CSE 4553
Machine Learning
Lecture 7: Introduction to Neural Network

Winter 2024

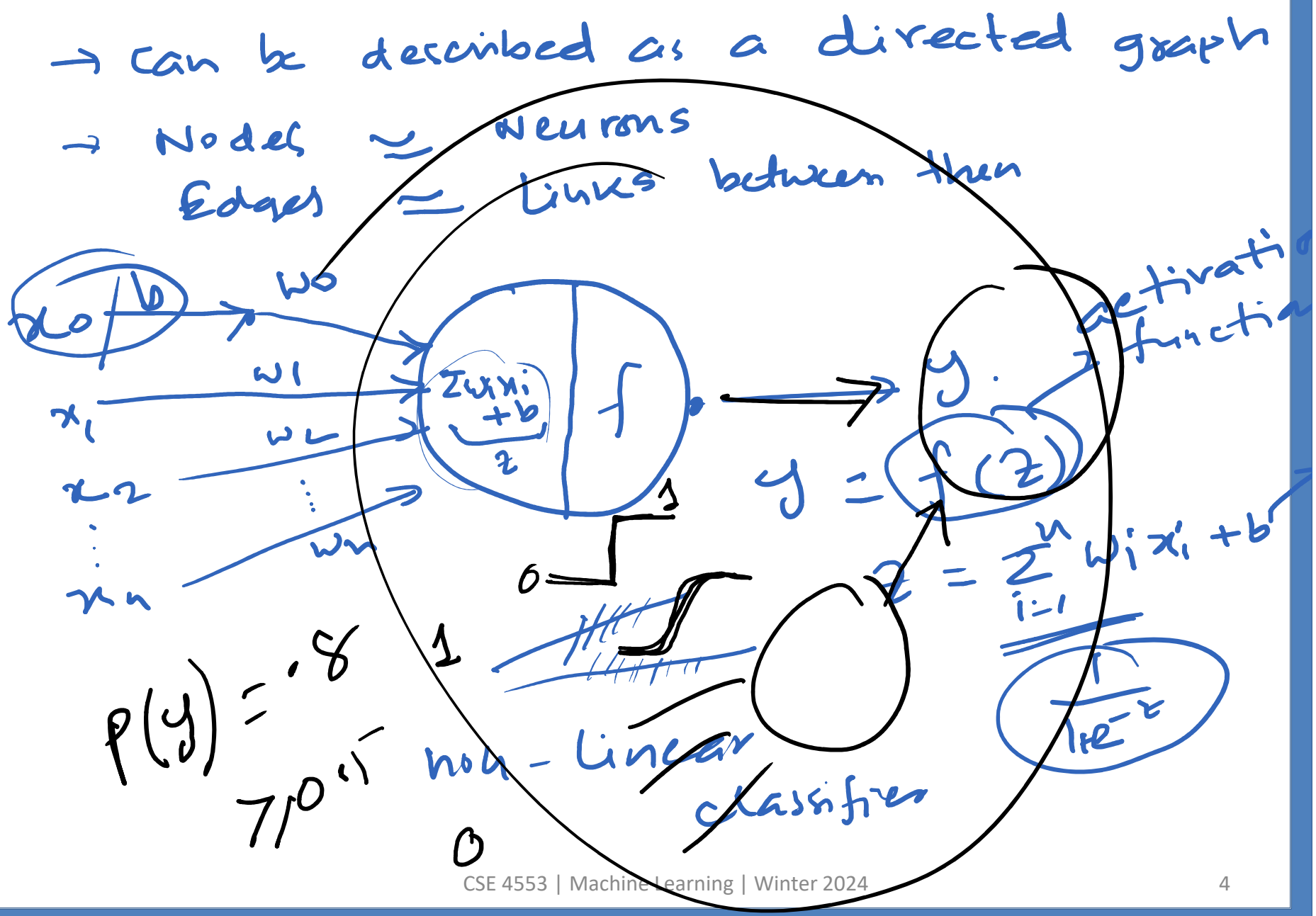
Hasan Mahmud | hasan@iut-dhaka.edu

Contents

- Neural Network basics
- Activation functions
- Types of ANN

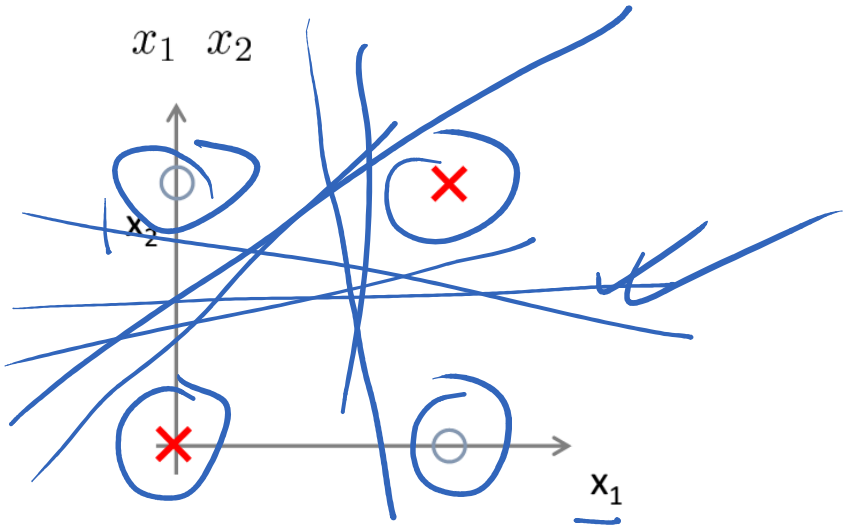
Introduction

- One of the main weakness of linear models is that they are linear!
- Decision trees and kNN classifiers can model non-linear boundaries!
- Neural networks are yet another non-linear classifier!
- Take the biological inspiration further by chaining together perceptrons
- Here are the three reasons you should study neural computation:
  - To understand how the brain actually works: it's very big and very complicated and made of stuff that dies when you poke it, so we need to use computer simulations.
  - To understand a style of parallel computation inspired by neurons and their adaptive connections: it's a very different style from sequential computation.
  - To solve practical problems by using novel learning algorithms inspired by the brain: learning algorithms can be very useful even if they are not how the brain actually works.

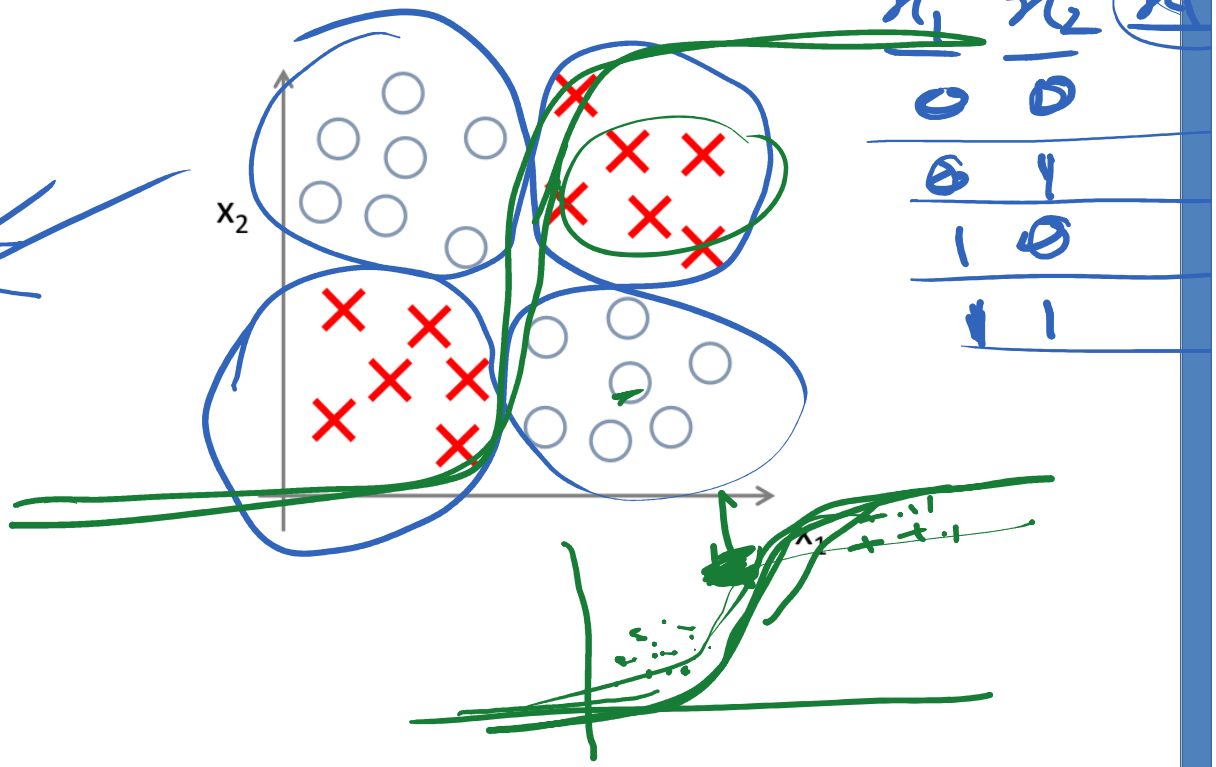

→ can be described as a directed graph
→ Nodes ≃ Neurons
Edges ≃ Links between them
x0 / b
w0
w1
wn
x1
x2
xn
Σwixi +b
z
f
y
activation function
y = f(z)
z = Σ wixi +b
i=1
n
1
0
p(y) = .8
1 if
0
non-linear classifier
1/(1+e^-z)

