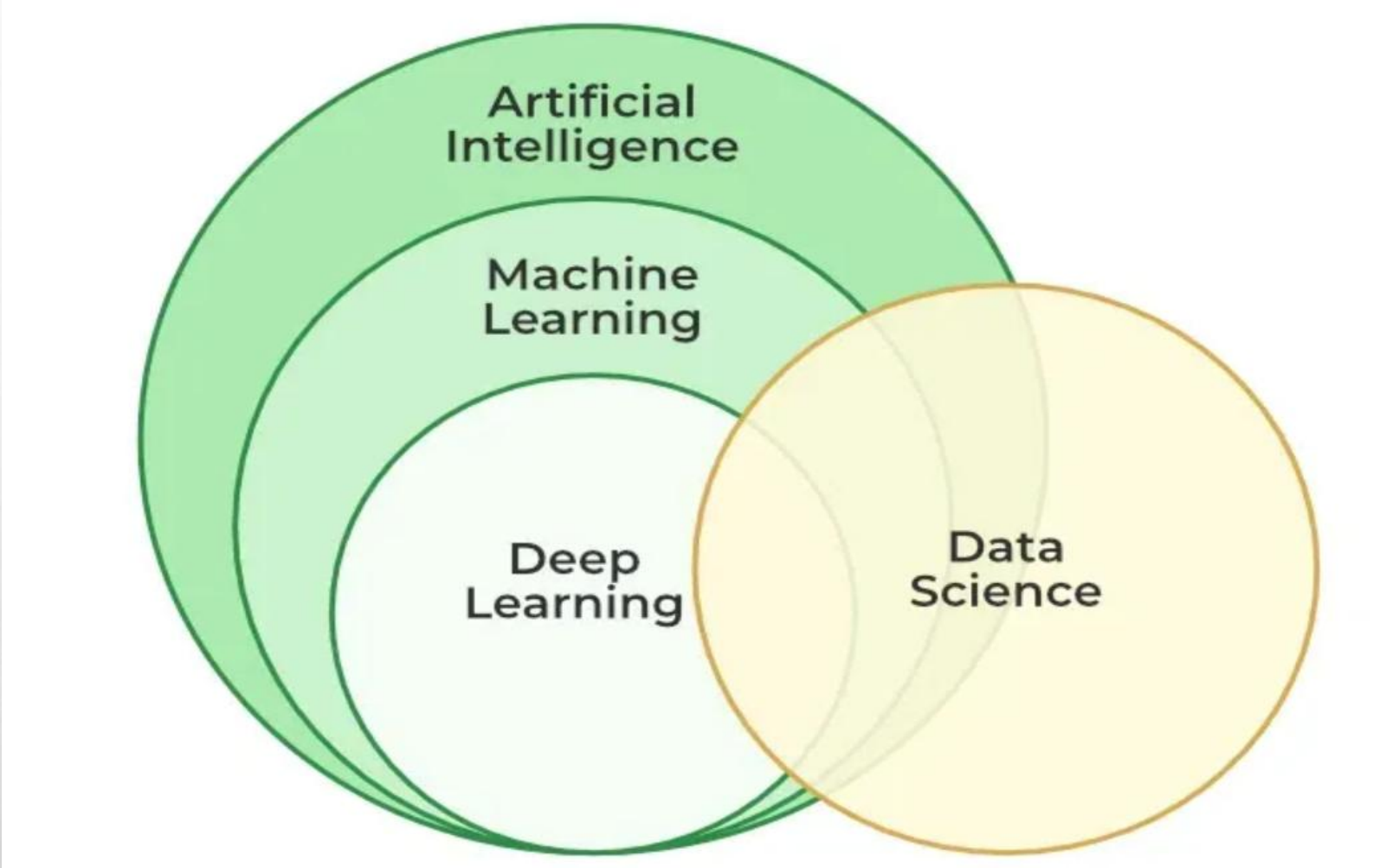
UNIT 2

DEEP LEARNING

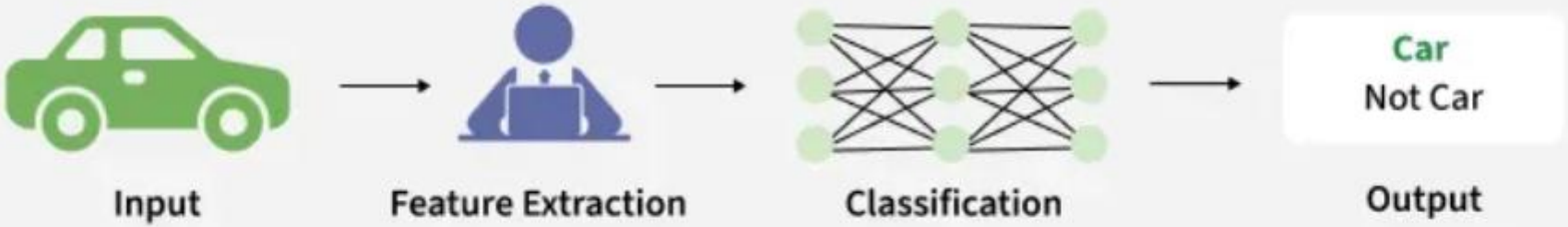
BY MS. PRIYANKA (ASSISTANT PROFESSOR)

INTRODUCTION TO DEEP LEARNING

- DEEP LEARNING MIMICS NEURAL NETWORKS OF THE HUMAN BRAIN, IT ENABLES COMPUTERS TO AUTONOMOUSLY UNCOVER PATTERNS AND MAKE INFORMED DECISIONS FROM VAST AMOUNTS OF UNSTRUCTURED DATA.


Artificial Intelligence
Machine Learning
Deep Learning
Data Science

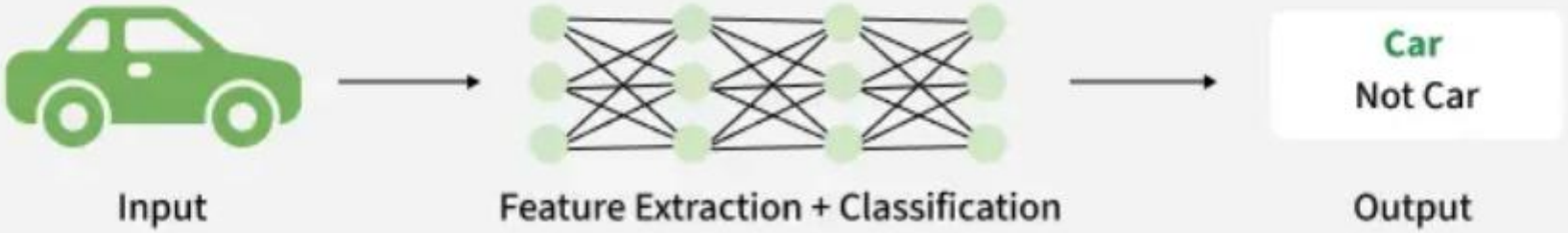
Machine Learning



Uses algorithms and learns on its own but may need human intervention to correct errors

---

Deep Learning

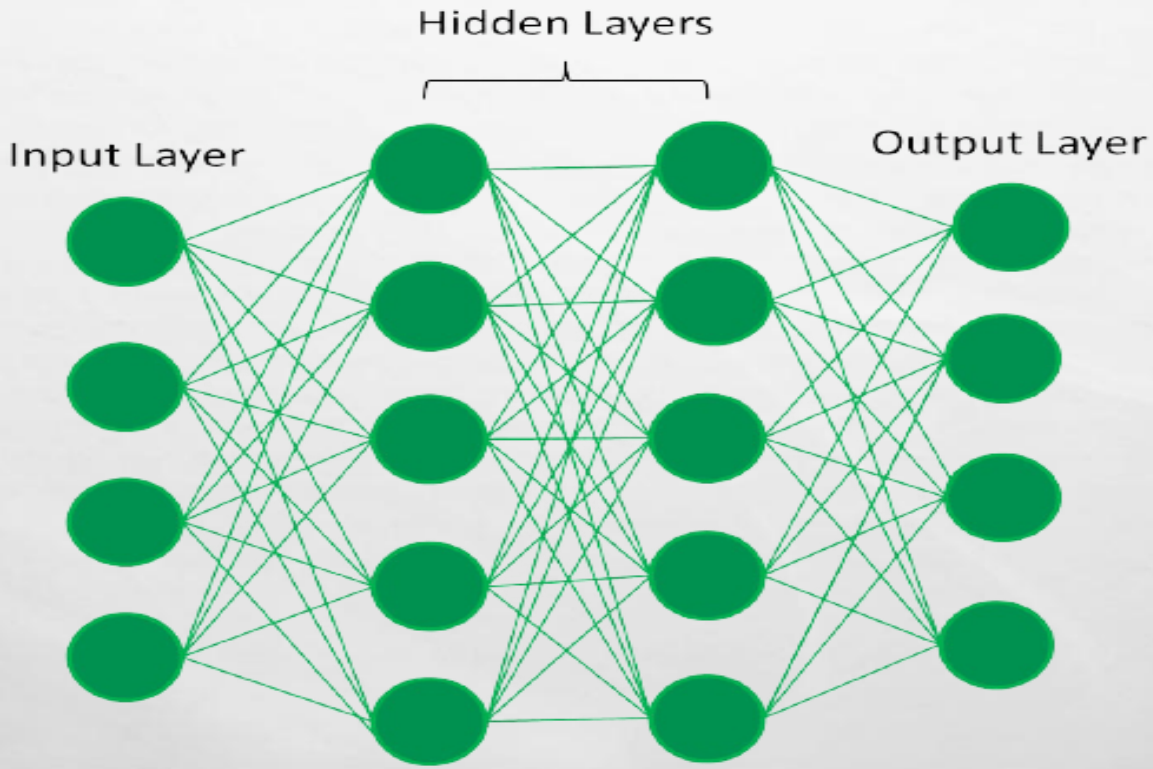


Uses advanced computing, its own neural network, to adapt with little to no human intervention

HOW DEEP LEARNING WORKS?

- NEURAL NETWORK CONSISTS OF LAYERS OF INTERCONNECTED NODES OR NEURONS THAT COLLABORATE TO PROCESS INPUT DATA.
- IN A FULLY CONNECTED DEEP NEURAL NETWORK DATA FLOWS THROUGH MULTIPLE LAYERS WHERE EACH NEURON PERFORMS NONLINEAR TRANSFORMATIONS, ALLOWING THE MODEL TO LEARN INTRICATE REPRESENTATIONS OF THE DATA.
- IN A DEEP NEURAL NETWORK THE INPUT LAYER RECEIVES DATA WHICH PASSES THROUGH HIDDEN LAYERS THAT TRANSFORM THE DATA USING NONLINEAR FUNCTIONS. THE FINAL OUTPUT LAYER GENERATES THE MODEL'S PREDICTION.

NEURAL NETWORKS



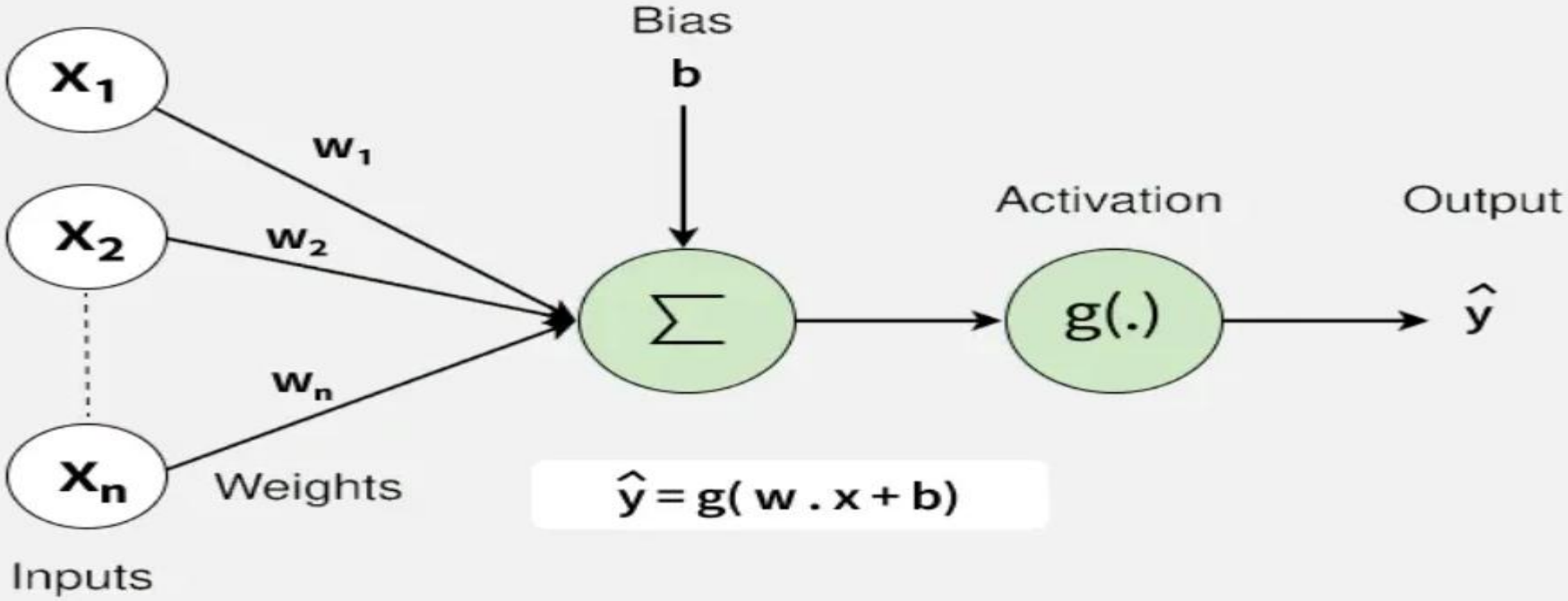
Machine Learning	Deep Learning
Apply statistical algorithms to learn the hidden patterns and relationships in the dataset.	Uses artificial neural network architecture to learn the hidden patterns and relationships in the dataset.
Can work on the smaller amount of dataset	Requires the larger volume of dataset compared to machine learning
Better for the low-label task.	Better for complex task like image processing, natural language processing, etc.
Takes less time to train the model.	Takes more time to train the model.
A model is created by relevant features which are manually extracted from images to detect an object in the image.	Relevant features are automatically extracted from images. It is an end-to-end learning process.
Less complex and easy to interpret the result.	More complex, it works like the black box interpretations of the result are not easy.
It can work on the CPU or requires less computing power as compared to deep learning.	It requires a high-performance computer with GPU.

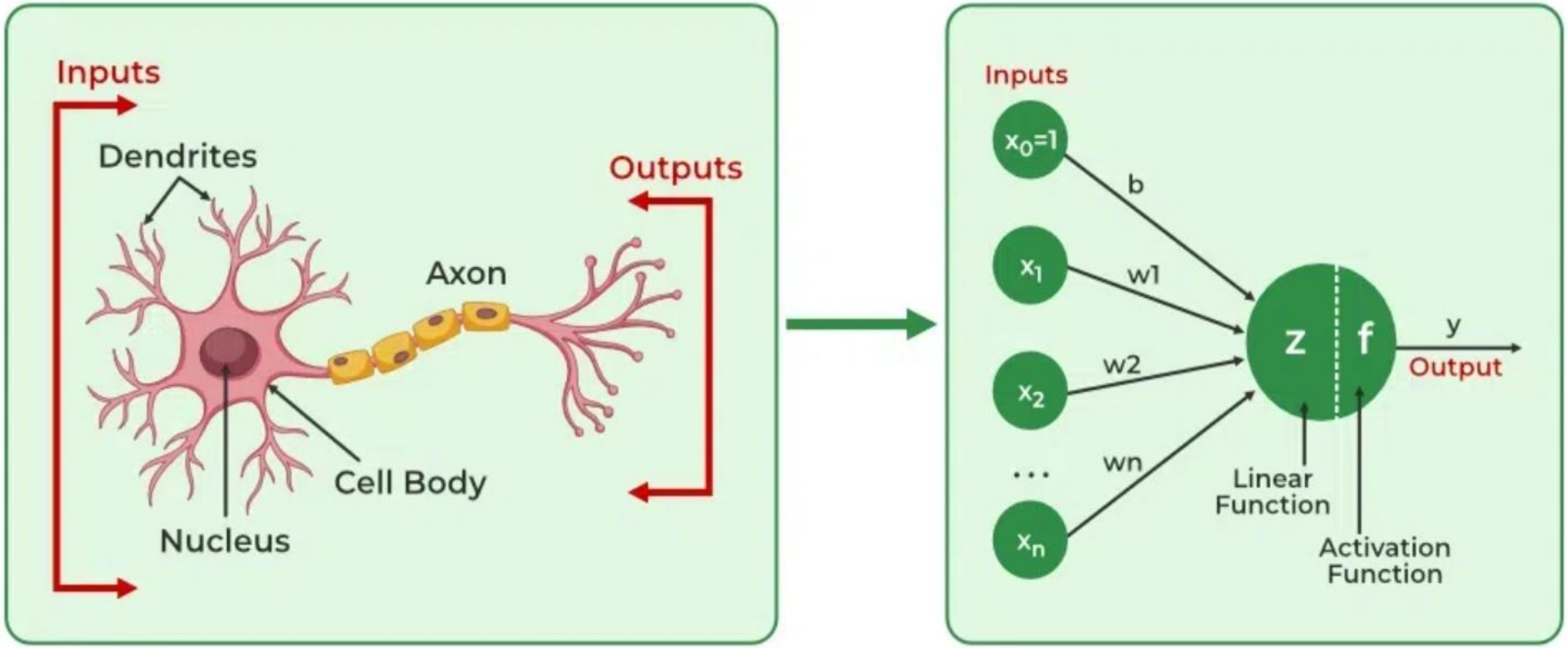
EVOLUTION OF NEURAL ARCHITECTURES

- THE JOURNEY OF DEEP LEARNING BEGAN WITH THE PERCEPTRON, A SINGLE-LAYER NEURAL NETWORK INTRODUCED IN THE 1950S. WHILE INNOVATIVE, PERCEPTRONS COULD ONLY SOLVE LINEARLY SEPARABLE PROBLEMS HENCE FAILING AT MORE COMPLEX TASKS LIKE THE XOR PROBLEM.
- THIS LIMITATION LED TO THE DEVELOPMENT OF MULTI-LAYER PERCEPTRONS (MLPS).
- IT INTRODUCED HIDDEN LAYERS AND NON-LINEAR ACTIVATION FUNCTIONS.
- MLPS TRAINED USING BACKPROPAGATION COULD MODEL COMPLEX, NON-LINEAR RELATIONSHIPS MARKING A SIGNIFICANT LEAP IN NEURAL NETWORK CAPABILITIES.
- THIS EVOLUTION FROM PERCEPTRONS TO MLPS LAID THE GROUNDWORK FOR ADVANCED ARCHITECTURES LIKE CNNS AND RNNS, SHOWCASING THE POWER OF LAYERED STRUCTURES IN SOLVING REAL-WORLD PROBLEMS.

WHAT IS A NEURAL NETWORK?

- NEURAL NETWORKS ARE MACHINE LEARNING MODELS THAT MIMIC THE COMPLEX FUNCTIONS OF THE HUMAN BRAIN. THESE MODELS CONSIST OF INTERCONNECTED NODES OR NEURONS THAT PROCESS DATA, LEARN PATTERNS AND ENABLE TASKS SUCH AS PATTERN RECOGNITION AND DECISION-MAKING.

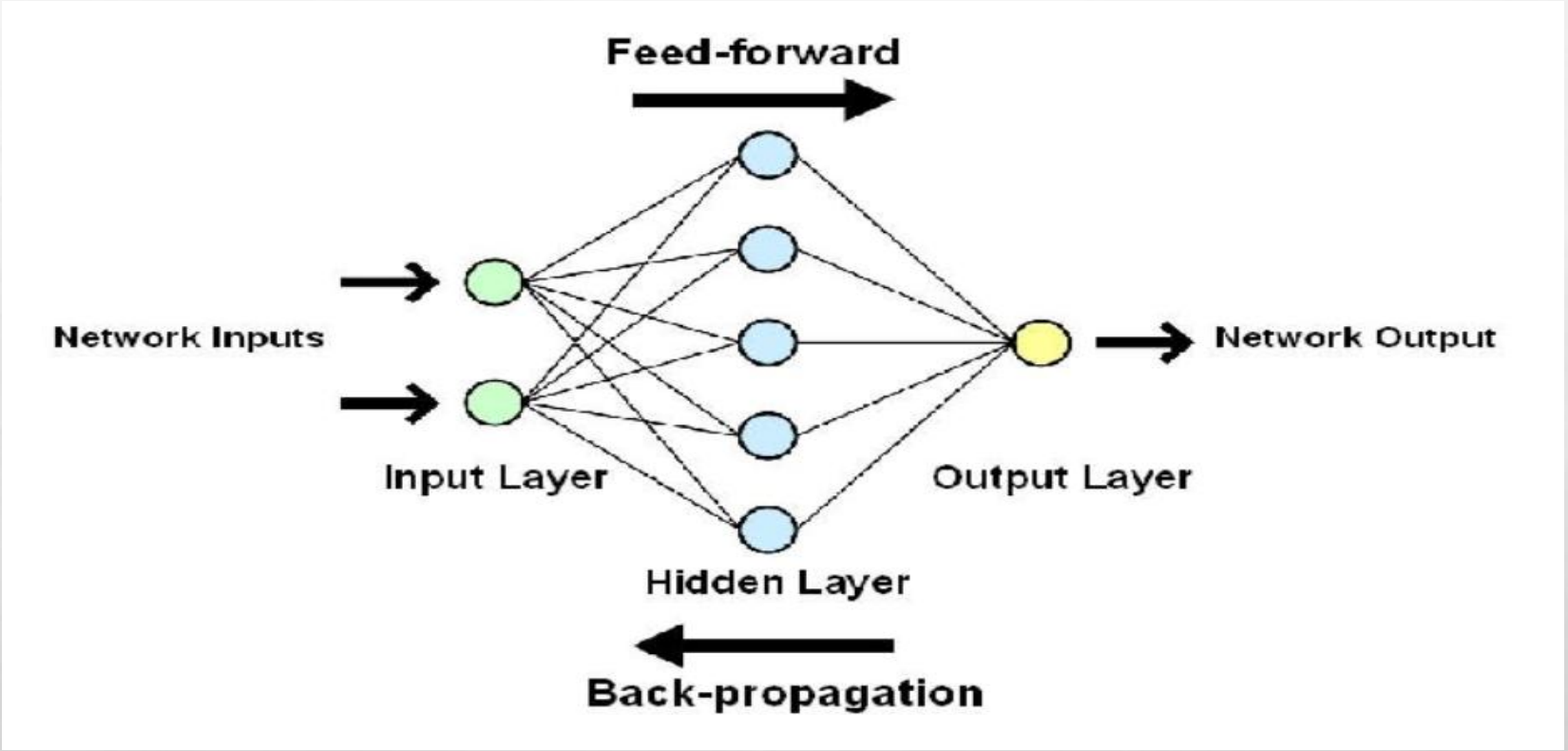



Inputs
Dendrites
Outputs
Axon
Cell Body
Nucleus
Inputs
$x_0=1$
b
x_1
w1
x_2
w2
...
wn
x_n
z
f
y
Output
Linear Function
Activation Function

UNDERSTANDING NEURAL NETWORKS IN DEEP LEARNING

- NEURAL NETWORKS ARE CAPABLE OF LEARNING AND IDENTIFYING PATTERNS DIRECTLY FROM DATA WITHOUT PRE-DEFINED RULES. THESE NETWORKS ARE BUILT FROM SEVERAL KEY COMPONENTS:
- **NEURONS**: THE BASIC UNITS THAT RECEIVE INPUTS, EACH NEURON IS GOVERNED BY A THRESHOLD AND AN ACTIVATION FUNCTION.
- **CONNECTIONS**: LINKS BETWEEN NEURONS THAT CARRY INFORMATION, REGULATED BY WEIGHTS AND BIASES.
- **WEIGHTS AND BIASES**: THESE PARAMETERS DETERMINE THE STRENGTH AND INFLUENCE OF CONNECTIONS.
- **PROPAGATION FUNCTIONS**: MECHANISMS THAT HELP PROCESS AND TRANSFER DATA ACROSS LAYERS OF NEURONS.
- **LEARNING RULE**: THE METHOD THAT ADJUSTS WEIGHTS AND BIASES OVER TIME TO IMPROVE ACCURACY.

ARTIFICIAL NEURAL NETWORK



LAYERS IN NEURAL NETWORK ARCHITECTURE

- **INPUT LAYER:** THIS IS WHERE THE NETWORK RECEIVES ITS INPUT DATA. EACH INPUT NEURON IN THE LAYER CORRESPONDS TO A FEATURE IN THE INPUT DATA.
- **HIDDEN LAYERS:** THESE LAYERS PERFORM MOST OF THE COMPUTATIONAL HEAVY LIFTING. A NEURAL NETWORK CAN HAVE ONE OR MULTIPLE HIDDEN LAYERS. EACH LAYER CONSISTS OF UNITS (NEURONS) THAT TRANSFORM THE INPUTS INTO SOMETHING THAT THE OUTPUT LAYER CAN USE.
- **OUTPUT LAYER:** THE FINAL LAYER PRODUCES THE OUTPUT OF THE MODEL. THE FORMAT OF THESE OUTPUTS VARIES DEPENDING ON THE SPECIFIC TASK LIKE CLASSIFICATION, REGRESSION.

