Module 3
Decision Trees

Dr. Mahesh G Huddar

Dept. of Computer Science and Engineering

Decision Trees

- *Decision Trees* is one of the most widely used Classification Algorithm
- **Features**
  - Method for approximating *discrete-valued* functions (including boolean)
  - Learned functions are represented as *decision trees* (or *if-then-else* rules*)*
  - Expressive hypotheses space, including disjunction
  - Robust to noisy data

Decision Tree for Boolean Functions

(a) $A \wedge \neg B$

(b) $A \vee [B \wedge C]$

(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$

Decision Tree for Boolean Functions

- Every Variable in Boolean function such as A, B, C etc. has two possibilities that is True and False
- Every Boolean function is either True or False
- If the Boolean function is true we write YES (Y)
- If the Boolean function is False we write NO (N)

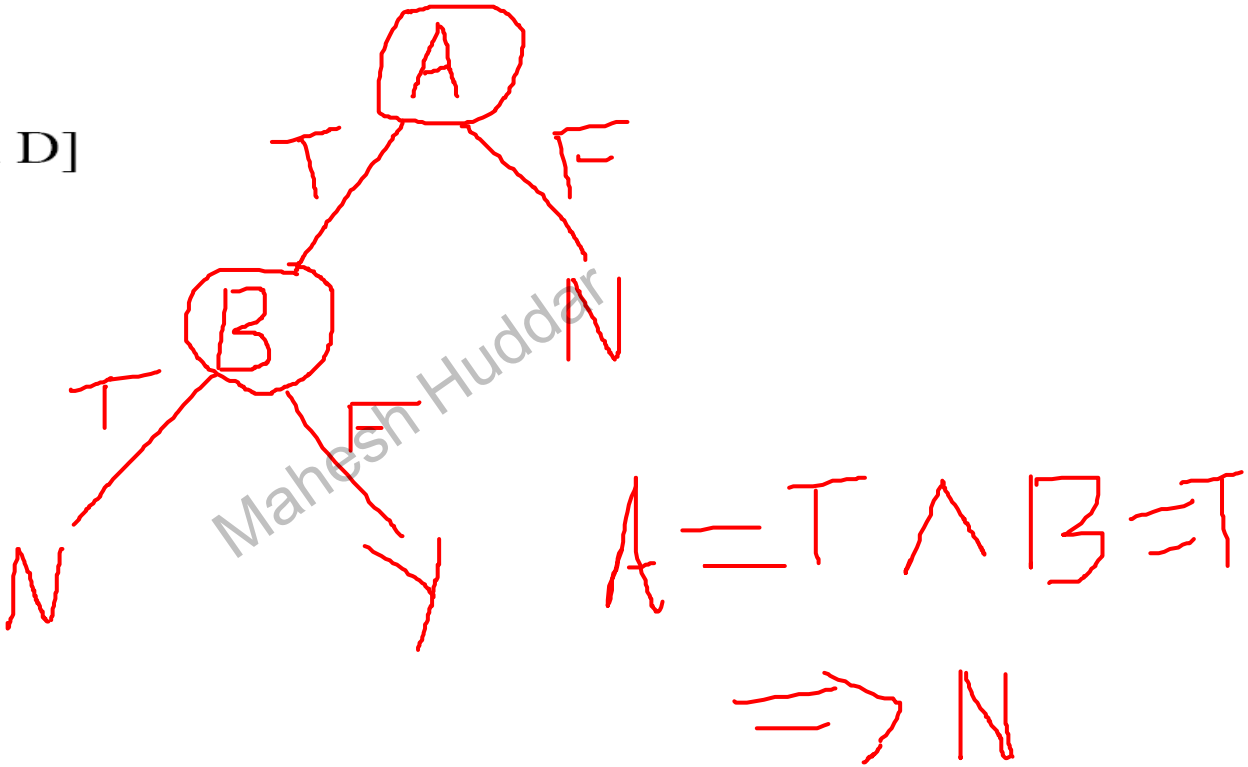
3.1 Give decision trees to represent the following boolean functions:

(a) $A \wedge \neg B$

(b) $A \vee [B \wedge C]$

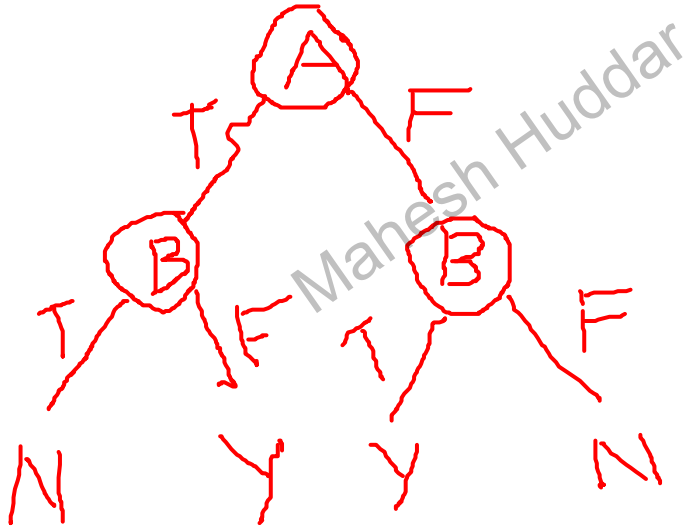
(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$



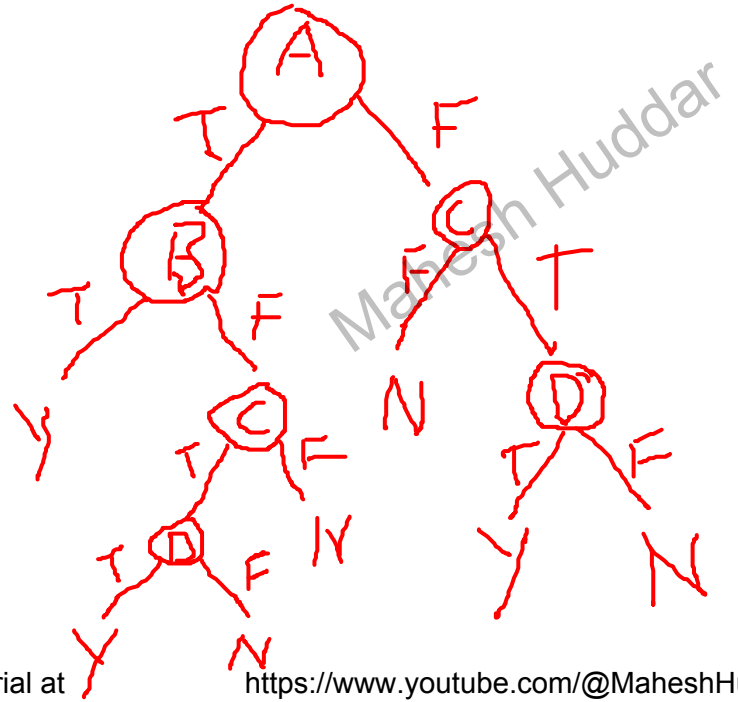
Decision Tree for Boolean Functions

(c) A XOR B = $(A \wedge \neg B) \vee (\neg A \wedge B)$

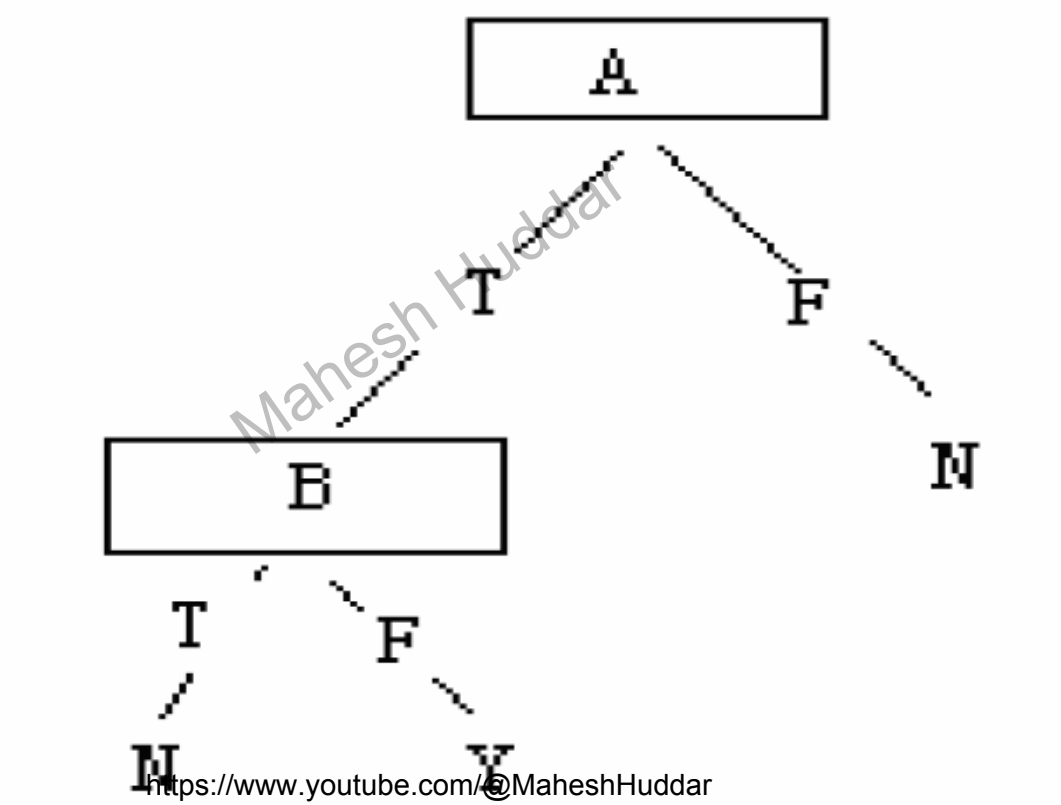


Decision Tree for Boolean Functions

(d) $[A \wedge B] \vee [C \wedge D]$



(a) $A \wedge \neg B$



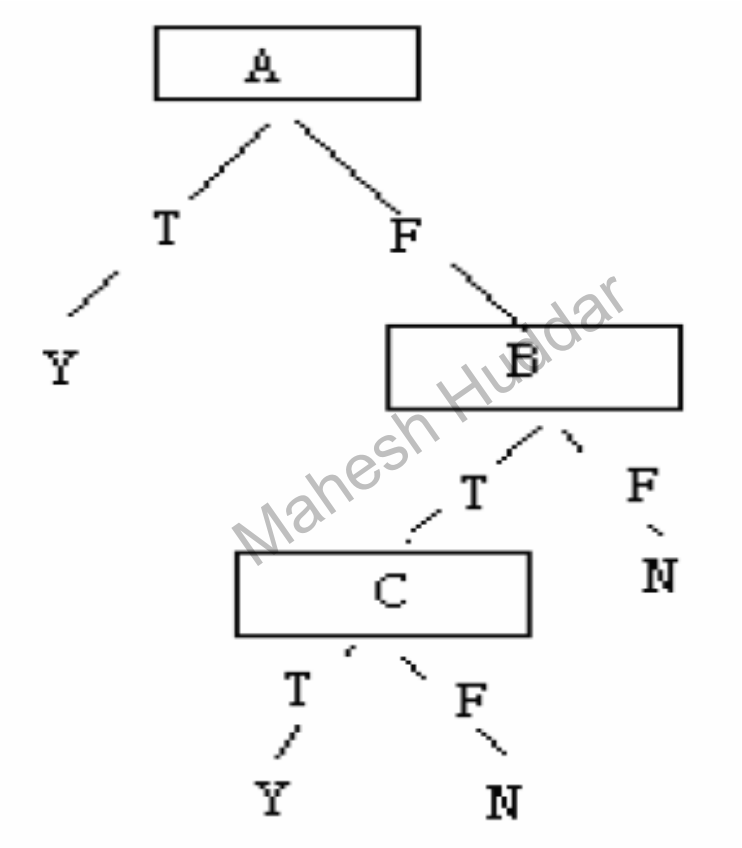
3.1 Give decision trees to represent the following boolean functions:

(a) $A \wedge \neg B$

(b) $A \vee [B \wedge C]$

(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$



3.1 Give decision trees to represent the following boolean functions:

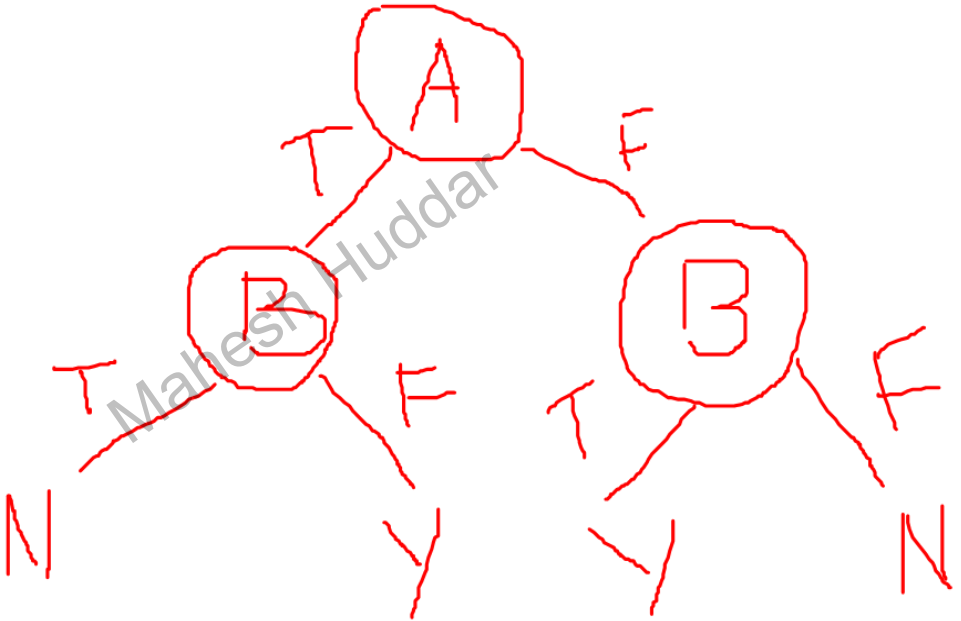
(a) $A \wedge \neg B$

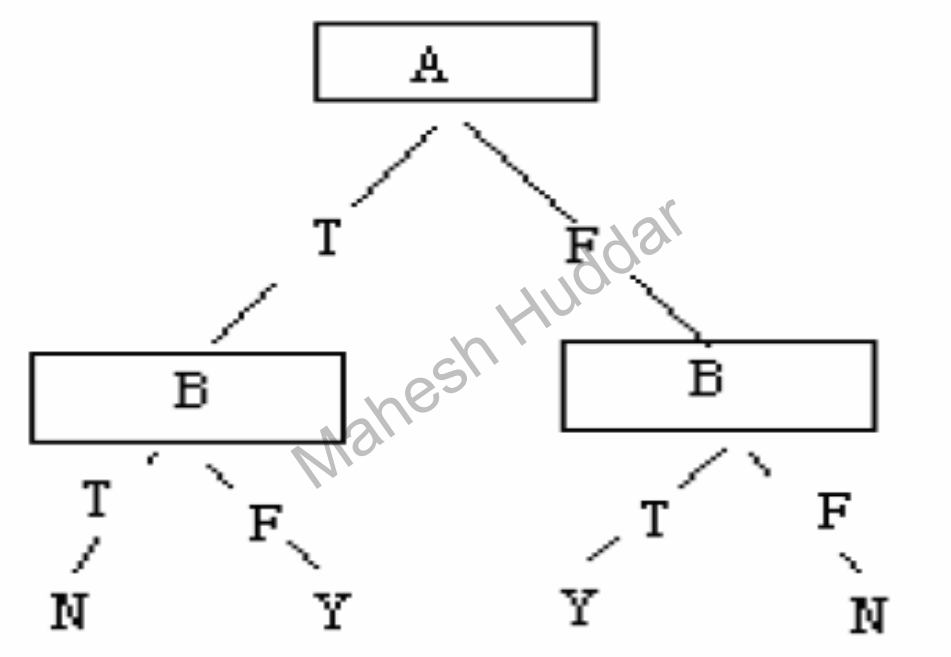
(b) $A \vee [B \wedge C]$

(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$

$(A \wedge \neg B) \vee (\neg A \wedge B)$





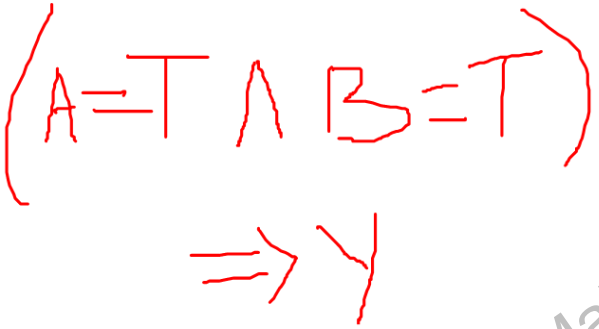
3.1 Give decision trees to represent the following boolean functions:

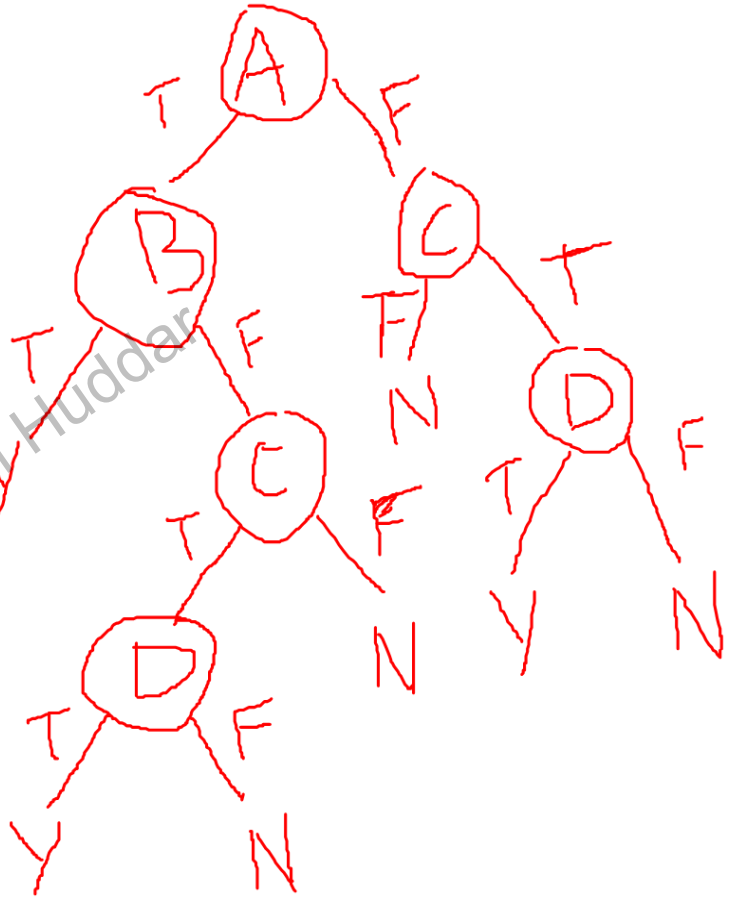
(a) $A \wedge \neg B$

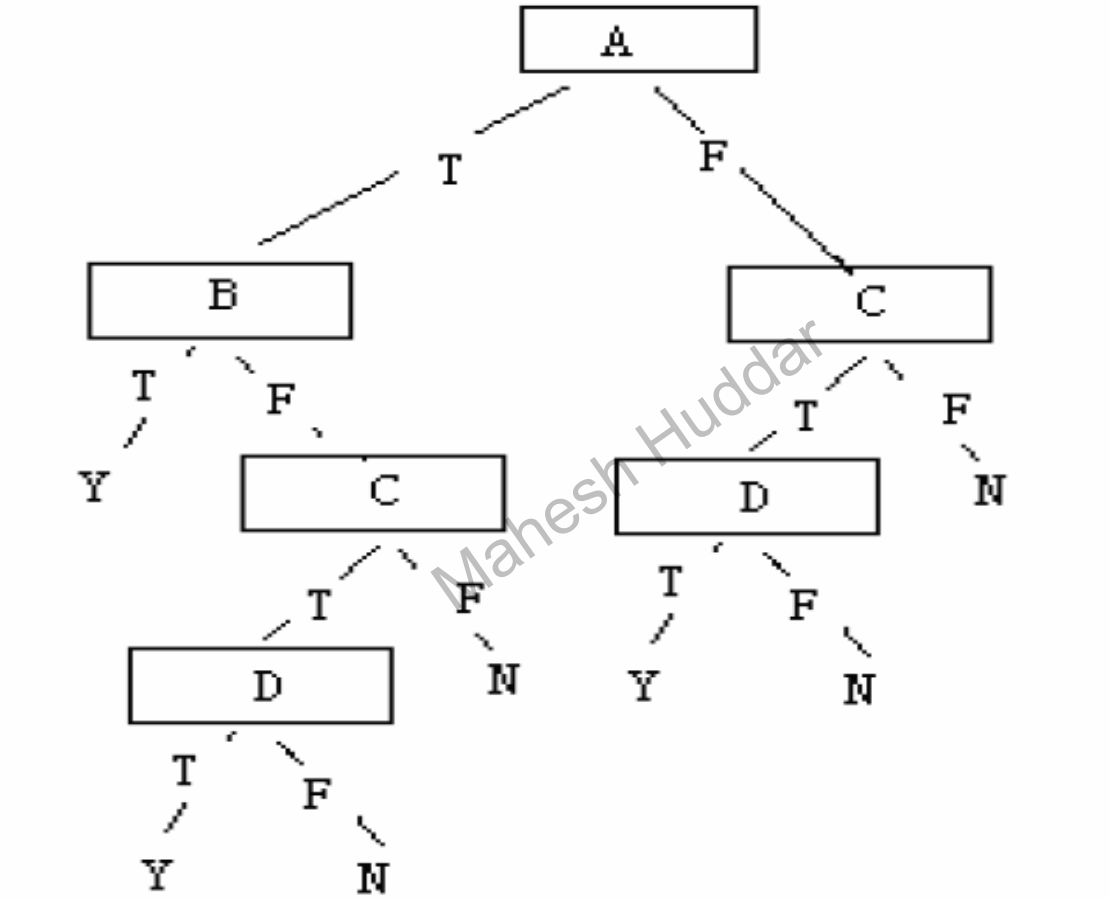
(b) $A \vee [B \wedge C]$

(c) A XOR B

(d) $[A \wedge B] \vee [C \wedge D]$



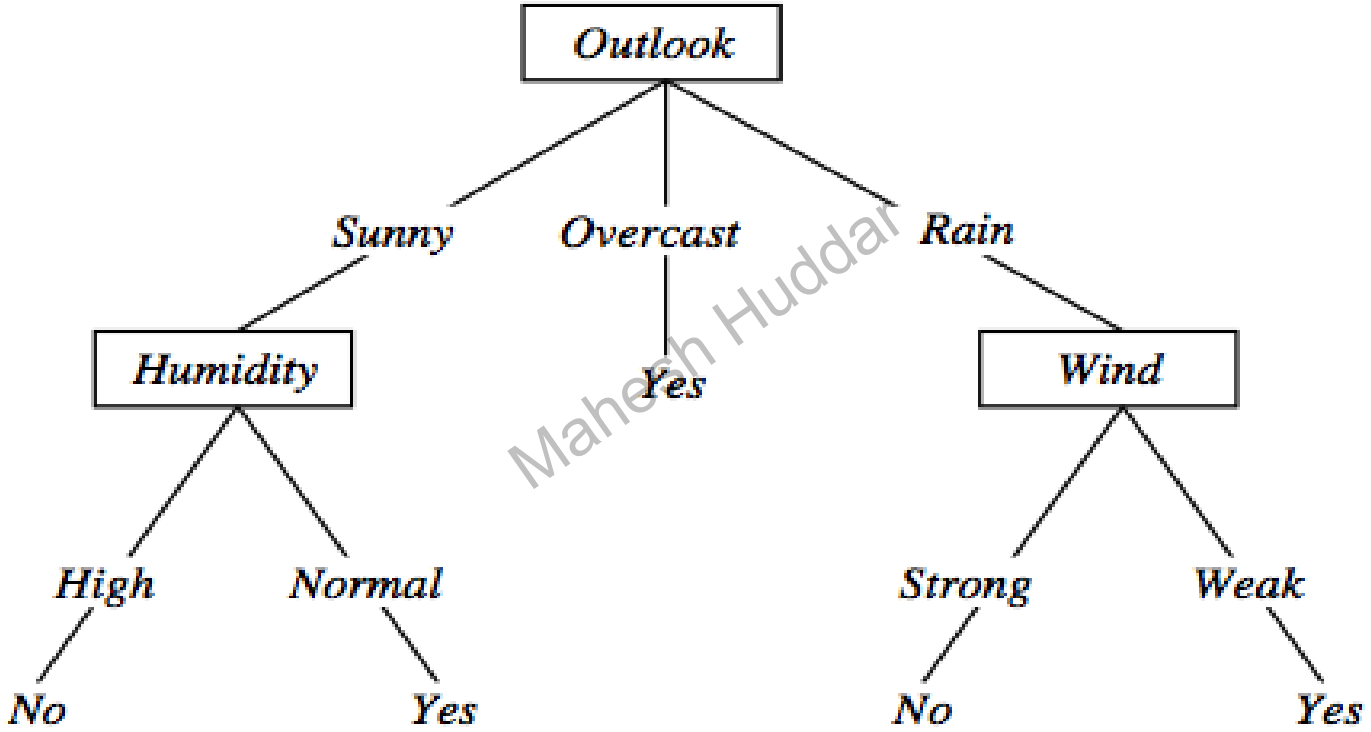




Example

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree Representation (PlayTennis)