Non-linear problems

Simple Non-Linear Classification Example

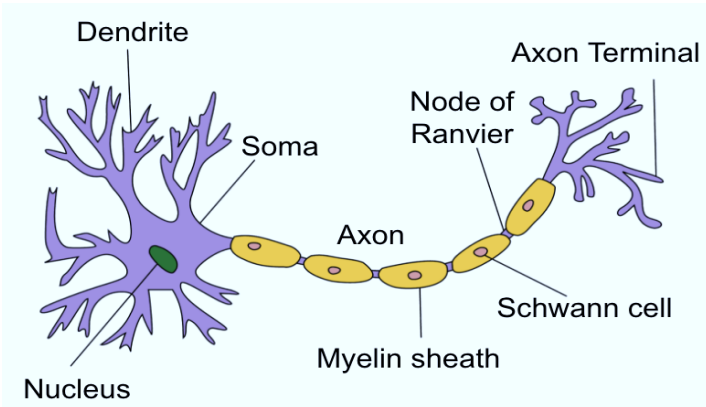


$y = x_1 \text{ XOR } x_2$

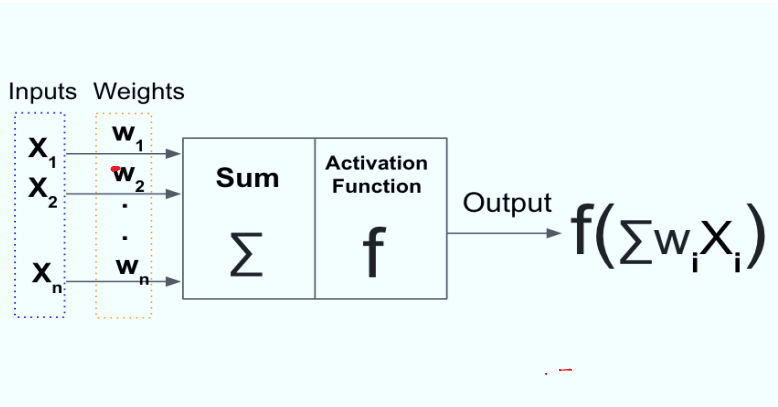


Biological neuron VS. Artificial neuron

- Biological Neurons are the core components of the human brain. A neuron consists of a cell body, dendrites, and an axon. It processes and transmit information to other neurons by emitting electrical signals. Each neuron receives input signals from its dendrites and produces output signals along its axon. The axon branches out and connects via synapses to dendrites of other neurons.
- A basic model for how the neurons work goes as follows: Each synapse has a strength that is learnable and control the strength of influence of one neuron on another. The dendrites carry the signals to the target neuron's body where they get summed. If the final sum is above a certain threshold, the neuron get fired, sending a spike along its axon.



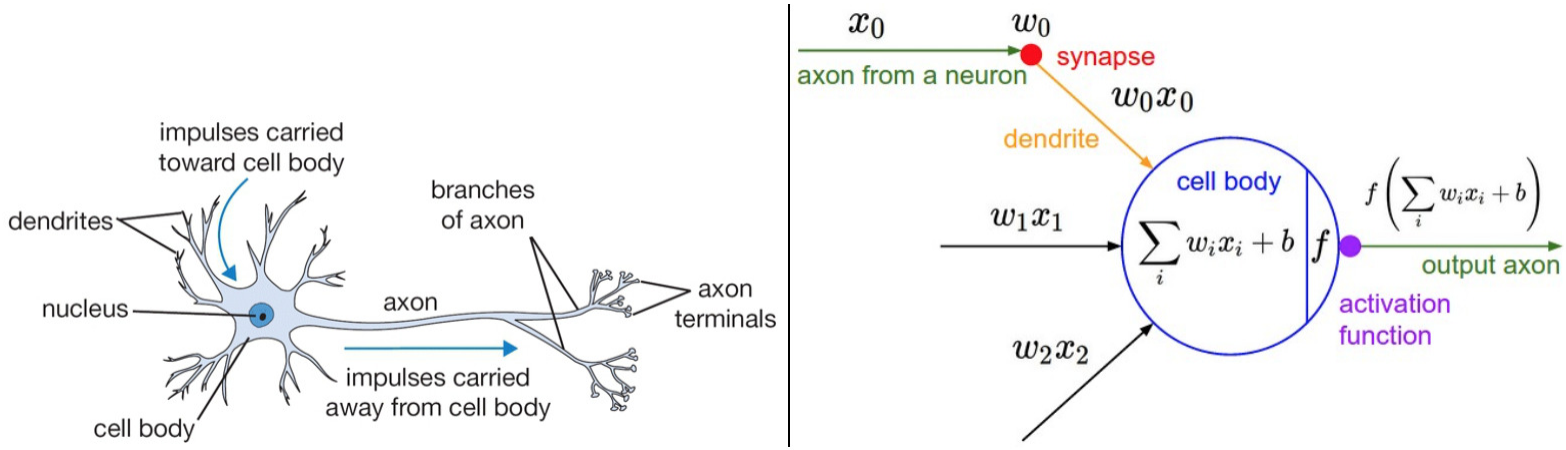
Structure of a typical neuron
(source: Wikipedia)



Structure of artificial neuron

Biological neuron VS. Artificial neuron

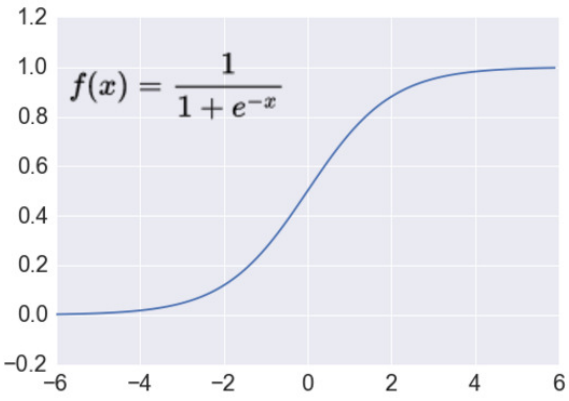
- Artificial neurons are inspired by biological neurons, and try to formulate the model in a computational form.
- An artificial neuron has a finite number of inputs with weights associated to them, and an activation function (also called transfer function).
- The output of the neuron is the result of the activation function applied to the weighted sum of inputs. Artificial neurons are connected with each others to form artificial neural networks.



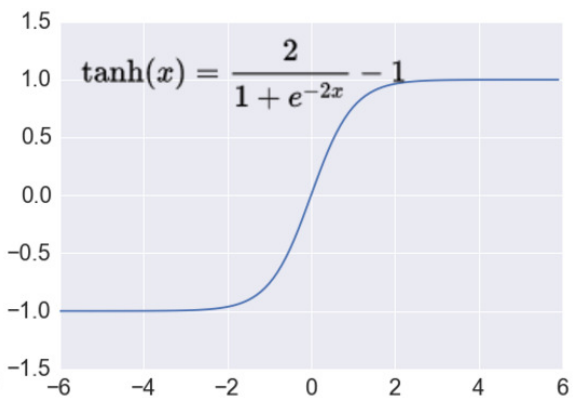
Activation function

- Activation functions transform the weighted sum of inputs that goes into the artificial neurons. These functions should be non-linear to encode complex patterns of the data. The most popular activation functions are Sigmoid, Tanh and ReLU.

