Large Language Models (LLMs) Bootcamp

Deep Learning & Backpropagation

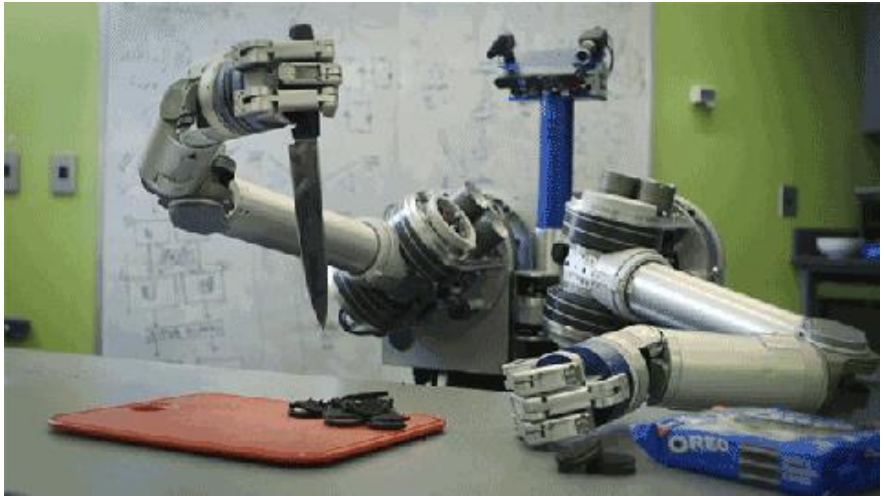
OUTLINE

- Introduction to Deep Learning
- Applications of Deep Learning
- Neural networks
  - Activation Functions
  - Feed Forward Neural Networks
  - Neural Network Hyperparameters
- How do Neural Networks Learn?

Introduction to Deep Learning

Machine Learning

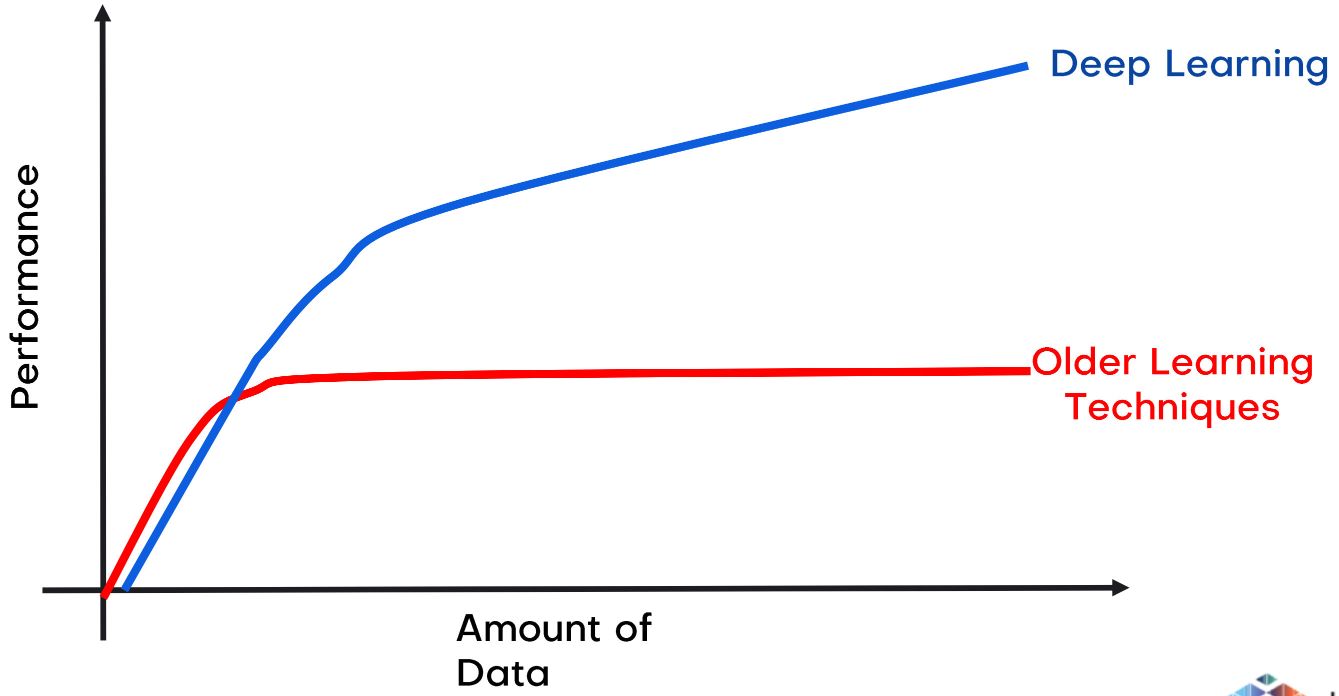
Machine Learning is a type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed.



Deep Learning

- Part of the machine learning field
- Exceptionally effective at identifying and learning patterns.
- Utilizes learning algorithms that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of our brain.

Comparison



Performance
Deep Learning
Older Learning Techniques
Amount of Data

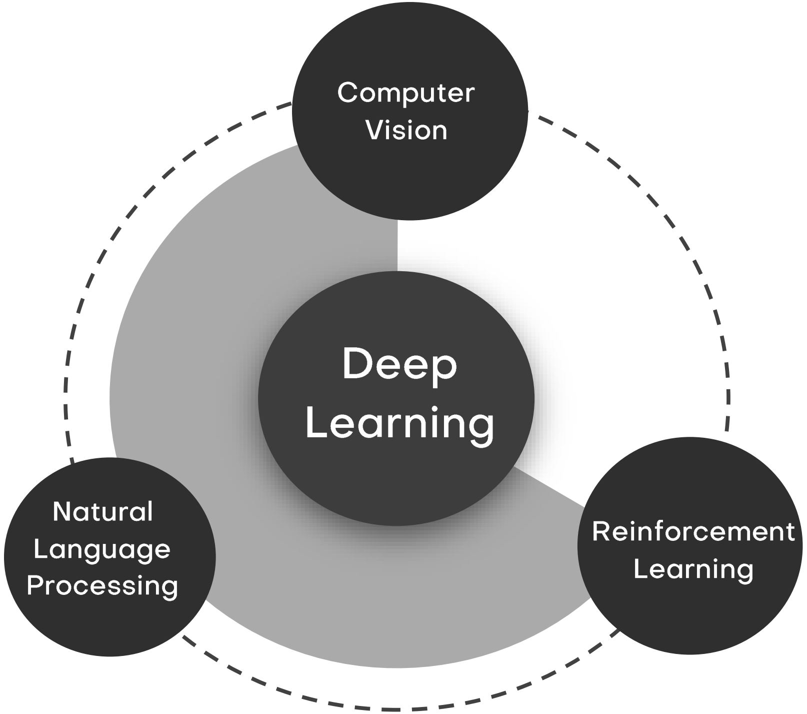
Applications of Deep Learning

Applications of Deep Learning

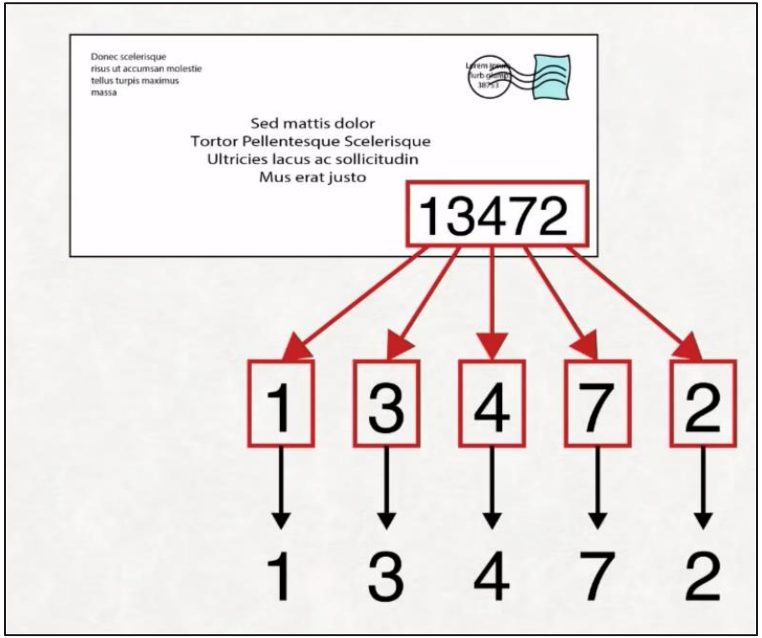
- Computer Vision: image processing, face recognition, object detection, video analytics

- Deep Reinforcement Learning

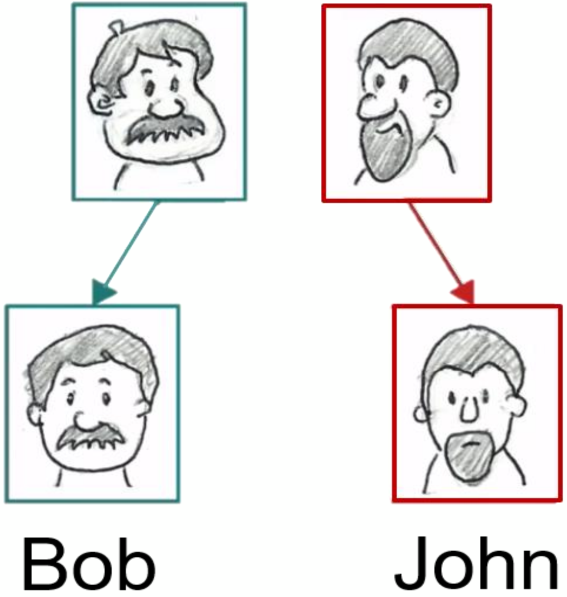
- Natural Language Processing: sentiment analysis, speech recognition, text-to-speech, conversational AI



