Introduction to Deep Learning

Lecture 9,10

Marwa Al-Hadi

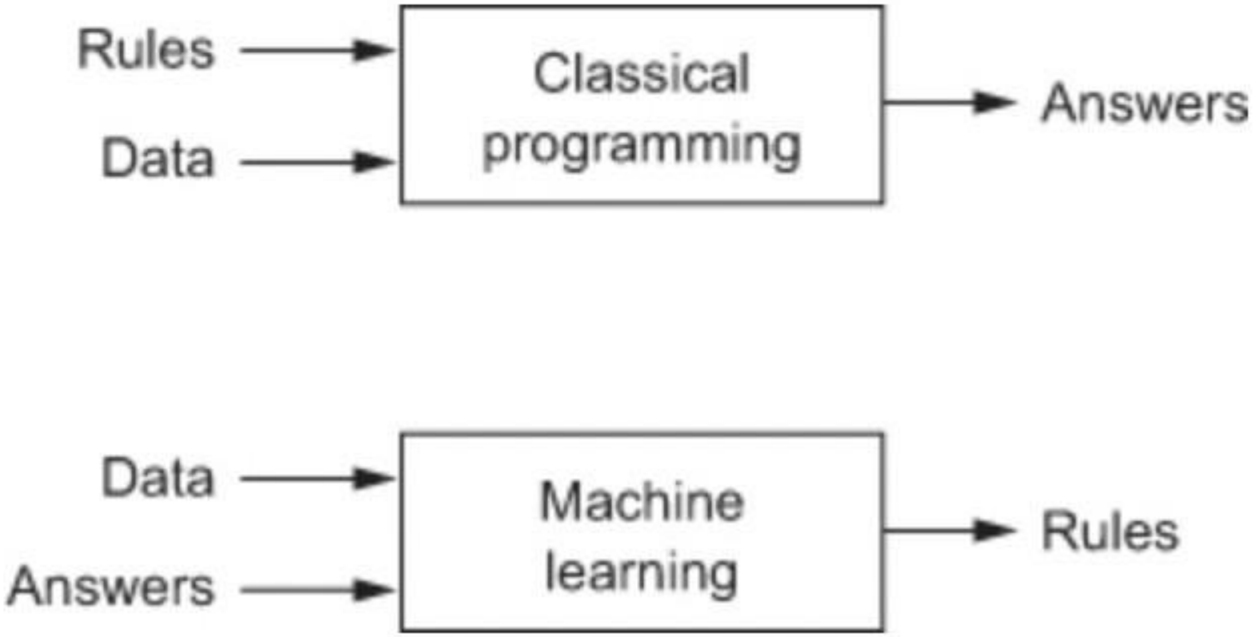
Data science final product

1. Data preprocessing
2. Feature engineering
3. Data balancing
4. feature selection
5. Deep learning or machine learning algorithm
6. Final production (interface or mobile app)

What is Deep Learning?

- Deep Learning is
  - a subset of Machine Learning that
  - uses mathematical functions to map the input to the output.
  - These functions can extract non-redundant information or patterns from the data, which enables them to
  - form a relationship between the input and the output.
  - This is **known** as learning, and the process of learning is called training.

Programming Patterns



Deep Learning

- Modern deep learning models
  - use artificial neural networks or
  - simply neural networks to extract information.

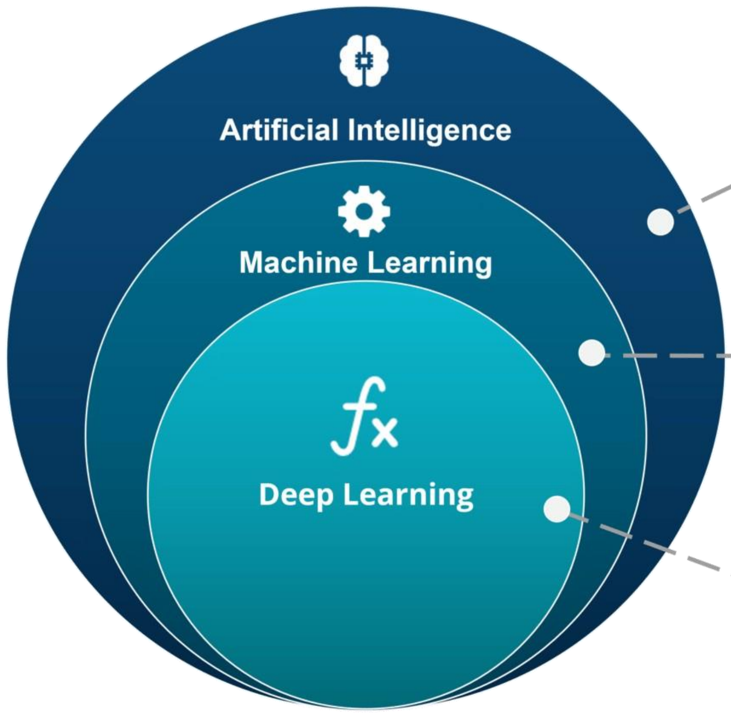
Deep Learning

- These neural networks are **made up** of
  - a simple mathematical function that can be stacked on top of each other and
  - arranged in the form of layers,
  - giving them a sense of depth, hence the term **Deep Learning.**

Deep Learning

- Deep learning can also be
  - thought of as an approach to Artificial Intelligence,
  - a smart combination of hardware and software to solve tasks requiring human intelligence.

Deep Learning



ARTIFICIAL INTELLIGENCE
A technique which enables machines to mimic human behaviour

MACHINE LEARNING
Subset of AI technique which use statistical methods to enable machines to improve with experience

DEEP LEARNING
Subset of ML which make the computation of multi-layer neural network feasible

Deep Learning

- Deep Learning was
  - first theorized in the 1980s, but
  - it has only become useful recently because:
    - It requires large amounts of labeled data
    - It requires significant computational power (high performing GPUs)

Neuron and deep learning

neurons are the core building blocks of deep learning models.

In deep learning, neurons are organized into layers that make up neural networks, which are designed to learn complex patterns and representations from data.

Neuron and deep learning

Here’s how neurons play a central role in deep learning:

1. **Neurons as the Basis of Neural Networks:**

In deep learning,

2. neural networks consist of multiple layers of neurons.

These layers are categorized into:

•**Input layer**: Where the data is fed into the network.

•**Hidden layers**: Where computations are performed using neurons.

•**Output layer**: Where the result of the computations is output.

Neuron and deep learning

- **Each neuron** in a hidden layer
  - receives inputs,
  - applies weights and biases
  - , passes the result through an activation function,
  - and produces an output.
- This **process** is repeated across all layers in the network,
- allowing the model to learn intricate features from the data.

A neuron

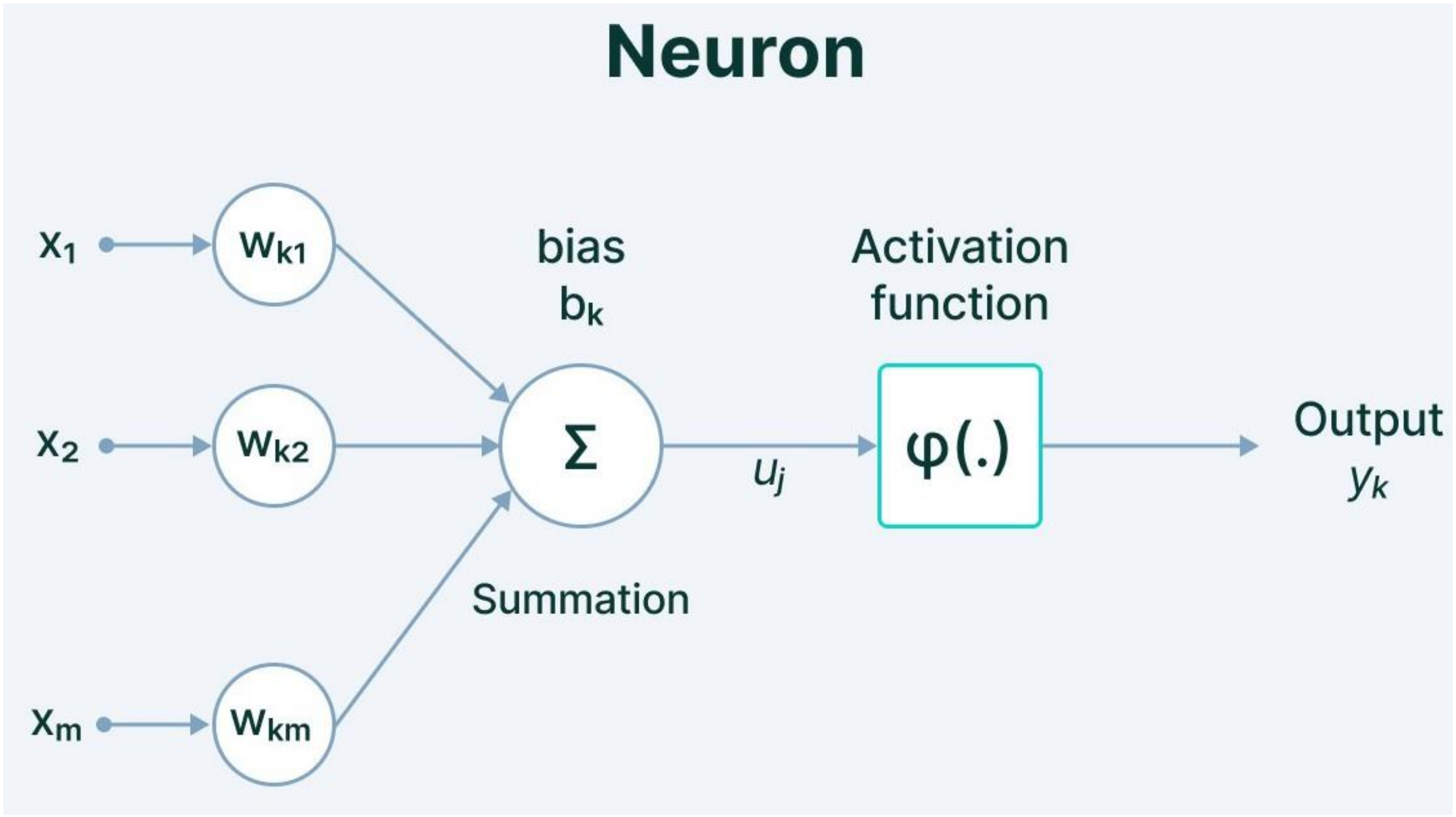
A **neuron** is the basic unit of a neural network, inspired by the biological neurons in the human brain. It takes in input, processes it, and produces an output, which is then passed to other neurons in the network.

A neuron

In the **context of artificial neural networks (ANNs)**, the main components of a neuron are:

1. **Input:** The data received by the neuron (can be from the dataset or previous neurons).
2. **Weights:** Each **input** has an associated weight, which determines the importance of that input.
3. **Bias:** A value added to the weighted sum of inputs to shift the activation function.
4. **Summation:** The weighted sum of inputs plus the bias.
5. **Activation Function:** A function applied to the summed value to produce the output. This helps to introduce non-linearity into the model.

Neuron


Neuron
x_1
x_2
x_m
w_{k1}
w_{k2}
w_{km}
bias
b_k
Σ
Summation
u_j
Activation function
$\varphi(.)$
Output
y_k

Example of computes a weight of neuron

- Inputs: $x_1 = 2, x_2 = 3$
- Weights: $w_1 = 0.5, w_2 = 0.6$
- Bias: $b = 0.1$

1. **Calculate the Weighted Sum:**

The first step in the neuron is to compute the **weighted sum** of inputs, which is calculated as:

$$\text{Weighted Sum} = w_1 \cdot x_1 + w_2 \cdot x_2 + b$$

Substitute the given values:

$$\text{Weighted Sum} = 0.5 \cdot 2 + 0.6 \cdot 3 + 0.1 = 1 + 1.8 + 0.1 = 2.9$$

Until here perceptron

Con..

2. Apply the Activation Function:

Next, we pass the weighted sum through the **activation function**. We'll use the **sigmoid function**, which is defined as:

From here its neuron

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

Substitute the weighted sum of 2.9 into the sigmoid function:

$$\sigma(2.9) = \frac{1}{1 + e^{-2.9}} \approx \frac{1}{1 + 0.055} \approx \frac{1}{1.055} \approx 0.947$$

3. Output:

The output of the neuron is approximately **0.947**.

Final Output:

$$\text{Output} = 0.947$$

Neuron

- The **neuronal perception of deep learning** is generally **motivated by two main ideas:**
  - It is assumed that the human brain proves that intelligent behavior is possible, and—by **reverse engineering**, it is possible to build an intelligent system

Neuron

- The **neuronal perception of deep learning** is generally **motivated by two main ideas:**
  - Another perspective is that to understand the working of the human brain and the principles that underlie its intelligence is to build a mathematical model that could shed light on the fundamental scientific questions.

Neuron and perceptron

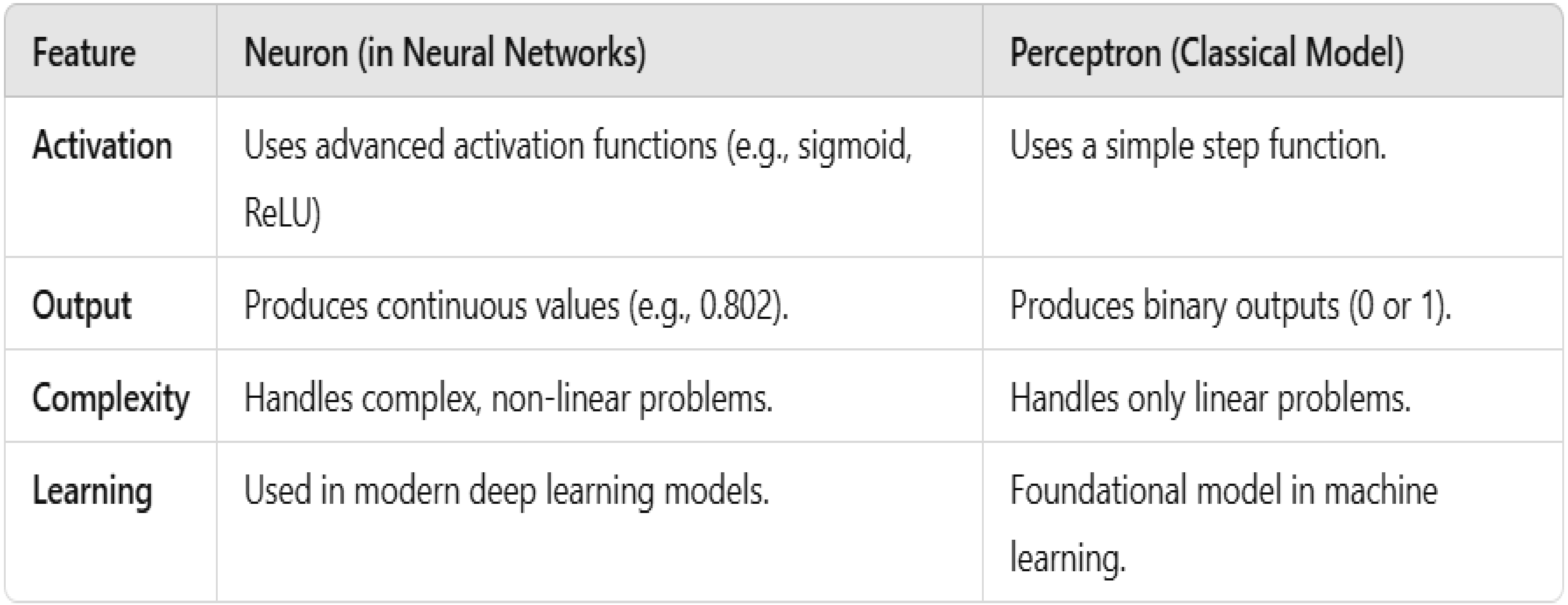
A **neuron** is

- a more advanced and flexible computational unit used in deep learning.

A perceptron is

- a basic building block of early neural networks, primarily used for simple binary classification tasks.

Key difference



Feature	Neuron (in Neural Networks)	Perceptron (Classical Model)
Activation	Uses advanced activation functions (e.g., sigmoid, ReLU)	Uses a simple step function.
Output	Produces continuous values (e.g., 0.802).	Produces binary outputs (0 or 1).
Complexity	Handles complex, non-linear problems.	Handles only linear problems.
Learning	Used in modern deep learning models.	Foundational model in machine learning.