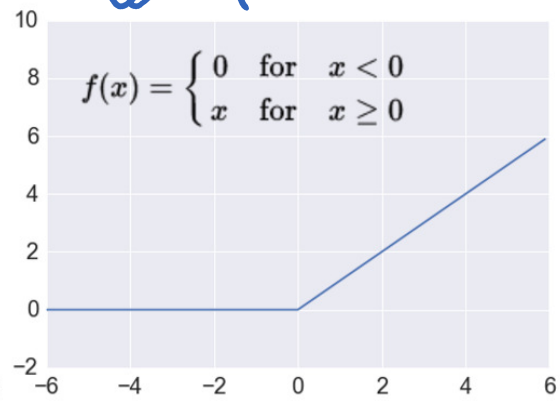
Sigmoid



TanH

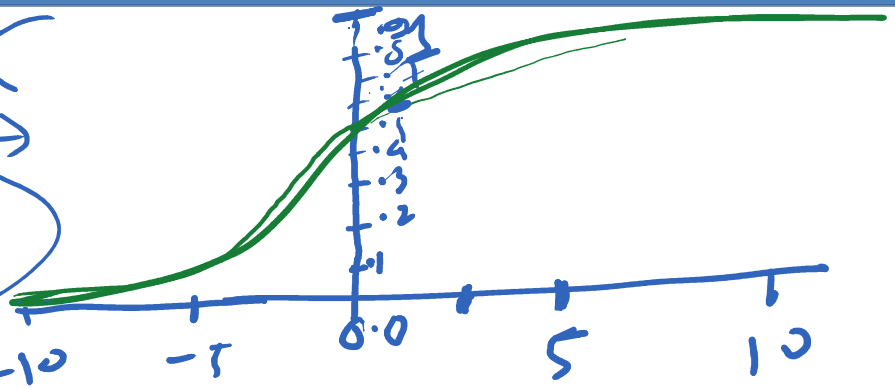


ReLU



Sigmoid

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



input = real-valued

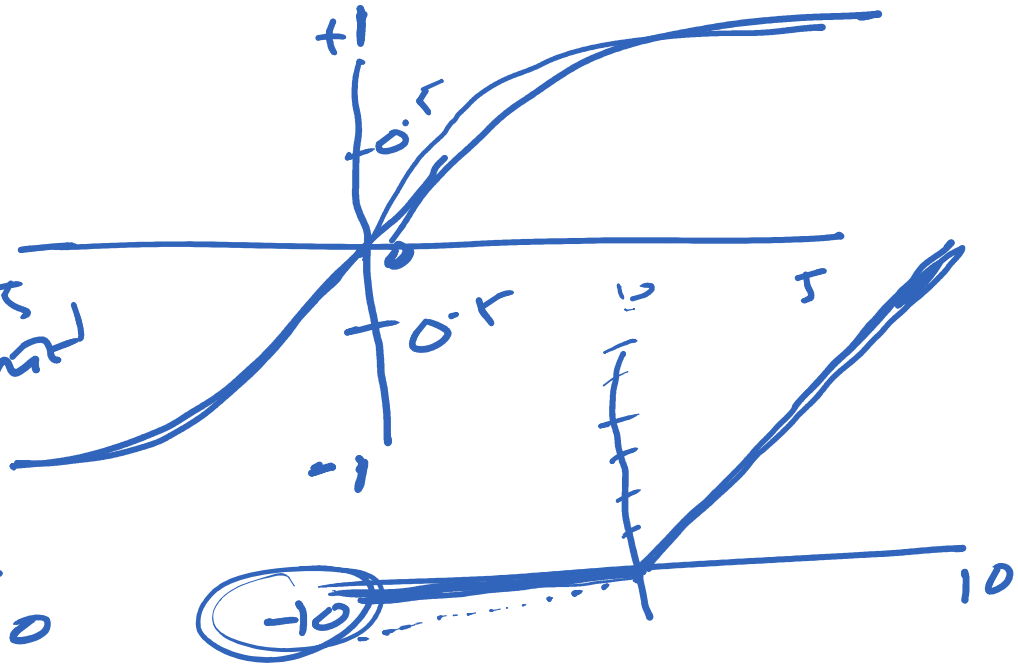
output = 0 to 1

Tanh

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

hyperbolic tangent

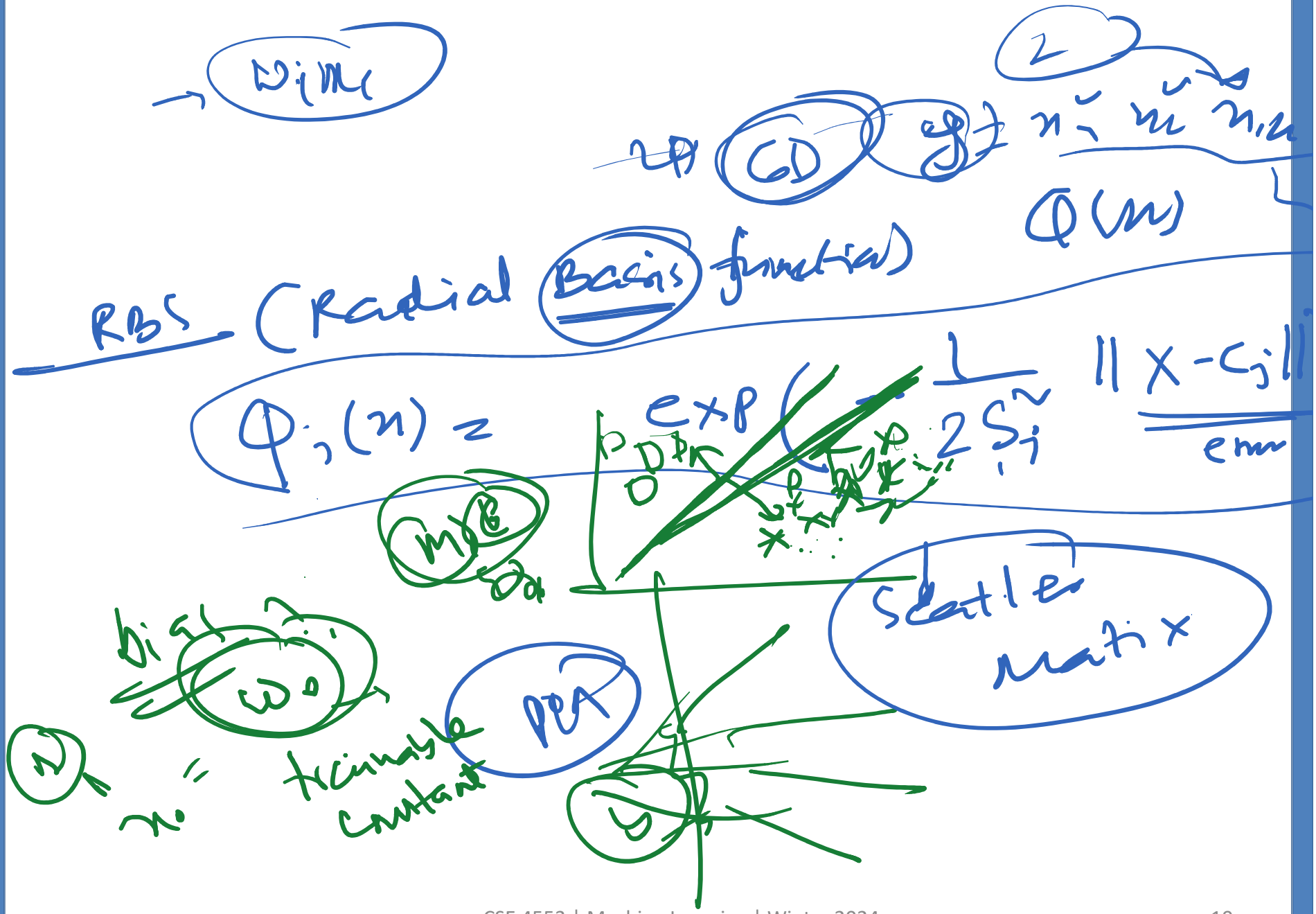
output = -1 to 1



ReLU (Rectified Linear Unit)

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

Replaced neg values by 0


RBS (Radial Basis functions)
exp
scatter matrix
PCA

Single Neuron Model

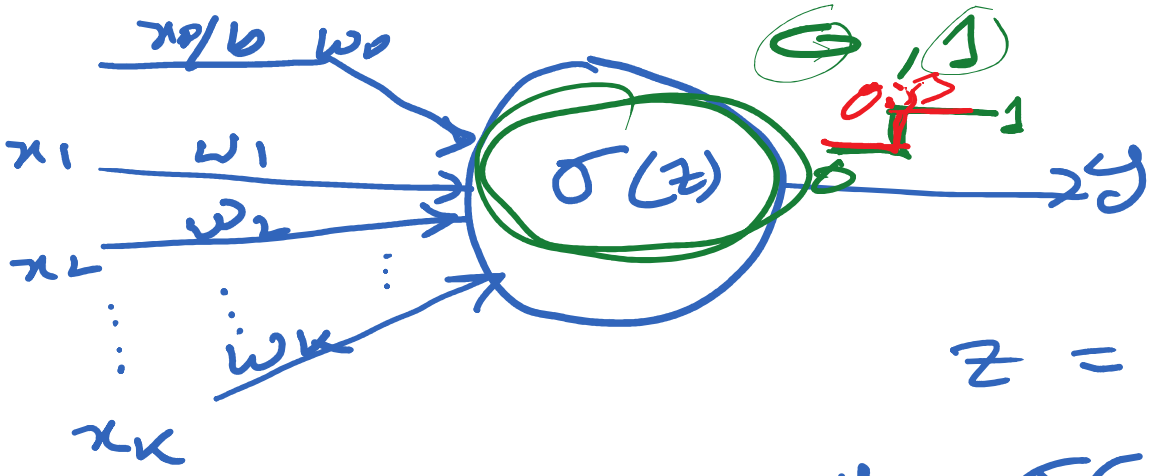
input: K; $x_1, x_2, x_3, \ldots, x_K$.

