Deep Learning
CSC-Elective

Instructor : Dr. Muhammad Ismail Mangrio

Slides prepared by Dr. M Asif Khan

ismail@iba-suk.edu.pk

Week 1

Reading list

Main books

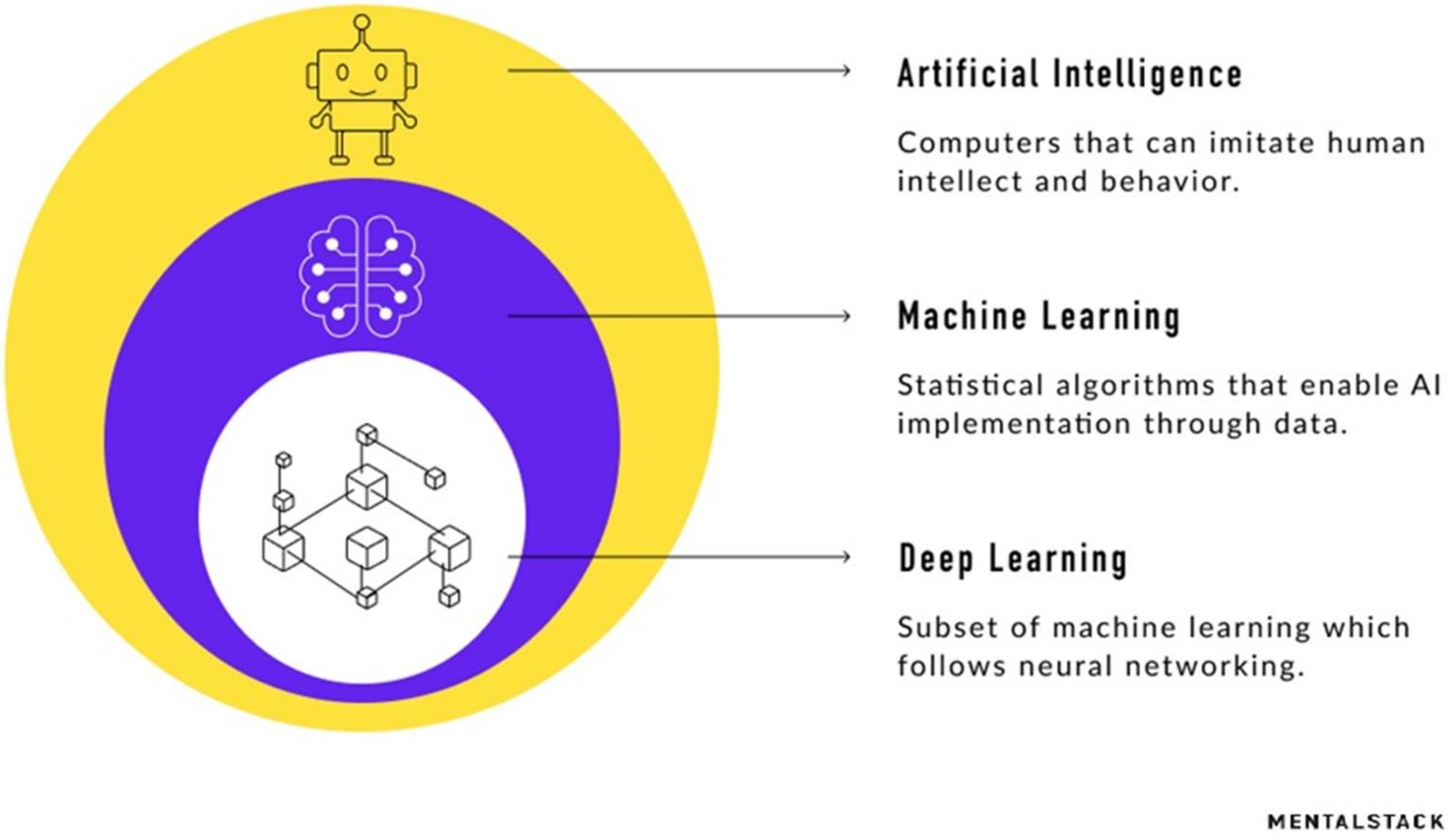
- Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems
- Deep Learning with Python, By FRANÇOIS CHOLLET
- Generative AI on AWS – Building Context-aware multimodal reasoning applications By Chris Fregly, Antje Barth & Shelbee Eigenbrode, 1st edition

DIGITAL &amp; WEB RESOURCES:

1. https://www.tutorialspoint.com/python_deep_learning/python_deep_learning_training_a_neural_network.htm
2. https://www.tutorialspoint.com/tensorflow/index.htm
3. https://www.datacamp.com/courses/introduction-to-deep-learning-in-python
4. https://www.tutorialspoint.com/natural_language_processing/index.htm
5. https://www.geeksforgeeks.org/deep-learning-tutorial/
6. https://www.datacamp.com/tutorial/tutorial-deep-learning-tutorial

Contents

AI Vs ML Vs DL



Artificial Intelligence

Computers that can imitate human intellect and behavior.

Machine Learning

Statistical algorithms that enable AI implementation through data.

Deep Learning

Subset of machine learning which follows neural networking.

AI Vs ML Vs DL

- In **AI** we are **mimicking human intellect** and **behavior**
  - In **ML** we are making machines to **learning without explicitly being programmed**
    - In **DL** we are making machines to **learning** based on **artificial neural networks**

AI Vs ML Vs DL

- It seems logical to look at the **human brain's architecture** for inspiration on how to **build an intelligent machine**.
- This is the **key idea** that inspired **artificial neural networks** (ANNs).
- However, although **planes were inspired by birds**, they **don't have to flap** their wings.
- Similarly, **ANN**s have gradually become quite **different** from their **biological cousins**.
- **Some researchers** even argue that we should **drop the biological analogy** altogether (e.g., by saying "units" rather than "neurons").

Revision of ML concepts

- Encoding categorical data
- Scaling
- Regression
- Classification
- Performance metrics
- Visualization of the results

Artificial Neural Network (Intro)

- ANNs are at very **core of Deep Learning**.
- They are:
  - **versatile, powerful**, and **scalable**,
- Making them ideal to **tackle large and highly complex** Machine Learning tasks, such as:
  - classifying billions of images (e.g., Google Images),
  - powering speech recognition
  - services (e.g., Apple's Siri),
  - recommending the best videos to watch to hundreds of millions of users every day (e.g., YouTube),
  - or learning to beat the world champion (DeepMind's AlphaGo).