Example of perceptron

Imagine a perceptron that determines whether the sum of weighted inputs exceeds a threshold.

1. **Inputs**: $x_1 = 1$, $x_2 = -2$
2. **Weights**: $w_1 = 2$, $w_2 = 3$
3. **Bias**: $b = -4$
4. **Threshold/Step Function**: $\text{output} = 1$ if $z > 0$, else 0

The computation:

$$z = w_1 x_1 + w_2 x_2 + b$$

$$z = (2)(1) + (3)(-2) + (-4) = 2 - 6 - 4 = -8$$

Apply the step function:

- If $z > 0$, output 1
- Since $z = -8$, output 0

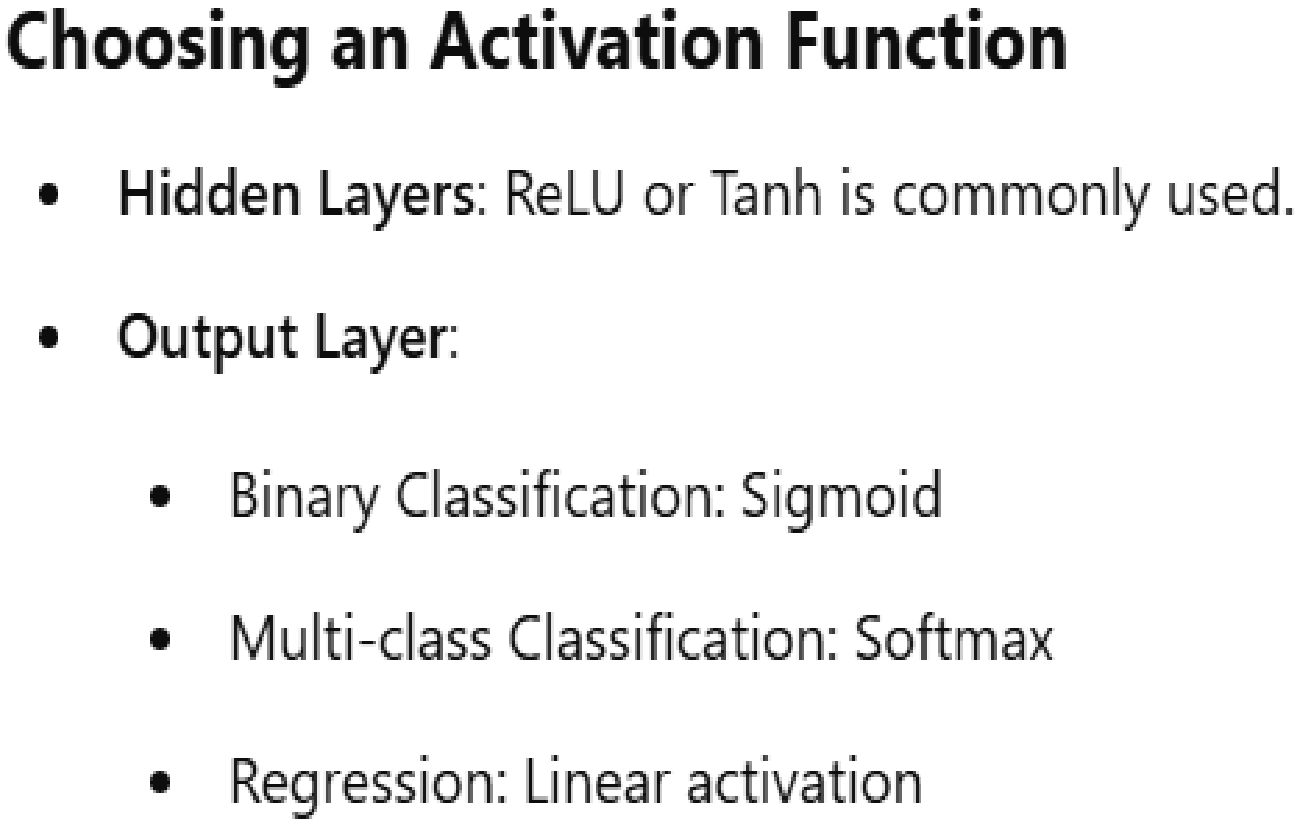
Output: 0

Activation functions

An **activation function** is

- a mathematical function applied to the output of a neuron in a neural network.
- It determines whether a neuron should be activated (pass its output forward) and introduces non-linearity into the model,
- enabling it to learn complex patterns.

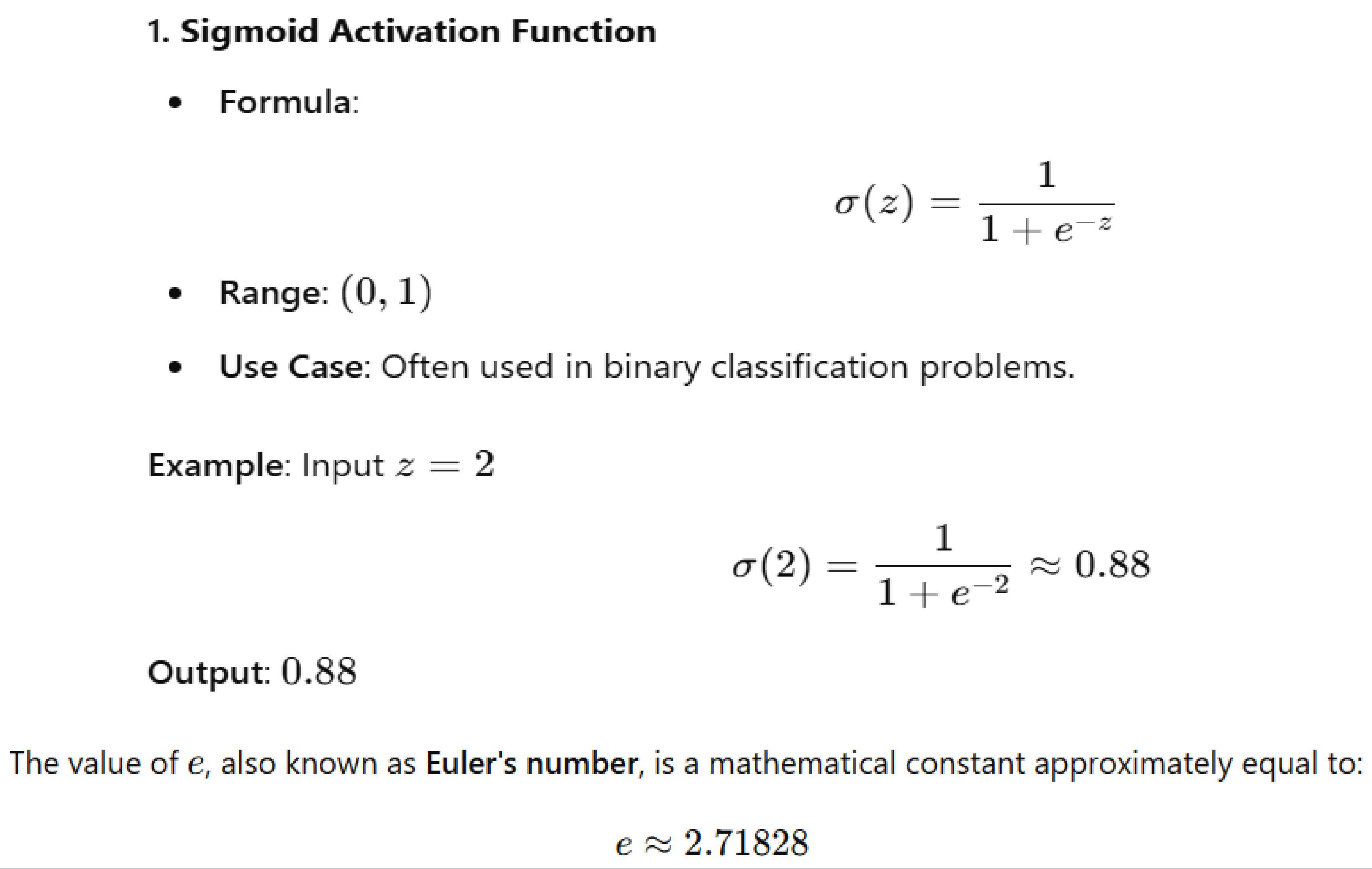
Choosing an activation functions



Choosing an Activation Function

- **Hidden Layers**: ReLU or Tanh is commonly used.
- **Output Layer**:
  - Binary Classification: Sigmoid
  - Multi-class Classification: Softmax
  - Regression: Linear activation

Con..



1. **Sigmoid Activation Function**

- **Formula**:

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

- **Range**: $(0, 1)$
- **Use Case**: Often used in binary classification problems.

Example: Input $z = 2$

$$\sigma(2) = \frac{1}{1 + e^{-2}} \approx 0.88$$

Output: 0.88

The value of e, also known as **Euler's number**, is a mathematical constant approximately equal to:

$$e \approx 2.71828$$

Con..

2. ReLU (Rectified Linear Unit)

- **Formula**:

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

- **Range**: $[0, \infty)$
- **Use Case**: Commonly used in hidden layers of deep neural networks.

Example: Input $z = -3$, $z = 4$

$$f(-3) = \max(0, -3) = 0$$

$$f(4) = \max(0, 4) = 4$$

Output: 0 for $z = -3$, 4 for $z = 4$

Con..

3. Tanh (Hyperbolic Tangent)

- Formula:

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

- **Range**: $(-1, 1)$
- **Use Case**: Often used in hidden layers where values need to center around zero.

Example: Input $z = 1$

$$\tanh(1) = \frac{e^1 - e^{-1}}{e^1 + e^{-1}} \approx 0.76$$

Output: 0.76

Con..

4. Softmax

- **Formula**:

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- **Range**: $(0, 1)$ (normalized probabilities)
- **Use Case**: Typically used in the output layer of classification tasks to produce probabilities for each class.

Example: For inputs $z = [1, 2, 3]$:

$$f(1) = \frac{e^1}{e^1 + e^2 + e^3} \approx 0.09$$

$$f(2) = \frac{e^2}{e^1 + e^2 + e^3} \approx 0.24$$

$$f(3) = \frac{e^3}{e^1 + e^2 + e^3} \approx 0.67$$

Output: Probabilities: $[0.09, 0.24, 0.67]$

Deep Learning vs. Machine Learning

- **Deep Learning** can essentially do everything that machine learning does, but not the other way
- around.

  For instance, **machine learning** is useful when the
- dataset is small, which means that the data is carefully preprocessed.

Deep Learning vs. Machine Learning

- **Data preprocessing** requires human intervention. It also means that when the dataset is large and complex, **machine learning** algorithms will fail to extract information, and it will underfit.

Deep Learning vs. Machine Learning

- Generally, **machine learning** is alternatively termed shallow learning because it is very effective for smaller datasets.

  Deep learning, on the other hand, is extremely powerful when the dataset is large.

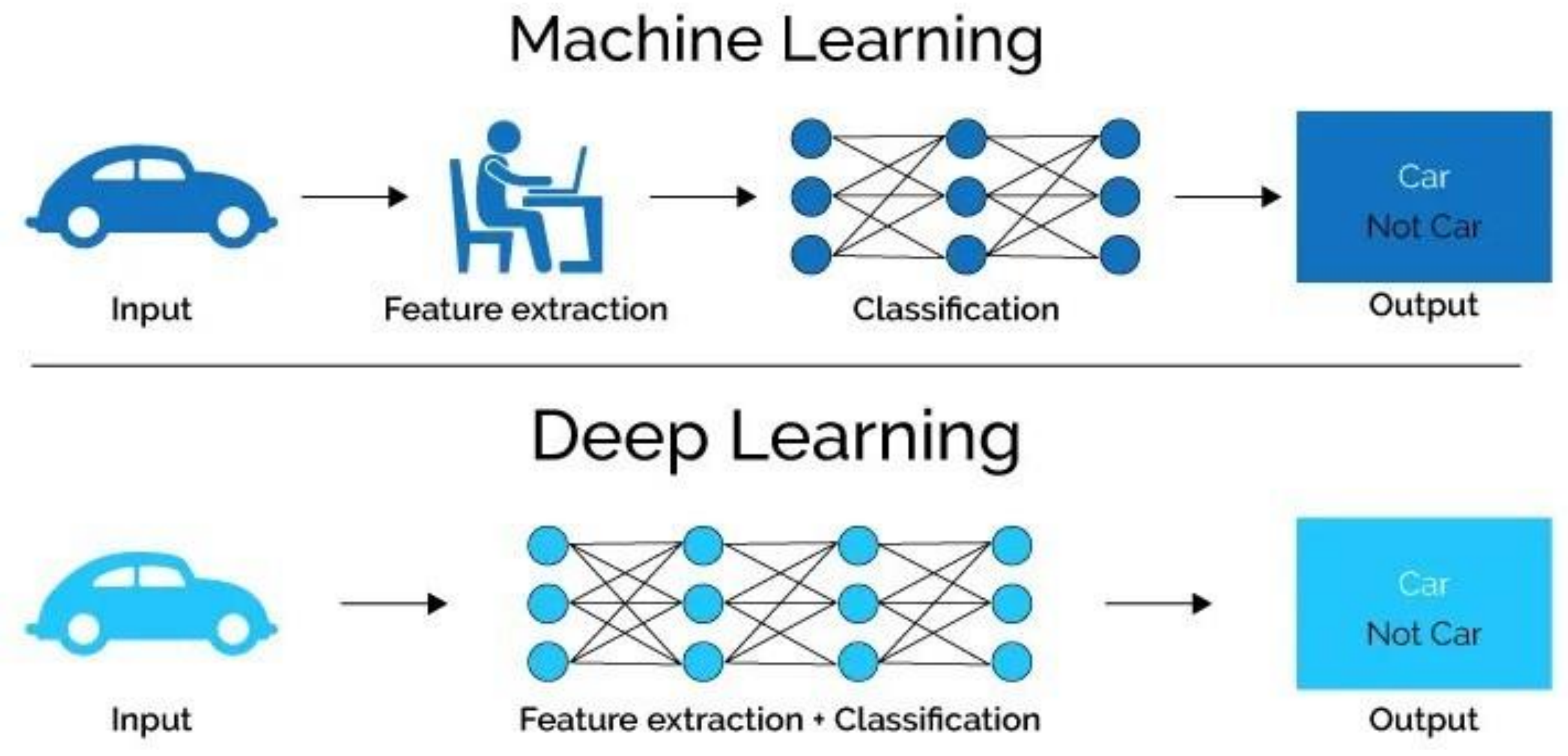
Deep Learning vs. Machine Learning

Deep learning can learn any complex patterns from the data and can draw accurate conclusions on its own. In fact, deep learning is so powerful that it can even process unstructured data (social media activity, etc.)

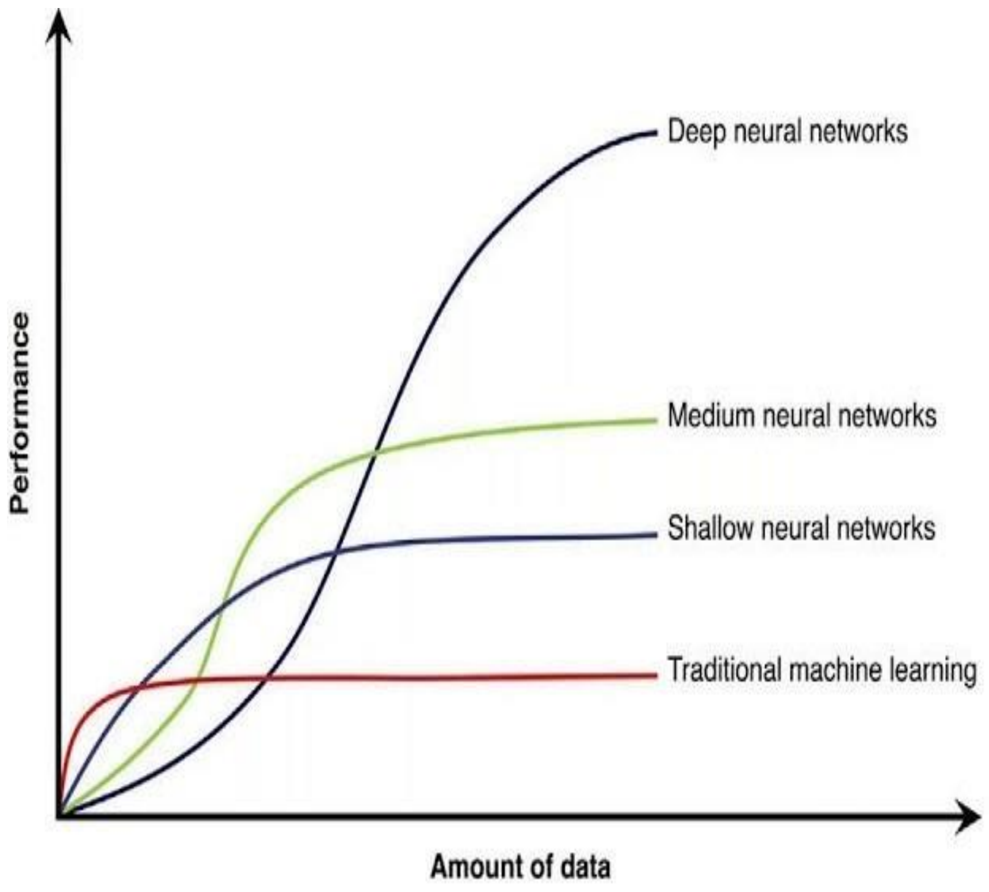
Deep Learning vs. Machine Learning

- Furthermore, **deep learning** can also generate new data samples and find anomalies that machine learning algorithms and human eyes can miss.