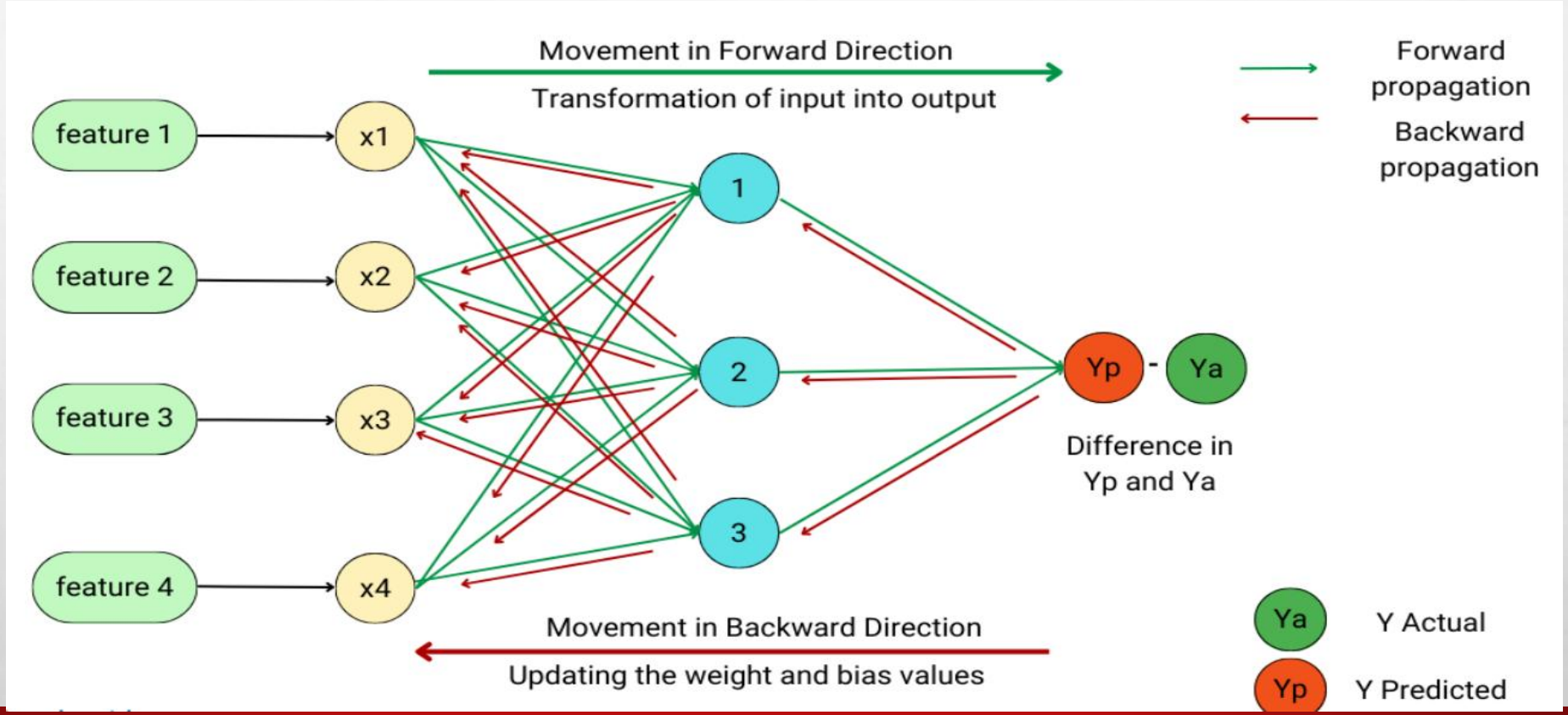
WORKING OF NEURAL NETWORKS

- **1. FORWARD PROPAGATION**
- WHEN DATA IS INPUT INTO THE NETWORK, IT PASSES THROUGH THE NETWORK IN THE FORWARD DIRECTION, FROM THE INPUT LAYER THROUGH THE HIDDEN LAYERS TO THE OUTPUT LAYER. THIS PROCESS IS KNOWN AS FORWARD PROPAGATION. HERE'S WHAT HAPPENS DURING THIS PHASE:

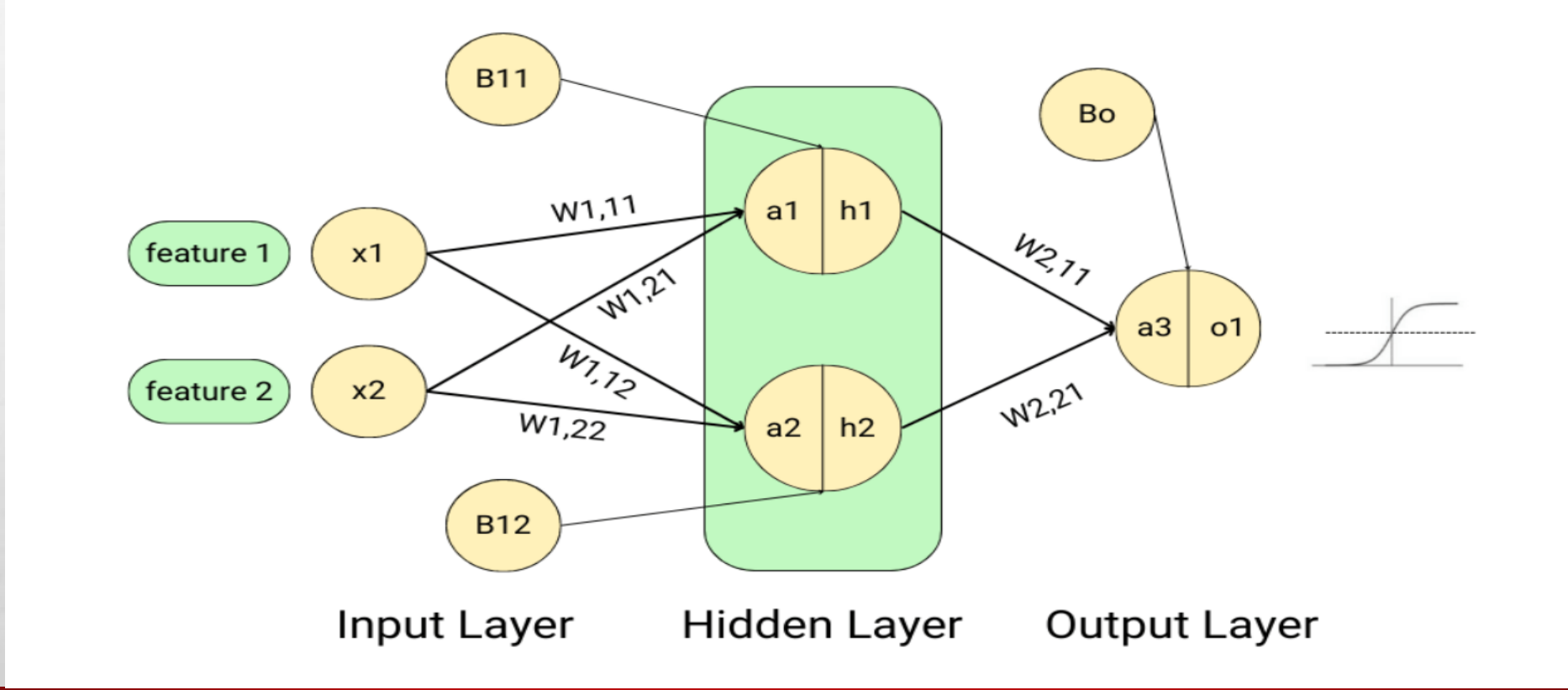
- **1. LINEAR TRANSFORMATION:** EACH NEURON IN A LAYER RECEIVES INPUTS WHICH ARE MULTIPLIED BY THE WEIGHTS ASSOCIATED WITH THE CONNECTIONS. THESE PRODUCTS ARE SUMMED TOGETHER AND A BIAS IS ADDED TO THE SUM. THIS CAN BE REPRESENTED MATHEMATICALLY AS:
- $Z=W1X1+W2X2+\ldots+WNXN+BZ=W1X1+W2X2+\ldots+WNXN+B$
- WHERE,
- *W* REPRESENTS THE WEIGHTS
- *X* REPRESENTS THE INPUTS
- *B* IS THE BIAS

- **2. ACTIVATION:** THE RESULT OF THE LINEAR TRANSFORMATION (DENOTED AS *Z*) IS THEN PASSED THROUGH AN ACTIVATION FUNCTION. THE ACTIVATION FUNCTION IS CRUCIAL BECAUSE IT INTRODUCES NON-LINEARITY INTO THE SYSTEM, ENABLING THE NETWORK TO LEARN MORE COMPLEX PATTERNS. POPULAR ACTIVATION FUNCTIONS INCLUDE RELU, SIGMOID AND TANH.

FORWARD PROPAGATION



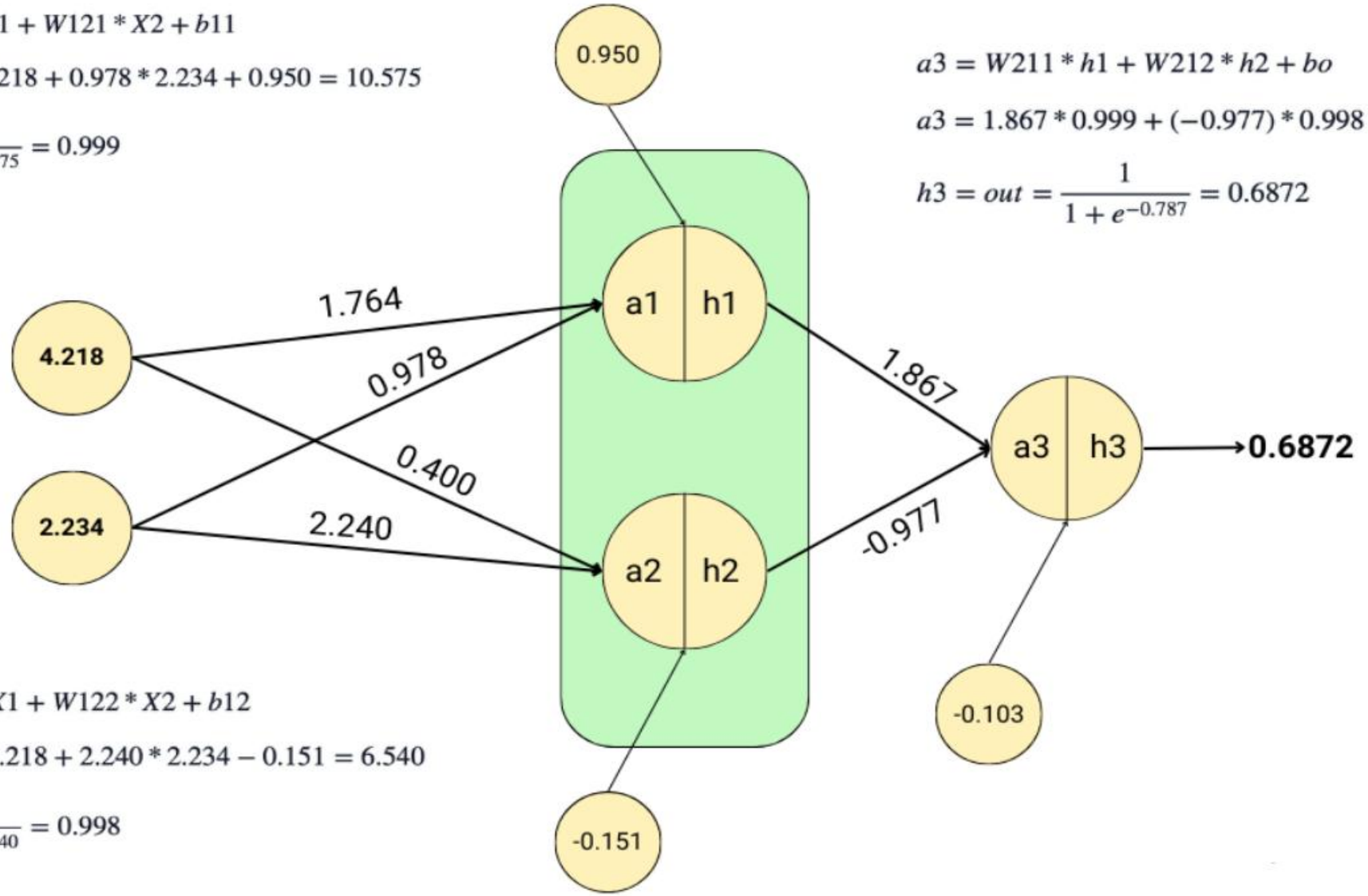
MATHEMATICAL INTUITION



$$a1 = W111 * X1 + W121 * X2 + b11$$

$$a1 = 1.764 * 4.218 + 0.978 * 2.234 + 0.950 = 10.575$$

$$h1 = \frac{1}{1 + e^{-10.575}} = 0.999$$



$$a3 = W211 * h1 + W212 * h2 + bo$$

$$a3 = 1.867 * 0.999 + (-0.977) * 0.998 - 0.103 = 0.787$$

$$h3 = out = \frac{1}{1 + e^{-0.787}} = 0.6872$$

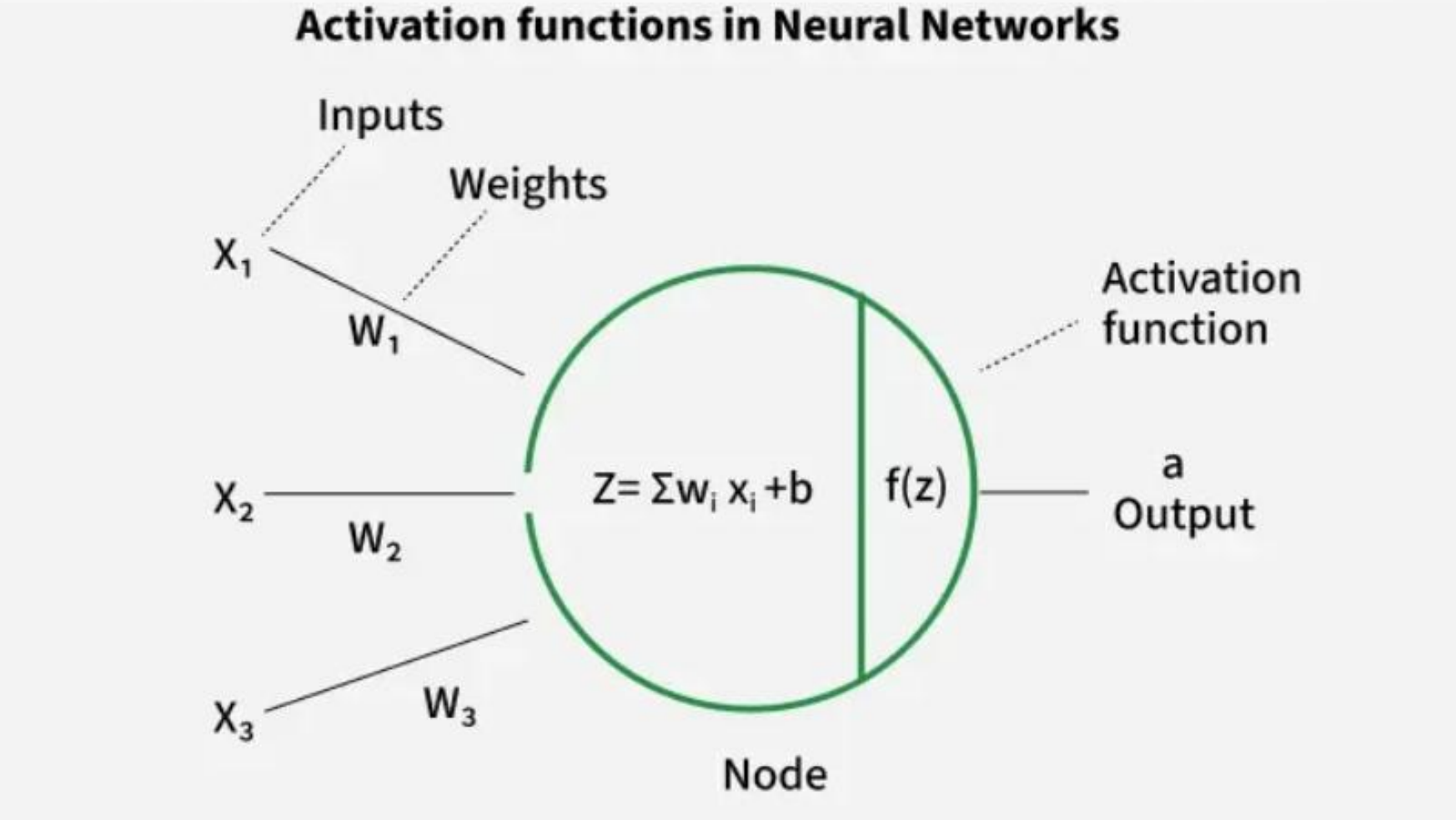
$$a2 = W112 * X1 + W122 * X2 + b12$$

$$a2 = 0.400 * 4.218 + 2.240 * 2.234 - 0.151 = 6.540$$

$$h2 = \frac{1}{1 + e^{-6.540}} = 0.998$$

ACTIVATION FUNCTIONS IN NEURAL NETWORKS

- WHILE BUILDING A NEURAL NETWORK, ONE KEY DECISION IS SELECTING THE ACTIVATION FUNCTION FOR BOTH THE HIDDEN LAYER AND THE OUTPUT LAYER.
- IT IS A MATHEMATICAL FUNCTION APPLIED TO THE OUTPUT OF A NEURON. IT INTRODUCES NON-LINEARITY INTO THE MODEL, ALLOWING THE NETWORK TO LEARN AND REPRESENT COMPLEX PATTERNS IN THE DATA.
- WITHOUT THIS NON-LINEARITY FEATURE A NEURAL NETWORK WOULD BEHAVE LIKE A LINEAR REGRESSION MODEL NO MATTER HOW MANY LAYERS IT HAS.
- ACTIVATION FUNCTION DECIDES WHETHER A NEURON SHOULD BE ACTIVATED BY CALCULATING THE WEIGHTED SUM OF INPUTS AND ADDING A BIAS TERM.
- THIS HELPS THE MODEL MAKE COMPLEX DECISIONS AND PREDICTIONS BY INTRODUCING NON-LINEARITIES TO THE OUTPUT OF EACH NEURON.


Activation functions in Neural Networks
Inputs
Weights
X_1
W_1
X_2
W_2
X_3
W_3
$Z= \Sigma w_i x_i + b$
f(z)
Activation function
a
Output
Node

INTRODUCING NON-LINEARITY IN NEURAL NETWORK

- NON-LINEARITY MEANS THAT THE RELATIONSHIP BETWEEN INPUT AND OUTPUT IS NOT A STRAIGHT LINE. IN SIMPLE TERMS THE OUTPUT DOES NOT CHANGE PROPORTIONALLY WITH THE INPUT. A COMMON CHOICE IS THE RELU FUNCTION DEFINED AS $\Sigma(X)=MAX(0,X)$.
- IMAGINE YOU WANT TO CLASSIFY APPLES AND BANANAS BASED ON THEIR SHAPE AND COLOR.
- IF WE USE A LINEAR FUNCTION, IT CAN ONLY SEPARATE THEM USING A STRAIGHT LINE.
- BUT REAL-WORLD DATA IS OFTEN MORE COMPLEX LIKE OVERLAPPING COLORS, DIFFERENT LIGHTING, ETC.
- BY ADDING A NON-LINEAR ACTIVATION FUNCTION LIKE RELU, SIGMOID OR TANH THE NETWORK CAN CREATE CURVED DECISION BOUNDARIES TO SEPARATE THEM CORRECTLY.

EFFECT OF NON-LINEARITY

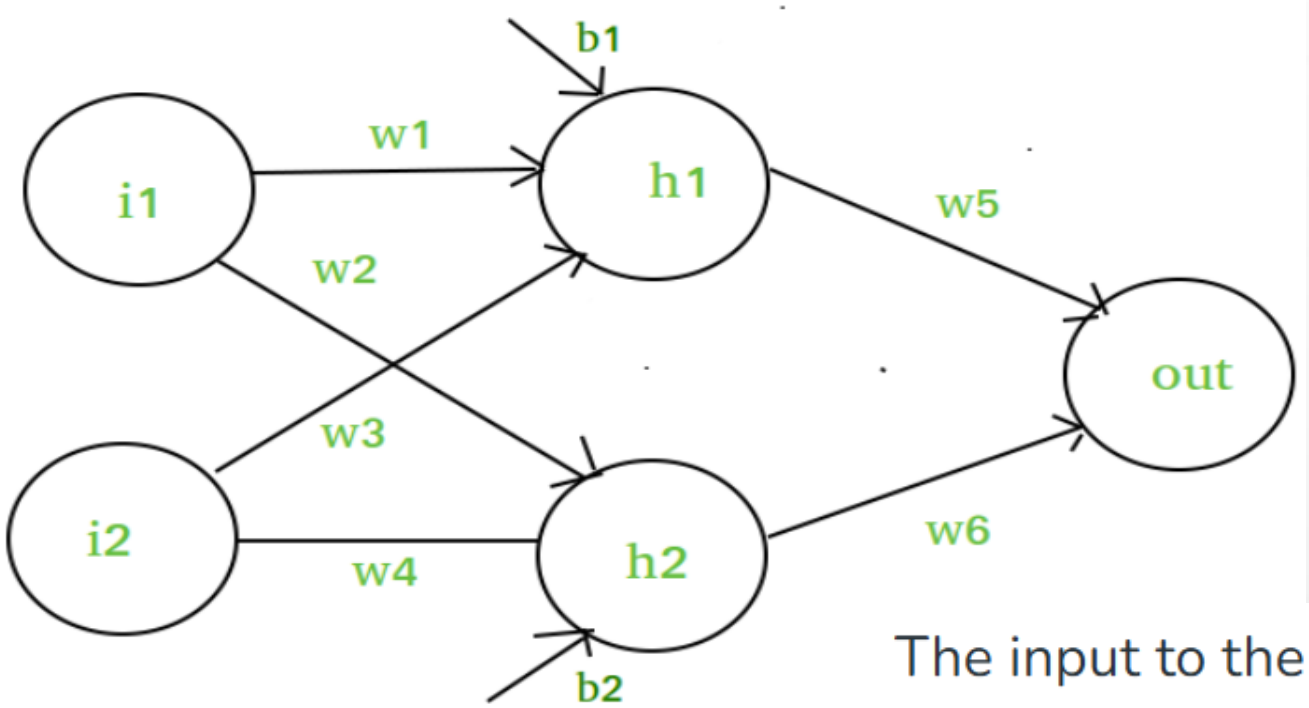
- THE INCLUSION OF THE RELU ACTIVATION FUNCTION *"SIGMA"* ALLOWS *H*1 TO INTRODUCE A NON-LINEAR DECISION BOUNDARY IN THE INPUT SPACE. THIS NON-LINEARITY ENABLES THE NETWORK TO LEARN MORE COMPLEX PATTERNS THAT ARE NOT POSSIBLE WITH A PURELY LINEAR MODEL SUCH AS:
- MODELING FUNCTIONS THAT ARE NOT LINEARLY SEPARABLE.
- INCREASING THE CAPACITY OF THE NETWORK TO FORM MULTIPLE DECISION BOUNDARIES BASED ON THE COMBINATION OF WEIGHTS AND BIASES.

MATHEMATICAL PROOF OF NEED OF NON-LINEARITY IN NEURAL NETWORKS

To illustrate the need for non-linearity in neural networks with a specific example let's consider a network with two input nodes (i_1 and i_2), a single hidden layer containing neurons h_1 and h_2 and an output neuron (out).

We will use w_1, w_2 as weights connecting the inputs to the hidden neuron and w_5 as the weight connecting the hidden neuron to the output. We'll also include biases (b_1 for the hidden neuron and b_2 for the output neuron) to complete the model.

1. **Input Layer**: Two inputs i_1 and i_2.
2. **Hidden Layer**: Two neuron h_1 and h_2
3. **Output Layer**: One output neuron.



The input to the hidden neuron h_1 is calculated as a weighted sum of the inputs plus a bias:

$$h_1 = i_1.w_1 + i_2.w_3 + b_1$$

$$h_2 = i_1.w_2 + i_2.w_4 + b_2$$

The output neuron is then a weighted sum of the hidden neuron's output plus a bias:

$$\text{output} = h_1.w_5 + h_2.w_6 + \text{bias}$$

In order to add non-linearity, we will be using sigmoid activation function in the output layer:

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

$$\text{final output} = \sigma(h_1.w_5 + h_2.w_6 + \text{bias})$$

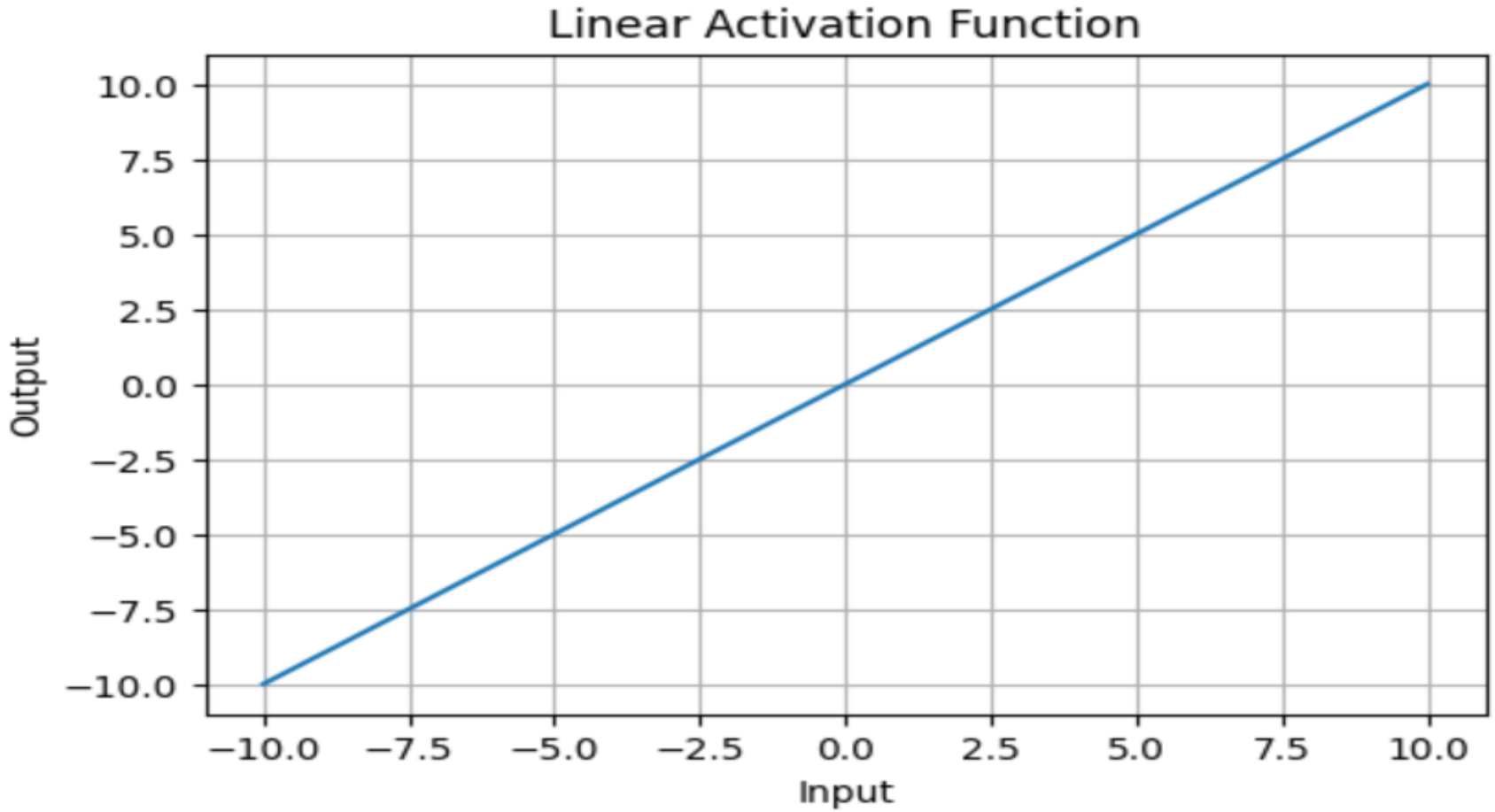
$$\text{final output} = \frac{1}{1+e^{-(h_1.w_5+h_2.w_6+\text{bias})}}$$

This gives the final output of the network after applying the sigmoid activation function in output layers, introducing the desired non-linearity.

