Module 1

Outlines

- Information flow in a neural network
- Understanding the basic structure of ANN
- ANN

What is Deep Learning

Deep learning is a subset of machine learning that focuses on using artificial neural networks to model and process complex patterns in data. These neural networks consist of multiple layers (hence the term "deep"), allowing them to learn hierarchical representations of data.

Key Aspects of Deep Learning:

1. **Neural Networks**: Deep learning models use artificial neural networks inspired by the human brain, typically composed of input, hidden, and output layers.
2. **Multiple Layers**: Unlike traditional machine learning, deep learning networks have multiple hidden layers, which enable them to extract intricate features from raw data.
3. **Feature Learning**: Instead of relying on hand-crafted features, deep learning models learn relevant features automatically from large amounts of data.
4. **Training with Large Datasets:** Deep learning requires substantial amounts of data and computational power to perform effectively.
5. **Backpropagation & Optimization:** The learning process involves backpropagation and optimization techniques like stochastic gradient descent (SGD) to adjust model parameters.

Commonly used Deep Learning Architectures

- **Convolutional Neural Networks (CNNs)** – Used for image recognition, object detection, and video analysis.
- **Recurrent Neural Networks (RNNs)** – Effective for sequential data like speech and text.
- **Transformers** – Advanced architectures used in NLP (e.g., GPT, BERT).
- **Generative Adversarial Networks (GANs)** – Used for generating realistic images and other synthetic data.
- **Autoencoders** – Used for unsupervised learning and anomaly detection.

Applications of Deep Learning:

- **Computer Vision** – Face recognition, medical imaging, self-driving cars.
- **Natural Language Processing (NLP)** – Chatbots, language translation, sentiment analysis.
- **Speech Recognition** – Virtual assistants like Siri and Alexa.
- **Recommendation Systems** – Personalized content on platforms like Netflix and YouTube.
- **Autonomous Systems** – Robotics and automated decision-making.

Difference between Machine Learning and Deep Learning

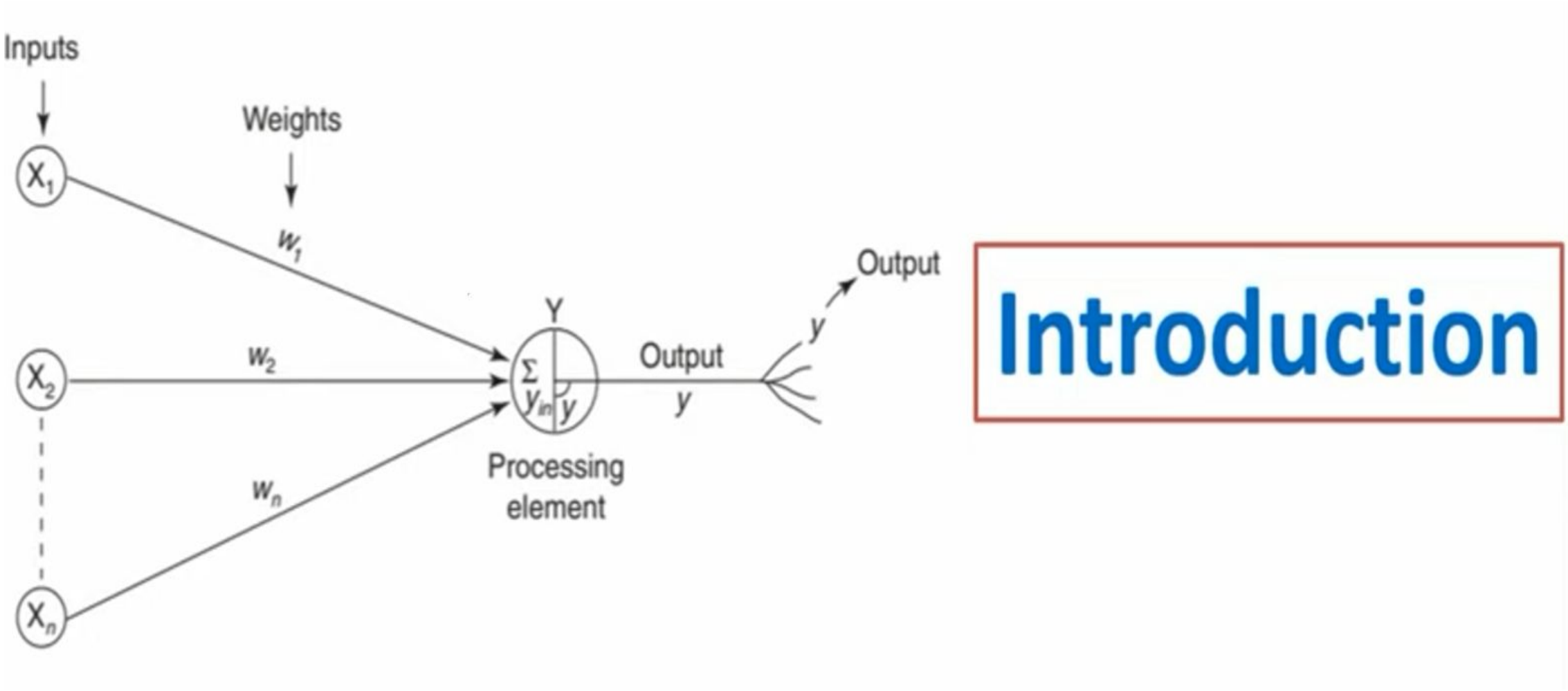
Feature	Machine Learning	Deep Learning
Definition	A subset of AI that enables systems to learn from data and improve without being explicitly programmed.	A subset of machine learning that uses artificial neural networks to learn patterns from data.
Feature Extraction	Requires manual feature extraction by domain experts.	Automatically extracts features from raw data.
Data Dependency	Works well with small to medium-sized datasets.	Requires large datasets for optimal performance.
Computational Power	Can run on standard computers with moderate resources.	Needs high-performance GPUs/TPUs for training.
Training Time	Generally shorter training time.	Longer training time due to complex architectures.
Interpretability	Easier to interpret and explain results.	Often considered a "black box" due to complexity.
Use Cases	Fraud detection, recommendation systems, predictive analytics, spam detection.	Image recognition, speech processing, autonomous driving, NLP (e.g., chatbots, GPT models).

Difference between Machine Learning and Deep Learning

Main Conceptual Difference

- **Machine Learning** relies on structured data and requires feature engineering, where domain experts define which features (characteristics) the model should focus on.
- **Deep Learning** eliminates the need for manual feature extraction by automatically learning hierarchical features from raw data through **neural networks**.

Basic Structure of ANN



Basic Structure of ANN(Cont…)

- An artificial neural network (ANN) may be defined as an information-processing model that is inspired by the way biological nervous systems, such as the brain, process information.
- This model tries to replicate only the most basic functions of the brain.
- An ANN is composed of a large number of highly interconnected processing units (neurons) working in unison to solve specific problems.
- Like human being, Artificial neural networks learn by example.
- An ANN is configured for a specific application, such as spam classification, Face Recognition, pattern recognition through a learning process.

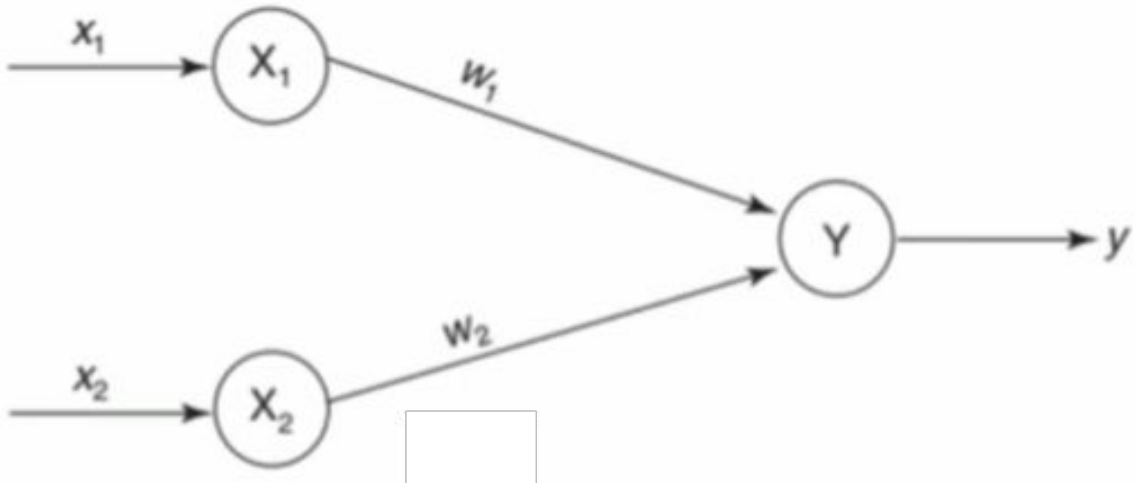
Basic Structure of ANN(Cont…)

- Each neuron is connected with the other by a connection link.
- Each connection link is associated with weights which contain information about the input signal.
- This information is used by the neuron network to solve a particular problem.
- ANNs' collective behavior is characterized by their ability to learn, recall and generalize training patterns or data similar to that of a human brain.
- They have the capability to model networks of original neurons as found in the brain.
- Thus, the ANN processing elements are called neurons or artificial neurons.

Basic Structure of ANN(Cont…)

- Each neuron has an internal state of its own.
- This internal state is called the activation level of neuron
- The activation signal of a neuron is transmitted to other neurons.
- Remember, a neuron can send only one signal at a time, which can be transmitted to several other neurons.

Basic Structure of ANN(Cont…)

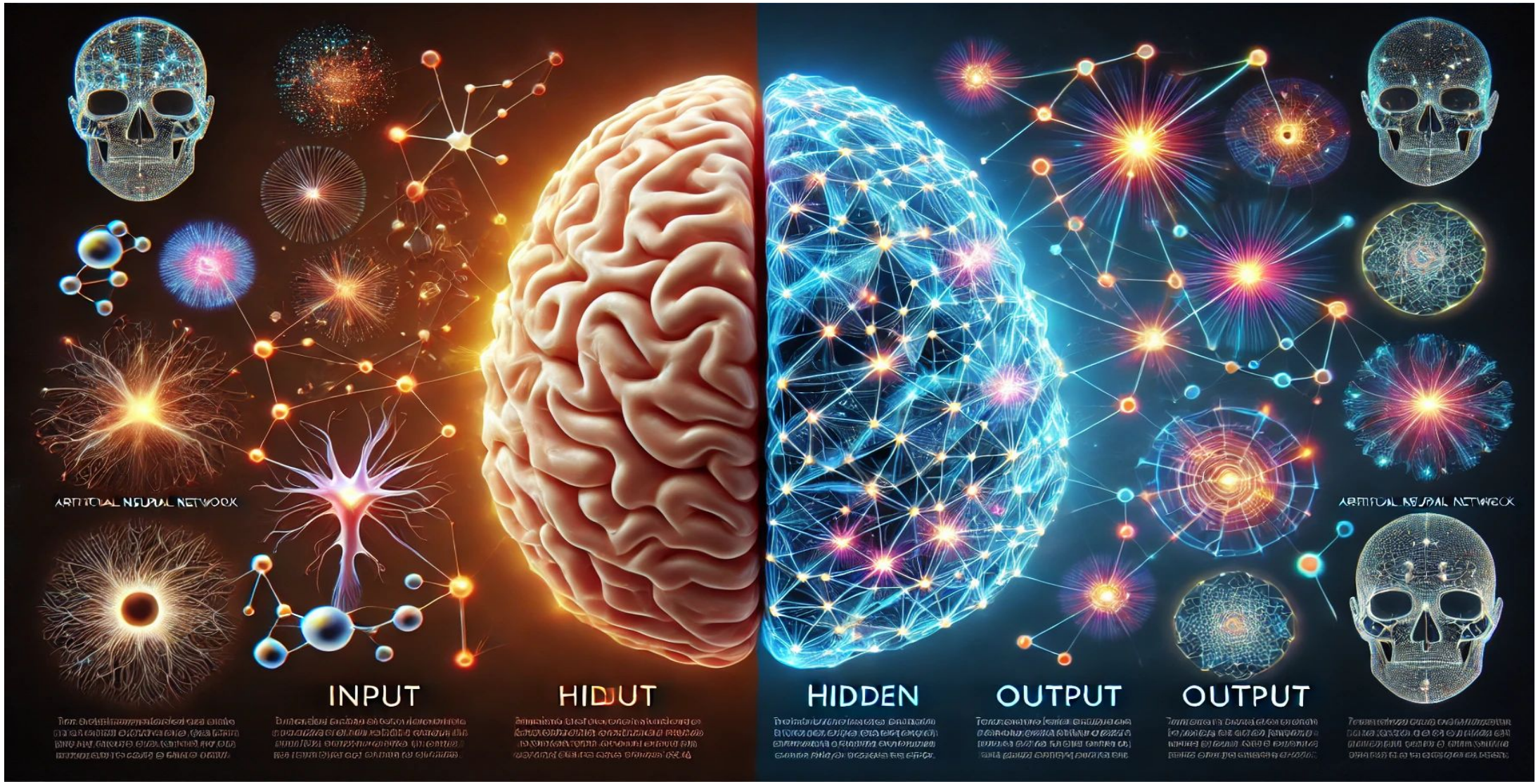


$$y = f(y_{in})$$

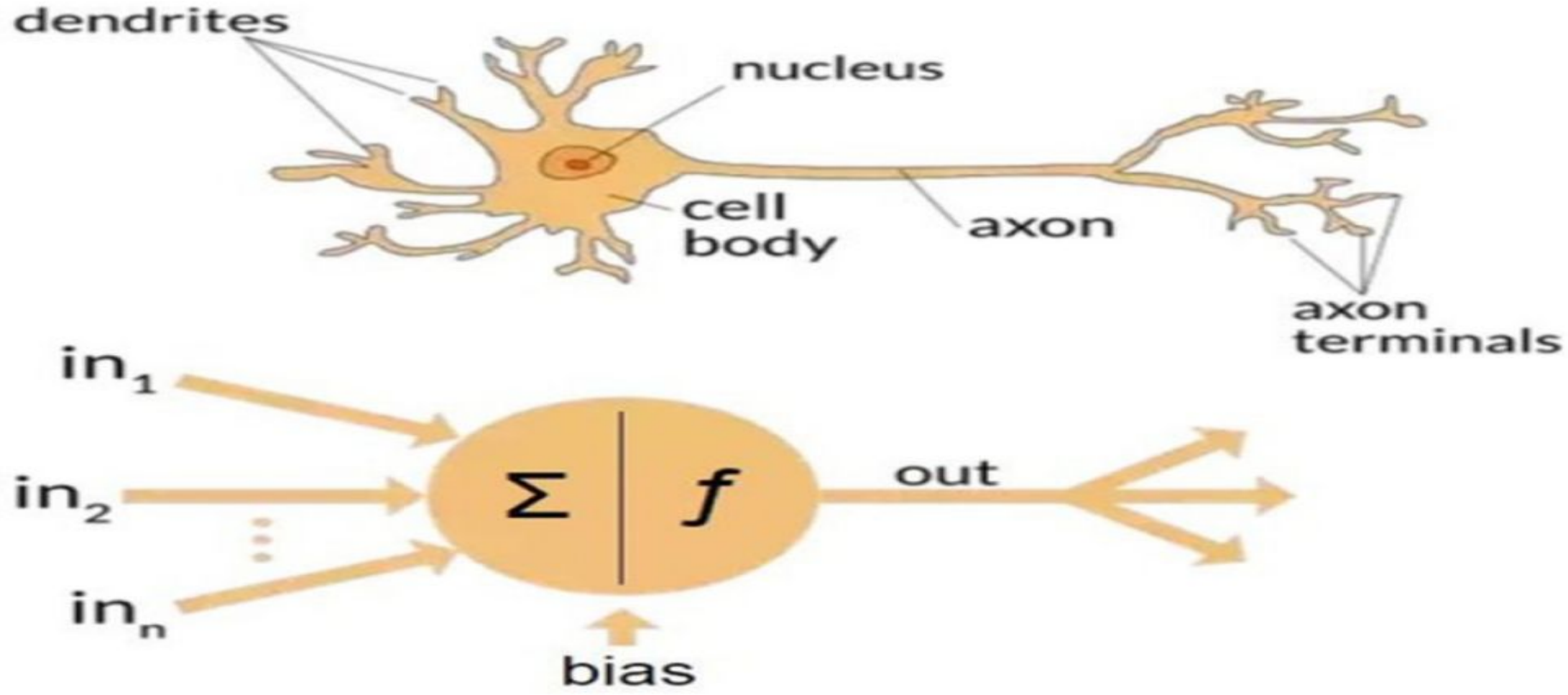
- Input neurons X1 and X2 are connected to the output neuron Y, over a weighted interconnection links (W1 and W2).
- For the above simple neuron net architecture, the net input has to be calculated in the following way:

$$y_{in} = x_1 w_1 + x_2 w_2$$

Basic Structure of ANN(Cont…)