Deep Learning vs. Machine Learning



Why Deep Learning ?



Why Now?

- *Algorithm Advancements*
- *GPU Computing*
- *Availability of Larger Training Data*

Deep Learning vs. Machine Learning

- On the downside, **deep learning** is computationally expensive compared to machine learning, which also means that it requires a lot of time to process.

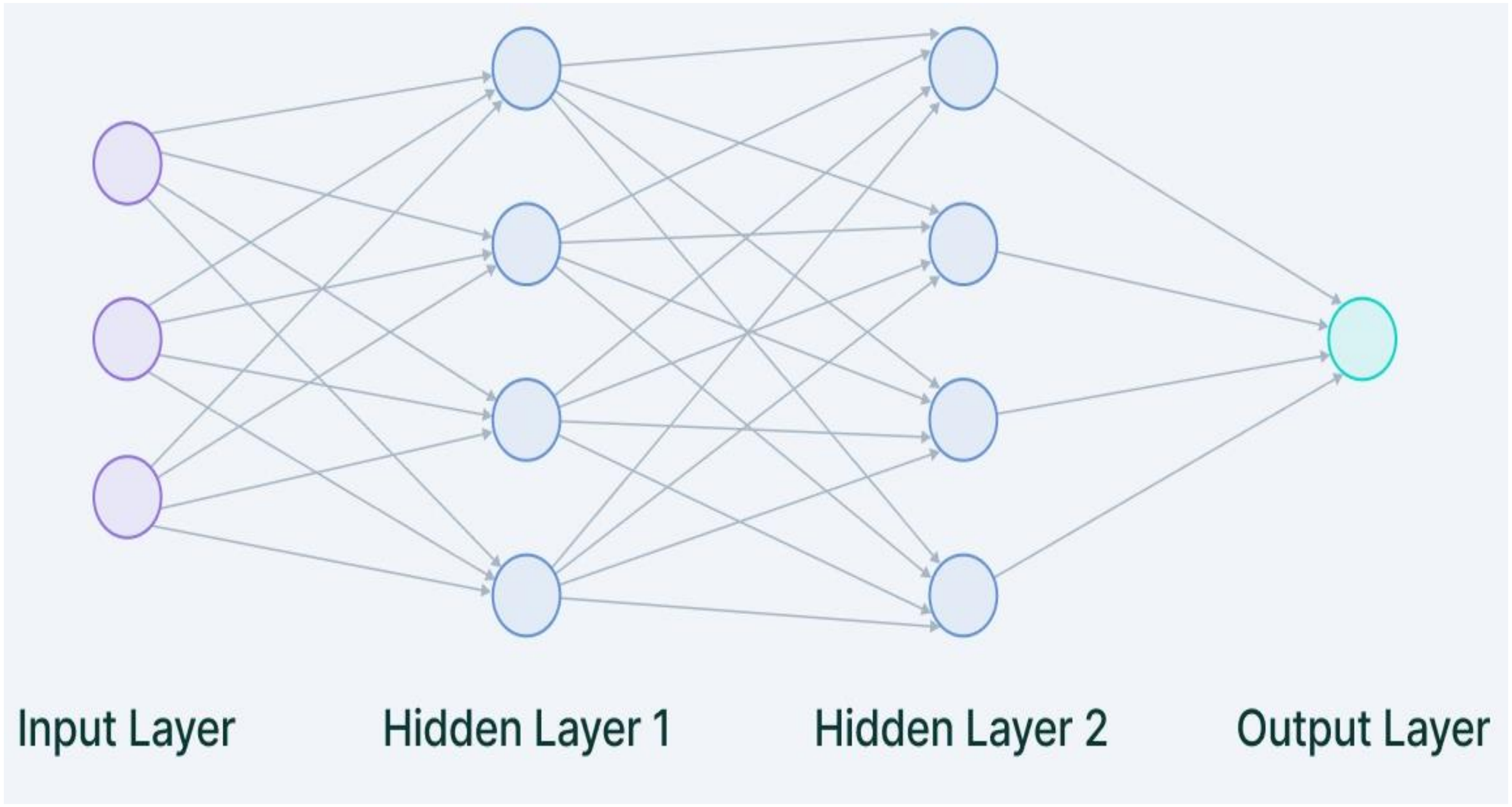
Deep Learning vs. Machine Learning

-
- **Deep Learning and Machine Learning** are both **capable of different types of learning**: Supervised Learning (labeled data), Unsupervised Learning (unlabeled data), and Reinforcement Learning.

How does Deep Learning work?

- **Deep Neural Networks**
  - have multiple layers of interconnected artificial neurons or nodes that are stacked together.
- - Each of these nodes has a simple mathematical function - usually a linear function that performs extraction and mapping of information.
-
  - There are three layers to a deep neural network: the input layer, hidden layers, and the output layer.

How does Deep Learning work?



Types of Neural Network

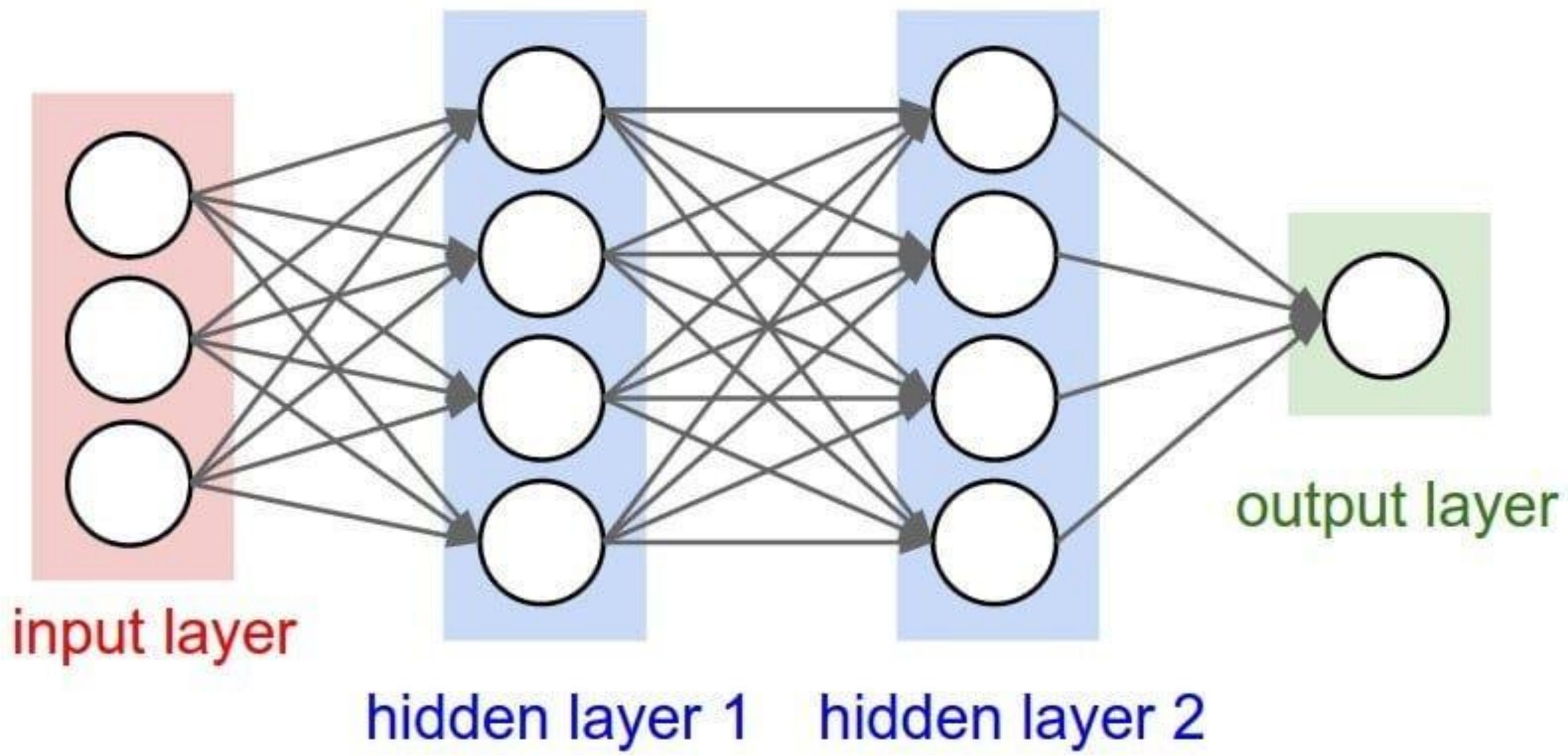
- Artificial Neural Network
- Convolutional Neural Network
- Recurrent Neural Network
- Long short term Memory

ANN

An **Artificial Neural Network (ANN)** is a

- computational model inspired by the structure and functioning of the biological brain.
- It is made up of interconnected units called **neurons**, which process information collectively
- to solve complex problems like classification, regression, and clustering.

ANN

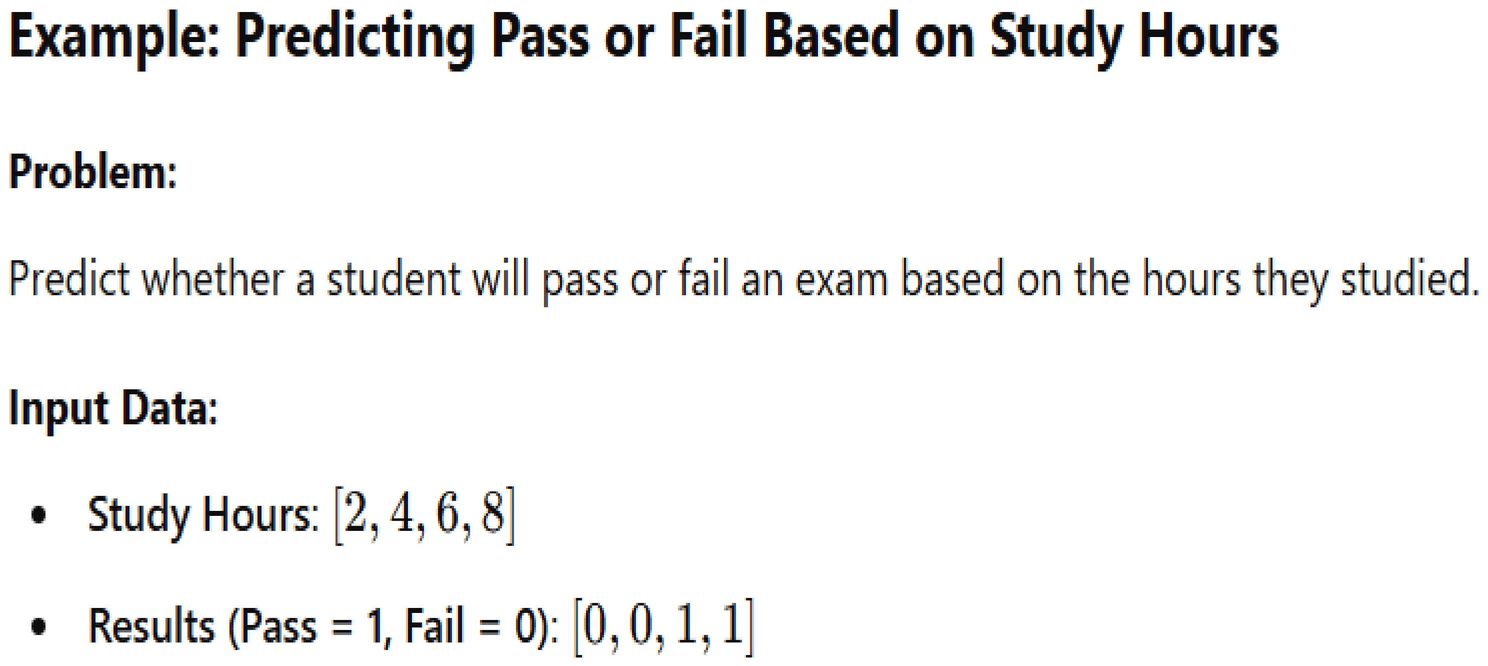


ANN

Components of ANN

1. **Input Layer**: Accepts the input features (data).
2. **Hidden Layers**: Intermediate layers where computations and transformations occur, enabling the network to learn complex patterns.
3. **Output Layer**: Produces the final prediction or output.
4. **Weights and Biases**: Parameters that the model adjusts during training to minimize errors.
5. **Activation Functions**: Introduce non-linearity to the model, enabling it to learn complex relationships.

ANN



Example: Predicting Pass or Fail Based on Study Hours

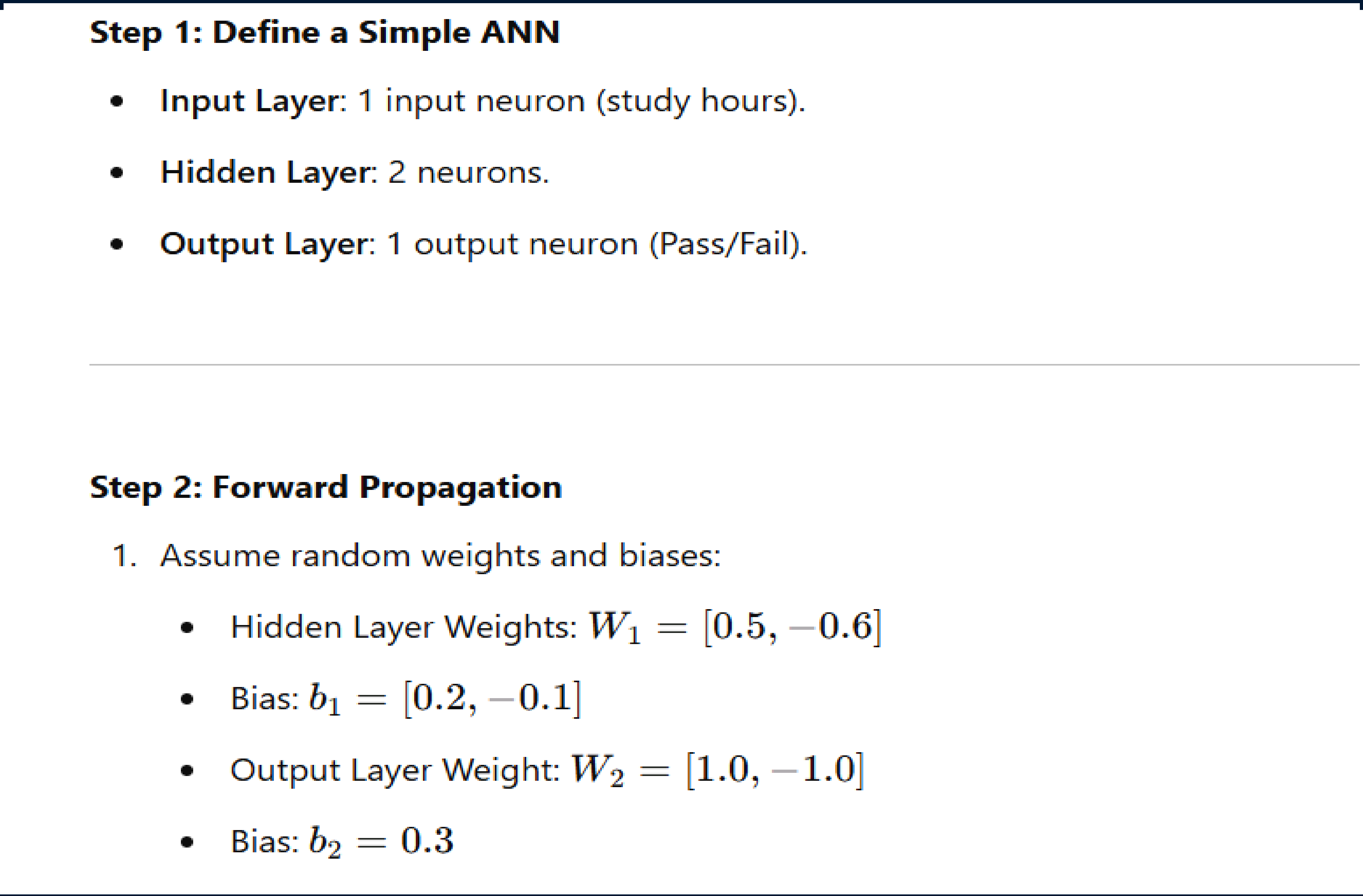
Problem:

Predict whether a student will pass or fail an exam based on the hours they studied.