TYPES OF ACTIVATION FUNCTIONS IN DEEP LEARNING

1. LINEAR ACTIVATION FUNCTION

- LINEAR ACTIVATION FUNCTION RESEMBLES STRAIGHT LINE DEFINE BY Y=X. NO MATTER HOW MANY LAYERS THE NEURAL NETWORK CONTAINS IF THEY ALL USE LINEAR ACTIVATION FUNCTIONS THE OUTPUT IS A LINEAR COMBINATION OF THE INPUT.
- THE RANGE OF THE OUTPUT SPANS FROM **(–∞ TO +∞).**
- LINEAR ACTIVATION FUNCTION IS USED AT JUST ONE PLACE I.E. OUTPUT LAYER.
- USING LINEAR ACTIVATION ACROSS ALL LAYERS MAKES THE NETWORK'S ABILITY TO LEARN COMPLEX PATTERNS LIMITED.



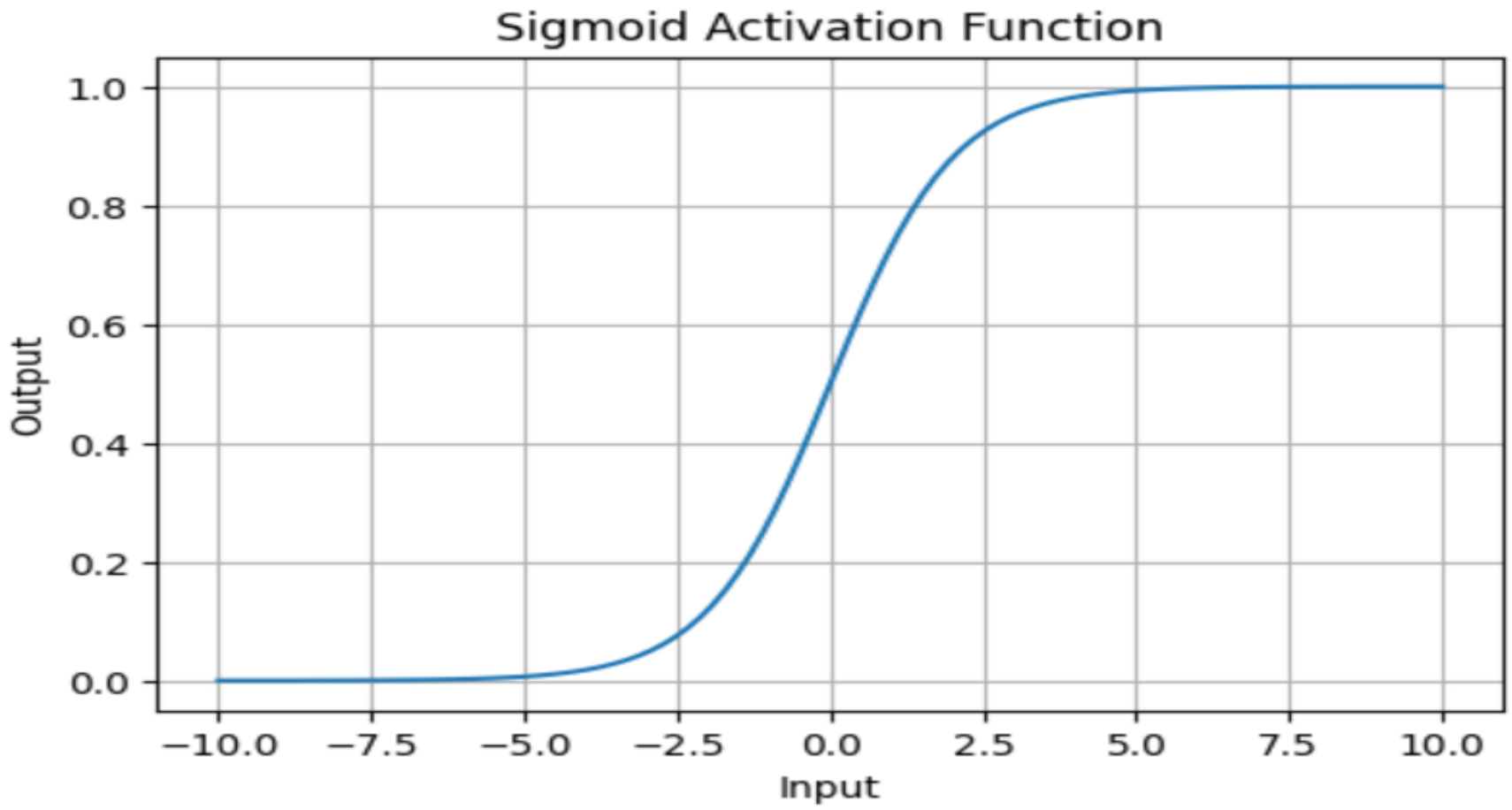
Linear Activation Function or Identity Function returns the input as the output

2. NON-LINEAR ACTIVATION FUNCTIONS

1. SIGMOID FUNCTION

- **SIGMOID ACTIVATION FUNCTION** IS CHARACTERIZED BY 'S' SHAPE. IT IS MATHEMATICALLY DEFINED AS
- THIS FORMULA ENSURES A SM$A = \frac{1}{1+e^{-x}}$ CONTINUOUS OUTPUT THAT IS ESSENTIAL FOR GRADIENT-BASED OPTIMIZATION METHODS.
- IT ALLOWS NEURAL NETWORKS TO HANDLE AND MODEL COMPLEX PATTERNS THAT LINEAR EQUATIONS CANNOT.
- THE OUTPUT RANGES BETWEEN 0 AND 1, HENCE USEFUL FOR BINARY CLASSIFICATION.
- THE FUNCTION EXHIBITS A STEEP GRADIENT WHEN X VALUES ARE BETWEEN -2 AND 2. THIS SENSITIVITY MEANS THAT SMALL CHANGES IN INPUT X CAN CAUSE SIGNIFICANT CHANGES IN OUTPUT Y WHICH IS CRITICAL DURING THE TRAINING PROCESS.

- **Output range**: (0 to 1)
- **Shape**: S-shaped curve.
- **Use case**: Often used in the output layer when we need probabilities (like in binary classification).
- **Problem**: It squashes all values between 0 and 1, so negative inputs still become positive (e.g., –10 → ~0, not negative). This can sometimes make learning slower.



Sigmoid or Logistic Activation Function Graph

2. TANH ACTIVATION FUNCTION

- **TANH FUNCTION** (HYPERBOLIC TANGENT FUNCTION) IS A SHIFTED VERSION OF THE SIGMOID, ALLOWING IT TO STRETCH ACROSS THE Y-AXIS. IT IS DEFINED AS:

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1.$$

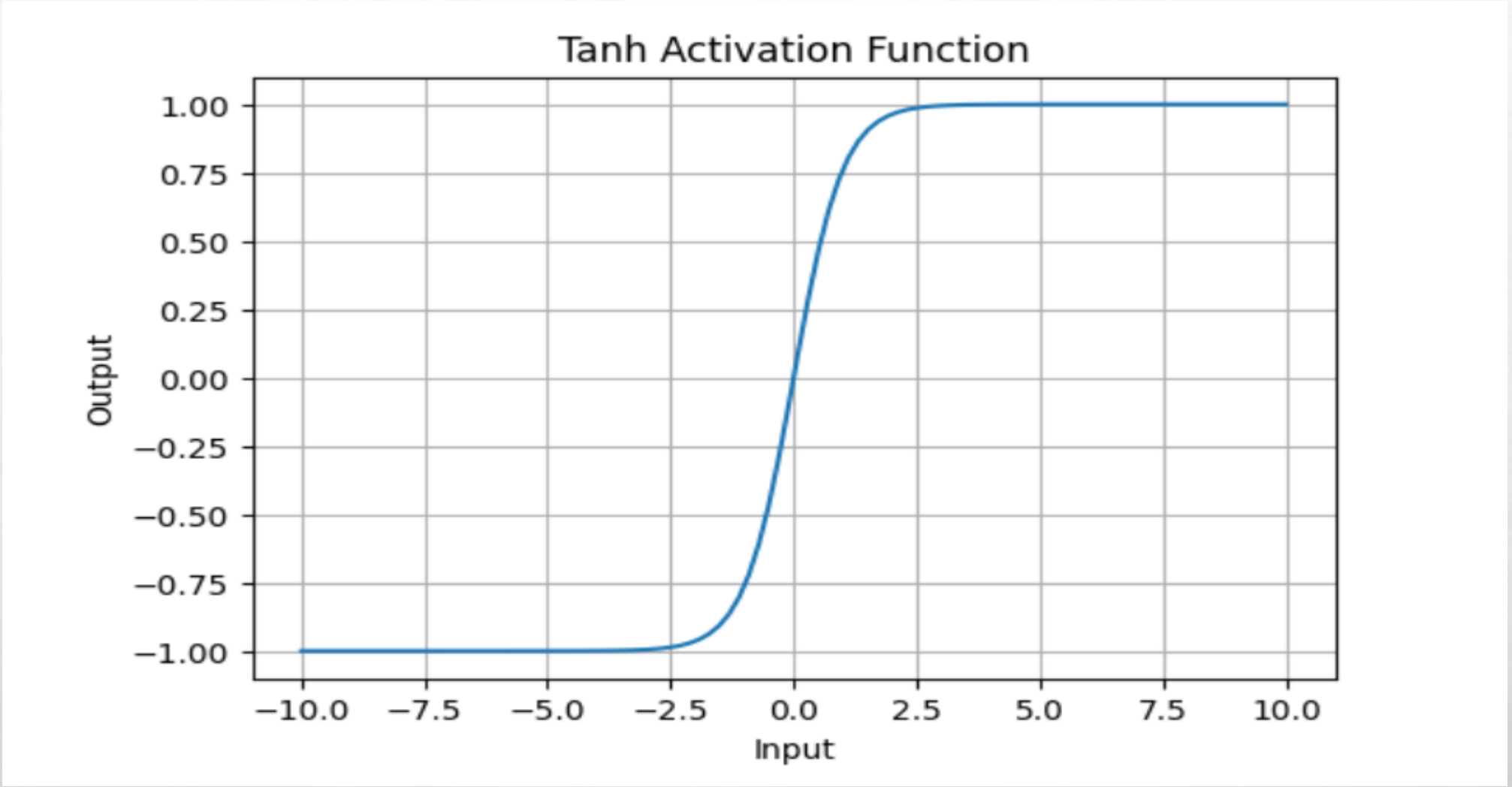
Alternatively, it can be expressed using the sigmoid function:

$$\tanh(x) = 2 \times \text{sigmoid}(2x) - 1$$

- **Value Range**: Outputs values from -1 to +1.
- **Non-linear**: Enables modeling of complex data patterns.
- **Use in Hidden Layers**: Commonly used in hidden layers due to its zero-centered output, facilitating easier learning for subsequent layers.

TANH FUNCTION

- **Output range**: (–1 to 1)
- **Shape**: Also, S-shaped, but centered at 0.
- **Use case**: Good for hidden layers, because it outputs both positive and negative values, which helps the network learn faster.
- **Advantage over Sigmoid**: Since it’s centered at 0, positive/negative outputs make optimization easier.


Tanh Activation Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Output
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Input

DISADVANTAGES OF SIGMOID

- **1. SIGMOID IS NOT ZERO-CENTERED**
- SIGMOID MAPS EVERYTHING INTO **(0, 1)**.
- EXAMPLE:
  - INPUT = –10 → OUTPUT ≈ 0
  - INPUT = +10 → OUTPUT ≈ 1
- NOTICE: **BOTH NEGATIVE AND POSITIVE INPUTS GIVE NON-NEGATIVE OUTPUTS** (ALWAYS ≥ 0).
- THIS CAUSES THE GRADIENTS (DERIVATIVES) DURING BACKPROPAGATION TO ALSO BE ALWAYS POSITIVE OR ALWAYS VERY SMALL.
-

- **2. EFFECT ON WEIGHT UPDATES**
- IN GRADIENT DESCENT, WEIGHT UPDATES DEPEND ON THE **SIGN** OF THE GRADIENT.
- IF THE ACTIVATION IS ALWAYS POSITIVE (LIKE SIGMOID), THE **GRADIENTS FOR WEIGHTS KEEP POINTING IN THE SAME DIRECTION**, INSTEAD OF BALANCING BETWEEN POSITIVE AND NEGATIVE CORRECTIONS.
- THIS LEADS TO **ZIG-ZAGGING UPDATES** AND SLOWER CONVERGENCE.
-

- **3. VANISHING GRADIENT PROBLEM**
- FOR VERY LARGE POSITIVE OR NEGATIVE INPUTS, THE SIGMOID SATURATES NEAR 0 OR 1.
- IN THESE REGIONS, THE GRADIENT (SLOPE) ≈ 0.
- THIS MEANS WEIGHTS STOP UPDATING EFFECTIVELY, MAKING LEARNING EXTREMELY SLOW.

WHY TANH IS BETTER IN HIDDEN LAYERS

- TANH OUTPUTS IN **(–1 TO 1)** → ZERO-CENTERED.
- THIS MEANS NEGATIVE INPUTS GIVE NEGATIVE OUTPUTS, POSITIVE INPUTS GIVE POSITIVE OUTPUTS.
- SO, THE GRADIENT UPDATES CAN **MOVE IN BOTH DIRECTIONS**, LEADING TO **FASTER AND SMOOTHER LEARNING** COMPARED TO SIGMOID.

- An activation function is **zero-centered** if its output can be both **negative** and **positive**, with 0 as the “center point.”
- Example:
- **Tanh** outputs values between –1 and +1.
  - If input is negative → output is negative (e.g., tanh(–2) ≈ –0.96).
  - If input is positive → output is positive (e.g., tanh(+2) ≈ +0.96).
  - If input = 0 → output = 0.
- So, the function is “balanced” around **0** → that’s why we call it **zero-centered**.

3. RELU (RECTIFIED LINEAR UNIT) FUNCTION

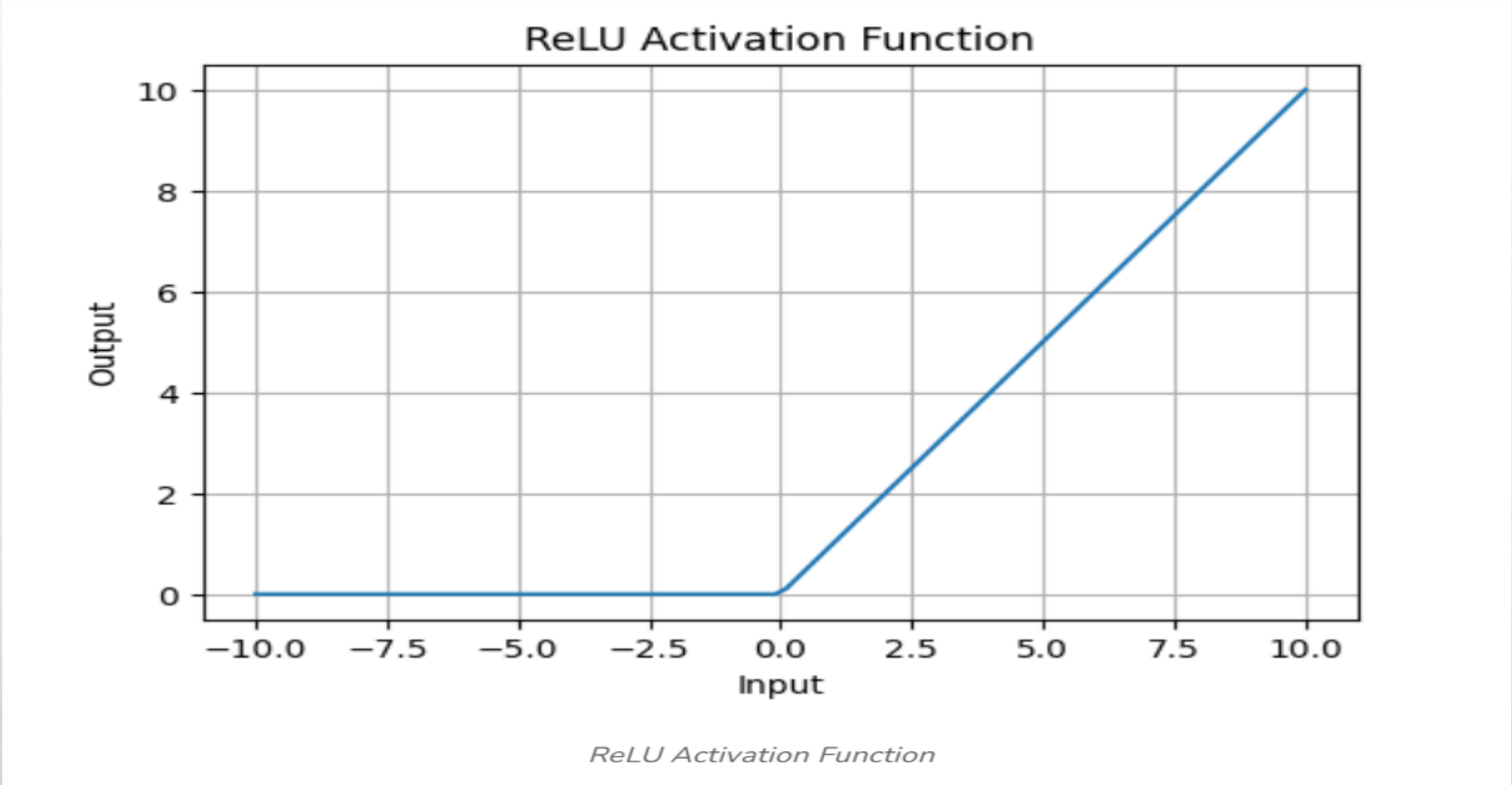
- **RELU ACTIVATION** IS DEFINED BY **A(X)=MAX(0,X),** THIS MEANS THAT IF THE INPUT X IS POSITIVE, RELU RETURNS X, IF THE INPUT IS NEGATIVE, IT RETURNS 0.
- **VALUE RANGE**: **[0,∞),** MEANING THE FUNCTION ONLY OUTPUTS NON-NEGATIVE VALUES.
- **NATURE**: IT IS A NON-LINEAR ACTIVATION FUNCTION, ALLOWING NEURAL NETWORKS TO LEARN COMPLEX PATTERNS AND MAKING BACKPROPAGATION MORE EFFICIENT.
- **ADVANTAGE OVER OTHER ACTIVATION:** RELU IS LESS COMPUTATIONALLY EXPENSIVE THAN TANH AND SIGMOID BECAUSE IT INVOLVES SIMPLER MATHEMATICAL OPERATIONS. AT A TIME ONLY A FEW NEURONS ARE ACTIVATED MAKING THE NETWORK SPARSE MAKING IT EFFICIENT AND EASY FOR COMPUTATION.

ADVANTAGES OF RELU

- RELU = F(X) = MAX(0, X)
- ITS DERIVATIVE IS EITHER **1 (FOR POSITIVE X)** OR **0 (FOR NEGATIVE X)**.
- FOR POSITIVE INPUTS, GRADIENT PASSES BACK **WITHOUT SHRINKING**.
- THIS KEEPS GRADIENTS **STRONG AND STABLE** ACROSS MANY LAYERS.
- SO DEEPER LAYERS **CONTINUE LEARNING EFFECTIVELY**.

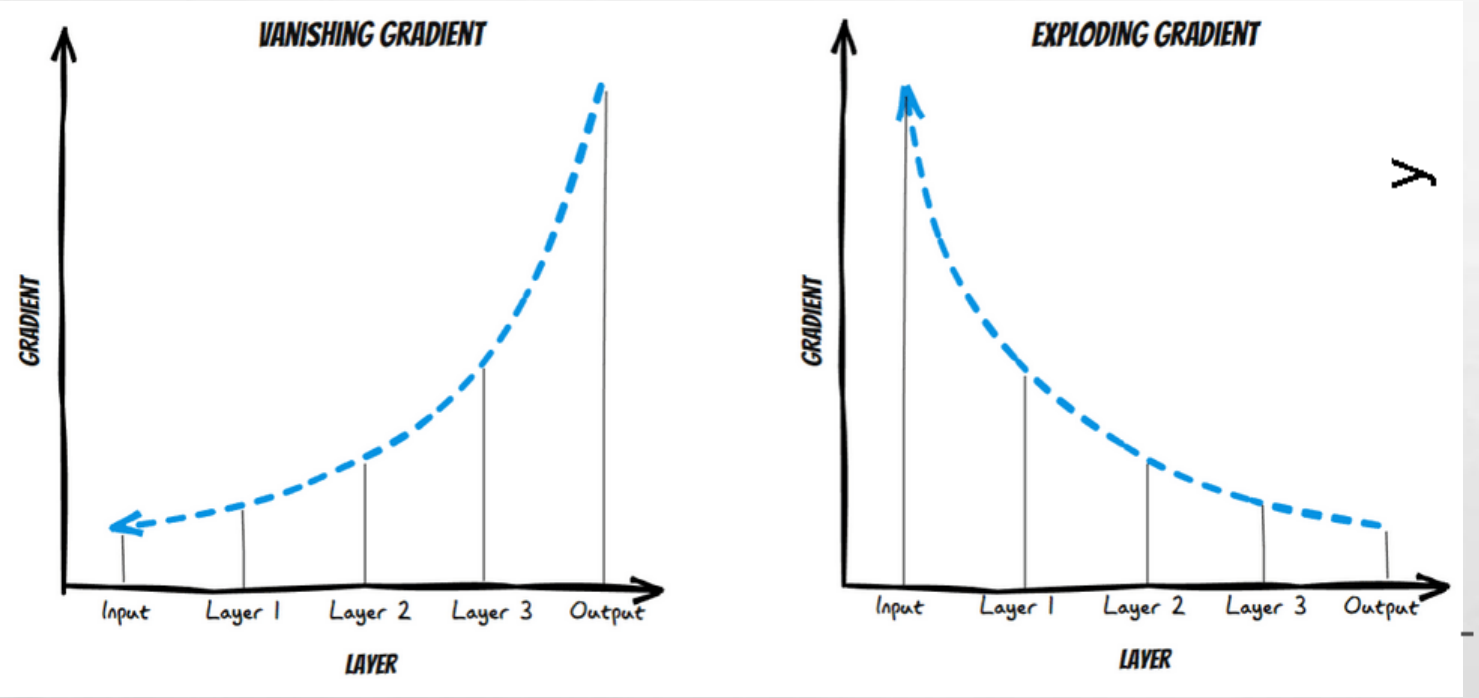
- **WHY IT’S USEFUL**
- **SIMPLE & FAST** → JUST COMPARE WITH 0.
- **NO SATURATION** FOR POSITIVE INPUTS (UNLIKE SIGMOID/TANH) → AVOIDS THE VANISHING GRADIENT PROBLEM IN MOST CASES.
- MAKES TRAINING **MUCH FASTER** IN DEEP NETWORKS.
-

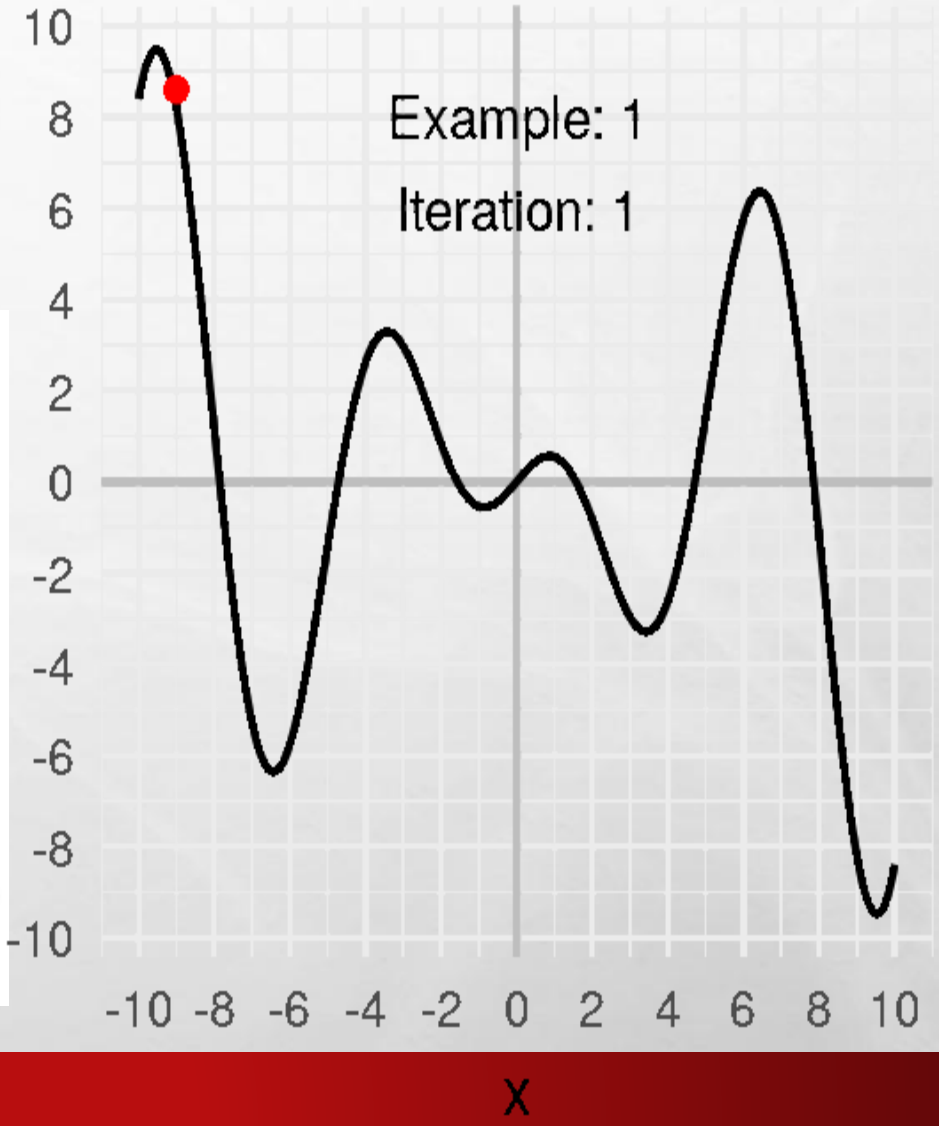
- **DRAWBACK (THE "DYING RELU" PROBLEM)**
- IF TOO MANY INPUTS ARE NEGATIVE, OUTPUTS ARE ALWAYS 0.
- NEURON MAY “DIE” (STOP LEARNING) BECAUSE GRADIENT IS ALSO 0.