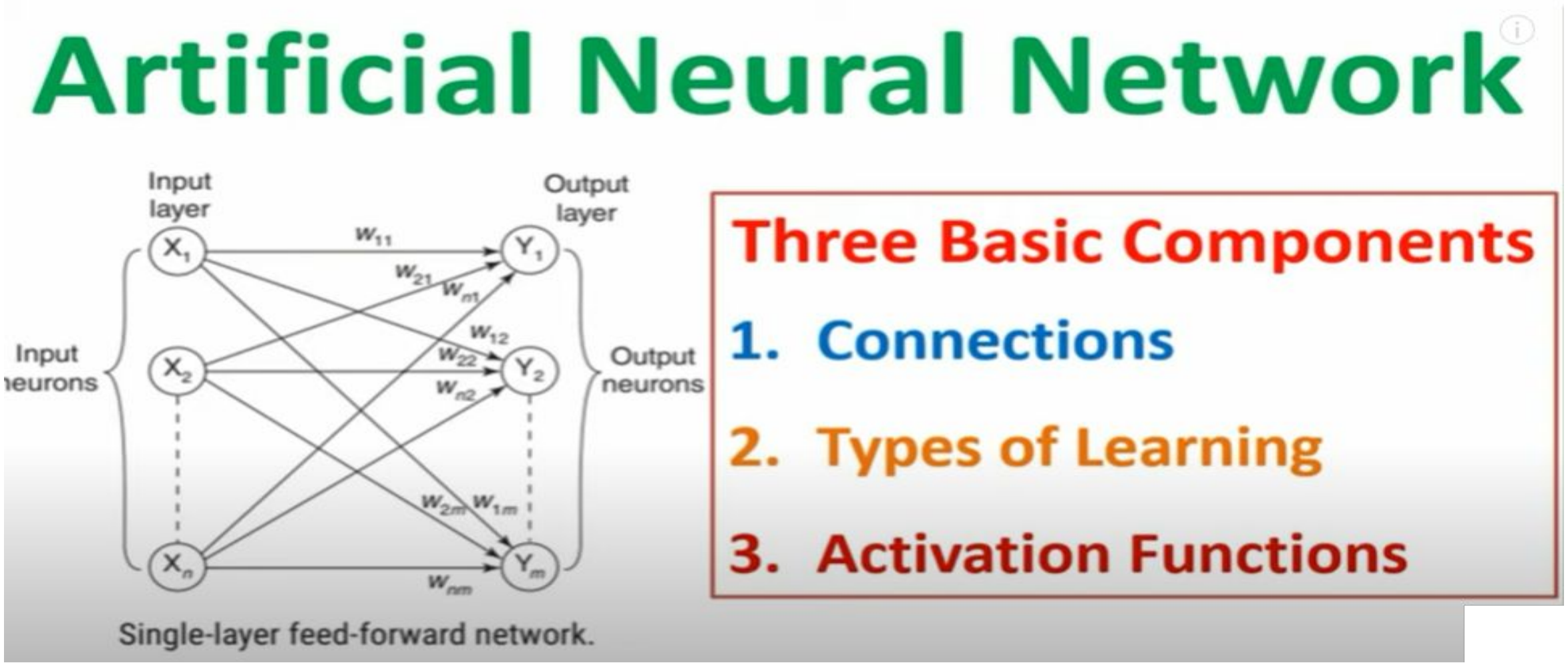
Comparison of BNN and ANN



Difference between ANN and BNN

#	Aspect	Artificial Neural Network (ANN)	Biological Neural Network (BNN)
1.	**Basic Unit**	Artificial Neuron (Perceptron)	Biological Neuron
2.	**Structure**	Fixed layers (input, hidden, output)	Dynamic, self-rewiring connections
3.	**Signal Transmission**	Numerical weights & activation functions	Electrical & chemical signals (neurotransmitters)
4.	**Processing Type**	Sequential & limited parallelism	Highly parallel & distributed
5.	**Learning Mechanism**	Backpropagation, gradient descent	Hebbian learning, synaptic plasticity
6.	**Adaptability**	Requires retraining for new tasks	Learns & adapts continuously
7.	**Energy Efficiency**	High power consumption (e.g., GPUs)	Extremely efficient (~20W for the brain)
8.	**Fault Tolerance**	Can fail if key nodes are removed	Self-repairing, highly robust
9.	**Generalization**	Needs large datasets for training	Learns from few examples, strong generalization
10.	**Application**	Used for AI tasks (e.g., image recognition, NLP)	Enables cognition, perception, creativity

Artificial Neural Network



3 Basic components or entities of ANN

The models of ANN are specified by the three basic entities namely:

1. the model's synaptic interconnections;
2. the training or learning rules adopted for updating and adjusting the connection weights;
3. their activation functions.

1. Connections

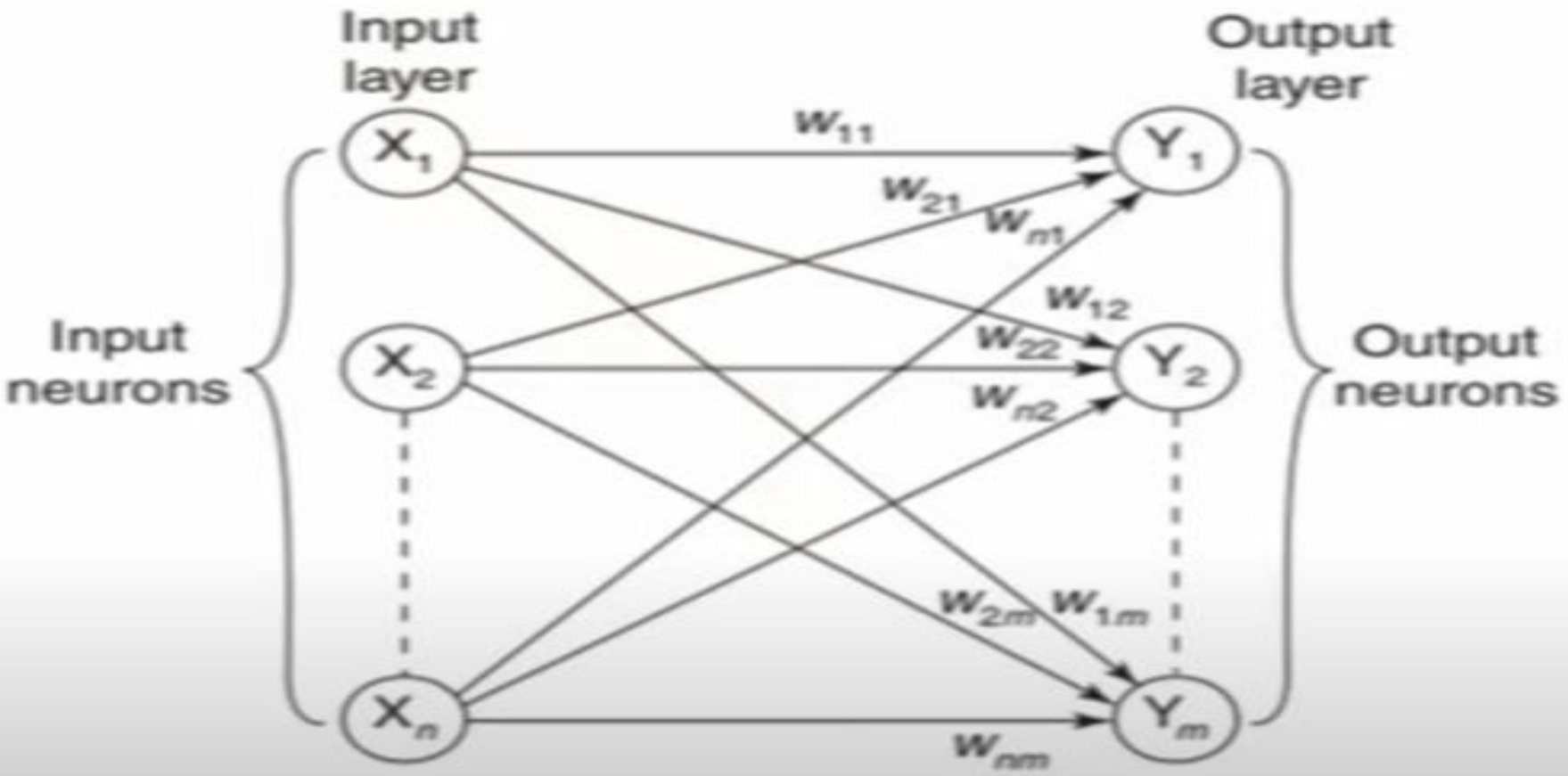
- An ANN consists of a set of highly interconnected processing elements (neurons) such that each processing element output is found to be connected through weights to the other processing elements or to itself; delay lead and lag-free connections are allowed.
- Hence, the arrangements of these processing elements and the geometry of their interconnections are essential for an ANN.

1. Connections

There exist five basic types of neuron connection architectures. They are:

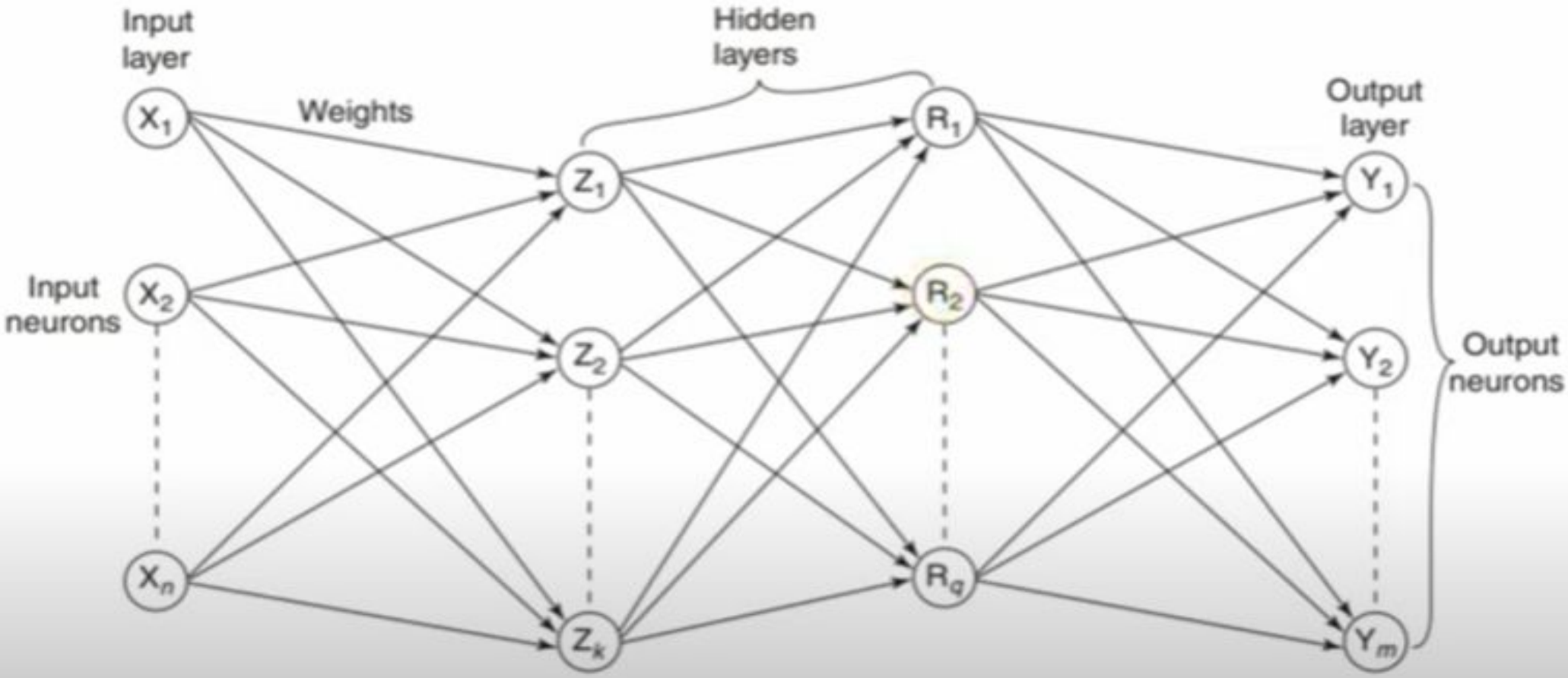
1. single-layer feed-forward network
2. multilayer feed-forward network
3. single node with its own feedback
4. single-layer recurrent network
5. multilayer recurrent network.

1. Connections



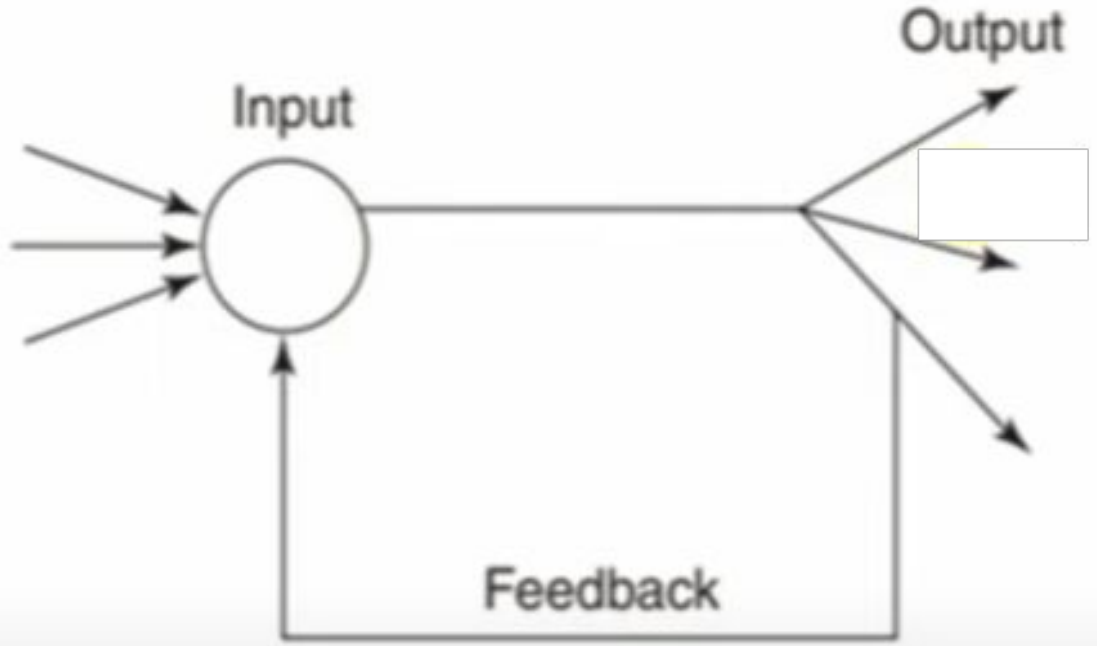
Single-layer feed-forward network.

1. Connections



Multilayer feed-forward network.