Input Data:

- Study Hours: $[2, 4, 6, 8]$
- Results (Pass = 1, Fail = 0): $[0, 0, 1, 1]$

ANN



Step 1: Define a Simple ANN

- **Input Layer**: 1 input neuron (study hours).
- **Hidden Layer**: 2 neurons.
- **Output Layer**: 1 output neuron (Pass/Fail).

Step 2: Forward Propagation

1. Assume random weights and biases:
   - Hidden Layer Weights: $W_1 = [0.5, -0.6]$
   - Bias: $b_1 = [0.2, -0.1]$
   - Output Layer Weight: $W_2 = [1.0, -1.0]$
   - Bias: $b_2 = 0.3$

ANN

2. For an input $x = 6$:
   - Compute hidden layer activations:

$$z_1 = (0.5 \cdot 6) + 0.2 = 3.2, \quad a_1 = \mathrm{ReLU}(3.2) = 3.2$$

$$z_2 = (-0.6 \cdot 6) - 0.1 = -3.7, \quad a_2 = \mathrm{ReLU}(-3.7) = 0$$

   - Compute output layer activation:

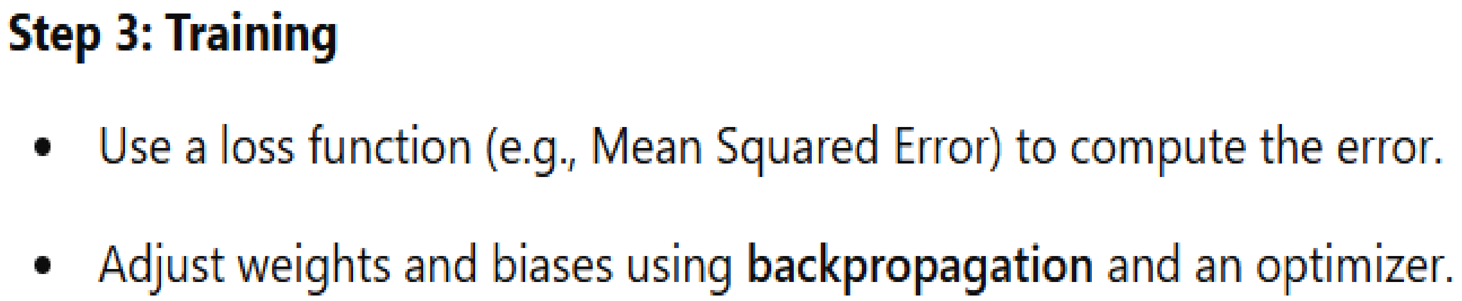
$$z_{\text{out}} = (1.0 \cdot 3.2) + (-1.0 \cdot 0) + 0.3 = 3.5$$

   Apply activation function (Sigmoid):

$$y = \sigma(3.5) = \frac{1}{1 + e^{-3.5}} \approx 0.97$$

3. Predicted Result:
   - $y = 0.97$ → Pass.

ANN



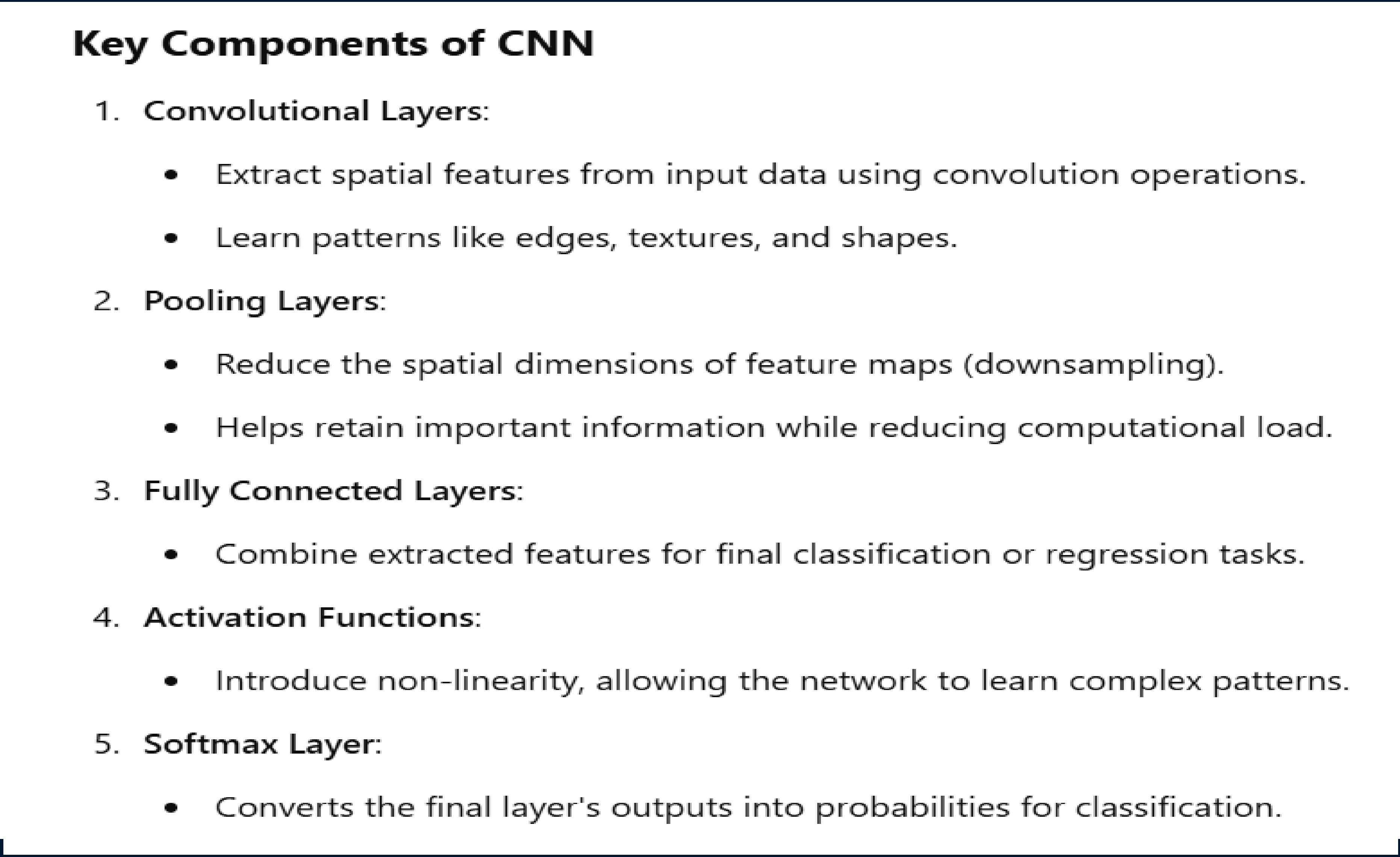
Step 3: Training

- Use a loss function (e.g., Mean Squared Error) to compute the error.
- Adjust weights and biases using **backpropagation** and an optimizer.

CNN

- The **Convolutional Neural Networks** or CNNs are primarily used for tasks related to computer vision or image processing.
- CNNs are extremely good in modeling spatial data such as 2D or 3D images and videos.
- They can extract features and patterns within an image, enabling tasks such as image classification or object detection.

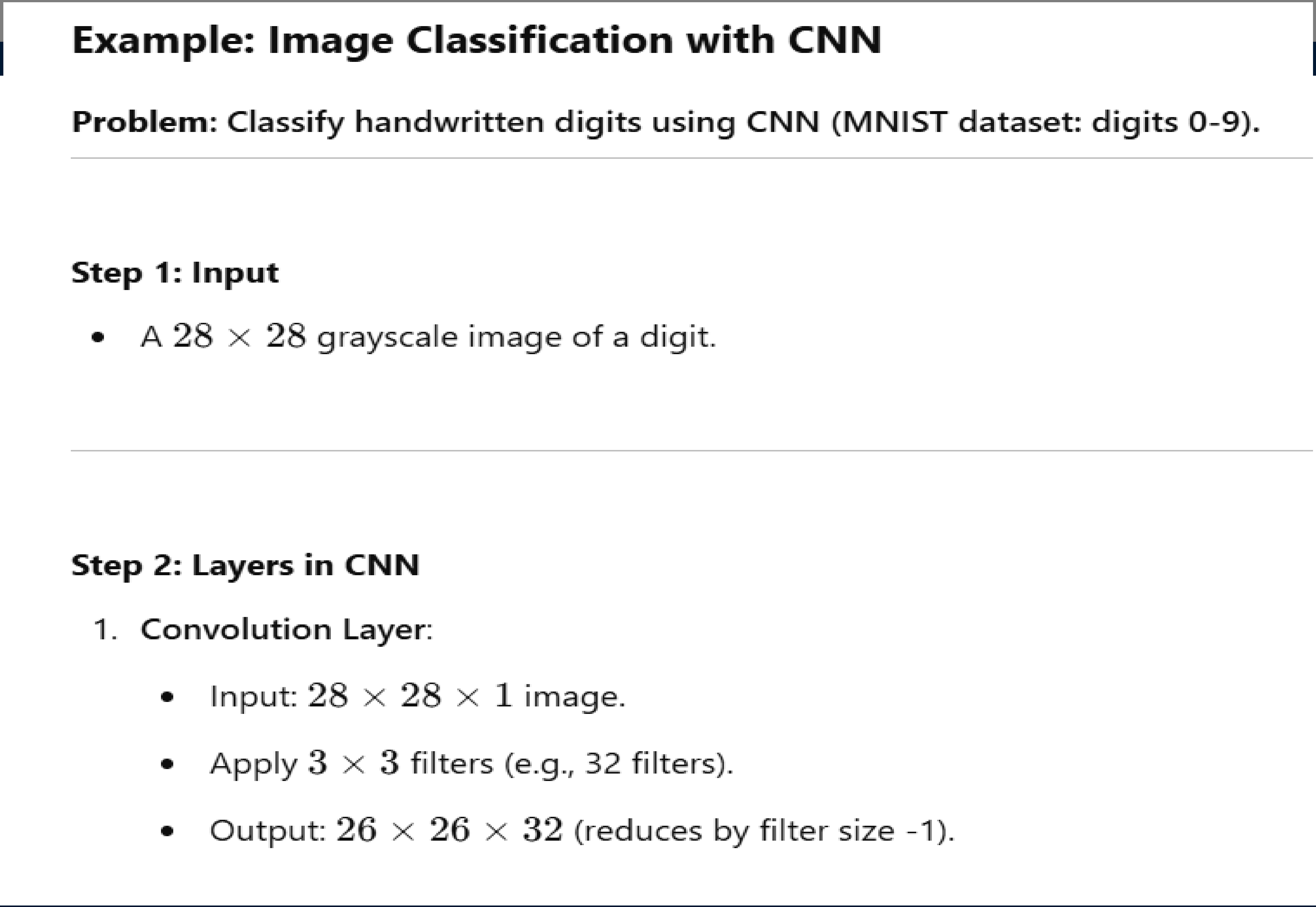
CNN



Key Components of CNN

1. **Convolutional Layers**:
   - Extract spatial features from input data using convolution operations.
   - Learn patterns like edges, textures, and shapes.
2. **Pooling Layers**:
   - Reduce the spatial dimensions of feature maps (downsampling).
   - Helps retain important information while reducing computational load.
3. **Fully Connected Layers**:
   - Combine extracted features for final classification or regression tasks.
4. **Activation Functions**:
   - Introduce non-linearity, allowing the network to learn complex patterns.
5. **Softmax Layer**:
   - Converts the final layer's outputs into probabilities for classification.

CNN



Example: Image Classification with CNN

Problem: Classify handwritten digits using CNN (MNIST dataset: digits 0-9).

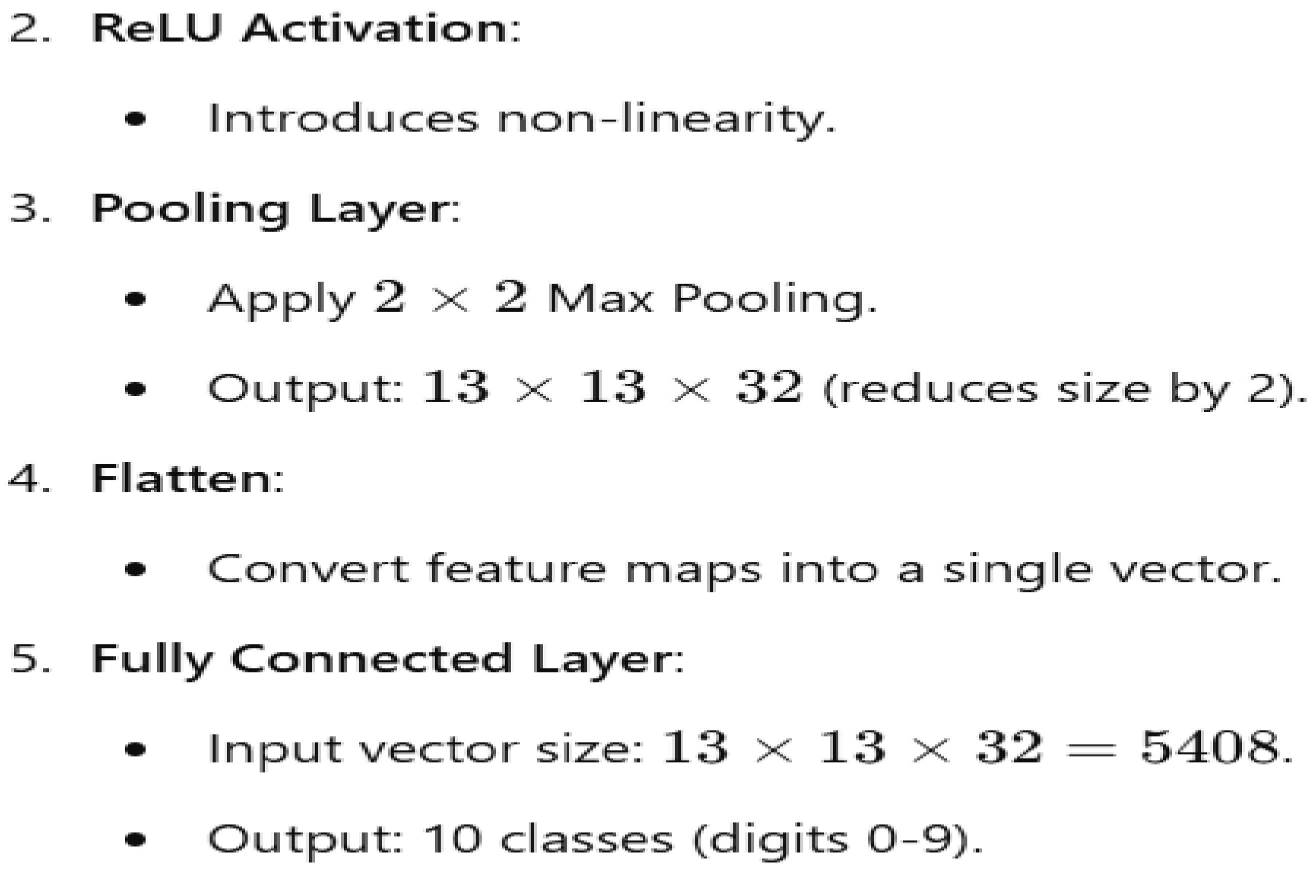
Step 1: Input

- A 28×28 grayscale image of a digit.