ReLU Activation Function

VANISHING GRADIENT





3. EXPONENTIAL LINEAR UNITS

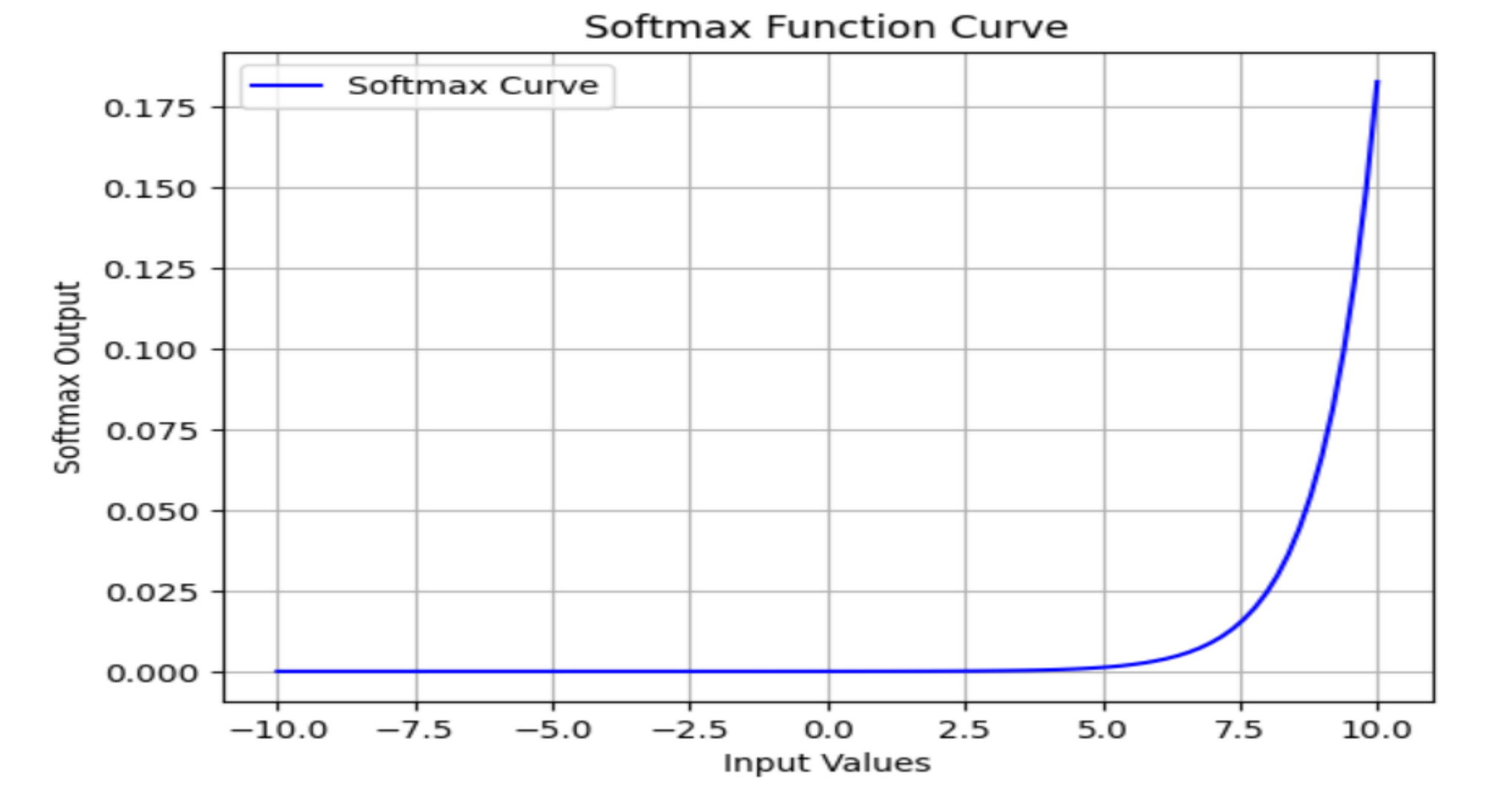
1. SOFTMAX FUNCTION

- **SOFTMAX FUNCTION** IS DESIGNED TO HANDLE MULTI-CLASS CLASSIFICATION PROBLEMS. IT TRANSFORMS RAW OUTPUT SCORES FROM A NEURAL NETWORK INTO PROBABILITIES. IT WORKS BY SQUASHING THE OUTPUT VALUES OF EACH CLASS INTO THE RANGE OF 0 TO 1 WHILE ENSURING THAT THE SUM OF ALL PROBABILITIES EQUALS 1.
- SOFTMAX IS A NON-LINEAR ACTIVATION FUNCTION.
- THE SOFTMAX FUNCTION ENSURES THAT EACH CLASS IS ASSIGNED A PROBABILITY, HELPING TO IDENTIFY WHICH CLASS THE INPUT BELONGS TO.

- IMAGINE A MODEL GIVES RAW SCORES: [2.0, 1.0, 0.1].
- THESE NUMBERS ARE NOT EASY TO INTERPRET.
- SOFTMAX TURNS THEM INTO PROBABILITIES: [0.65, 0.24, 0.11].
- NOW YOU CAN SAY:
  - CLASS 1 → 65% CHANCE
  - CLASS 2 → 24% CHANCE
  - CLASS 3 → 11% CHANCE

WHERE IT'S USED

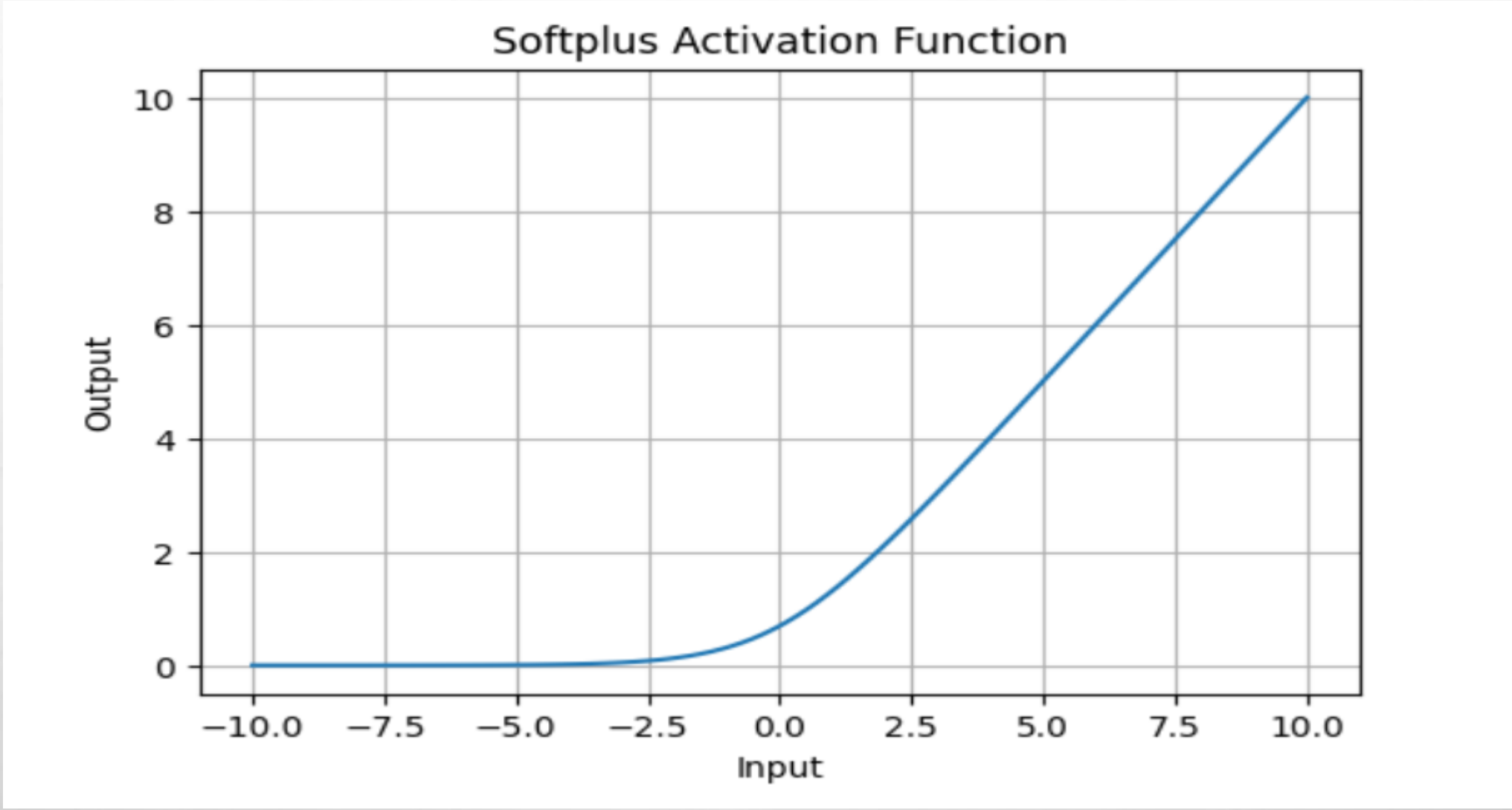
- ALMOST ALWAYS IN THE **OUTPUT LAYER OF CLASSIFICATION PROBLEMS** (WHEN YOU HAVE MULTIPLE CLASSES).
- EXAMPLE: IMAGE CLASSIFICATION (CAT, DOG, HORSE).


Softmax Function Curve
Softmax Curve
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Softmax Output
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Input Values

2. SOFTPLUS FUNCTION

- **SOFTPLUS FUNCTION** IS DEFINED MATHEMATICALLY AS: $A(x) = \log(1 + e^x)$

- THIS EQUATION ENSURES THAT THE OUTPUT IS ALWAYS POSITIVE AND DIFFERENTIABLE AT ALL POINTS WHICH IS AN ADVANTAGE OVER THE TRADITIONAL RELU FUNCTION.
- **NATURE**: THE SOFTPLUS FUNCTION IS NON-LINEAR.
- **RANGE**: THE FUNCTION OUTPUTS VALUES IN THE RANGE **(0,∞),** SIMILAR TO RELU, BUT WITHOUT THE HARD ZERO THRESHOLD THAT RELU HAS.
- **SMOOTHNESS**: SOFTPLUS IS A SMOOTH, CONTINUOUS FUNCTION, MEANING IT AVOIDS THE SHARP DISCONTINUITIES OF RELU WHICH CAN SOMETIMES LEAD TO PROBLEMS DURING OPTIMIZATION.

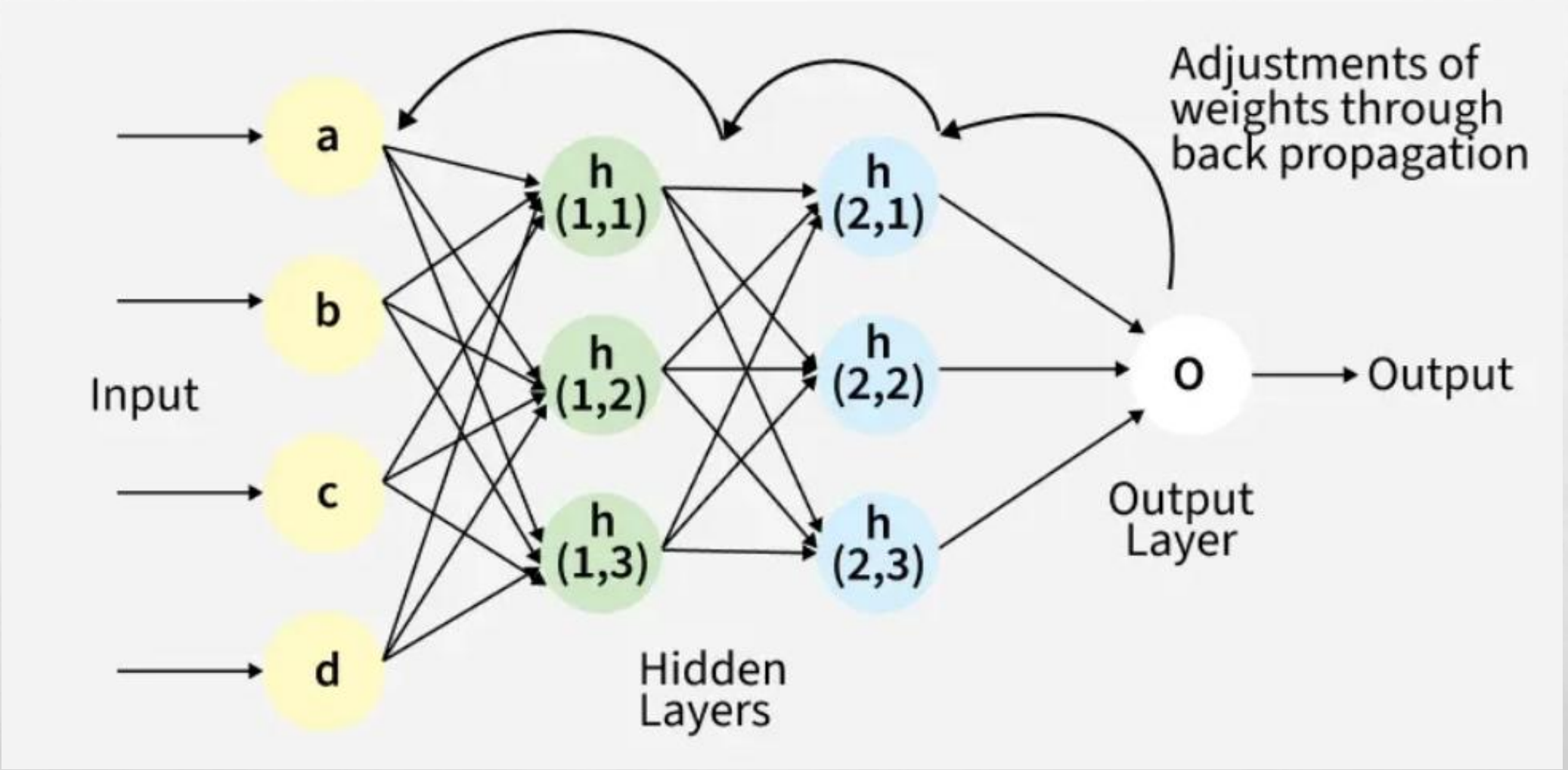

Softplus Activation Function
Output
Input
10
8
6
4
2
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

IMPACT OF ACTIVATION FUNCTIONS ON MODEL PERFORMANCE

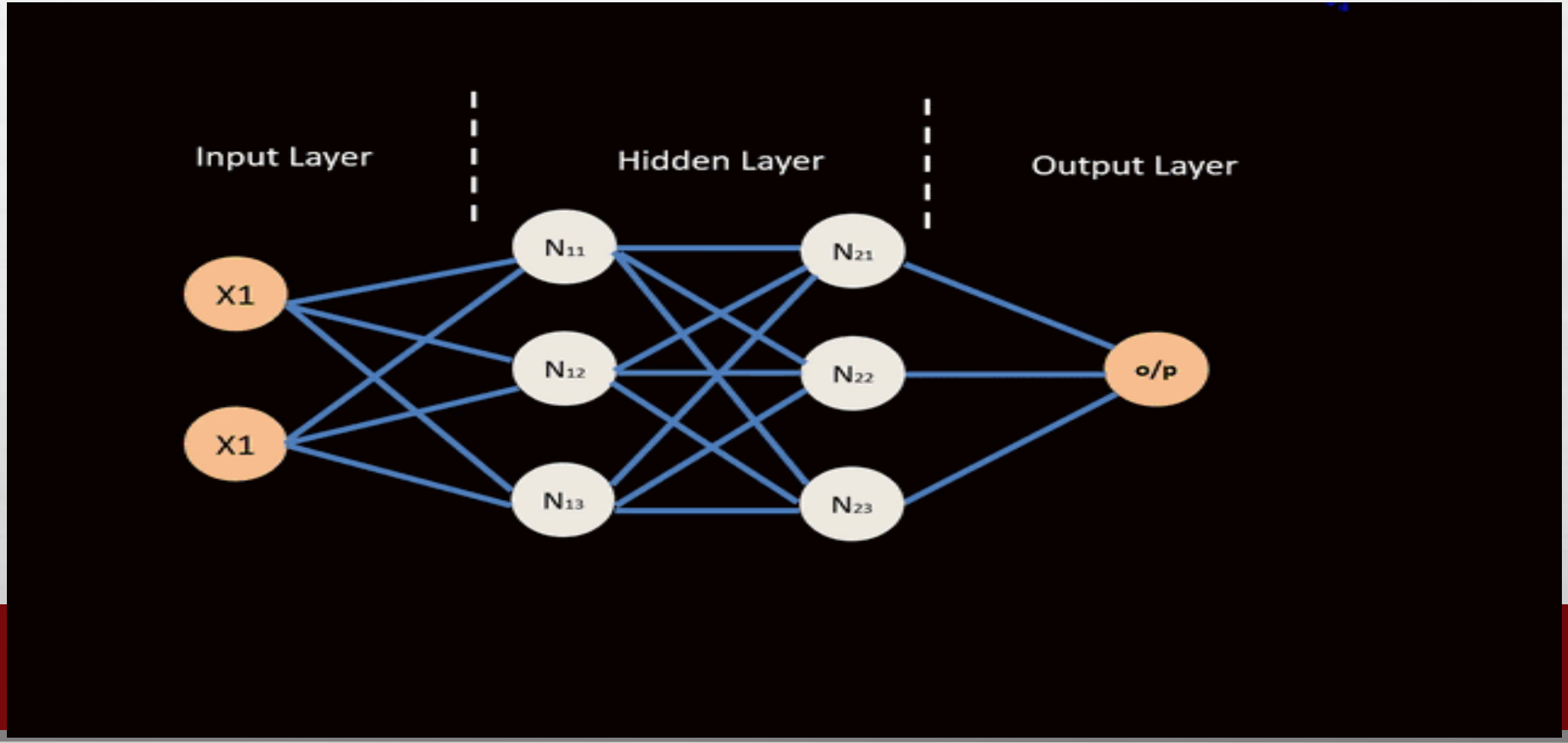
- THE CHOICE OF ACTIVATION FUNCTION HAS A DIRECT IMPACT ON THE PERFORMANCE OF A NEURAL NETWORK IN SEVERAL WAYS:
- **CONVERGENCE SPEED:** FUNCTIONS LIKE RELU ALLOW FASTER TRAINING BY AVOIDING THE VANISHING GRADIENT PROBLEM WHILE SIGMOID AND TANH CAN SLOW DOWN CONVERGENCE IN DEEP NETWORKS.
- **GRADIENT FLOW:** ACTIVATION FUNCTIONS LIKE RELU ENSURE BETTER GRADIENT FLOW, HELPING DEEPER LAYERS LEARN EFFECTIVELY. IN CONTRAST SIGMOID CAN LEAD TO SMALL GRADIENTS, HINDERING LEARNING IN DEEP LAYERS.
- **MODEL COMPLEXITY:** ACTIVATION FUNCTIONS LIKE SOFTMAX ALLOW THE MODEL TO HANDLE COMPLEX MULTI-CLASS PROBLEMS, WHEREAS SIMPLER FUNCTIONS LIKE RELU OR LEAKY RELU ARE USED FOR BASIC LAYERS.

BACKPROPAGATION

- BACK PROPAGATION IS ALSO KNOWN AS "BACKWARD PROPAGATION OF ERRORS" IS A METHOD USED TO TRAIN NEURAL NETWORK . ITS GOAL IS TO REDUCE THE DIFFERENCE BETWEEN THE MODEL'S PREDICTED OUTPUT AND THE ACTUAL OUTPUT BY ADJUSTING THE WEIGHTS AND BIASES IN THE NETWORK.
- IT WORKS ITERATIVELY TO ADJUST WEIGHTS AND BIAS TO MINIMIZE THE COST FUNCTION. IN EACH EPOCH THE MODEL ADAPTS THESE PARAMETERS BY REDUCING LOSS BY FOLLOWING THE ERROR GRADIENT. IT OFTEN USES OPTIMIZATION ALGORITHMS LIKE **GRADIENT DESCENT** OR **STOCHASTIC GRADIENT DESCENT**. THE ALGORITHM COMPUTES THE GRADIENT USING THE CHAIN RULE FROM CALCULUS ALLOWING IT TO EFFECTIVELY NAVIGATE COMPLEX LAYERS IN THE NEURAL NETWORK TO MINIMIZE THE COST FUNCTION.


Adjustments of weights through back propagation
a
b
c
d
Input
h (1,1)
h (1,2)
h (1,3)
h (2,1)
h (2,2)
h (2,3)
Hidden Layers
O
Output
Output Layer

BACKPROPAGATION

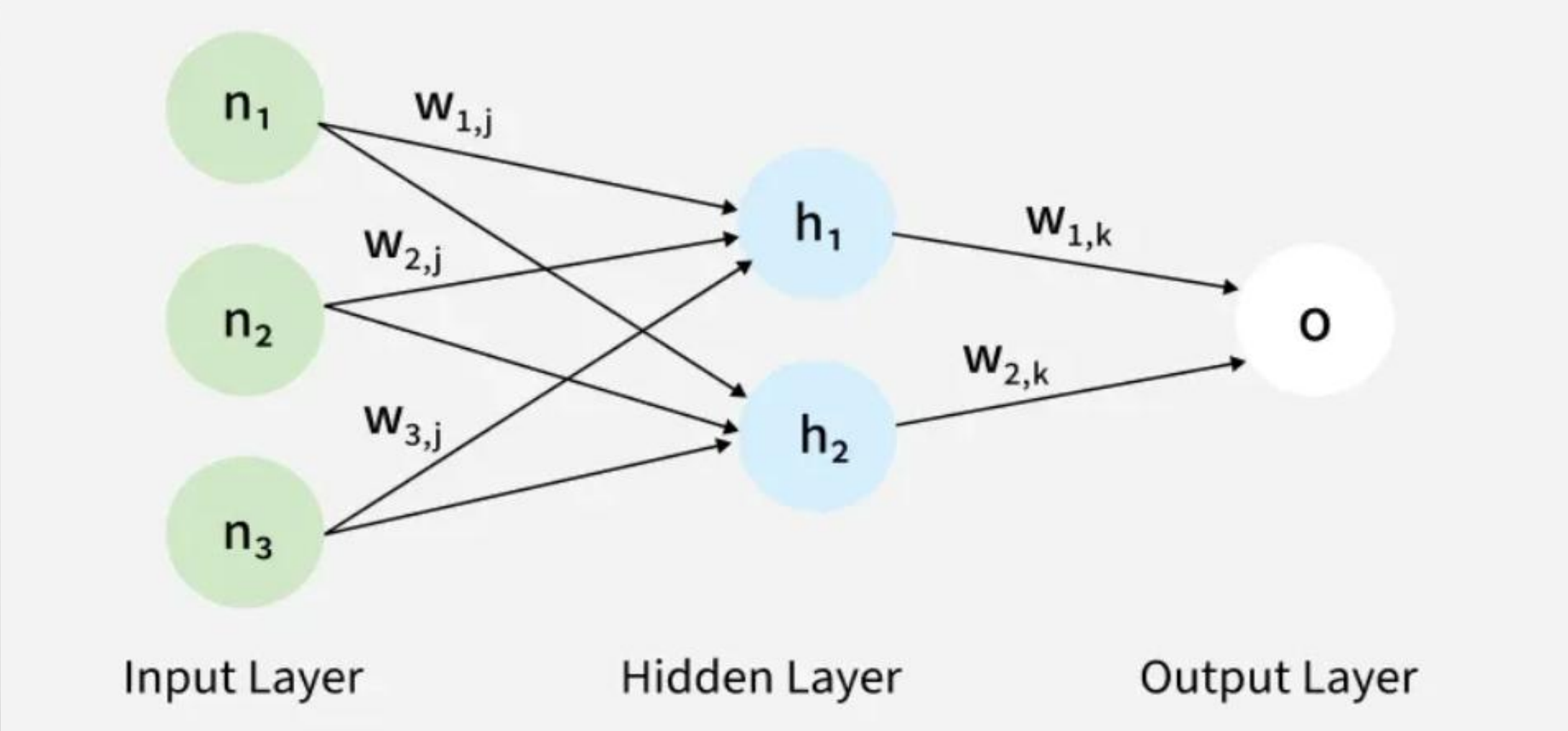


WORKING OF BACK PROPAGATION ALGORITHM

- THE BACK PROPAGATION ALGORITHM INVOLVES TWO MAIN STEPS: THE **FORWARD PASS** AND THE **BACKWARD PASS**.