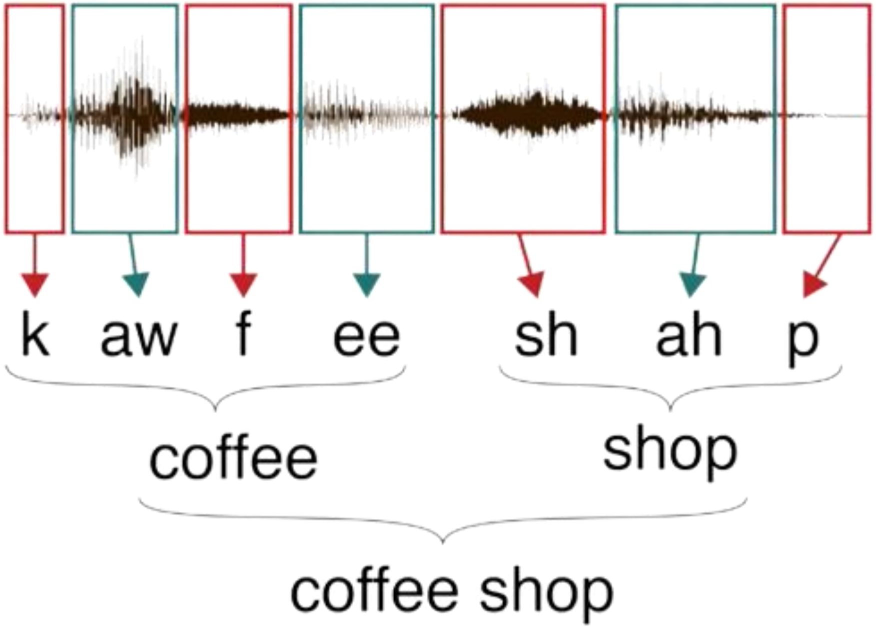
Optical Character Recognition OCR



Face Recognition



Virtual Assistants



Time-series Regression

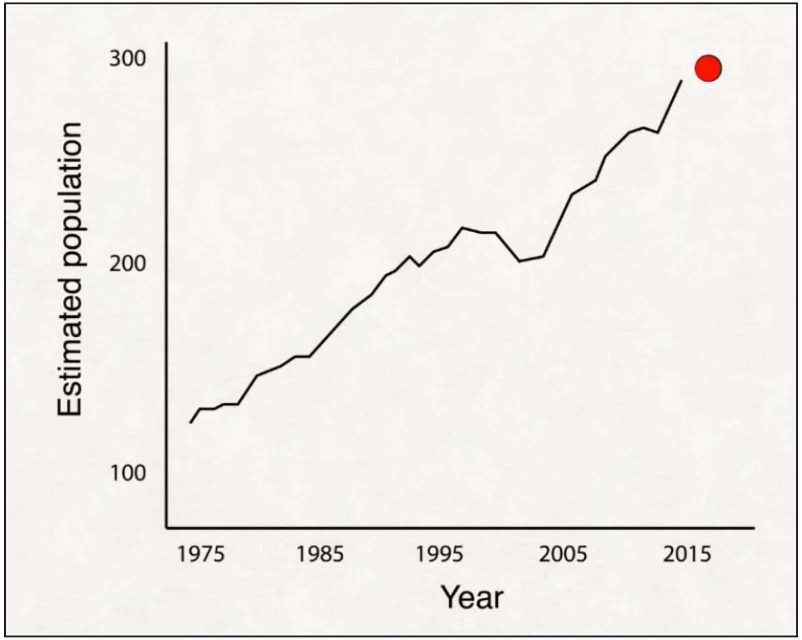
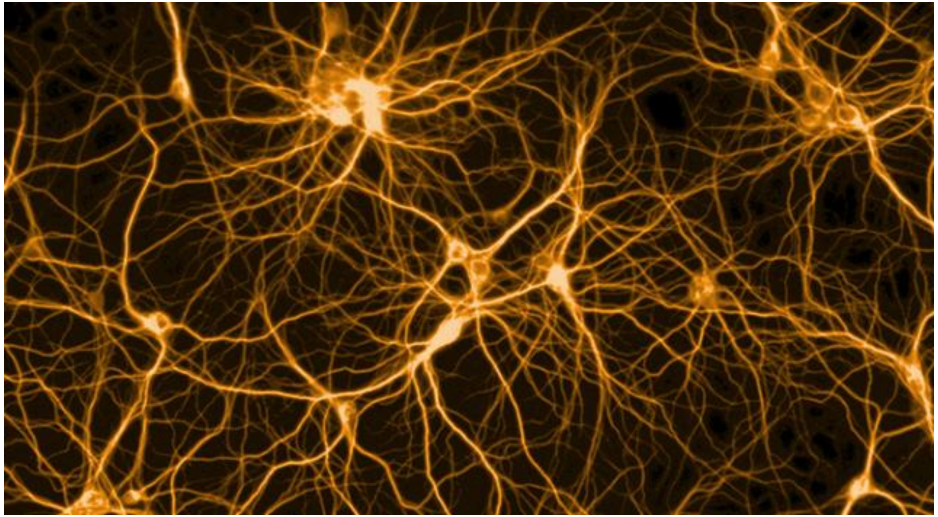


Image Denoising

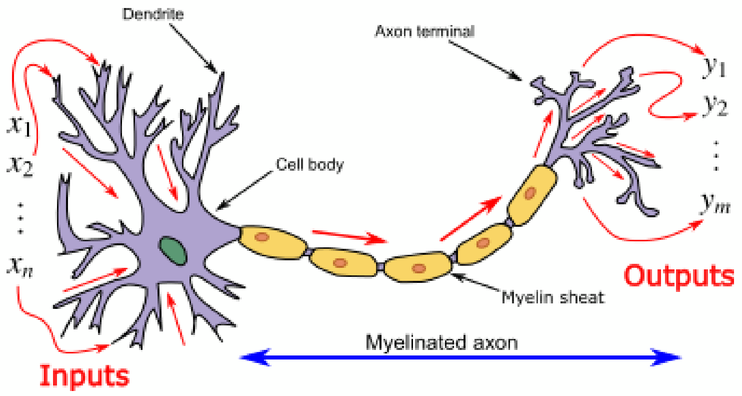
Neural Networks

Analogy

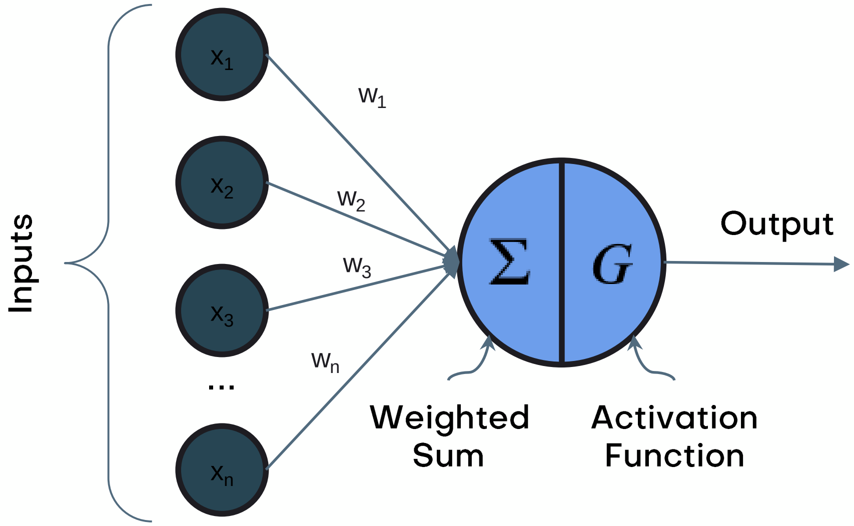
Our brain has lots of neurons connected together and the strength of the connections between neurons represents long term knowledge.



Analogy



Dendrite
Axon terminal
Cell body
Myelin sheat
Myelinated axon
x_1
x_2
x_n
y_1
y_2
y_m
Inputs
Outputs

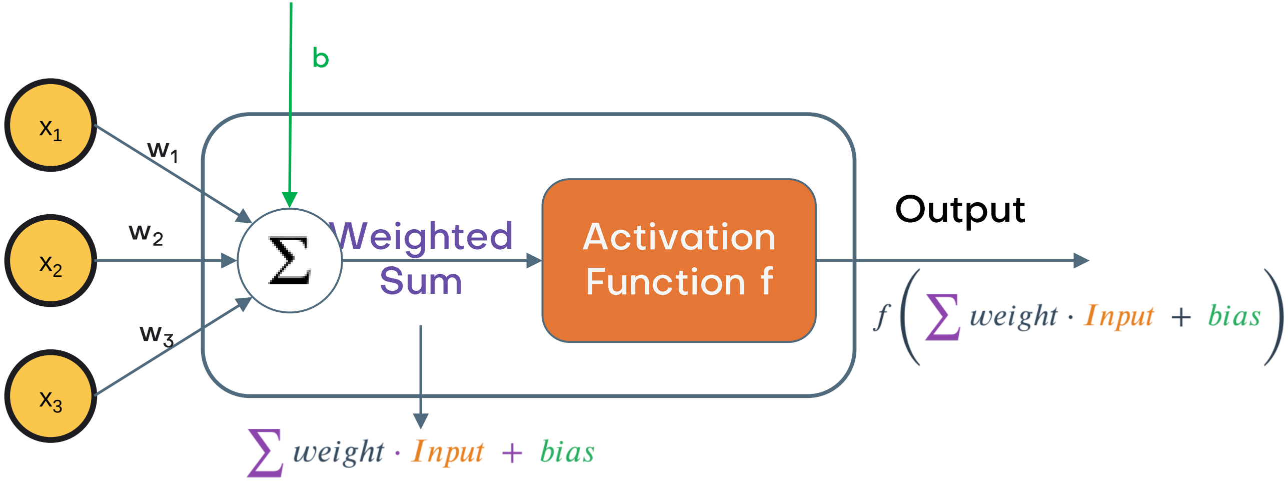


Inputs
x_1
x_2
x_3
...
x_n
w_1
w_2
w_3
w_n
Σ
G
Output
Weighted Sum
Activation Function

Structure of an Artificial Neuron

The inside of an artificial neuron has 2 fundamental parts:

- The first part computes the weighted sum of the inputs (the net output)
- The second part receives the weighted sum and gives the final output.