Artificial Neural Network (Intro)

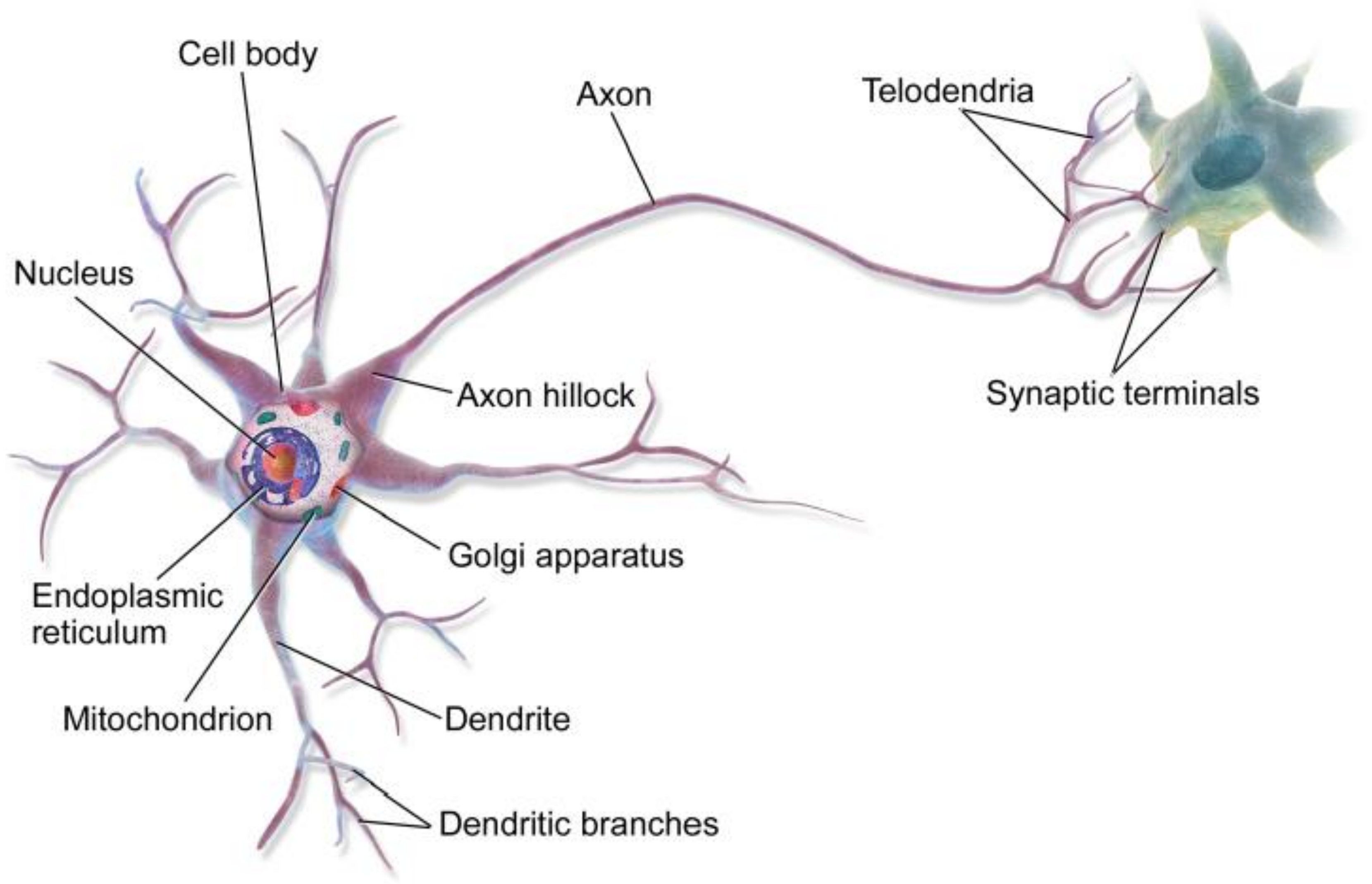
- First introduced back in 1943 by neurophysiologist **Warren McCulloch** and the mathematician **Walter Pitts**.
- They presented a simplified computational model of **how biological neurons might work together** in animal brains to perform complex computations using **propositional logic**.
- Many other architectures have been invented..

Artificial Neural Network (Intro)

- Now a **huge quantity of data available** to train neural networks, **ANN**s frequently **outperform ML techniques** on very large & complex problems.
- Tremendous **increase in computing** power since 1990s now makes it **possible to train large NN** in a reasonable amount of time.
- Some **theoretical limitations of ANNs** have turned out to be benign in practice. E.g., many people thought that ANN training algorithms were doomed because they were likely to **get stuck in local optima**
- but it turns out that this is rather rare in practice (or when it is the case, they are usually fairly close to the global optimum).

SUKKUR IBA UNIVERSITY COMPUTER SCIENCE

Artificial Neural Network (Biological Neuron)



Artificial Neural Network (Biological Neuron)

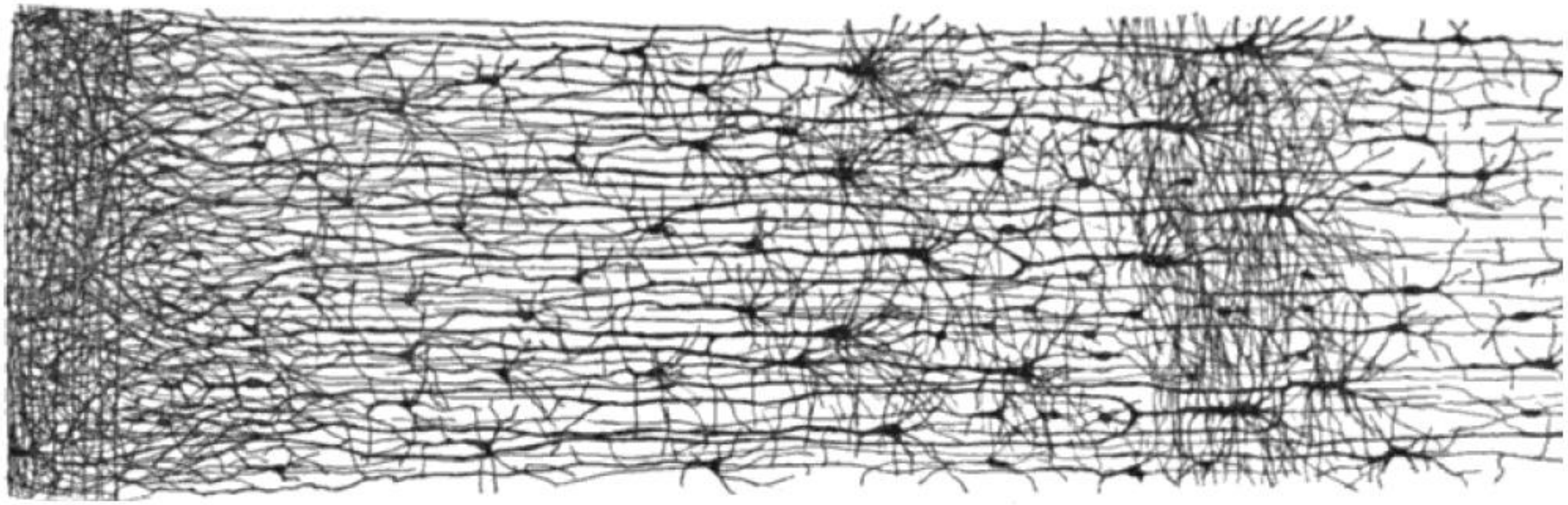
- It is an unusual-looking cell mostly found in animal **cerebral cortexes**, composed of a cell body containing the **nucleus** and most of the cell's complex components, and many **branching** extensions called **dendrites**,
- plus one **very long extension** called the **axon**.
- The **axon's len**gth may be just a few times longer than cell body, or **up to tens of thousands** of times longer.
- Axon splits off into many branches called **telodendria**
  - at the tip of these branches are minuscule structures called **synaptic terminals** (or simply synapses), which are connected to the dendrites (or directly to the cell body) of other neurons.