weights: $w_1, w_2, w_3, \ldots, w_K$.

bias: b

Activation function: $\sigma(z)$

output: y



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

$$\sigma(z) = \text{sign}(z)$$

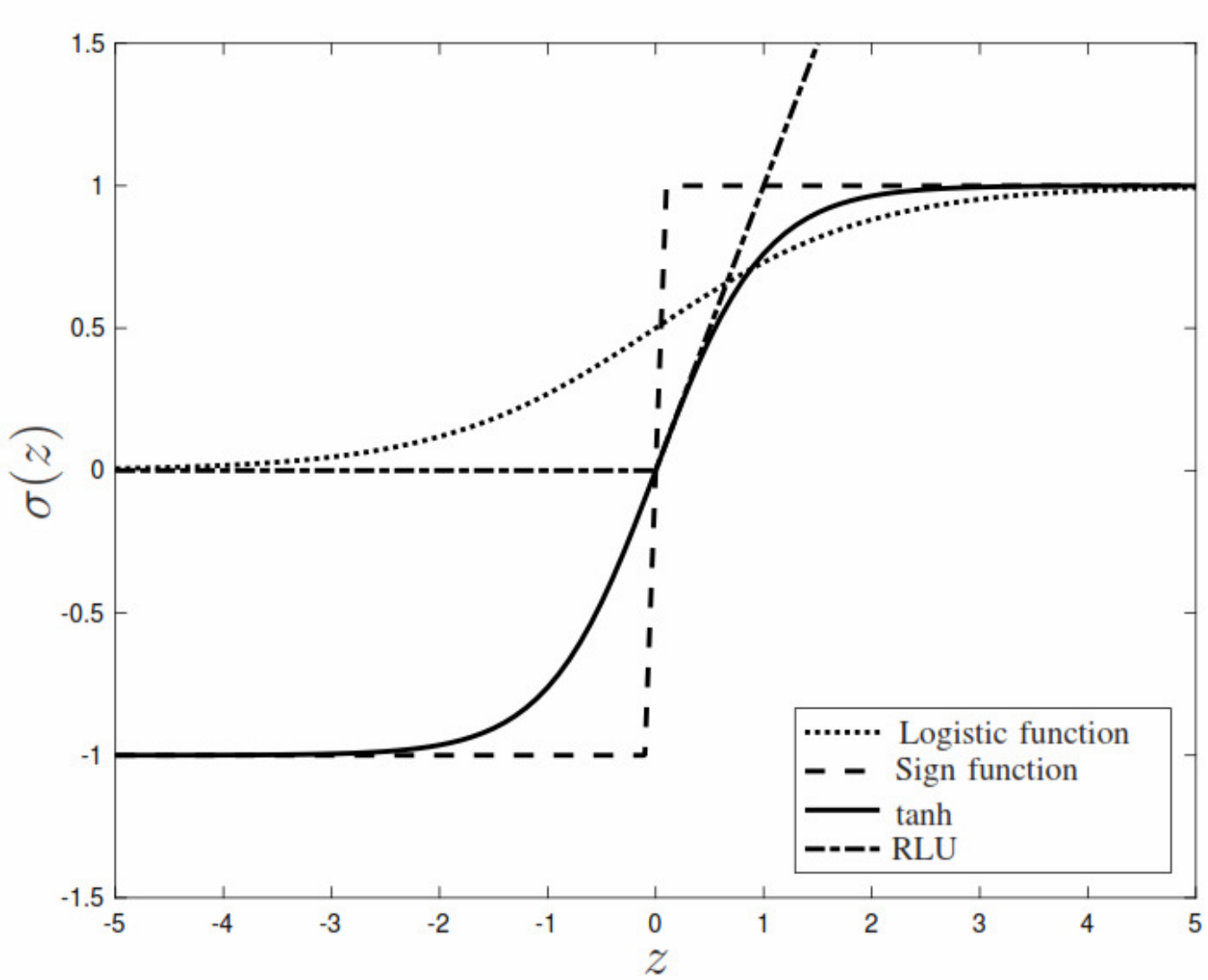
$$\sigma(z) = \tanh(z) = \frac{e}{e}$$

$$z = \sum_{i=1}^{K} w_i x_i + b \rightarrow (1)$$

$$y = \sigma(z) \rightarrow (2)$$

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

Different Activation Functions

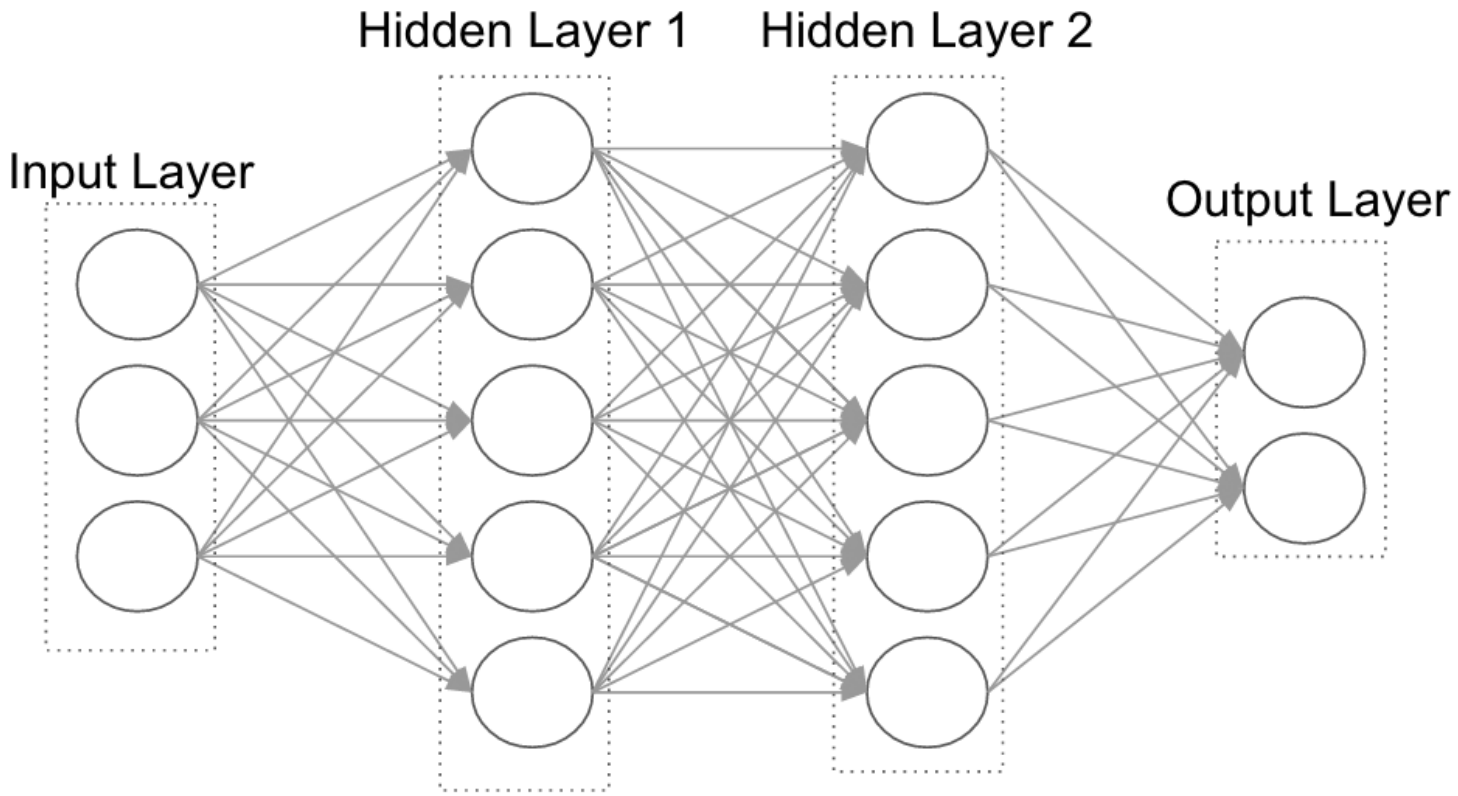


Types of Neural Network

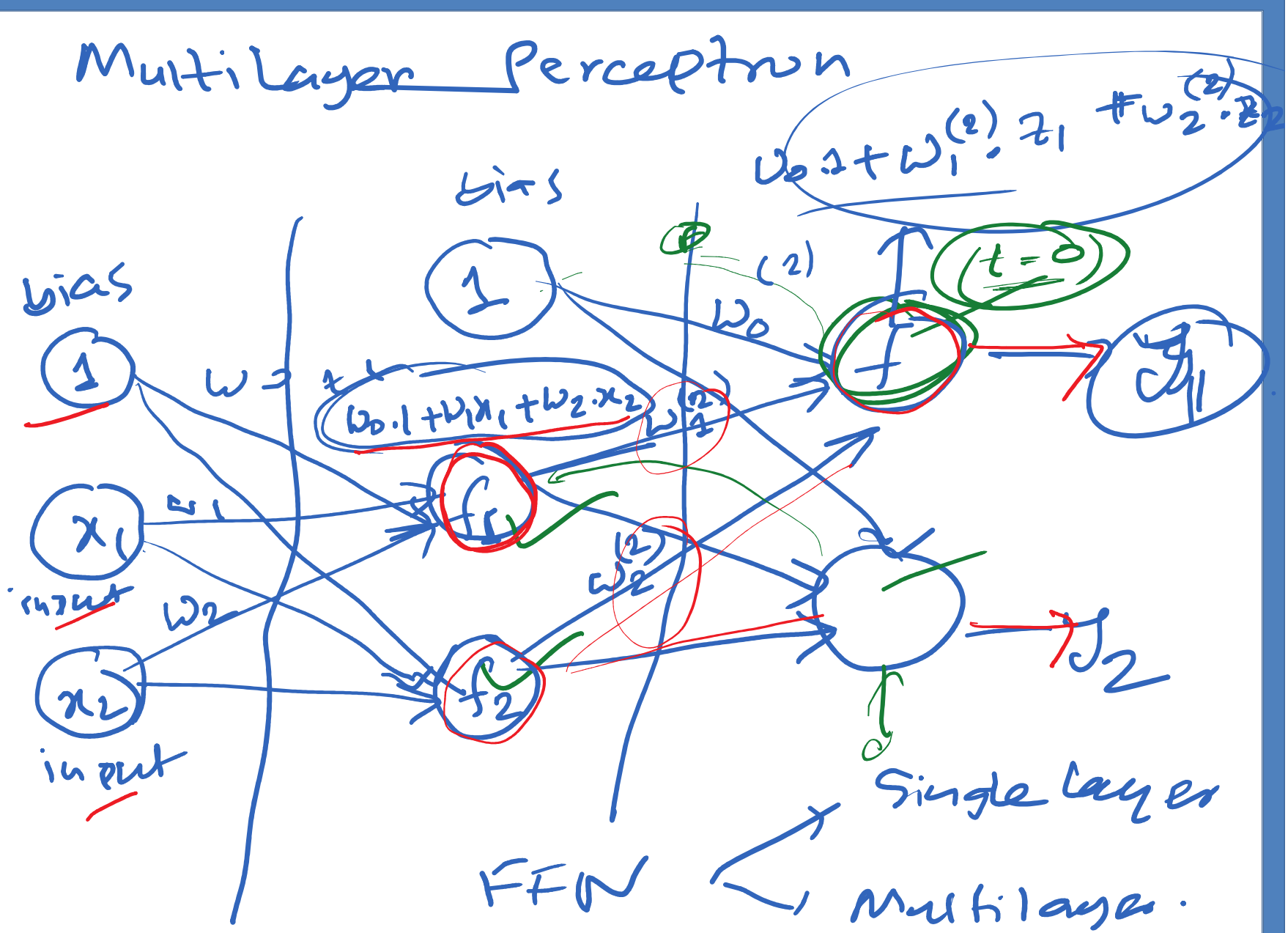
- Feed Forward Neural Network (FFN)
- Back Propagation Neural Network (BPN)

FFN

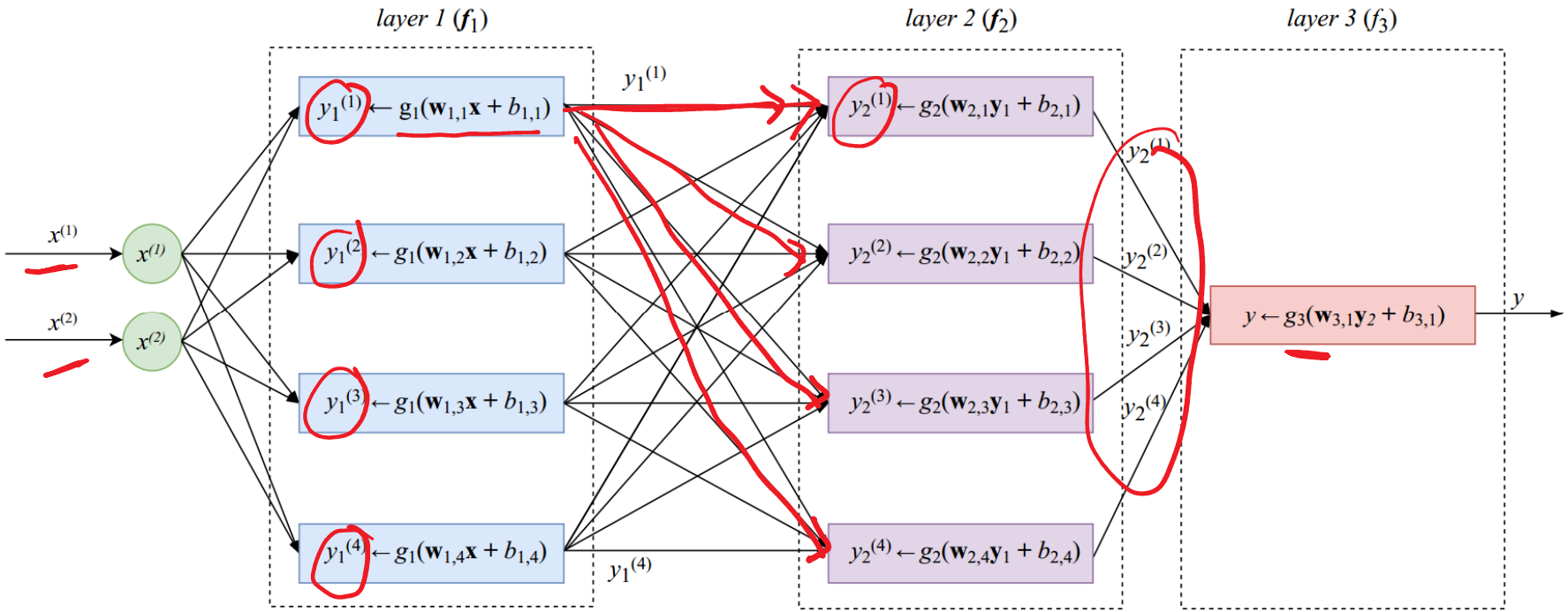
- Feedforward Neural Networks are the simplest form of Artificial Neural Networks.