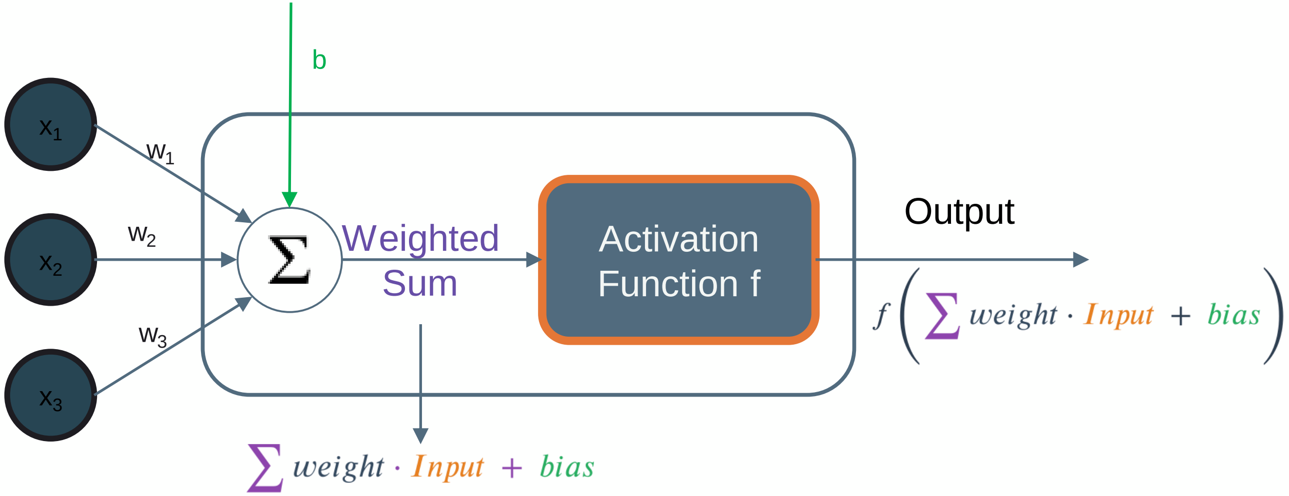


Activation Functions

Common Activation Functions

The following are the most famous activation functions:

- Binary step
- ReLU
- Sigmoid
- Tanh

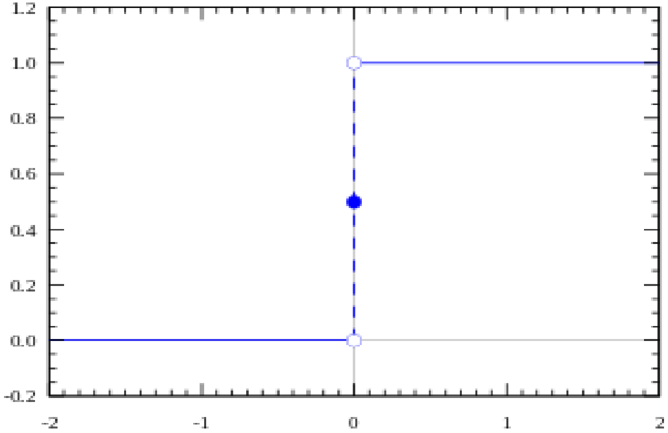


Binary Step

It tells if an input value is higher or lower than 0

Y = Sign(x)

Input Value: x



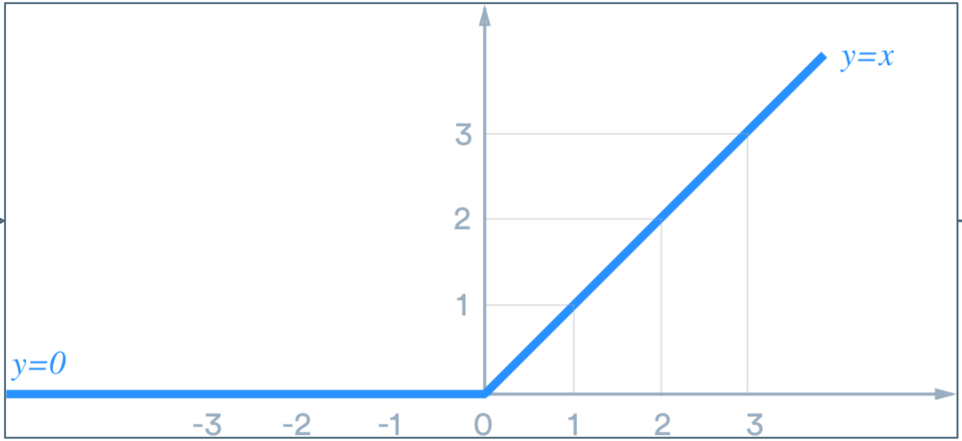
Output Value y

If (x>=0):
    y=1
else:
    y=0

Rectified Linear Unit (ReLu)

Y = max(0, x)

Input Value: x



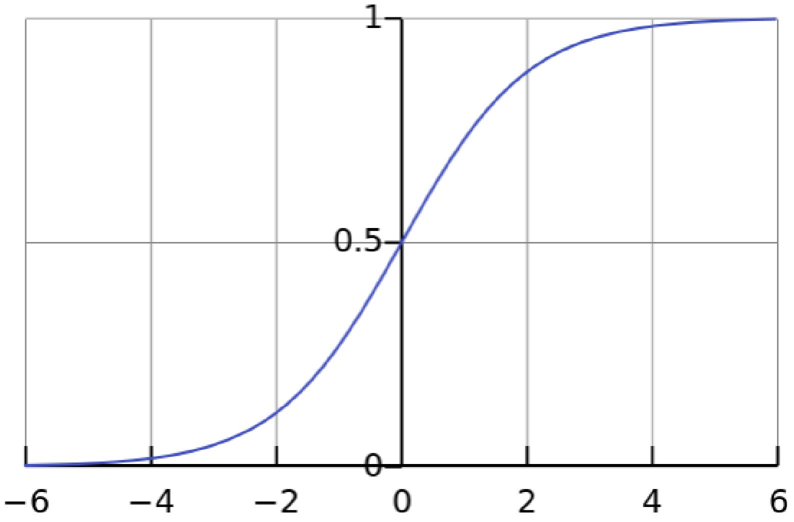
Output Value y

If (x>=0):
    y=x
else:
    y=0

Sigmoid Activation Function

$$y = \frac{1}{1 + e^{-x}}$$

Input Value: x



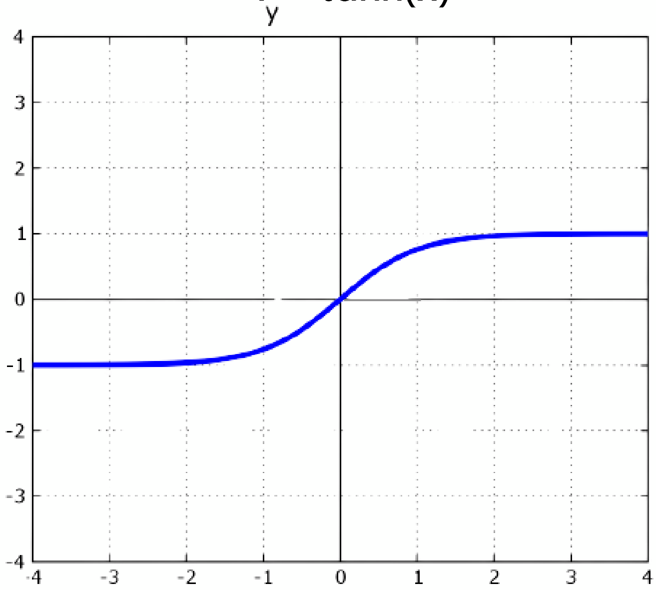
Output Value y

If (x>=0):
    0.5 <= y < 1
else:
    0 <= y < 0.5

Hyperbolic Tangent (tanh)

Y = tanh(x)

Input Value: x



Output Value y

If (x>=0):
0<=y<=1

else:
-1<=y<0

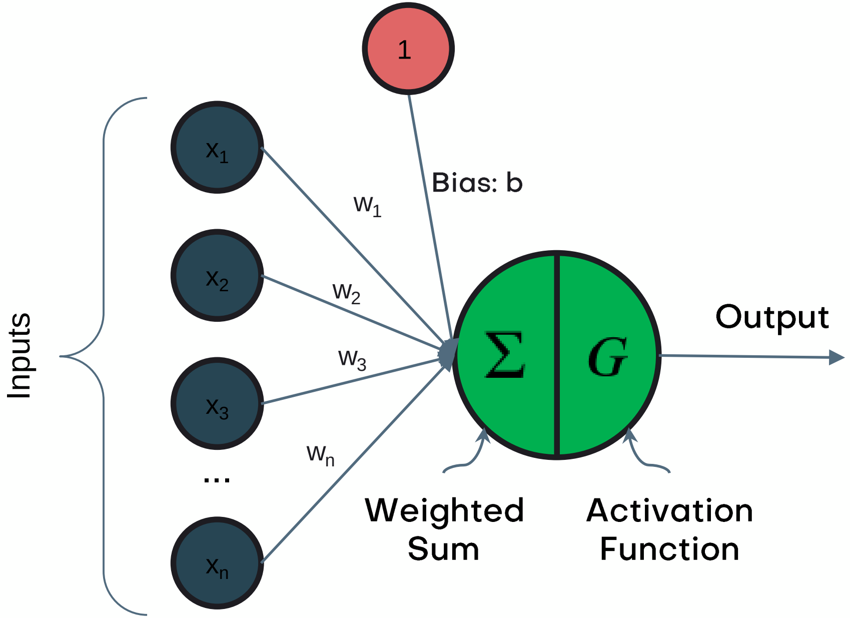
Feed Forward Neural Networks

The Perceptron

- Single-output neuron
- Binary step activation function
- Used to classify between linearly separable classes