⟨*Outlook=Sunny, Temp=Hot, Humidity=High, Wind=Strong*⟩ No

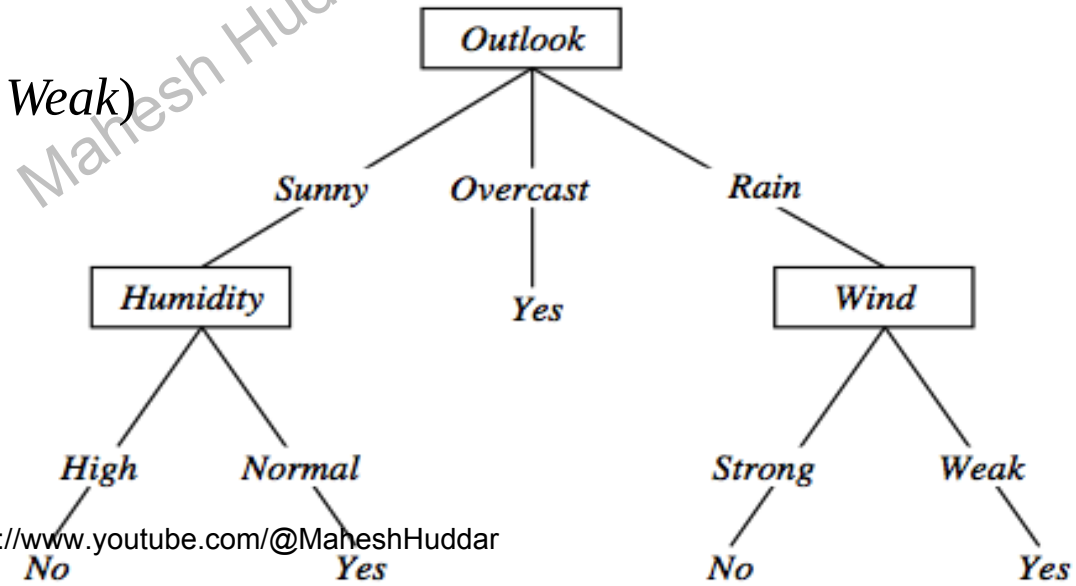
Decision trees expressivity

- Decision trees represent a disjunction of conjunctions on constraints on the value of attributes:

$(Outlook = Sunny \wedge Humidity = Normal) \vee$

$(Outlook = Overcast) \vee$

$(Outlook = Rain \wedge Wind = Weak)$



Decision tree representation (PlayTennis)

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example.
- This process is then repeated for the subtree rooted at the new node.

Decision tree representation (PlayTennis)

- In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee \quad (Outlook = Overcast)$$
$$\vee \quad (Outlook = Rain \wedge Wind = Weak)$$

Example

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Although a variety of decision tree learning methods have been developed with somewhat differing capabilities and requirements, decision tree learning is generally best suited to problems with the following characteristics:

1. ***Instances are represented by attribute-value pairs.*** Instances are described by a fixed set of attributes (e.g., ***Temperature)*** and their values (e.g., ***Hot).*** The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., ***Hot, Mild, Cold).*** However, extensions to the basic algorithm allow handling real-valued attributes as well (e.g., representing ***Temperature*** numerically).

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

2. ***The target function has discrete output values.*** The decision tree is usually used for Boolean classification (e.g., ***yes*** or ***no)*** kind of example. Decision tree methods easily extend to learning functions with more than two possible output values. A more substantial extension allows learning target functions with real-valued outputs, though the application of decision trees in this setting is less common.

3. ***Disjunctive descriptions may be required.*** Decision trees naturally represent disjunctive expressions.

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

4. ***The training data may contain errors.*** Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
5. ***The training data may contain missing attribute values.*** Decision tree methods can be used even when some training examples have unknown values (e.g., if the ***Humidity*** of the day is known for only some of the training examples).

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

- Many practical problems have been found to fit these characteristics.
- Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease, equipment malfunctions by their cause, and loan applicants by their likelihood of defaulting on payments.
- Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as ***classification problems.***

THE BASIC DECISION TREE LEARNING ALGORITHM

- Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees.
- This approach is exemplified by the ID3 algorithm (Quinlan 1986) and its successor C4.5 (Quinlan 1993), which form the primary focus of our discussion here.
- The basic algorithm for decision tree learning, corresponding approximately to the ID3 algorithm.
- Next, we consider a number of extensions to this basic algorithm, including extensions incorporated into C4.5 and other more recent algorithms for decision tree learning.

Example

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- We would like to select the attribute that is most useful for classifying examples.
- What is a good quantitative measure of the worth of an attribute? We will define a statistical property, called ***information gain,*** that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree.

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

- ***Entropy,*** characterizes the (im)purity of an arbitrary collection of examples.
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this boolean classification is

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

- where p+, is the proportion of positive examples in S and p-, is the proportion of negative examples in S.
- In all calculations involving entropy we define 0 log 0 to be 0.

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

- Entropy measures the (*im)purity* of a collection of examples. It depends from the distribution of the random variable *p*.
  - *S* is a collection of training examples
  - p_+ the proportion of positive examples in *S*
  - p_- the proportion of negative examples in *S*

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

Examples

$Entropy\ (S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \quad [0 \log_2 0 = 0]$

$Entropy\ ([14+, 0-]) = -14/14 \log_2 (14/14) - 0 \log_2 (0) = 0$

$Entropy\ ([9+, 5-]) = -9/14 \log_2 (9/14) - 5/14 \log_2 (5/14) = 0{,}94$

$Entropy\ ([7+, 7-]) = -7/14 \log_2 (7/14) - 7/14 \log_2 (7/14) =$

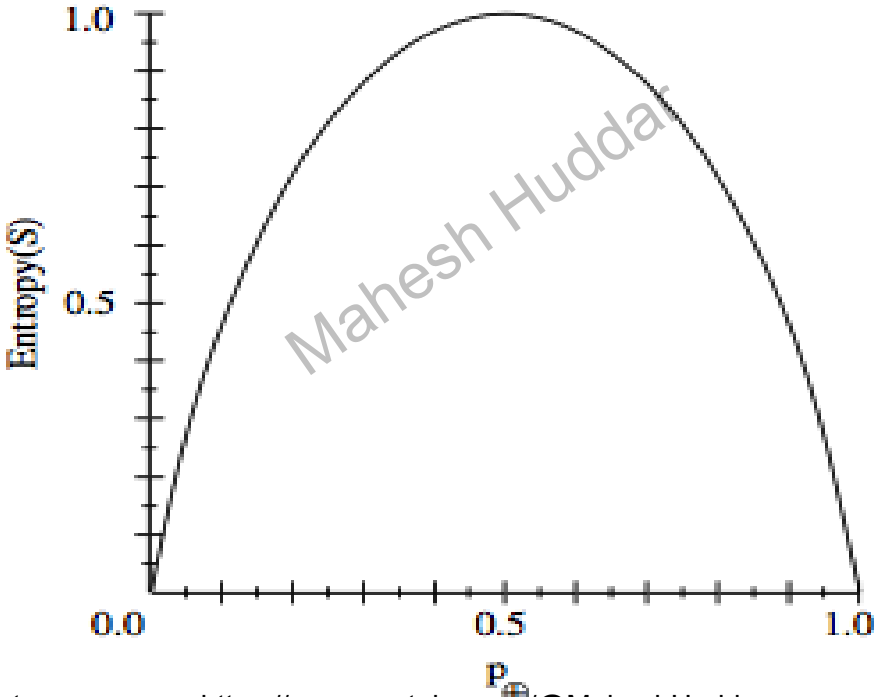
$= 1/2 + 1/2 = 1$

Example

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

Entropy



CONSTRUCTING DECISION TREE – ID3 ALGORITHM

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data.
- Now, the ***information gain,*** is simply the expected reduction in entropy caused by partitioning the examples according to this attribute.
- More precisely, the information gain, ***Gain(S, A)*** of ***an*** attribute **A,** relative to a collection of examples ***S,*** is defined as,

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- where ***Values(A)*** is the set of all possible values for attribute A, and ***S,*** is the subset of S for which attribute A has value v (i.e., $S_v = \{s \in S | A(s) = v\}$)

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

- For example, suppose ***S*** is a collection of training-example days described by attributes including ***Wind,*** which can have the values ***Weak*** or ***Strong.***

$$Values(Wind) = Weak, Strong$$

$$S = [9+, 5-]$$

$$S_{Weak} \leftarrow [6+, 2-]$$

$$S_{Strong} \leftarrow [3+, 3-]$$

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - (8/14) Entropy(S_{Weak}) - (6/14) Entropy(S_{Strong})$$

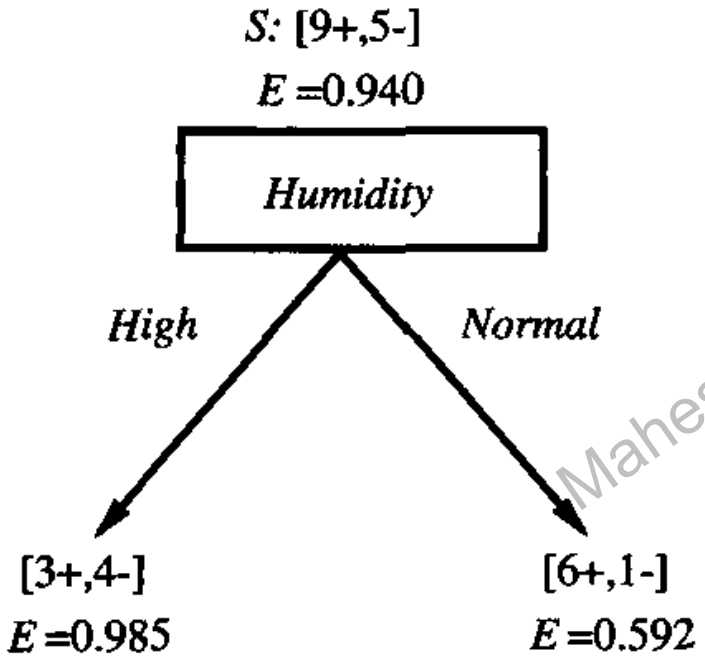
$$= 0.940 - (8/14)0.811 - (6/14)1.00$$

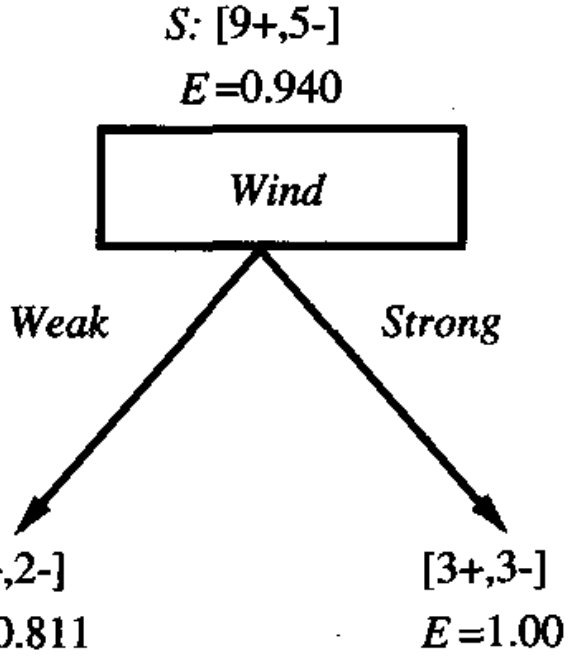
$$= 0.048$$

CONSTRUCTING DECISION TREE – ID3 ALGORITHM

- Information gain is precisely the measure used by ID3 to select the best attribute at
- each step in growing the tree.
- The use of information gain to evaluate the relevance of attributes.
- Here the information gain of two different attributes, ***Humidity*** and ***Wind,*** is computed in order to determine which is the better attribute for classifying the training examples.

CONSTRUCTING DECISION TREE – ID3 ALGORITHM





Gain (S, Humidity)

= .940 - (7/14).985 - (7/14).592

= .151

Gain (S, Wind)

= .940 - (8/14).811 - (6/14)1.0

= .048

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
    - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
    - The decision attribute for *Root* $\leftarrow A$
    - For each possible value, v_i, of A,
        - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
        - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
        - If $Examples_{v_i}$ is empty
            - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
            - Else below this new branch add the subtree
              ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {A}))
- End
- Return *Root*

DECISION TREE – ID3 ALGORITHM NUMERICAL EXAMPLE

Day	Outlook	Temp	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Outlook

$$Values\,(Outlook) = Sunny, Overcast, Rain$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14} = 0.94$$

$$S_{Sunny} \leftarrow [2+, 3-] \qquad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.971$$

$$S_{Overcast} \leftarrow [4+, 0-] \qquad Entropy(S_{Overcast}) = -\frac{4}{4}log_2\frac{4}{4} - \frac{0}{4}log_2\frac{0}{4} = 0$$

$$S_{Rain} \leftarrow [3+, 2-] \qquad Entropy(S_{Rain}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.971$$

$$Gain\,(S, Outlook) = Entropy(S) - \sum_{v \in \{Sunny, Overcast, Rain\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook)$$

$$= Entropy(S) - \frac{5}{14}Entropy(S_{Sunny}) - \frac{4}{14}Entropy(S_{Overcast}) - \frac{5}{14}Entropy(S_{Rain})$$

$$Gain(S, Outlook) = 0.94 - \frac{5}{14}0.971 - \frac{4}{14}\,0 - \frac{5}{14}0.971 = 0.2464$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Temp

$$Values\ (Temp) = Hot, Mild, Cool$$

$$S = [9+, 5-]$$

$$Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$$

$$S_{Hot} \leftarrow [2+, 2-]$$

$$Entropy(S_{Hot}) = -\frac{2}{4} log_2 \frac{2}{4} - \frac{2}{4} log_2 \frac{2}{4} = 1.0$$

$$S_{Mild} \leftarrow [4+, 2-]$$

$$Entropy(S_{Mild}) = -\frac{4}{6} log_2 \frac{4}{6} - \frac{2}{6} log_2 \frac{2}{6} = 0.9183$$

$$S_{Cool} \leftarrow [3+, 1-]$$

$$Entropy(S_{Cool}) = -\frac{3}{4} log_2 \frac{3}{4} - \frac{1}{4} log_2 \frac{1}{4} = 0.8113$$

$$Gain\ (S, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Temp)$$

$$= Entropy(S) - \frac{4}{14} Entropy(S_{Hot}) - \frac{6}{14} Entropy(S_{Mild}) - \frac{4}{14} Entropy(S_{Cool})$$

$$Gain(S, Temp) = 0.94 - \frac{4}{14} 1.0 - \frac{6}{14} 0.9183 - \frac{4}{14} 0.8113 = 0.0289$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Humidity

$$Values\ (Humidity) = High, Normal$$

$$S = [9+, 5-] \qquad Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$$

$$S_{High} \leftarrow [3+, 4-] \qquad Entropy(S_{High}) = -\frac{3}{7} log_2 \frac{3}{7} - \frac{4}{7} log_2 \frac{4}{7} = 0.9852$$

$$S_{Normal} \leftarrow [6+, 1-] \qquad Entropy(S_{Normal}) = -\frac{6}{7} log_2 \frac{6}{7} - \frac{1}{7} log_2 \frac{1}{7} = 0.5916$$

$$Gain\ (S, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Humidity)$$

$$= Entropy(S) - \frac{7}{14} Entropy(S_{High}) - \frac{7}{14} Entropy(S_{Normal})$$

$$Gain(S, Humidity) = 0.94 - \frac{7}{14} 0.9852 - \frac{7}{14}\ 0.5916 = 0.1516$$

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Attribute: Wind

$$Values\ (Wind) = Strong, Weak$$

$$S = [9+, 5-]$$

$$Entropy(S) = -\frac{9}{14} log_2 \frac{9}{14} - \frac{5}{14} log_2 \frac{5}{14} = 0.94$$

$$S_{Strong} \leftarrow [3+, 3-]$$

$$Entropy(S_{Strong}) = 1.0$$

$$S_{Weak} \leftarrow [6+, 2-]$$

$$Entropy(S_{Weak}) = -\frac{6}{8} log_2 \frac{6}{8} - \frac{2}{8} log_2 \frac{2}{8} = 0.8113$$

$$Gain\ (S, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Wind) = Entropy(S) - \frac{6}{14} Entropy(S_{Strong}) - \frac{8}{14} Entropy(S_{Weak})$$

$$Gain(S, Wind) = 0.94 - \frac{6}{14}\ 1.0 - \frac{8}{14}\ 0.8113 = 0.0478$$

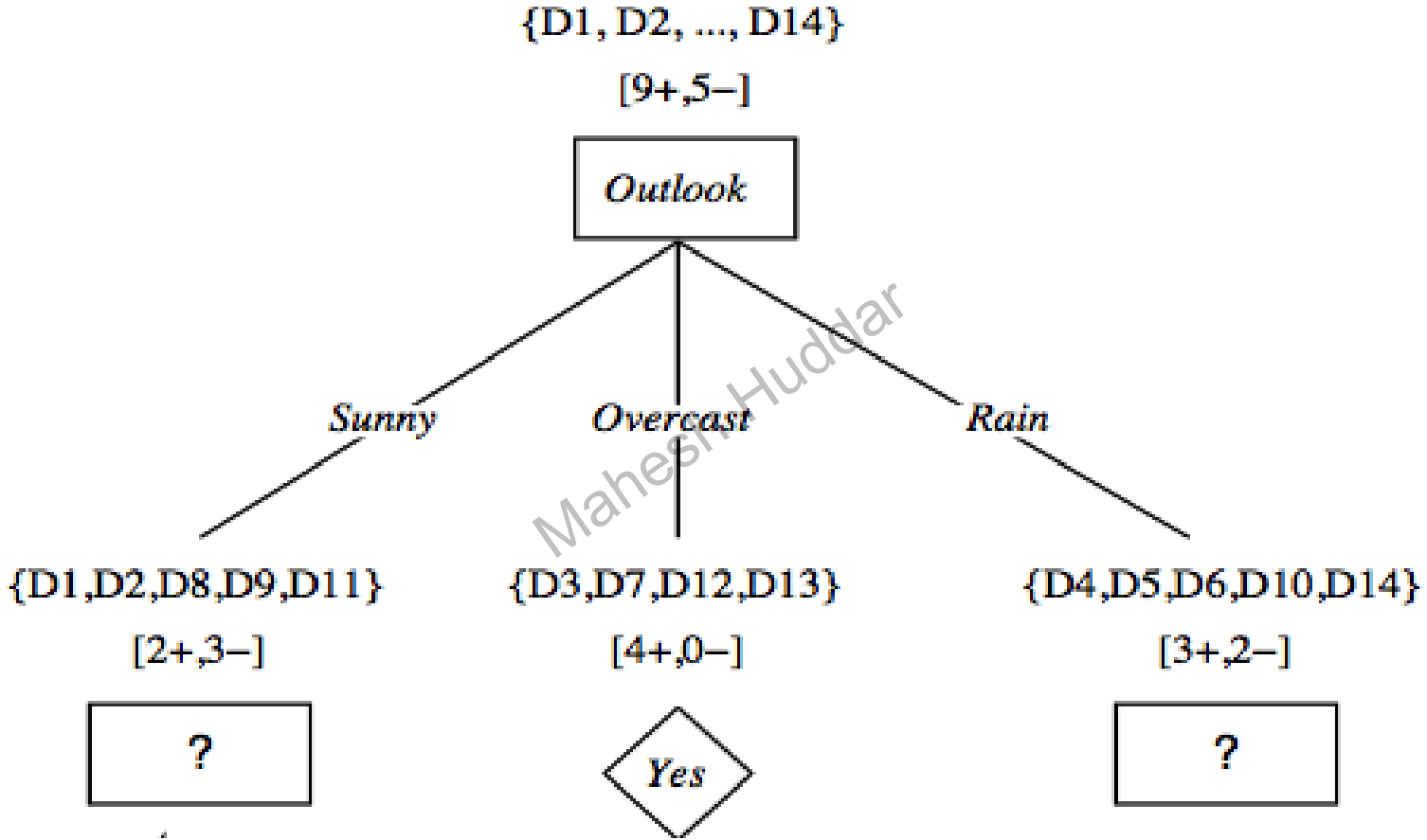
Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

$$Gain(S, Outlook) = 0.2464$$

$$Gain(S, Temp) = 0.0289$$

$$Gain(S, Humidity) = 0.1516$$

$$Gain(S, Wind) = 0.0478$$



Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

Attribute: Temp

$$Values\,(Temp) = Hot, Mild, Cool$$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Mild}) = 1.0$