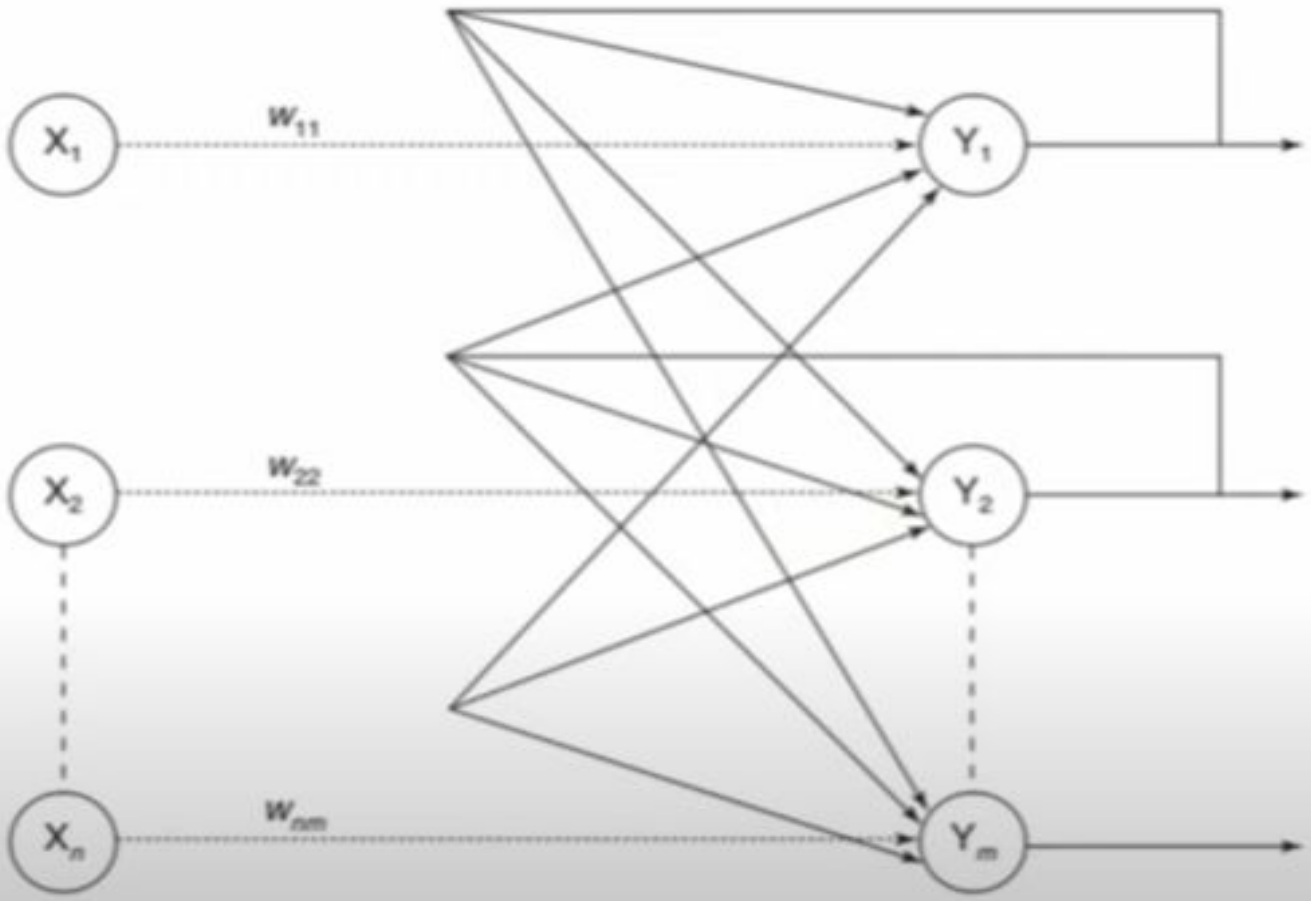
1. Connections



(A)

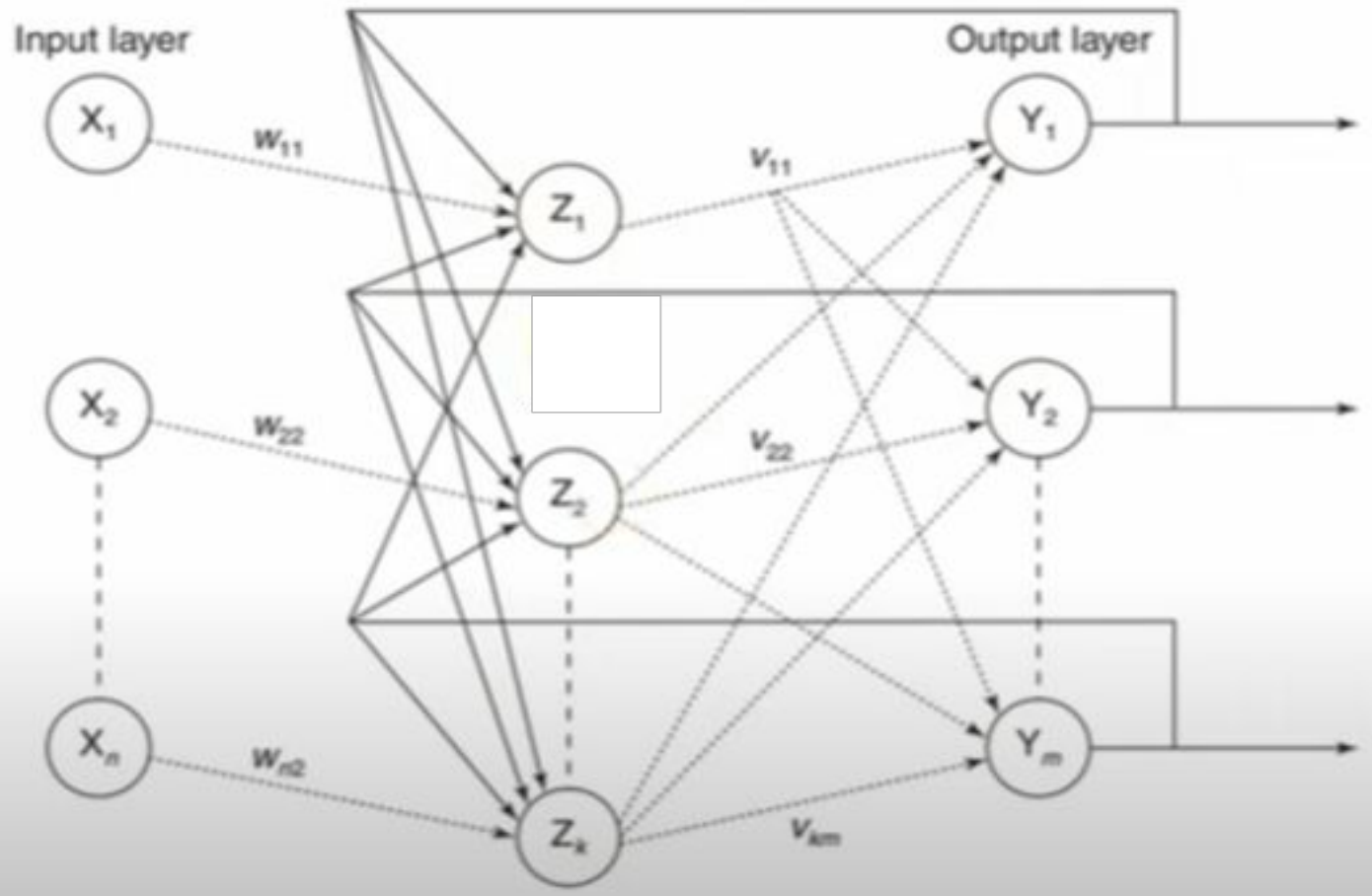
Single node with own feedback.

1. Connections



Single-layer recurrent network.

1. Connections



Multilayer recurrent network.

2. Learning

- The main property of an ANN is its capability to learn.
- Learning or training is a process by means of which a neural network adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response.
- Broadly, there are two kinds of learning in ANNs:
  1. **Parameter learning:** It updates the connecting weights in a neural net.
  2. **Structure learning**: It focuses on the change in network structure (which includes the number of processing elements as well as their connection types).

3. Activation Function

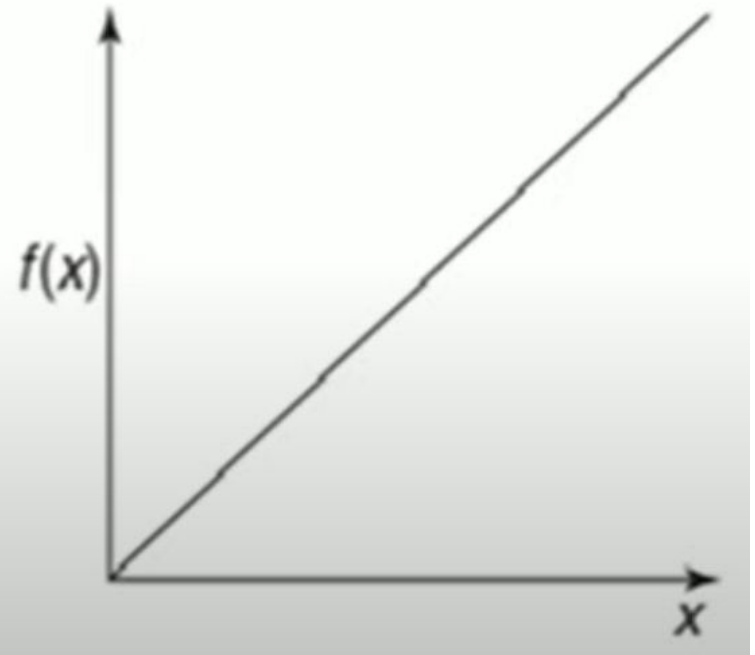
There are several activation functions.

1. Identity function: It is a linear function and can be defined as

$$f(x) = x \quad \text{for all } x$$

The output here remains the same as input.

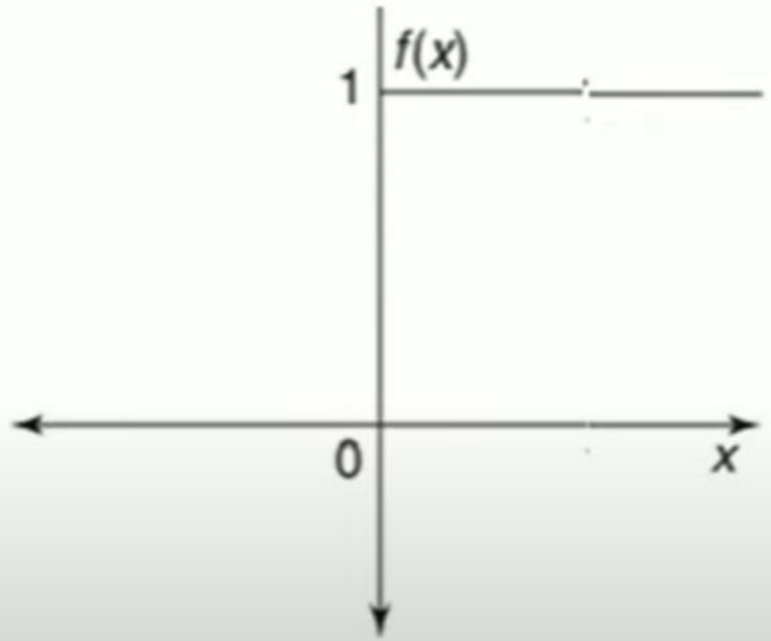
The input layer uses the identity activation function.



3. Activation Function

2. Binary step function: This function can be defined as

$$f(x) = \begin{cases} 1 & \text{if} \quad x \geq \theta \\ 0 & \text{if} \quad x < \theta \end{cases}$$



where q represents the threshold value.

This function is most widely used in single-layer nets

to convert the net input to an output that is a binary (1 or 0).

3. Activation Function

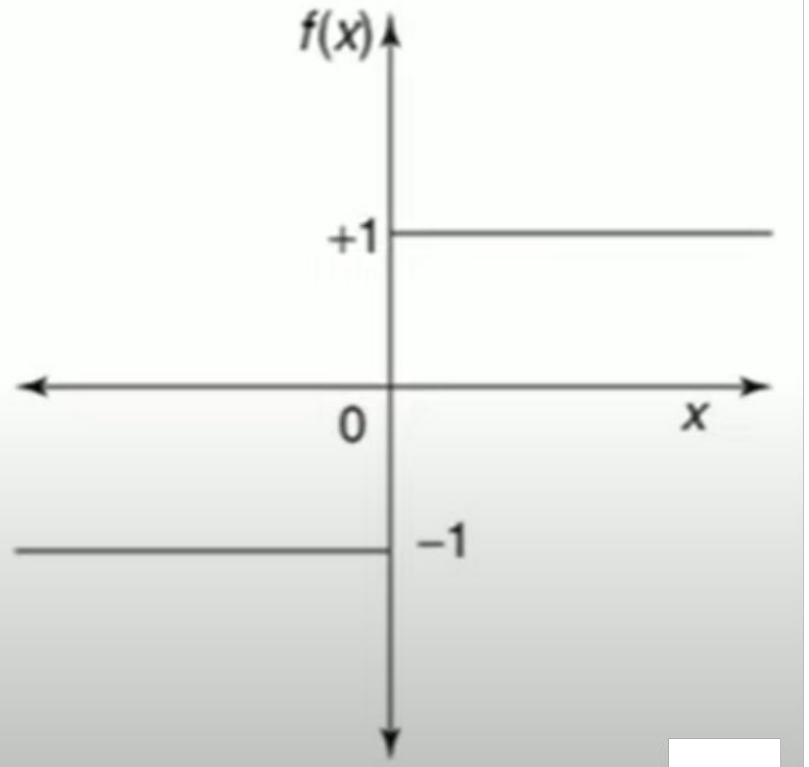
3. Bipolar step function: This function can be defined as

$$f(x) = \begin{cases} 1 & \text{if} \quad x \geq \theta \\ -1 & \text{if} \quad x < \theta \end{cases}$$

where θ represents the threshold value.

This function is also used in single-layer nets to convert

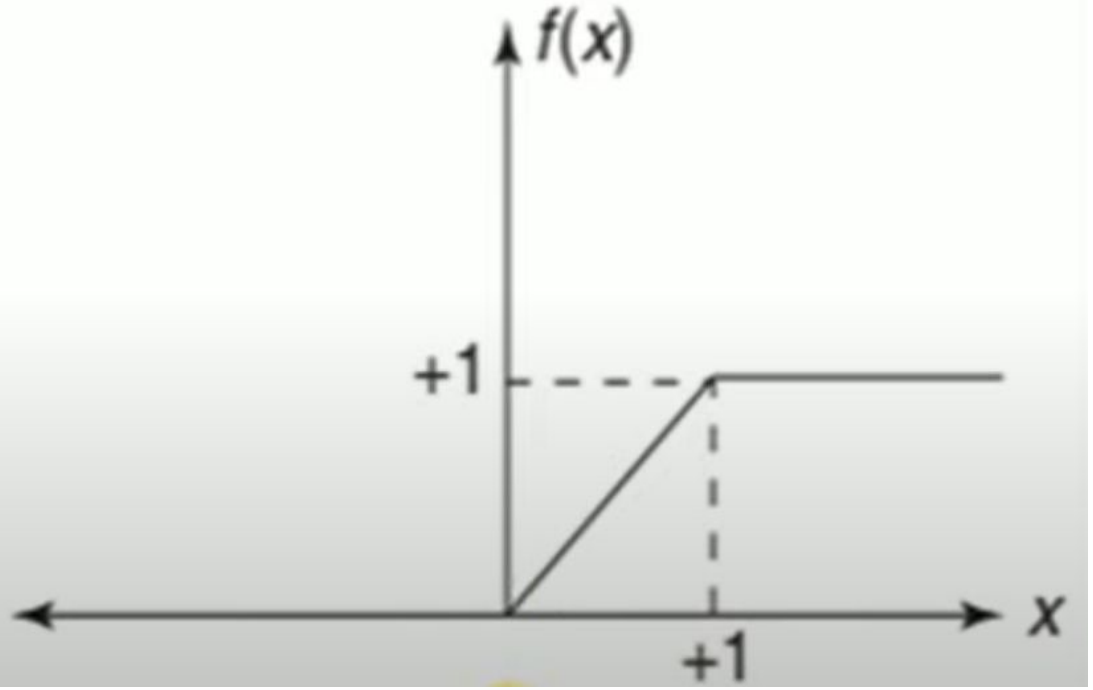
the net input to an output that is bipolar (+1 or –1).



3. Activation Function

4. Ramp function: The ramp function is defined as

$$f(x)=\begin{cases}1 & \text{if} \quad x>1\\ x & \text{if} \quad 0\le x\le 1\\ 0 & \text{if} \quad x<0\end{cases}$$



3. Activation Function

5. Sigmoidal functions: The sigmoidal functions are widely used in back-propagation nets because of the relationship between the value of the functions at a point and the value of the derivative at that point which reduces the computational burden during training.

Sigmoidal functions are of two types:

- Binary sigmoid function
- Bipolar sigmoid function

3. Activation Function

5.1 Binary sigmoid function: It is also termed as logistic sigmoid function or unipolar sigmoid function. It can be defined as

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

where λ is the steepness parameter. The derivative of this function is

$$f'(x) = \lambda f(x)[1-f(x)]$$

Here the range of the sigmoid function is from 0 to 1.