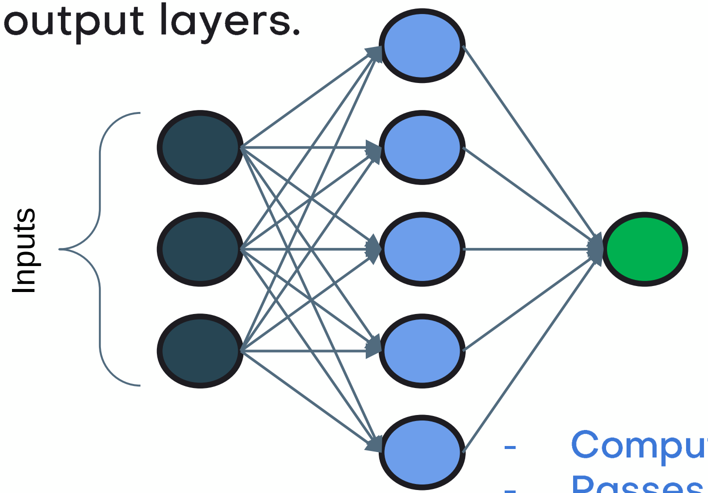
$$y = sign\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Parameters to learn



Single Hidden Layer Network

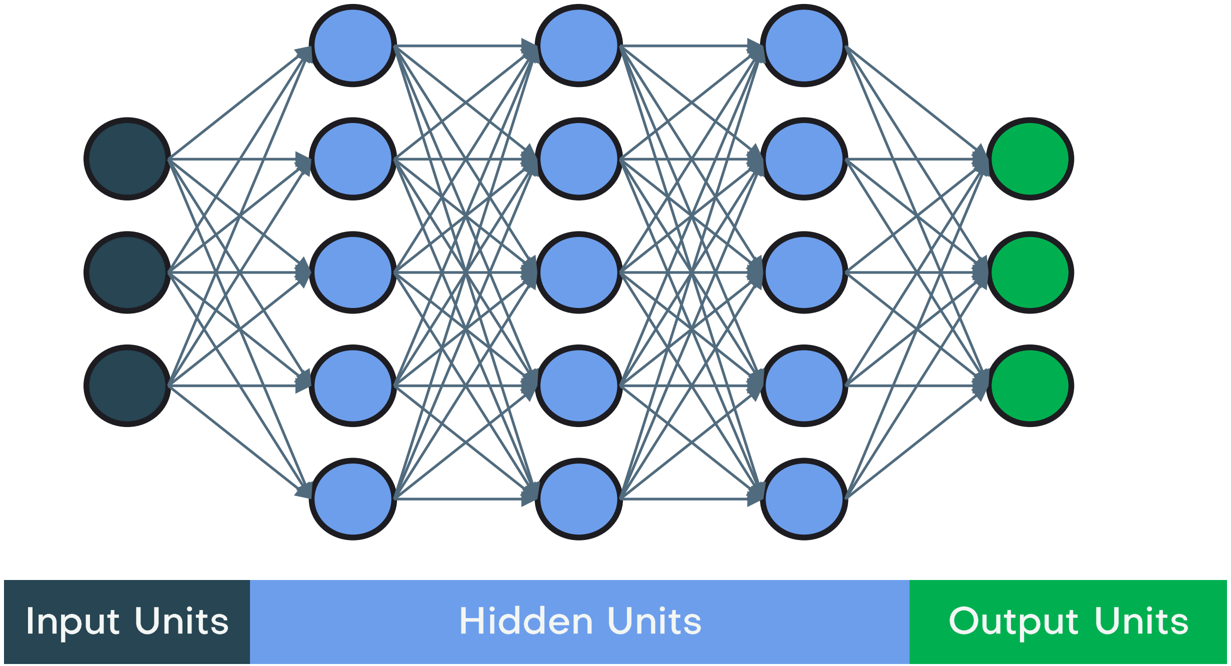
- We have one layer, called ‘hidden’, between the input and the output layers.



- Computes the weighted sum of the outputs of the previous layer.
- Passes sum to an activation function to obtain the final output.

- Computes the weighted sum of the inputs.
- Passes sum to an activation function to obtain output to be passed as input to next layer.

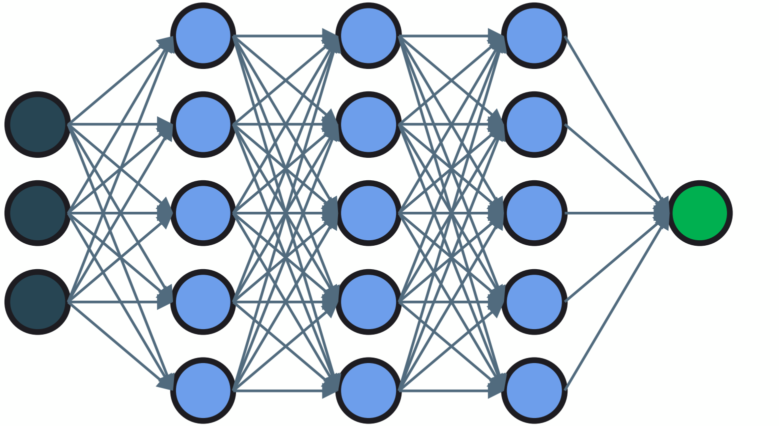
Deep Neural Networks



Neural Network Hyperparameters

Types of Activation Functions

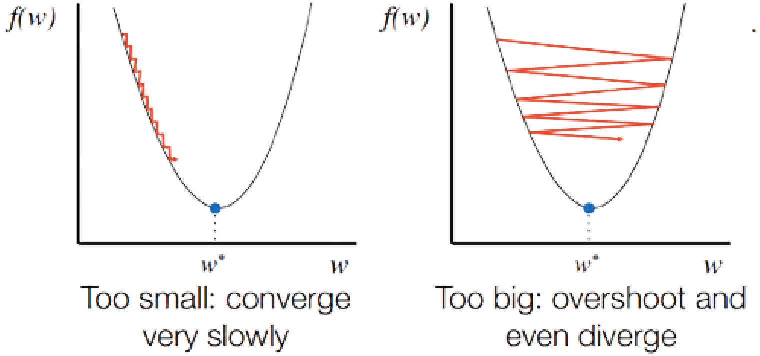
- There's no Rule of Thumb.
- People tend to use ReLU in their hidden layers.
- The output layer activation should be consistent with the type of problem.
- The step function is not much used because it contains a discontinuity at 0 that leads to problems in derivatives computed during learning phase.



Learning Rate

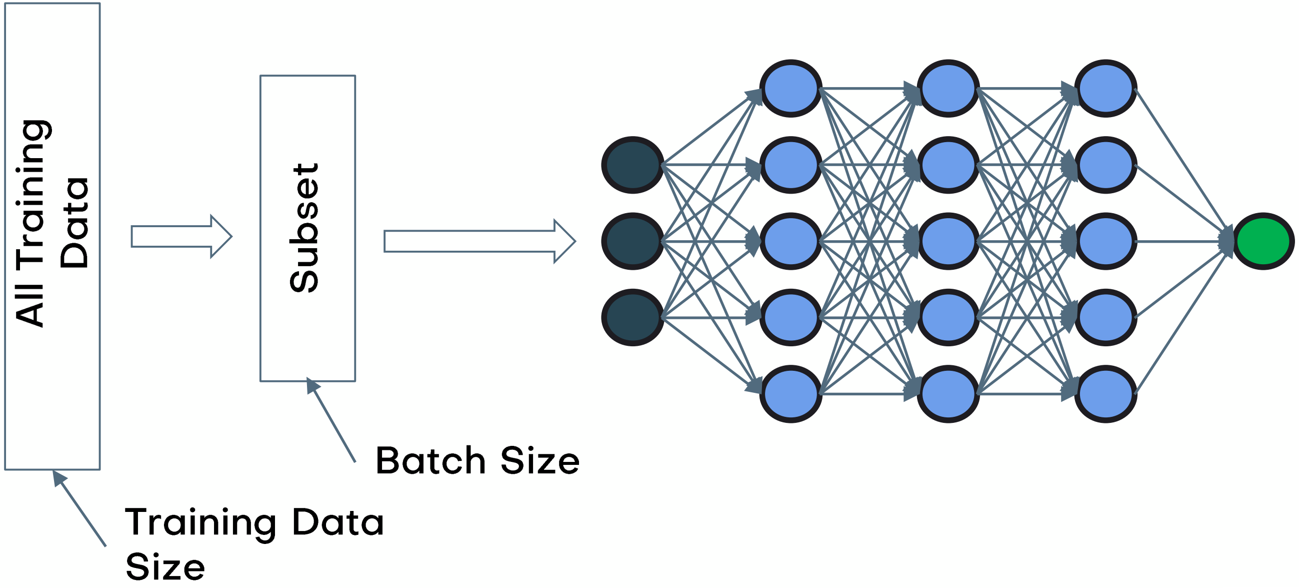
- Choosing the right learning rate is an essential step for reaching convergence.