FORWARD PASS WORK

- IN **FORWARD PASS** THE INPUT DATA IS FED INTO THE INPUT LAYER. THESE INPUTS COMBINED WITH THEIR RESPECTIVE WEIGHTS ARE PASSED TO HIDDEN LAYERS. FOR EXAMPLE, IN A NETWORK WITH TWO HIDDEN LAYERS (H1 AND H2) THE OUTPUT FROM H1 SERVES AS THE INPUT TO H2. BEFORE APPLYING AN ACTIVATION FUNCTION, A BIAS IS ADDED TO THE WEIGHTED INPUTS.
- EACH HIDDEN LAYER COMPUTES THE WEIGHTED SUM (`A`) OF THE INPUTS THEN APPLIES AN ACTIVATION FUNCTION LIKE **RELU (RECTIFIED LINEAR UNIT)** TO OBTAIN THE OUTPUT (`O`). THE OUTPUT IS PASSED TO THE NEXT LAYER WHERE AN ACTIVATION FUNCTION SUCH AS **SOFTMAX** CONVERTS THE WEIGHTED OUTPUTS INTO PROBABILITIES FOR CLASSIFICATION.

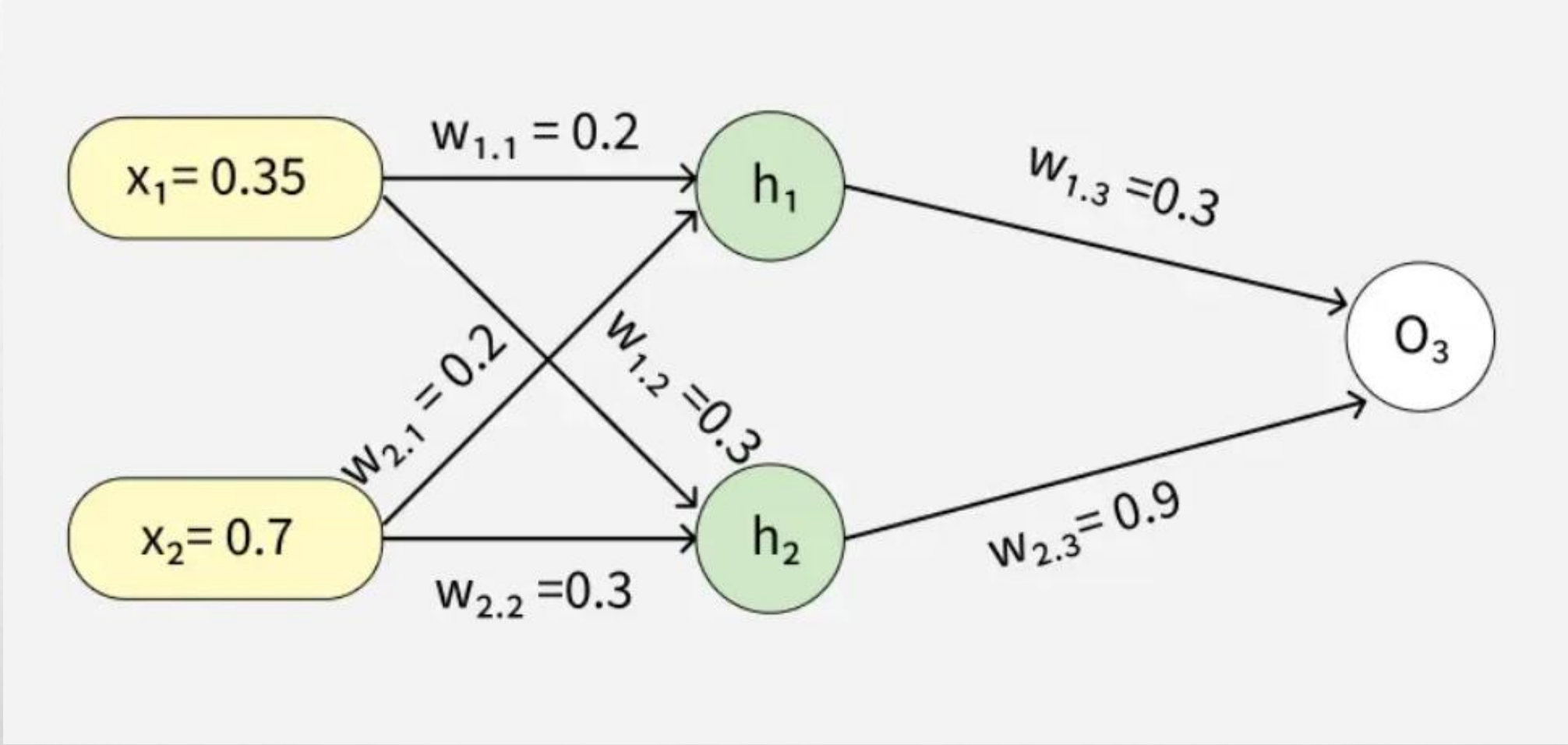

n_1
n_2
n_3
$w_{1,j}$
$w_{2,j}$
$w_{3,j}$
h_1
h_2
$w_{1,k}$
$w_{2,k}$
O
Input Layer
Hidden Layer
Output Layer

BACKWARD PASS

- IN THE BACKWARD PASS THE ERROR (THE DIFFERENCE BETWEEN THE PREDICTED AND ACTUAL OUTPUT) IS PROPAGATED BACK THROUGH THE NETWORK TO ADJUST THE WEIGHTS AND BIASES. ONE COMMON METHOD FOR ERROR CALCULATION IS THE **MEAN SQUARED ERROR (MSE)** GIVEN BY:
- $\mathbf{MSE} = \left(\boldsymbol{PREDICTED\ OUTPUT} - \boldsymbol{ACTUAL\ OUTPUT}\right)^{\mathbf{2}}$
- ONCE THE ERROR IS CALCULATED THE NETWORK ADJUSTS WEIGHTS USING **GRADIENTS** WHICH ARE COMPUTED WITH THE CHAIN RULE. THESE GRADIENTS INDICATE HOW MUCH EACH WEIGHT AND BIAS SHOULD BE ADJUSTED TO MINIMIZE THE ERROR IN THE NEXT ITERATION. THE BACKWARD PASS CONTINUES LAYER BY LAYER ENSURING THAT THE NETWORK LEARNS AND IMPROVES ITS PERFORMANCE. THE ACTIVATION FUNCTION THROUGH ITS DERIVATIVE PLAYS A CRUCIAL ROLE IN COMPUTING THESE GRADIENTS DURING BACK PROPAGATION.

EXAMPLE OF BACK PROPAGATION IN MACHINE LEARNING

- ASSUME THE NEURONS USE THE SIGMOID ACTIVATION FUNCTION FOR THE FORWARD AND BACKWARD PASS. THE TARGET OUTPUT IS 0.5 AND THE LEARNING RATE IS 1.


$x_1 = 0.35$
$x_2 = 0.7$
$w_{1.1} = 0.2$
$w_{2.1} = 0.2$
$w_{1.2} = 0.3$
$w_{2.2} = 0.3$
h_1
h_2
$w_{1.3} = 0.3$
$w_{2.3} = 0.9$
O_3

FORWARD PROPAGATION

1. Initial Calculation

The weighted sum at each node is calculated using:

$$aj = \sum(wi, j * xi)$$

Where,

- a_j is the weighted sum of all the inputs and weights at each node
- $w_{i,j}$ represents the weights between the i^{th} input and the j^{th} neuron
- x_i represents the value of the i^{th} input

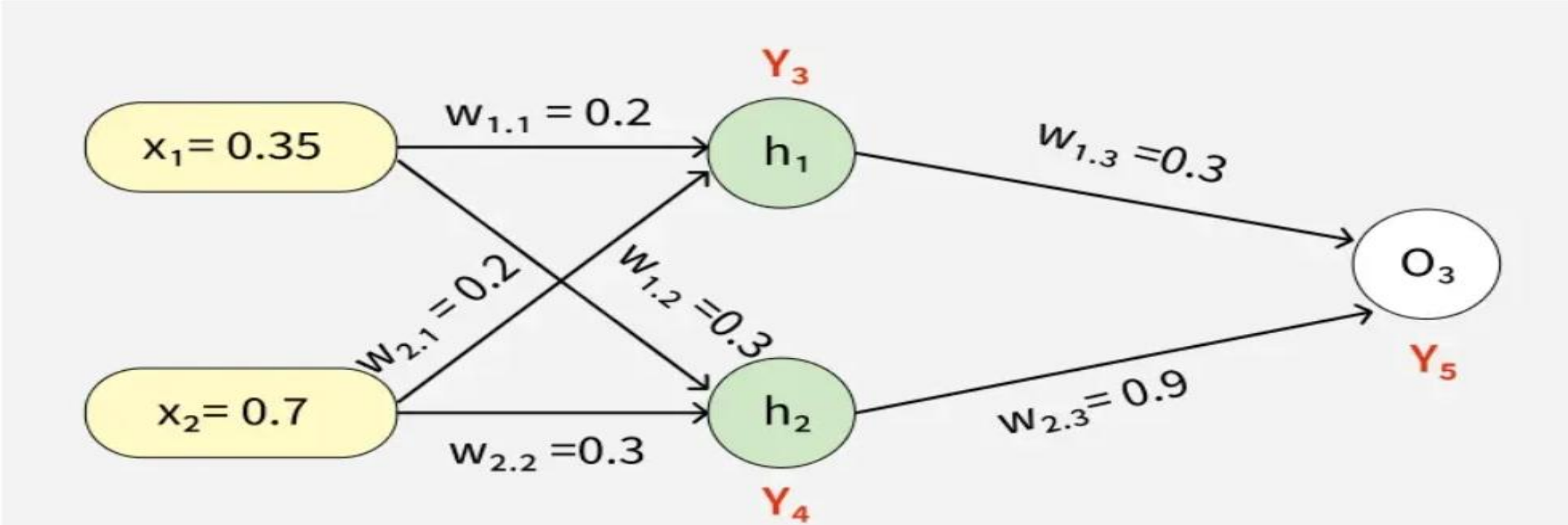
o **(output):** After applying the activation function to a, we get the output of the neuron:

$$o_j = activation\ function(a_j)$$

2. Sigmoid Function

The sigmoid function returns a value between 0 and 1, introducing non-linearity into the model.

$$y_j = \frac{1}{1+e^{-a_j}}$$



3. Computing Outputs

At h1 node

$$\begin{aligned} a_1 &= (w_{1,1}x_1) + (w_{2,1}x_2) \\ &= (0.2 * 0.35) + (0.2 * 0.7) \\ &= 0.21 \end{aligned}$$

Once we calculated the a_1 value, we can now proceed to find the y_3 value:

$$y_j = F(a_j) = \frac{1}{1+e^{-a_1}}$$

$$y_3 = F(0.21) = \frac{1}{1+e^{-0.21}}$$

$$y_3 = 0.56$$

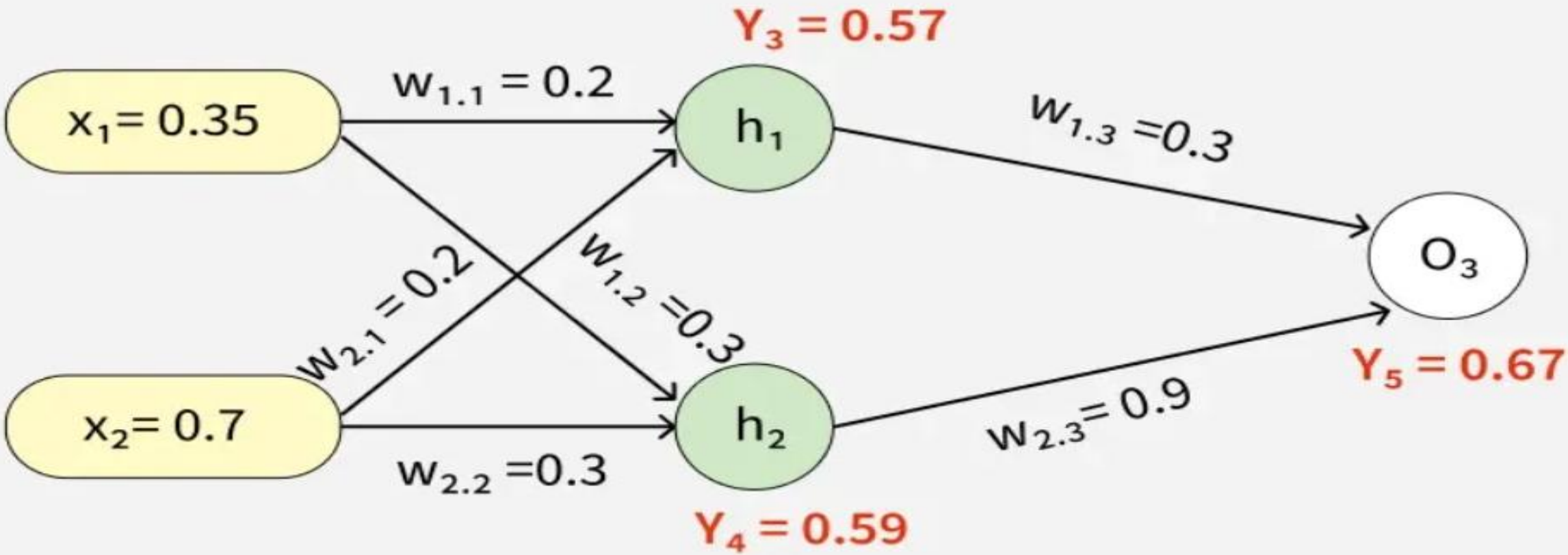
Similarly find the values of y_4 at **h_2** and y_5 at O_3

$$a_2 = (w_{1,2} * x_1) + (w_{2,2} * x_2) = (0.3 * 0.35) + (0.3 * 0.7) = 0.315$$

$$y_4 = F(0.315) = \frac{1}{1+e^{-0.315}}$$

$$a3 = (w_{1,3} * y_3) + (w_{2,3} * y_4) = (0.3 * 0.57) + (0.9 * 0.59) = 0.702$$

$$y_5 = F(0.702) = \frac{1}{1+e^{-0.702}} = 0.67$$



4. Error Calculation

Our actual output is 0.5 but we obtained 0.67. To calculate the error we can use the below formula:

$$Error_j = y_{target} - y_5$$

$$=> 0.5 - 0.67 = -0.17$$

Using this error value we will be backpropagating.

BACK PROPAGATION

1. Calculating Gradients

The change in each weight is calculated as:

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

Where:

- δ_j is the error term for each unit,
- η is the learning rate.

2. Output Unit Error

For O3:

$$\delta_5 = y_5(1 - y_5)(y_{target} - y_5)$$

$$= 0.67(1 - 0.67)(-0.17) = -0.0376$$

3. Hidden Unit Error

For h1:

$$\delta_3 = y_3(1 - y_3)(w_{1,3} \times \delta_5)$$

$$= 0.56(1 - 0.56)(0.3 \times -0.0376) = -0.0027$$

For h2:

$$\delta_4 = y_4(1 - y_4)(w_{2,3} \times \delta_5)$$

$$= 0.59(1 - 0.59)(0.9 \times -0.0376) = -0.0819$$

4. Weight Updates

For the weights from hidden to output layer:

$$\Delta w_{2,3} = 1 \times (-0.0376) \times 0.59 = -0.022184$$

New weight:

$$w_{2,3}(\text{new}) = -0.022184 + 0.9 = 0.877816$$

For weights from input to hidden layer:

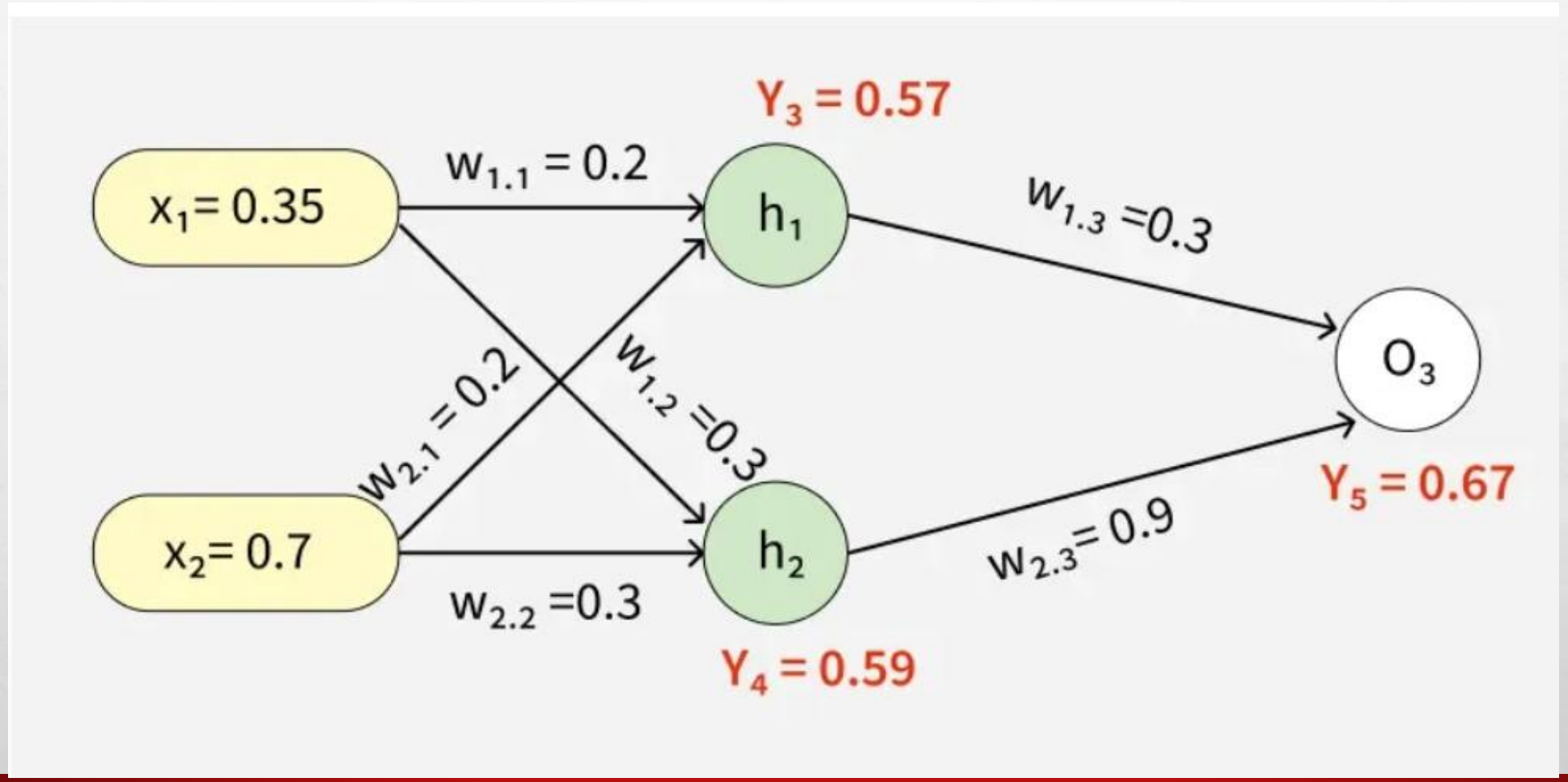
$$\Delta w_{1,1} = 1 \times (-0.0027) \times 0.35 = 0.000945$$

New weight:

$$w_{1,1}(\text{new}) = 0.000945 + 0.2 = 0.200945$$

Similarly other weights are updated:

- $w_{1,2}(\text{new}) = 0.273225$
- $w_{1,3}(\text{new}) = 0.086615$
- $w_{2,1}(\text{new}) = 0.269445$
- $w_{2,2}(\text{new}) = 0.18534$


$Y_3 = 0.57$
$x_1 = 0.35$
$w_{1.1} = 0.2$
h_1
$w_{1.3} = 0.3$
O_3
$w_{2.1} = 0.2$
$w_{1.2} = 0.3$
$Y_5 = 0.67$
$x_2 = 0.7$
h_2
$w_{2.3} = 0.9$
$w_{2.2} = 0.3$
$Y_4 = 0.59$

After updating the weights the forward pass is repeated yielding:

- $y_3 = 0.57$
- $y_4 = 0.56$
- $y_5 = 0.61$

Since $y_5 = 0.61$ is still not the target output the process of calculating the error and backpropagating continues until the desired output is reached.

This process demonstrates how Back Propagation iteratively updates weights by minimizing errors until the network accurately predicts the output.

$$Error = y_{target} - y_5$$

$$= 0.5 - 0.61 = -0.11$$

This process is said to be continued until the actual output is gained by the neural network.

LOSS FUNCTIONS

- A **LOSS FUNCTION** IS A MATHEMATICAL WAY TO MEASURE HOW GOOD OR BAD A MODEL'S PREDICTIONS ARE COMPARED TO THE ACTUAL RESULTS. IT GIVES A SINGLE NUMBER THAT TELLS US HOW FAR OFF THE PREDICTIONS ARE. THE SMALLER THE NUMBER, THE BETTER THE MODEL IS DOING. LOSS FUNCTIONS ARE USED TO TRAIN MODELS.

LOSS FUNCTIONS ARE IMPORTANT BECAUSE THEY:

- **GUIDE MODEL TRAINING:** DURING TRAINING, ALGORITHMS SUCH AS GRADIENT DESCENT USE THE LOSS FUNCTION TO ADJUST THE MODEL'S PARAMETERS AND TRY TO REDUCE THE ERROR AND IMPROVE THE MODEL'S PREDICTIONS.
- **MEASURE PERFORMANCE:** BY FINDING THE DIFFERENCE BETWEEN PREDICTED AND ACTUAL VALUES AND IT CAN BE USED FOR EVALUATING THE MODEL'S PERFORMANCE.
- **AFFECT LEARNING BEHAVIOR:** DIFFERENT LOSS FUNCTIONS CAN MAKE THE MODEL LEARN IN DIFFERENT WAYS DEPENDING ON WHAT KIND OF MISTAKES THEY MAKE.