$S_{Cool} \leftarrow [1+, 0-]$ $\quad Entropy(S_{Cool}) = 0.0$

$$Gain\,(S_{Sunny}, Temp) = Entropy(S_{Sunny}) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S_{Sunny}) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild}) - \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S_{sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}\,1 - \frac{1}{5}0.0 = 0.570$$

Day	Temp	Humidity	Wind	Play Tennis
Dl	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
Dl1	Mild	Normal	Strong	Yes

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$$S_{Sunny} = [2+, 3-]$$

$$Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$$

$$S_{high} \leftarrow [0+, 3-]$$

$$Entropy(S_{High}) = 0.0$$

$$S_{Normal} \leftarrow [2+, 0-]$$

$$Entropy(S_{Normal}) = 0.0$$

$$Gain\,(S_{Sunny}, Humidity) = Entropy(S_{Sunny}) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity)$$

$$= Entropy(S_{Sunny}) - \frac{3}{5}Entropy(S_{High}) - \frac{2}{5}Entropy(S_{Normal})$$

$$Gain(S_{sunny}, Humidity) = 0.97 - \frac{3}{5}\,0.0 - \frac{2}{5}0.0 = 0.97$$

Day	Temp	Humidity	Wind	Play Tennis
Dl	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
Dl1	Mild	Normal	Strong	Yes

Attribute: Wind

$$Values\,(Wind) = Strong, Weak$$

$$S_{Sunny} = [2+, 3-]$$

$$Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$$

$$S_{Strong} \leftarrow [1+, 1-]$$

$$Entropy(S_{Strong}) = 1.0$$

$$S_{Weak} \leftarrow [1+, 2-]$$

$$Entropy(S_{Weak}) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$$

$$Gain(S_{Sunny}, Wind) = Entropy(S_{Sunny}) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Wind)$$

$$= Entropy(S_{Sunny}) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

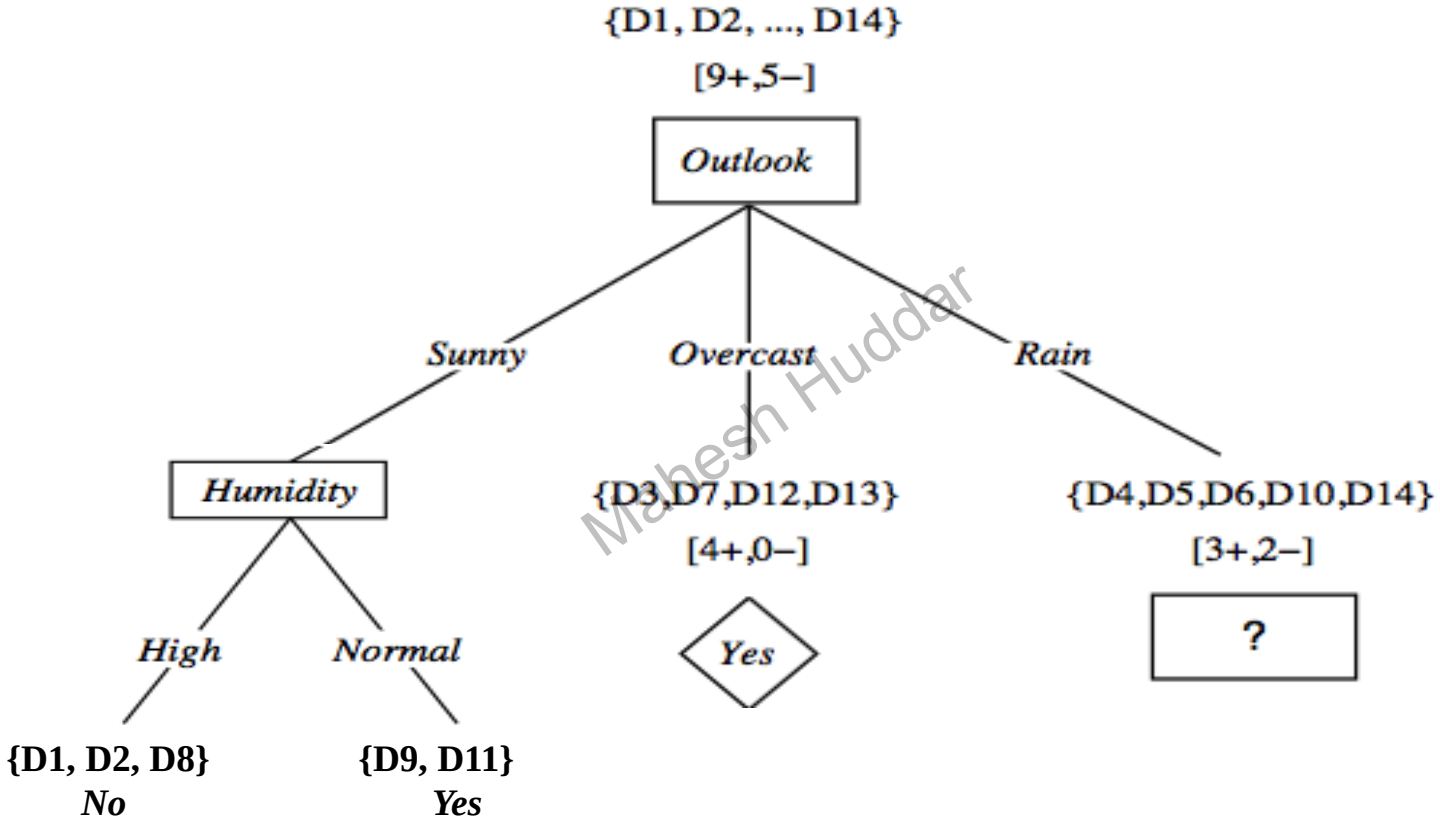
$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5}1.0 - \frac{3}{5}\,0.918 = 0.0192$$

Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

$$Gain(S_{sunny}, Temp) = 0.570$$

$$Gain(S_{sunny}, Humidity) = 0.97$$

$$Gain(S_{sunny}, Wind) = 0.0192$$



Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute: Temp

$$Values\,(Temp) = Hot, Mild, Cool$$

$S_{Rain} = [3+, 2-]$

$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 0-]$

$Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [2+, 1-]$

$Entropy(S_{Mild}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$S_{Cool} \leftarrow [1+, 1-]$

$Entropy(S_{Cool}) = 1.0$

$$Gain\,(S_{Rain}, Temp) = Entropy(S_{Rain}) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S_{Rain}) - \frac{0}{5}Entropy(S_{Hot}) - \frac{3}{5}Entropy(S_{Mild}) - \frac{2}{5}Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5}0.0 - \frac{3}{5}\,0.918 - \frac{2}{5}\,1.0 = 0.0192$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
Dl0	Mild	Normal	Weak	Yes
Dl4	Mild	High	Strong	No

Attribute: Humidity

$$Values\,(Humidity) = High, Normal$$

$S_{Rain} = [3+, 2-]$

$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} =$

0.97

$S_{High} \leftarrow [1+, 1-]$

$Entropy(S_{High}) = 1.0$

$S_{Normal} \leftarrow [2+, 1-]$

$Entropy(S_{Normal}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} =$

0.9183

$$Gain\,(S_{Rain}, Humidity) = Entropy(S_{Rain}) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity)$$

$$= Entropy(S_{Rain}) - \frac{2}{5}Entropy(S_{High}) - \frac{3}{5}Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5}\,1.0 - \frac{3}{5}0.918 = 0.0192$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
Dl0	Mild	Normal	Weak	Yes
Dl4	Mild	High	Strong	No

Attribute: Wind

$Values\ (wind) = Strong, Weak$

$S_{Rain} = [3+, 2-]$

$Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} =$

0.97

$S_{Strong} \leftarrow [0+, 2-]$

$Entropy(S_{Strong}) = 0.0$

$S_{Weak} \leftarrow [3+, 0-]$

$Entropy(S_{weak}) = 0.0$

$$Gain\ (S_{Rain}, Wind) = Entropy(S_{Rain}) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Wind) = Entropy(S_{Rain}) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

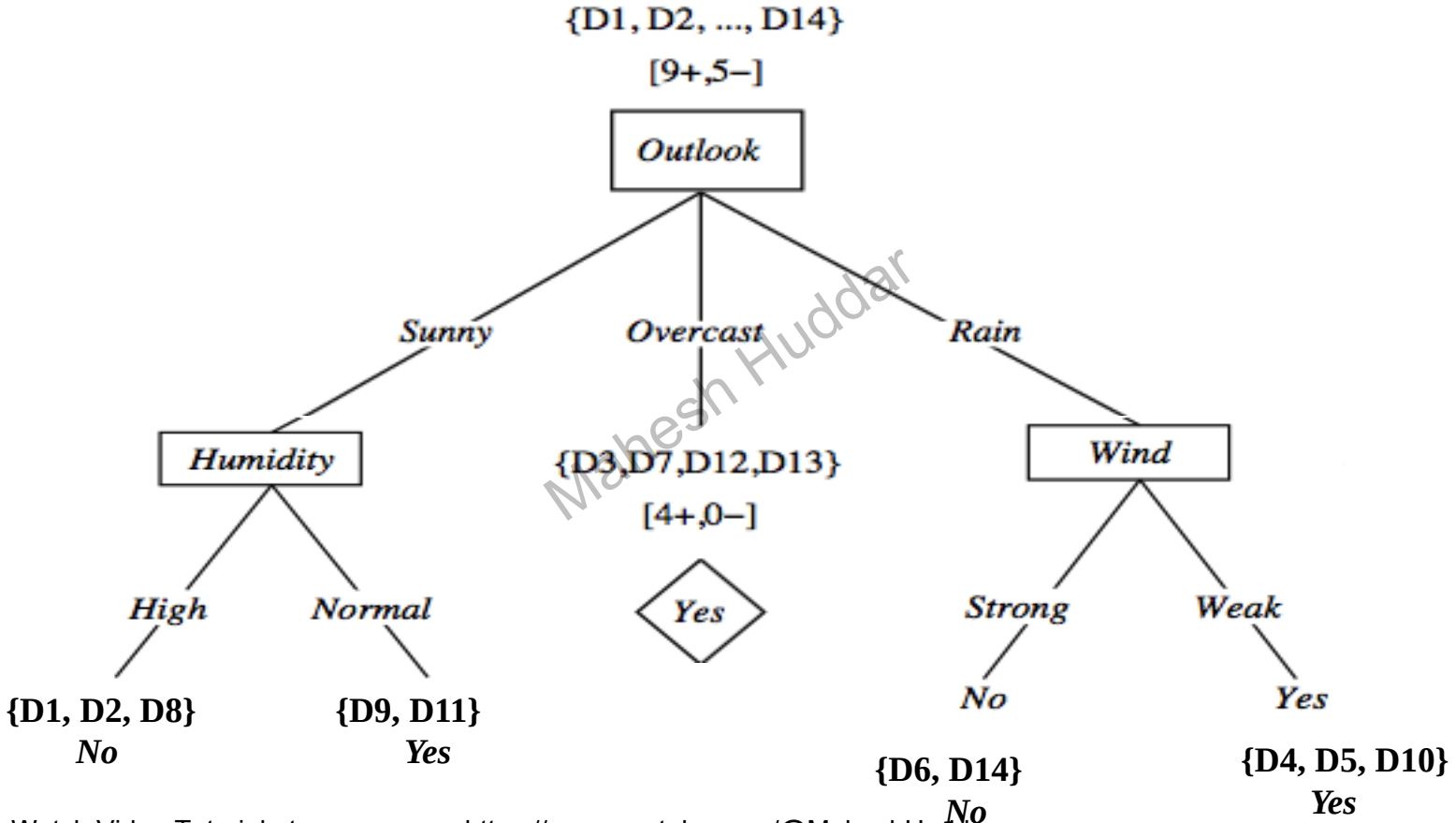
$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\ 0.0 - \frac{3}{5}0.0 = 0.97$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
Dl0	Mild	Normal	Weak	Yes
Dl4	Mild	High	Strong	No

$$Gain(S_{Rain}, Temp) = 0.0192$$

$$Gain(S_{Rain}, Humidity) = 0.0192$$

$$Gain(S_{Rain}, Wind) = 0.97$$



DECISION TREE EXAMPLE

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

1. What is the entropy of this collection of training examples with respect to the target function classification?
2. What is the information gain of *a2* relative to these training examples?
3. Draw decision tree for the given dataset.

Decision Tree Algorithm – ID3 Solved Example

1. What is the entropy of this collection of training examples with respect to the target function classification?
2. What is the information gain of *a1* and *a2* relative to these training examples?
3. Draw decision tree for the given dataset.

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Attribute: a1

$Values\ (a1) = T, F$

$S = [3+,\ 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+,\ 1-]$ $\quad Entropy(S_T) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$S_F \leftarrow [1+,\ 2-]$ $\quad Entropy(S_F) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$

Example - 2
Decision Tree Algorithm – ID3
Solved Example

$$Gain\ (S, a1) = Entropy(S) - \sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, a1) = Entropy(S) - \frac{3}{6}Entropy(S_T) - \frac{3}{6}Entropy(S_F)$$

$$Gain(S, a1) = 1.0 - \frac{3}{6} * 0.9183 - \frac{3}{6} * 0.9183 = 0.0817$$

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Example - 2
Decision Tree Algorithm – ID3
Solved Example

Attribute: a2

$Values\ (a2) = T, F$

$S = [3+,\ 3-]$ $\quad Entropy(S) = 1.0$

$S_T = [2+,\ 2-]$ $\quad Entropy(S_T) = 1.0$

$S_F \leftarrow [1+,\ 1-]$ $\quad Entropy(S_F) = 1.0$

$$Gain\ (S, a2) = Entropy(S) - \sum_{v \in \{T,F\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

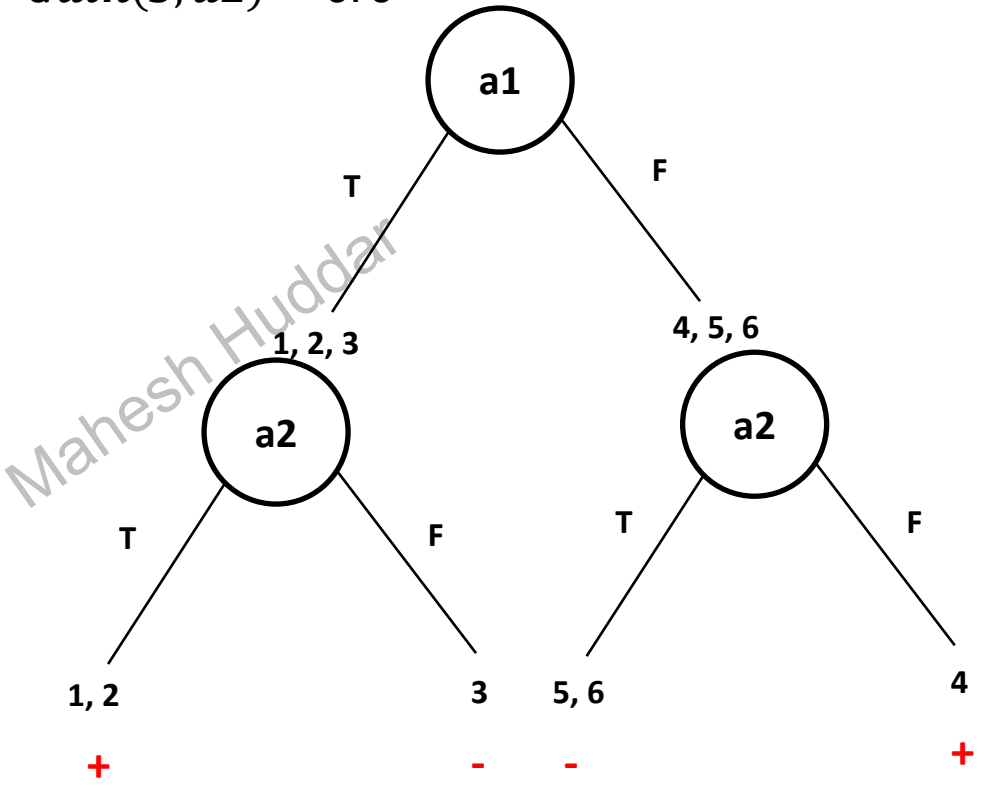
$$Gain(S, a2) = Entropy(S) - \frac{4}{6} Entropy(S_T) - \frac{2}{6} Entropy(S_F)$$

$$Gain(S, a2) = 1.0 - \frac{4}{6} * 1.0 - \frac{2}{6} * 1.0 = 0.0$$

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

$$Gain(S, a1) = 0.0817 \; - \; Maximum\ Gain$$

$$Gain(S, a2) = 0.0$$



Example - 2
Decision Tree Algorithm – ID3
Solved Example

DECISION TREE EXAMPLE

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

1. Construct the decision tree for the following tree using ID3 Algorithm

Decision Tree Algorithm – ID3 Solved Example

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

Example - 3
Decision Tree Algorithm – ID3
Solved Example

Attribute: a1

$$Values\ (a1) = True,\ False$$

$$S = [6+,\ 4-] \qquad Entropy(S) = -\frac{6}{10} log_2 \frac{6}{10} - \frac{4}{10} log_2 \frac{4}{10} = 0.9709$$

$$S_{True} = [1+, 4-] \qquad Entropy(S_{True}) = -\frac{1}{5} log_2 \frac{1}{5} - \frac{4}{5} log_2 \frac{4}{5} = 0.7219$$

$$S_{Flase} \leftarrow [5+,\ 0-] \qquad Entropy(S_{False}) = 0.0$$

$$Gain(S, a1) = Entropy(S) - \sum_{v \in \{True, False\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, a1) = Entropy(S) - \frac{5}{10} Entropy(S_{True}) - \frac{5}{10} Entropy(S_{False})$$

$$Gain(S, a1) = 0.9709 - \frac{5}{10} * 0.7219 - \frac{5}{10} * 1 = 0.6099$$

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

Example - 3
Decision Tree Algorithm – ID3
Solved Example

Attribute: a2

$$Values\ (a2) = Hot,\ Cool$$

$$S = [6+,\ 4-] \qquad Entropy(S) = -\frac{6}{10}log_2\frac{6}{10} - \frac{4}{10}log_2\frac{4}{10} = 0.9709$$

$$S_{Hot} = [2+, 3-] \qquad Entropy(S_{Hot}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.9709$$

$$S_{Cool} \leftarrow [4+,\ 1-] \qquad Entropy(S_{Cool}) = -\frac{4}{5}log_2\frac{4}{5} - \frac{1}{5}log_2\frac{1}{5} = 0.7219$$

$$Gain\ (S, a2) = Entropy(S) - \sum_{v \in \{Hot, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, a2) = Entropy(S) - \frac{5}{10}Entropy(S_{Hot}) - \frac{5}{10}Entropy(S_{Cool})$$

$$Gain(S, a2) = 0.9709 - \frac{5}{10} * 0.9709 - \frac{5}{10} * 0.7219 = 0.1245$$

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

Example - 3
Decision Tree Algorithm – ID3
Solved Example

Attribute: a3

$$Values\ (a3) = High,\ Normal$$

$$S = [6+,\ 4-] \qquad Entropy(S) = -\frac{6}{10} log_2 \frac{6}{10} - \frac{4}{10} log_2 \frac{4}{10} = 0.9709$$

$$S_{High} = [2+, 4-] \qquad Entropy(S_{High}) = -\frac{2}{6} log_2 \frac{2}{6} - \frac{4}{6} log_2 \frac{4}{6} = 0.9183$$

$$S_{Normal} \leftarrow [4+,\ 0-] \qquad Entropy(S_{Normal}) = 0.0$$

$$Gain\ (S, a3) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, a3) = Entropy(S) - \frac{6}{10} Entropy(S_{High}) - \frac{4}{10} Entropy(S_{Normal})$$

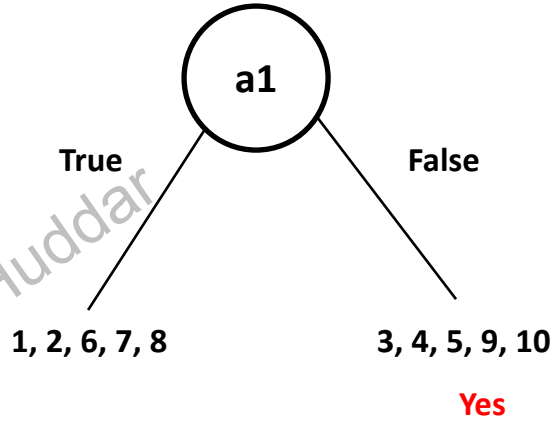
$$Gain(S, a3) = 0.9709 - \frac{6}{10} * 0.9183 - \frac{4}{10} * 0.0 = 0.4199$$