- These networks have 3 types of layers: Input layer, hidden layer and output layer. In these networks, data moves from the input layer through the hidden nodes (if any) and to the output nodes.
- Next slide has an example of a fully-connected feedforward neural network with 2 hidden layers. "Fully-connected" means that each node is connected to all the nodes in the next layer.
- Note that, the number of hidden layers and their size are the only free parameters. The larger and deeper the hidden layers, the more complex patterns we can model in theory.



Feedforward neural network with 2 hidden layers

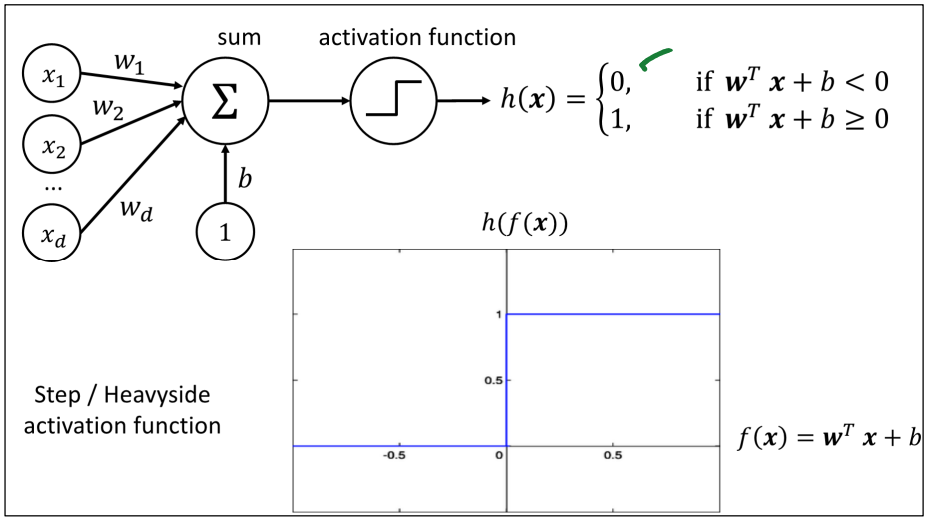

Multilayer Perceptron
bias
bias
input
input
Single layer
FFN
Multilayer.