Artificial Neural Network (Biological Neuron)

- Biological neurons receive short **electrical impulses** called signals from other neurons via these synapses.
- When a neuron **receives enough signals** from other neurons within a few milliseconds, it **fires its own signals**.
- They are organized in a vast network of **billions of neurons;** each neuron typically connected to thousands of other neurons.
- **Highly complex computations** can be performed by a vast network of **simple neurons**.

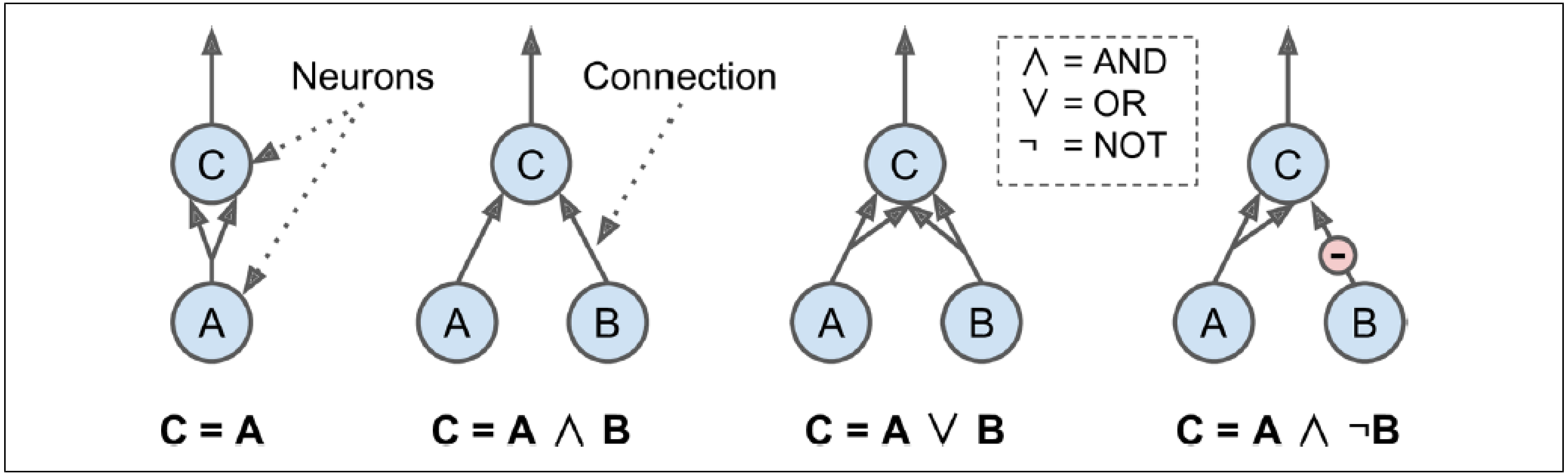
Artificial Neural Network (Biological Neuron)

- Multiple layers in a biological neural network



Artificial Neural Network (Logical Computations with Neurons)

- Warren McCulloch and Walter Pitts proposed a very simple model of the biological neuron, which later became an artificial neuron:

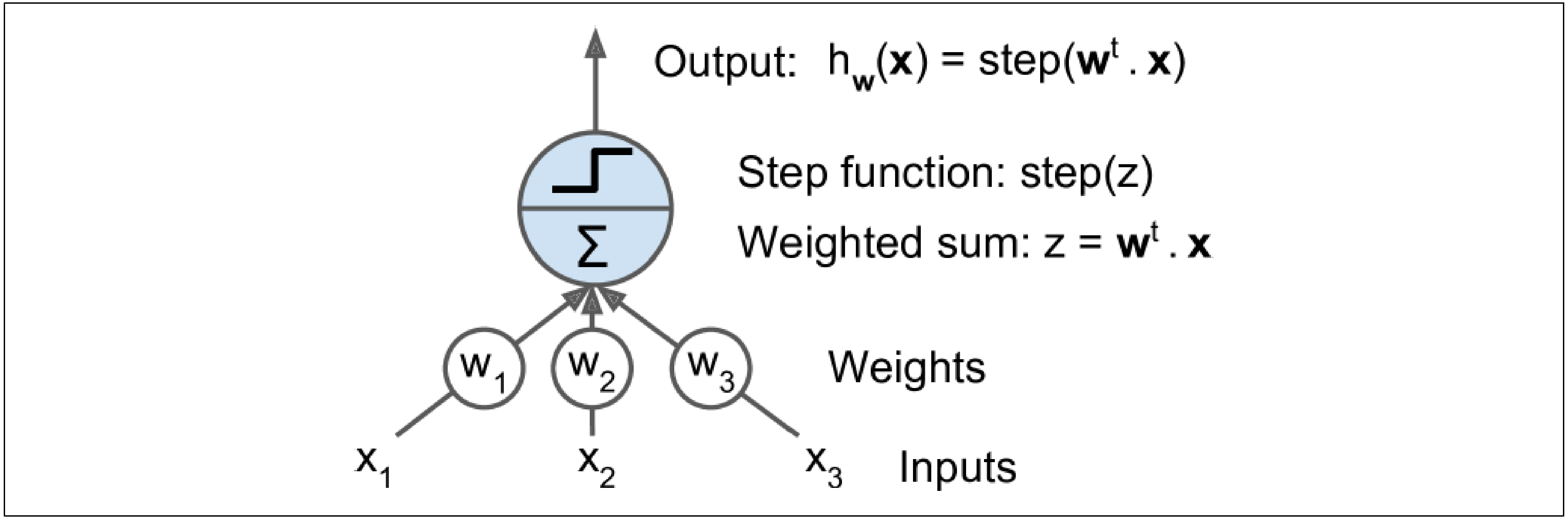


Artificial Neural Network (The Perceptron)

- The Perceptron is **one of the simplest ANN architectures**, invented in 1957 by Frank Rosenblatt.
- It is based on slightly different neuron, called **a linear threshold unit** (**LTU**):
  - the **inputs and output** are now numbers (instead of binary on/off values) and each input connection is **associated with a weight**.
  - The LTU computes a **weighted sum** of its inputs ($z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n = w^T \cdot x$),
  - Then **applies a step function** to that sum and outputs the result: $hw(x) = \text{step}(z) = \text{step}(w^T .x)$.