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

$$Gain(S, a1) = 0.6099 \; - \; Maximum\ Gain$$

$$Gain(S, a2) = 0.1245$$

$$Gain(S, a3) = 0.4199$$



Example - 3
Decision Tree Algorithm – ID3
Solved Example

Instance	a2	a3	Classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

Example - 3
Decision Tree Algorithm – ID3
Solved Example

Attribute: a2

$$Values\ (a2) = Hot,\ Cool$$

$$S_{a1} = [1+,\ 4-] \qquad Entropy(S_{a1}) = -\frac{1}{5}log_2\frac{1}{5} - \frac{4}{5}log_2\frac{4}{5} = 0.7219$$

$$S_{Hot} = [1+, 3-] \qquad Entropy(S_{Hot}) = -\frac{1}{4}log_2\frac{1}{4} - \frac{3}{4}log_2\frac{3}{4} = 0.8112$$

$$S_{Cool} \leftarrow [0+,\ 1-] \qquad Entropy(S_{Cool}) = 0.0$$

$$Gain\ (S, a2) = Entropy(S) - \sum_{v \in \{Hot, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, a2) = Entropy(S) - \frac{4}{5}Entropy(S_{Hot}) - \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S, a2) = 0.9709 - \frac{4}{5} * 0.8112 - \frac{1}{5} * 0.0 = 0.3219$$

Instance	a2	a3	Classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

Example - 3
Decision Tree Algorithm – ID3
Solved Example

Attribute: a3

$$Values\,(a3) = High,\ Normal$$

$$S_{a1} = [1+,\ 4-] \qquad Entropy(S_{a1}) = -\frac{1}{5}log_2\frac{1}{5} - \frac{4}{5}log_2\frac{4}{5} = 0.7219$$

$$S_{High} = [0+, 4-] \qquad Entropy(S_{High}) = 0.0$$

$$S_{Normal} \leftarrow [1+,\ 0-] \qquad Entropy(S_{Normal}) = 0.0$$

$$Gain\,(S, a3) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

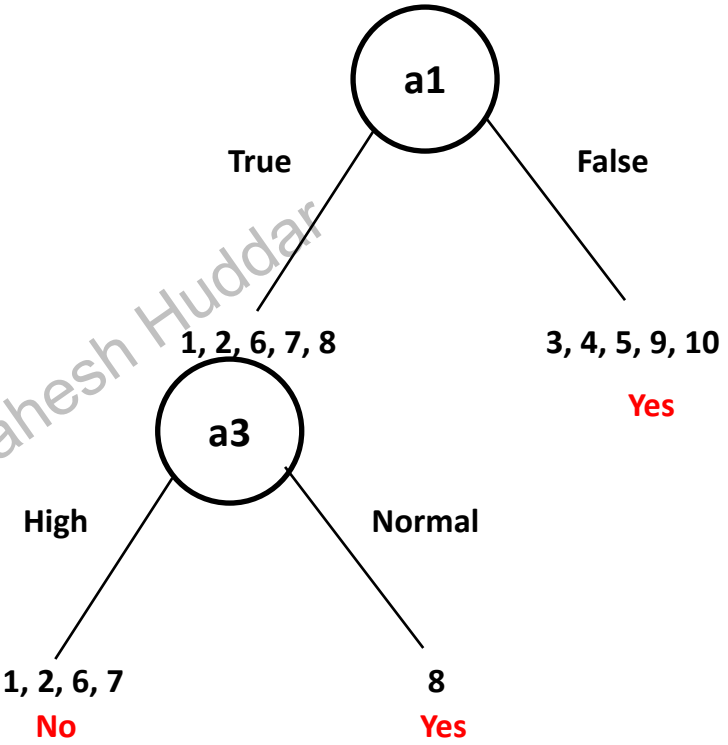
$$Gain(S, a3) = Entropy(S) - \frac{4}{5}Entropy(S_{High}) - \frac{1}{5}Entropy(S_{Normal})$$

$$Gain(S, a3) = 0.9709 - \frac{4}{5} * 0.0 - \frac{1}{5} * 0.0 = 0.7219$$

$$Gain(S_{a1}, a2) = 0.3219$$

$$Gain(S_{a1}, a3) = 0.7219 - \textit{Maximum Gain}$$

Instance	a2	a3	Classification
1	Hot	High	No
2	Hot	High	No
6	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes



Example - 3
Decision Tree Algorithm – ID3
Solved Example

When to use Decision Trees

- Problem characteristics:
  - Instances can be described by attribute value pairs
  - Target function is discrete valued
  - Disjunctive hypothesis may be required
  - Possibly noisy training data samples
    - Robust to errors in training data
    - Missing attribute values
- Different classification problems:
  - Equipment classification
  - Medical diagnosis
  - Credit risk analysis
  - Several tasks in natural language processing

Issues in decision trees learning

- Overfitting
  - Reduced error pruning
  - Rule post-pruning
- Continuous valued attributes
- Alternative measures for selecting attributes
- Handling training examples with missing attribute values
- Handling attributes with different costs
- Improving computational efficiency
- Most of these improvements in C4.5 (Quinlan, 1993)

Overfitting: definition

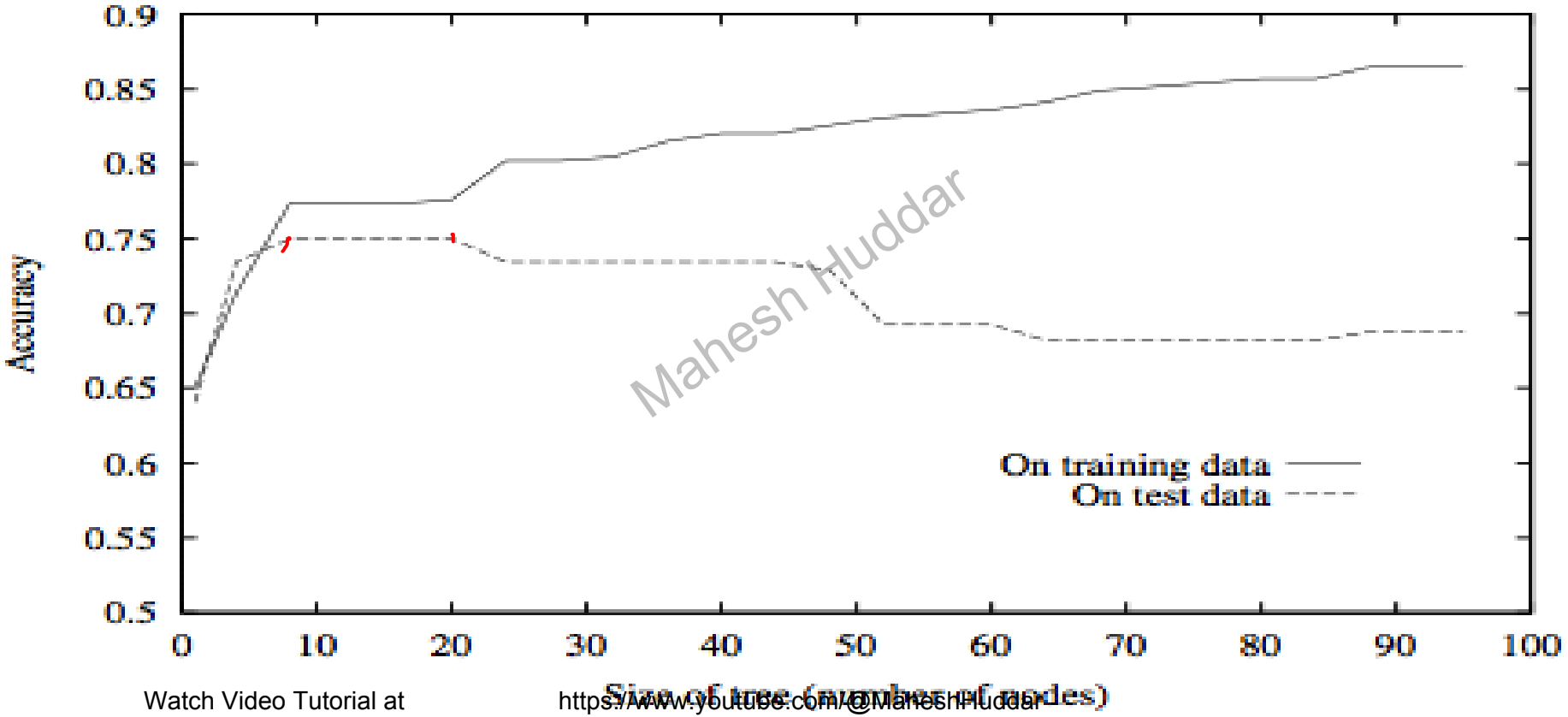
- Building trees that "adapt too much" to the training examples may lead to "overfitting".
- Consider error of hypothesis h over
  - training data: $error_D(h)$ empirical error
  - entire distribution X of data: $error_X(h)$ expected error
- Hypothesis *h* *overfits* training data if there is an alternative hypothesis $h' \in H$ such that

$$error_D(h) < error_D(h') \quad \text{and}$$

$$error_X(h') < error_X(h)$$

i.e. h' behaves better over unseen data

Overfitting in decision tree learning



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

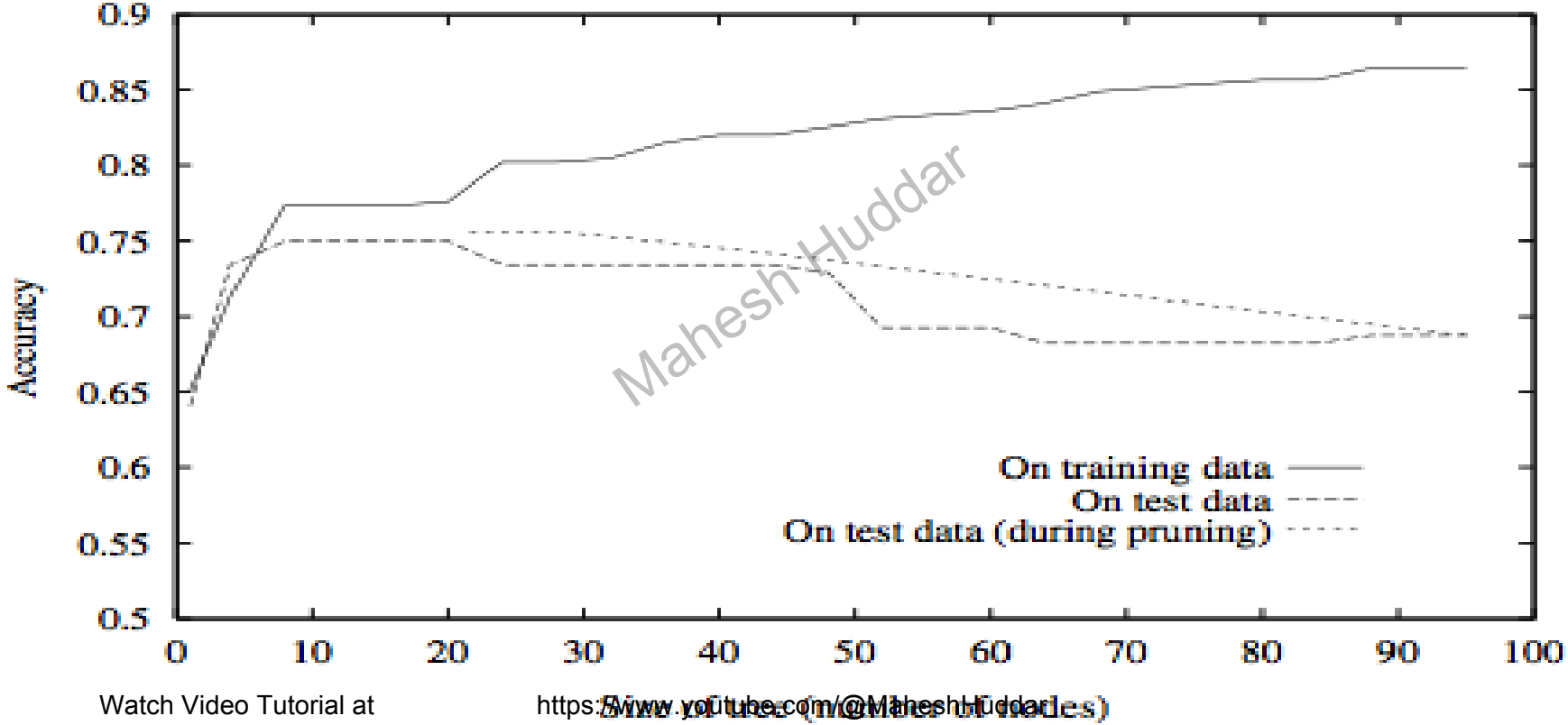
Avoid overfitting in Decision Trees

- **Two strategies:**
  1. Stop growing the tree earlier, before perfect classification
  2. Allow the tree to *overfit* the data, and then *post-prune* the tree
     - Training and validation set: split the training in two parts (training and validation) and use validation to assess the utility of *post-pruning*
       - *Reduced error pruning*
       - *Rule Post pruning*

Reduced-error pruning (Quinlan 1987)

- Each node is a candidate for pruning
- *Pruning* consists in removing a subtree rooted in a node: the node becomes a leaf and is assigned the most common classification
- Nodes are removed only if the resulting tree performs no worse on the validation set.
- Nodes are pruned iteratively: at each iteration the node whose removal most increases accuracy on the validation set is pruned.
- Pruning stops when no pruning increases accuracy

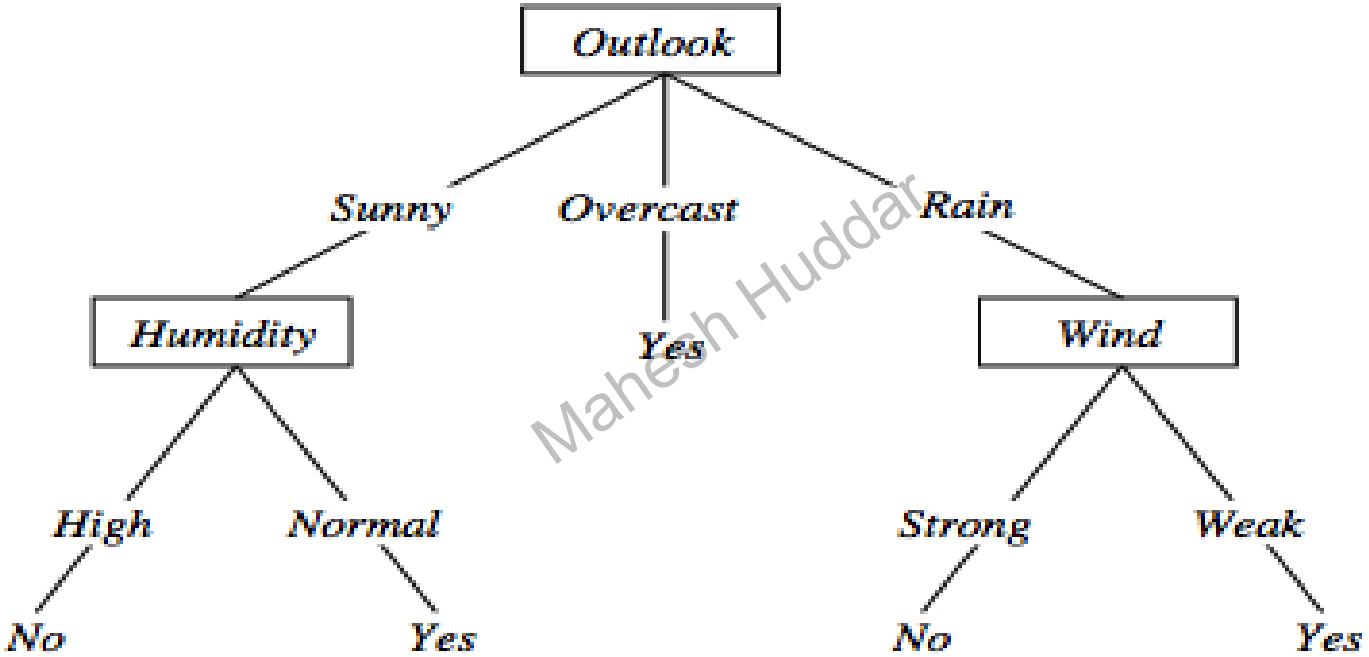
Effect of reduced error pruning



Rule post-pruning

1. Create the decision tree from the training set
2. Convert the tree into an equivalent set of rules
   - Each path corresponds to a rule
   - Each node along a path corresponds to a pre-condition
   - Each leaf classification to the post-condition
3. Prune (generalize) each rule by removing those preconditions whose removal improves accuracy over validation set
4. Sort the rules in estimated order of accuracy, and consider them in sequence when classifying new instances

Converting to rules



Rule Post-Pruning

- Convert tree to rules (one for each path from root to a leaf)
- For each antecedent in a rule, remove it if error rate on validation set does not decrease
- Sort final rule set by accuracy


```
Outlook=sunny ^ humidity=high -> No
Outlook=sunny ^ humidity=normal -> Yes
Outlook=overcast -> Yes
Outlook=rain ^ wind=strong -> No
Outlook=rain ^ wind=weak -> Yes
```

Compare first rule to:

```
Outlook=sunny->No
Humidity=high->No
```

Calculate accuracy of 3 rules based on validation set and pick best version.

Why converting to rules?

- Each distinct path produces a different rule: a condition removal may be based on a local (contextual) criterion. Node pruning is global and affects all the rules
- In rule form, tests are not ordered and there is no book-keeping involved when conditions (nodes) are removed
- Converting to rules improves readability for humans

Dealing with continuous-valued attributes

- So far discrete values for attributes and for outcome.
- Given a continuous-valued attribute A, dynamically create a new attribute A_c

$$A_c = \text{True } if\ A < c, \text{ False } otherwise$$

- How to determine threshold value c ?
- Example. *Temperature* in the *PlayTennis* example
  - Sort the examples according to *Temperature*

Temperature	40	48	60	72	80	90
PlayTennis	No	No	Yes	Yes	Yes	No

  - Determine candidate thresholds by averaging consecutive values where there is a change in classification: (48+60)/2=54 and (80+90)/2=85
  - Evaluate candidate thresholds (attributes) according to information gain. The best is $Temperature_{>54}$. The new attribute competes with the other ones

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Handling incomplete training data

- How to cope with the problem that the value of some attribute may be missing?
  - *Example*: Blood-Test-Result in a medical diagnosis problem
- The strategy: use other examples to guess attribute
  1. Assign the value that is most common among the training examples at the node
  2. Assign a probability to each value, based on frequencies, and assign values to missing attribute, according to this probability distribution
- Missing values in new instances to be classified are treated accordingly, and the most probable classification is chosen (C4.5)

Handling attributes with different costs

- Instance attributes may have an associated cost: we would prefer decision trees that use low-cost attributes
- ID3 can be modified to take into account costs:

1. Tan and Schlimmer (1990)

$$\frac{Gain^2(S, A)}{Cost(S, A)}$$

2. Nunez (1988)

$$\frac{2^{Gain(S, A)} - 1}{(Cost(A) + 1)^w} \quad w \in [0,1]$$

Inductive Bias - ID3 Decision Tree Learning

- Given a collection of training examples, there are typically many decision trees consistent with these examples.
- Describing the inductive bias of ID3 therefore consists of describing the basis by which it chooses one of these consistent hypotheses over the others.
- Which of these decision trees does ID3 choose?
- It chooses the first acceptable tree it encounters in its simple-to-complex, hill climbing search through the space of possible trees.

Inductive Bias - ID3 Decision Tree Learning

- Approximate inductive bias of ID3:
  - Shorter trees are preferred over larger trees.
- *A closer approximation to the inductive bias of ID3:*
  - *Shorter trees are preferred over longer trees.*
  - *Trees that place high information gain attributes close to the root are preferred over those that do not.*

Inductive Bias - ID3 Decision Tree Learning

- **Why Prefer Short Hypotheses?**
- Arguments in favor:
  - There are fewer short hypotheses than long ones
  - If a short hypothesis fits data unlikely to be a coincidence
- Arguments against:
  - Not every short hypothesis is a reasonable one.
- **Occam's razor:** *"The simplest explanation is usually the best one."*
  - a principle usually attributed14th-century English logician and Franciscan friar, William of Ockham.
  - The term razor refers to the act of *shaving away* unnecessary assumptions to get to the simplest explanation.

Two kinds of biases

- Preference or search biases
  - ID3 searches through incompletely through *complete* hypotheses space; from simple to complex hypotheses, until termination condition is reached.
- Restriction or language biases
  - *It searches the hypotheses space completely*
  - *Candidate-Elimination* search is *complete*

Artificial Neural Networks

Dr. Mahesh G Huddar

Dept. of Computer Science and Engineering

Artificial Neural Networks

- Introduction
- Neural Network Representation
- Appropriate Problems for Neural Network Learning
- Perceptrons
- Multilayer Networks and BACKPROPAGATION Algorithms
- Remarks on the BACKPROPAGATION Algorithms

Artificial Neural Networks

- ANN learning well-suited to problems which the training data corresponds to noisy, complex data (inputs from cameras or microphones)
- Can also be used for problems with symbolic representations
- Most appropriate for problems where
  - Instances have many attribute-value pairs
  - Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
  - Training examples may contain errors
  - Long training times are acceptable
  - Fast evaluation of the learned target function may be required
  - The ability for humans to understand the learned target function is not important

Appropriate Problems – for ANN

- ***Instances are represented by many attribute-value pairs.*** The target function to be learned is defined over instances that can be described by a vector of predefined features, such as the pixel values in the ALVINN example. These input attributes may be highly correlated or independent of one another. Input values can be any real values.
- ***The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.*** For example, in the ALVINN system the output is a vector of 30 attributes, each corresponding to a recommendation regarding the steering direction. The value of each output is some real number between 0 and 1, which in this case corresponds to the confidence in predicting the corresponding steering direction. We can also train a single network to output both the steering command and suggested acceleration, simply by concatenating the vectors that encode these two output predictions.

Appropriate Problems – for ANN

The training examples may contain errors. ANN learning methods are quite robust to noise in the training data.

Long training times are acceptable. Network training algorithms typically require longer training times than, say, decision tree learning algorithms. Training times can range from a few seconds to many hours, depending on factors such as the number of weights in the network, the number of training examples considered, and the settings of various learning algorithm parameters.

Appropriate Problems – for ANN

Fast evaluation of the learned target function may be required. Although ANN learning times are relatively long, evaluating the learned network, in order to apply it to a subsequent instance, is typically very fast. For example, ALVINN applies its neural network several times per second to continually update its steering command as the vehicle drives forward.