Feed-Forward Neural Network Architecture

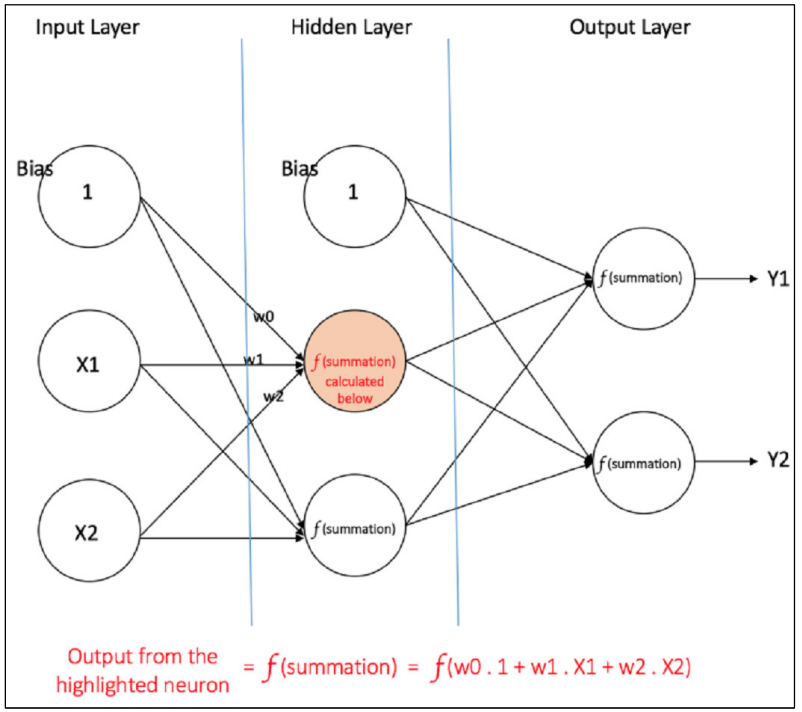


A multilayer perceptron with two-dimensional input, two layers with four units and one output layer with one unit

Types of FFNN



Single layer perceptron



Multi layer perceptron

$$\begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad [x_0\ x_1\ x_2\ x_3 \cdots x_d] \quad \sum_{i=0}^{d} w_i x_i$$

→ Weight learning ??

Single unit perceptron learning algorithm

GRADIENT-DESCENT($training\_examples, \eta$)

Initialize each w_i to some small random value

Until the termination condition is met, Do

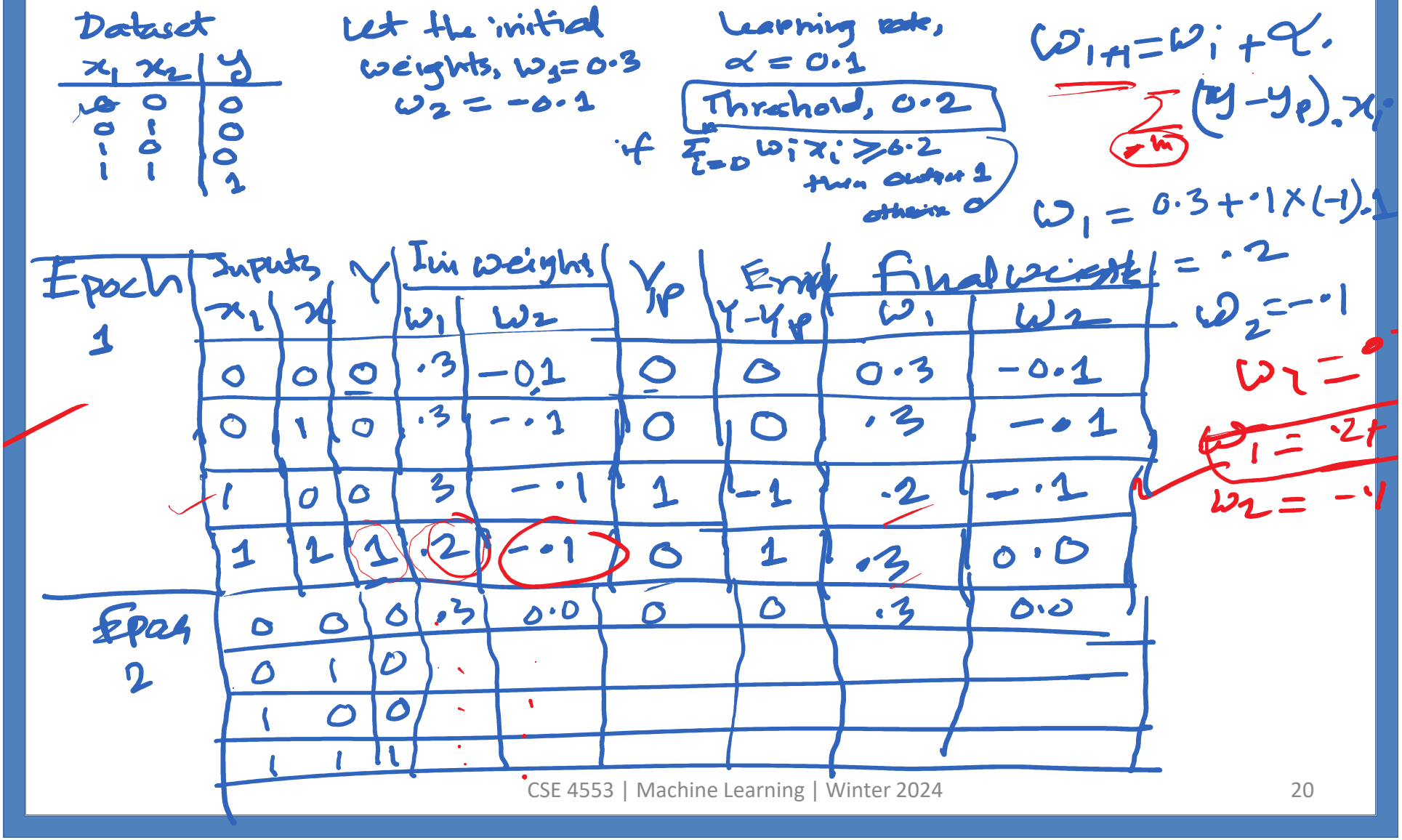
- Initialize each Δw_i to zero.
- For each $\langle \vec{x}, t \rangle$ in $training\_examples$, Do
  - Input instance $\vec{x}$ to unit and compute output o
  - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