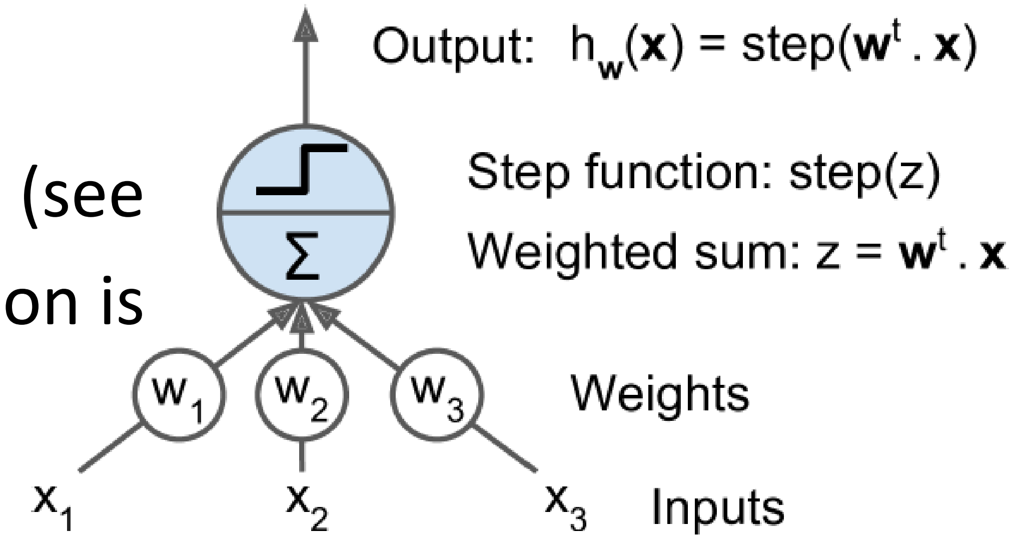
Artificial Neural Network (The Perceptron)

- A Linear Threshold Unit (LTU)


Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{w}^t \cdot \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^t \cdot \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Artificial Neural Network (The Perceptron)

- The most **common step function** used in Perceptrons is the **Heaviside** step function (see Equation 10-1). **Sometimes the sign** function is used instead.
- A **single LTU** can be used for **simple linear binary classification**



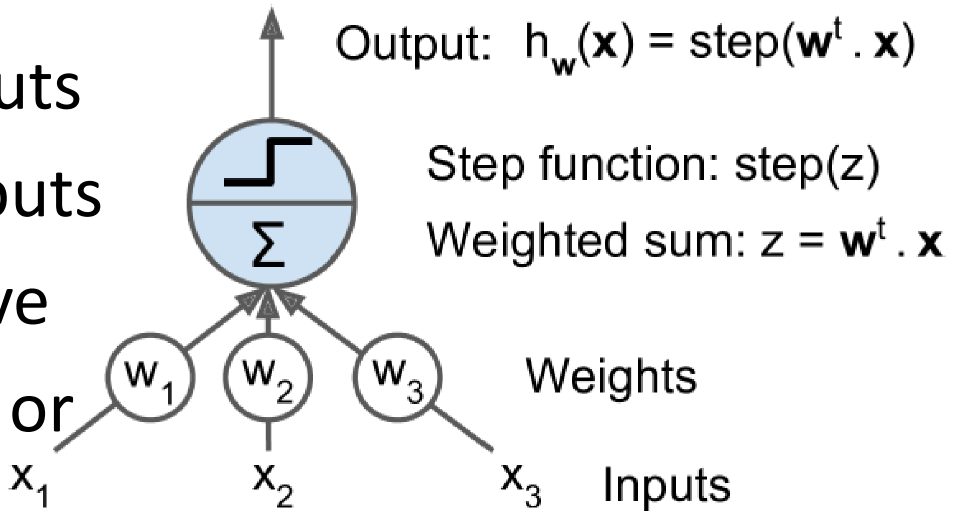
Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{w}^t \cdot \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^t \cdot \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Equation 10-1. Common step functions used in Perceptrons

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

Artificial Neural Network (The Perceptron)

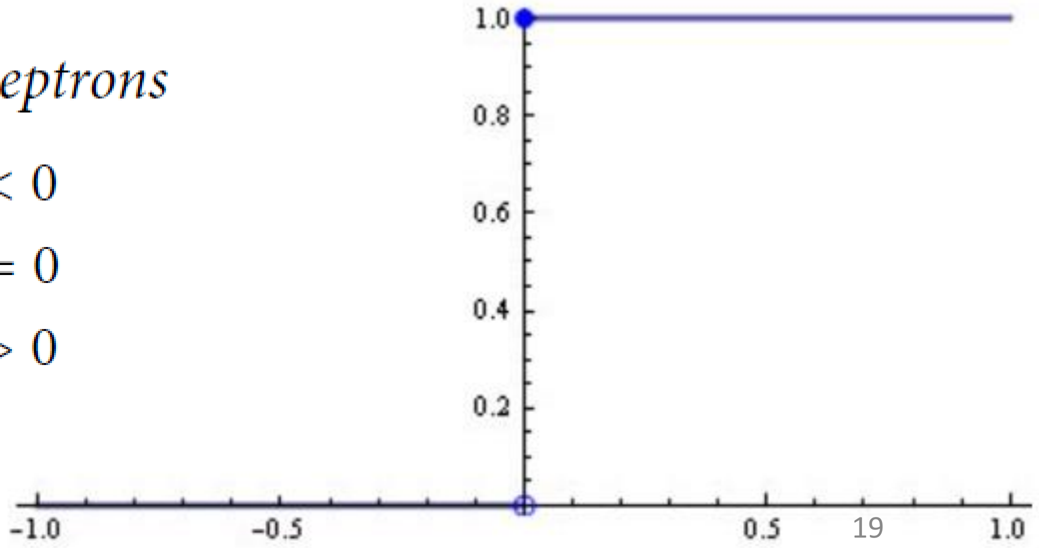
- It **computes a linear combination** of the inputs and if the **result exceeds a threshold**, it outputs the **positive class** or else outputs the negative class (just like a Logistic Regression classifier or a linear SVM).



