $\mathbf{b}^{[3]}$

Parameters we need to learn!

Neural Networks

Learning Weights:

- We assume we have training data D given by

$$D = \{(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)\} \subseteq \mathcal{X}^d \times \mathcal{Y}$$

- 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):

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

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

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

- We can use gradient descent to learn the weight matrices and bias vectors.

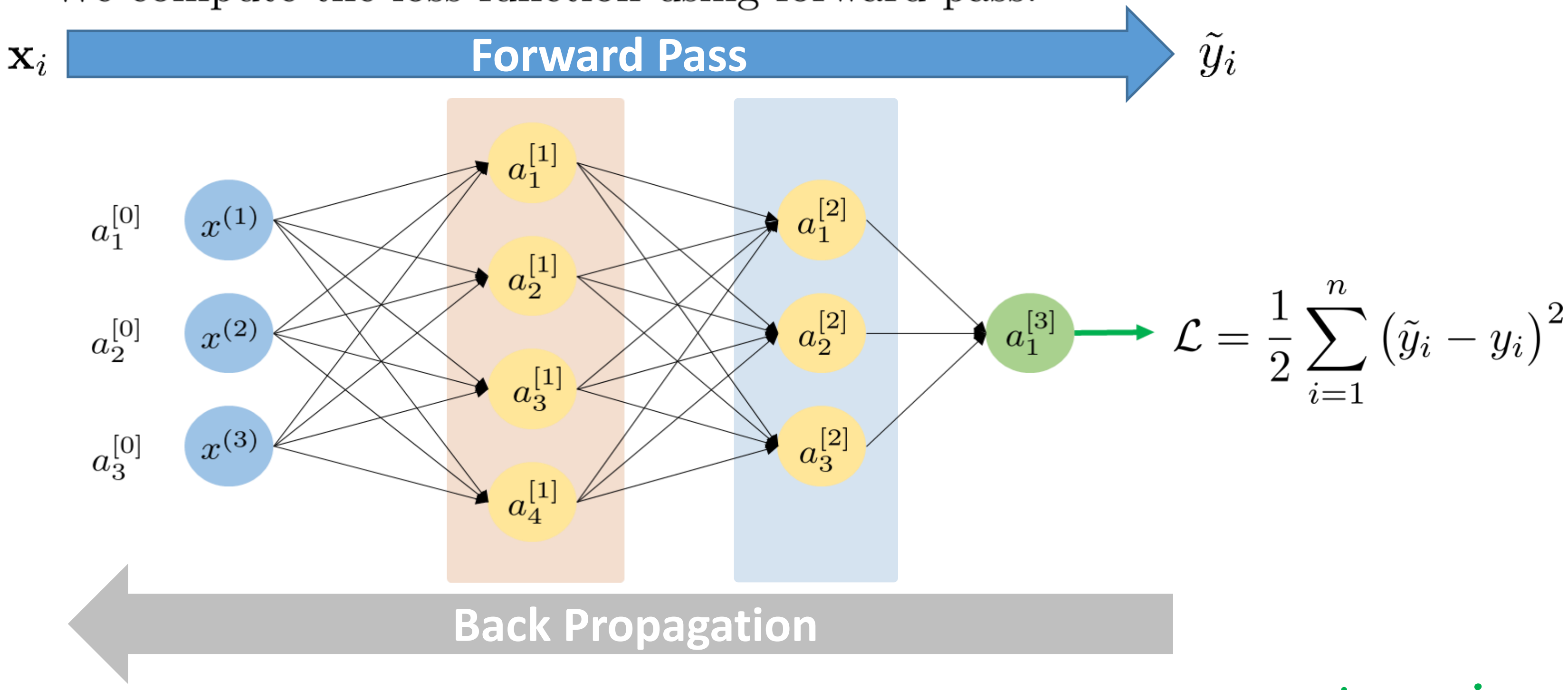
- Given our prior knowledge, output y is a composite function of input $\mathbf{x}$. Therefore, it is continuous and differentiable and we can use chain rule to compute the gradient.