The ability of humans to understand the learned target function is not important. The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules

Neural Network History

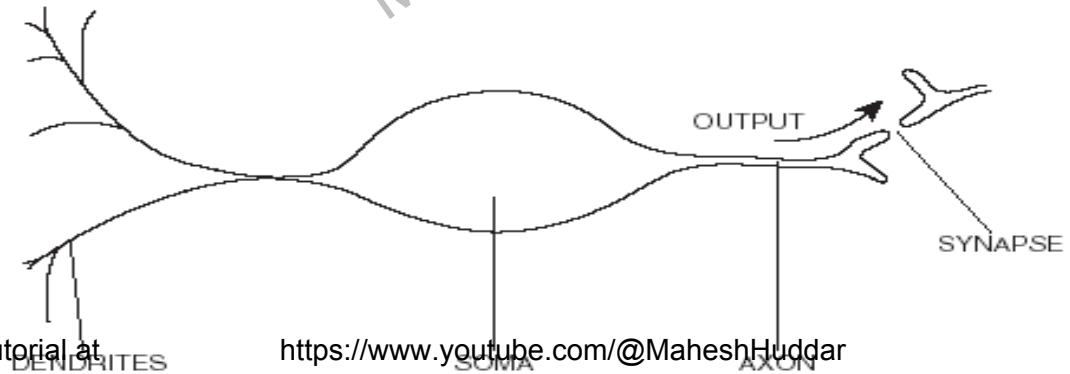
- History traces back to the 50's but became popular in the 80's with work by Rumelhart, Hinton, and Mclelland
  - A General Framework for Parallel Distributed Processing in Parallel Distributed Processing: Explorations in the Microstructure of Cognition
- Peaked in the 90's.:
  - Hundreds of variants
  - Less a model of the actual brain than a useful tool, but still some debate
- Numerous applications
  - Handwriting, face, speech recognition
  - Vehicles that drive themselves
  - Models of reading, sentence production, dreaming
- Debate for philosophers and cognitive scientists
  - Can human consciousness or cognitive abilities be explained by a connectionist model or does it require the manipulation of symbols?

Biological Motivation

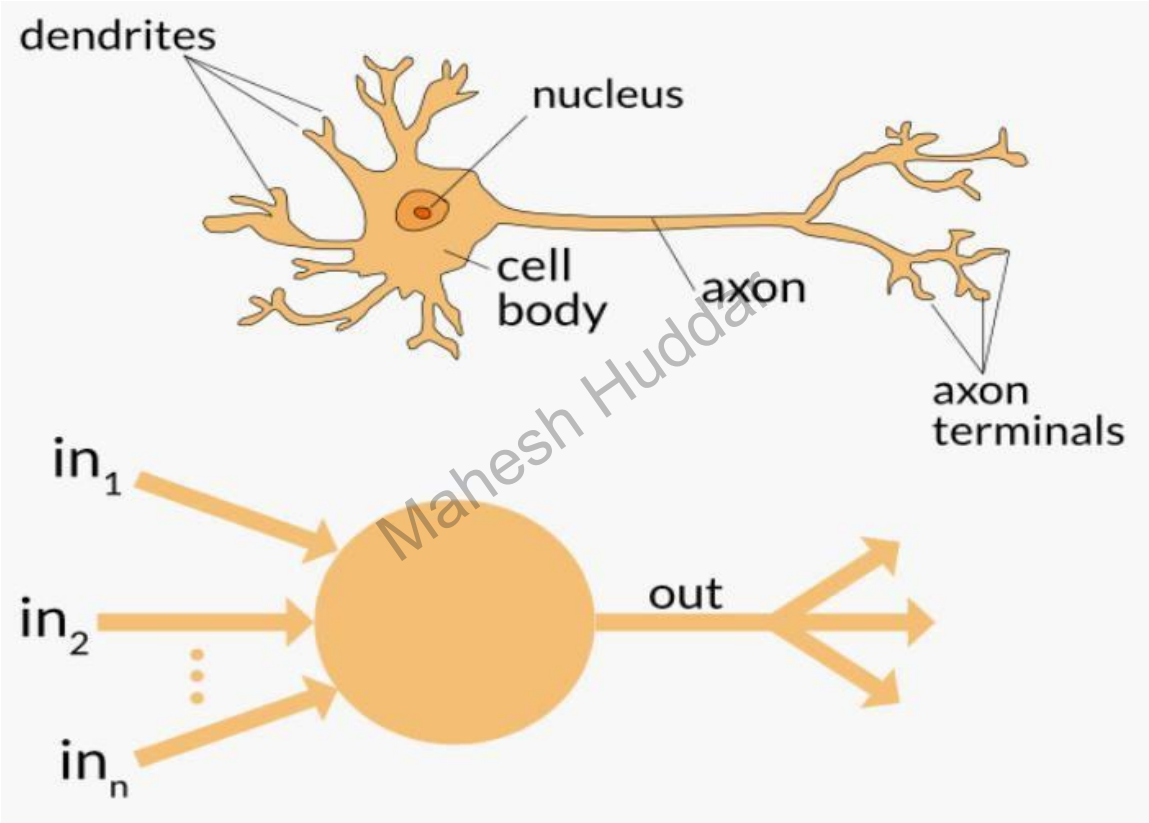
- The study of artificial neural networks (ANNs) has been inspired by the observation that biological learning systems are built of very complex webs of interconnected ***Neurons***
- Human information processing system consists of brain neuron: basic building block cell that communicates information to and from various parts of body
- Simplest model of a neuron: considered as a threshold unit –a processing element (PE)
- Collects inputs & produces output if the sum of the input exceeds an internal threshold value

Biological Motivation

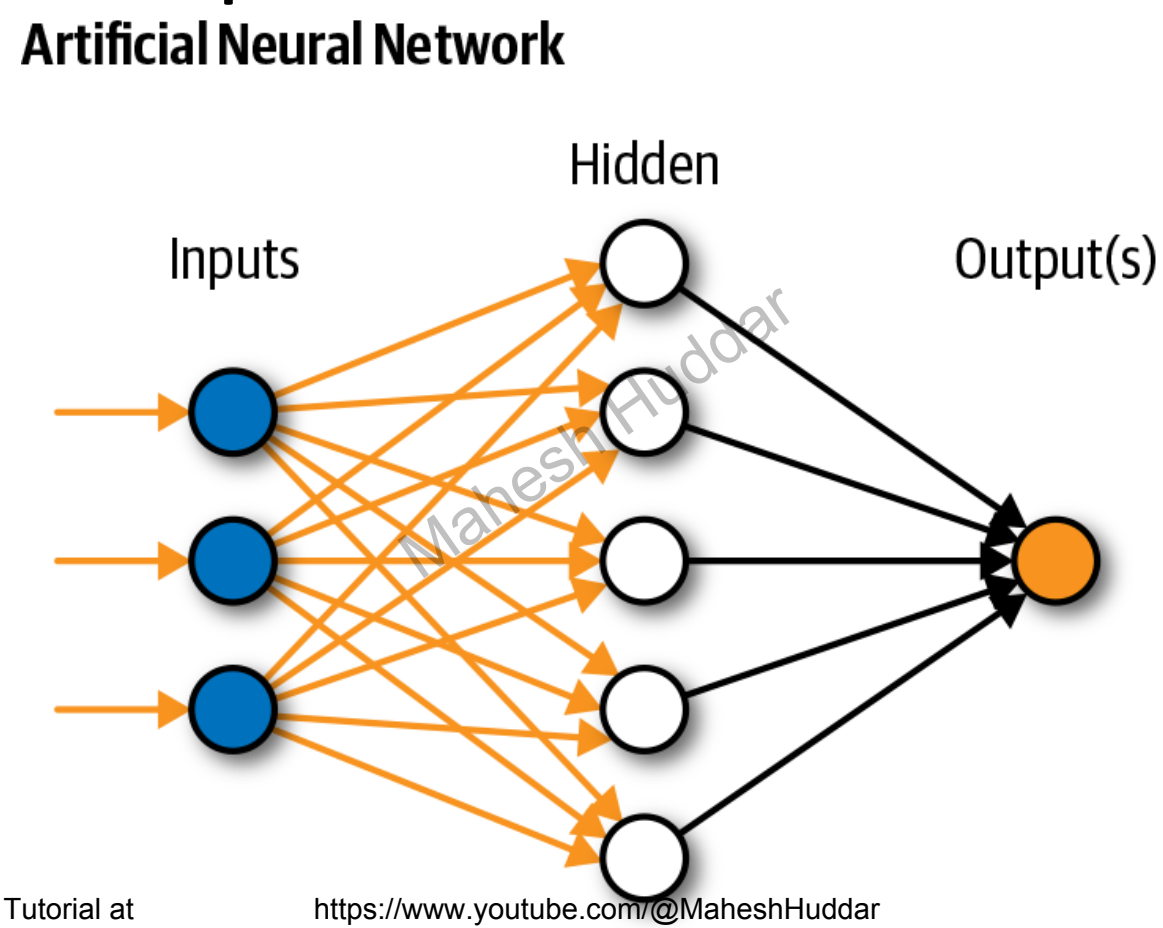
- The human brain is made up of billions of simple processing units – neurons.
- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron.



Biological Motivation

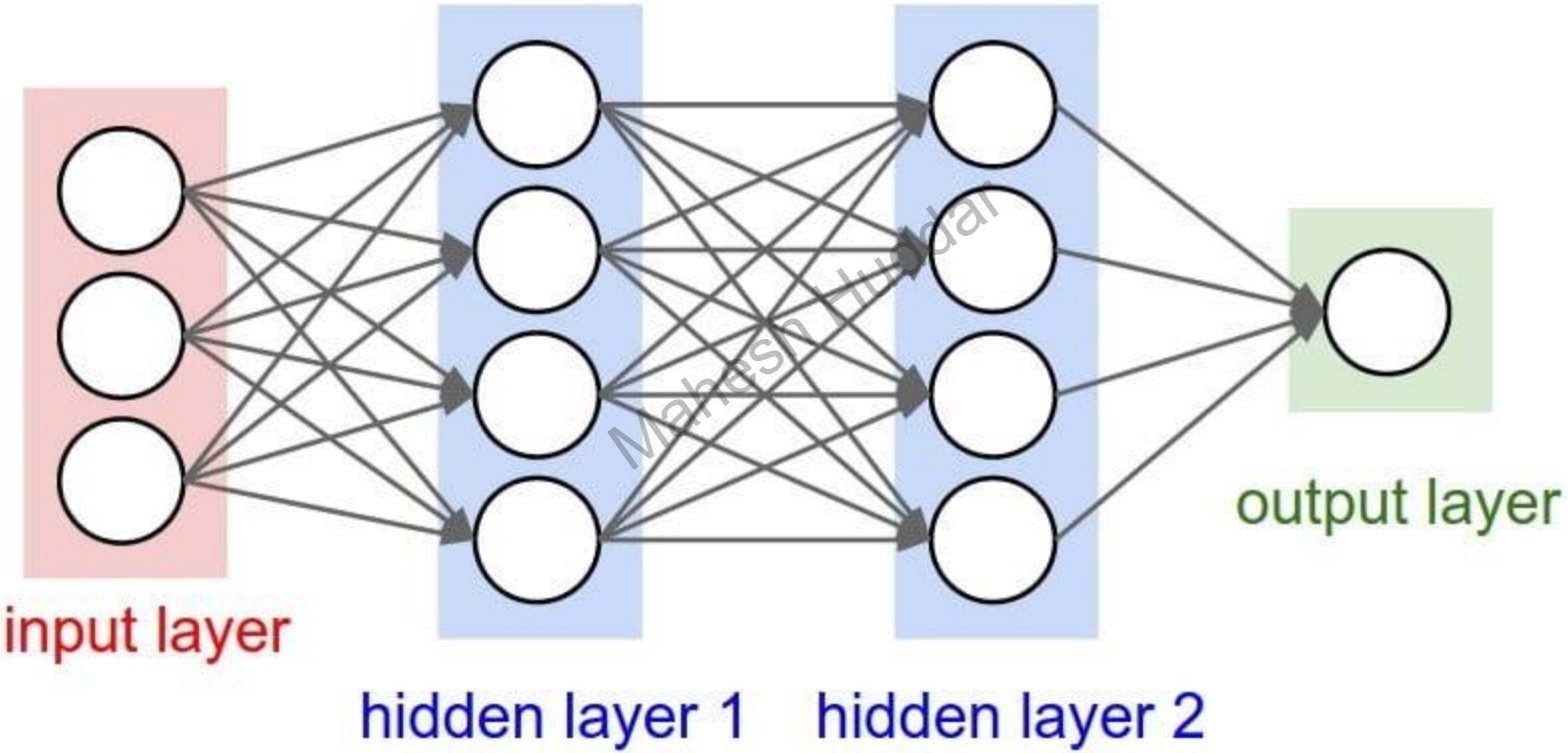


Simplest Neural Network



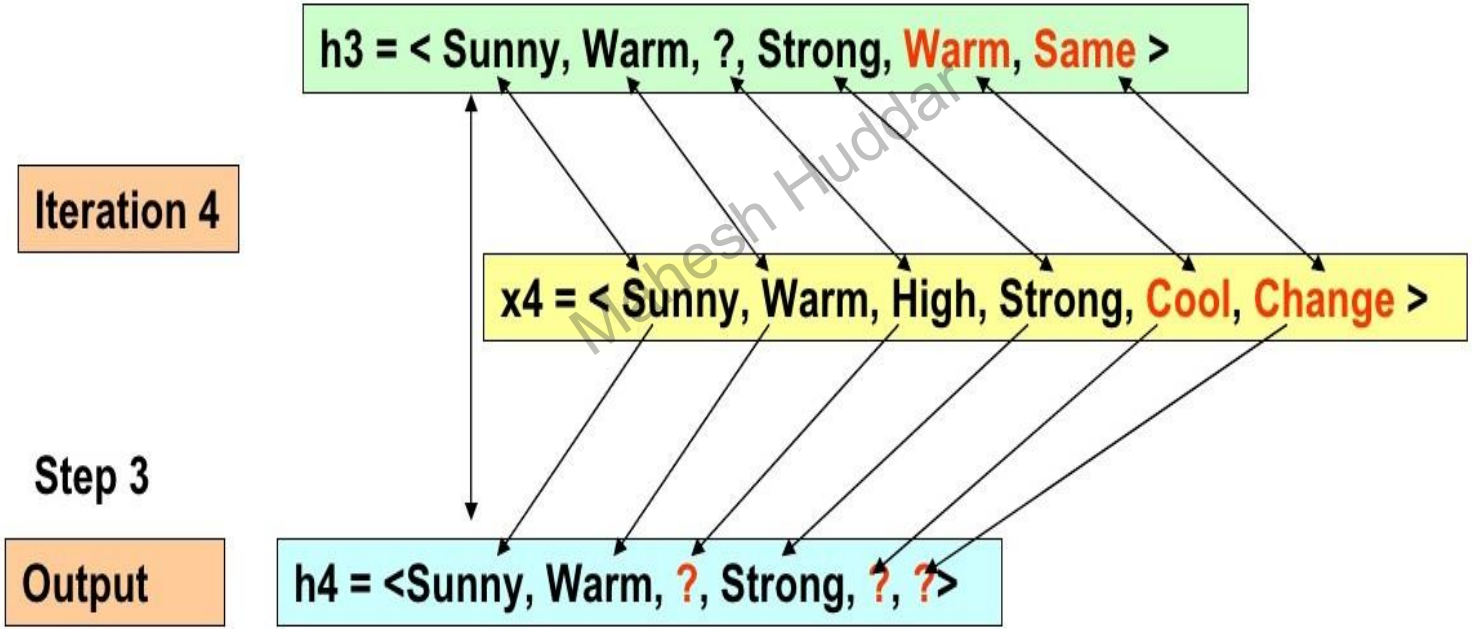
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Simplest Neural Network



FIND-S: Step-2

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes



Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

S_0: ⟨∅, ∅, ∅, ∅, ∅. ∅⟩

S_1: ⟨*Sunny,Warm, Normal, Strong, Warm, Same*⟩

S_2: S_3: ⟨*Sunny,Warm, ?, Strong, Warm, Same*⟩

S_4 ⟨*Sunny, Warm, ?, Strong, ?, ?*⟩

G_4: ⟨*Sunny, ?, ?, ?, ?, ?*⟩ ⟨*?, Warm, ?, ?, ?, ?*⟩

G_3: ⟨*Sunny,?,?,?,?,?*⟩ ⟨*?,Warm,?,?,?,?*⟩ ⟨*?,?,Normal,?,?,?*⟩ ⟨*?, ?,?,?,Cool,?*⟩ ⟨*?,?,?,?,?,Same*⟩

G_0: G_1: G_2: ⟨?, ?, ?, ?, ?, ?⟩

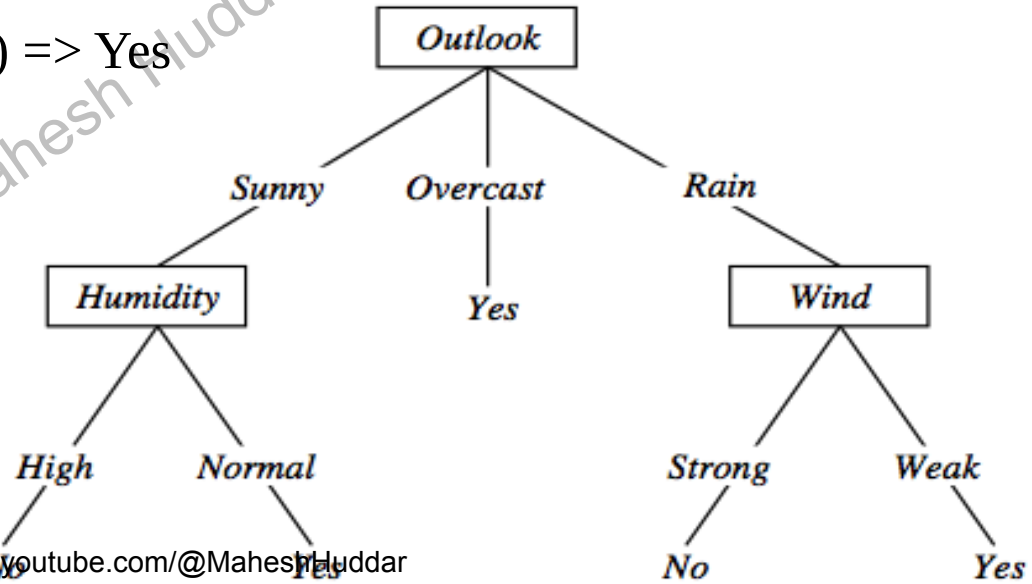
Decision Trees

- Decision trees represent a disjunction of conjunctions on constraints on the value of attributes:

  $(Outlook = Sunny \wedge Humidity = Normal)$ => Yes

  $(Outlook = Overcast)$ => Yes

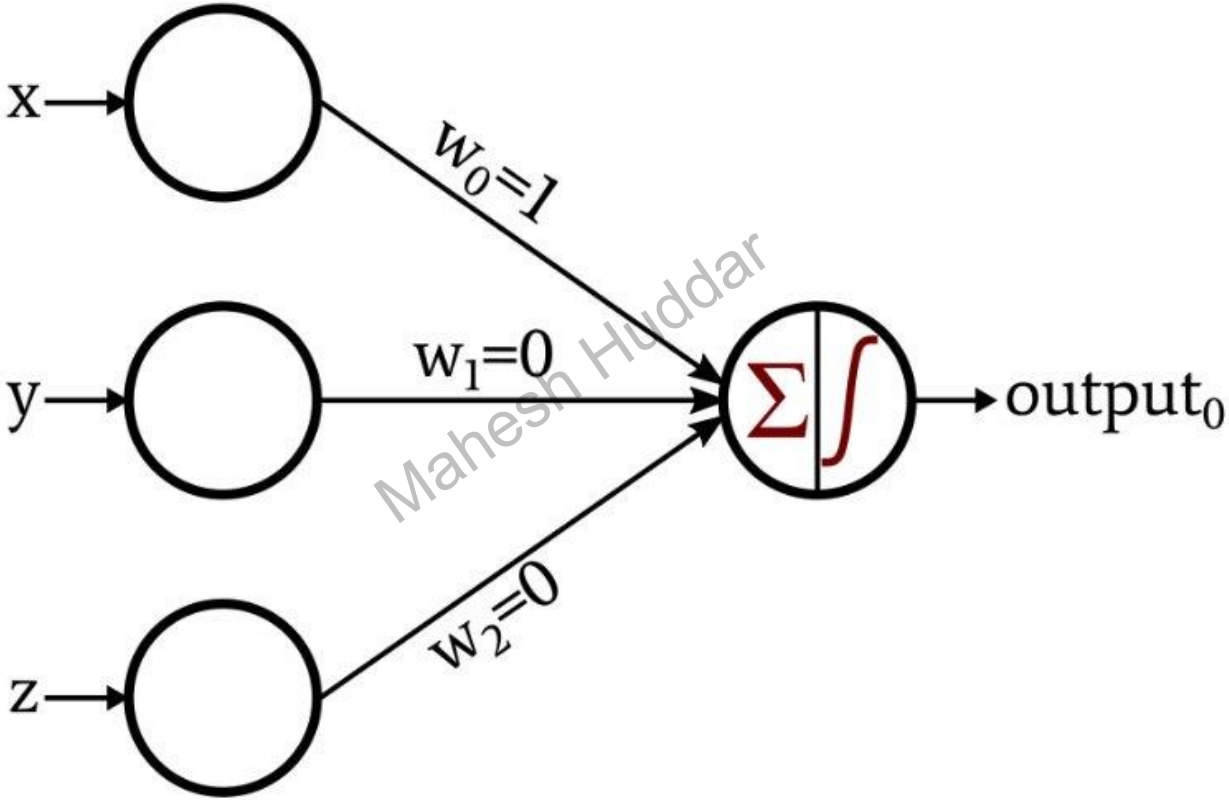
  $(Outlook = Rain \wedge Wind = Weak)$ => Yes



Artificial Neurons

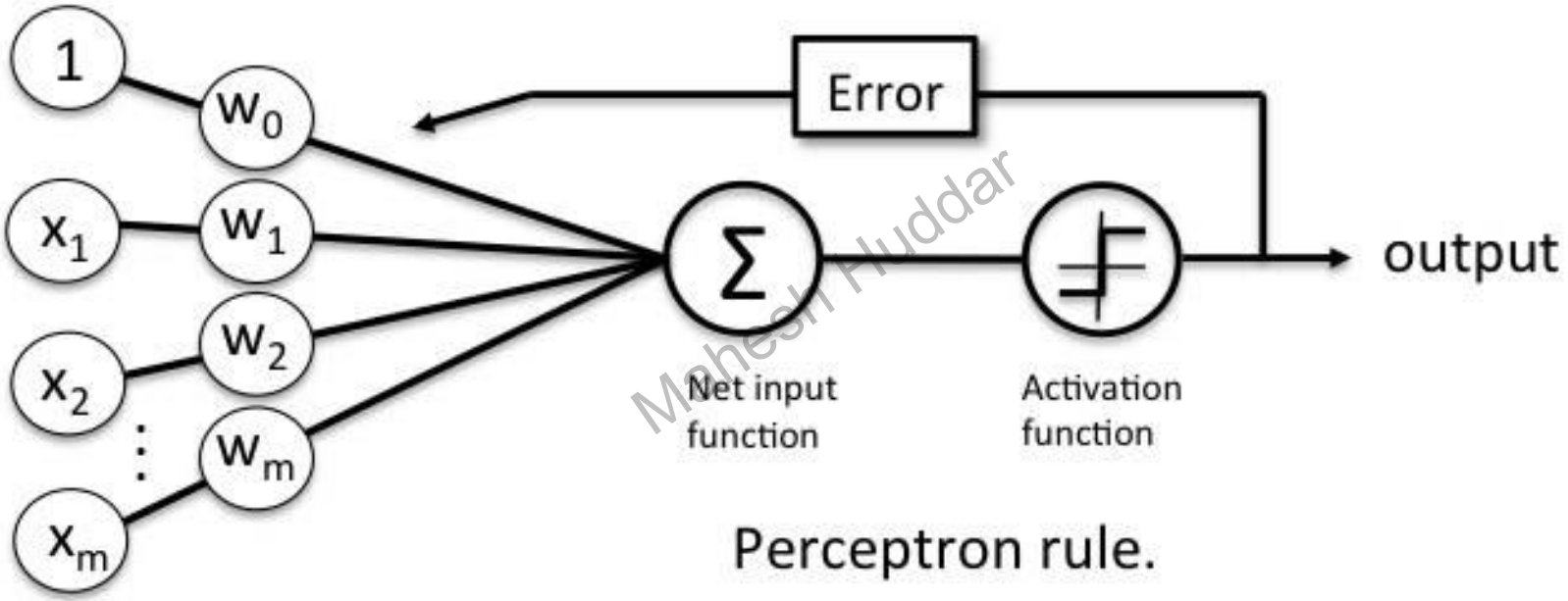
- Artificial neurons are based on biological neurons.
- Each neuron in the network receives one or more inputs.
- An activation function is applied to the inputs, which determines the output of the neuron – the activation level.