Output: $h_w(\mathbf{x}) = \text{step}(\mathbf{w}^t \cdot \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^t \cdot \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

Equation 10-1. Common step functions used in Perceptrons

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \ge 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$



Artificial Neural Network (The Perceptron)

- **Training** an LTU means **finding the right values for** w0, w1, and w2.
- A **Perceptron is** simply composed of a **single layer of LTUs**, with each neuron connected to all the inputs.
- The name Perceptron sometimes means **a tiny network with a single LTU**.
- Special **passthrough** neurons called input neurons: they just output whatever input they are fed.
- Moreover, an extra **bias feature**, typically represented using a special type of neuron called a bias neuron, which just outputs 1 all the time.
  - Now $z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + \mathbf{x_{0(bais)}}$
  - Why we add bais?

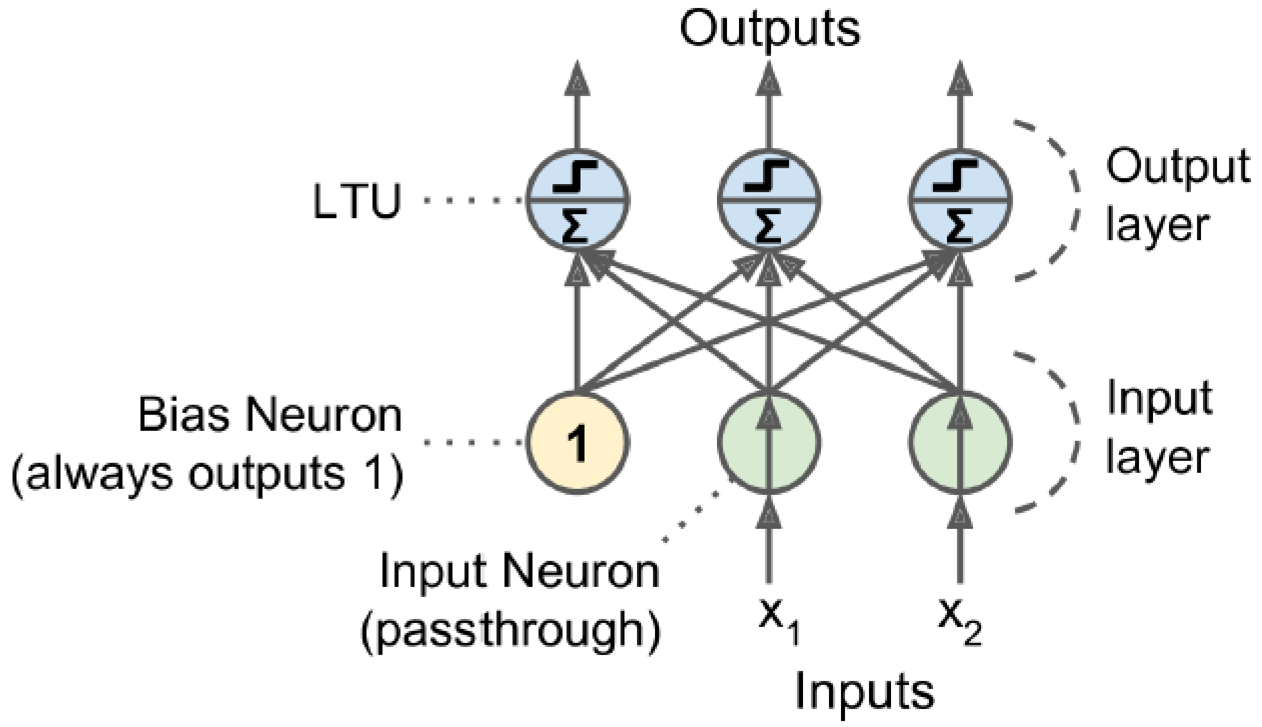
Artificial Neural Network (The Perceptron)

- Worked Example
  - **Inputs**: x = [2,3]
  - **Weights**: w = [0.6,0.8]
  - **Bias**: b = −2

- Practice
  - Use w=[1,−0.5], b=0.5
    1. x=[1,1] → z=?, Output = ?
    2. x=[2,1] → z=?, Output = ?
    3. x=[0,3] → z=?, Output = ?

Artificial Neural Network (The Perceptron)

- A Perceptron with two I/O and three O/P is represented in Figure 10-5.
- This Perceptron can classify instances simultaneously into three different binary classes, which makes it a multioutput classifier.



Artificial Neural Network (The Perceptron training)

- Donald Hebb suggested that when a biological **neuron often triggers another neuron**, the **connection** between these two neurons **grows stronger.**
- This idea was later summarized by Siegrid Löwel in this catchy phrase:
  - “Cells that fire together, wire together.”
- This rule later became known as **Hebb’s rule** (or Hebbian learning);
  - that is, the **connection weight** between two neurons **is increased** whenever they have the **same output**.

Artificial Neural Network (The Perceptron training)

- Perceptrons are **trained using a variant of this rule** that takes into account the **error** made by the network; **it does not reinforce connections** that lead to the **wrong output**.
- More specifically, the Perceptron is **fed one training instance** at a time, and for each instance it makes its predictions.
- For every output neuron that produced a wrong prediction, it **reinforces the connection weights from the inputs** that would have contributed to the correct prediction. The rule is shown in Equation 10-2.

Equation 10-2. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

Artificial Neural Network (The Perceptron training)