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



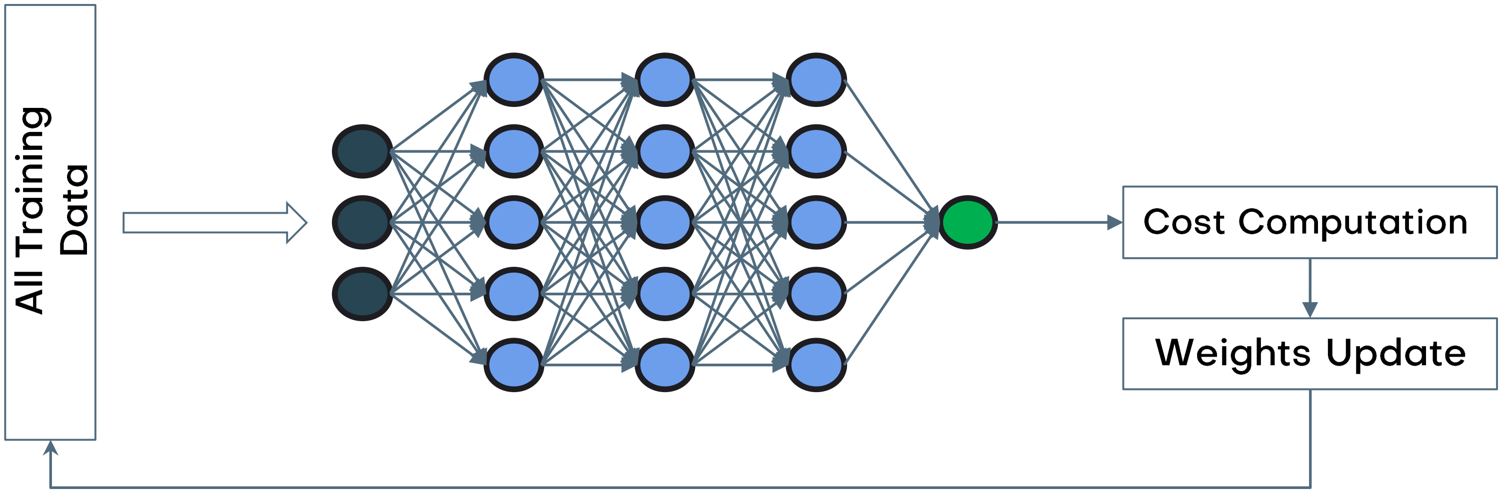
Batch Size

The amount of data you feed to the network before performing an update in the weights.

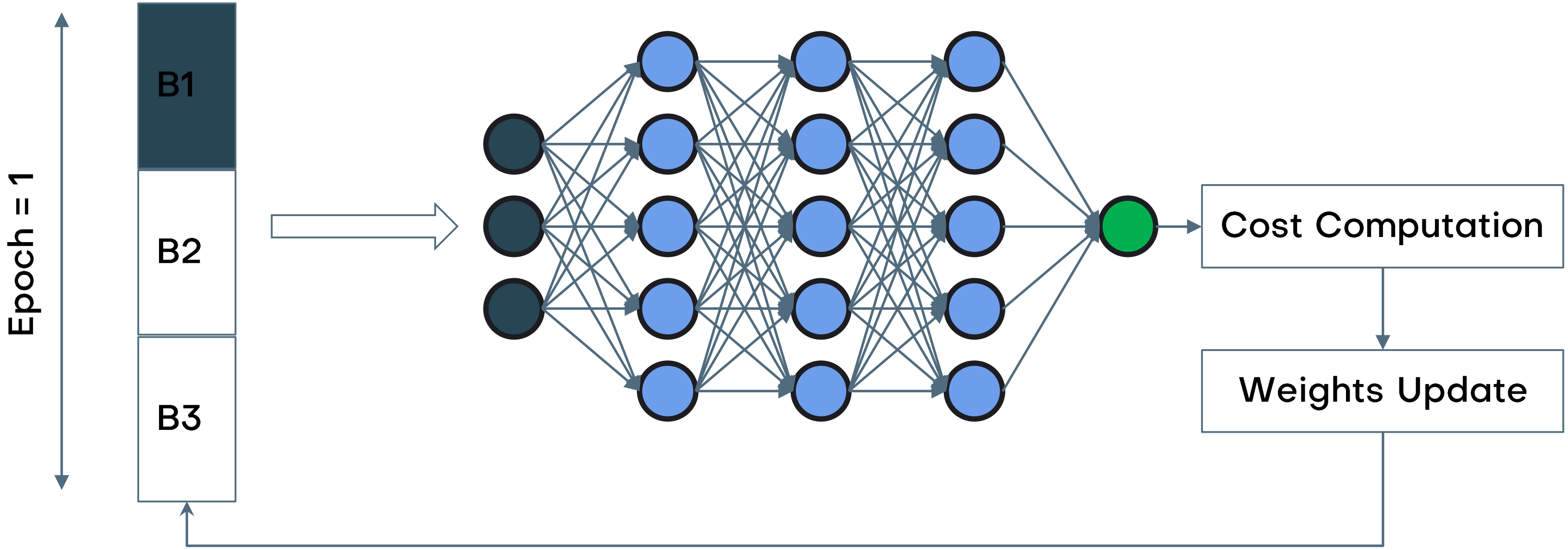


Epochs

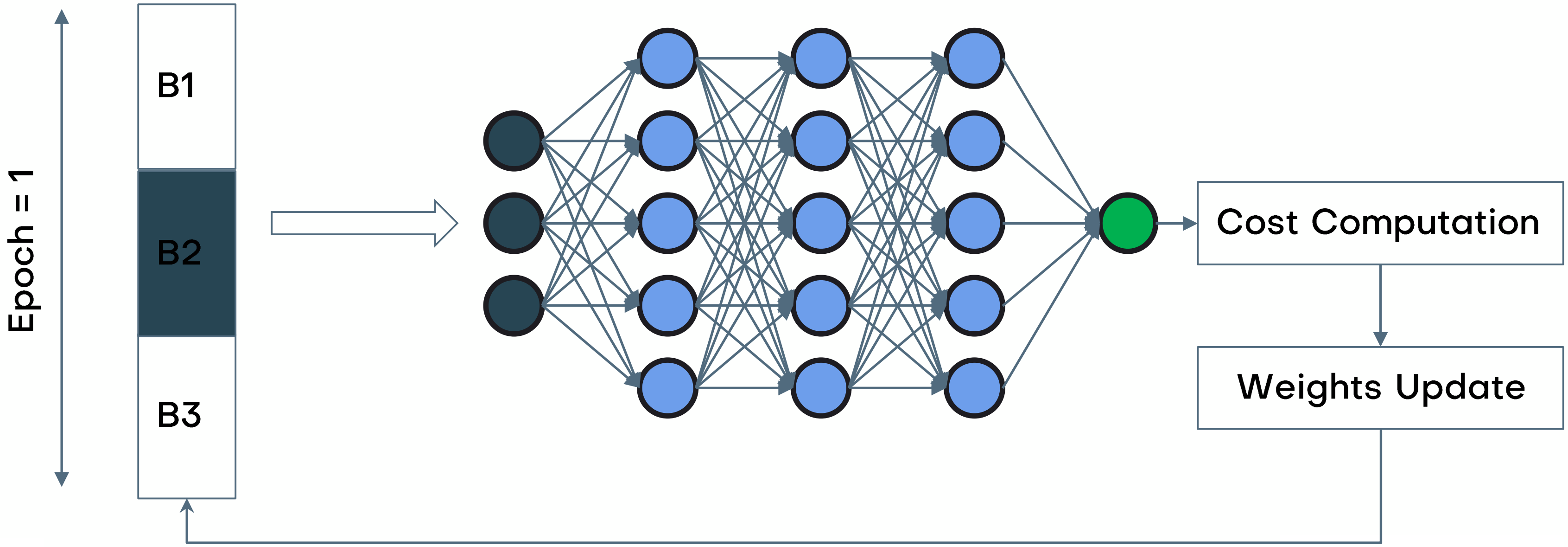
The number of times we feed ALL the training set examples to our network.