Artificial Neurons

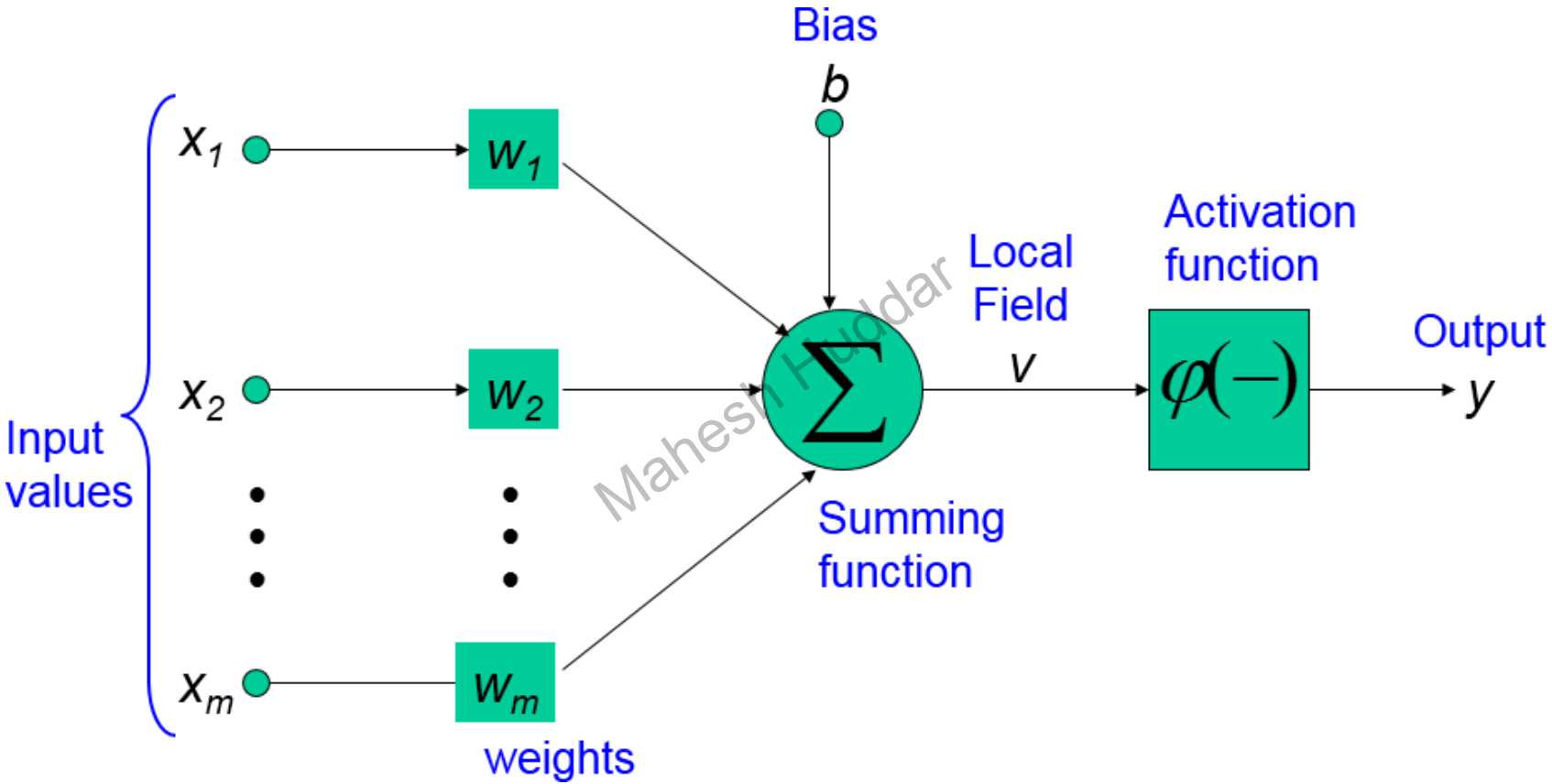


Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Artificial Neurons



Artificial Neurons



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Artificial Neurons

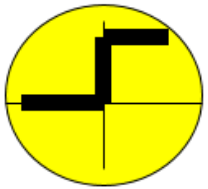
- A typical activation function works as follows:

$$X = \sum_{i=1}^{n} w_i x_i \qquad Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \le t \end{cases}$$

- Each node *i* has a weight, w_i associated with it.
- The input to node *i* is x_i.
- *t* is the threshold.
- So if the weighted sum of the inputs to the neuron is above the threshold, then the neuron fires.

Artificial Neurons

- The charts on the right show three typical activation functions.

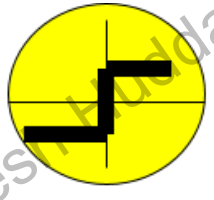


Step function

(Linear Threshold Unit)

step(x) = 1, if x >= threshold

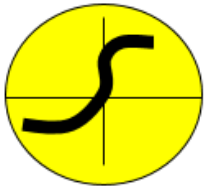
0, if x < threshold



Sign function

sign(x) = +1, if x >= 0

-1, if x < 0



Sigmoid function

sigmoid(x) = $1/(1+e^{-x})$

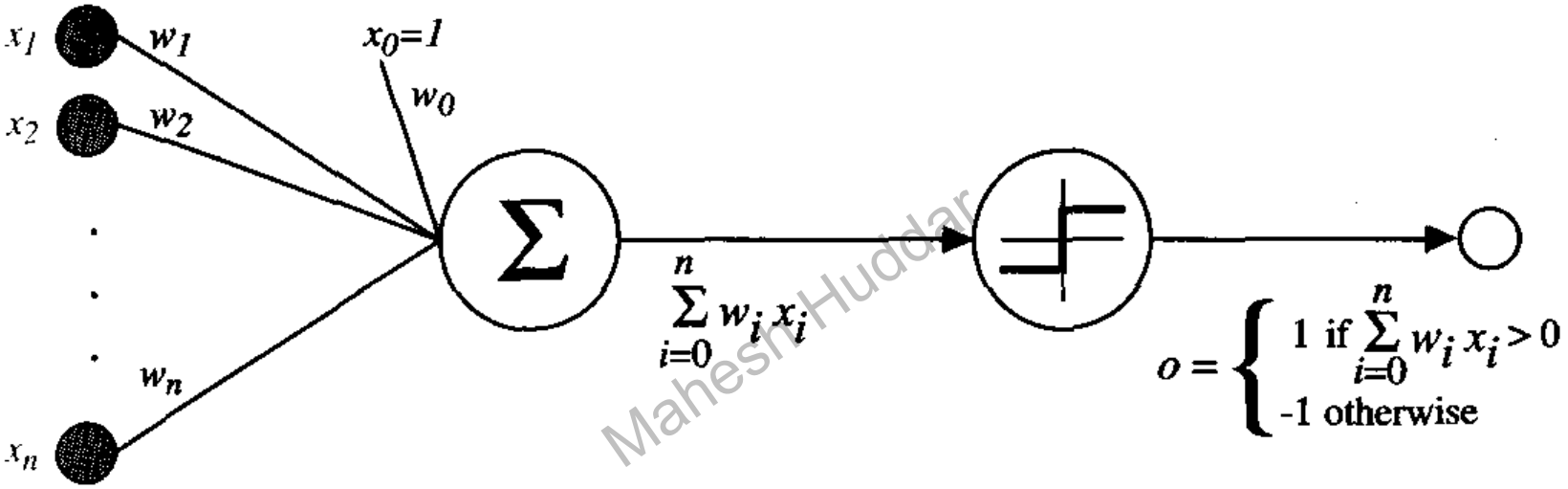
PERCEPTRON

- One type of ANN system is based on a unit called a perceptron.
- A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.
- More precisely, given inputs ***x1*** through ***xn,*** the output ***o(x1,*** . . . , ***xn)*** computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

- where each ***wi*** is a real-valued constant, or weight, that determines the contribution of input ***xi*** to the perceptron output

PERCEPTRON



$$o(x_1, \ldots, x_n) = \begin{cases} 1 \text{ if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

PERCEPTRON

- Sometimes we write the perceptron function as,

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

where

$$sgn(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

- Learning a perceptron involves choosing values for the weights ***wo,*** . . . , ***wn.***
- Therefore, the space H of candidate hypotheses considered in perceptron learning is the set of all possible real-valued weight vectors.

$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

The Perceptron Training Rule

- One way to learn an acceptable weight vector is to begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example.
- This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the ***perceptron training rule,*** which revises the weight ***wi*** associated with input ***xi*** according to the rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

The Perceptron Training Rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

- Here ***t*** is the target output for the current training example, ***o*** is the output generated by the perceptron, and *n* is a positive constant called the ***learning rate.***
- The role of the learning rate is to moderate the degree to which weights are changed at each step.
- It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.

The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions weights 0.6 and 0.6

AND function

$Threshhold\ \theta = 1$

Learning Rate $n = 0.5$

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

If A=0 & B=0 → 0*0.6 + 0*0.6 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*0.6 + 1*0.6 = 0.6

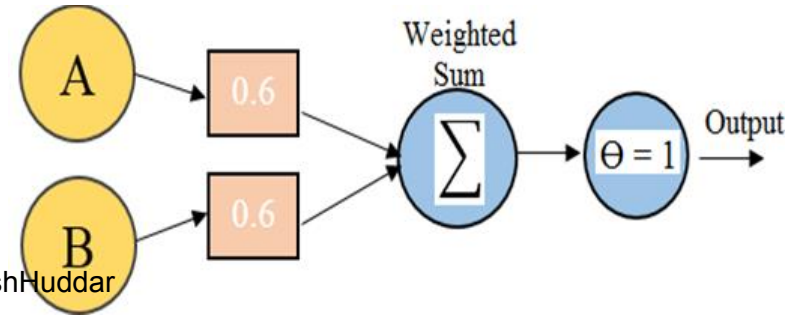
This is not greater than the threshold, so the output = 0

If A=1 & B=0 → 1*0.6 + 0*0.6 = 0.6

This is not greater than the threshold, so the output = 0

If A=1 & B=1 → 1*0.6 + 1*0.6 = 1.2

This exceeds the threshold, so the output = 1



The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions weights 1.2 and 0.6

AND function

$Threshhold\ \theta = 1$

Learning Rate $n = 0.5$

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

If A=0 & B=0 → 0*1.2 + 0*0.6 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*1.2 + 1*0.6 = 0.6

This is not greater than the threshold, so the output = 0

If A=1 & B=0 → 1*1.2 + 0*0.6 = 1.2

This is greater than the threshold, so the output = 1

But the expected output is 0

The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions weights 1.2 and 0.6

AND function

$$Threshhold\ \theta = 1$$

$$\text{Learning Rate } n = 0.5$$

$$wi = wi + n(t - o)xi$$

$$w1 = 1.2 + 0.5(0 - 1)1 = 0.7$$

$$w2 = 0.6 + 0.5(0 - 1)0 = 0.6$$

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1

The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions weights 0.7 and 0.6

AND function

If A=0 & B=0 → 0*0.7 + 0*0.6 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*0.7 + 1*0.6 = 0.6

This is not greater than the threshold, so the output = 0

If A=1 & B=0 → 1*0.7 + 0*0.6 = 0.7

This is greater than the threshold, so the output = 0

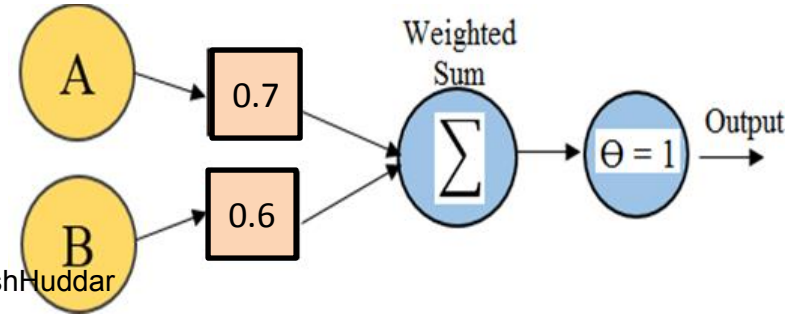
If A=1 & B=0 → 1*0.7 + 1*0.6 = 1.3

This is greater than the threshold, so the output = 1

$$Threshhold\ \theta = 1$$

$$\text{Learning Rate } n = 0.5$$

A	B	A ^ B
0	0	0
0	1	0
1	0	0
1	1	1



The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions with initial weights 0.6 and 0.6

$Threshhold\ \theta = 1$

Learning Rate $n = 0.5$

OR function

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

If A=0 & B=0 → 0*0.6 + 0*0.6 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*0.6 + 1*0.6 = 0.6

This is not greater than the threshold, so the output = 0

But the expected output is 1

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)0 = 0.6$$

$$w2 = 0.6 + 0.5(1 - 0)1 = 1.1$$

The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions

OR function

$Threshhold\ \theta = 1$

Learning Rate $n = 0.5$

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

If A=0 & B=0 → 0*0.6 + 0*1.1 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*0.6 + 1*1.1 = 1.1

This is greater than the threshold, so the output = 1

If A=1 & B= → 1*1.1 + 0*1.1 = 1.1

This is greater than the threshold, so the output = 1

But the expected output is 1

$$wi = wi + n(t - o)xi$$

$$w1 = 0.6 + 0.5(1 - 0)1 = 1.1$$

$$w2 = 1.1 + 0.5(1 - 0)0 = 1.1$$

The Perceptron Training Rule

A single perceptron can be used to represent many Boolean functions

OR function

$Threshhold\ \theta = 1$

Learning Rate $n = 0.5$

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

If A=0 & B=0 → 0*1.1 + 0*1.1 = 0

This is not greater than the threshold of 1, so the output = 0

If A=0 & B=1 → 0*1.1 + 1*1.1 = 1.1

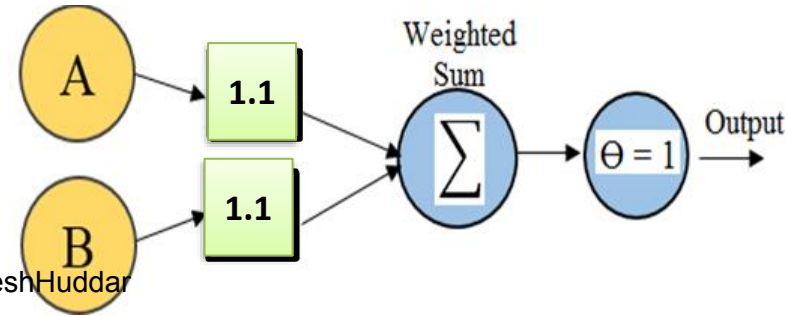
This is greater than the threshold, so the output = 1.

If A=1 & B= → 1*1.1 + 0*1.1 = 1.1

This is greater than the threshold, so the output = 1.

If A=1 & B= → 1*1.1 + 1*1.1 = 2.2

This is greater than the threshold, so the output = 1.



The Perceptron Training Rule

Perceptron_training_rule (X, η)
initialize **w** (wi ← an initial (small) random value)
repeat
    for each training instance (**x**, *t***x**) ∈ X
        compute the real output *o***x**=*Summation(****w****.*x*)*
        if (*t***x** ≠ *o***x**)
            for each wi
                wi ← wi + Δ*wi*
                Δ*wi* ← η(t**x**-*o***x**)xi
            end for
        end if
    end for
until all the training instances in X are correctly classified
return **w**

X: training data
η: learning rate (small positive constant, e.g., 0.1)

Examples

- **x** is correctly classified, *o***x**–*o***x**=0

→ no update

- *o***x**=-1 but *t***x**=1, t**x**-*o***x**>0

→ wi is increased if xi>0, decreased otherwise

→**w**.**x** is increased

- *o***x**=1, but *out***x**=-1, *out***x**-*o***x**<0

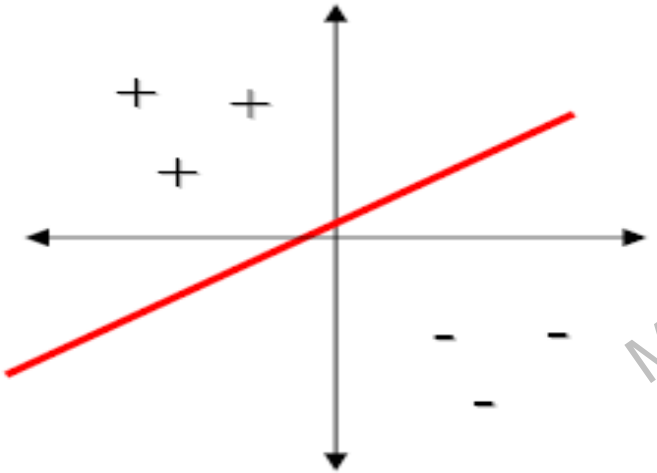
→wi is decreased if xi>0, increased otherwise

→**w**.**x** is decreased

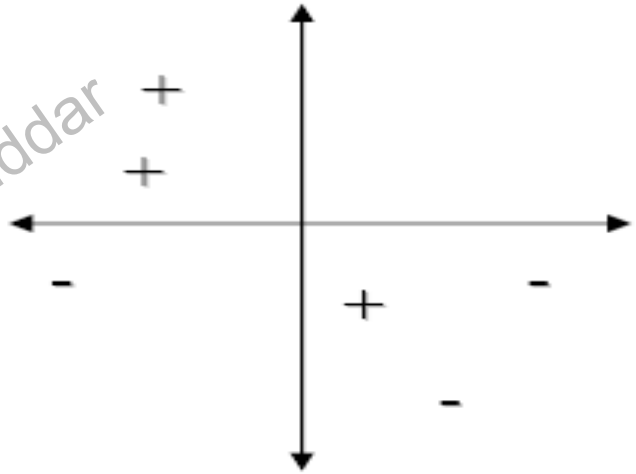
Representational Power of Perceptron's

- We can view the perceptron as representing a hyperplane decision surface in the n-dimensional space of instances (i.e., points).
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side.
- The equation for this decision hyperplane $\vec{w}.\vec{x} = 0$.
- Of course, some sets of positive and negative examples cannot be separated by any hyperplane.
- Those that can be separated are called linearly separable sets of examples.

Representational Power of Perceptron's



Linearly separable



Non-linearly separable

Representational Power of Perceptron's

- Perceptrons can represent all of the primitive boolean functions AND, OR, NAND, and NOR.
- Unfortunately, however, some boolean functions cannot be represented by a single perceptron, such as the XOR function whose value is 1 if and only if **xl != x2.**
- Note the set of linearly nonseparable training examples shown in Figure (b) corresponds to this XOR function.