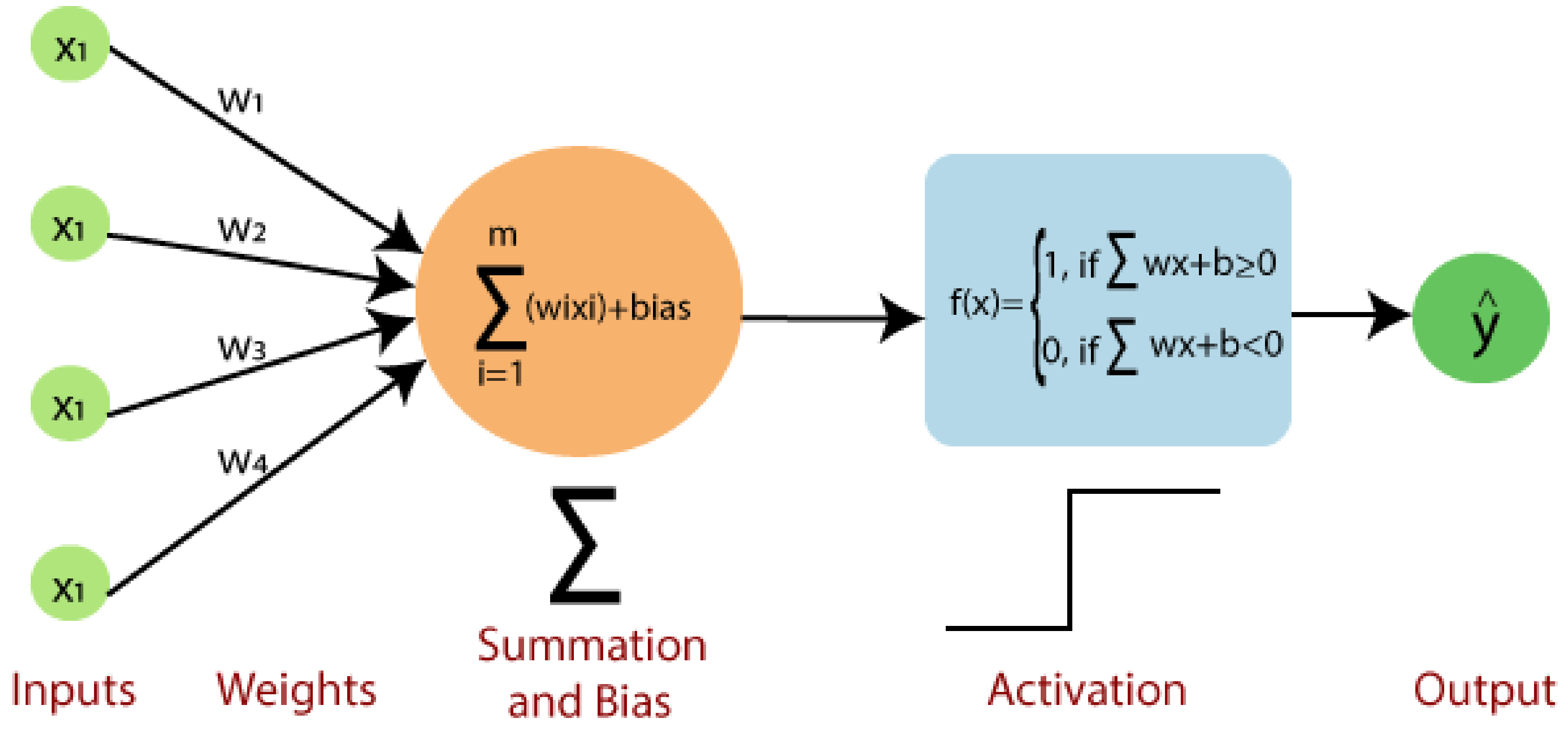
- With the below formula the **weights** of the neurons are **updated during training phase**.
- $w_{i,j}$ is the connection weight between the i_{th} input neuron and the j_{th} output neuron (before update).
- $wi,j^{(next\ step)}$ → Updated weight (after learning).
- x_i is the i_{th} input value of the current training instance.
- y'_j is the output of the j_{th} output neuron for current training instance.
- y_j is the target output of the j_{th} output neuron for current training instance.
- η (eta) is the learning rate.

Equation 10-2. Perceptron learning rule (weight update)

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

Artificial Neural Network (Single Layer Perceptron)

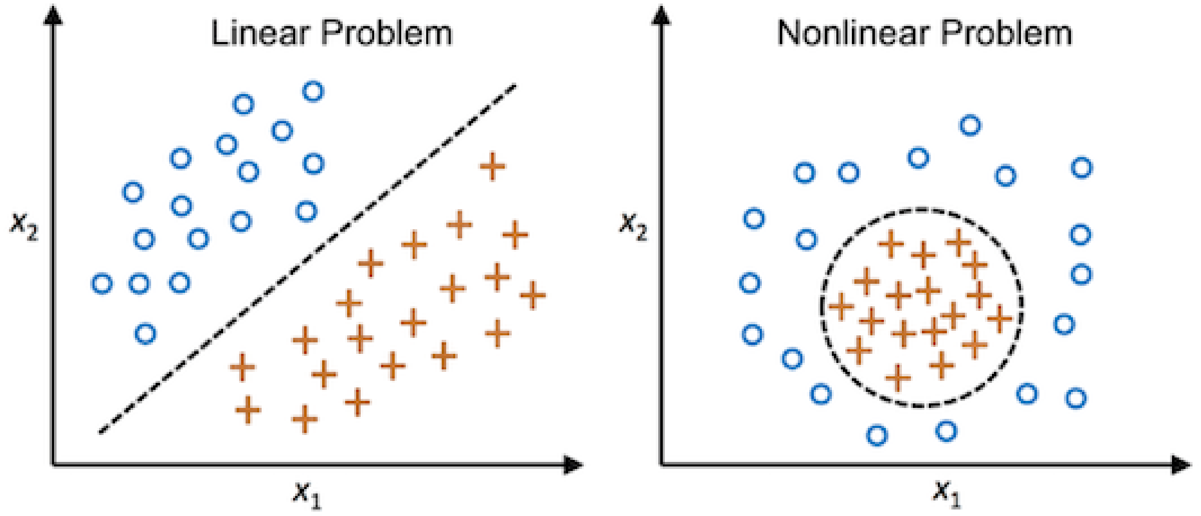
- Till now we have only studied **single layer perceptron**, show in the figure.
- It has **input neurons** (x1, x2, .. xi), **weighted summation**, **bias**, **activation** functions (linear, sgn, Heaviside, sigmoid etc) and **output**.



X1
X1
X1
X1
W1
W2
W3
W4
$\sum_{i=1}^{m}$(wixi)+bias
f(x)= {1, if ∑ wx+b≥0 ; 0, if ∑ wx+b<0
ŷ
∑
Inputs
Weights
Summation and Bias
Activation
Output

Artificial Neural Network (Single Layer Perceptron)

- A single-layered perceptron model **consists feed-forward network** and also includes a threshold transfer function (step func) inside the model.
- The main objective of the single-layer perceptron model is to **analyze the linearly separable objects with binary outcomes**.



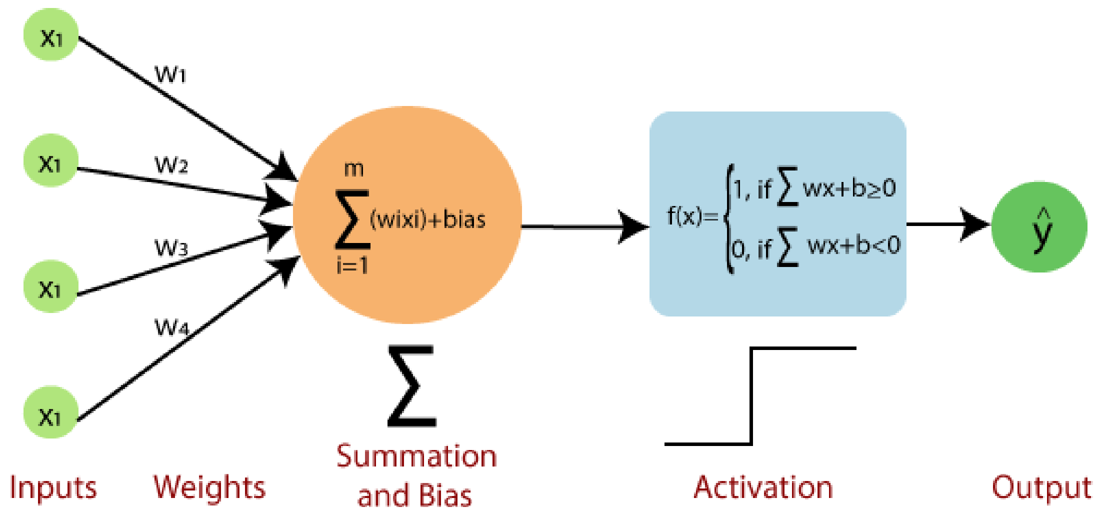
Artificial Neural Network (Single Layer Perceptron)

- The decision boundary of each output neuron is linear, so SLP are **incapable of learning complex patterns** (just like Logistic Regression classifiers).
- Scikit-Learn provides a Perceptron class that implements a single LTU network.