Step 2: Layers in CNN

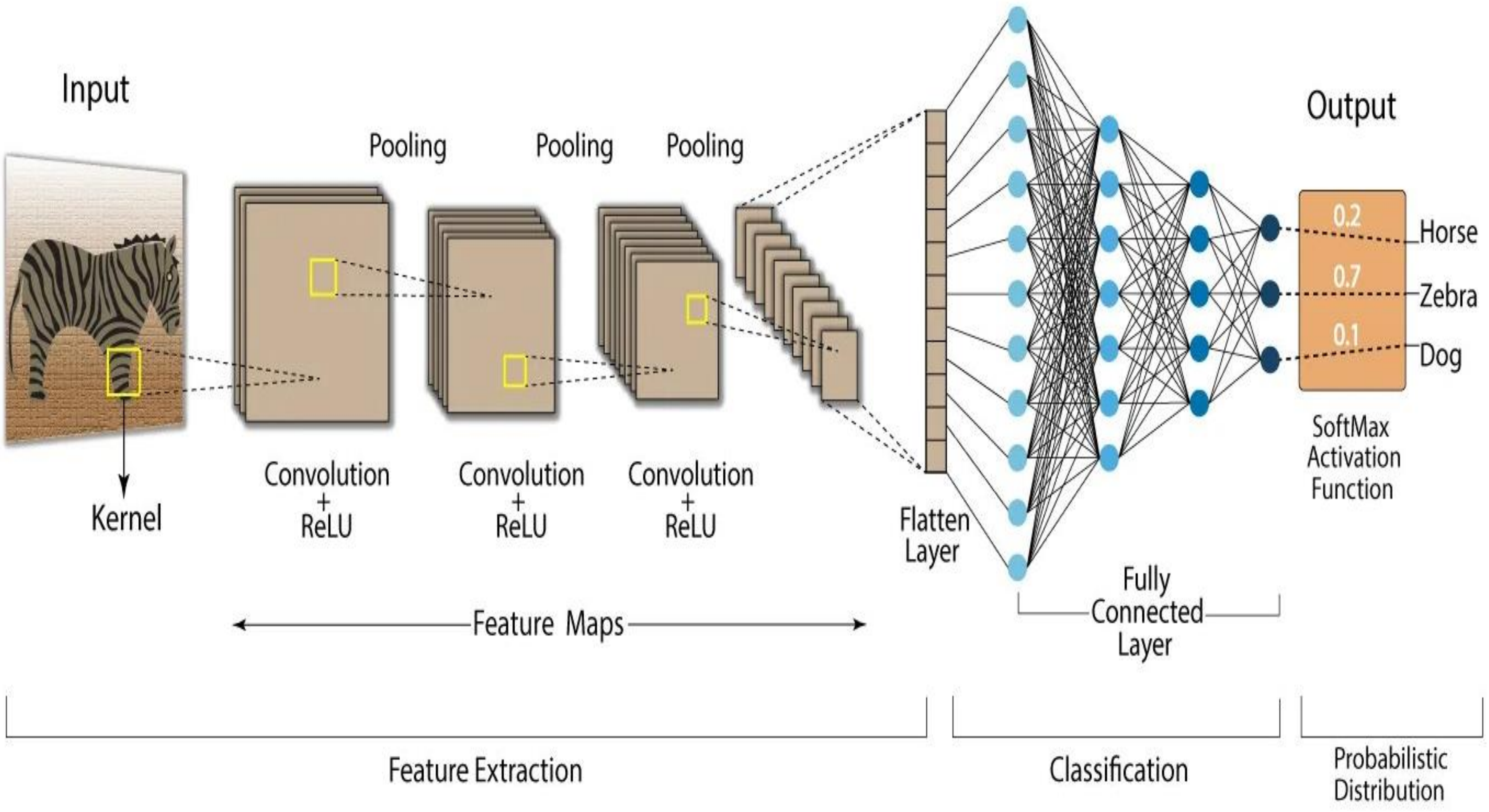
1. **Convolution Layer**:
    - Input: $28 \times 28 \times 1$ image.
    - Apply 3×3 filters (e.g., 32 filters).
    - Output: $26 \times 26 \times 32$ (reduces by filter size -1).

CNN



2. **ReLU Activation**:
   - Introduces non-linearity.
3. **Pooling Layer**:
   - Apply 2×2 Max Pooling.
   - Output: $13 \times 13 \times 32$ (reduces size by 2).
4. **Flatten**:
   - Convert feature maps into a single vector.
5. **Fully Connected Layer**:
   - Input vector size: $13 \times 13 \times 32 = 5408$.
   - Output: 10 classes (digits 0-9).

CNN

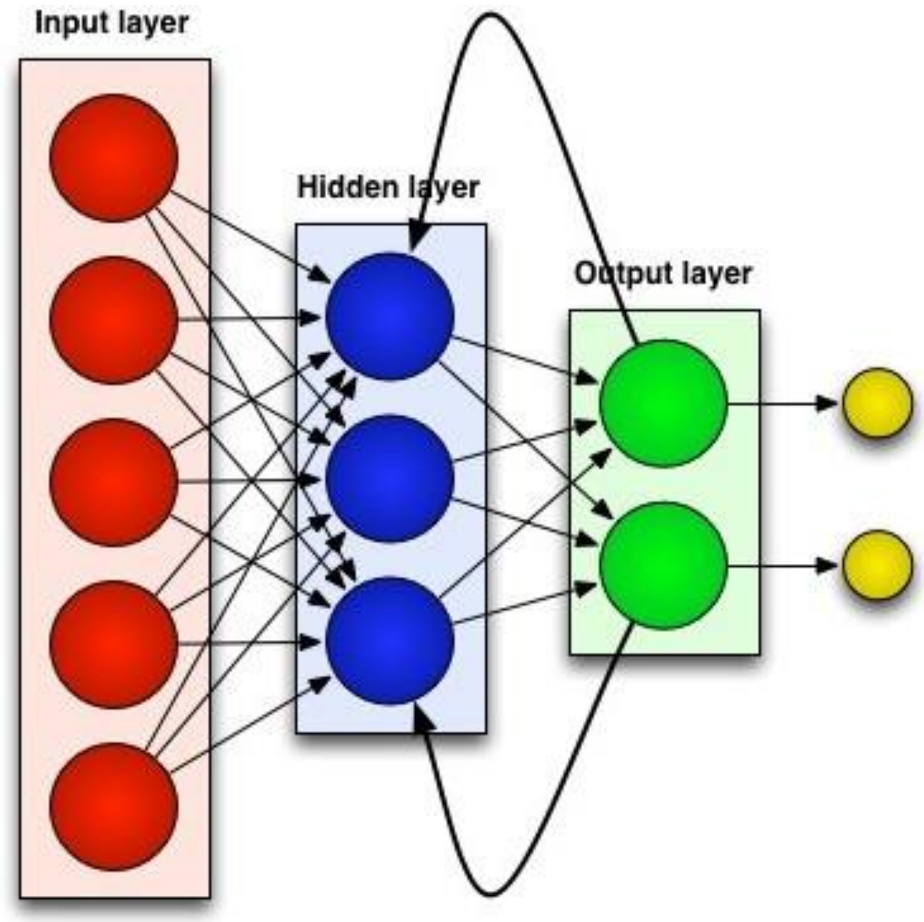

Input
Kernel
Pooling
Pooling
Pooling
Convolution + ReLU
Convolution + ReLU
Convolution + ReLU
Feature Maps
Flatten Layer
Fully Connected Layer
Output
0.2
0.7
0.1
Horse
Zebra
Dog
SoftMax Activation Function
Feature Extraction
Classification
Probabilistic Distribution

RNN

- The Recurrent Neural Networks or RNN are
  - primarily used to model sequential data,
-
  - such as text, audio, or any type of data that

    represents sequence or time.

RNN



RNN

Mathematics Behind RNN

At each time step t:

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

- h_t: Hidden state at time t
- x_t: Input at time t
- W_{xh}: Weight matrix for input to hidden layer
- W_{hh}: Weight matrix for recurrent connection
- W_{hy}: Weight matrix for hidden to output
- b_h, b_y: Bias terms
- y_t: Output at time t
- $\tanh$: Activation function

Problem Statement

Given a sequence of numbers:

Input: $[1, 2, 3]$

Task: Predict the next number in the sequence.

Step-by-Step Explanation

1. **Initialize the RNN**:
   - Hidden state h_t is initialized as $h_0 = 0$.
   - Use a **weight matrix** W, **recurrent weight** U, and **bias** b for computations:
     - $W = 0.5$, $U = 0.8$, $b = 0.1$
2. **Activation Function**:
   - Use the **tanh** activation function:

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

3. **Equations**:
   - At each time step t, compute the hidden state h_t:

$$h_t = \tanh(W \cdot x_t + U \cdot h_{t-1} + b)$$

   - Compute the output y_t:

$$y_t = V \cdot h_t + c$$

   Assume $V = 1$ and $c = 0.1$.

Step-by-Step Calculation

Step 1: Input $x_1 = 1$

- $h_0 = 0$ (initial hidden state)
- Compute h_1:

$$h_1 = \tanh(0.5 \cdot 1 + 0.8 \cdot 0 + 0.1) = \tanh(0.5 + 0 + 0.1) = \tanh(0.6)$$

  Approximate $\tanh(0.6) = 0.537$.

- Compute y_1:

$$y_1 = 1 \cdot 0.537 + 0.1 = 0.637$$

Step 2: Input $x_2 = 2$

- $h_1 = 0.537$ (from previous step)
- Compute h_2:

$$h_2 = \tanh(0.5 \cdot 2 + 0.8 \cdot 0.537 + 0.1) = \tanh(1.0 + 0.4296 + 0.1) = \tanh(1.5296)$$

  Approximate $\tanh(1.5296) = 0.910$.

- Compute y_2:

$$y_2 = 1 \cdot 0.910 + 0.1 = 1.010$$

Step 3: Input $x_3 = 3$

- $h_2 = 0.910$ (from previous step)
- Compute h_3:

$$h_3 = \tanh(0.5 \cdot 3 + 0.8 \cdot 0.910 + 0.1) = \tanh(1.5 + 0.728 + 0.1) = \tanh(2.328)$$

  Approximate $\tanh(2.328) = 0.981$.

- Compute y_3:

$$y_3 = 1 \cdot 0.981 + 0.1 = 1.081$$

- For $x_1 = 1$, the model predicts $y_1 = 0.637$
- For $x_2 = 2$, the model predicts $y_2 = 1.010$
- For $x_3 = 3$, the model predicts $y_3 = 1.081$

Add to the next number in the sequence,
Ex: x1=1+0.6=2

Step 2: Compute y_4

The output at time step $t = 4$ is:

$$y_4 = V \cdot h_4 + c$$

Substitute $V = 1$ and $c = 0.1$:

$$y_4 = 1 \cdot 0.993 + 0.1 = 1.093$$

Prediction for $x_4 = 4$

The predicted output is:

$$y_4 \approx 1.093$$

RNN applications

1. **Natural Language Processing**:
   - Language translation
   - Text generation
   - Sentiment analysis
2. **Time Series Analysis**:
   - Stock price prediction
   - Weather forecasting
3. **Speech and Audio Processing**:
   - Speech recognition
   - Music generation
4. **Healthcare**:
   - ECG signal analysis
   - Patient monitoring

LSTM

LSTM (Long Short-Term Memory) is a

- type of recurrent neural network (RNN) designed to
- capture long-term dependencies in sequential data.

Unlike a standard RNN, which may struggle with long sequences, LSTMs maintain a memory cell that allows them to remember information over extended time steps.

Key Components of LSTM

1. **Memory Cell** (C_t): Stores information that can be updated, added to, or forgotten.
2. **Gates**: Control the flow of information:
    - **Forget Gate** (f_t): Decides what information to forget.
    - **Input Gate** (i_t): Determines what new information to store.
    - **Output Gate** (o_t): Controls what part of the memory cell to output.

At each time step t:

1. **Forget Gate**:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

   Decides what portion of the cell state to forget.

2. **Input Gate**:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

   Updates the cell state with new information.

3. **Cell State Update**:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

4. **Output Gate**:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \odot \tanh(C_t)$$

Here:

- x_t: Input at time t.
- h_{t-1}: Previous hidden state.
- C_t: Current cell state.
- σ: Sigmoid activation function.
- $\tanh$: Hyperbolic tangent activation function.
- $\odot$: Element-wise multiplication.

Problem

Predict the next number in a sequence: **[3, 5, 7, ?]**.

Step 1: Input

- Sequence: $x = [3, 5, 7]$
- Assume hidden size = 2.

Step 2: Initialize Parameters

Random weights for simplicity:

- $W_f = [0.5, -0.3], b_f = 0.1$
- $W_i = [0.7, 0.2], b_i = -0.2$
- $W_o = [0.6, 0.4], b_o = 0.05$
- $W_C = [0.8, -0.1], b_C = 0.0$

Step 3: Time Step Calculations

Time Step 1 ($x_1 = 3$):

1. Forget Gate:

$$f_1 = \sigma(W_f \cdot x_1 + b_f) = \sigma(0.5 \cdot 3 - 0.3 + 0.1) = \sigma(1.3) \approx 0.785$$

2. Input Gate:

$$i_1 = \sigma(W_i \cdot x_1 + b_i) = \sigma(0.7 \cdot 3 + 0.2 - 0.2) = \sigma(2.1) \approx 0.891$$

3. Candidate Cell State:

$$\tilde{C}_1 = \tanh(W_C \cdot x_1 + b_C) = \tanh(0.8 \cdot 3 - 0.1) = \tanh(2.3) \approx 0.980$$

4. Cell State:

$$C_1 = f_1 \cdot C_0 + i_1 \cdot \tilde{C}_1 = 0.785 \cdot 0 + 0.891 \cdot 0.980 \approx 0.873$$

5. Output Gate:

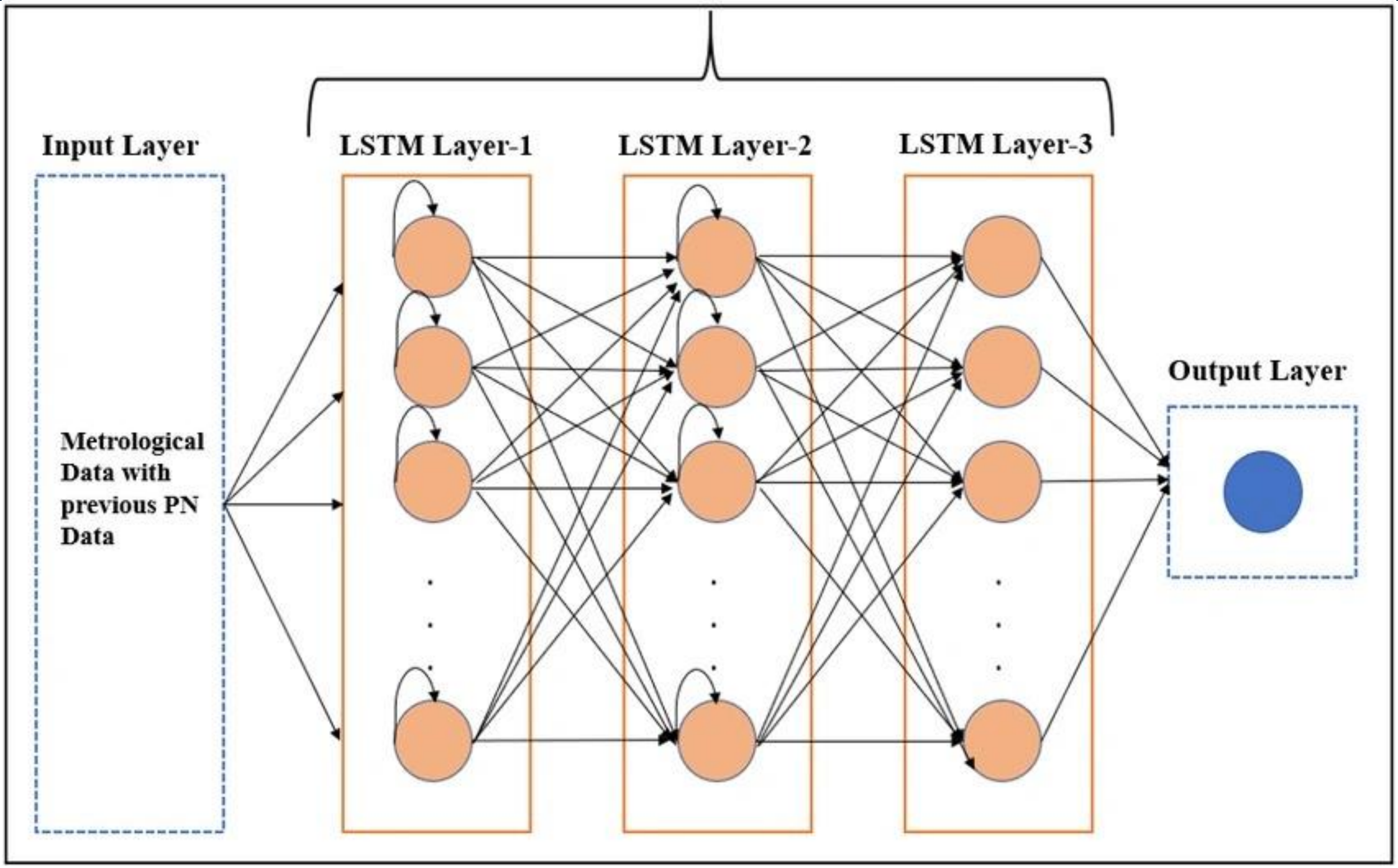
$$o_1 = \sigma(W_o \cdot x_1 + b_o) = \sigma(0.6 \cdot 3 + 0.4 + 0.05) = \sigma(2.25) \approx 0.904$$