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

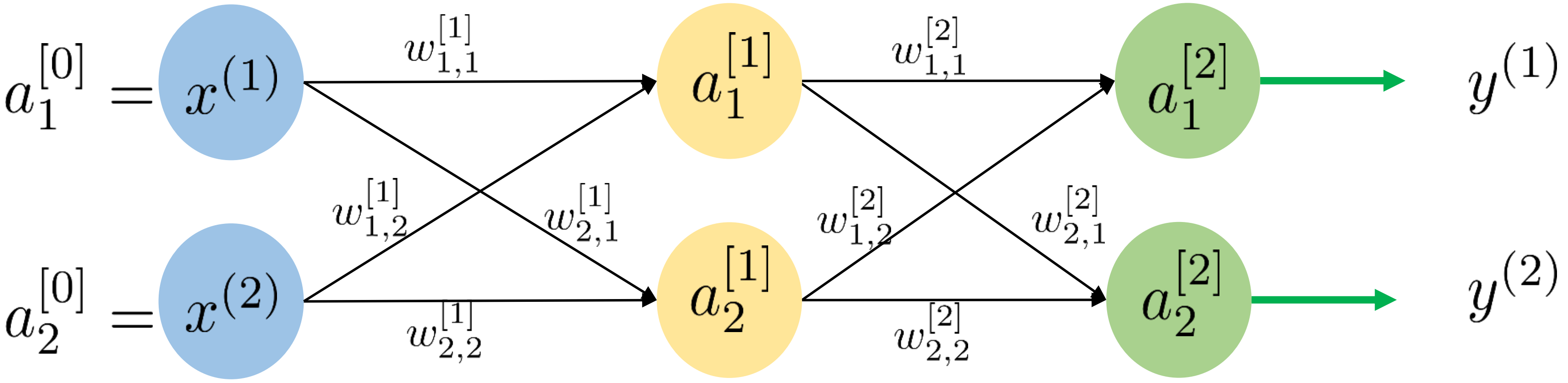
Learning problem reduces to the question of calculating gradient (partial derivatives) of loss function.

- 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

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

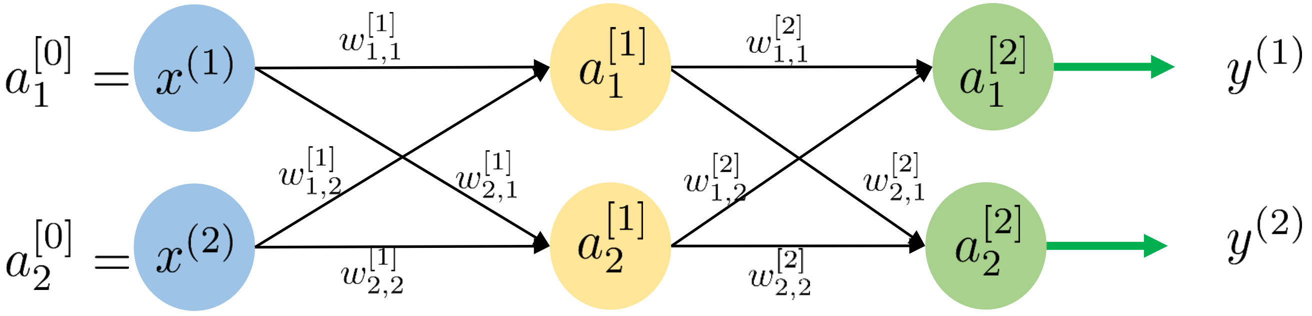
- Initial values of weights and biases:

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

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

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

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

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

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

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

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

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

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

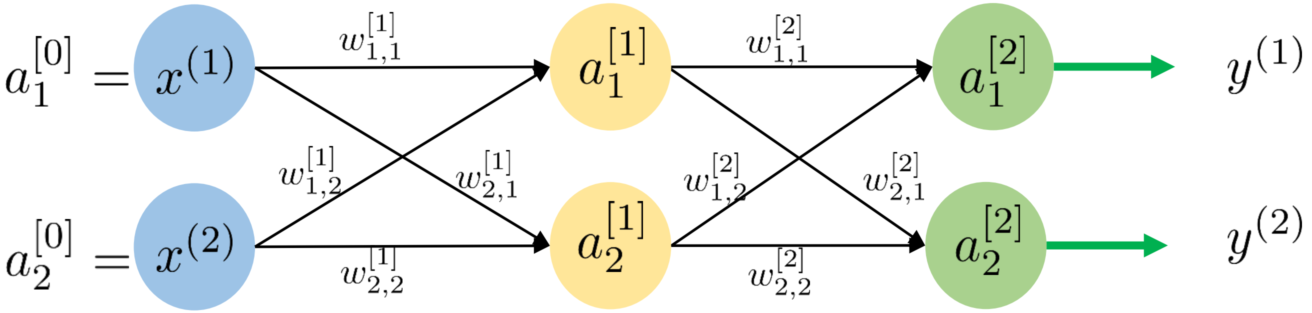
$$z_2^{[2]} = \mathbf{w}_2^{[2]^T}\mathbf{x} + b_2^{[2]} = 1.225, \quad a_2^{[2]} = g(1.225) = 0.7729 = \tilde{y}^{(2)}$$

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 $\dfrac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}}$
- Traverse the path from the loss function back to the weight $w_{1,1}^{[2]}$:

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

$$\tilde{y}^{(1)} = \sigma(z_1^{[2]})$$

$$z_1^{[2]} = w_{1,1}^{[2]} a_1^{[1]} + w_{1,2}^{[2]} a_2^{[1]} + b_1^{[2]}$$

$$\frac{\partial \mathcal{L}}{\partial w_{1,1}^{[2]}} = \frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} \frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} \frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = 0.0821$$

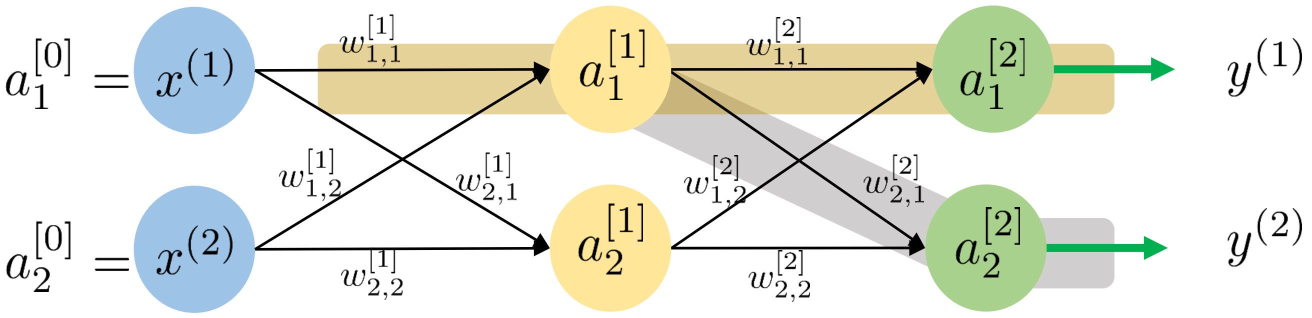
$$\frac{\partial \mathcal{L}}{\partial \tilde{y}^{(1)}} = \tilde{y}^{(1)} - y^{(1)} = 0.7414$$

$$\frac{\partial \tilde{y}^{(1)}}{\partial z_1^{[2]}} = \sigma(\partial z_1^{[2]})\left(1 - \sigma(\partial z_1^{[2]})\right) = 0.1868$$

$$\frac{\partial z_1^{[2]}}{\partial w_{1,1}^{[2]}} = a_1^{[a]} = 0.5933$$

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 $\dfrac{\partial \mathcal{L}}{\partial w_{1,1}^{[1]}}$
- 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.