KINDS OF LOSS FUNCTIONS

1. REGRESSION LOSS FUNCTIONS

- THESE ARE USED WHEN YOUR MODEL NEEDS TO **PREDICT A CONTINUOUS NUMBER** SUCH AS PREDICTING THE PRICE OF A PRODUCT OR AGE OF A PERSON. POPULAR REGRESSION LOSS FUNCTIONS ARE:

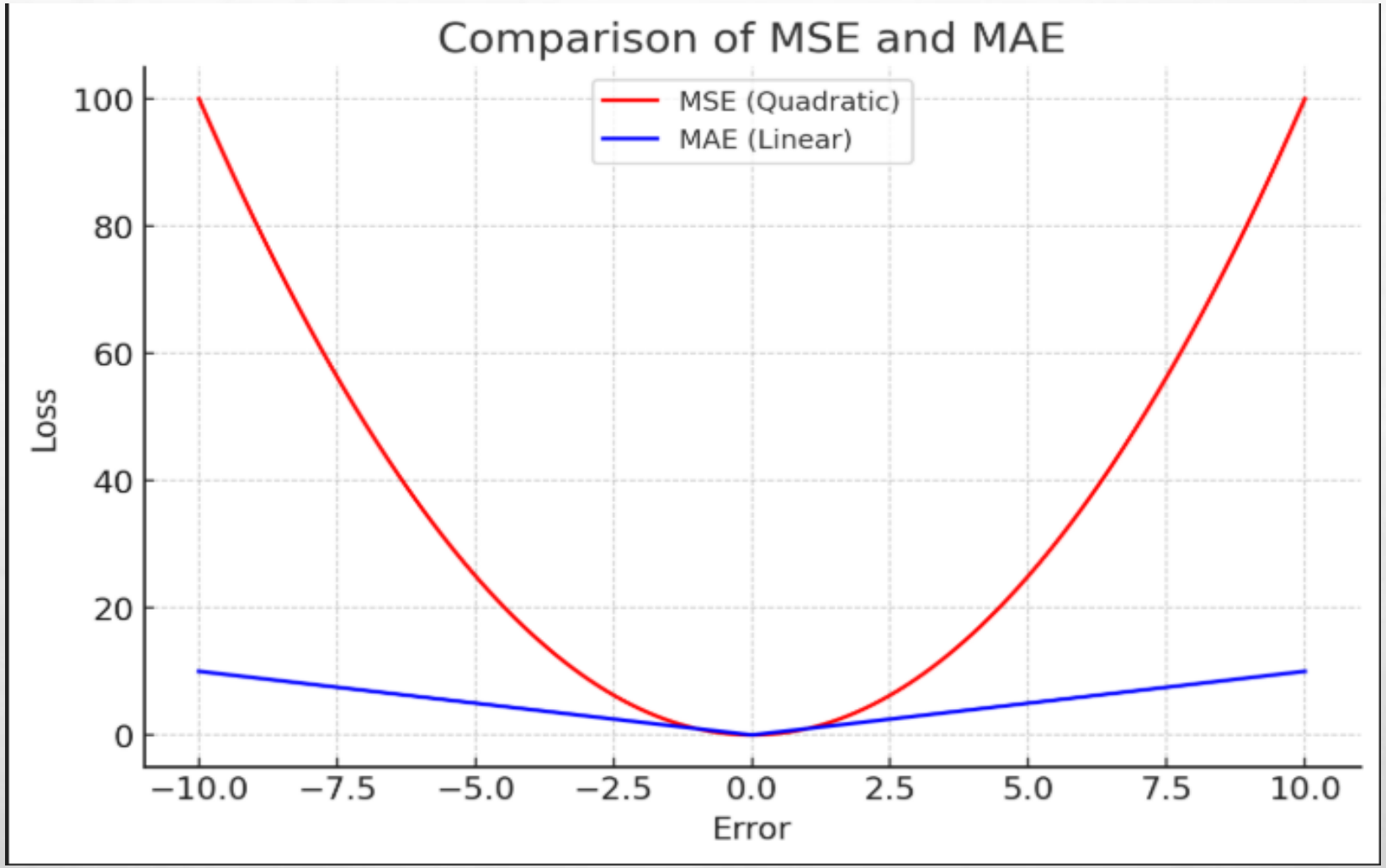
1. MEAN SQUARED ERROR (MSE) LOSS

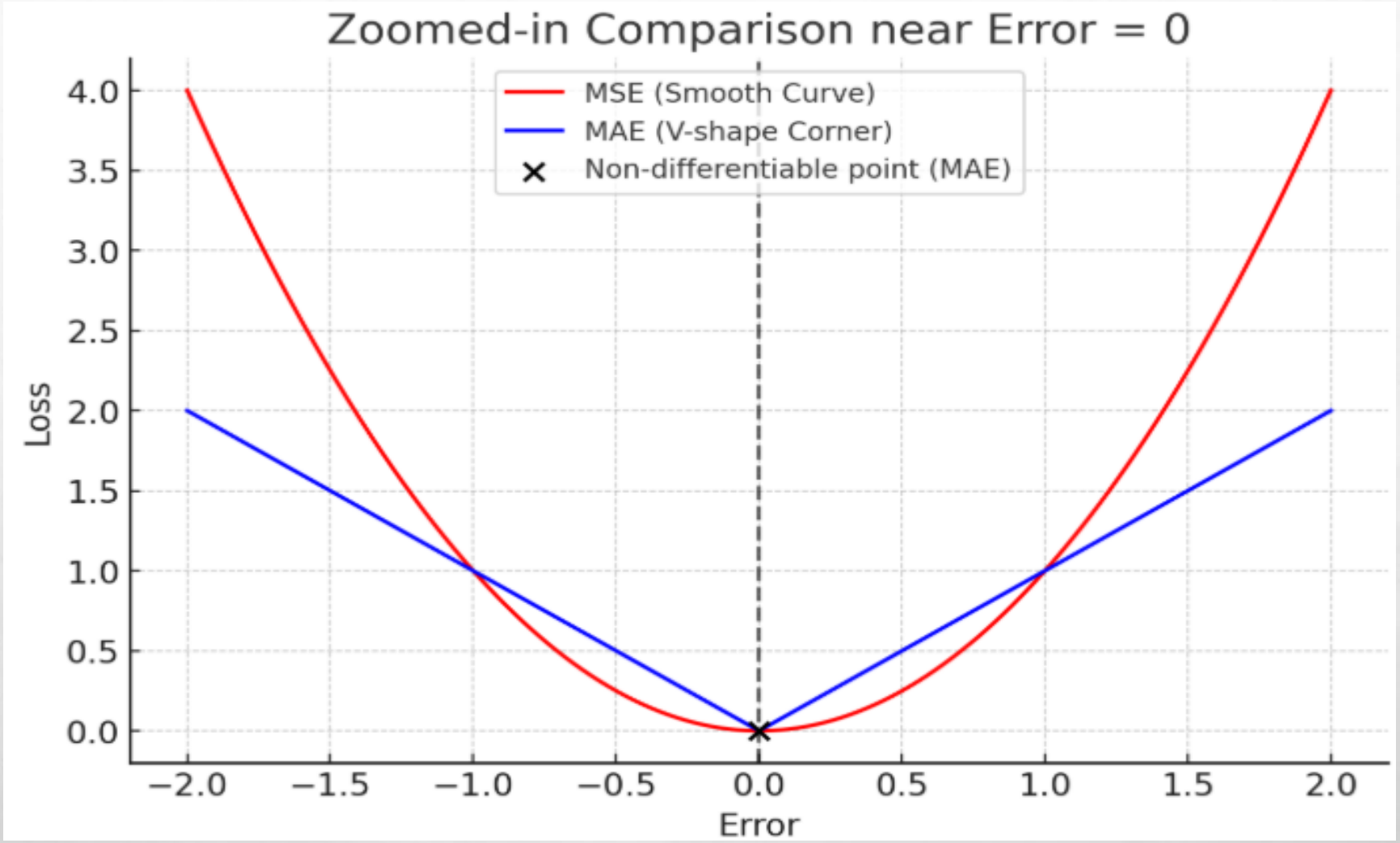
- MEAN SQUARED ERROR (MSE) LOSS IS ONE OF THE MOST WIDELY USED LOSS FUNCTIONS FOR REGRESSION TASKS. IT CALCULATES THE AVERAGE OF THE SQUARED DIFFERENCES BETWEEN THE PREDICTED VALUES AND THE ACTUAL VALUES. IT IS SIMPLE TO UNDERSTAND AND SENSITIVE TO OUTLIERS BECAUSE THE ERRORS ARE SQUARED WHICH CAN AFFECT THE LOSS.

- $\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2$

2. MEAN ABSOLUTE ERROR (MAE) LOSS

- MEAN ABSOLUTE ERROR (MAE) LOSS IS ANOTHER COMMONLY USED LOSS FUNCTION FOR REGRESSION. IT CALCULATES THE AVERAGE OF THE ABSOLUTE DIFFERENCES BETWEEN THE PREDICTED VALUES AND THE ACTUAL VALUES. IT IS LESS SENSITIVE TO OUTLIERS COMPARED TO MSE. BUT IT IS NOT DIFFERENTIABLE AT ZERO WHICH CAN CAUSE ISSUES FOR SOME OPTIMIZATION ALGORITHMS.
- MAE $= \frac{1}{n} \sum_{i=1}^{n} | y_i - \hat{y}_i |$

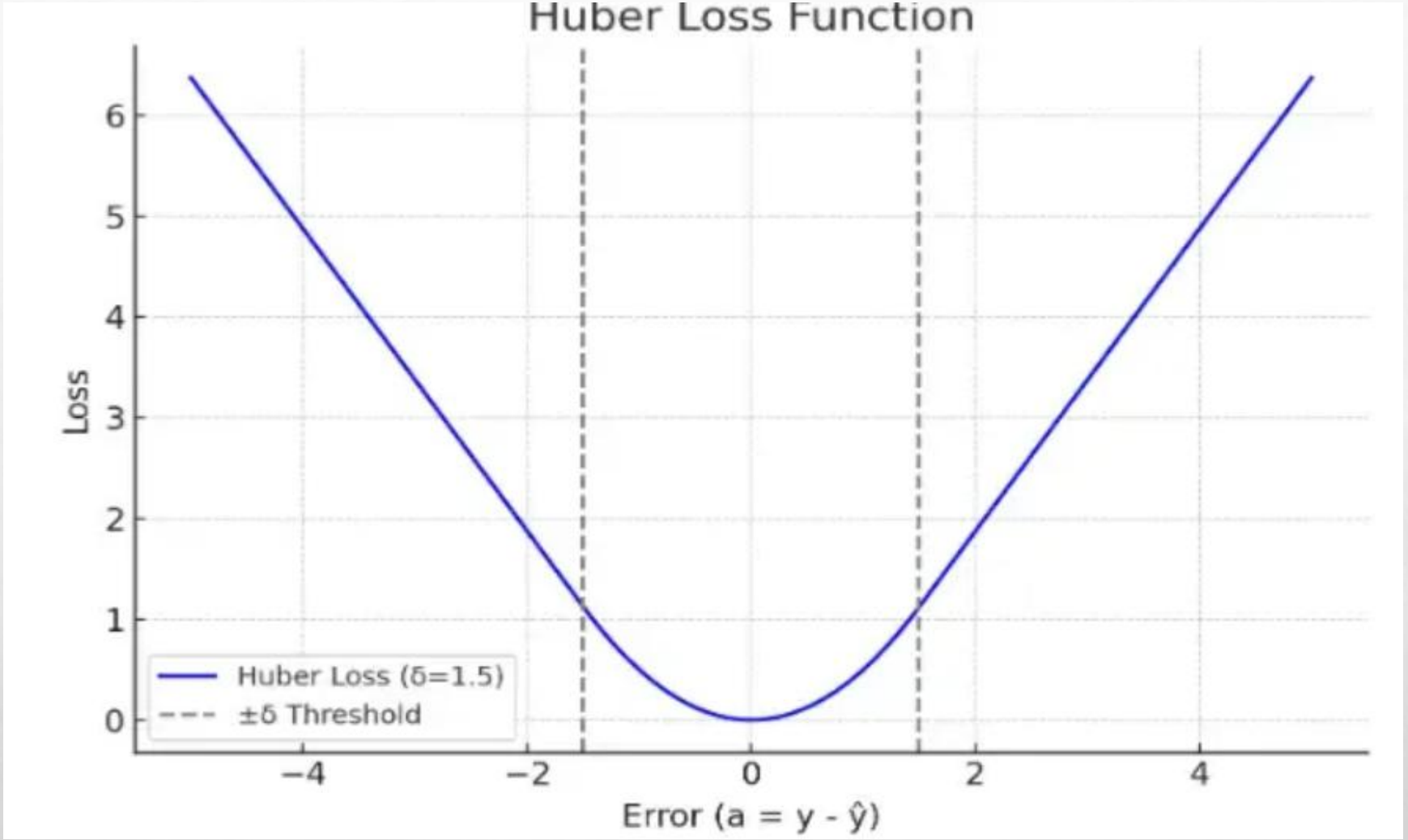

Comparison of MSE and MAE
MSE (Quadratic)
MAE (Linear)
Loss
100
80
60
40
20
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Error


Zoomed-in Comparison near Error = 0
MSE (Smooth Curve)
MAE (V-shape Corner)
Non-differentiable point (MAE)
Loss
Error
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

3. HUBER LOSS

- HUBER LOSS COMBINES THE ADVANTAGES OF MSE AND MAE. IT IS LESS SENSITIVE TO OUTLIERS THAN MSE AND DIFFERENTIABLE EVERYWHERE UNLIKE MAE. IT REQUIRES TUNING OF THE PARAMETER δ. HUBER LOSS IS DEFINED AS:

- $$\begin{cases} \frac{1}{2}\left(y_i - \hat{y}_i\right)^2 & \text{FOR } |\, y_i - \hat{y}_i \,| \leq \delta \\ \delta \,|\, y_i - \hat{y}_i \,| - \frac{1}{2}\delta^2 & \text{FOR } |\, y_i - \hat{y}_i \,| > \delta \end{cases}$$


Huber Loss Function
Loss
6
5
4
3
2
1
0
−4
−2
0
2
4
Error (a = y - ŷ)
Huber Loss (δ=1.5)
±δ Threshold

2. CLASSIFICATION LOSS FUNCTIONS

- CLASSIFICATION LOSS FUNCTIONS ARE USED TO EVALUATE HOW WELL A CLASSIFICATION MODEL'S PREDICTIONS MATCH THE ACTUAL CLASS LABELS. THERE ARE DIFFERENT TYPES OF CLASSIFICATION LOSS FUNCTIONS:

1. BINARY CROSS-ENTROPY LOSS (LOG LOSS)

- BINARY CROSS-ENTROPY LOSS IS ALSO KNOWN AS LOG LOSS AND IS USED FOR BINARY CLASSIFICATION PROBLEMS. IT MEASURES THE PERFORMANCE OF A CLASSIFICATION MODEL WHOSE OUTPUT IS A PROBABILITY VALUE BETWEEN 0 AND 1.

$$\text{Binary Cross-Entropy} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

where:

- n is the number of data points
- y_i is the actual binary label (0 or 1)
- $\hat{y}_i$ is the predicted probability.

HOW IT WORKS

- IF THE TRUE LABEL $y = 1$, THE LOSS IS:
- −LOG(Y^)
- → ENCOURAGES MODEL TO MAKE $\hat{y}$ CLOSE TO **1**.
- IF THE TRUE LABEL $y = 0$, THE LOSS IS:
- −LOG(1−Y^)
- → ENCOURAGES MODEL TO MAKE $\hat{y}$CLOSE TO **0**.

INTUITION

- **Logarithm penalty**: If the model predicts wrong with high confidence, the penalty is very large.
- Example: True = 1, Predicted = 0.01 → Loss = $-\log(0.01) \approx 4.6$
- **Confident & correct predictions** → very low loss.
- **Confident & wrong predictions** → very high loss.

2. CATEGORICAL CROSS-ENTROPY LOSS

- CATEGORICAL CROSS-ENTROPY LOSS IS USED FOR MULTICLASS CLASSIFICATION PROBLEMS. IT MEASURES THE PERFORMANCE OF A CLASSIFICATION MODEL WHOSE OUTPUT IS A PROBABILITY DISTRIBUTION OVER MULTIPLE CLASSES.

2. CATEGORICAL CROSS-ENTROPY LOSS

$$\text{Categorical Cross-Entropy} = -\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij})$$

where:

- n is the number of data points
- k is the number of classes,
- y_{ij} is the binary indicator (0 or 1) if class label j is the correct classification for data point i
- $\hat{y}_{ij}$ is the predicted probability for class j.

EXAMPLE

- SUPPOSE WE CLASSIFY IMAGES INTO 3 CLASSES: CAT, DOG, HORSE.
- TRUE = DOG → $y = [0, 1, 0]$
- MODEL PREDICTS → $\hat{y} = [0.1, 0.7, 0.2]$
- SINCE THE MODEL PREDICTED DOG WITH 70% CONFIDENCE → LOSS IS LOW.
- IF THE MODEL INSTEAD PREDICTED $[0.7, 0.2, 0.1]$,THEN:
- HIGHER LOSS, BECAUSE IT WAS LESS CONFIDENT ABOUT THE CORRECT CLASS.

3. SPARSE CATEGORICAL CROSS-ENTROPY LOSS

- SPARSE CATEGORICAL CROSS-ENTROPY LOSS IS SIMILAR TO CATEGORICAL CROSS-ENTROPY LOSS BUT IS USED WHEN THE TARGET LABELS ARE INTEGERS INSTEAD OF ONE-HOT ENCODED VECTORS. IT IS EFFICIENT FOR LARGE DATASETS WITH MANY CLASSES.

$$\text{Sparse Categorical Cross-Entropy} = -\sum_{i=1}^{n} \log(\hat{y}_{i,y_i})$$

- WHERE y_i IS THE INTEGER REPRESENTING THE CORRECT CLASS FOR DATA POINT I.

•In **Categorical Cross-Entropy (CCE)**, the true labels are **one-hot-encoded**.

Example for 3 classes:

•Cat → [1, 0, 0]

•Dog → [0, 1, 0]

•Horse → [0, 0, 1]

•In **Sparse Categorical Cross-Entropy (SCCE)**, the true labels are stored as **integer class indices**, not one-hot.
Example for 3 classes:

•Cat → 0

•Dog → 1

•Horse → 2

EXAMPLE

- SUPPOSE WE CLASSIFY INTO 3 CLASSES AGAIN: CAT, DOG, HORSE.
- TRUE LABEL = DOG → STORED AS 1 (INSTEAD OF [0,1,0]).
- PREDICTED PROBABILITIES → $\hat{y} = [0.1, 0.7, 0.2]$.
- ✓SAME RESULT AS CATEGORICAL CROSS-ENTROPY.

4. KULLBACK-LEIBLER DIVERGENCE LOSS (KL DIVERGENCE)

- KL DIVERGENCE MEASURES HOW ONE PROBABILITY DISTRIBUTION DIVERGES FROM A SECOND EXPECTED PROBABILITY DISTRIBUTION. IT IS OFTEN USED IN PROBABILISTIC MODELS. IT IS SENSITIVE TO SMALL DIFFERENCES IN PROBABILITY DISTRIBUTIONS.

$$\text{KL Divergence} = \sum_{i=1}^{n} \sum_{j=1}^{k} y_{ij} \log \left(\frac{y_{ij}}{\hat{y}_{ij}} \right)$$

EXAMPLE

- TRUE DISTRIBUTION (P):
- CAT = 0.7, DOG = 0.2, HORSE = 0.1
- PREDICTED DISTRIBUTION (Q):
- CAT = 0.6, DOG = 0.3, HORSE = 0.1
- KL DIVERGENCE WILL BE **SMALL** BECAUSE THE DISTRIBUTIONS ARE CLOSE.
- BUT IF THE MODEL PREDICTED Q = [0.1, 0.8, 0.1], KL DIVERGENCE WILL BE **LARGE**, SINCE THE MODEL IS “CONFIDENTLY WRONG.”

5. HINGE LOSS

- HINGE LOSS IS USED FOR TRAINING CLASSIFIERS ESPECIALLY FOR SUPPORT VECTOR MACHINES (SVMS). IT IS SUITABLE FOR BINARY CLASSIFICATION TASKS AS IT IS NOT DIFFERENTIABLE AT ZERO.

$$\text{Hinge Loss} = \frac{1}{n}\sum_{i=1}^{n} \max(0, 1 - y_i \cdot \hat{y}_i)$$

where:

- y_i is the actual label (-1 or 1)
- $\hat{y}_i$ is the predicted value.

INTUITION

- IF THE MODEL PREDICTS CORRECTLY **AND WITH ENOUGH MARGIN** ($y \cdot f(x) \geq 1$,(LOSS = 0)
- IF THE MODEL IS CORRECT BUT **TOO CLOSE TO THE BOUNDARY** ($0 < y \cdot f(x) < 1$,(THERE IS A **POSITIVE LOSS** → ENCOURAGES MODEL TO INCREASE CONFIDENCE.
- IF THE MODEL IS **WRONG** ($y \cdot f(x) \leq 0$,(LOSS IS LARGE → STRONG PENALTY.
- THIS ENCOURAGES THE CLASSIFIER TO NOT JUST BE CORRECT, BUT TO BE **CONFIDENTLY CORRECT** WITH A MARGIN.

EXAMPLE

- SUPPOSE TRUE LABEL = +1
- PREDICTION SCORE $f(x) = 2$: L=MAX(0,1−(1∗2))=0 → CORRECT & CONFIDENT
- PREDICTION SCORE $f(x) = 0.5$: L=MAX(0,1−(1∗0.5))=0.5 → :CORRECT BUT NOT CONFIDENT ENOUGH
- PREDICTION SCORE $f(x) = -0.5$: L=MAX(0,1−(1∗−0.5))=1.5 → :WRONG, BIG PENALTY

WHERE IT'S USED

- **Support Vector Machines (SVMs)** for maximum-margin classification.
- Some neural networks for binary classification tasks.

3. RANKING LOSS FUNCTIONS

- RANKING LOSS FUNCTIONS ARE USED TO EVALUATE MODELS THAT PREDICT THE RELATIVE ORDER OF ITEMS. THESE ARE COMMONLY USED IN TASKS SUCH AS RECOMMENDATION SYSTEMS AND INFORMATION RETRIEVAL.

1. CONTRASTIVE LOSS

- CONTRASTIVE LOSS IS USED TO LEARN EMBEDDINGS SUCH THAT SIMILAR ITEMS ARE CLOSER IN THE EMBEDDING SPACE WHILE DISSIMILAR ITEMS ARE FARTHER APART. IT IS OFTEN USED IN SIAMESE NETWORKS.

$$\text{Contrastive Loss} = \frac{1}{2N} \sum_{i=1}^{N} \left(y_i \cdot d_i^2 + (1 - y_i) \cdot \max(0, m - d_i)^2 \right)$$

where:

- d_i is the distance between a pair of embeddings
- y_i is 1 for similar pairs and 0 for dissimilar pairs
- m is a margin.

INTUITION

- If **similar pair** (y=1): Loss = D^2 → encourages network to make embeddings close together.
- If **dissimilar pair** (y=0): Loss = max(0, m - D)^2→ no loss if they're farther apart than margin, but penalty if too close.

EXAMPLE

- SUPPOSE MARGIN $m = 2$:
- PAIR IS SIMILAR, DISTANCE $D = 0.5 \rightarrow$ LOSS = 0.25(SMALL, SINCE THEY'RE ALREADY CLOSE)
- PAIR IS DISSIMILAR, DISTANCE $D = 0.5 \rightarrow$ LOSS = $(2 - 0.5)^2 = 2.25$)BIG PENALTY, THEY MUST BE PUSHED APART)
- PAIR IS DISSIMILAR, DISTANCE $D = 3 \rightarrow$ LOSS = 0 (THEY'RE ALREADY FAR ENOUGH)

2. TRIPLET LOSS

- TRIPLET LOSS IS USED TO LEARN EMBEDDINGS BY COMPARING THE RELATIVE DISTANCES BETWEEN TRIPLETS: ANCHOR, POSITIVE EXAMPLE AND NEGATIVE EXAMPLE.

Triplet Loss =

$$\frac{1}{N}\sum_{i=1}^{N}\left[\|f(x_i^a)-f(x_i^p)\|_2^2-\|f(x_i^a)-f(x_i^n)\|_2^2+\alpha\right]_+$$

where:

- f(x) is the embedding function
- x_i^a is the anchor
- x_i^p is the positive example
- x_i^n is the negative example
- α is a margin.

INTUITION

- $D(A, P)$ =DISTANCE BETWEEN **ANCHOR & POSITIVE** → SHOULD BE **SMALL**
- $D(A, N)$ =DISTANCE BETWEEN **ANCHOR & NEGATIVE** → SHOULD BE **LARGE**
- LOSS TRIES TO ENFORCE:
- $D(A, P) + m < D(A, N)$
- SO THE ANCHOR IS CLOSER TO POSITIVE THAN NEGATIVE BY AT LEAST MARGIN m.

EXAMPLE

- LET'S SAY MARGIN $m = 0.5$:
- $D(A, P) = 0.8$,$D(A, N) = 1.0$ →LOSS = MAX(0,0.8−1.0+0.5)=0.3 (NOT ENOUGH MARGIN, PENALIZED)
- $D(A, P) = 0.8$,$D(A, N) = 2.0$ →LOSS = MAX(0,0.8−2.0+0.5)= 0 (GOOD SEPARATION, NO PENALTY)

WHERE IT'S USED?

- **FaceNet** (Google) for face recognition
- **Speaker verification** (voice embeddings)
- **Recommender systems** (learning user-item embeddings)

3. MARGIN RANKING LOSS

- MARGIN RANKING LOSS MEASURES THE RELATIVE DISTANCES BETWEEN PAIRS OF ITEMS AND ENSURES THAT THE CORRECT ORDERING IS MAINTAINED WITH A SPECIFIED MARGIN.

$$\text{Margin Ranking Loss} = \frac{1}{N}\sum_{i=1}^{N}\max(0, -y_i \cdot (s_i^+ - s_i^-) + \text{margin})$$

where:

- s_i^+ and s_i^- are the scores for the positive and negative samples
- y_i is the label indicating the correct ordering.

INTUITION

- If the correct ordering already exists **with enough margin**, loss = 0 ✅
- If the wrong ordering (or not enough separation), loss > 0 → pushes the model to increase the gap.

EXAMPLE

- LET'S SAY MARGIN $m = 1$:
- CASE 1: $y = +1, x_1 = 3, x_2 = 1$, L=MAX(0,−(1·(3−1))+1)=MAX(0,−2+1)=0
  →CORRECT & CONFIDENT.
- CASE 2: $y = +1, x_1 = 2, x_2 = 2$, L=MAX(0,−(1·(0))+1)=1
  →NO MARGIN, PENALIZED.
- CASE 3: $y = -1, x_1 = 4, x_2 = 5$, L=MAX(0,−(−1·(4−5))+1)=MAX(0,−(1)+1)=0
  →CORRECT ORDERING.

WHERE IT'S USED?

- **Information retrieval** (ranking search results).
- **Recommendation systems** (ranking items).
- **Learning-to-rank tasks** like RankNet, LambdaRank, etc.

4. IMAGE AND RECONSTRUCTION LOSS FUNCTIONS

- THESE LOSS FUNCTIONS ARE USED TO EVALUATE MODELS THAT GENERATE OR RECONSTRUCT IMAGES ENSURING THAT THE OUTPUT IS AS CLOSE AS POSSIBLE TO THE TARGET IMAGES

1. PIXEL-WISE CROSS-ENTROPY LOSS

- PIXEL-WISE CROSS-ENTROPY LOSS IS USED FOR IMAGE SEGMENTATION TASKS WHERE EACH PIXEL IS CLASSIFIED INDEPENDENTLY.

$$\text{Pixel-wise Cross-Entropy} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

where:

- N is the number of pixels,
- C is the number of classes
- $y_{i,c}$ is the binary indicator for the correct class of pixel
- $\hat{y}_{i,c}$ is the predicted probability for class c.

INTUITION

- Each pixel is treated like a tiny classification problem.
- If the model assigns **high probability** to the correct class → **low loss**.
- If the model assigns **low probability** to the correct class → **high loss**.

EXAMPLE

- SUPPOSE 1 PIXEL HAS 3 POSSIBLE CLASSES:
- TRUE LABEL = CLASS 2
- PREDICTED PROBABILITIES = [0.1, 0.7, 0.2]
- LOSS = −LOG(0.7)≈0.357 (LOW, BECAUSE PREDICTION WAS CONFIDENT AND CORRECT).
- IF PREDICTED PROBABILITIES = [0.8, 0.1, 0.1], LOSS = (HIGH, BECAUSE MODEL WAS WRONG AND CONFIDENT).