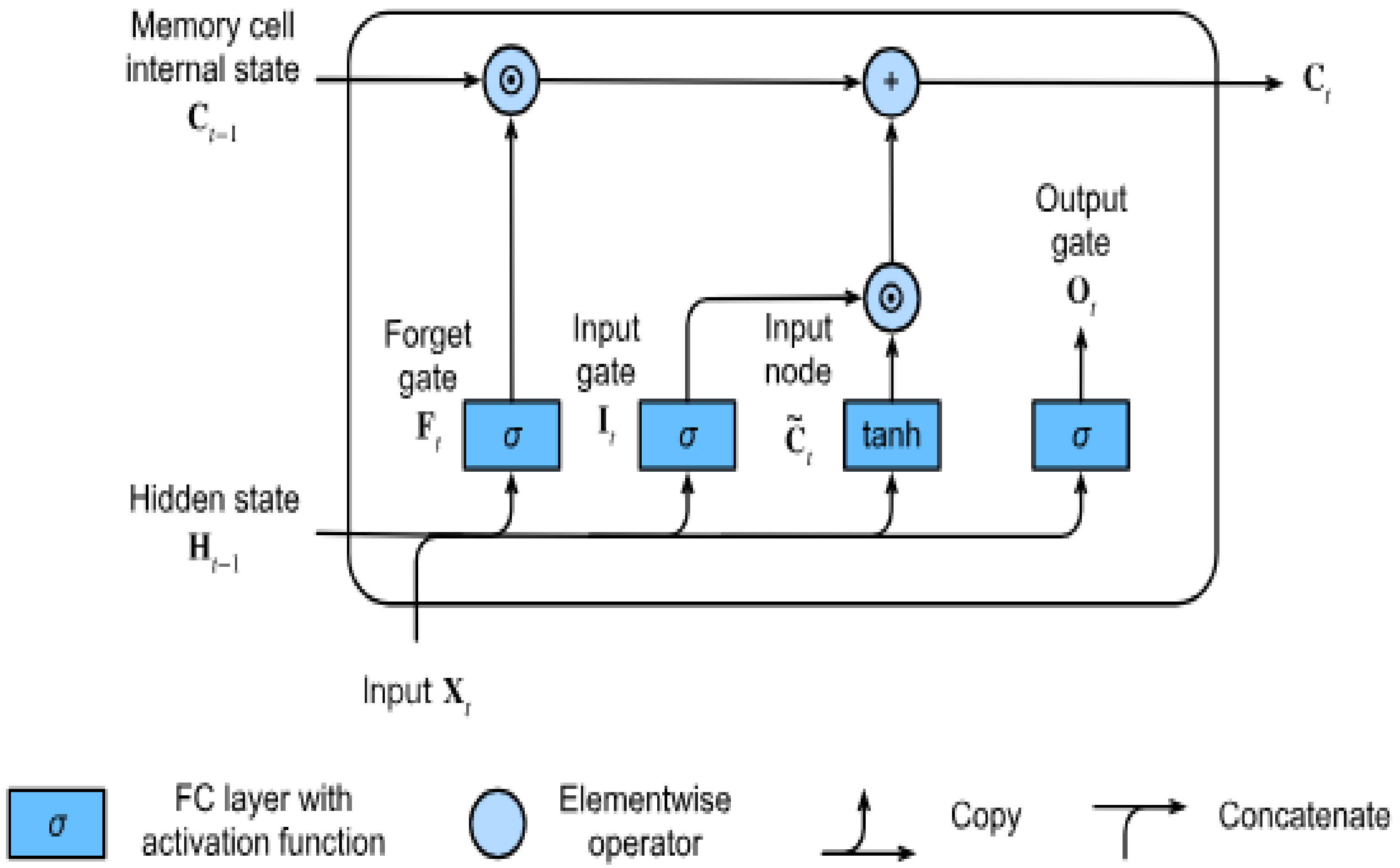
6. Hidden State:

$$h_1 = o_1 \cdot \tanh(C_1) = 0.904 \cdot \tanh(0.873) \approx 0.683$$

Time Step 2 ($x_2 = 5$):

Repeat the calculations with $x_2 = 5$, C_1, and h_1.


Input Layer
LSTM Layer-1
LSTM Layer-2
LSTM Layer-3
Output Layer
Metrological Data with previous PN Data



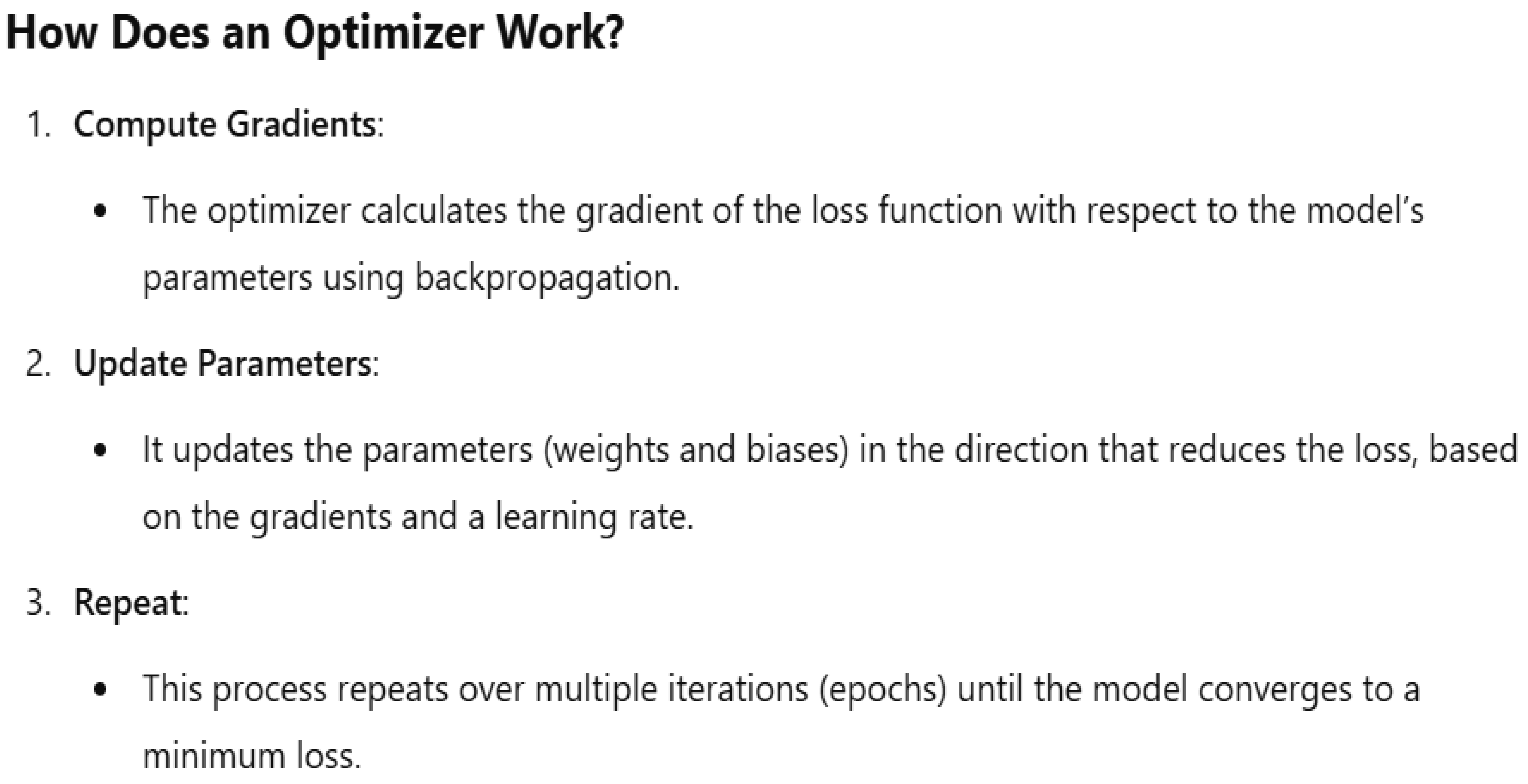
Memory cell internal state $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
Output gate $\mathbf{O}_t$
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Input node $\tilde{\mathbf{C}}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
Input $\mathbf{X}_t$
FC layer with activation function
Elementwise operator
Copy
Concatenate

Optimization

An **optimizer** is

- an algorithm used to update and adjust the weights and biases of a neural network during training to minimize the loss function.
- The goal of an optimizer is to find the set of parameters (weights) that result in the best performance of the model on the given task.

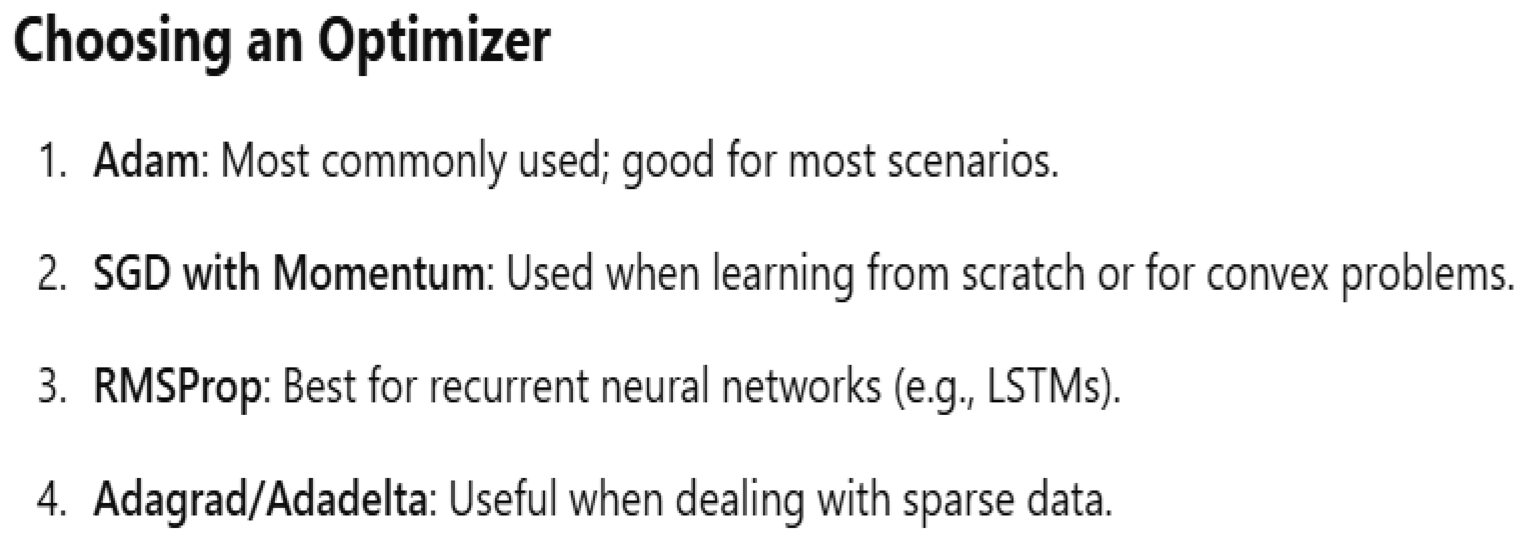
Con..



How Does an Optimizer Work?

1. **Compute Gradients**:
    - The optimizer calculates the gradient of the loss function with respect to the model's parameters using backpropagation.
2. **Update Parameters**:
    - It updates the parameters (weights and biases) in the direction that reduces the loss, based on the gradients and a learning rate.
3. **Repeat**:
    - This process repeats over multiple iterations (epochs) until the model converges to a minimum loss.

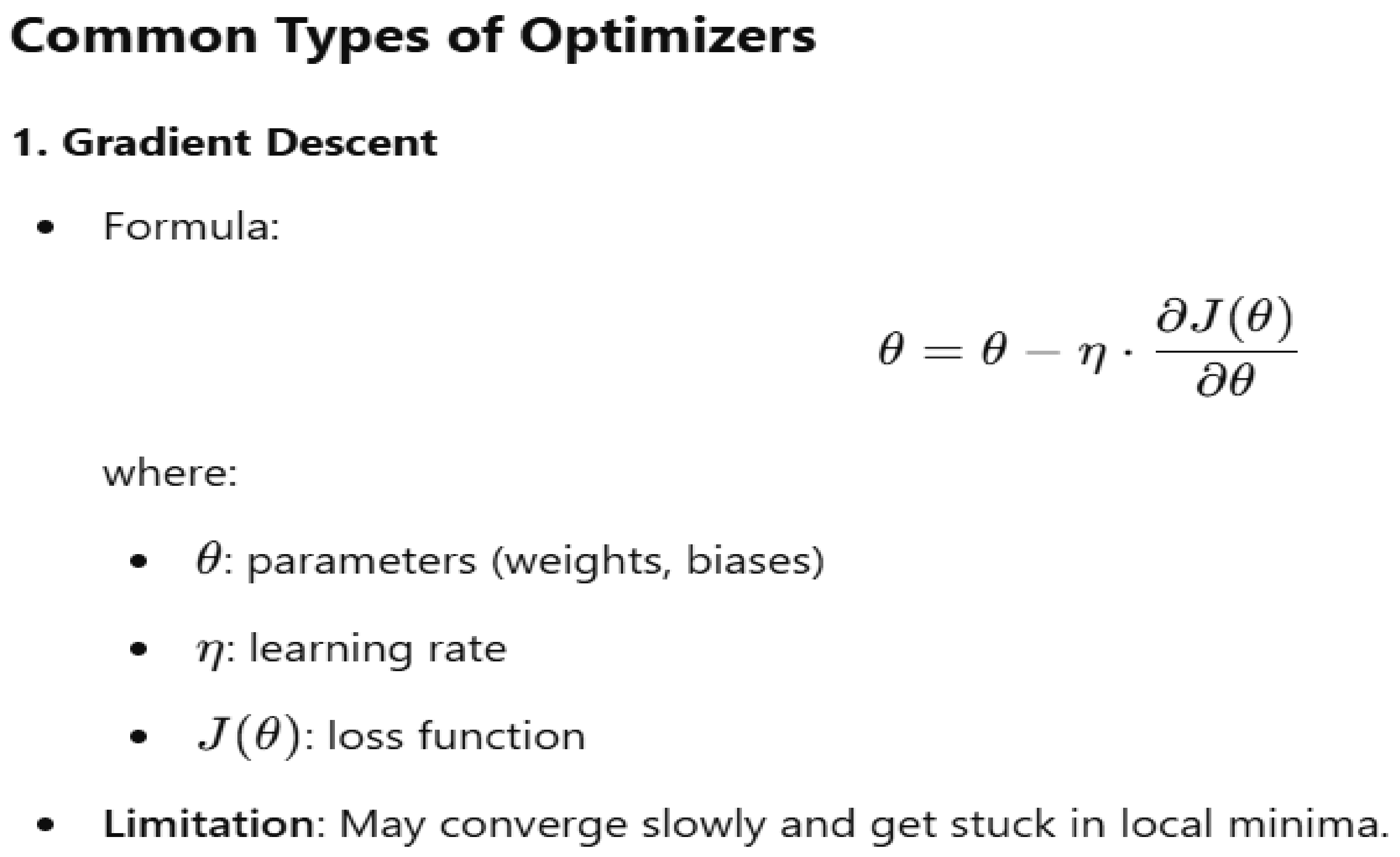
Con



Choosing an Optimizer

1. **Adam**: Most commonly used; good for most scenarios.
2. **SGD with Momentum**: Used when learning from scratch or for convex problems.
3. **RMSProp**: Best for recurrent neural networks (e.g., LSTMs).
4. **Adagrad/Adadelta**: Useful when dealing with sparse data.