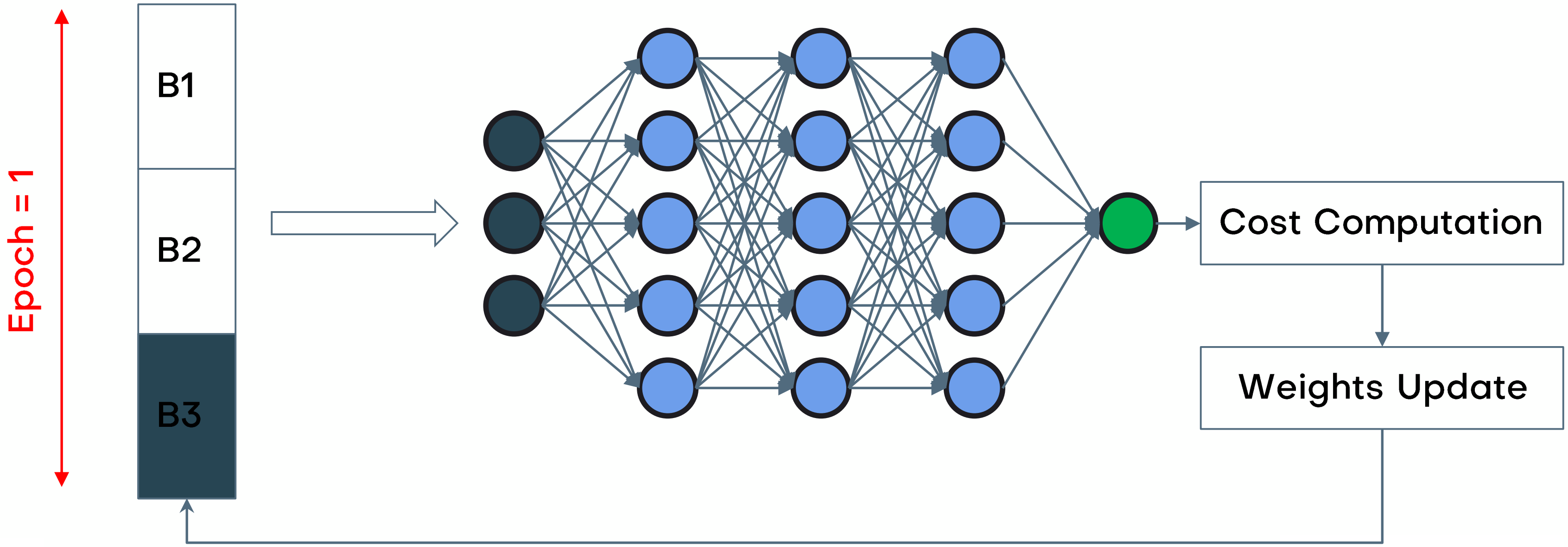
Batch size vs. Epochs



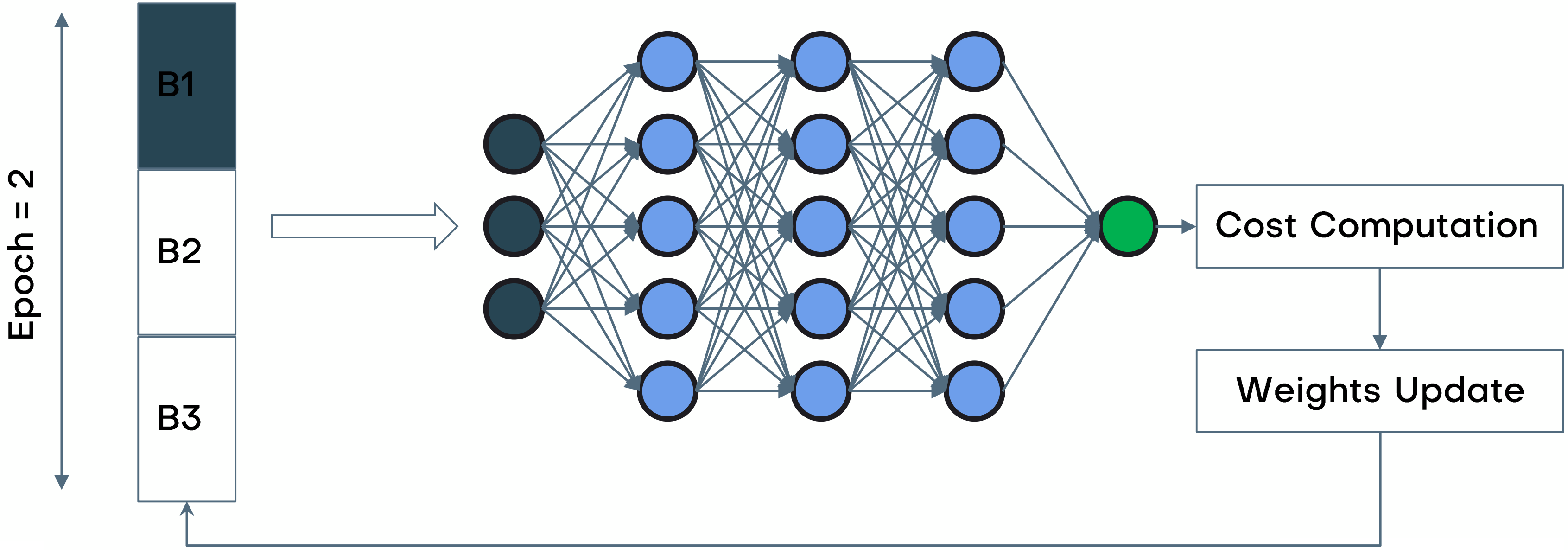
Batch size vs. Epochs



Batch size vs. Epochs



Batch size vs. Epochs



Hands-on: Build your first Neural Network

How do Neural Networks Learn?

Data Structure

We take a general case where the input data consists of:

- m examples.
- Each of them having n features.
- 1 Output

$m = 5$
Examples

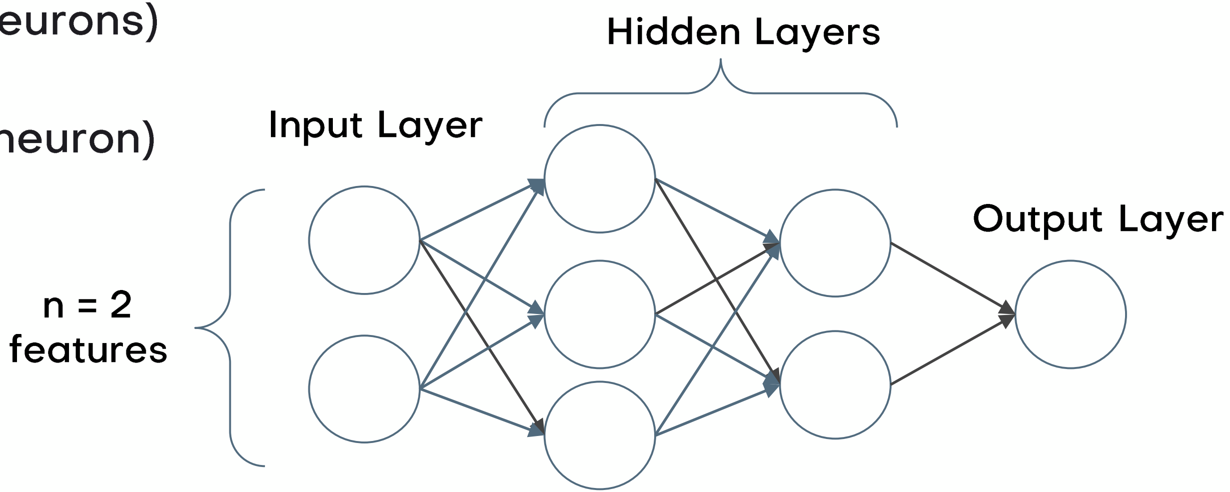
$n = 2$ features	x0	0.2	0.2	0.2	0.3	0.2
	x1	0.1	0.3	0.1	0.4	0.1

Output	y	1	0	1	0	1

Network Structure

In the general case we took, a neural network will consist of:

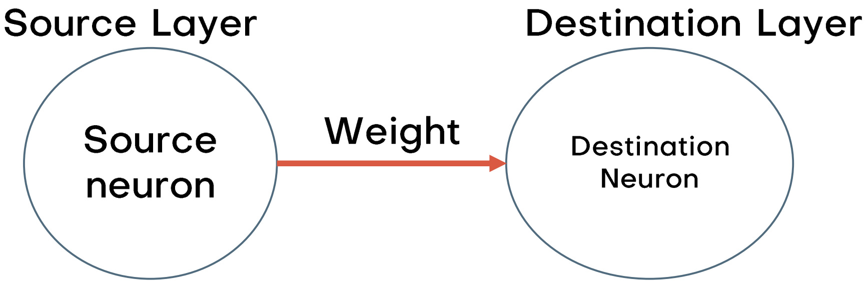
- Input Layer (of n neurons)
- Hidden Layers
- Output layer (of 1 neuron)



Weights

A weight $W^{l}_{(i,j)}$ is defined by 3 characteristics:

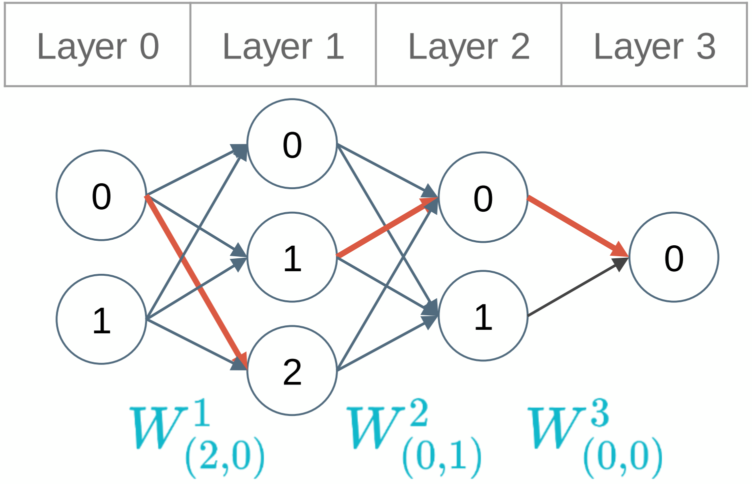
- l: Layer index (destination layer)
- i: Index of the neuron in layer l (destination neuron)
- j: Index of the neuron in layer l-1 (source neuron)



Weights

A weight $W^{l}_{(i,j)}$ is defined by 3 characteristics:

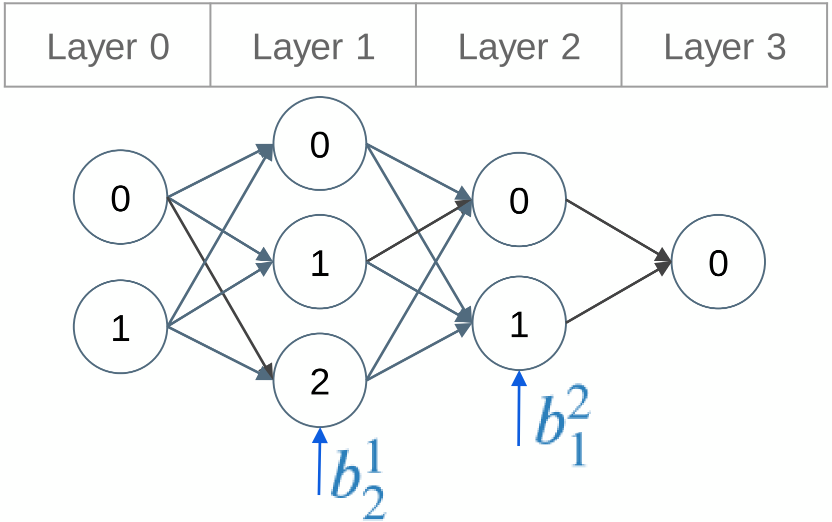
- l: Layer index (destination layer)
- i: Index of the neuron in layer l (destination neuron)
- j: Index of the neuron in layer l-1 (source neuron)