OPTIMIZATION ALGORITHMS

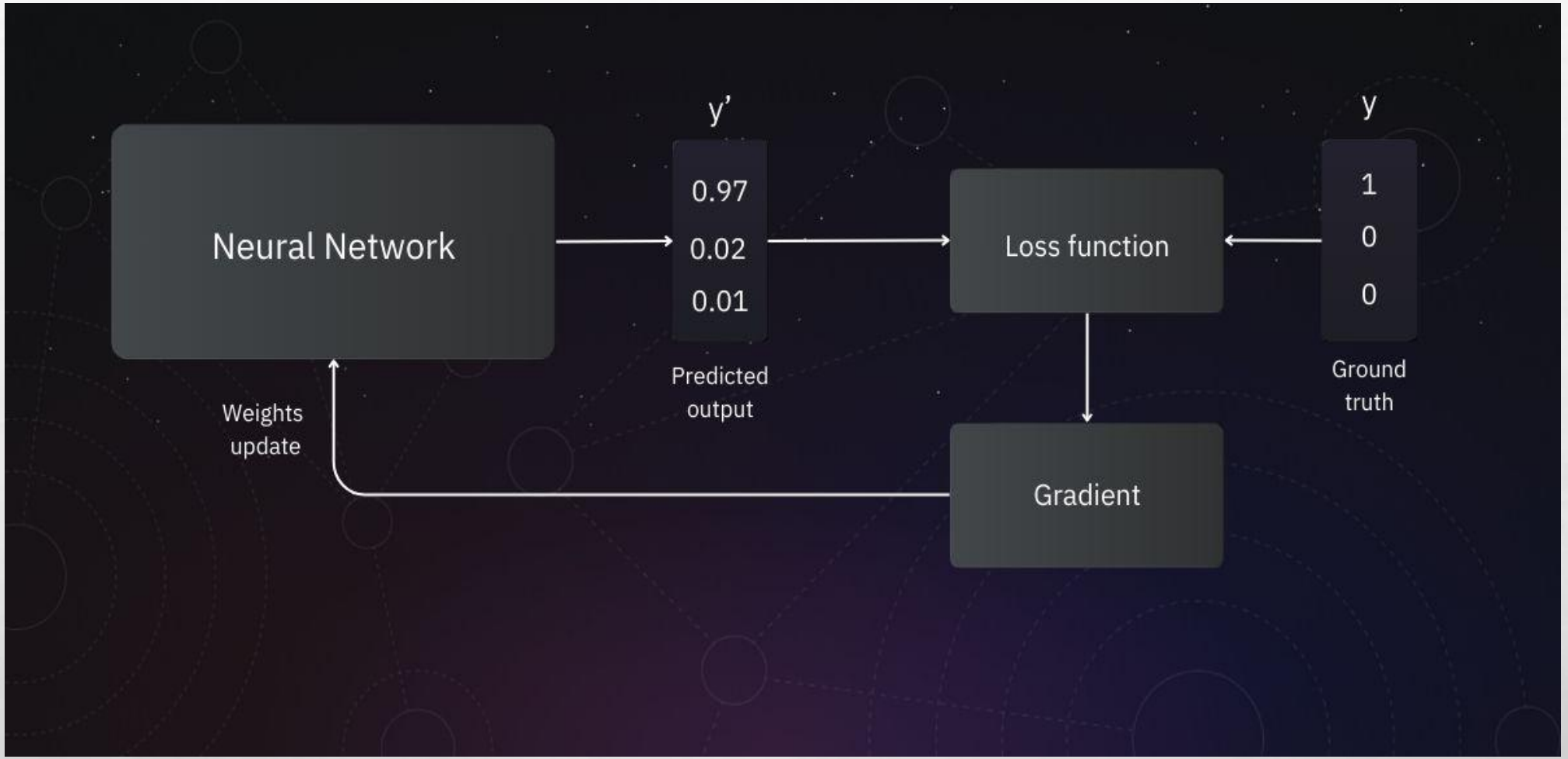
- THE OPTIMIZERS USED FOR TRAINING DEEP LEARNING MODELS ARE BASED ON GRADIENT DESCENT, TRYING TO SHIFT THE MODEL'S WEIGHTS TOWARDS THE OBJECTIVE FUNCTION'S MINIMUM.
- A RANGE OF OPTIMIZATION ALGORITHMS IS USED TO TRAIN DEEP LEARNING MODELS, EACH AIMING TO ADDRESS A PARTICULAR SHORTCOMING OF THE BASIC GRADIENT DESCENT APPROACH.
- STOCHASTIC GRADIENT DESCENT (SGD) AND MINI-BATCH GRADIENT DESCENT SPEED UP TRAINING AND ARE SUITABLE FOR LARGER DATASETS.
- ADAGRAD ADAPTS LEARNING RATES TO PARAMETERS BUT MAY SLOW DOWN LEARNING OVER TIME. RMSPROP AND ADADELTA BUILD ON ADAGRAD'S APPROACH, ADDRESSING ITS DIMINISHING LEARNING RATES, WITH ADADELTA REMOVING THE NEED FOR A SET LEARNING RATE.
- ADAM COMBINES THE ADVANTAGES OF ADAGRAD AND RMSPROP AND IS EFFECTIVE ACROSS A WIDE RANGE OF DEEP-LEARNING TASKS.

OPTIMIZATION ALGORITHMS

- OPTIMIZATION ALGORITHMS PLAY A CRUCIAL ROLE IN TRAINING DEEP LEARNING MODELS. THEY CONTROL HOW A NEURAL NETWORK IS INCREMENTALLY CHANGED TO MODEL THE COMPLEX RELATIONSHIPS ENCODED IN THE TRAINING DATA.

WHAT IS A MODEL-OPTIMIZATION ALGORITHM?

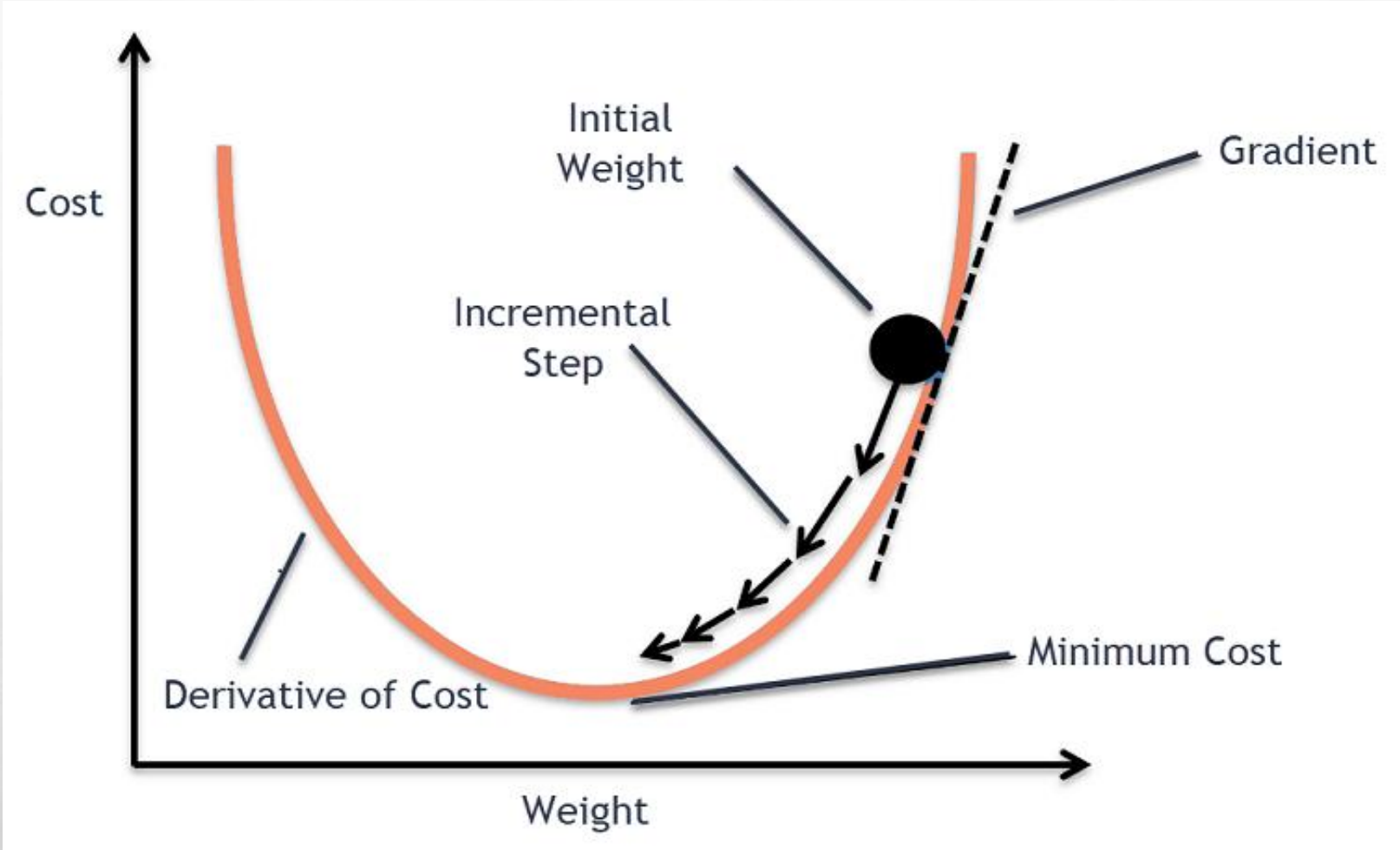
- A DEEP LEARNING MODEL COMPRISES MULTIPLE LAYERS OF INTERCONNECTED NEURONS ORGANIZED INTO LAYERS. EACH NEURON COMPUTES AN ACTIVATION FUNCTION ON THE INCOMING DATA AND PASSES THE RESULT TO THE NEXT LAYER. THE ACTIVATION FUNCTIONS INTRODUCE NON-LINEARITY, ALLOWING FOR COMPLEX MAPPINGS BETWEEN INPUTS AND OUTPUTS.
- THE CONNECTION STRENGTH BETWEEN NEURONS AND THEIR ACTIVATIONS ARE PARAMETRIZED BY WEIGHTS AND BIASES. THESE PARAMETERS ARE ITERATIVELY ADJUSTED DURING TRAINING TO MINIMIZE THE DISCREPANCY BETWEEN THE MODEL'S OUTPUT AND THE DESIRED OUTPUT GIVEN BY THE TRAINING DATA. THE DISCREPANCY IS QUANTIFIED BY A LOSS FUNCTION.
-


Neural Network
y'
0.97
0.02
0.01
Predicted output
Loss function
y
1
0
0
Ground truth
Weights update
Gradient

- THIS ADJUSTMENT IS GOVERNED BY AN OPTIMIZATION ALGORITHM. OPTIMIZERS UTILIZE GRADIENTS COMPUTED BY BACKPROPAGATION TO DETERMINE THE DIRECTION AND MAGNITUDE OF PARAMETER UPDATES, AIMING TO NAVIGATE THE MODEL'S HIGH-DIMENSIONAL PARAMETER SPACE EFFICIENTLY. OPTIMIZERS EMPLOY VARIOUS STRATEGIES TO BALANCE EXPLORATION AND EXPLOITATION, SEEKING TO ESCAPE LOCAL MINIMA AND CONVERGE TO OPTIMAL OR NEAR-OPTIMAL SOLUTIONS.

WHAT IS GRADIENT DESCENT?

- GRADIENT DESCENT IS AN ALGORITHM DESIGNED TO MINIMIZE A FUNCTION BY ITERATIVELY MOVING TOWARDS THE MINIMUM VALUE OF THE FUNCTION. IT'S AKIN TO A HIKER TRYING TO FIND THE LOWEST POINT IN A VALLEY SHROUDED IN FOG. THE HIKER STARTS AT A RANDOM LOCATION AND CAN ONLY FEEL THE SLOPE OF THE GROUND BENEATH THEIR FEET. TO REACH THE VALLEY'S LOWEST POINT, THE HIKER TAKES STEPS IN THE DIRECTION OF THE STEEPEST DESCENT.


Cost
Initial
Weight
Gradient
Incremental
Step
Minimum Cost
Derivative of Cost
Weight

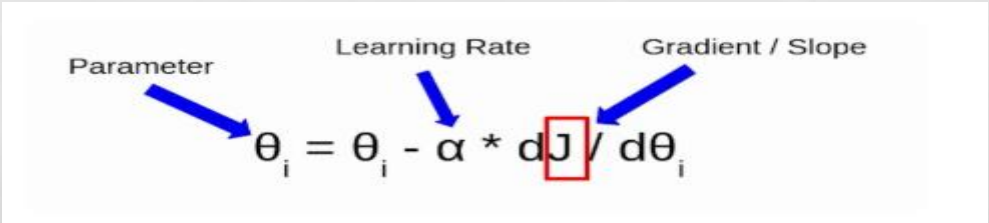
- ALL DEEP LEARNING MODEL OPTIMIZATION ALGORITHMS WIDELY USED TODAY ARE BASED ON GRADIENT DESCENT. HENCE, HAVING A GOOD GRASP OF THE TECHNICAL AND MATHEMATICAL DETAILS IS ESSENTIAL.
- **OBJECTIVE**: GRADIENT DESCENT AIMS TO FIND A FUNCTION'S PARAMETERS (WEIGHTS) THAT MINIMIZE THE COST FUNCTION. IN THE CASE OF A DEEP LEARNING MODEL, THE COST FUNCTION IS THE AVERAGE OF THE LOSS FOR ALL TRAINING SAMPLES AS GIVEN BY THE LOSS FUNCTION. WHILE THE LOSS FUNCTION IS A FUNCTION OF THE MODEL'S OUTPUT AND THE GROUND TRUTH, THE COST FUNCTION IS A FUNCTION OF THE MODEL'S WEIGHTS AND BIASES.

HOW IT WORKS

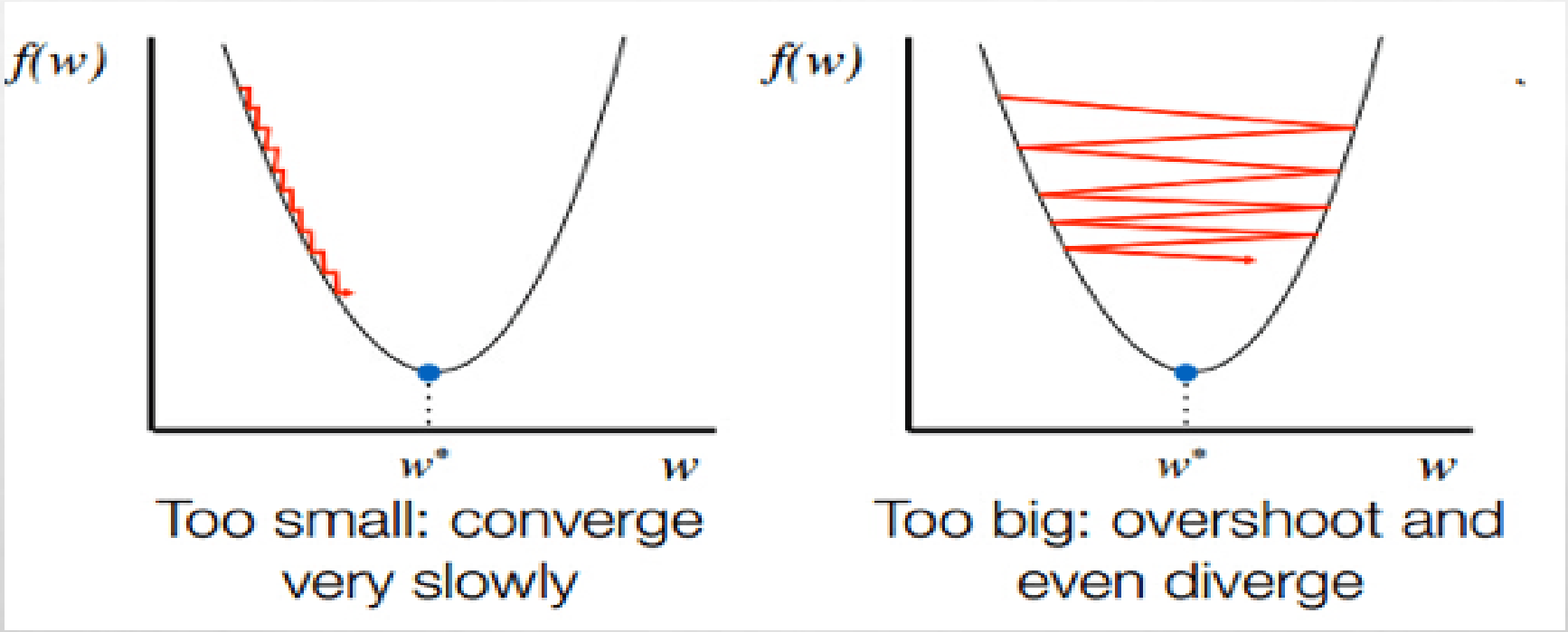
- **INITIALIZATION**: START WITH RANDOM VALUES FOR THE MODEL'S WEIGHTS.
- **GRADIENT COMPUTATION**: CALCULATE THE GRADIENT OF THE COST FUNCTION WITH RESPECT TO EACH PARAMETER. THE GRADIENT IS A VECTOR THAT POINTS IN THE DIRECTION OF THE STEEPEST INCREASE OF THE FUNCTION. IN THE CONTEXT OF OPTIMIZATION, WE'RE INTERESTED IN THE NEGATIVE GRADIENT, WHICH POINTS TOWARDS THE DIRECTION OF THE STEEPEST DECREASE.
- **UPDATE PARAMETERS**: ADJUST THE MODEL'S PARAMETERS IN THE DIRECTION OPPOSITE TO THE GRADIENT. THIS STEP IS DONE BY SUBTRACTING A FRACTION OF THE GRADIENT FROM THE CURRENT VALUES OF THE PARAMETERS. THE SIZE OF THIS STEP IS DETERMINED BY THE LEARNING RATE, A HYPERPARAMETER THAT CONTROLS HOW FAST OR SLOW WE MOVE TOWARD THE OPTIMAL WEIGHTS.

MATHEMATICAL REPRESENTATION:

- THE UPDATE RULE FOR EACH PARAMETER w CAN BE MATHEMATICALLY REPRESENTED AS
- **NEW VALUE = OLD VALUE – STEP SIZE**
- **HERE THETA ARE NOTHING BUT THE WEIGHTS ONLY WHICH IS THE STANDARD FORMULA OF GRADIENT DESCENT , SO DON'T GET CONFUSED WITH THE NOTATIONS.**



- THE LEARNING RATE IS A CRUCIAL HYPERPARAMETER THAT NEEDS TO BE CHOSEN CAREFULLY. IF IT'S TOO SMALL, THE ALGORITHM WILL CONVERGE VERY SLOWLY. IF IT'S TOO LARGE, THE ALGORITHM MIGHT OVERSHOOT THE MINIMUM AND FAIL TO CONVERGE.



CHALLENGES:

- **LOCAL MINIMA AND SADDLE POINTS**: IN COMPLEX COST LANDSCAPES, GRADIENT DESCENT CAN GET STUCK IN LOCAL MINIMA OR SADDLE POINTS, ESPECIALLY IN NON-CONVEX OPTIMIZATION PROBLEMS COMMON IN DEEP LEARNING.

- **CHOOSING THE RIGHT LEARNING RATE**: FINDING AN OPTIMAL LEARNING RATE REQUIRES EXPERIMENTATION AND TUNING.

STOCHASTIC GRADIENT DESCENT (SGD)

- STOCHASTIC GRADIENT DESCENT (SGD) IS A VARIANT OF THE TRADITIONAL GRADIENT DESCENT OPTIMIZATION ALGORITHM THAT INTRODUCES RANDOMNESS INTO THE OPTIMIZATION PROCESS TO IMPROVE CONVERGENCE SPEED AND POTENTIALLY ESCAPE LOCAL MINIMA. TO UNDERSTAND THE INTUITION BEHIND SGD, WE CAN AGAIN INVOKE THE ANALOGY OF A HIKER DESCENDING A FOGGY VALLEY. IF GRADIENT DESCENT REPRESENTS A CAUTIOUS HIKER WHO CAREFULLY EVALUATES THE SLOPE AROUND THEM BEFORE TAKING A STEP, STOCHASTIC GRADIENT DESCENT IS AKIN TO A MORE IMPULSIVE HIKER WHO DECIDES THEIR NEXT STEP BASED ONLY ON THE SLOPE OF THE GROUND IMMEDIATELY BENEATH THEIR FEET.

OBJECTIVE:

- LIKE GRADIENT DESCENT, THE PRIMARY GOAL OF SGD IS TO MINIMIZE THE COST FUNCTION OF A MODEL BY ITERATIVELY ADJUSTING ITS PARAMETERS (WEIGHTS). HOWEVER, SGD AIMS TO ACHIEVE THIS GOAL MORE EFFICIENTLY BY USING ONLY A SINGLE TRAINING EXAMPLE AT A TIME TO INFORM THE UPDATE OF THE MODEL'S PARAMETERS.
-

HOW IT WORKS:

- **INITIALIZATION**: START WITH A RANDOM SET OF PARAMETERS FOR THE MODEL.
- **GRADIENT COMPUTATION**: INSTEAD OF CALCULATING THE GRADIENT OF THE COST FUNCTION OVER THE ENTIRE TRAINING DATA, SGD COMPUTES THE GRADIENT BASED ON A SINGLE RANDOMLY SELECTED TRAINING EXAMPLE.
- **UPDATE PARAMETERS**: UPDATE THE MODEL'S PARAMETERS USING THIS COMPUTED GRADIENT. THE PARAMETERS ARE ADJUSTED IN THE DIRECTION OPPOSITE TO THE GRADIENT, SIMILAR TO BASIC GRADIENT DESCENT.

MATHEMATICAL REPRESENTATION:

$$w = w - \alpha \left[\frac{\partial Loss}{\partial w} \right]$$

Here, *w* represents the model's parameters (weights), α is the learning rate, and del loss/ delw is the gradient of the cost function

CHALLENGES:

- **VARIANCE**: THE UPDATES CAN BE NOISY DUE TO THE RELIANCE ON A SINGLE EXAMPLE, POTENTIALLY CAUSING THE COST FUNCTION TO FLUCTUATE. AS A RESULT, THE ALGORITHM DOES NOT CONVERGE TO A MINIMUM BUT JUMPS AROUND THE COST LANDSCAPE.

- **HYPERPARAMETER TUNING**: CORRECTLY SETTING THE LEARNING RATE REQUIRES EXPERIMENTATION.

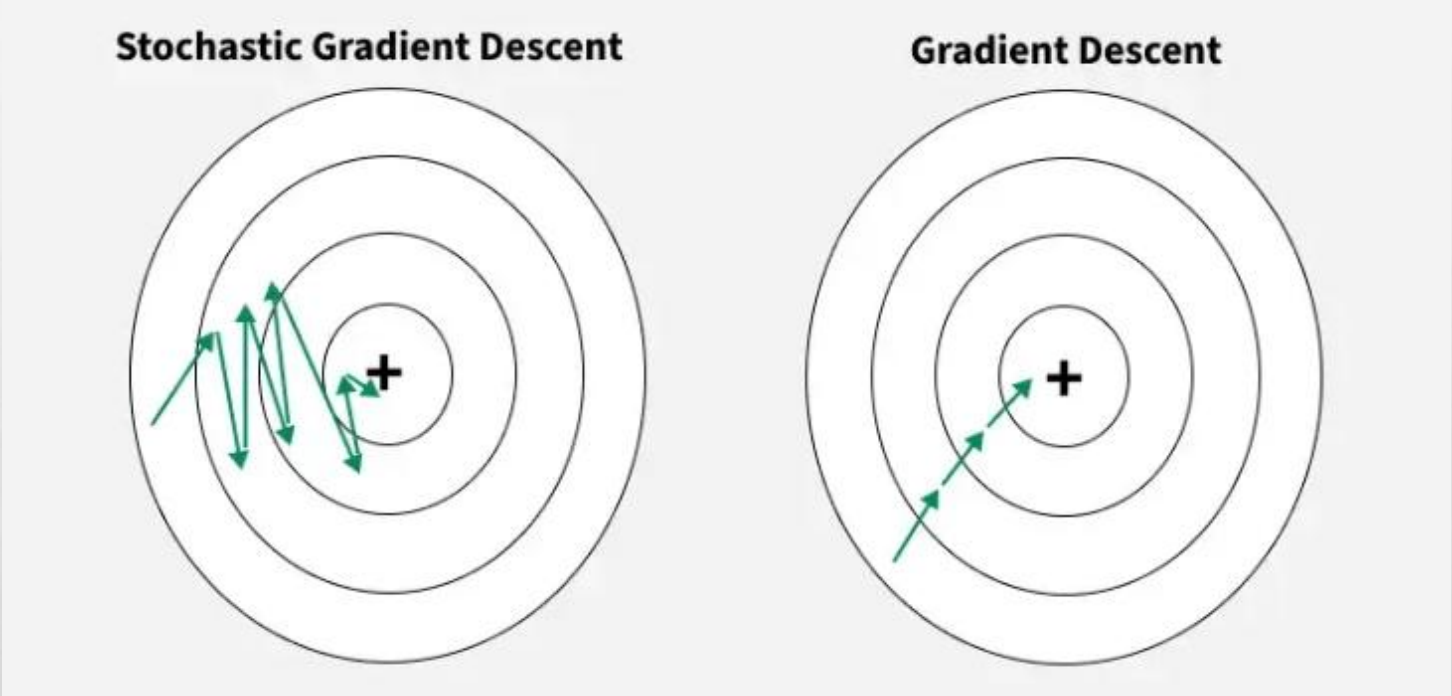
ADVANTAGES

- **EFFICIENCY**: USING ONLY ONE EXAMPLE AT A TIME, SGD SIGNIFICANTLY REDUCES THE COMPUTATIONAL REQUIREMENTS, MAKING IT FASTER AND MORE SCALABLE THAN GRADIENT DESCENT.