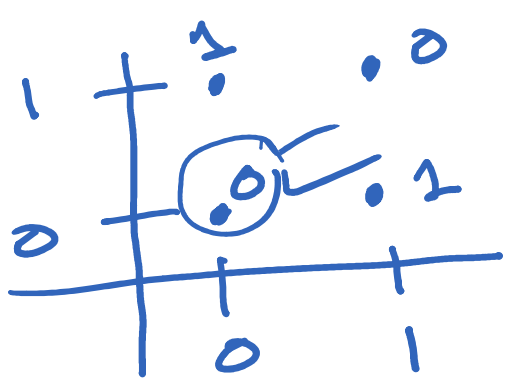
- For each linear unit weight w_i, Do

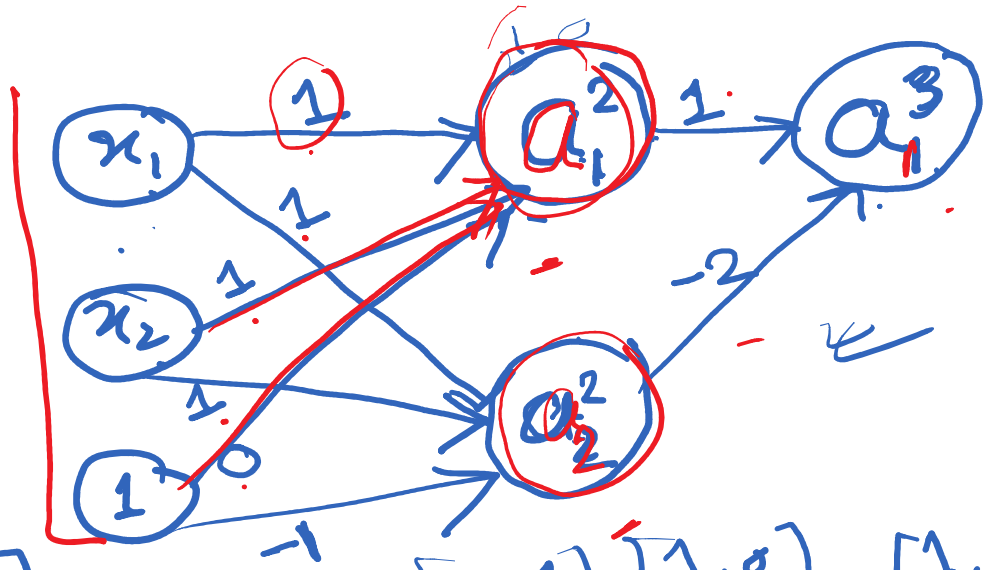
$$w_i \leftarrow w_i + \Delta w_i$$

Single unit perceptron learning algorithm – Numerical Example



XOR problem using FFNN





$[x_1, x_2] = [0, 0]$, ReLU $[0,1], [1,0], [1,1]$?

00

$a_1^2 = \max(0, 1*x_1 + 1*x_2 + 0*1) = 0$ ✓

01, 10

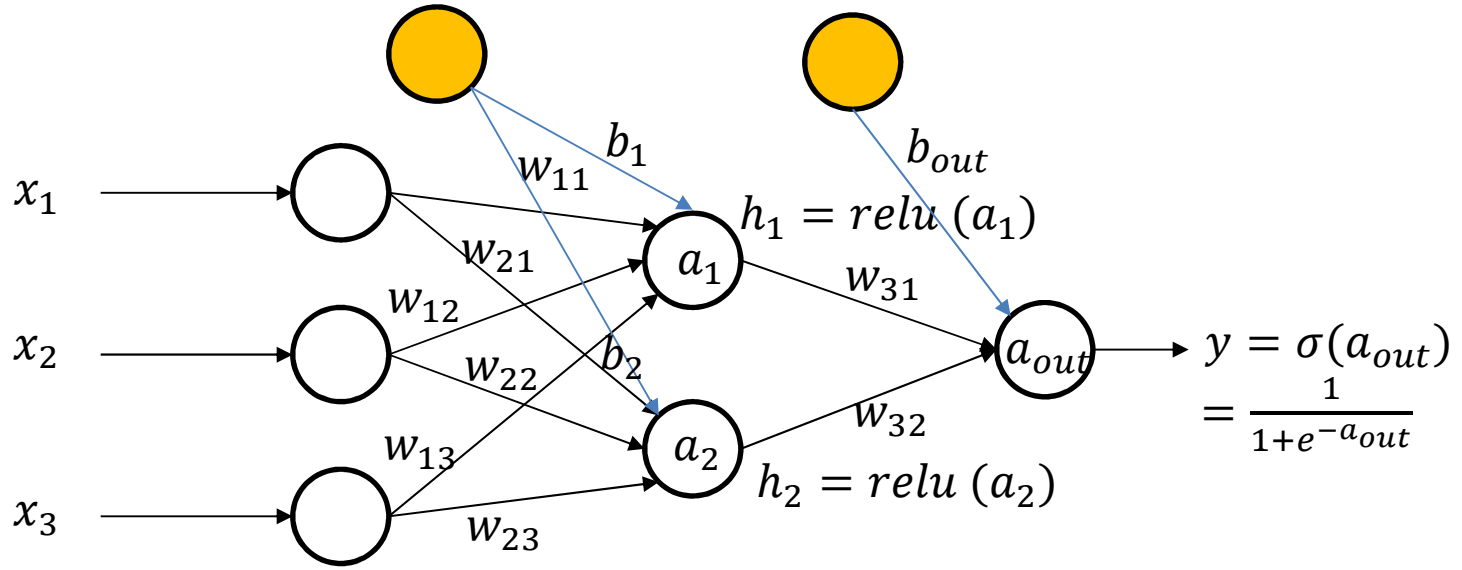
$a_2^2 = \max(0, 1*x_1 + 1*x_2 + -1*1) = 0$

11

$a_1^3 = \max(0, 1*a_1^2 - 2*a_2^2) = 0$ ✓

Cost function, $C(W, b) = \frac{1}{2N} \sum_{i=1}^{k} (a(x_i) - y_i)$

Forward pass



$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$w_{hidden} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{21} & w_{23} \end{bmatrix} \quad w_{out} = [w_{31} \quad w_{32}]$$

$$b_{hidden} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

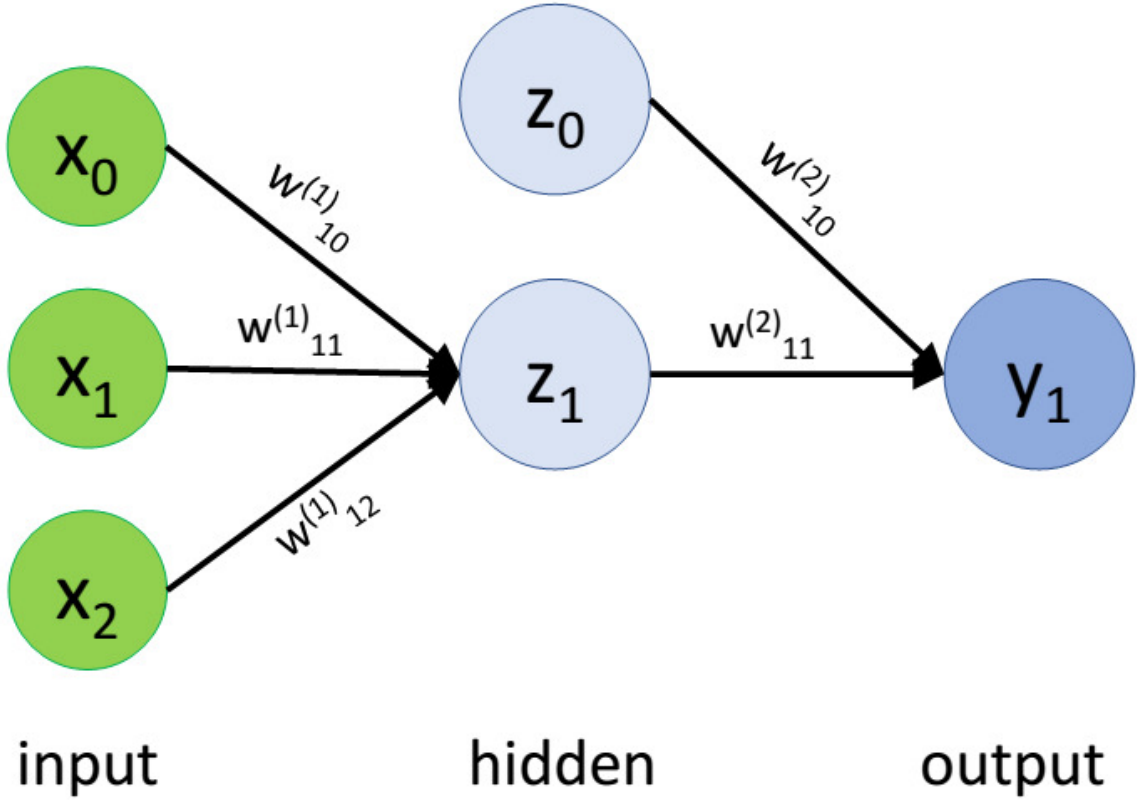
BPN

- Backward Propagation of Errors, often abbreviated as *BackProp* is one of the several ways in which an artificial neural network (ANN) can be trained. It is a supervised training scheme, which means, it learns from labeled training data.
- *BackProp* is like "learning from mistakes". The supervisor corrects the ANN whenever it makes mistakes.
- An ANN consists of nodes in different layers; input layer, intermediate hidden layer(s) and the output layer. The connections between nodes of adjacent layers have "weights" associated with them.
- The goal of learning is to assign correct weights for these edges. Given an input vector, these weights determine what the output vector is.
- BackProp Algorithm:
  - Initially all the edge weights are randomly assigned. For every input in the training dataset, the ANN is activated and its output is observed.
  - This output is compared with the desired output that we already know, and the error is "propagated" back to the previous layer.
  - This error is noted and the weights are "adjusted" accordingly.
  - This process is repeated until the output error is below a predetermined threshold.