Weights are usually initialized randomly

Bias

- A bias is a characteristic of one neuron.
- We represent a bias by b_i^l where:
    - i is the index of the given neuron
    - l is the index of the given layer

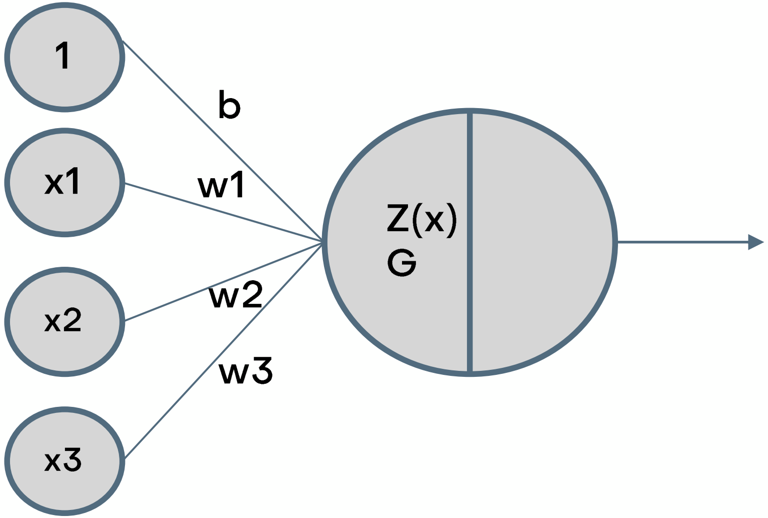


Bias values are usually initialized randomly

Feed Forward

Forward Pass

Recall the function that a single neuron having several inputs operates on.



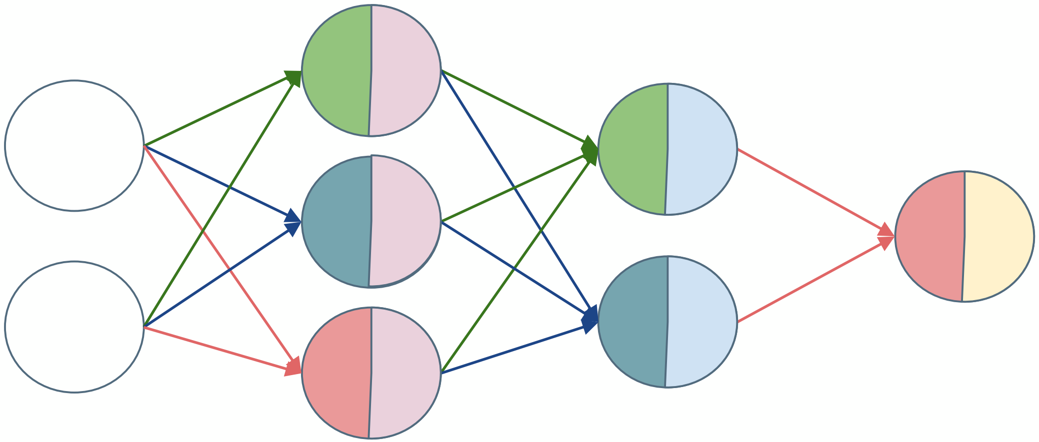
$$Z(x) = b + x_1 * w_1 + x_2 w_2 + x_3 w_3$$

$$A(x) = G(Z(x))$$

Forward Pass

Now this function will be executed for each neuron in each layer of the network.

- Neurons in each layer operate simultaneously.
- Activation functions are the same in each layer.

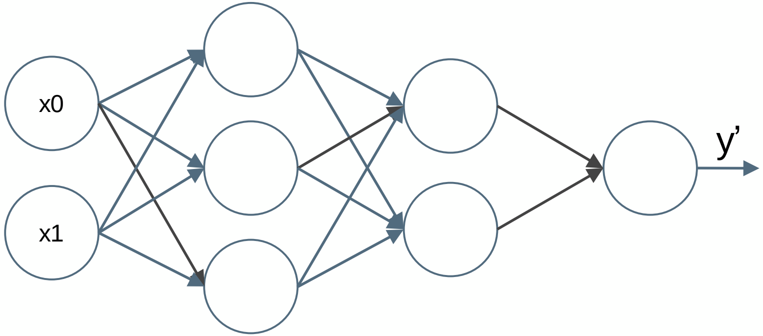


Cost Function

Loss Function

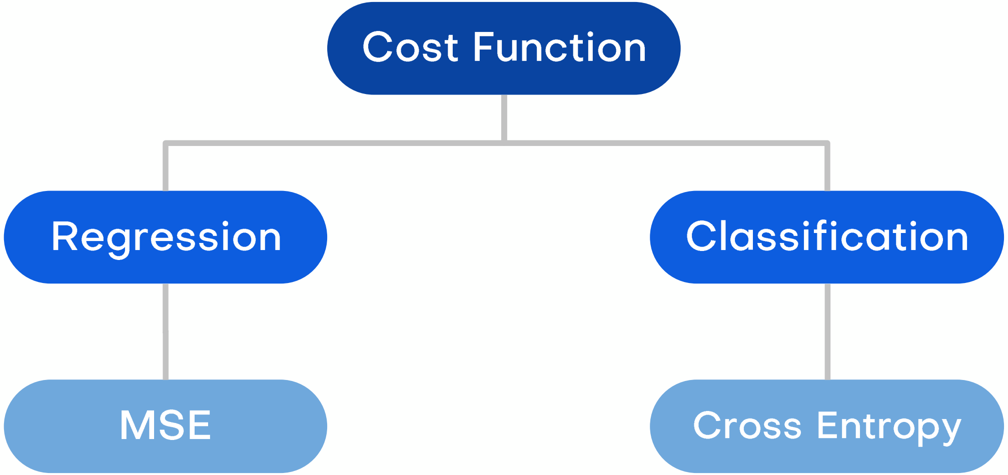
After the Forward Propagation is complete, we obtain a predicted value y' corresponding to the inputs we provided.

We define the loss function as some error metric between the predicted value (y') and the correct value (y) that we have for a specific input.



Cost Function

The cost function tells you how much your model is “making mistakes” by averaging the loss functions computed on all individual training samples.

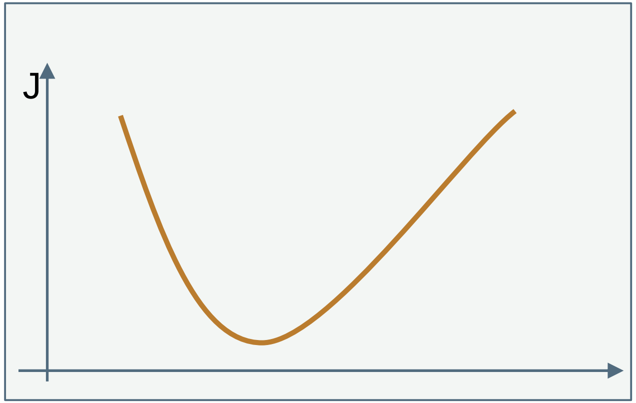


BackPropagation

This algorithm tries to minimize the cost function J by adjusting the weights.

Given:

- A function J = f(w)
- The functio $\frac{dJ}{dw} = \frac{df(w)}{dw}$
- The learning rate α



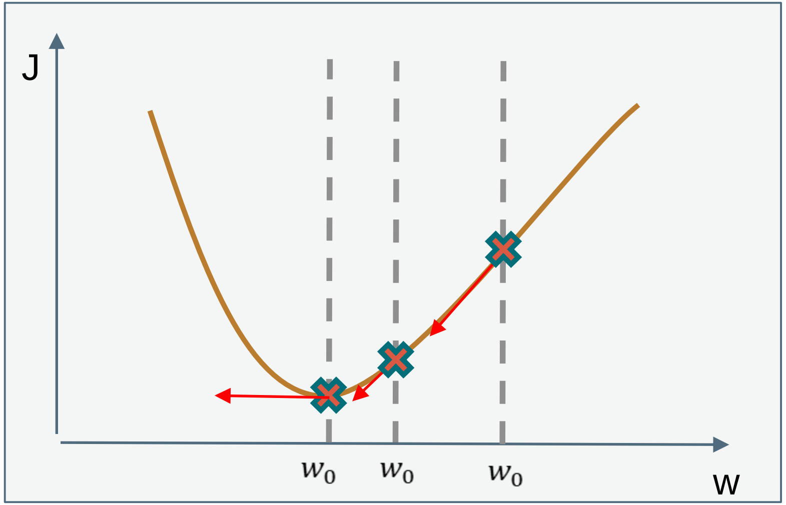
Gradient Descent

We will use the Gradient Descent algorithm to find the minimum.