Con



Common Types of Optimizers

1. Gradient Descent

- Formula:

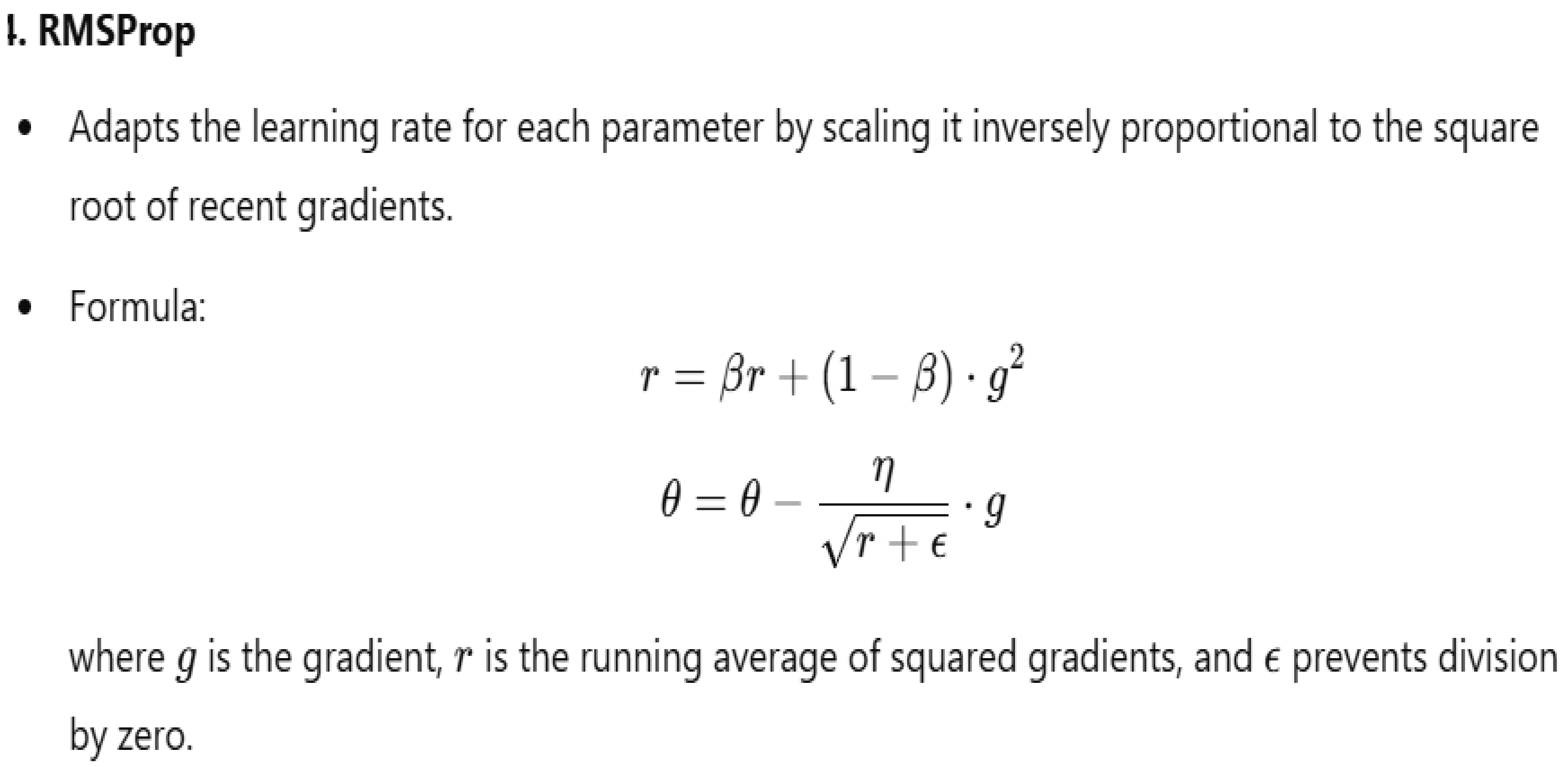
$$\theta = \theta - \eta \cdot \frac{\partial J(\theta)}{\partial \theta}$$

  where:

  - θ: parameters (weights, biases)
  - η: learning rate
  - $J(\theta)$: loss function

- **Limitation**: May converge slowly and get stuck in local minima.

Con



I. RMSProp

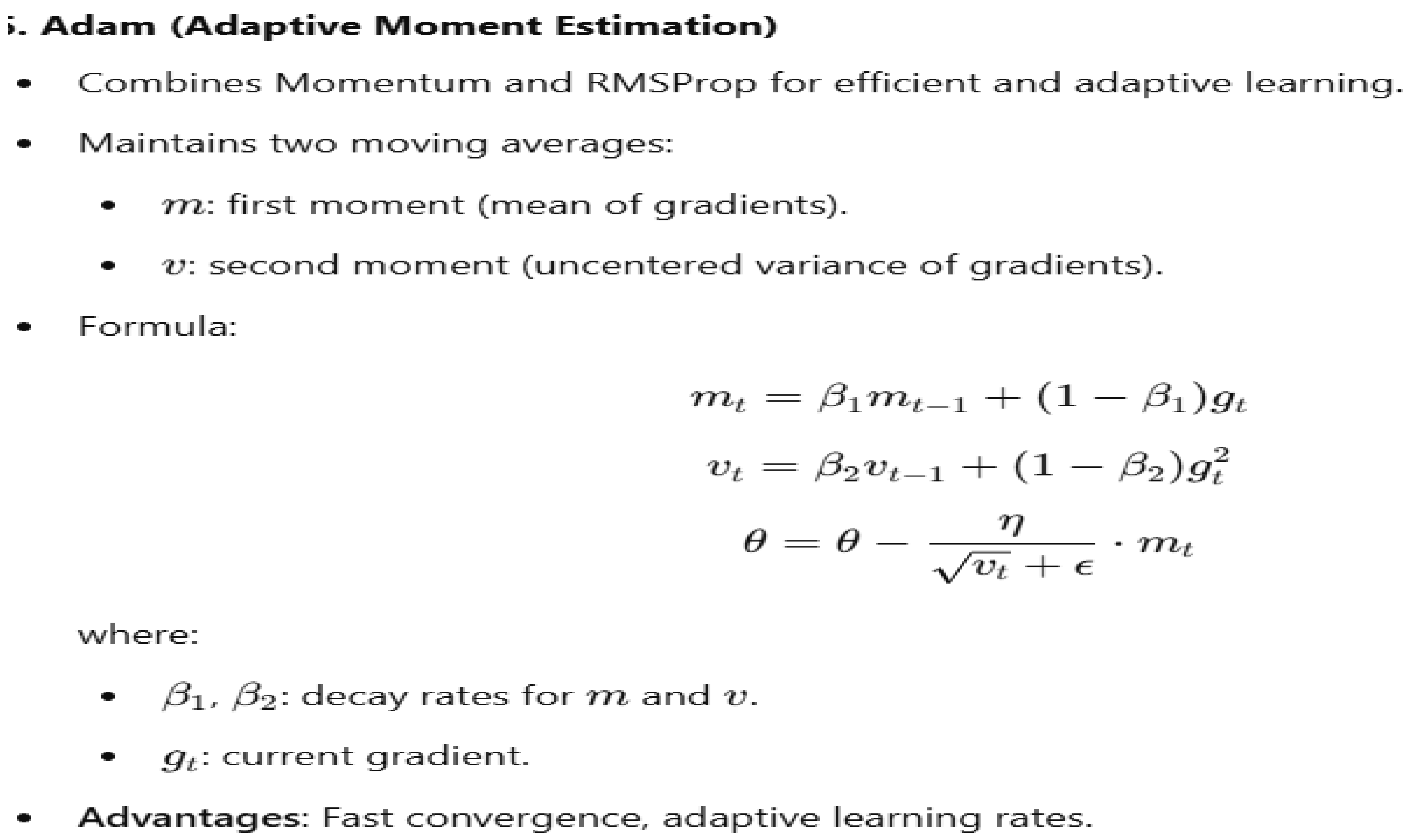
- Adapts the learning rate for each parameter by scaling it inversely proportional to the square root of recent gradients.
- Formula:

$$r = \beta r + (1 - \beta) \cdot g^2$$

$$\theta = \theta - \frac{\eta}{\sqrt{r + \epsilon}} \cdot g$$

where g is the gradient, r is the running average of squared gradients, and ϵ prevents division by zero.

Con..



. Adam (Adaptive Moment Estimation)

- Combines Momentum and RMSProp for efficient and adaptive learning.
- Maintains two moving averages:
    - m: first moment (mean of gradients).
    - v: second moment (uncentered variance of gradients).
- Formula:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\theta = \theta - \frac{\eta}{\sqrt{v_t} + \epsilon} \cdot m_t$$

where:

- β_1, β_2: decay rates for m and v.
- g_t: current gradient.

- **Advantages**: Fast convergence, adaptive learning rates.

Deep Learning: Limitations

- Data availability
- The complexity of the model
- Lacks global generalization
- Incapable of Multitasking
- Hardware dependence

Deep Learning: Limitations

- Data availability
  - Deep learning models require a lot of data to learn the representation, structure, distribution, and pattern of the data.
  - If there isn't enough varied data available, then the model will not learn well and will lack generalization (it won't perform well on unseen data).
  - The model can only generalize well if it is trained on large amounts of data.

Deep Learning: Limitations

- The complexity of the model
  - Designing a deep learning model is often a trial and error process.
  - A simple model is most likely to underfit, i.e. not able to extract information from the training set, and a very complex model is most likely to overfit, i.e., not able to generalize well on the test dataset.
  - Deep learning models will perform well when their complexity is appropriate to the complexity of the data.

Deep Learning: Limitations

- Lacks global generalization
  - A simple neural network can have thousands to tens of thousands of parameters.
  - The idea of global generalization is that all the parameters in the model should cohesively update themselves to reduce the generalization error or test error as much as possible. However, because of the complexity of the model, it is very difficult to achieve zero generalization error on the test set.
  - Hence, the deep learning model will always lack global generalization which can at times yield wrong results.

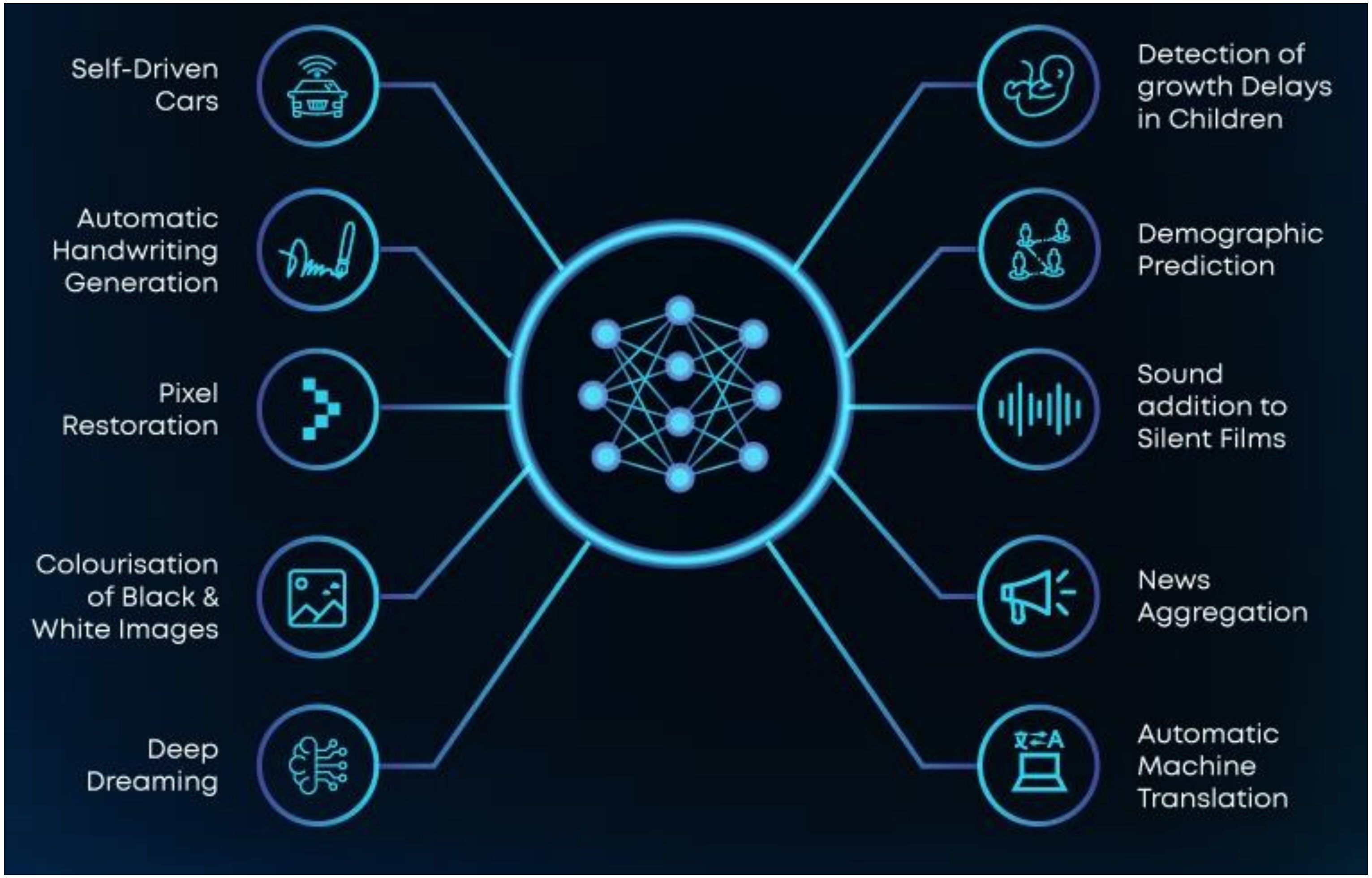
Deep Learning: Limitations

- Incapable of Multitasking
  - Deep neural networks are incapable of multitasking.
  - These models can only perform targeted tasks, i.e., process data on which they are trained. For instance, a model trained on classifying cats and dogs will not classify men and women.
  - Furthermore, applications that require reasoning or general intelligence are completely beyond what the current generation’s deep learning techniques can do, even with large sets of data.

Deep Learning: Limitations

- Hardware dependence
  - As mentioned before, deep learning models are computationally expensive.
  - These models are so complex that a normal CPU will not be able to withstand the computational complexity.
  - However, multicore high-performing graphics processing units (GPUs) and tensor processing units (TPUs) are required to effectively train these models in a shorter time.
  - Although these processors save time, they are expensive and use large amounts of energy.