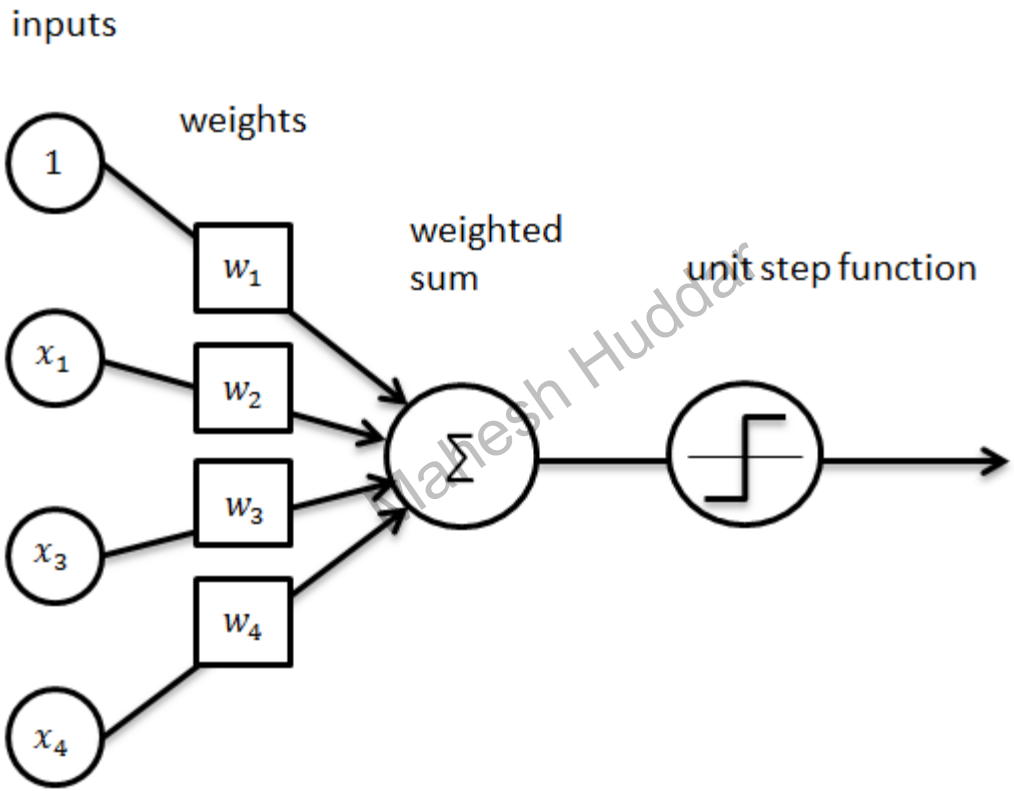
Representational Power of Perceptron's

- The ability of perceptrons to represent AND, OR, NAND, and NOR is important because ***every*** boolean function can be represented by some network of interconnected units based on these primitives.
- In fact, every boolean function can be represented by some network of perceptrons only two levels deep, in which the inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage.
- One way is to represent the boolean function in disjunctive normal form (i.e., as the disjunction (OR) of a set of conjunctions (ANDs) of the inputs and their negations).
- Note that the input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight.
- Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

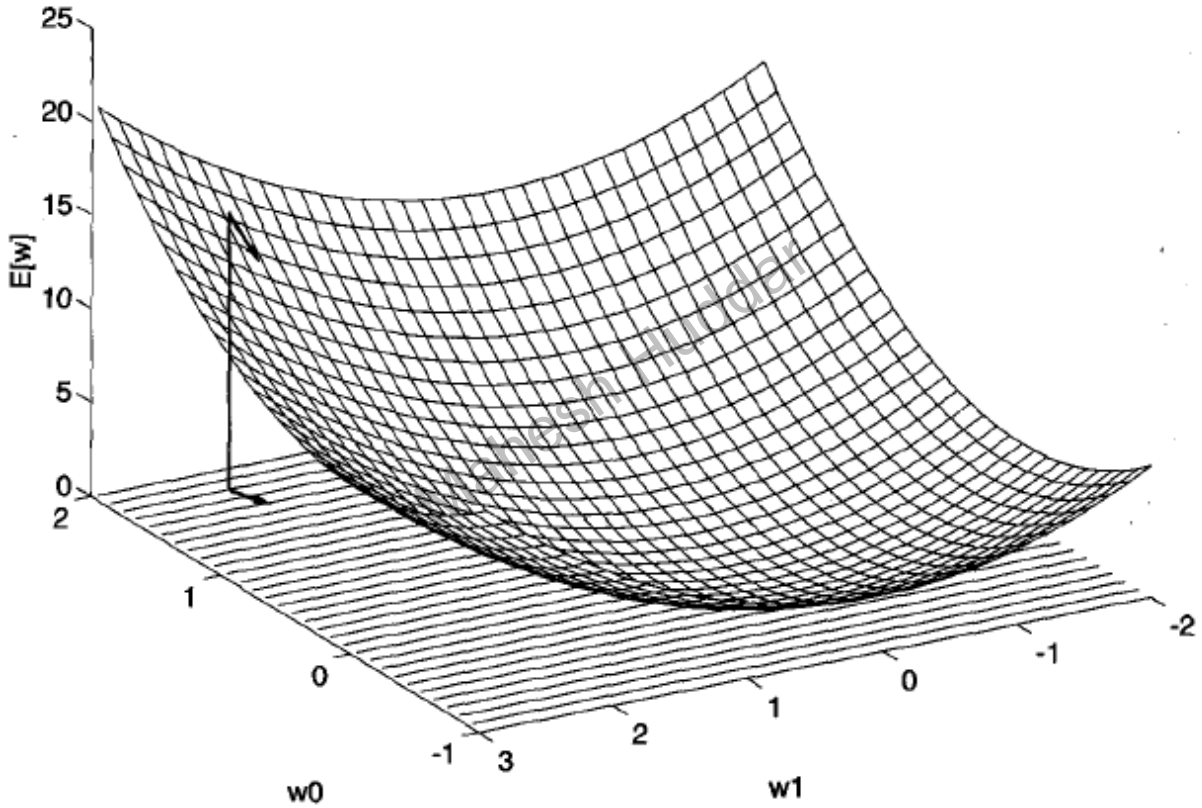
Gradient Descent and the Delta Rule

- Although the perceptron rule finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable.
- **A** second training rule, called the ***delta rule,*** is designed to overcome this difficulty.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use ***gradient descent*** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- This rule is important because gradient descent provides the basis for the BACKPROPAGATON algorithm, which can learn networks with many interconnected units.
- It is also important because gradient descent can serve as the basis for learning algorithms that must search through hypothesis spaces containing many different types of continuously parameterized hypotheses.

Gradient Descent and the Delta Rule

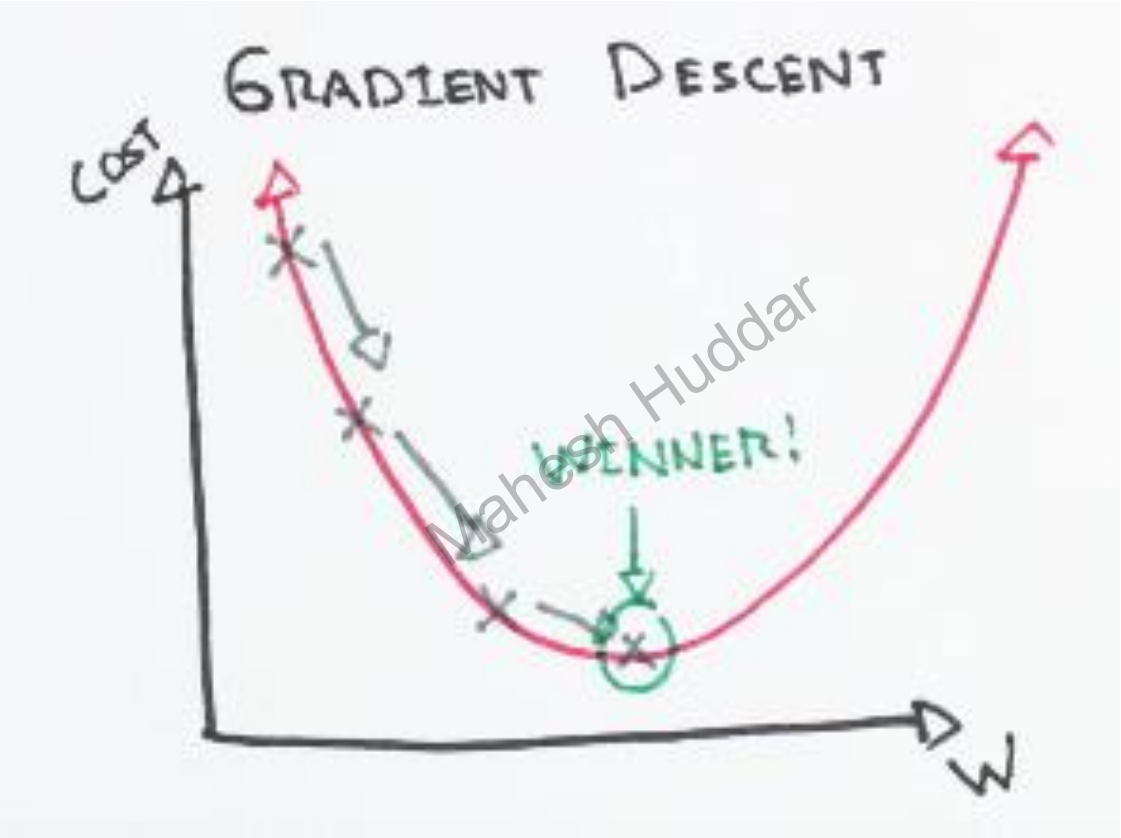


Gradient Descent and the Delta Rule



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Gradient Descent and the Delta Rule



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Gradient Descent and the Delta Rule

- The delta training rule is best understood by considering the task of training an ***unthresholded*** perceptron; that is, a ***linear unit*** for which the output ***o*** is given by

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- Thus, a linear unit corresponds to the first stage of a perceptron, without the threshold.
- In order to derive a weight learning rule for linear units, let us begin by specifying a measure for the ***training error*** of a hypothesis (weight vector), relative to the training examples.
- Although there are many ways to define this error, one common measure is

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- where D is the set of training examples, ***td*** is the target output for training example *d*, and ***od*** is the output of the linear unit for training example d.

Derivation of Gradient Descent Rule

- How can we calculate the direction of steepest descent along the error surface?
- This direction can be found by computing the derivative of E with respect to each component of the vector $\vec{w}$.
- This vector derivative is called the ***gradient*** of E with respect to $\vec{w}$, written as $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

- Since the gradient specifies the direction of steepest increase of E, the training rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

$$\Delta w_i = \eta(t - o)x_i$$

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

Derivation of Gradient Descent Rule

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

- Here *n* is a positive constant called the learning rate, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that ***decreases*** E.
- This training rule can also be written in its component form

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n}\right]$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Derivation of Gradient Descent Rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id})$$

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

$$w_i \leftarrow w_i + \Delta w_i$$

Gradient-Descent$(training\_examples, \eta)$

Each training example is a pair of the form $\langle \vec{x}, t \rangle$*, where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i := \Delta w_i + \eta(t - o)x_i$$

  - For each linear unit weight w_i, Do

$$w_i := w_i + \Delta w_i$$

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

- Gradient descent is an important general paradigm for learning.
- It is a strategy for searching through a large or infinite hypothesis space that can be applied whenever
  1. the hypothesis space contains continuously parameterized hypotheses (e.g., the weights in a linear unit), and
  2. the error can be differentiated with respect to these hypothesis parameters.
- The key practical difficulties in applying gradient descent are
  1. converging to a local minimum can sometimes be quite slow (i.e., it can require many thousands of gradient descent steps), and
  2. if there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

Stochastic Gradient-Descent$(training\_examples, \eta)$

Each training example is a pair of the form $\langle \vec{x}, t \rangle$*, where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$w_i := w_i + \eta(t - o)x_i$$

STOCHASTIC APPROXIMATION TO GRADIENT DESCENT

The key differences between standard gradient descent and stochastic gradient descent are:

- In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.
- Summing over multiple examples in standard gradient descent requires more computation per weight update step. On the other hand, because it uses the true gradient, standard gradient descent is often used with a larger step size per weight update than stochastic gradient descent.
- In cases where there are multiple local minima with respect to $E(\vec{w})$, stochastic gradient descent can sometimes avoid falling into these local minima because it uses the various $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search.

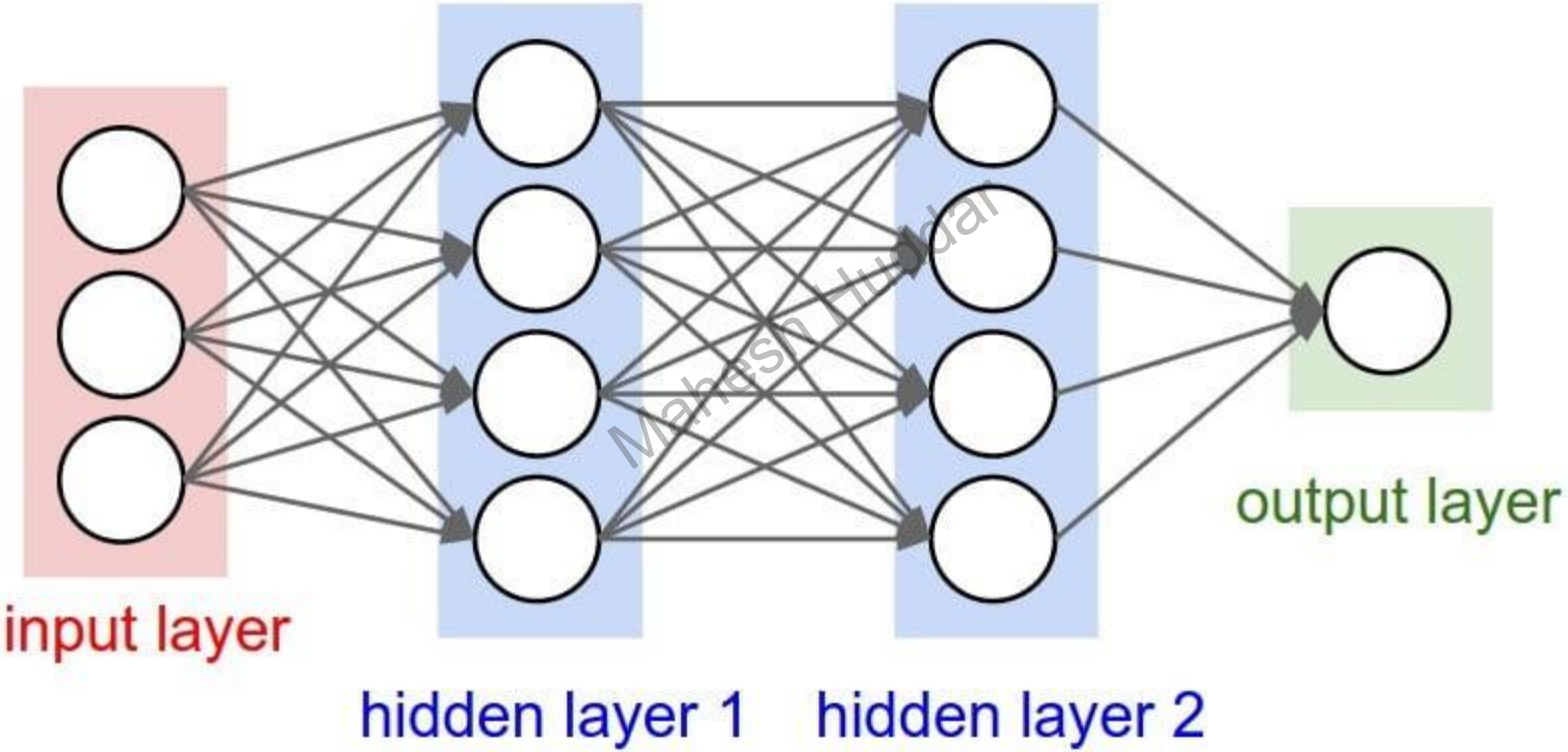
Gradient Descent vs Stochastic Gradient Descent

Standard Gradient Descent	Stochastic Gradient Descent
Error is summed over all examples before updating weights	Weights are updated examining each training example
Summing over multiple examples require more computation per weight update step	Less computation as individual weights are updated
Difficult when there are multiple local minimum	Uses various Ed (w) rather than E(w), hence handles multiple local minima in the ease.

MULTILAYER NETWORKS

- Multilayer neural networks can classify a range of functions, including non linearly separable ones.
- Each input layer neuron connects to all neurons in the hidden layer.
- The neurons in the hidden layer connect to all neurons in the output layer.

MULTILAYER NETWORKS



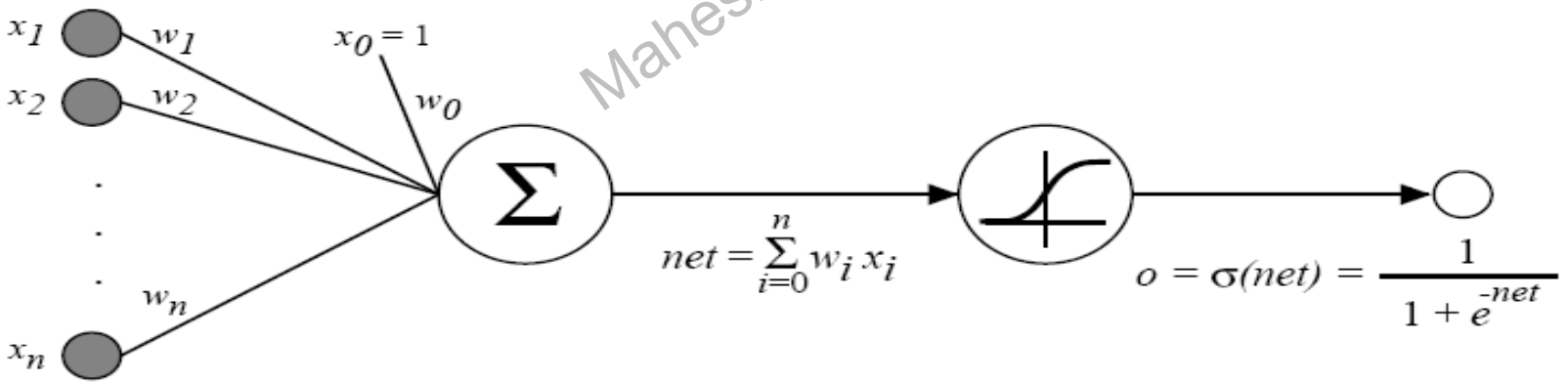
MULTILAYER NETWORKS

A Differentiable Threshold Unit

- What type of unit shall we use as the basis for constructing multilayer networks?
- At first we might be tempted to choose the linear units discussed in the previous section, for which we have already derived a gradient descent learning rule.
- However, multiple layers of cascaded linear units still produce only linear functions, and we prefer networks capable of representing highly nonlinear functions.
- The perceptron unit is another possible choice, but its discontinuous threshold makes it undifferentiable and hence unsuitable for gradient descent.
- What we need is a unit whose output is a nonlinear function of its inputs, but whose output is also a differentiable function of its inputs.
- One solution is the sigmoid unit-a unit very much like a perceptron, but based on a smoothed, differentiable threshold function.

MULTILAYER NETWORKS

- The sigmoid unit is illustrated in below Figure. Like the perceptron, the sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result.
- In the case of the sigmoid unit, however, the threshold output is a continuous function of its input.



MULTILAYER NETWORKS

More precisely, the sigmoid unit computes its output ***o*** as

$$o = \sigma(\vec{w} \cdot \vec{x})$$

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

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

The BACKPROPAGATIAON Algorithm

- Multilayer neural networks learn in the same way as perceptrons.
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights.
- *E* sums the errors over all of the network output units

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

The BACKPROPAGATIAON Algorithm

x_{ji} = the ith input to unit j

w_{ji} = the weight associated with the ith input to unit j

$net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)