3. Activation Function

5.2 Bipolar sigmoid function: is defined as

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

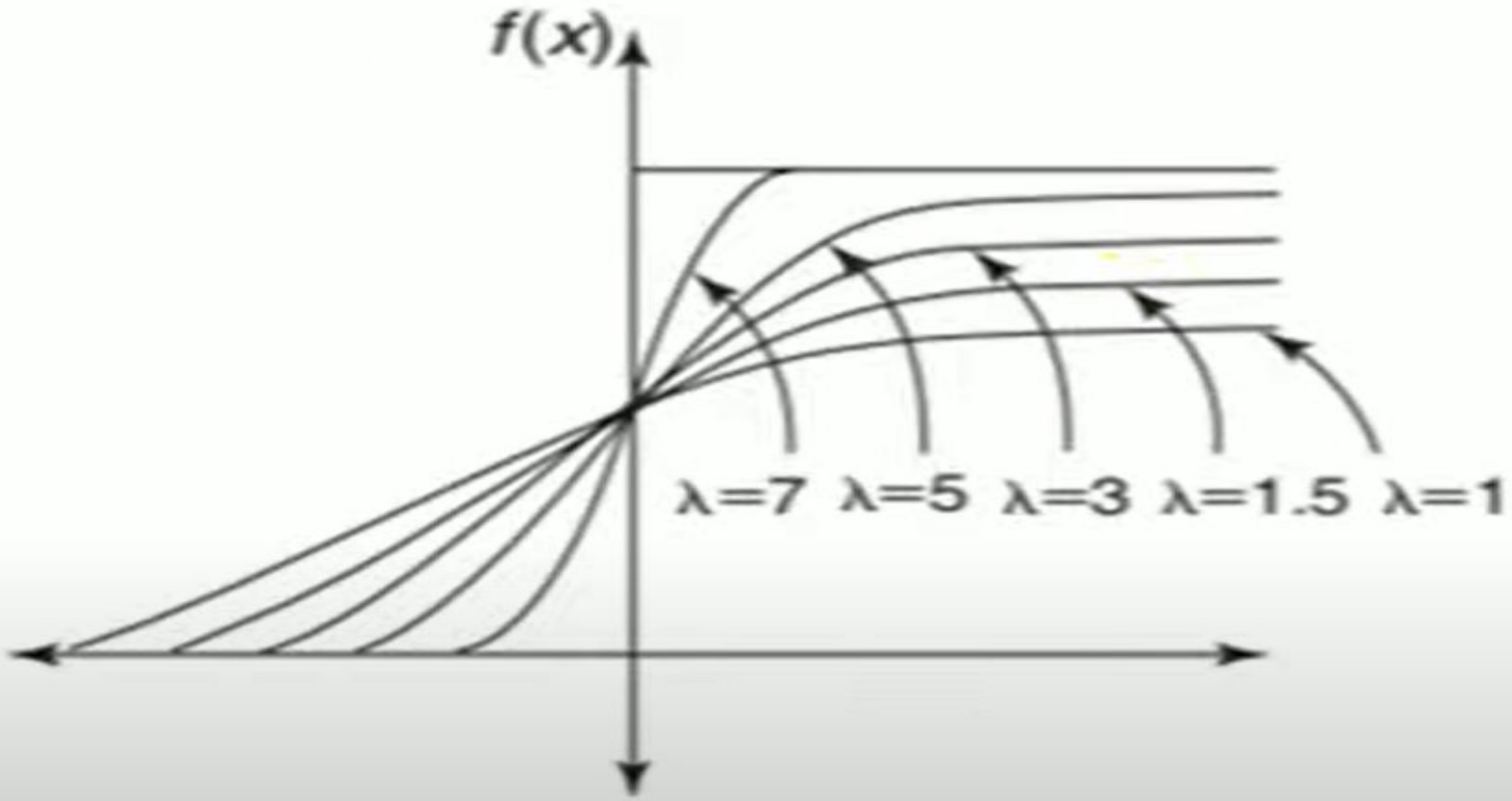
where λ is the steepness parameter and

the sigmoid function range is between –1 and +1.

The derivative of this function can be

$$f'(x) = \frac{\lambda}{2}[1+f(x)][1-f(x)]$$

3. Activation Function



3. Activation Function

3. Tanh (Hyperbolic Tangent) Activation Function

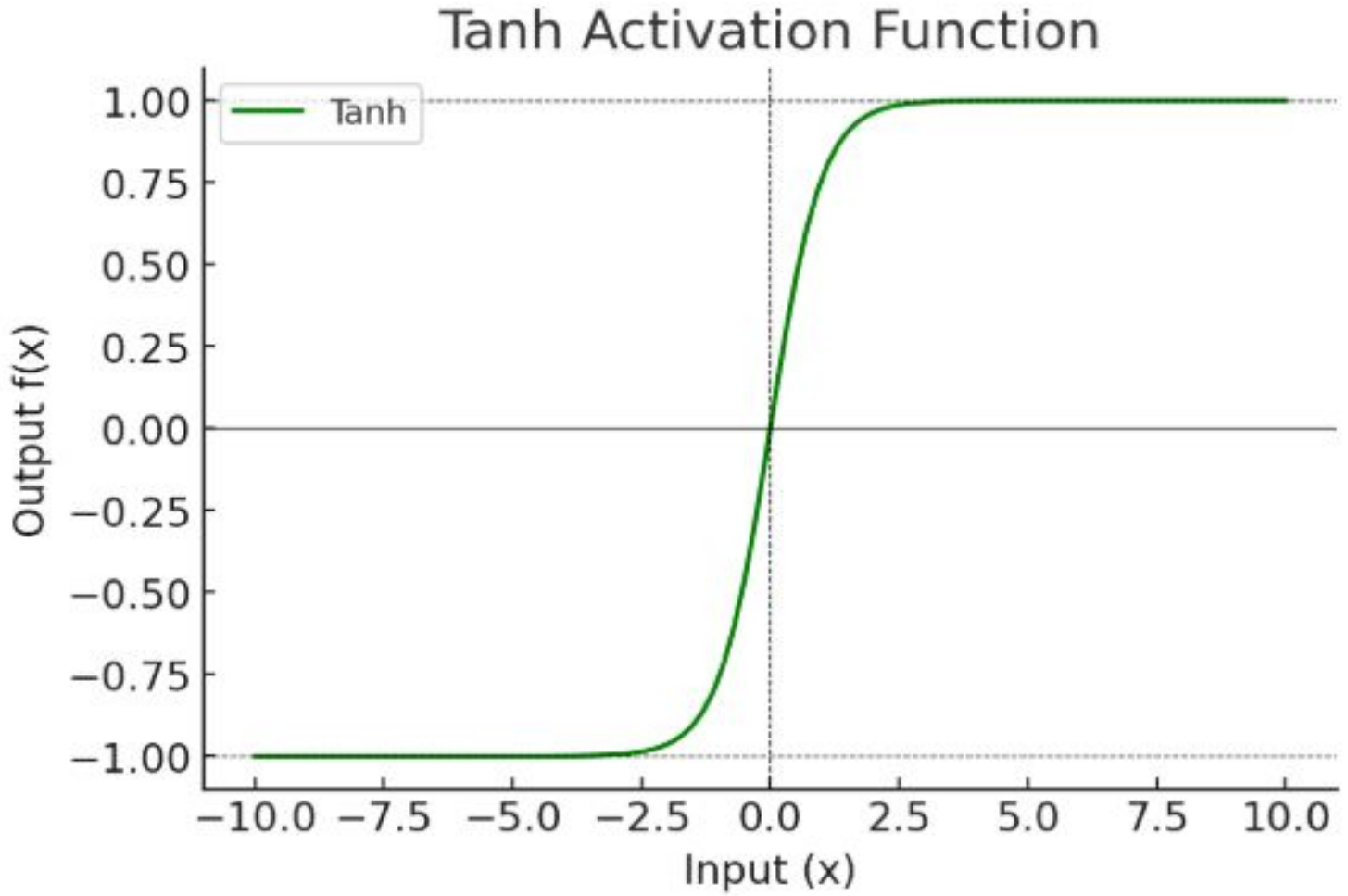
Equation:

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

Properties:

- **Range:** (-1,1)
- **Nature:** S-shaped (sigmoidal) but zero-centered
- **Use Case:** Preferred over Sigmoid for hidden layers (better gradient flow)
- **Advantages:**
  - **Zero-centered:** Leads to better weight updates
  - **Stronger gradients:** Steeper than Sigmoid
- **Disadvantages:**
  - **Vanishing Gradient Problem:** Similar to Sigmoid for extreme values

3. Activation Function



3. Activation Function

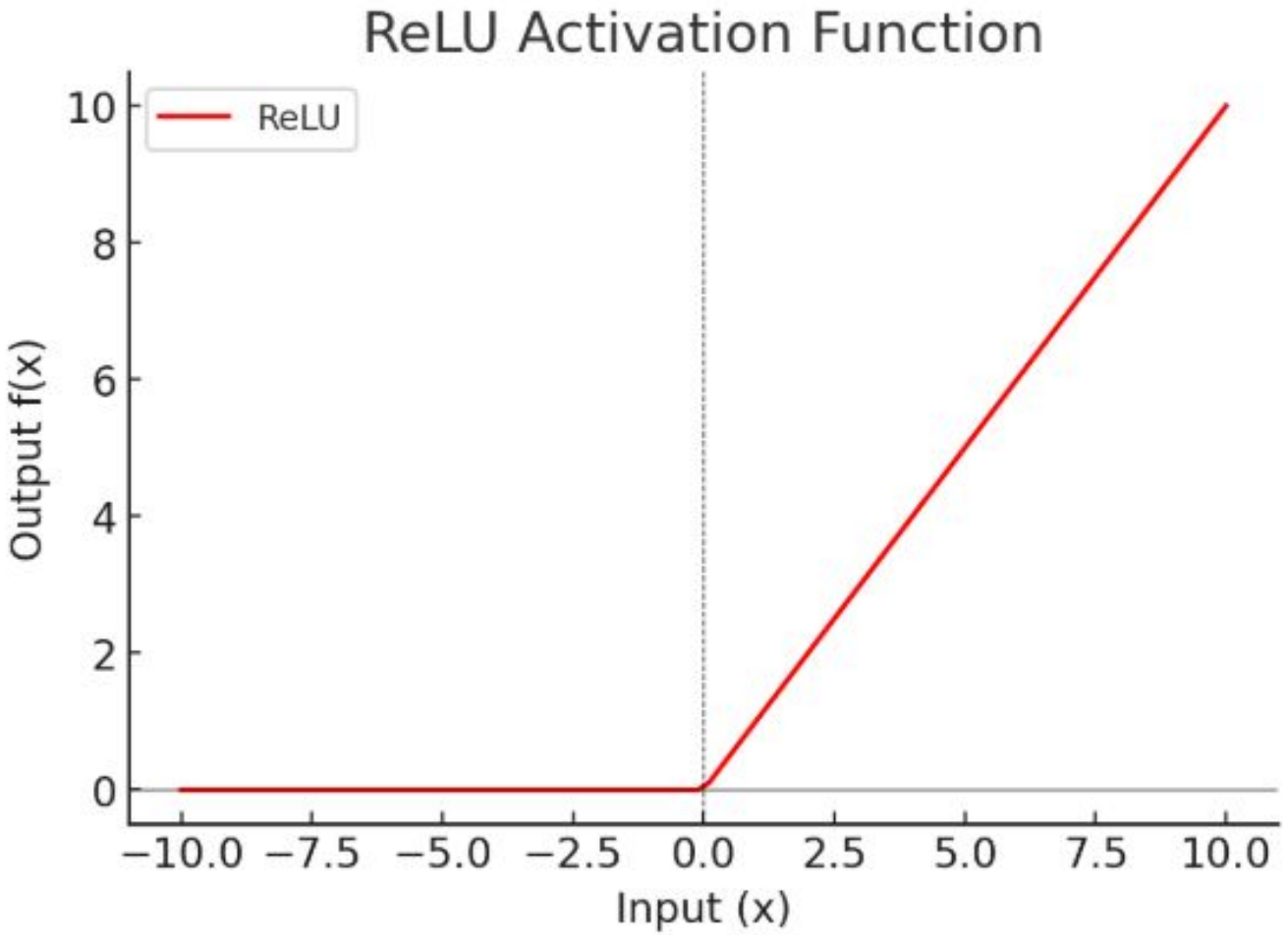
Equation:

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

Properties:

- **Range:** (0, ∞)
- **Nature:** Linear for positive values, zero for negatives
- **Use Case:** Most commonly used in deep networks due to efficiency
- **Advantages:**
  - **Solves vanishing gradient problem** for positive values
  - **Computationally efficient** (simple max operation)
- **Disadvantages:**
  - **Dying ReLU Problem:** If inputs become negative, neurons stop learning (gradient = 0).

3. Activation Function



3. Activation Function

Summary of Sigmoid, ReLU, and Tanh Activation Functions

Activation Function	Equation	Range	Pros	Cons	Use Cases
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0,1)	Good for probabilities, smooth output	Vanishing gradient, not zero-centered	Binary classification (output layer)
ReLU	$f(x) = \max(0, x)$	(0,∞)	Fast, avoids vanishing gradient	Dying ReLU problem (zero gradient for negative values)	Hidden layers in deep learning
Tanh	$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	(-1,1)	Zero-centered, better gradient flow than Sigmoid	Vanishing gradient at extremes	Hidden layers (alternative to ReLU)

Choosing the Right Activation Function

- **Sigmoid:** Used for binary classification in the output layer.
- **ReLU:** The most popular choice for hidden layers in deep networks.
- **Tanh:** Better than Sigmoid but still suffers from vanishing gradient.

Importance of Activation Function

- First, we understand the importance of the activation function in ANN.
- Let us assume a person is performing some task.
- To make the task more efficient and to obtain correct results, some force, or Motivation may be given.
- This force, or Motivation helps in achieving the correct results.
- In a similar way, the activation function is applied over the net input of the network to calculate the output of an ANN.

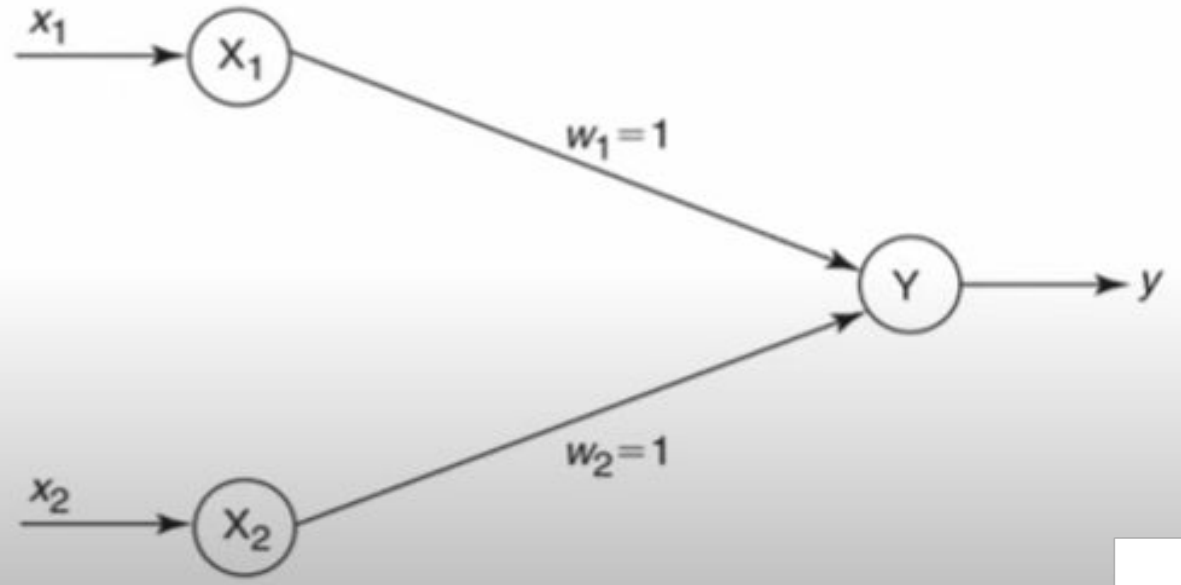
Types of Neural Networks and its functionalities

Type of Neural Network	Architecture	Functionality	Use Cases
1. Perceptron	Single-layer, single neuron	Basic classification (linear separation)	Logical operations (AND, OR)
2. Feedforward Neural Network (FNN)	Input → Hidden → Output (fully connected)	General pattern recognition, regression	Simple image and text classification
3. Convolutional Neural Network (CNN)	Convolutional + pooling layers	Feature extraction from images	Image recognition, object detection
4. Recurrent Neural Network (RNN)	Loops within neurons (memory)	Sequential data processing	Speech recognition, language modeling
5. Long Short-Term Memory (LSTM)	Special RNN with memory gates	Captures long-term dependencies in sequences	Text generation, machine translation
6. Gated Recurrent Unit (GRU)	Simplified LSTM (fewer parameters)	Faster training for sequential tasks	Chatbots, time series prediction
7. Autoencoder	Encoder → Bottleneck → Decoder	Unsupervised learning, feature extraction	Noise reduction, anomaly detection
8. Generative Adversarial Network (GAN)	Generator vs. Discriminator	Generates new data (images, text)	Deepfake generation, image synthesis
9. Transformer Networks	Attention mechanism, no recurrence	Processes large sequences efficiently	NLP tasks (ChatGPT, BERT)
10. Radial Basis Function (RBF) Network	Uses radial basis activation	Function approximation	Handwriting recognition, medical diagnosis

McCulloch-Pitts Neuron

Implement AND function using McCulloch–Pitts Neuron

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

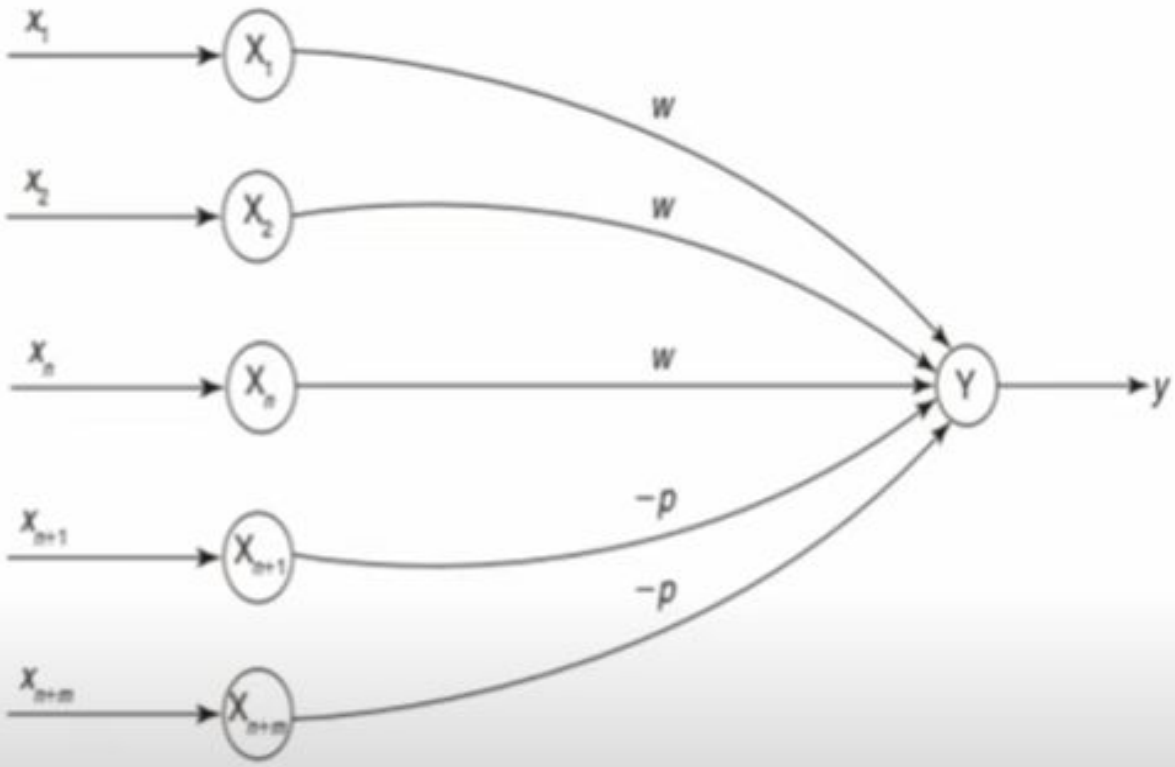


McCulloch-Pitts Neuron

Implement AND function using McCulloch–Pitts Neuron

- The McCulloch–Pitts neuron was the earliest neural network discovered in 1943.
- It is usually called as M–P neuron.
- Since the firing of the output neuron is based upon the threshold, the activation function here is defined as

$$f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq \theta \\ 0 & \text{if } y_{in} < \theta \end{cases}$$



- The threshold value should satisfy the following condition: $\theta > nw - p$

McCulloch-Pitts Neuron

- Consider the truth table for AND function
- The M–P neuron has no particular training algorithm
- In M-Pneuron, only analysis is being performed.
- Hence, assume the weights be w1 = 1 and w2 = 1.