- Pick a starting point w0 randomly
- Calculate $\left.\frac{dJ}{dw}\right|_{w=w_0}$
- Update w according to the formula

$$w_0 := w_0 - \alpha \left.\frac{dJ}{dw}\right|_{w=w_0}$$

- Get back to step 2



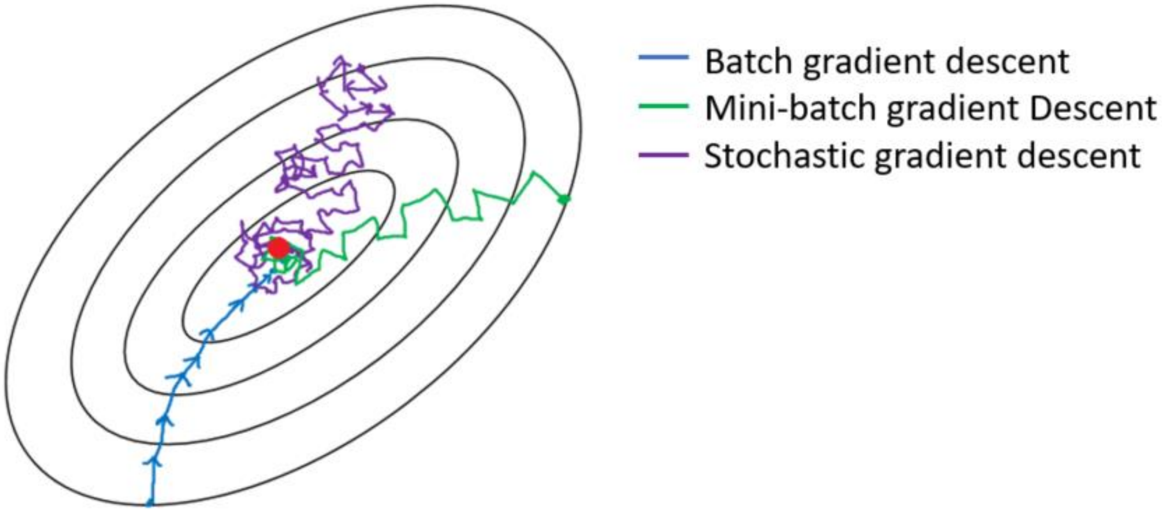
SDAIA
الهيئة السعودية للبيانات والذكاء الاصطناعي
Saudi Data & AI Authority

أكاديمية سدايا
SDAIA Academy

Variants of Gradient Descent

Optimization Problem	Samples in each Gradient Step	Number of Updates per Epoch
Batch Gradient Descent	The entire Dataset	1
Mini-Batch Gradient Descent	Subsets of the Dataset	$\frac{N}{\text{SIZE OF MINI-BATCH}}$
Stochastic Gradient Descent	Each Sample of the Dataset	N

Variants of Gradient Descent



Backward Pass

Backward Pass

The same GD principle applies to when you're dealing with a function (J) of multiple variables (Weights).

$$\frac{\partial J}{\partial W^3} = \begin{bmatrix} \frac{\partial J}{\partial W^3_{00}} & \frac{\partial J}{\partial W^3_{01}} \end{bmatrix} \quad \frac{\partial J}{\partial b^3} = \begin{bmatrix} \frac{\partial J}{\partial b^3_0} \end{bmatrix}$$

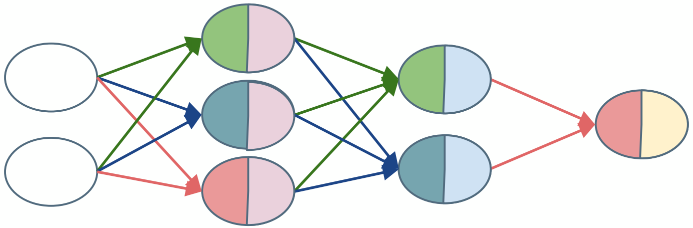
Layer 3

$$\frac{\partial J}{\partial W^2} = \begin{bmatrix} \frac{\partial J}{\partial W^2_{00}} & \frac{\partial J}{\partial W^2_{01}} & \frac{\partial J}{\partial W^2_{02}} \\ \frac{\partial J}{\partial W^2_{10}} & \frac{\partial J}{\partial W^2_{11}} & \frac{\partial J}{\partial W^2_{12}} \end{bmatrix} \quad \frac{\partial J}{\partial b^2} = \begin{bmatrix} \frac{\partial J}{\partial b^2_0} \\ \frac{\partial J}{\partial b^2_1} \end{bmatrix}$$

Layer 2

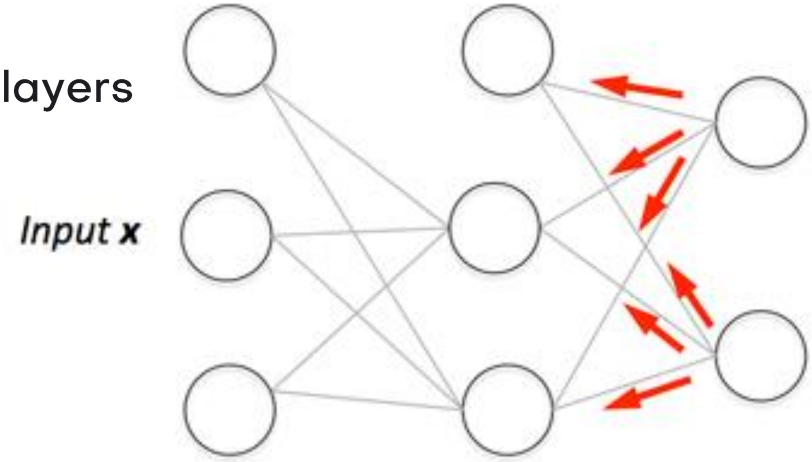
$$\frac{\partial J}{\partial W^1} = \begin{bmatrix} \frac{\partial J}{\partial W^1_{00}} & \frac{\partial J}{\partial W^1_{01}} \\ \frac{\partial J}{\partial W^1_{10}} & \frac{\partial J}{\partial W^1_{11}} \\ \frac{\partial J}{\partial W^1_{20}} & \frac{\partial J}{\partial W^1_{21}} \end{bmatrix} \quad \frac{\partial J}{\partial b^1} = \begin{bmatrix} \frac{\partial J}{\partial b^1_0} \\ \frac{\partial J}{\partial b^1_1} \\ \frac{\partial J}{\partial b^1_2} \end{bmatrix}$$

Layer 1



Backward Pass Process

- It starts at the output layer
- We compute the gradients of the loss function with respect to the weights
- Weights are updated
- We move backward to the hidden layers



Weight & Bias Update

Weight Update

After computing the derivatives, we compute the weights using the Gradient Descent update rule (where: alpha is the learning rate).

$$W^1 := W^1 - \alpha \frac{\partial J}{\partial W^1} \qquad b^1 := b^1 - \alpha \frac{\partial J}{\partial b^1}$$

$$W^2 := W^2 - \alpha \frac{\partial J}{\partial W^2} \qquad b^2 := b^2 - \alpha \frac{\partial J}{\partial b^2}$$

$$W^3 := W^3 - \alpha \frac{\partial J}{\partial W^3} \qquad b^3 := b^3 - \alpha \frac{\partial J}{\partial b^3}$$

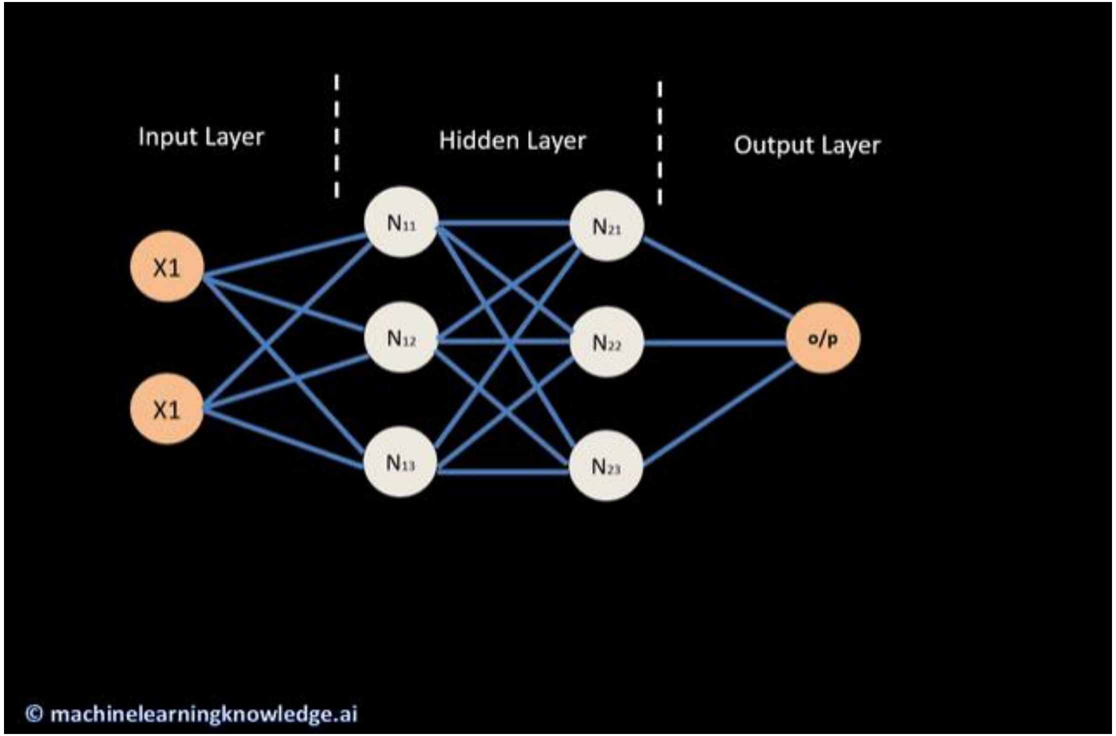
Repeat the Process again

Now that we've computed the new weights, we repeat the forward-backward process for several iterations.

We stop in 2 cases:

- We reach a maximum number of iterations
- We reach an optimum value for the weights

Recap



Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
© machinelearningknowledge.ai

شكراً لكم

Thank you

SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority

Assignment: Implementing Backpropagation