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype(np.int32) # Iris Setosa?
per_clf = Perceptron(random_state=42)

per_clf.fit(X, y)
y_pred = per_clf.predict([[2, 0.5]]) # see elearning for pynb code
```



x_1
x_1
x_1
x_1
w_1
w_2
w_3
w_4
$\sum_{i=1}^{m}$ (wixi)+bias
$f(x)=\begin{cases}1, \text{if} \sum wx+b\geq 0\\0, \text{if} \sum wx+b<0\end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

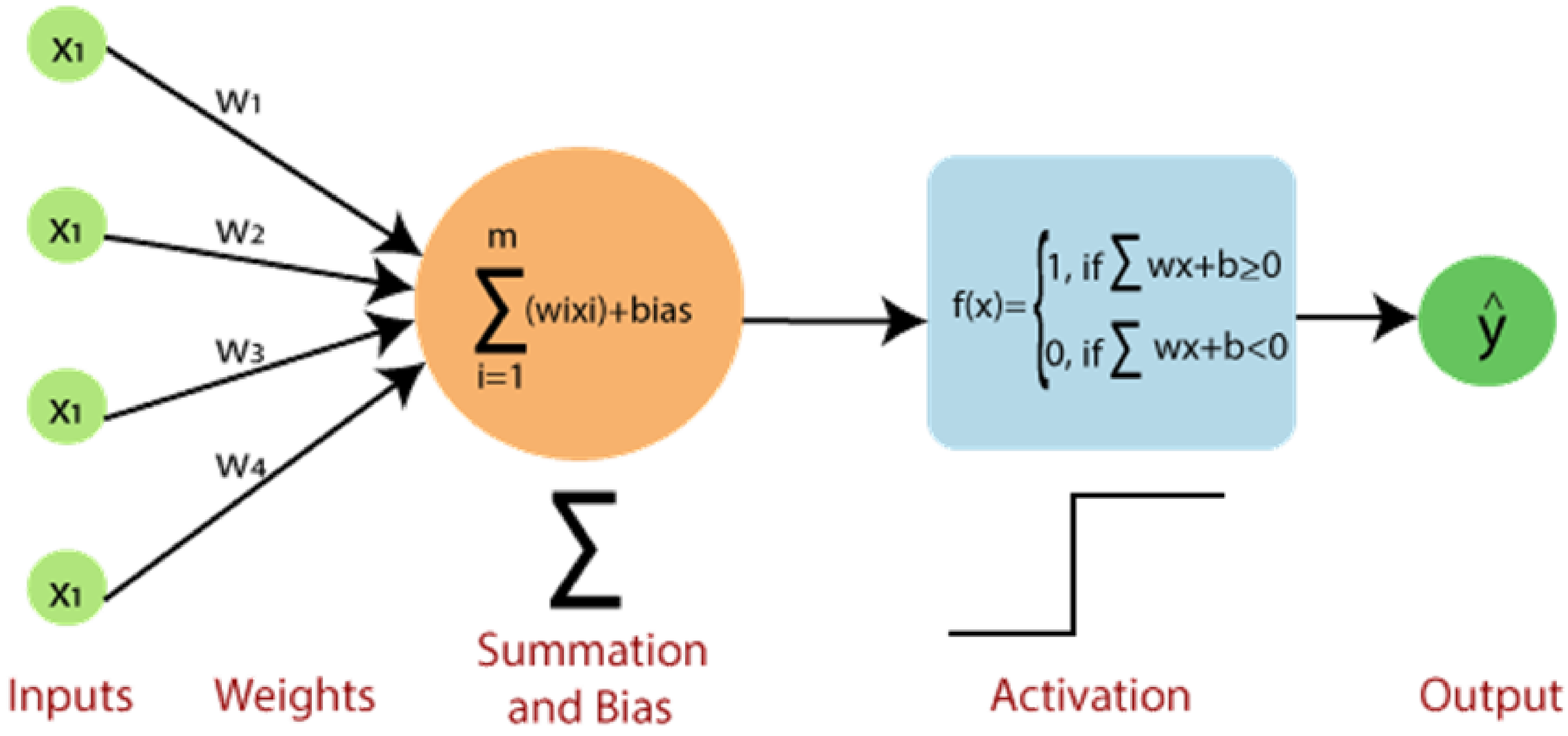
Class exercise (5 minutes)

- You have to train a single layer Perceptron to predict (binary classification) whether a bowler will take wicket in next cricket match if he/she has following history.
- Draw and frame your problem for single Perceptron
  - Identify and draw what inputs
  - Weights
  - Weighted sum
  - Activation function
  - Output
  - How training will be done

Hours practice	Hours rest	Hours domestic matches played	Wickets taken
100	200	50	Yes
200	100	10	No
150	120	20	No
80	150	70	Yes
100	130	60	Yes

Basic Components of Perceptron

- Input Nodes or Input Layer
- Weights, Bias and Weight sum
- Activation Function
- Output layer


X1
X1
X1
X1
W1
W2
W3
W4
$\sum_{i=1}^{m}(wixi)+bias$
$f(x)=\begin{cases}1, \text{ if } \sum wx+b\geq 0\\ 0, \text{ if } \sum wx+b<0\end{cases}$
$\hat{y}$
$\sum$
Inputs
Weights
Summation and Bias
Activation
Output

Basic Components of Perceptron

- **Weight and Bias**:
  - Weight represents **strength of the connection** between units.
  - Weight is directly proportional to strength of associated input neuron in deciding output.
  - **Bias** is the **line of intercept** in a linear equation (decision boundary for linear models).
  - It allows network to account for situations **where all input features are zero** or when there is no input at all.
- **Activation Function**:
  - Determines **whether the neuron will fire or not**.
  - Can be considered **primarily as a step function**.

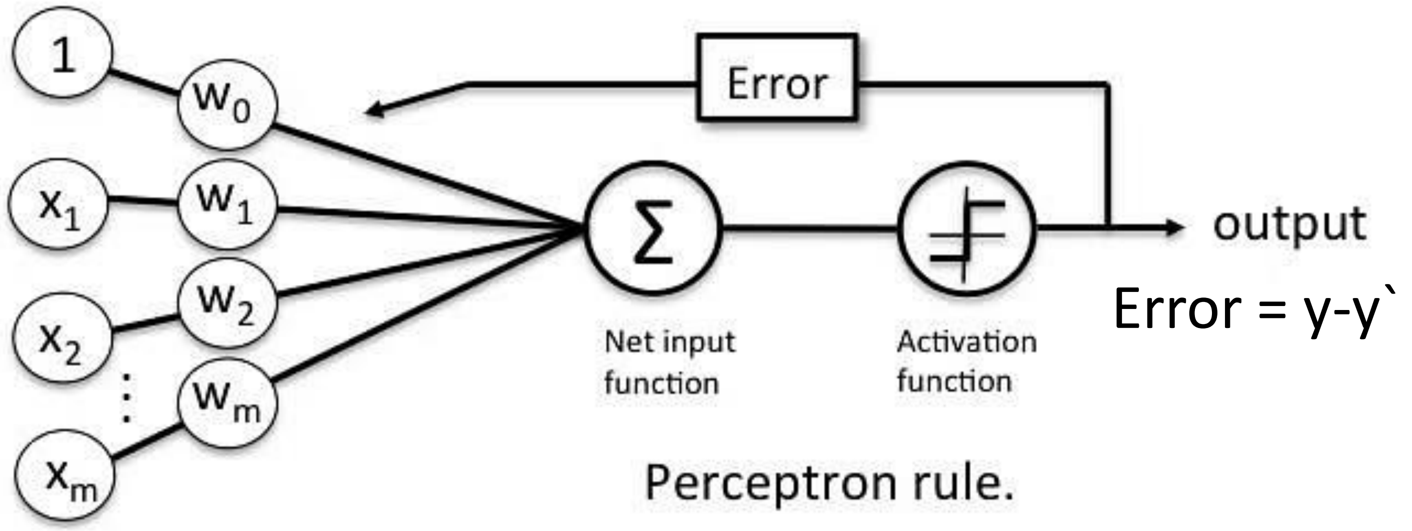
Additional reading on bias

- An additional parameter added to this sum before an activation function. Here’s why the bias term is crucial:
- **Handling Zero Inputs:** If all input features to a neuron are **zero**, without a bias term, the neuron would output zero due to the multiplication by zero weights (assuming no activation function changes this). This might not be desirable, especially if there could be meaningful outputs when inputs are zero. The bias term allows the neuron to output a non-zero value even when all inputs are zero. It provides an offset or a baseline activation level that the neuron can output regardless of the inputs.
- **Dealing with Missing Inputs**: In situations where there is **no input** (missing data), the bias term ensures that the neuron can still activate and contribute to the network’s output. Without a bias term, the absence of input might cause the neuron to remain **inactive** (outputting zero), which could lead to poor model performance or difficulties in training.
- **Learning Flexibility:** The bias term is a trainable parameter like the weights. During the training process, the neural network adjusts the bias term along with the weights to optimize the model’s performance. This flexibility allows the network to learn and adapt to different types of data and input conditions.