Backward pass

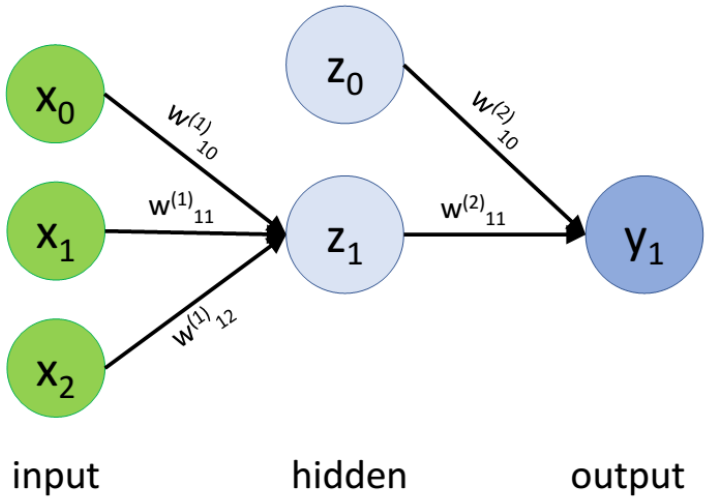
Example

First architecture



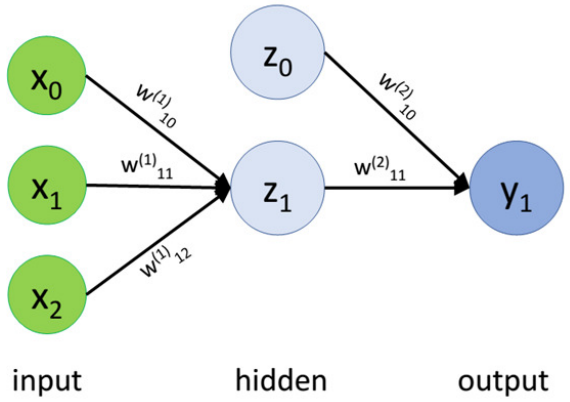
Computing activations

- In all examples, x = [x1 x2], plus include 1 for bias
- Assume sigmoid activation function
- Initialize all weights to 0.1
- First example: x = [1 0]
- Second example: x = [0 1]
- Third example: x = [1 1]



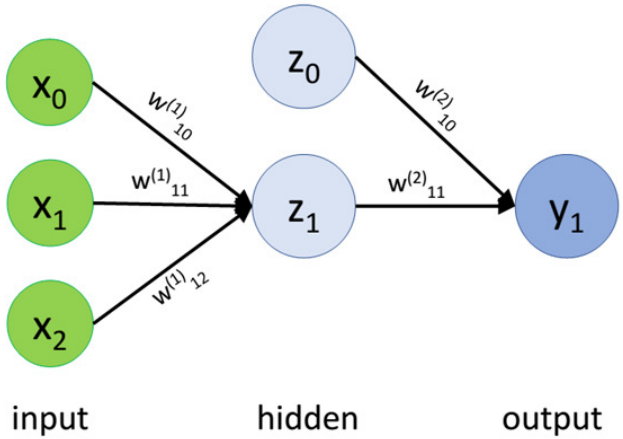
Computing activations (answers)

- First example:
  - At hidden: $z_1 = 1 / [1 + \exp(-(x_0 * w^{(1)}_{10} + x_1 * w^{(1)}_{11} + x_2 * w^{(1)}_{12}))]$
  - $= 1 / [1 + \exp(-(1*0.1+1*0.1+0*0.1))] = 0.5498$
  - At output: $y_1 = 1 / [1 + \exp(-(z_0 * w^{(2)}_{10} + z_1 * w^{(2)}_{11}))]$
  - $= 1 / [1 + \exp(-(1*0.1+0.5498*0.1))] = 0.5387 \rightarrow y_{pred} = 1$
- Second example:
  - At hidden: $z_1 = 1 / [1 + \exp(-(x_0 * w^{(1)}_{10} + x_1 * w^{(1)}_{11} + x_2 * w^{(1)}_{12}))]$
  - $= 1 / [1 + \exp(-(1*0.1+0*0.1+1*0.1))] = 0.5498$
  - At output: $y_1 = 1 / [1 + \exp(-(z_0 * w^{(2)}_{10} + z_1 * w^{(2)}_{11}))]$
  - $= 1 / [1 + \exp(-(1*0.1+0.5498*0.1))] = 0.5387 \rightarrow y_{pred} = 1$



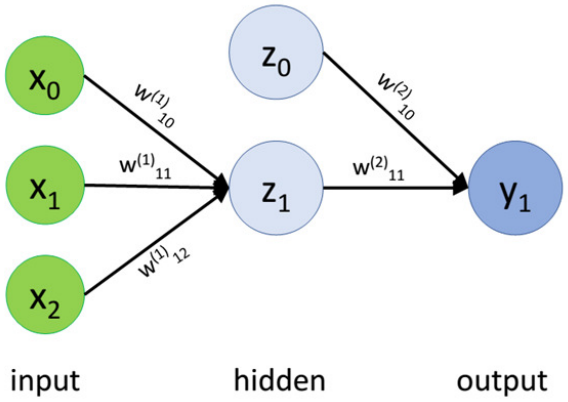
Computing activations (answers)

- Third example:
  - At hidden: $z_1 = 1 / [1 + \exp(-(x_0 * w^{(1)}_{10} + x_1 * w^{(1)}_{11} + x_2 * w^{(1)}_{12}))]$
  - $= 1 / [1 + \exp(-(1*0.1+1*0.1+1*0.1))] = 0.5744$
  - At output: $y_1 = 1 / [1 + \exp(-(z_0 * w^{(2)}_{10} + z_1 * w^{(2)}_{11}))]$
  - $= 1 / [1 + \exp(-(1*0.1+0.5744*0.1))] = 0.5393 \rightarrow y_{pred} = 1$



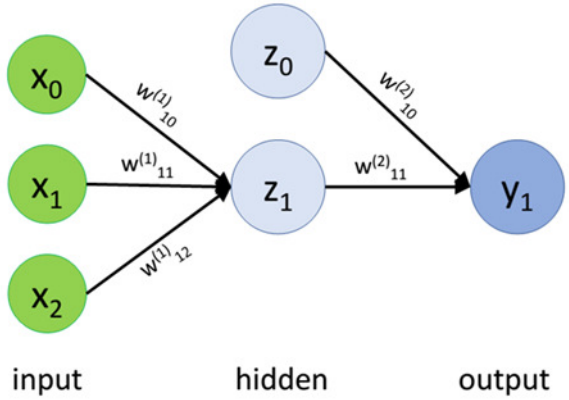
Training the first network

- Perform backpropagation using stochastic gradient descent (one sample at a time)
- Weights are initially all 0.1
- Learning rate is 0.3
- Sigmoid activation function at hidden and output
- d s(x) / dx = s(x) (1 – s(x)) dx
- Samples have the following labels:
  - First example: x = [1 0], y = 1
  - Second example: x = [0 1], y = 0
  - Third example: x = [1 1], y = 1
- Preview: What do you expect final weights to be?



Learning from first example

- First example: x = [1 0], y = 1
- Weights are $w^{(1)}_{10} = w^{(1)}_{11} = w^{(1)}_{12} = w^{(2)}_{10} = w^{(2)}_{11} = 0.1$
- Activations are $z_1 = 0.5498$, $y_1 = 0.5387$
- Compute errors:
  - $\delta_{y1} = y_1*(1-y_1)*(y_1-y_{true}) = 0.5387*(1-0.5387)*(0.5387-1) = -0.1146$
  - $\delta_{z1} = z_1*(1-z_1)*(w^{(2)}_{11}*\delta_{y1}) = 0.5498*(1-0.5498)*[0.1*-0.1146] = -0.0028$
- Update weights:
  - $w^{(2)}_{10} = w^{(2)}_{10} - 0.3*\delta_{y1}*z_0 = 0.1 + 0.3*0.1146*1 = 0.1343$
  - $w^{(2)}_{11} = w^{(2)}_{11} - 0.3*\delta_{y1}*z_1 = 0.1 + 0.3*0.1146*0.5498 = 0.1189$
  - $w^{(1)}_{10} = w^{(1)}_{10} - 0.3*\delta_{z1}*x_0 = 0.1 + 0.3*0.0028*1 = 0.1008$
  - $w^{(1)}_{11} = w^{(1)}_{11} - 0.3*\delta_{z1}*x_1 = 0.1 + 0.3*0.0028*1 = 0.1008$
  - $w^{(1)}_{12} = w^{(1)}_{12} - 0.3*\delta_{z1}*x_2 = 0.1 + 0.3*0.0028*0 = 0.1$



Limitations of Neural Networks

- Need substantial number of training samples
- Slow learning (convergence times)
- Inadequate parameter selection techniques that lead to poor minima
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image