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0

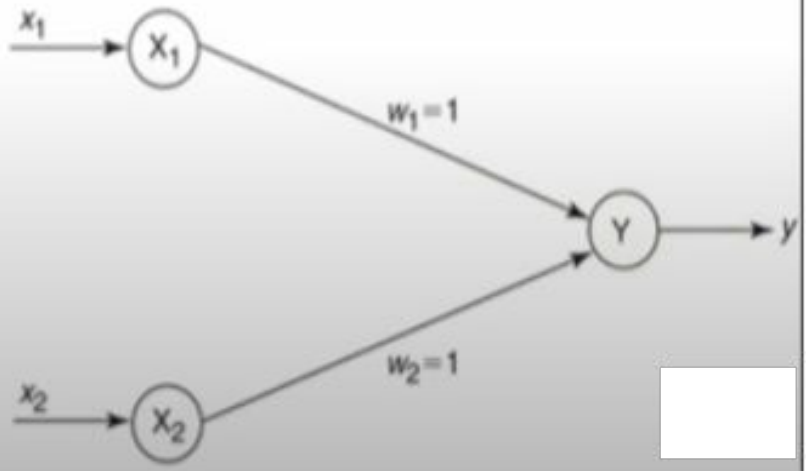
$$(1, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 1 \times 1 = 2$$

$$(1, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 1 \times 1 + 0 \times 1 = 1$$

$$(0, 1),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 1 \times 1 = 1$$

$$(0, 0),\ y_{in} = x_1 w_1 + x_2 w_2 = 0 \times 1 + 0 \times 1 = 0$$

Threshold value is set equal to 2 ($\theta = 2$).



McCulloch-Pitts Neuron

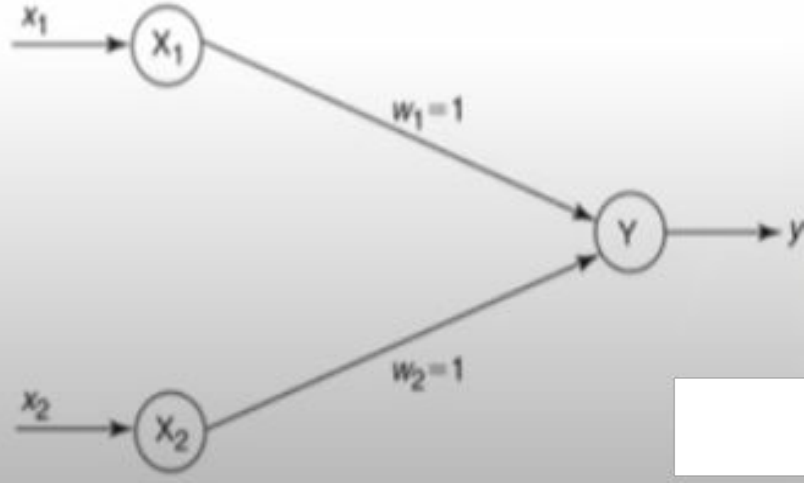
- This can also be obtained by

$$\theta \geq nw - p$$

- Here, n = 2, w = 1 (excitatory weights) and p = 0 (no inhibitory weights).
- Substituting these values in the above-mentioned equation we get $\theta \geq 2 \times 1 - 0 \Rightarrow \theta \geq 2$
- Thus, the output of neuron Y can be written

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq 2 \\ 0 & \text{if } y_{in} < 2 \end{cases}$$

x_1	x_2	y
1	1	1
1	0	0
0	1	0
0	0	0



What is Bias?

- Bias is a constant value (not dependent on input) added to the weighted sum of inputs before passing it through the activation function.
- Helps the neuron activate even when all input values are zero.
- Increases the flexibility of the model by shifting the activation threshold.
- Improves the learning ability of the network.

Mathematical Representation:

$$y = f\left(\sum(w_i \cdot x_i) + b\right)$$

What is Variance?

- Variance in an Artificial Neural Network (ANN) refers to how much the model's predictions change when trained on different datasets. It is a key concept in the **bias-variance tradeoff**, which affects the generalization ability of the model.

Bias	Variance	Model Behavior
High Bias, Low Variance	Underfitting	The model is too simple and cannot learn patterns well.
Balanced Bias & Variance	Good Generalization	The model learns patterns well and generalizes to new data.
Low Bias, High Variance	Overfitting	The model memorizes training data but fails on new data.

What is Variance?

Practical Example

- **High-Variance Model:**

- A deep neural network trained on **a small dataset** without regularization.
- It achieves **99% accuracy on training data** but **only 70% on test data** (overfitting).

- **Balanced Model:**

- A **moderate-sized** network trained with **dropout and L2 regularization** on a **large dataset**.
- It achieves **92% accuracy on training and 90% on test data** (good generalization).

Underfitting v/s Overfitting

- **Underfitting occurs** when the model is too simple to capture the underlying pattern in the training data, leading to poor performance on both the training and test datasets.
- **Overfitting happens** when the model learns the noise and details of the training data too well, causing poor generalization to new data (test/validation set).

Features	Underfitting	Overfitting
Complexity	Too simple	Too complex
Training Error	High	Low
Validation/Test Error	High	High
Generalization	Poor	Poor
Solution	Increase complexity, more training	Reduce complexity, regularization

What is Epoch?

- An epoch in Artificial Neural Networks (ANNs) is one complete pass of the entire training dataset through the network during training. It represents a full cycle of forward and backward propagation where the model updates its weights based on the training data.

Example of Epoch in Action

- Suppose you have **1,000 images** as training data.
- Batch size = **100**
- Each epoch will have **10 iterations** (1,000 ÷ 100).
- Training for **5 epochs** means:
  - The model sees **each image 5 times**.
  - A total of **50 iterations** (5 × 10).

What is Learning Rate?

The **learning rate** (often denoted as **α** or **η**) is a hyperparameter that controls how much the model's weights are updated during training in an optimization algorithm, such as **Stochastic Gradient Descent (SGD)**

During training, the model adjusts its parameters (weights) in response to the gradient of the loss function. The learning rate determines the **step size** for these updates:

$$W_{\text{new}} = W_{\text{old}} - \eta \cdot \nabla L(W)$$

where:

- W_{new} = Updated weights
- W_{old} = Current weights
- η = Learning rate
- $\nabla L(W)$ = Gradient of the loss function

Perceptron

- The first neural network model 'Perceptron', designed in 1958
- Perceptron is a linear binary classifier used for supervised learning.
- Perceptron Learning model is a combination of two concepts,

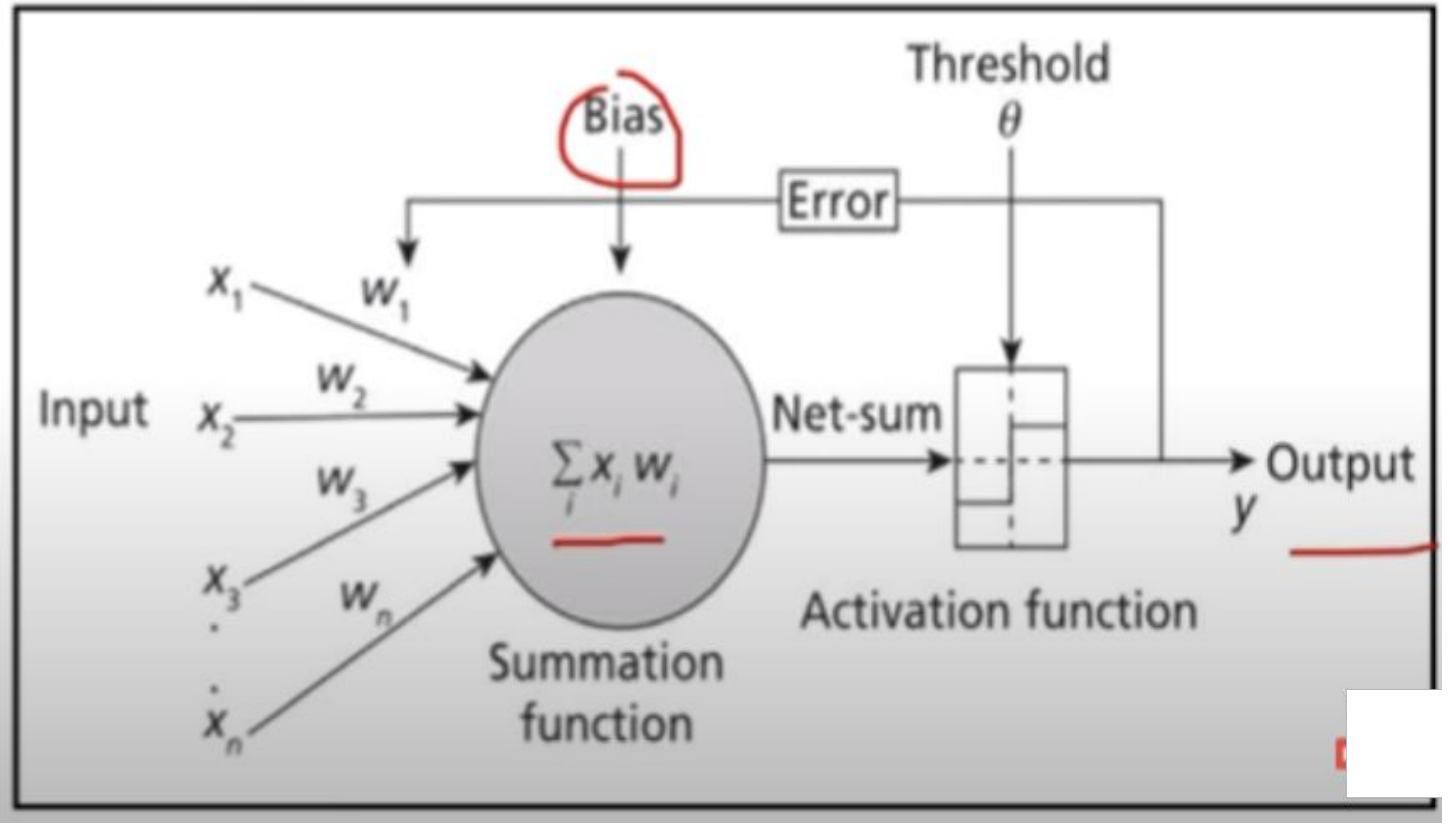
McCulloch-Pitts model of an artificial neuron and **Hebbian learning** rule of adjusting weights.

Perceptron(conti….)

The perceptron model consists of 4 steps:

1. Inputs from other neurons
2. Weights and bias
3. Net sum
4. Activation function

$$Y = \begin{cases} 1 & if\ f(x) \geq \theta \\ 0 & if\ f(x) < \theta \end{cases}$$



Perceptron Learning Algorithm

Set initial weights $w_1, w_2, \ldots, w_n$ and bias θ to a random value in the range [–0.5, 0.5].

For each Epoch,

1. Compute the weighted sum by multiplying the inputs with the weights and add the products.
2. Apply the activation function on the weighted sum:

$$Y = \text{Step}((x_1 w_1 + x_2 w_2) - \theta)$$

3. If the sum is above the threshold value, output the value as positive else output the value as negative.
4. Calculate the error by subtracting the estimated output $Y_{estimated}$ from the desired output $Y_{desired}$:

$$\text{error } e(t) = Y_{desired} - Y_{estimated}$$

Perceptron Learning Algorithm

5. Update the weights if there is an error:

$$\Delta w_i = \alpha \times e(t) \times x_i$$

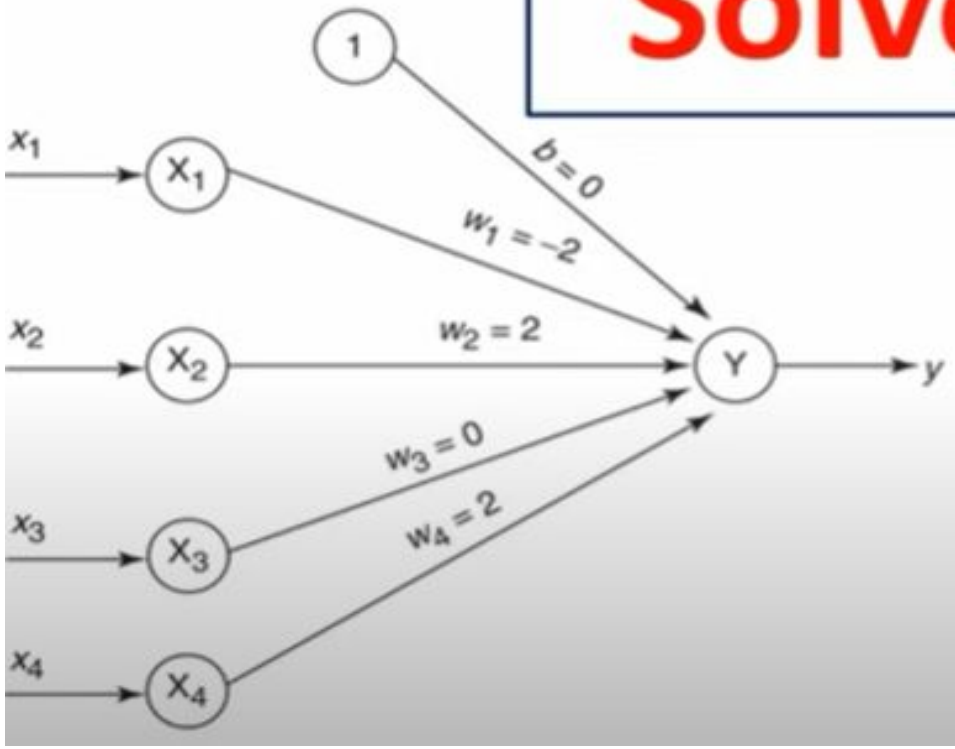
$$w_i = w_i + \Delta w_i$$

where, x_i is the input value, $e(t)$ is the error at step t, α is the learning rate and Δw_i is the difference in weight that has to be added to w_i.

Perceptron Learning Algorithm

Perceptron Learning Rule

Solved Example



Input					Target
x_1	x_2	x_3	x_4	b	(t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Perceptron Example

- Find the weights required to perform the following classification using perceptron network.
- The vectors (1, 1, 1, 1) and (- 1, 1 - 1, - 1) are belonging to the class 1, vectors (1, 1, 1, - 1) and (1, - 1, - 1, 1) are belonging to the class -1.
- Assume learning rate as 1
- and Initial weights as 0.