- **ESCAPE LOCAL MINIMA**: THE INHERENT NOISE IN SGD CAN HELP THE ALGORITHM ESCAPE SHALLOW LOCAL MINIMA, POTENTIALLY LEADING TO BETTER SOLUTIONS IN COMPLEX COST LANDSCAPES.

- **ONLINE LEARNING**: SGD IS WELL-SUITED FOR ONLINE LEARNING SCENARIOS WHERE THE MODEL NEEDS TO UPDATE CONTINUOUSLY AS NEW DATA ARRIVES.

STOCHASTIC GRADIENT DESCENT VS GRADIENT DESCENT



MINI-BATCH GRADIENT DESCENT

- MINI-BATCH GRADIENT DESCENT STRIKES A BALANCE BETWEEN THE THOROUGH, CALCULATED APPROACH OF GRADIENT DESCENT AND THE UNPREDICTABLE, SWIFT NATURE OF STOCHASTIC GRADIENT DESCENT (SGD). IMAGINE A GROUP OF HIKERS NAVIGATING THROUGH A FOGGY VALLEY. EACH HIKER INDEPENDENTLY ASSESSES A SMALL, DISTINCT SECTION OF THE SURROUNDING AREA BEFORE THE GROUP DECIDES ON THE BEST DIRECTION TO TAKE.

- BASED ON A BROADER BUT STILL LIMITED VIEW OF THE TERRAIN, THIS COLLECTIVE DECISION-MAKING PROCESS ALLOWS FOR A MORE INFORMED AND STEADY PROGRESSION TOWARD THE VALLEY'S LOWEST POINT COMPARED TO AN INDIVIDUAL HIKER'S ERRATIC JOURNEY

OBJECTIVE

- SIMILAR TO OTHER GRADIENT DESCENT VARIANTS, THE AIM OF MINI-BATCH GRADIENT DESCENT IS TO OPTIMIZE THE MODEL'S PARAMETERS TO MINIMIZE THE COST FUNCTION. IT SEEKS TO COMBINE THE EFFICIENCY OF SGD WITH THE STABILITY OF GRADIENT DESCENT BY USING A SUBSET OF THE TRAINING DATA TO COMPUTE GRADIENTS AND UPDATE PARAMETERS.

HOW IT WORKS

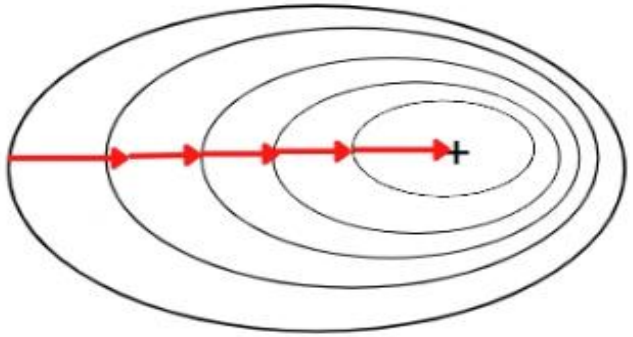
- **INITIALIZATION**: START WITH INITIAL RANDOM VALUES FOR THE MODEL'S PARAMETERS.
- **GRADIENT COMPUTATION**: INSTEAD OF CALCULATING THE GRADIENT USING THE ENTIRE DATASET (AS IN GRADIENT DESCENT) OR A SINGLE EXAMPLE (AS IN SGD), MINI-BATCH GRADIENT DESCENT COMPUTES THE GRADIENT USING A SMALL SUBSET OF THE TRAINING DATA, KNOWN AS A MINI-BATCH.
- **UPDATE PARAMETERS**: ADJUST THE PARAMETERS IN THE DIRECTION OPPOSITE TO THE COMPUTED GRADIENT. THIS ADJUSTMENT IS MADE BASED ON THE GRADIENT DERIVED FROM THE MINI-BATCH, AIMING TO REDUCE THE COST FUNCTION.

MATHEMATICAL REPRESENTATION:

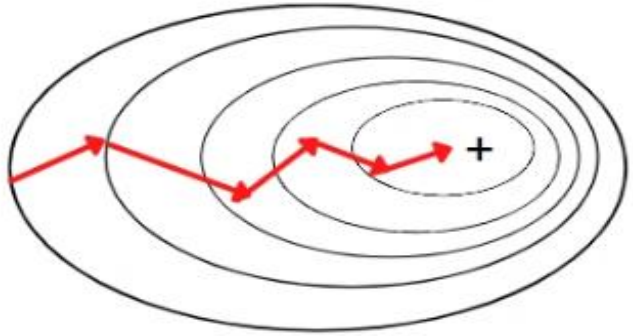
- REFER TO THE BELOW LINK FOR CLEAR EXPLANATION

https://www.analyticsvidhya.com/blog/2021/03/variants-of-gradient-descent-algorithm/

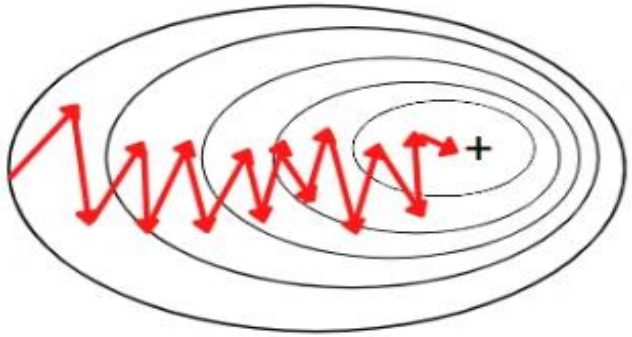
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



CHALLENGES

- **HYPERPARAMETER TUNING:** LIKE WITH THE OTHER VARIANTS WE'VE DISCUSSED SO FAR, SELECTING THE LEARNING RATE REQUIRES EXPERIMENTATION. FURTHER, WE NEED TO CHOOSE THE BATCH SIZE. IF THE BATCH SIZE IS TOO SMALL, WE FACE THE DRAWBACKS OF SGD, AND IF THE BATCH SIZE IS TOO LARGE, WE'RE PRONE TO THE ISSUES OF BASIC GRADIENT DESCENT.

ADVANTAGES:

- **EFFICIENCY AND STABILITY**: MINI-BATCH GRADIENT DESCENT OFFERS A COMPROMISE BETWEEN THE COMPUTATIONAL EFFICIENCY OF SGD AND THE STABILITY OF GRADIENT DESCENT.

- **PARALLELIZATION**: SINCE MINI-BATCHES ONLY CONTAIN A SMALL, FIXED NUMBER OF SAMPLES, THEY CAN BE COMPUTED IN PARALLEL, SPEEDING UP THE TRAINING PROCESS.

- **GENERALIZATION**: BY NOT USING THE ENTIRE DATASET FOR EACH UPDATE, MINI-BATCH GRADIENT DESCENT CAN HELP PREVENT OVERFITTING, LEADING TO MODELS THAT GENERALIZE BETTER ON UNSEEN DATA.

ADAGRAD (ADAPTIVE GRADIENT ALGORITHM)

- ADAGRAD (ADAPTIVE GRADIENT ALGORITHM) INTRODUCES AN INNOVATIVE TWIST TO THE CONVENTIONAL GRADIENT DESCENT OPTIMIZATION TECHNIQUE BY DYNAMICALLY ADAPTING THE LEARNING RATE, ALLOWING FOR A MORE NUANCED AND EFFECTIVE OPTIMIZATION PROCESS. IMAGINE A SCENARIO WHERE OUR GROUP OF HIKERS, NAVIGATING THE FOGGY VALLEY, NOW HAS ACCESS TO A MAP HIGHLIGHTING AREAS OF VARYING DIFFICULTY. WITH THIS MAP, THEY CAN ADJUST THEIR PACE — TAKING SMALLER STEPS IN STEEP, DIFFICULT TERRAIN AND LARGER STRIDES IN FLATTER REGIONS — TO OPTIMIZE THEIR PATH TOWARD THE VALLEY'S BOTTOM.

OBJECTIVE:

- ADAGRAD AIMS TO FINE-TUNE THE MODEL'S PARAMETERS TO MINIMIZE THE COST FUNCTION, SIMILAR TO GRADIENT DESCENT. ITS DISTINCTIVE FEATURE IS INDIVIDUALLY ADJUSTING LEARNING RATES FOR EACH PARAMETER BASED ON THE HISTORICAL GRADIENT INFORMATION FOR THOSE PARAMETERS. THIS LEADS TO MORE AGGRESSIVE LEARNING RATE ADJUSTMENTS FOR WEIGHTS TIED TO RARE BUT IMPORTANT FEATURES, ENSURING THESE PARAMETERS ARE OPTIMIZED ADEQUATELY WHEN THEIR RESPECTIVE FEATURES PLAY A ROLE IN PREDICTIONS.

- ASSIGNING HIGHER LEARNING RATES TO LESS FREQUENT FEATURES AND LOWER RATES TO MORE COMMON ONES, ADAGRAD EXCELS IN HANDLING SPARSE DATA. MOREOVER, IT ELIMINATES THE NEED FOR MANUAL LEARNING RATE ADJUSTMENTS, SIMPLIFYING THE TRAINING PROCESS.

HOW IT WORKS

- **INITIALIZATION**: BEGIN WITH RANDOM VALUES FOR THE MODEL'S PARAMETERS AND INITIALIZE A GRADIENT ACCUMULATION VARIABLE, TYPICALLY A VECTOR OF ZEROS, OF THE SAME SIZE AS THE PARAMETERS.

- **GRADIENT COMPUTATION**: SQUARE AND ACCUMULATE THE GRADIENTS IN THE GRADIENT ACCUMULATION VARIABLE, WHICH CONSEQUENTLY TRACKS THE SUM OF SQUARES OF THE GRADIENTS FOR EACH PARAMETER.

- **ADJUST LEARNING RATE**: MODIFY THE LEARNING RATE FOR EACH PARAMETER INVERSELY PROPORTIONAL TO THE SQUARE ROOT OF ITS ACCUMULATED GRADIENT, ENSURING PARAMETERS WITH SMALLER GRADIENTS TO HAVE LARGER UPDATES.

- **UPDATE PARAMETERS**: UPDATE EACH PARAMETER USING ITS ADJUSTED LEARNING RATE AND THE COMPUTED GRADIENT.

MATHEMATICAL REPRESENTATION

The updated learning rate for each parameter is calculated as follows:

$$\text{lr}_t = \frac{\eta}{\sqrt{G_t + \epsilon}}$$

Where:

- η is the global learning rate (a small constant value)
- G_t is the sum of squared gradients for a given parameter up to time step t
- ϵ is a small value added to avoid division by zero (often set to $1e-8$)

Here, the denominator $\sqrt{G_t + \epsilon}$ grows as the squared gradients accumulate, causing the learning rate to decrease over time, which helps to stabilize the training.

PARAMETER UPDATE

The model's parameters are updated by subtracting the product of the adaptive learning rate and the gradient at each step:

$$\theta_{t+1} = \theta_t - \mathrm{lr}_t \cdot \nabla_\theta$$

Where:

- θ_t is the current parameter
- $\nabla_\theta J(\theta)$is the gradient of the loss function with respect to the parameter

CHALLENGES

- **DIMINISHING LEARNING RATES**: AS TRAINING PROGRESSES, THE ACCUMULATED SQUARED GRADIENTS CAN GROW VERY LARGE, CAUSING THE LEARNING RATES TO SHRINK AND BECOME INFINITESIMALLY SMALL. THIS CAN PREMATURELY HALT THE LEARNING PROCESS.

ADVANTAGES

- **ADAPTIVE LEARNING RATES**: BY ADJUSTING THE LEARNING RATES BASED ON PAST GRADIENTS, ADAGRAD CAN EFFECTIVELY HANDLE DATA WITH SPARSE FEATURES AND DIFFERENT SCALES.

- **SIMPLICITY AND EFFICIENCY**: ADAGRAD SIMPLIFIES THE NEED FOR MANUAL TUNING OF THE LEARNING RATE, MAKING THE OPTIMIZATION PROCESS MORE STRAIGHTFORWARD.

RMSPROP (ROOT MEAN SQUARE PROPAGATION)

- RMSPROP WAS INTRODUCED TO SOLVE A PROBLEM INHERENT TO ADAGRAD: **THE DIMINISHING LEARNING RATE**. WHILE ADAGRAD ADAPTS THE LEARNING RATE BASED ON THE GRADIENT ACCUMULATION FOR EACH PARAMETER, IT TENDS TO REDUCE THE LEARNING RATE TOO DRASTICALLY OVER TIME, CAUSING THE MODEL TO STOP LEARNING EFFECTIVELY.

WHY DO WE NEED RMSPROP?

- TO UNDERSTAND RMSPROP, LET'S FIRST BRIEFLY DISCUSS **ADAGRAD'S CHALLENGE**. ADAGRAD WORKS WELL WITH SPARSE DATA — DATA WHERE MANY VALUES ARE ZERO. HOWEVER, AS THE LEARNING PROGRESSES, THE LEARNING RATE BECOMES SO SMALL THAT IT ALMOST STOPS MAKING UPDATES, PARTICULARLY IN MODELS WITH LONG TRAINING SESSIONS OR NON-CONVEX LOSS FUNCTIONS (LIKE IN NEURAL NETWORKS). RMSPROP HELPS OVERCOME THIS BY ADJUSTING THE LEARNING RATE IN A DIFFERENT WAY.

HOW RMSPROP WORKS

- RMSPROP KEEPS TRACK OF AN EXPONENTIALLY DECAYING AVERAGE OF THE SQUARED GRADIENTS. INSTEAD OF ACCUMULATING ALL PAST SQUARED GRADIENTS AS ADAGRAD DOES, RMSPROP USES A MOVING AVERAGE. THIS MEANS THAT RMSPROP GIVES MORE WEIGHT TO RECENT GRADIENTS, ALLOWING IT TO ADAPT MORE EFFECTIVELY WITHOUT DIMINISHING THE LEARNING RATE TOO QUICKLY.

- **THE RMSPROP FORMULA CAN BE WRITTEN AS:**

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot g_t^2$$

Here:

- v(t) is the accumulated moving average of squared gradients at time t.
- β is a decay rate, typically around **0.9** or **0.95**.
- g(t) represents the gradient at time t.

- **USING THIS, THE UPDATE RULE FOR EACH PARAMETER θ\THETAθ BECOMES:**

$$\theta = \theta - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t$$

where:
•η is the learning rate,
•ϵ is a small constant added to prevent division by zero.

EXAMPLE OF RMSPROP IN ACTION

- IMAGINE WE'RE TRAINING A NEURAL NETWORK TO RECOGNIZE HANDWRITTEN DIGITS. EACH TIME WE UPDATE THE NETWORK, GRADIENTS POINT IN THE DIRECTION THAT MINIMIZES THE LOSS. WITH ADAGRAD, AS THE UPDATES PROGRESS, THE LEARNING RATE MAY SHRINK TOO MUCH, STALLING THE TRAINING.
- USING RMSPROP, THE MOVING AVERAGE OF THE SQUARED GRADIENTS ALLOWS THE LEARNING RATE TO REMAIN STABLE ACROSS ITERATIONS. THE NETWORK LEARNS CONSISTENTLY WITHOUT DRASTIC SLOWDOWN, ULTIMATELY ACHIEVING BETTER PERFORMANCE.

ADVANTAGES OF RMSPROP

- **EFFICIENT WITH NON-CONVEX PROBLEMS**: RMSPROP PERFORMS WELL WITH COMPLEX, NON-CONVEX OPTIMIZATION PROBLEMS COMMON IN DEEP LEARNING, UNLIKE ADAGRAD, WHICH WORKS BEST IN CONVEX SETTINGS.
- **STABLE LEARNING RATE**: RMSPROP PREVENTS THE LEARNING RATE FROM DIMINISHING TOO QUICKLY, ALLOWING THE OPTIMIZER TO MOVE STEADILY TOWARDS THE MINIMUM.
- **EMPIRICALLY PROVEN PERFORMANCE**: RMSPROP IS WIDELY USED IN PRACTICE DUE TO ITS ROBUSTNESS AND STABILITY ACROSS A VARIETY OF NEURAL NETWORK ARCHITECTURES.

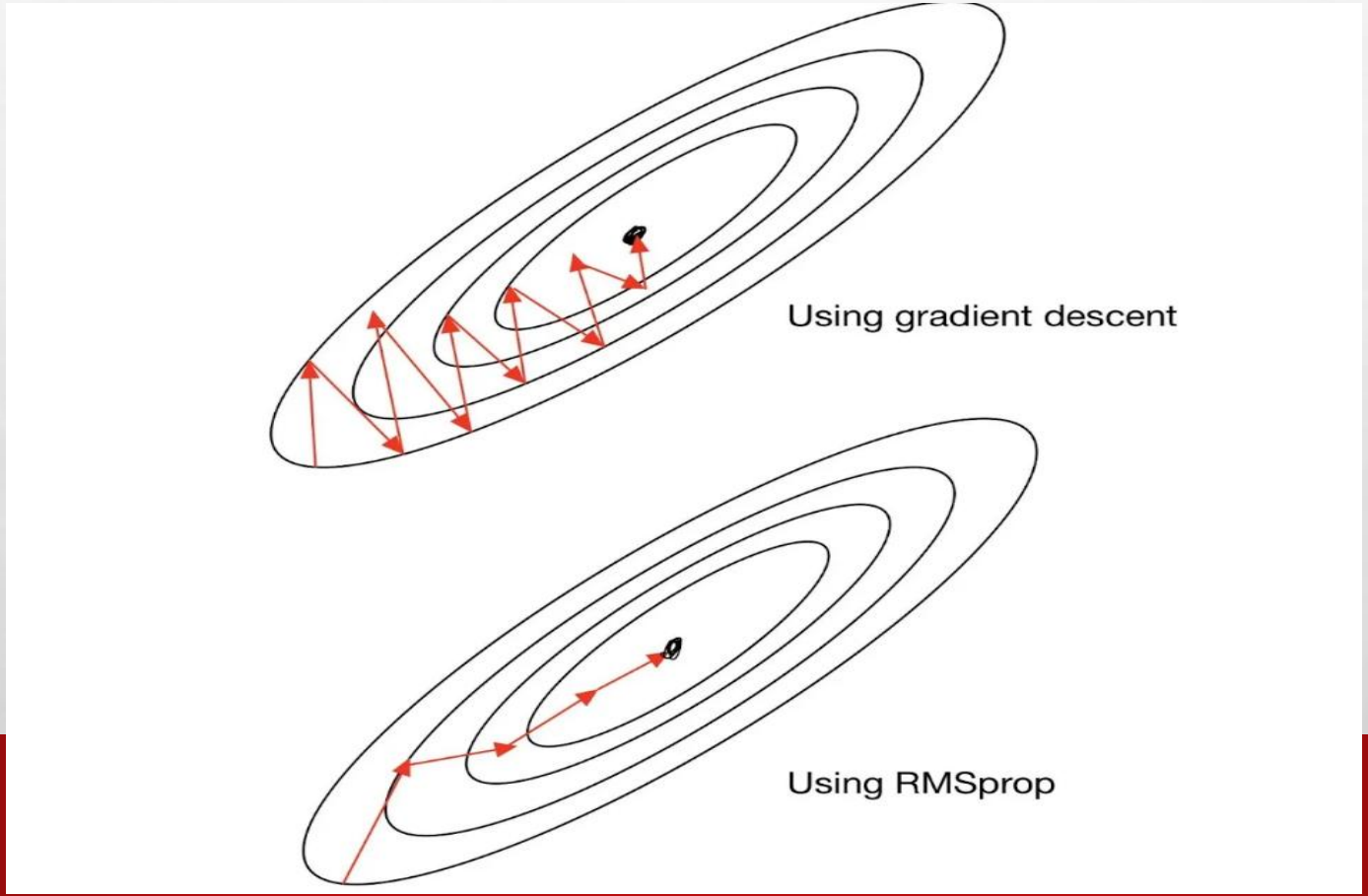
DISADVANTAGES OF RMSPROP

- WHILE RMSPROP IS A HIGHLY EFFECTIVE OPTIMIZER, IT HAS LARGELY BEEN SUPERSEDED BY **ADAM (ADAPTIVE MOMENT ESTIMATION)**, WHICH COMBINES THE BENEFITS OF BOTH RMSPROP AND MOMENTUM (ANOTHER OPTIMIZATION TECHNIQUE). ADAM GENERALLY OUTPERFORMS RMSPROP BUT COMES WITH ADDITIONAL COMPUTATIONAL OVERHEAD.

WHEN TO USE RMSPROP

- **RECURRENT NEURAL NETWORKS (RNNS)**: RMSPROP IS PARTICULARLY USEFUL IN RNNS AND OTHER MODELS PRONE TO VANISHING OR EXPLODING GRADIENTS.
- **SPARSE DATA**: RMSPROP HANDLES SPARSE DATA EFFECTIVELY, ADJUSTING LEARNING RATES DYNAMICALLY BASED ON RECENT GRADIENT VALUES.

TRAINING WITH GRADIENT DESCENT AND RMS PROP



ADADELTA

- ADADELTA CAN BE SEEN AS A MORE ROBUST VERSION OF THE ADAGRAD OPTIMIZER. IT IS BASED UPON ADAPTIVE LEARNING AND IS DESIGNED TO DEAL WITH SIGNIFICANT DRAWBACKS OF ADAGRAD AND RMS PROP OPTIMIZER. THE MAIN PROBLEM WITH THE ABOVE TWO OPTIMIZERS IS THAT THE INITIAL LEARNING RATE MUST BE DEFINED MANUALLY. ONE OTHER PROBLEM IS THE DECAYING LEARNING RATE WHICH BECOMES INFINITESIMALLY SMALL AT SOME POINT. DUE TO THIS, A CERTAIN NUMBER OF ITERATIONS LATER, THE MODEL CAN NO LONGER LEARN NEW KNOWLEDGE.
- TO DEAL WITH THESE PROBLEMS, ADADELTA USES TWO STATE VARIABLES TO STORE THE LEAKY AVERAGE OF THE SECOND MOMENT GRADIENT AND A LEAKY AVERAGE OF THE SECOND MOMENT OF CHANGE OF PARAMETERS IN THE MODEL.

$$\mathbf{s}_t = \rho \mathbf{s}_{t-1} + (1-\rho)\mathbf{g}_t^2.$$

$$\mathbf{x}_t = \mathbf{x}_{t-1} - \mathbf{g}'_t.$$

$$\mathbf{g}'_t = \frac{\sqrt{\Delta \mathbf{x}_{t-1} + \epsilon}}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t,$$

$$\Delta \mathbf{x}_t = \rho \Delta \mathbf{x}_{t-1} + (1-\rho)\mathbf{g}'^2_t,$$

Here St and delta Xt denote the state variables, g’t denotes rescaled gradient, delta Xt-1 denotes squares rescaled gradients, and epsilon represents a small positive integer to handle division by 0.

ADAM (ADAPTIVE MOMENT ESTIMATION)

- RMSPROP (ROOT MEAN SQUARE PROPAGATION) IS AN ADAPTIVE LEARNING RATE OPTIMIZATION ALGORITHM DESIGNED TO ADDRESS ADAGRAD'S DIMINISHING LEARNING RATES ISSUE.
- CONTINUING WITH THE ANALOGY OF HIKERS NAVIGATING A FOGGY VALLEY, RMSPROP EQUIPS OUR HIKERS WITH AN ADAPTIVE TOOL THAT ALLOWS THEM TO MAINTAIN A CONSISTENT PACE DESPITE THE TERRAIN'S COMPLEXITY. THIS TOOL EVALUATES THE RECENT TERRAIN AND ADJUSTS THEIR STEPS ACCORDINGLY, ENSURING THEY NEITHER GET STUCK IN DIFFICULT AREAS DUE TO EXCESSIVELY SMALL STEPS NOR OVERSHOOT THEIR TARGET WITH OVERLY LARGE STEPS. RMSPROP ACHIEVES THIS BY MODULATING THE LEARNING RATE BASED ON A MOVING AVERAGE OF THE SQUARED GRADIENTS.

OBJECTIVE:

- RMSPROP, LIKE ITS PREDECESSORS, AIMS TO OPTIMIZE THE MODEL'S PARAMETERS TO MINIMIZE THE COST FUNCTION. ITS KEY INNOVATION LIES IN ADJUSTING THE LEARNING RATE FOR EACH PARAMETER USING A MOVING AVERAGE OF RECENT SQUARED GRADIENTS, ENSURING EFFICIENT AND STABLE CONVERGENCE.

ADAM OPTIMIZER FORMULA

- THE ADAM OPTIMIZER HAS SEVERAL BENEFITS, DUE TO WHICH IT IS USED WIDELY. IT IS ADAPTED AS A BENCHMARK FOR DEEP LEARNING PAPERS AND RECOMMENDED AS A DEFAULT OPTIMIZATION ALGORITHM. MOREOVER, THE ALGORITHM IS STRAIGHTFORWARD TO IMPLEMENT, HAS A FASTER RUNNING TIME, LOW MEMORY REQUIREMENTS, AND REQUIRES LESS TUNING THAN ANY OTHER OPTIMIZATION ALGORITHM.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \left[\frac{\delta L}{\delta w_t} \right] v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left[\frac{\delta L}{\delta w_t} \right]^2$$

The above formula represents the working of adam optimizer. Here B1 and B2 represent the decay rate of the average of the gradients.

If the adam optimizer uses the good properties of all the algorithms and is the best available optimizer, then why shouldn't you use Adam in every application? And what was the need to learn about other algorithms in depth? This is because even Adam has some downsides. It tends to focus on faster computation time, whereas algorithms like stochastic gradient descent focus on data points. That's why algorithms like SGD generalize the data in a better manner at the cost of low computation speed. So, the optimization algorithms can be picked accordingly depending on the requirements and the type of data.

REGULARIZATION TECHNIQUES

REFER TO THIS LINK FOR REGULARIZATION

- HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/04/FUNDAMENTALS-DEEP-LEARNING-REGULARIZATION-TECHNIQUES/

THANK YOU