Deep Learning: Applications


Self-Driven Cars
Automatic Handwriting Generation
Pixel Restoration
Colourisation of Black & White Images
Deep Dreaming
Detection of growth Delays in Children
Demographic Prediction
Sound addition to Silent Films
News Aggregation
Automatic Machine Translation

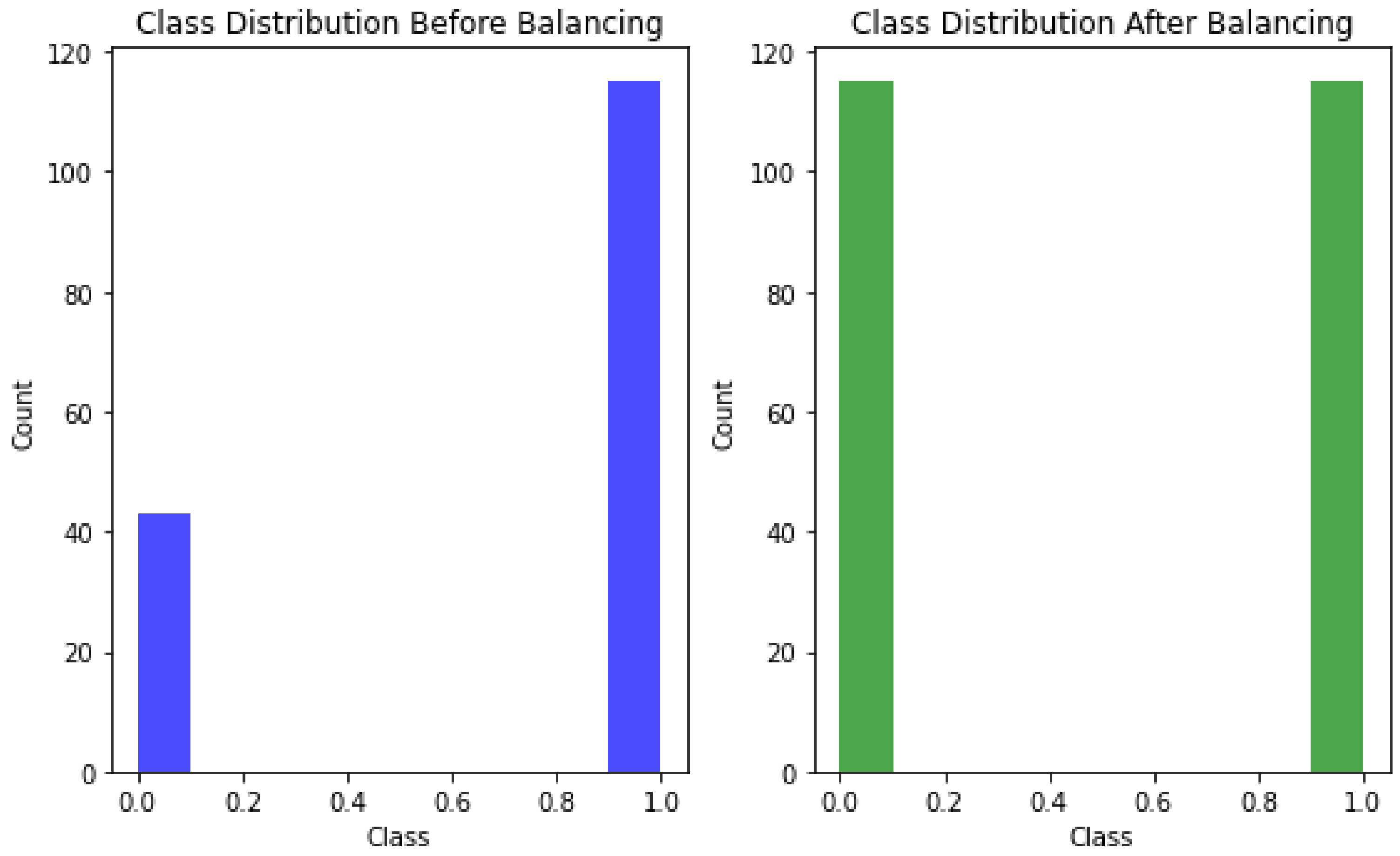
Data science final product

1. Data preprocessing
2. Feature engineering
3. Data balancing
4. feature selection
5. Deep learning or machine learning algorithm
6. Final production (interface or mobile app)

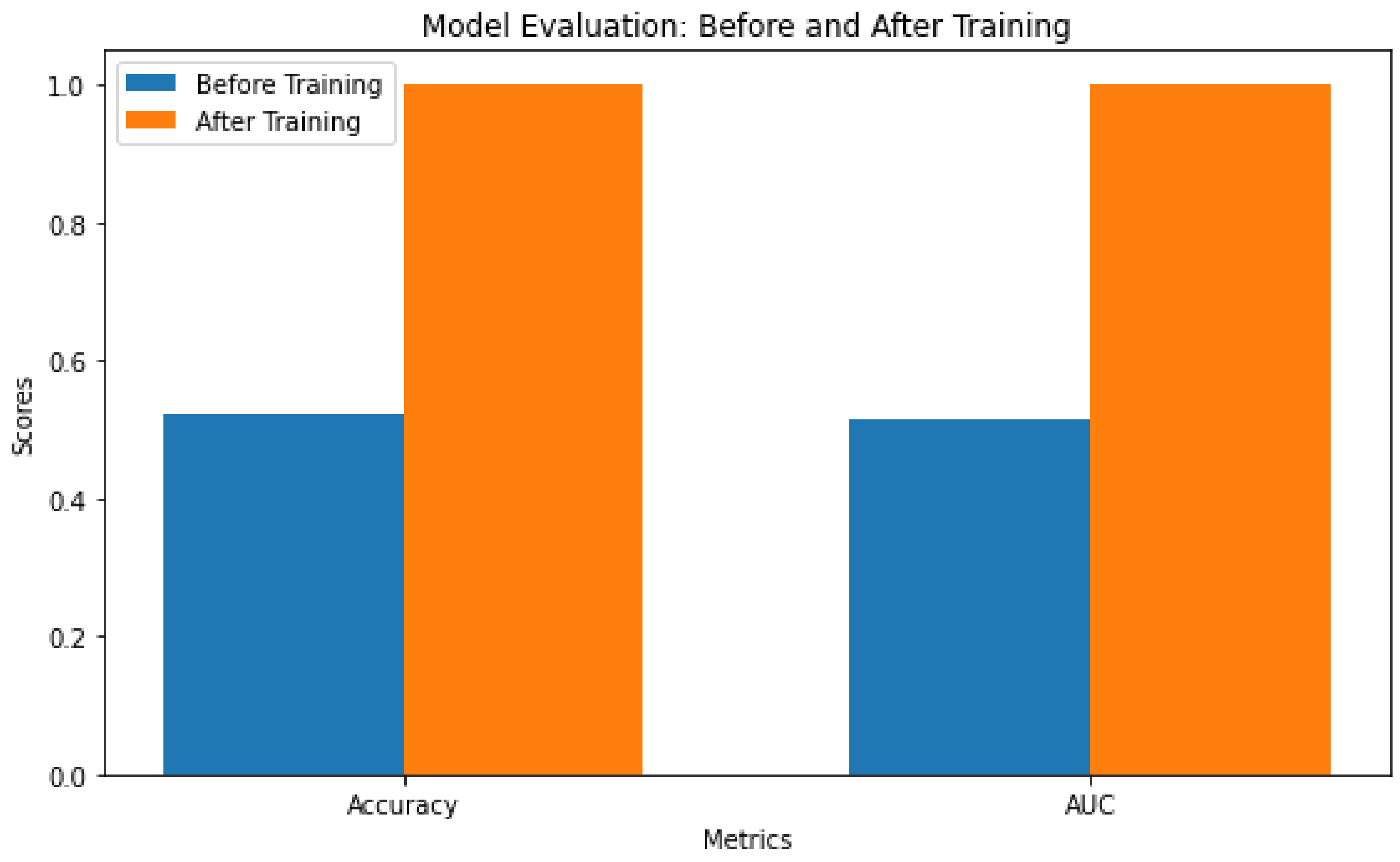
Full example using python

Attached code and file

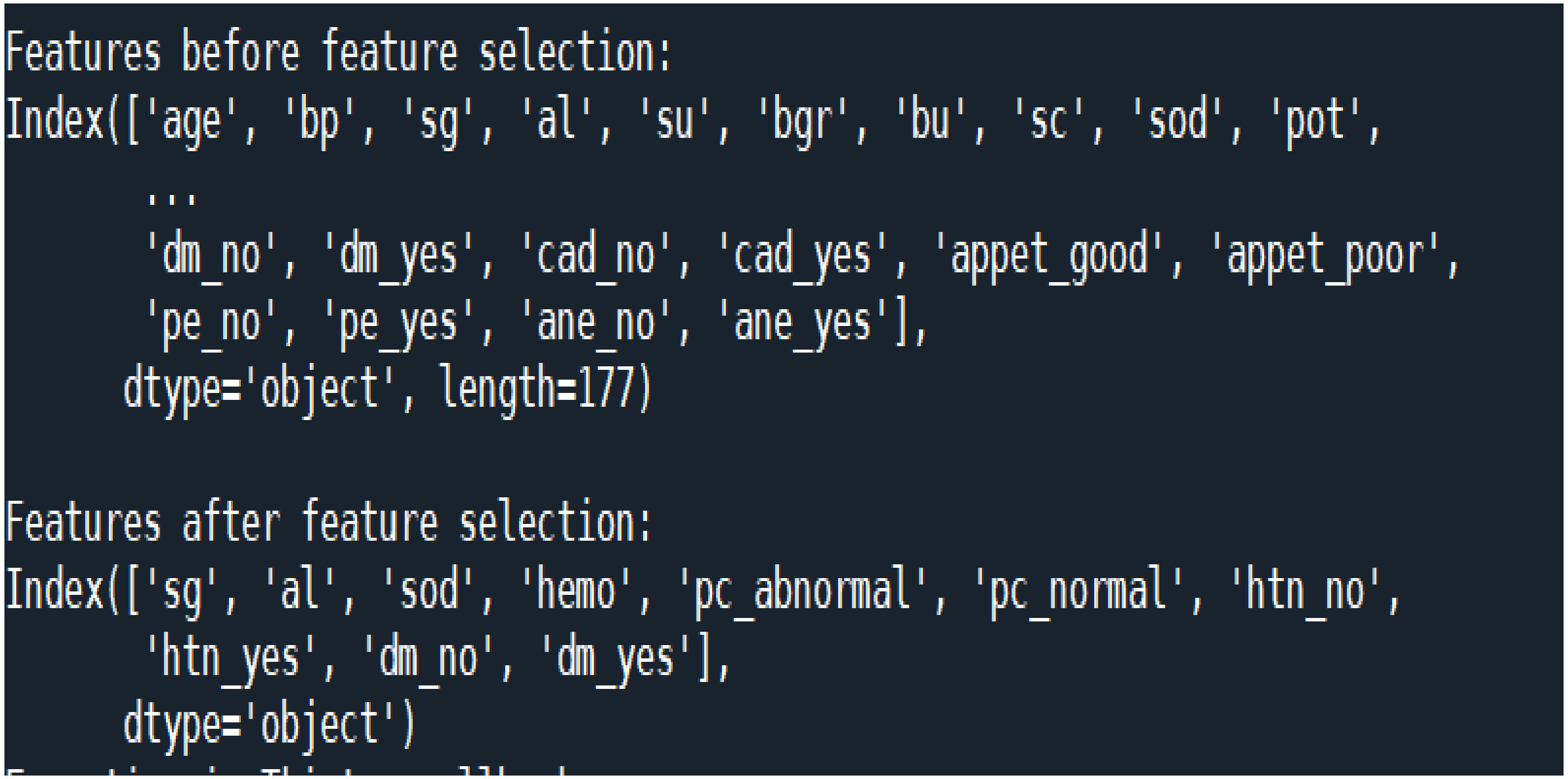
Balancing for the same example



Accuracy and auc



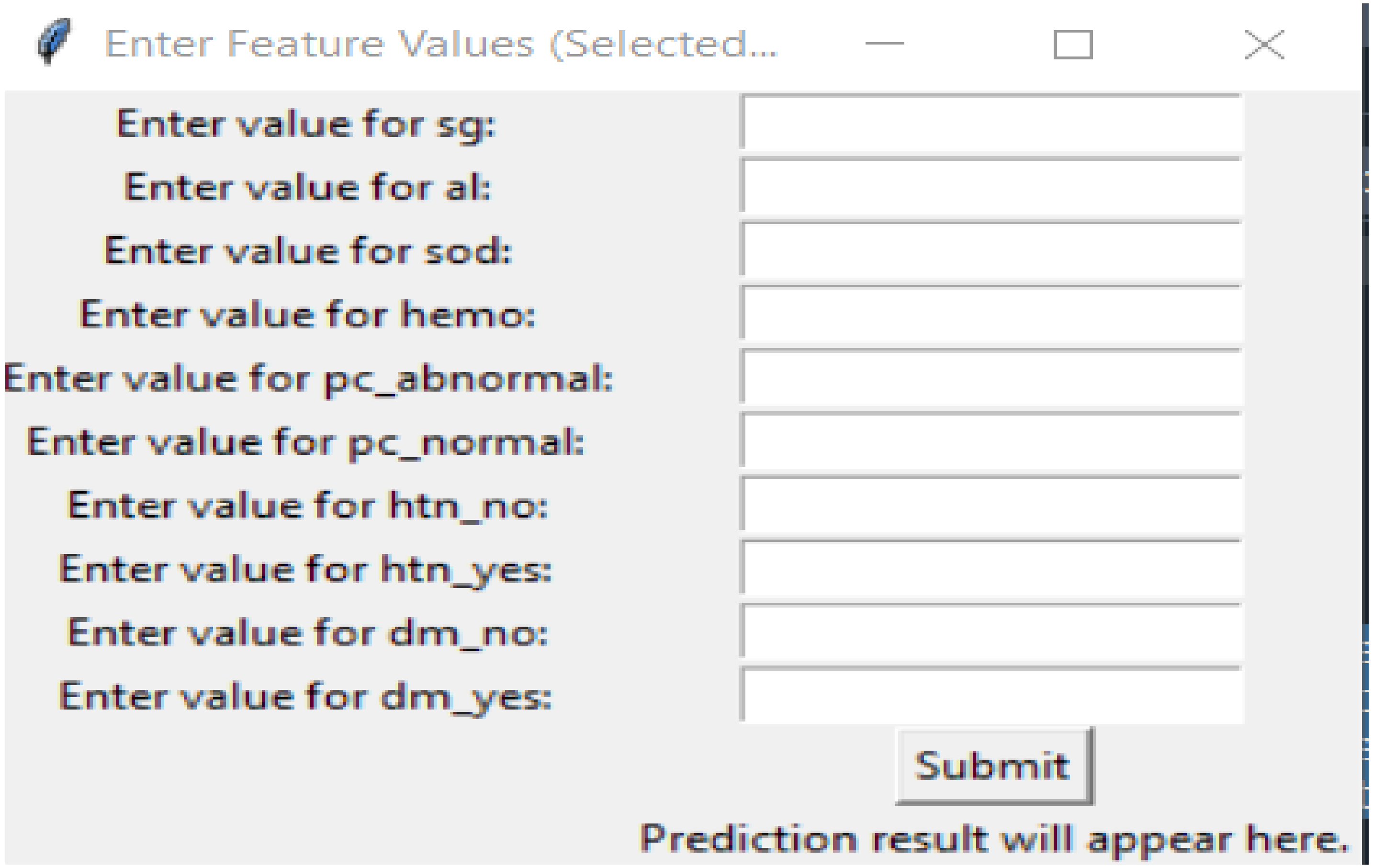
Selected features



```
Features before feature selection:
Index(['age', 'bp', 'sg', 'al', 'su', 'bgr', 'bu', 'sc', 'sod', 'pot',
       ...
       'dm_no', 'dm_yes', 'cad_no', 'cad_yes', 'appet_good', 'appet_poor',
       'pe_no', 'pe_yes', 'ane_no', 'ane_yes'],
      dtype='object', length=177)

Features after feature selection:
Index(['sg', 'al', 'sod', 'hemo', 'pc_abnormal', 'pc_normal', 'htn_no',
       'htn_yes', 'dm_no', 'dm_yes'],
      dtype='object')
```

Product



Enter Feature Values (Selected...

Enter value for sg:	
Enter value for al:	
Enter value for sod:	
Enter value for hemo:	
Enter value for pc_abnormal:	
Enter value for pc_normal:	
Enter value for htn_no:	
Enter value for htn_yes:	
Enter value for dm_no:	
Enter value for dm_yes:	

Prediction result will appear here.

A Quick Summary

- Machine learning requires data preprocessing, which involves human intervention.
- The neural networks in deep learning are capable of extracting features; hence no human intervention is required.
- Deep Learning can process unstructured data.
- Deep Learning is usually based on representative learning i.e., finding and extracting vital information or patterns that represent the entire dataset.
- Deep learning is computationally expensive and time-consuming.