Input					Target
x_1	x_2	x_3	x_4	b	(t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1

Perceptron Example

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta w_3 = \alpha t x_3;$$
$$\Delta w_4 = \alpha t x_4;$$
$$\Delta b = \alpha t$$

Inputs				Target	Net input	output	Weight changes					Weights				
$(x_1$	x_2	x_3	x_4	(t)	(y_{in})	(y)	$(\Delta w_1$	Δw_2	Δw_3	Δw_4	$\Delta b)$	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-1																
(1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
(-1	1	-1	-1	1	-1	-1	-1	1	-1	-1	1	0	2	0	0	2
(1	1	1	-1	-1	4	1	-1	-1	-1	1	-1	-1	1	-1	1	1
(1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-2	2	0	0	0

Perceptron Example

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta w_3 = \alpha t x_3;$$
$$\Delta w_4 = \alpha t x_4;$$
$$\Delta b = \alpha t$$

Inputs				Target	Net input	output	Weight changes					Weights				
$(x_1$	x_2	x_3	x_4	(t)	(y_{in})	(y)	$(\Delta w_1$	Δw_2	Δw_3	Δw_4	$\Delta b)$	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-1																
(1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
(-1	1	-1	-1	1	-1	-1	-1	1	-1	-1	1	0	2	0	0	2
(1	1	1	-1	-1	4	1	-1	-1	-1	1	-1	-1	1	-1	1	1
(1	-1	-1	1	-1	1	1	-1	1	1	-1	-1	-2	2	0	0	0
EPOCH-2																
(1	1	1	1	1	0	0	1	1	1	1	1	-1	3	1	1	1
(-1	1	-1	-1	1	3	1	0	0	0	0	0	-1	3	1	1	1
(1	1	1	-1	-1	4	1	-1	-1	-1	1	-1	-2	2	0	2	0
(1	-1	-1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0

Perceptron Example

$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$

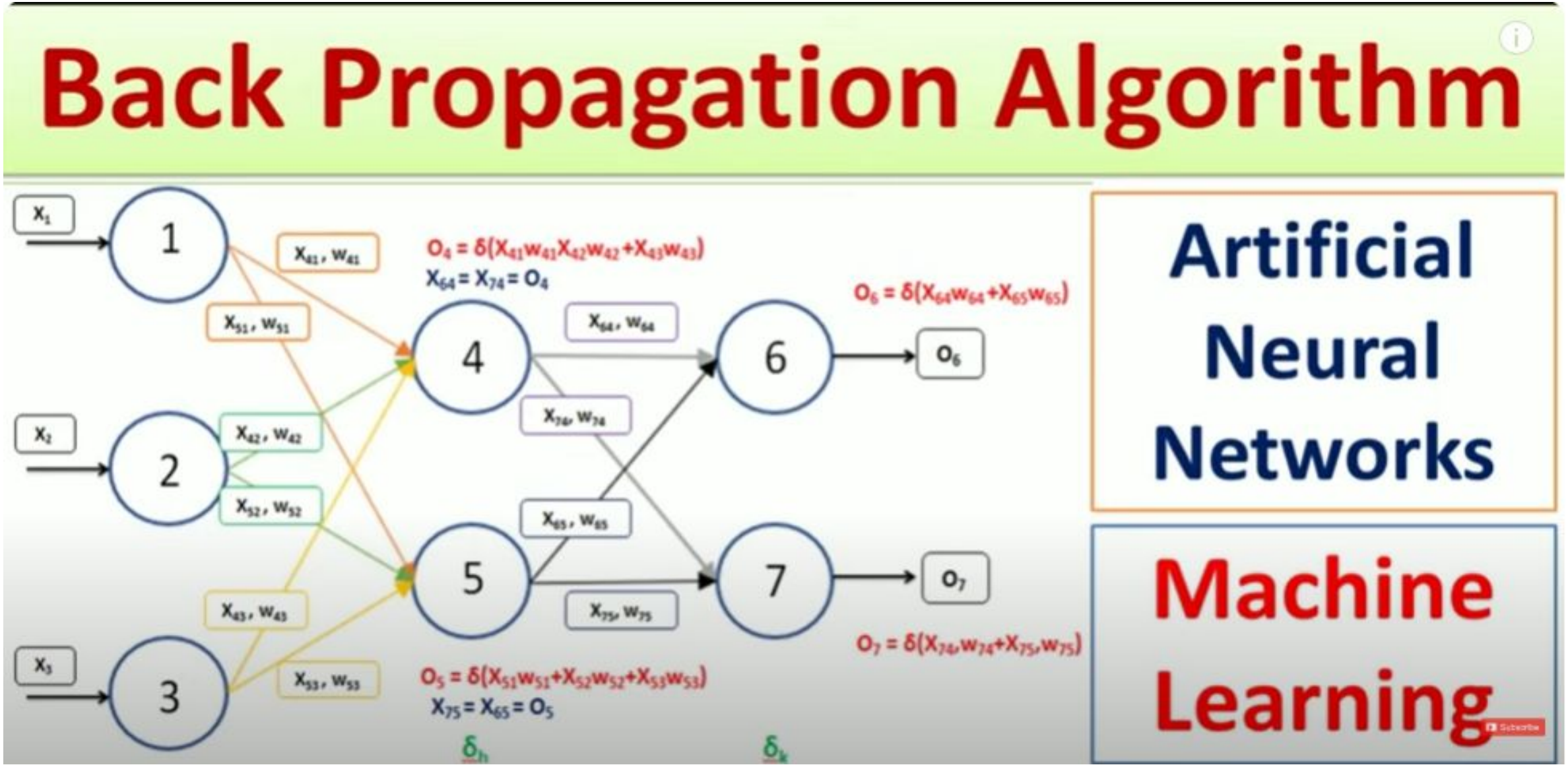
$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

$$\Delta w_1 = \alpha t x_1;$$
$$\Delta w_2 = \alpha t x_2;$$
$$\Delta w_3 = \alpha t x_3;$$
$$\Delta w_4 = \alpha t x_4;$$
$$\Delta b = \alpha t$$

EPOCH-3

1	1	1	1	1	2	1	0	0	0	0	0	-2	2	0	2	0
-1	1	-1	-1	1	2	1	0	0	0	0	0	-2	2	0	2	0
1	1	1	-1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0
1	-1	-1	1	-1	-2	-1	0	0	0	0	0	-2	2	0	2	0

Backpropagation in ANN



Backpropagation in ANN

BACKPROPAGATION (training_example, η, n_{in}, n_{out}, n_{hidden})

- Each training example is a pair of the form (x, t), where (x) is the vector of network input values, and (t) is the vector of target network output values.
- η is the learning rate (e.g., 0.05).
- n_i, is the number of network inputs,
- n_{hidden} the number of units in the hidden layer, and
- n_{out} the number of output units.
- The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}

Backpropagation in ANN

- Create a feed-forward network with n_i inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until the termination condition is met, Do
  - For each (x, t), in training examples, Do
    - Propagate the input forward through the network:

      1. Input the instance x, to the network and compute the output o_u of every unit u in the network.

    - Propagate the errors backward through the network

      2. For each network unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

      3. For each network unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

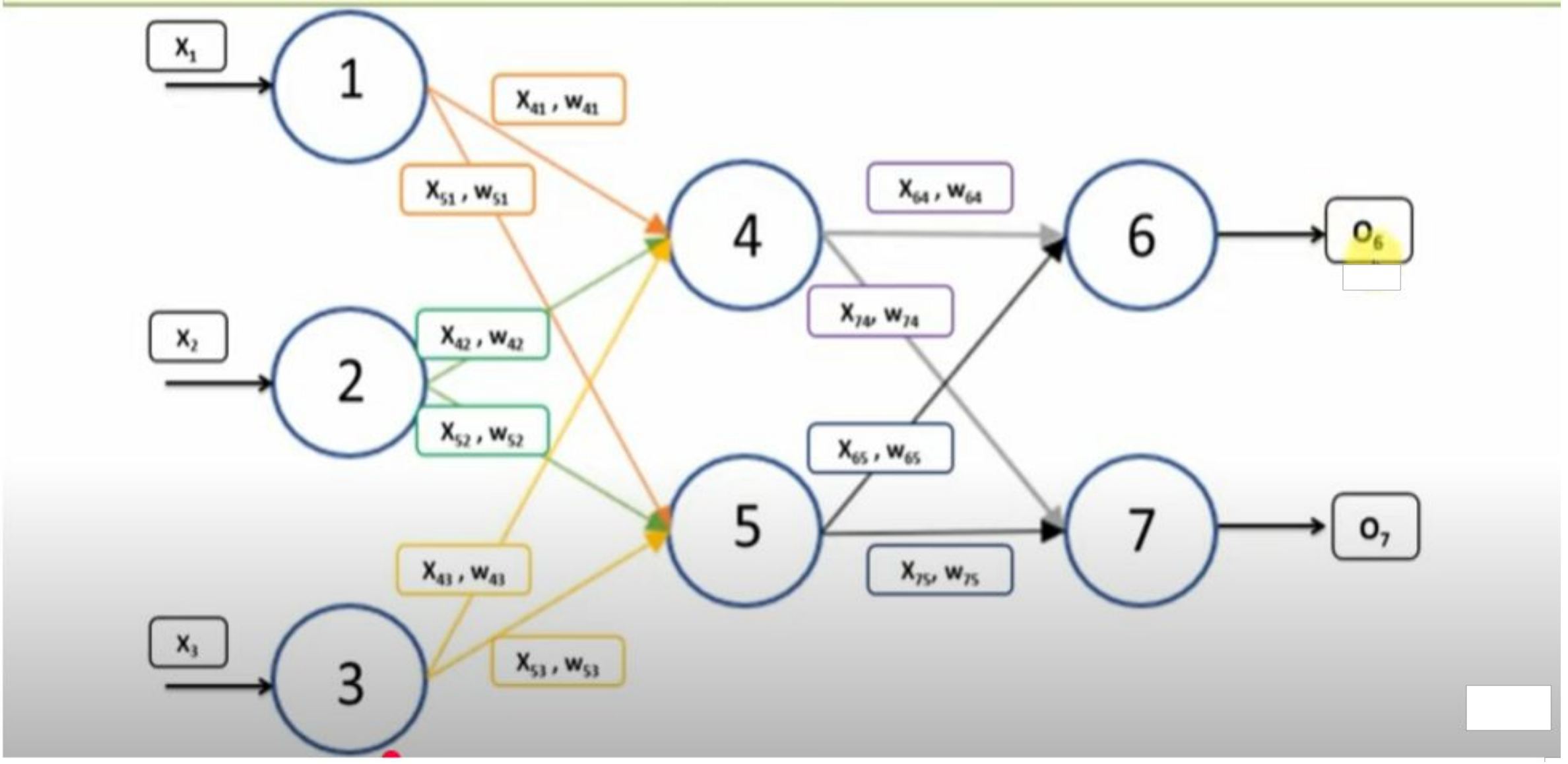
4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

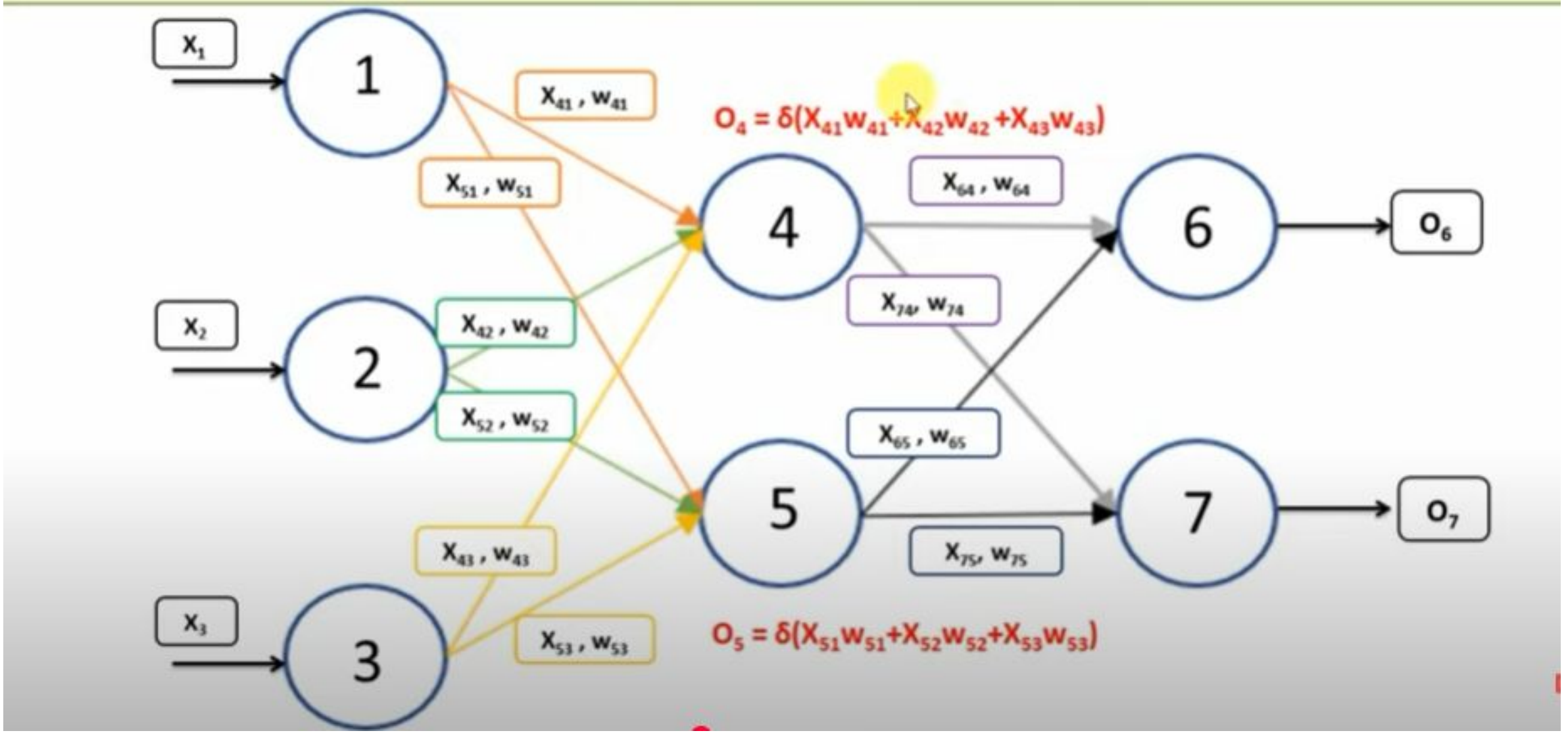
Where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