HW: Bias update equation?

Single layer Perceptron training and error

- The **Perceptron learning rule**, also known simply as the **Perceptron rule** or the **Delta rule**, is a supervised learning algorithm used for training a single-layer perceptron.
- It **is designed to adjust the weights** of the perceptron in response to misclassifications, **aiming to correctly classify** input patterns.



Perceptron rule.

Single layer Perceptron training and error (1/2)

- Here’s a step-by-step explanation of the **Perceptron learning rule**:
- Initialization:
  - **Initialize the weights** w=(w1,w2,…,wn) and the **bias** $\boldsymbol{b}$ to small random values or zeros.n represents the number of input features.
  - Provide the perceptron with an **input vector** x=(x1,x2,…,xn).
- Compute the **weighted sum** z=w⋅x+bz=w⋅x+b.
- Activation function:
  - **Apply an activation function** $\phi(z)$ to the weighted sum z.
  - The activation function typically used in a perceptron is a step function:

Error = y-y`

Single layer Perceptron training and error (2/2)

- Update rule:
  - Calculate the **error** $e=y-y$^ , where $\boldsymbol{y}$ **is the true** label (either 1 or 0 for binary classification) and $\boldsymbol{y}$**^ is the predicted** output from the activation function.
  - **Adjust the weights** and **bias** using the update rule:
    - $wj \leftarrow wj+\alpha \cdot e \cdot xj$
    - $b \leftarrow b+\alpha \cdot e$, where α is the learning rate, xj is the j-th component of the input vector x, and e is the error.
- Repeat:
  - Repeat steps 2 to 4 for each training example in the dataset.
  - **Continue iterating through the dataset multiple** times until the perceptron **converges** or until a **stopping criterion** is met (e.g., reaching a desired level of accuracy).

Single layer Perceptron training and error

- Worked Example of training a perceptron for AND gate:
  - Input: x=(1,0)
  - Current weights: w=(0.2,−0.1)
  - bias = 0
  - Learning rate: η=0.1
  - Target output: y^=0

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

- **Step 1:** Prediction using LTU:
  - y=step(w·x) = step(0.2∗1 + (−0.1∗0)) = step(0.2) = **1** **(is prediction correct?)**
- **Step 2:** Update Weights
  - For w_1

    :w_1 (next) = $w_1 + \eta(y\text{\^{}} - y)x_1$ = 0.2 + 0.1(0−1)(1) = 0.2 − 0.1 = **0.1**
  - For w_2:

    w_2 (next) = −0.1 + 0.1 (0−1) (0) = **−0.1**

New weights = (0.1, -0.1)

$$wj \leftarrow wj + \alpha \cdot e \cdot xj$$

$$b \leftarrow b + \alpha \cdot e$$

where α is the learning rate, xj is the j-th component of the input vector x, and e is the error.

Single layer Perceptron training and error

- Worked Example of training a perceptron for AND gate:
- Calculate predictions and update weights step by step:

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(\hat{y}_j - y_j\right)x_i$$

Input(X_1,X_2)	Actual output	Weighted sum	Predicted output	Error (Actual – Predicted output)	Updated W_1	Updated W_2	Updated Bias
(0,0)							
(0,1)							
(1,0)	0	0.2	1	-1	0.1	-0.1	?
(1,1)							

HW: Repeat this process to find final weight values (w1, and w2) which can be used to provide the same predicted and actual output.

Summary

- Biological Perceptron
- Discussed basic concepts of ANN
- Single Layer Perception
- Training and error for SLP