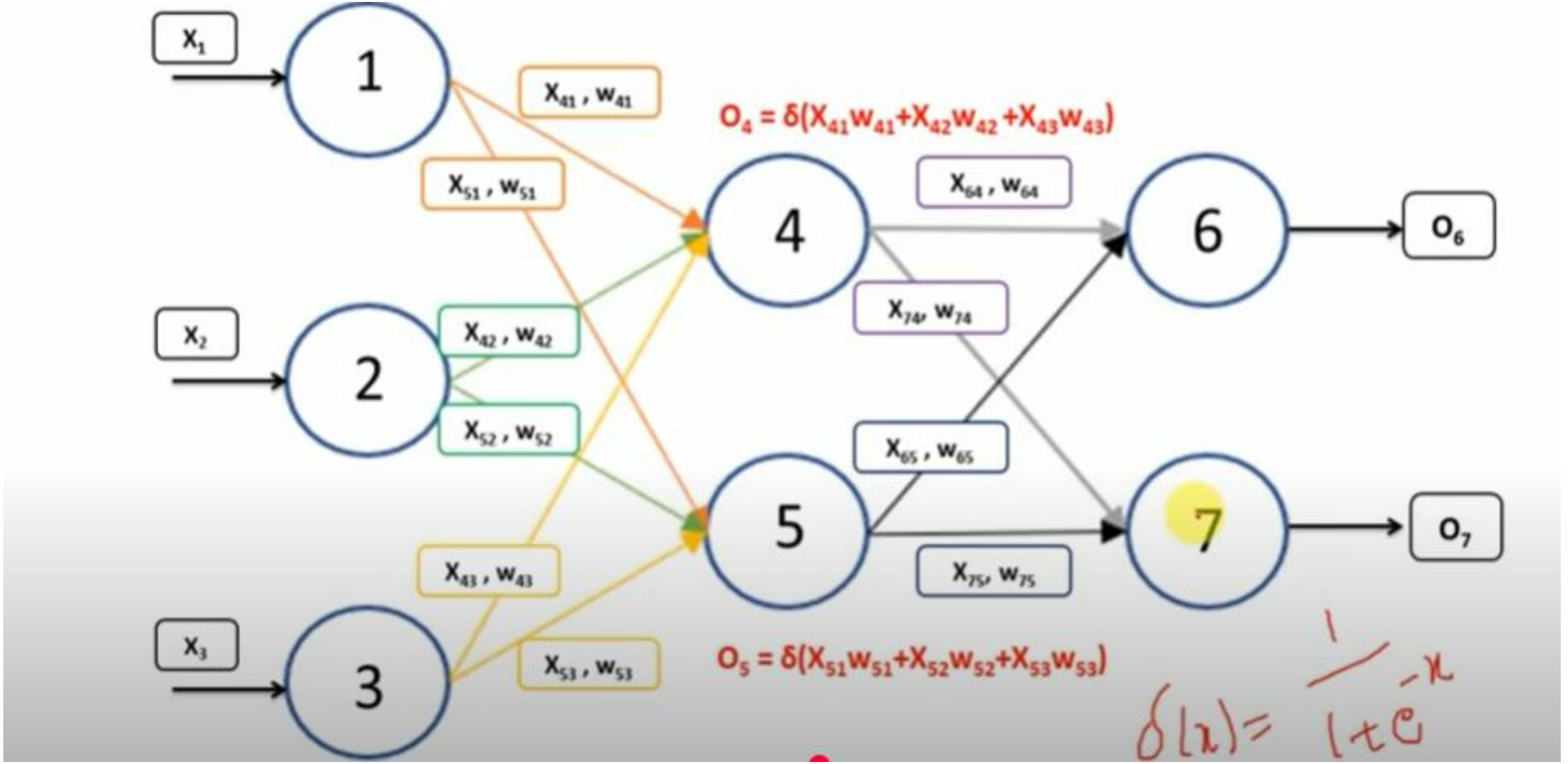
Backpropagation in ANN



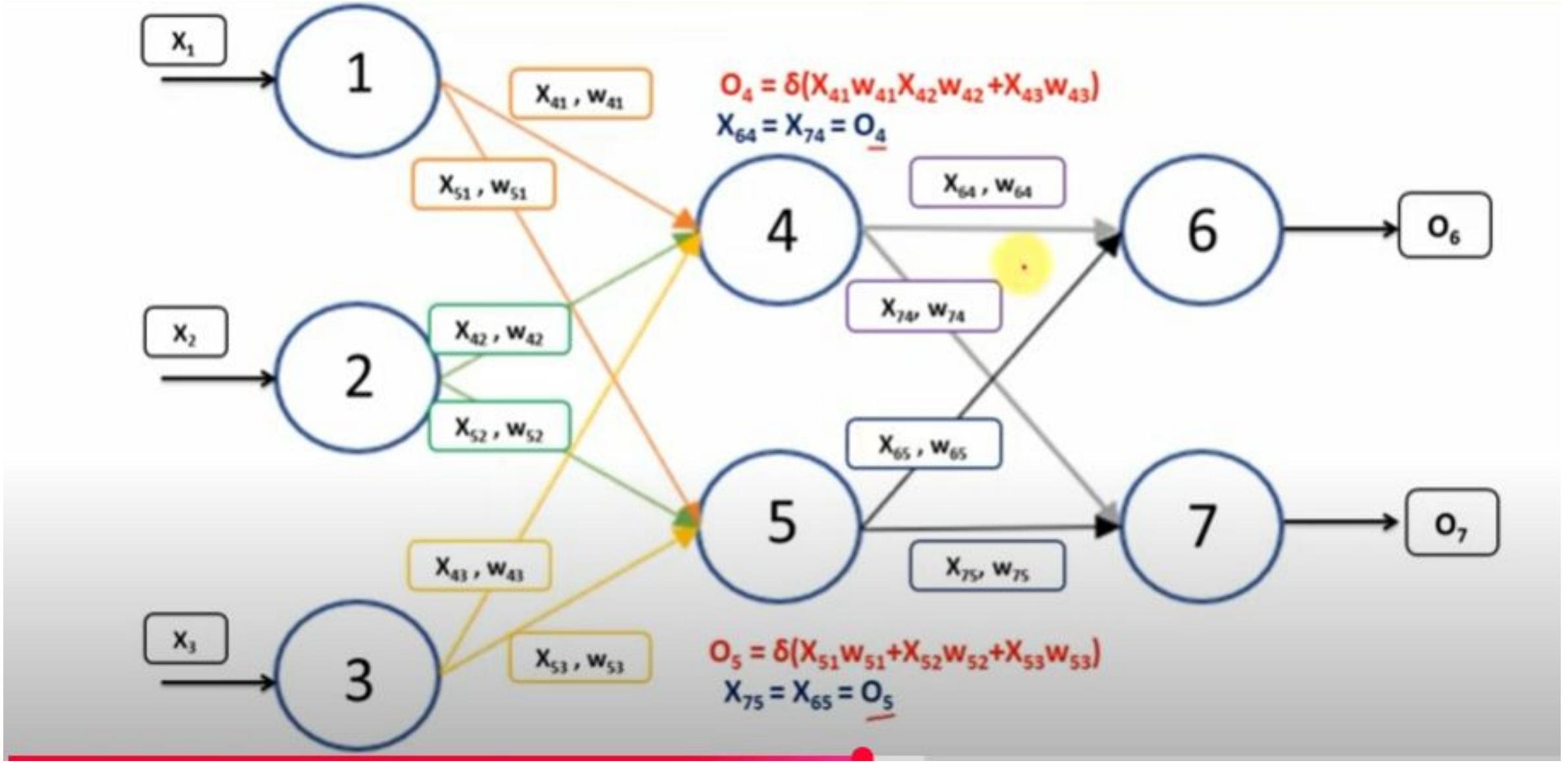
Backpropagation in ANN



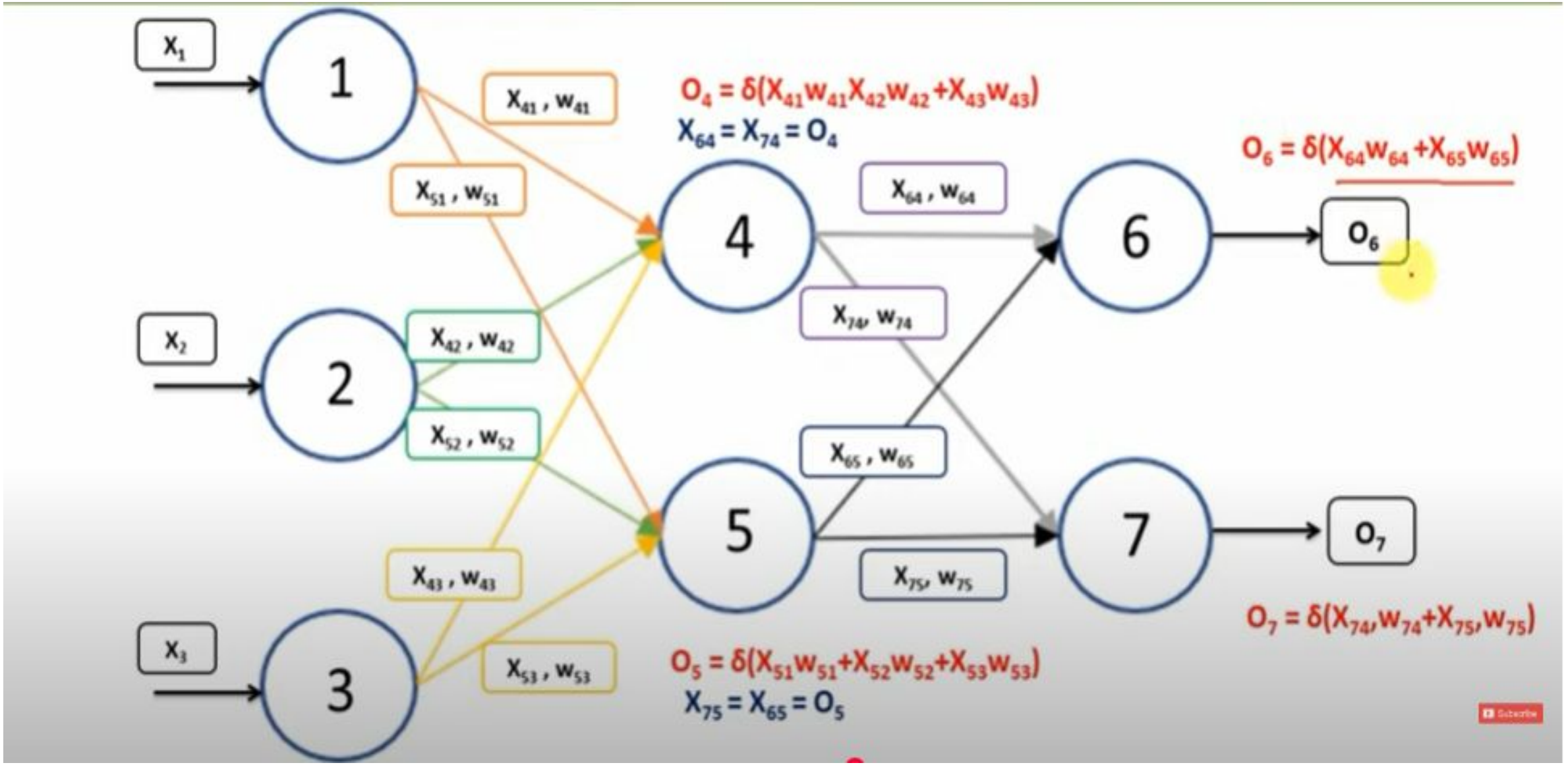
Backpropagation in ANN



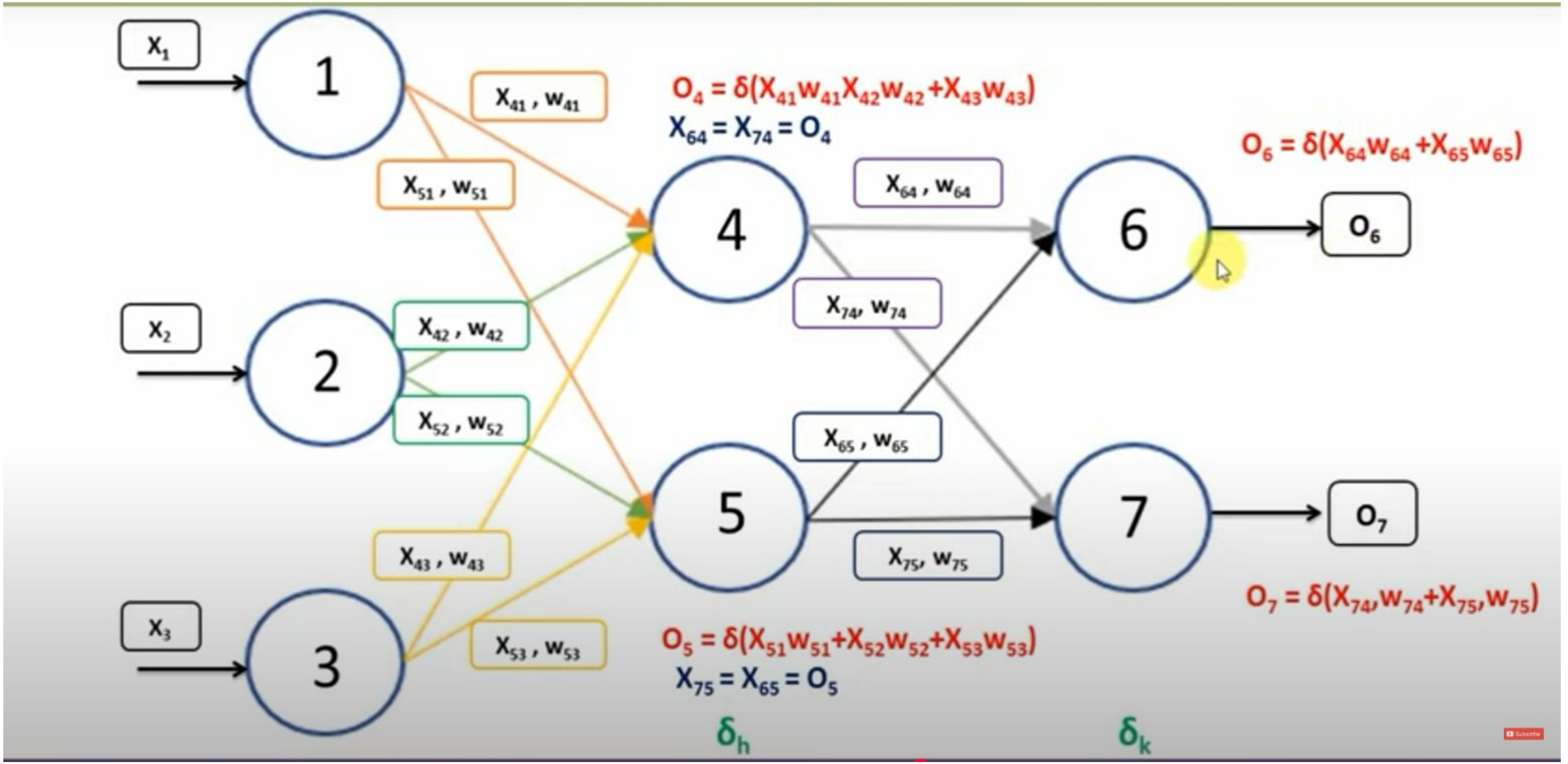
Backpropagation in ANN



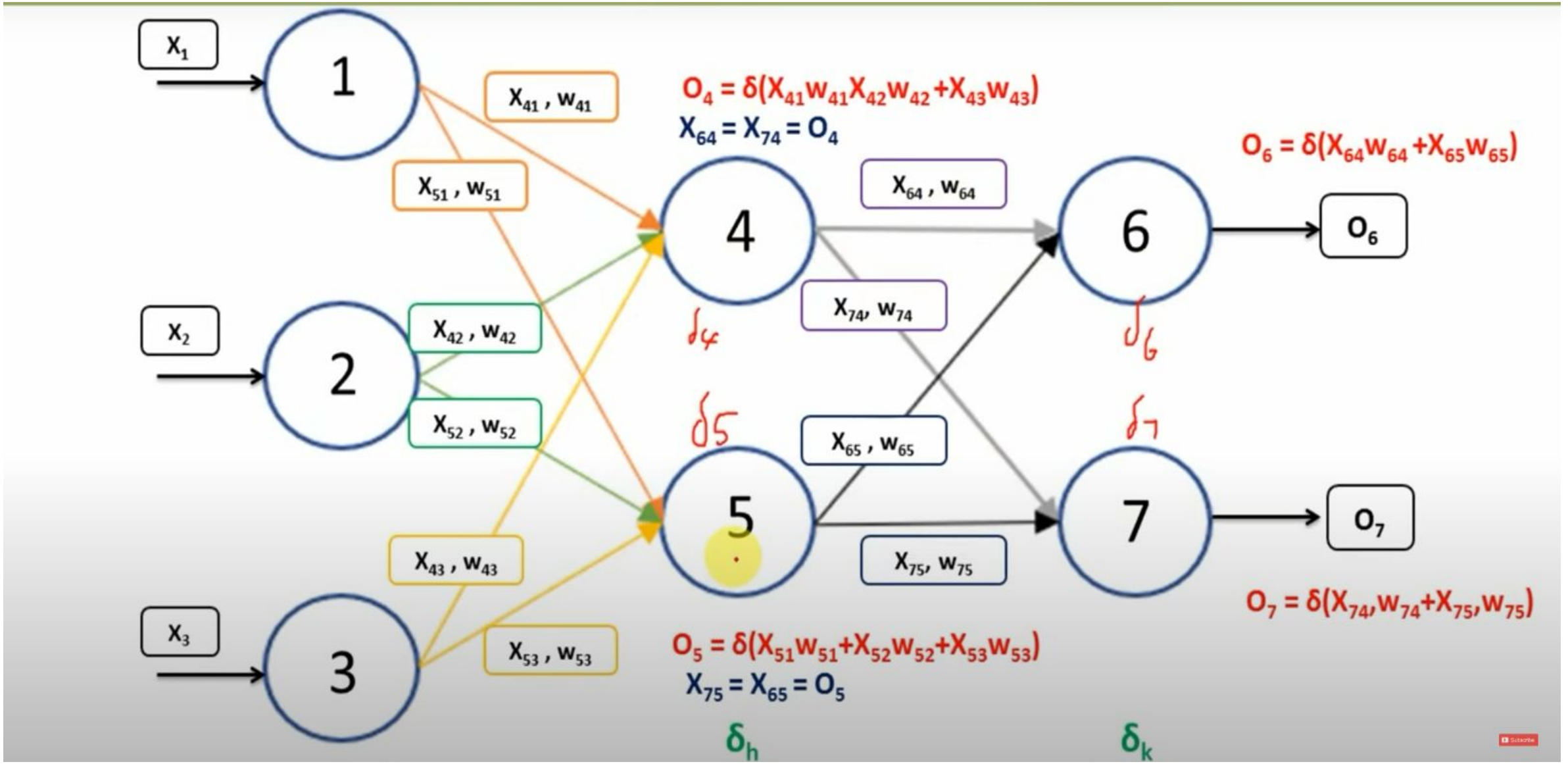
Backpropagation in ANN



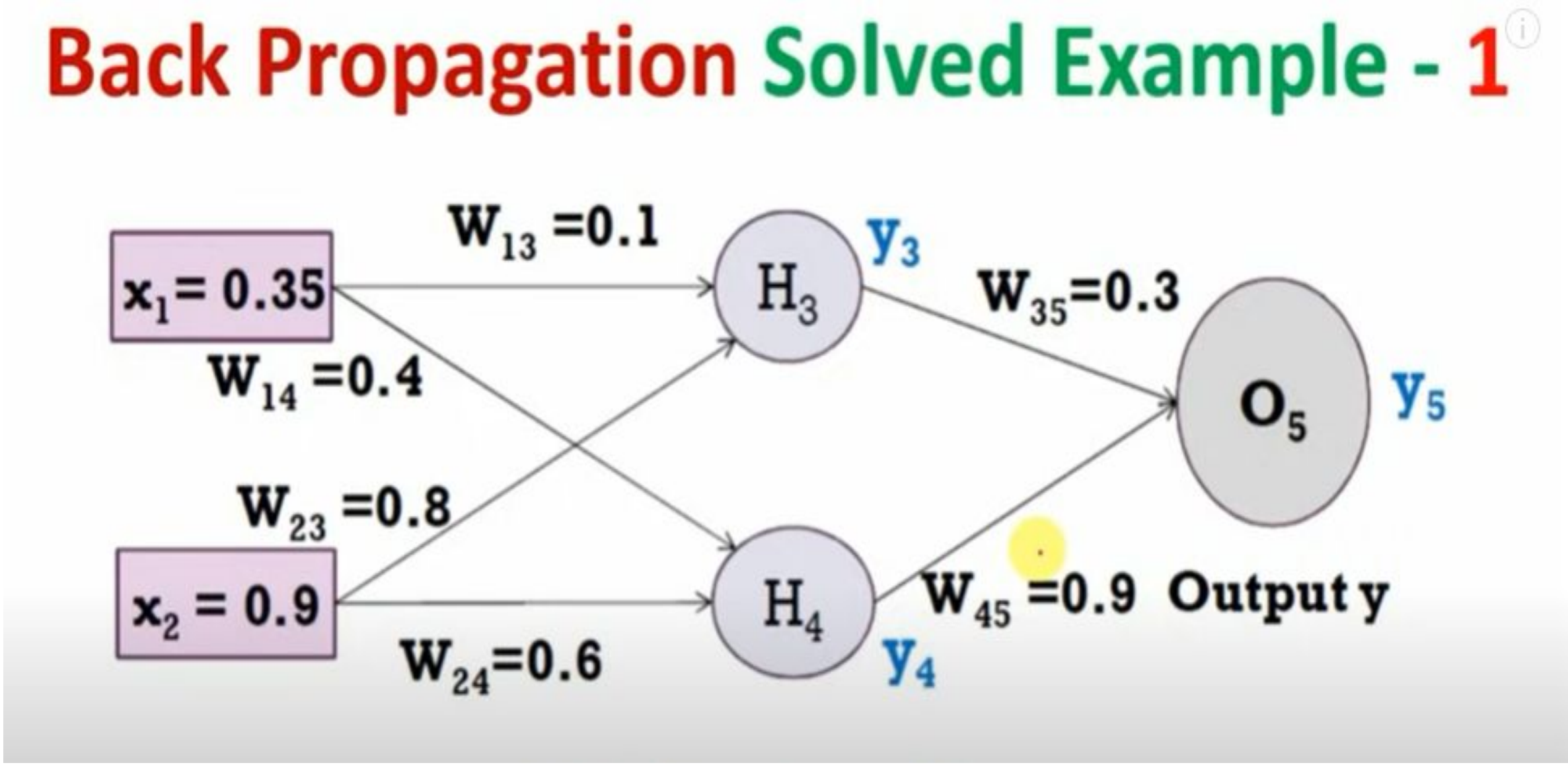
Backpropagation in ANN



Backpropagation in ANN

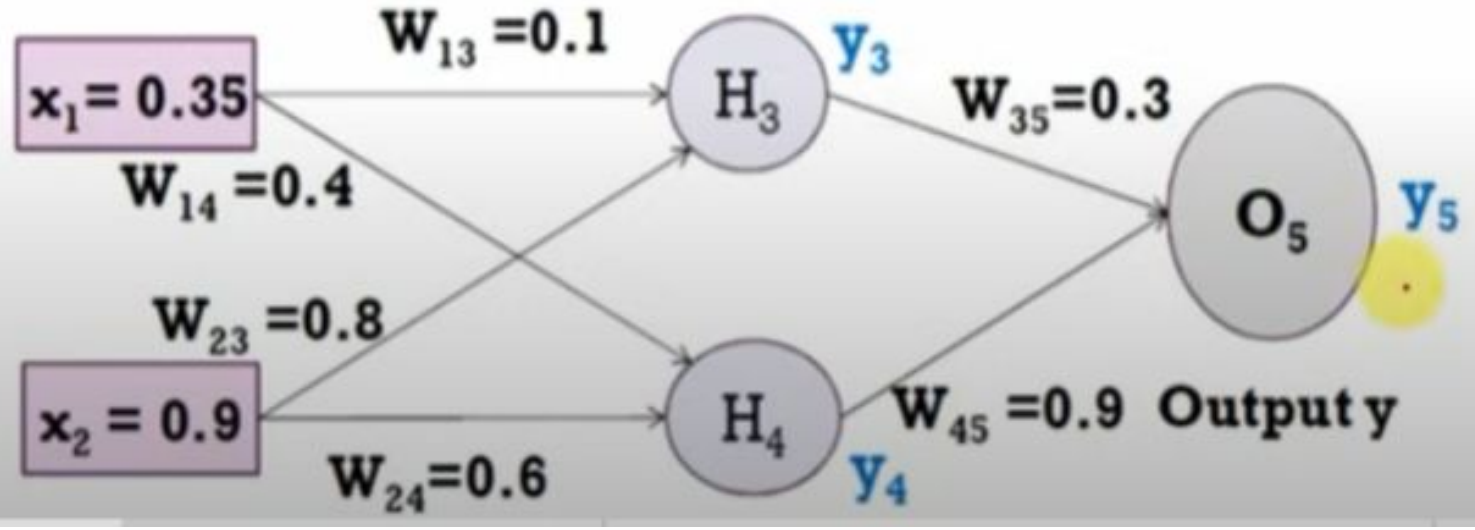


Backpropagation in ANN

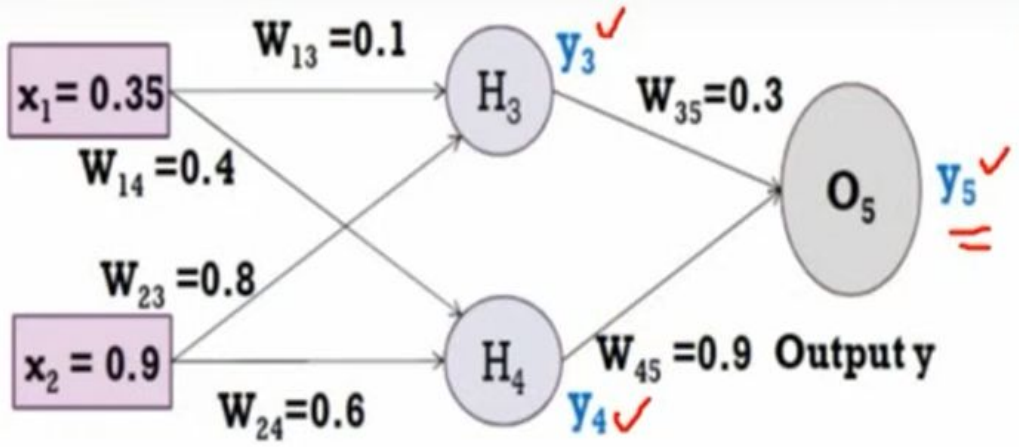


Backpropagation in ANN

- Assume that the neurons have a sigmoid activation function, perform a forward pass and a backward pass on the network. Assume that the actual output of y is 0.5 and learning rate is 1. Perform another forward pass.



Backpropagation in ANN



- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$ ✓

$= (0.1 * 0.35) + (0.8 * 0.9) = 0.755$

$y_3 = f(a_1) = 1/(1 + e^{-0.755}) = 0.68$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$ ✓

$= (0.4 * 0.35) + (0.6 * 0.9) = 0.68$

$y_4 = f(a_2) = 1/(1 + e^{-0.68}) = 0.6637$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$ ✓

$= (0.3 * 0.68) + (0.9 * 0.6637) = 0.801$ ✓

$y_5 = f(a_3) = 1/(1 + e^{-0.801}) = \mathbf{0.69}$ (Network Output)

$\text{Error} = y_{target} - y_5 = -0.19$

0.5 - 0.69

Backpropagation in ANN

- Each weight changed by:

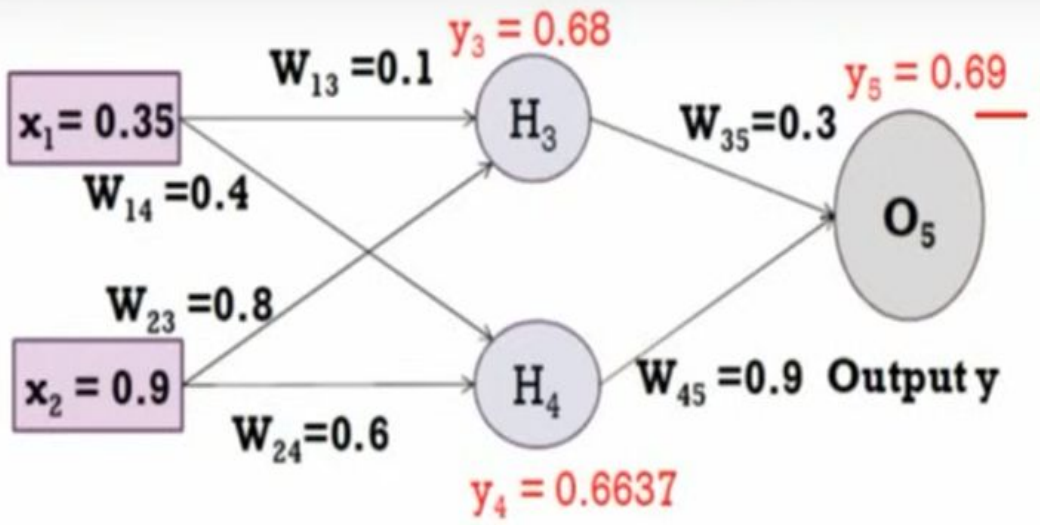
$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- where η is a constant called the learning rate
- tj is the correct teacher output for unit j
- δj is the error measure for unit j

Backpropagation in ANN



Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

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

$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$

For hidden unit:

$\delta_3 = y_3(1-y_3)\, w_{35} * \delta_5$

$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$

$$\Delta w_{ji} = \eta \delta_j o_i$$

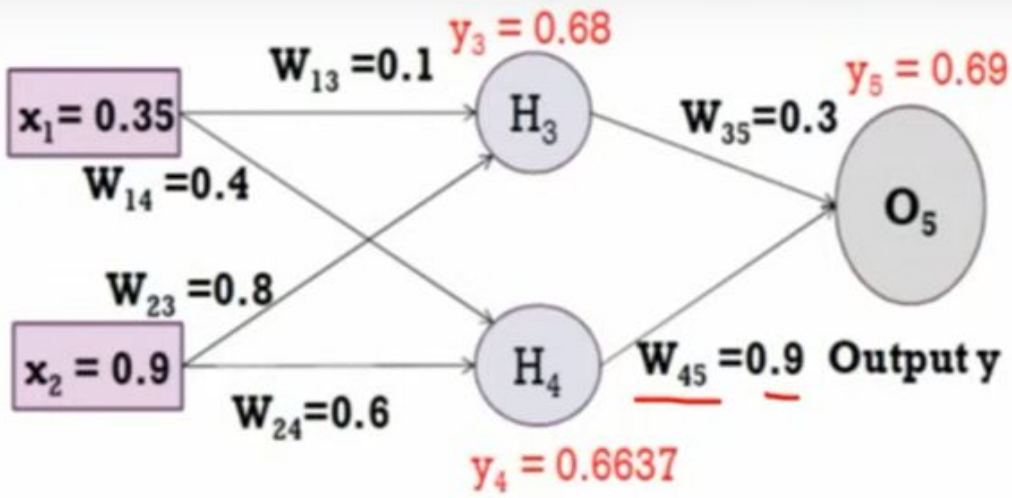
$$\delta_j = o_j(1-o_j)(t_j - o_j) \quad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \quad \text{if } j \text{ is a hidden unit}$$

$\delta_4 = y_4(1-y_4)w_{45} * \delta_5$

$= 0.6637*(1 - 0.6637)*(0.9 * -0.0406) = -0.0082$

Backpropagation in ANN



- Backward Pass: Compute δ3, δ4 and δ5.

For output unit:

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

$= 0.69*(1-0.69)*(0.5-0.69) = -0.0406$

For hidden unit:

$\delta_3 = y_3(1-y_3)\, w_{35} * \delta_5$

$= 0.68*(1 - 0.68)*(0.3 * -0.0406) = -0.00265$

$\delta_4 = y_4(1-y_4) w_{45} * \delta_5$

$= 0.6637*(1 - 0.6637)* (0.9 * -0.0406) = -0.0082$

Compute new weights

$$\Delta w_{ji} = \eta \delta_j o_i$$

$\Delta w_{45} = \eta\, \delta_5\, y_4 = 1 * -0.0406 * 0.6637 = -0.0269$

w_{45} (new) $= \Delta w_{45} + w_{45}$(old) $= -0.0269 + (0.9) = 0.8731$

$\Delta w_{14} = \eta\, \delta_4\, x_1 = 1 * -0.0082 * 0.35 = -0.00287$

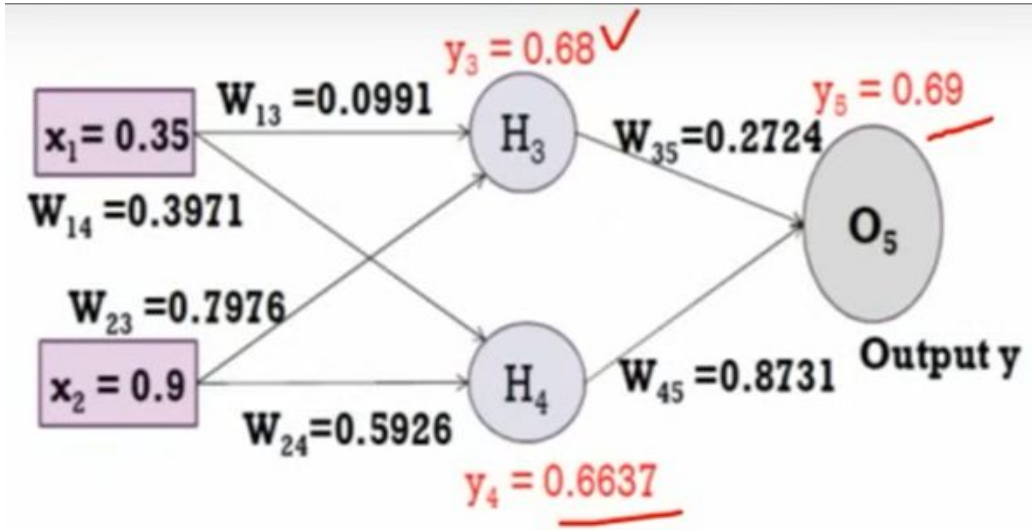
w_{14} (new) $= \Delta w_{14} + w_{14}$(old) $= -0.00287 + 0.4 = 0.3971$

Backpropagation in ANN

- Similarly, update all other weights

i	j	w_{ij}	δ_j	x_i	η	Updated w_{ij}
1	3	0.1	-0.00265	0.35	1	0.0991
2	3	0.8	-0.00265	0.9	1	0.7976
1	4	0.4	-0.0082	0.35	1	0.3971
2	4	0.6	-0.0082	0.9	1	0.5926
3	5	0.3	-0.0406	0.68	1	0.2724
4	5	0.9	-0.0406	0.6637	1	0.8731

Backpropagation in ANN



- Forward Pass: Compute output for y3, y4 and y5.

$$a_j = \sum_j (w_{i,j} * x_i) \qquad yj = F(aj) = \frac{1}{1 + e^{-a_j}}$$

$a_1 = (w_{13} * x_1) + (w_{23} * x_2)$
$= (0.0991 * 0.35) + (0.7976 * 0.9) = 0.7525$
$y_3 = f(a_1) = 1/(1 + e^{-0.7525}) = 0.6797$

$a_2 = (w_{14} * x_1) + (w_{24} * x_2)$
$= (0.3971 * 0.35) + (0.5926 * 0.9) = 0.6723$
$y_4 = f(a_2) = 1/(1 + e^{-0.6723}) = 0.6620$

$a_3 = (w_{35} * y_3) + (w_{45} * y_4)$
$= (0.2724 * 0.6797) + (0.8731 * 0.6620) = 0.7631$
$y_5 = f(a_3) = 1/(1 + e^{-0.7631}) = \mathbf{0.6820}$ (Network Output)

$\text{Error} = y_{target} - y_5 = -0.182$