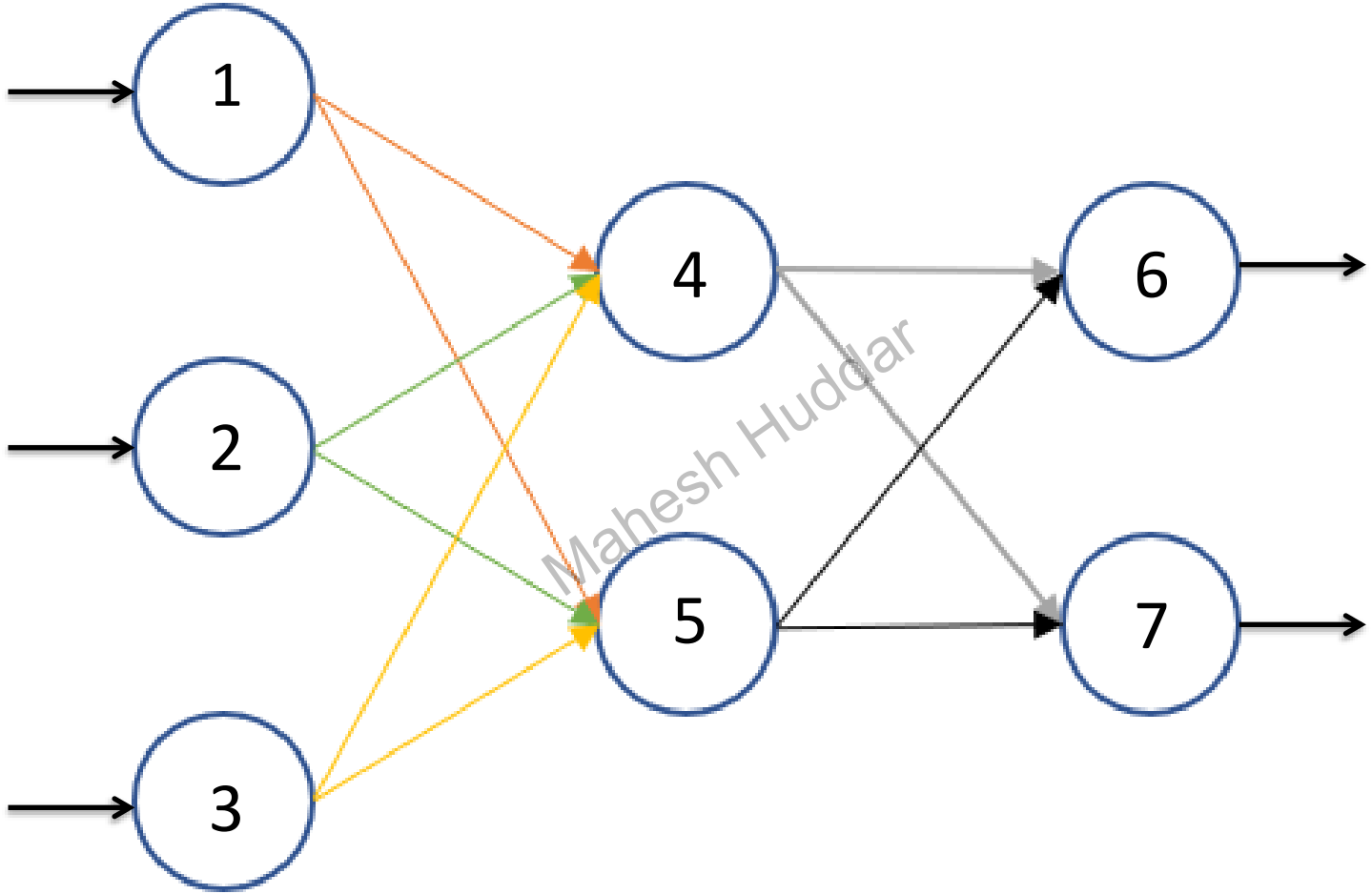
o_j = the output computed by unit j

t_j = the target output for unit j

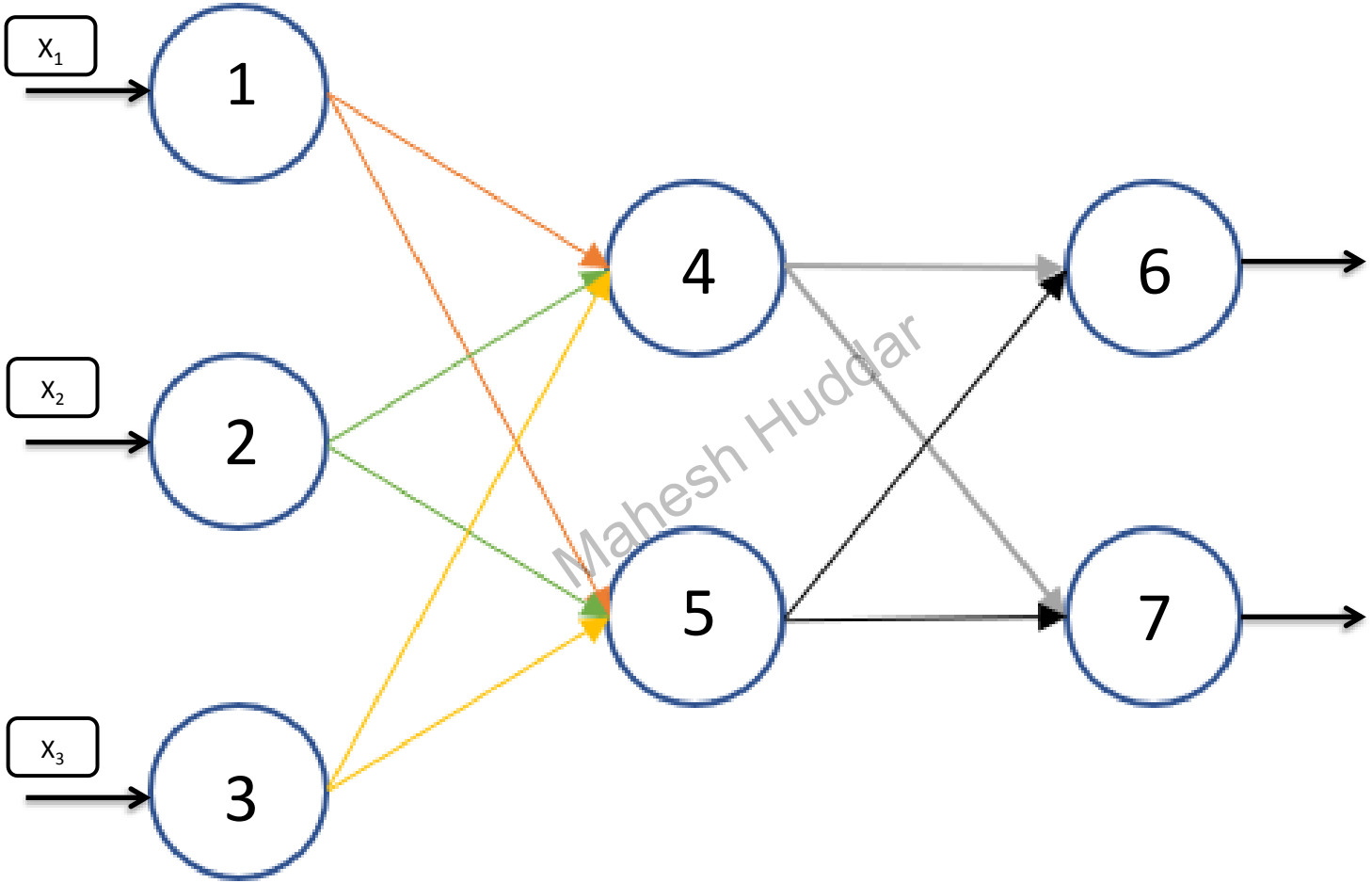
σ = the sigmoid function

$outputs$ = the set of units in the final layer of the network

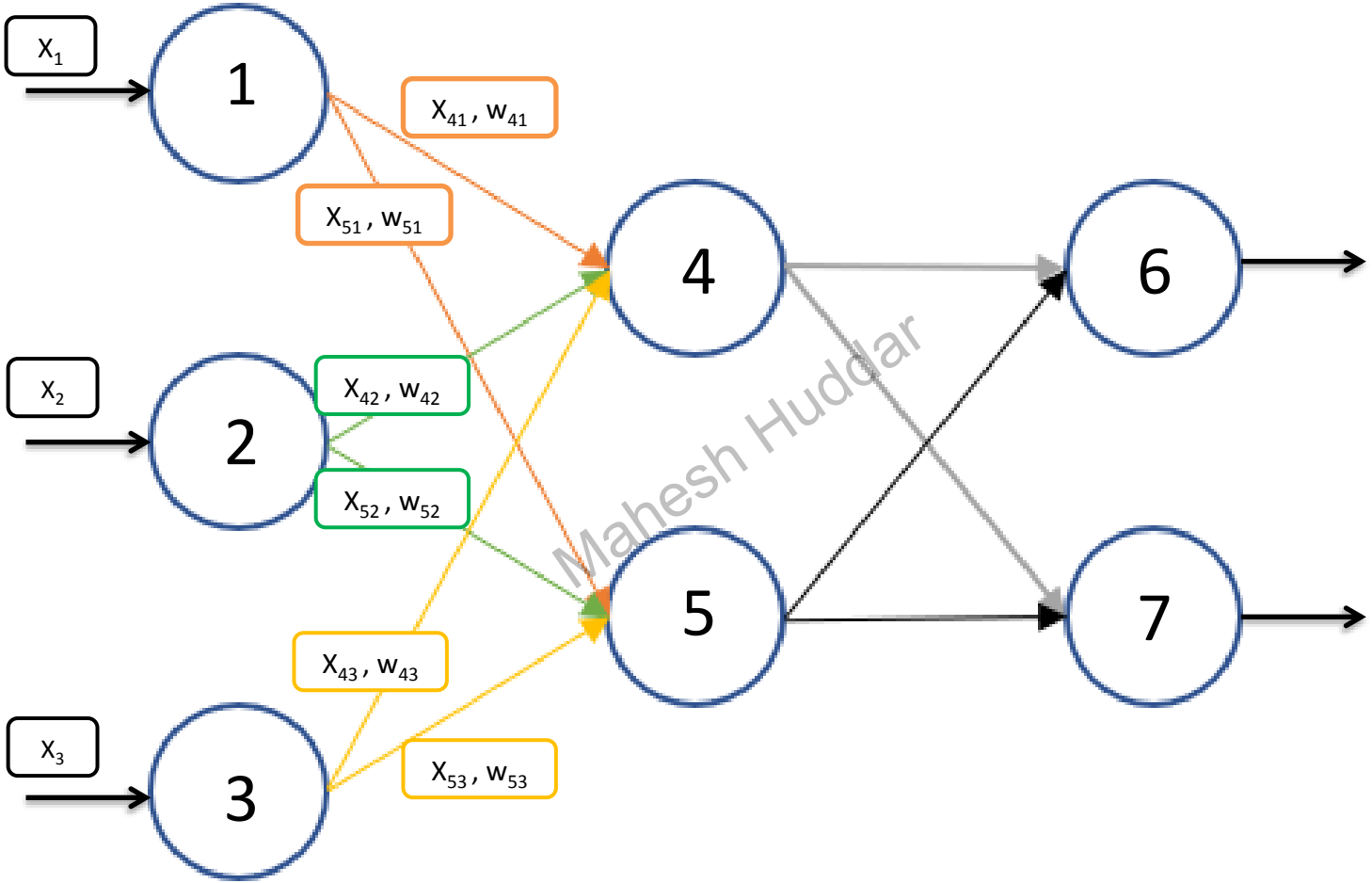
$Downstream(j)$ = the set of units whose immediate inputs include the output of unit j



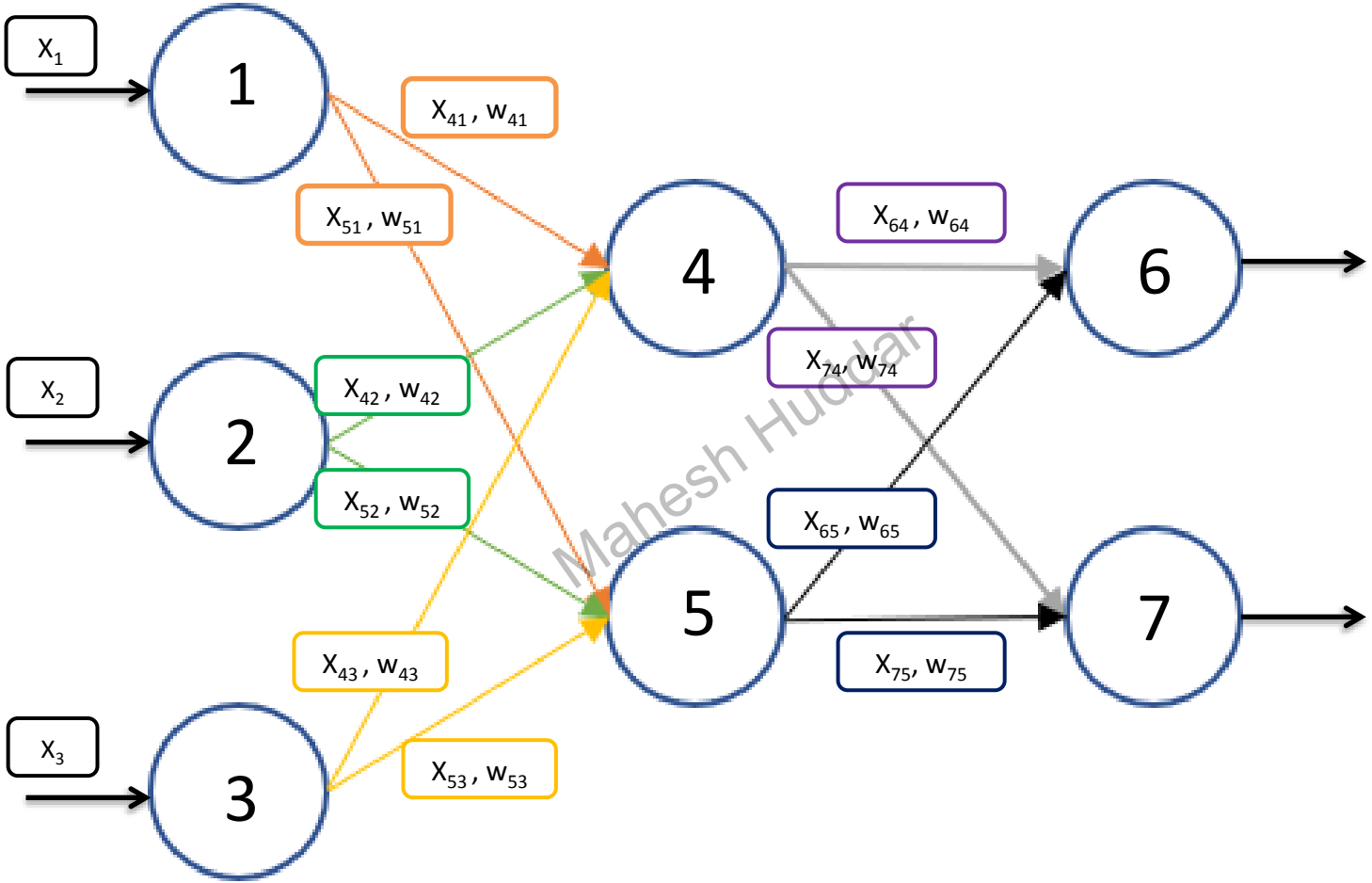
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



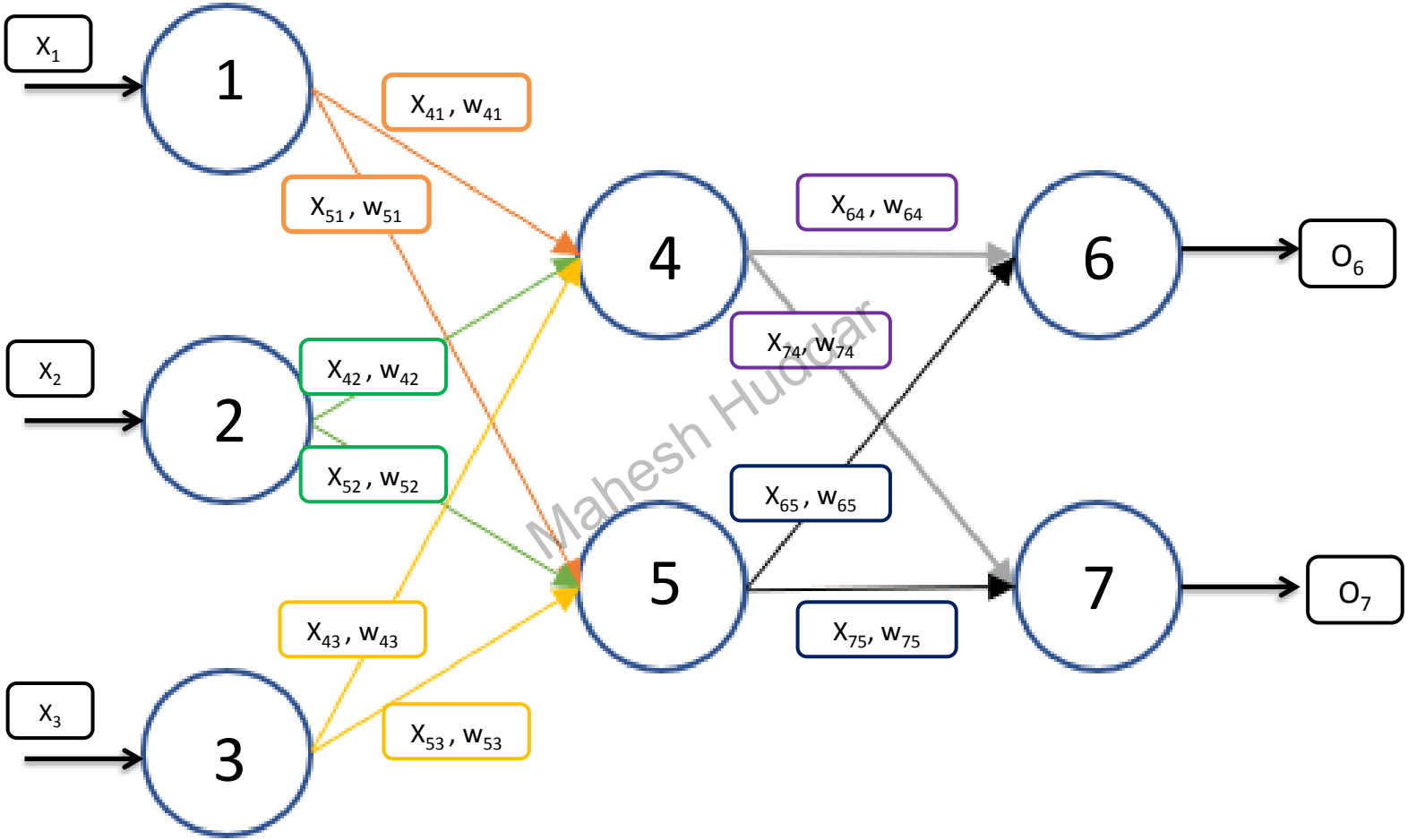
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



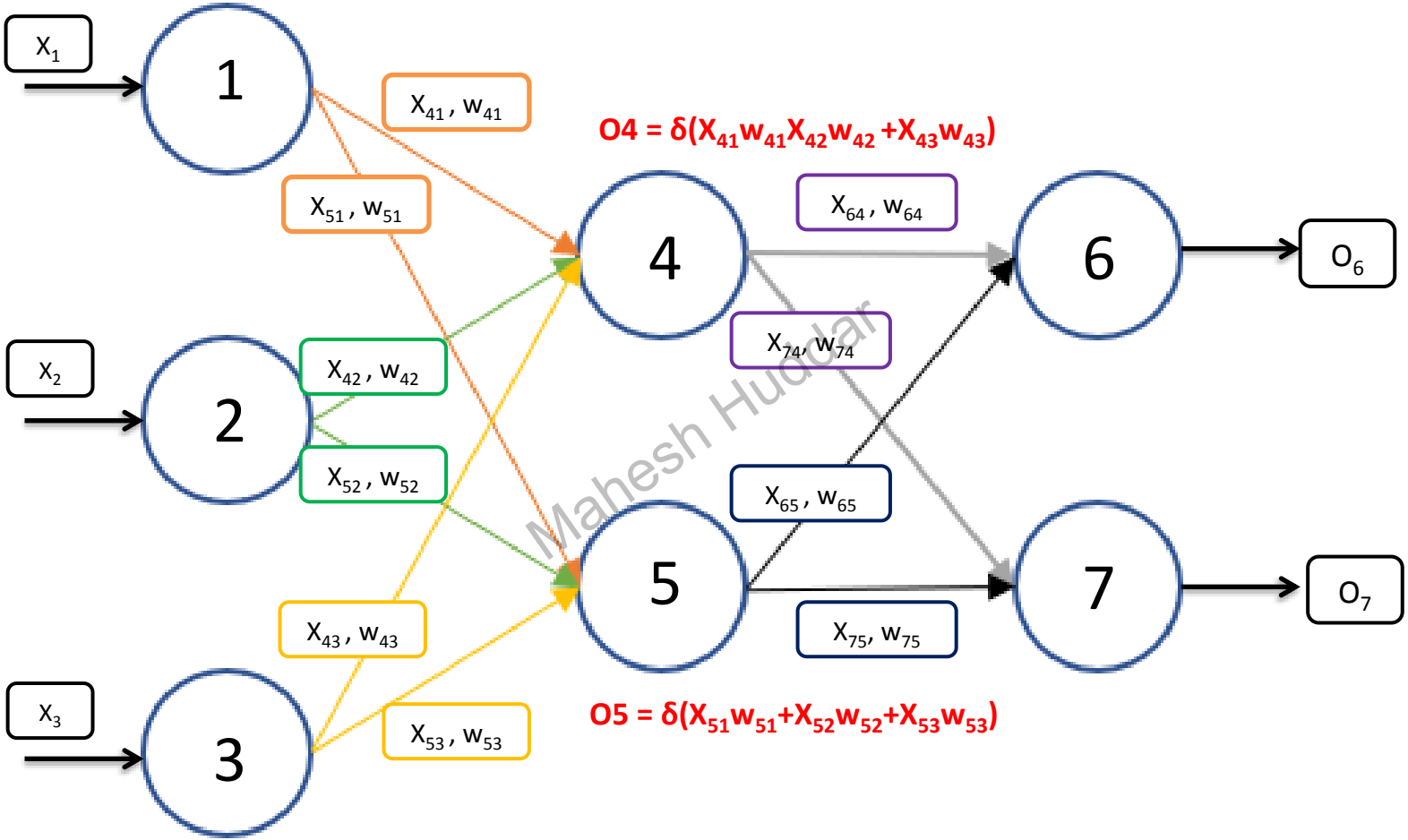
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



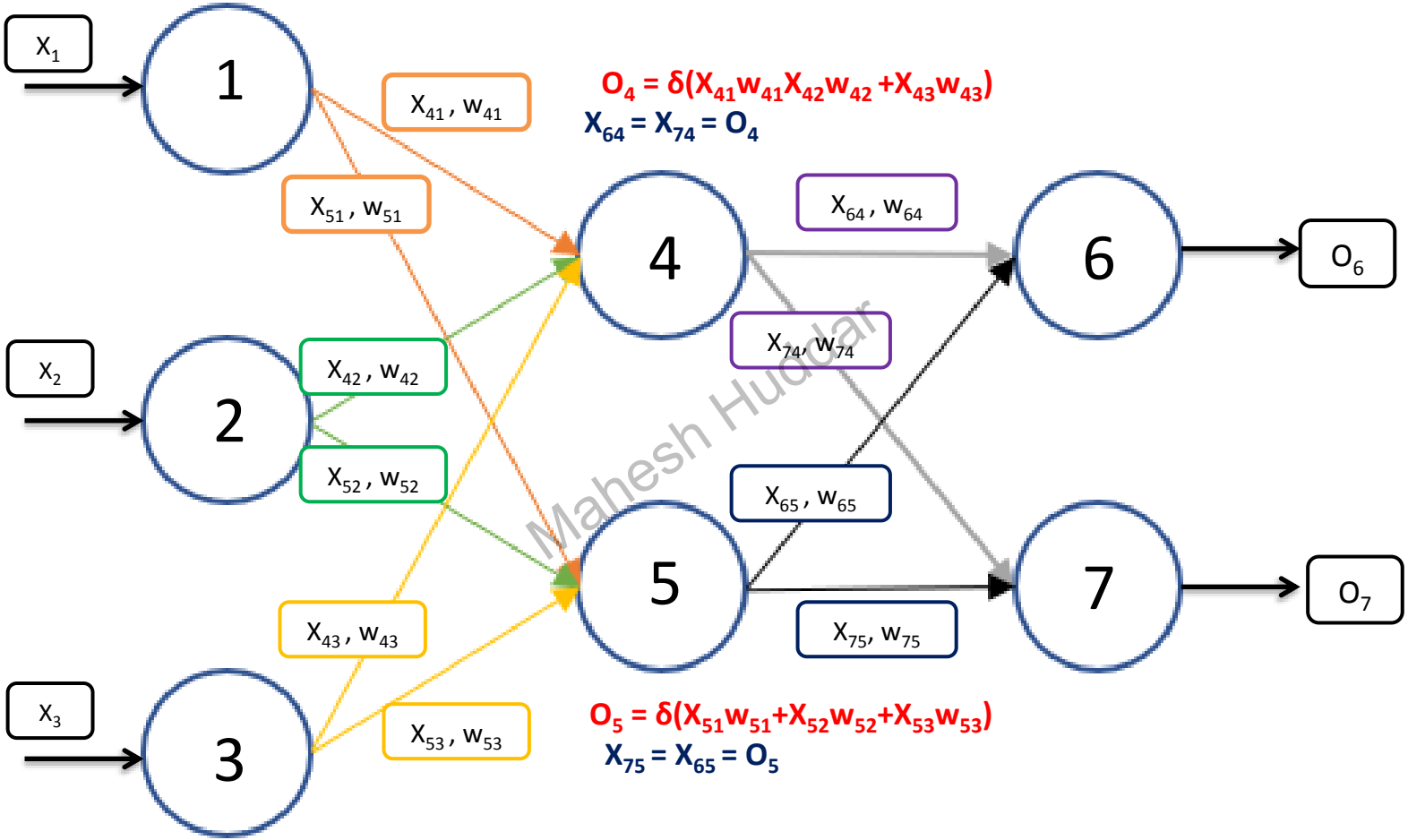
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



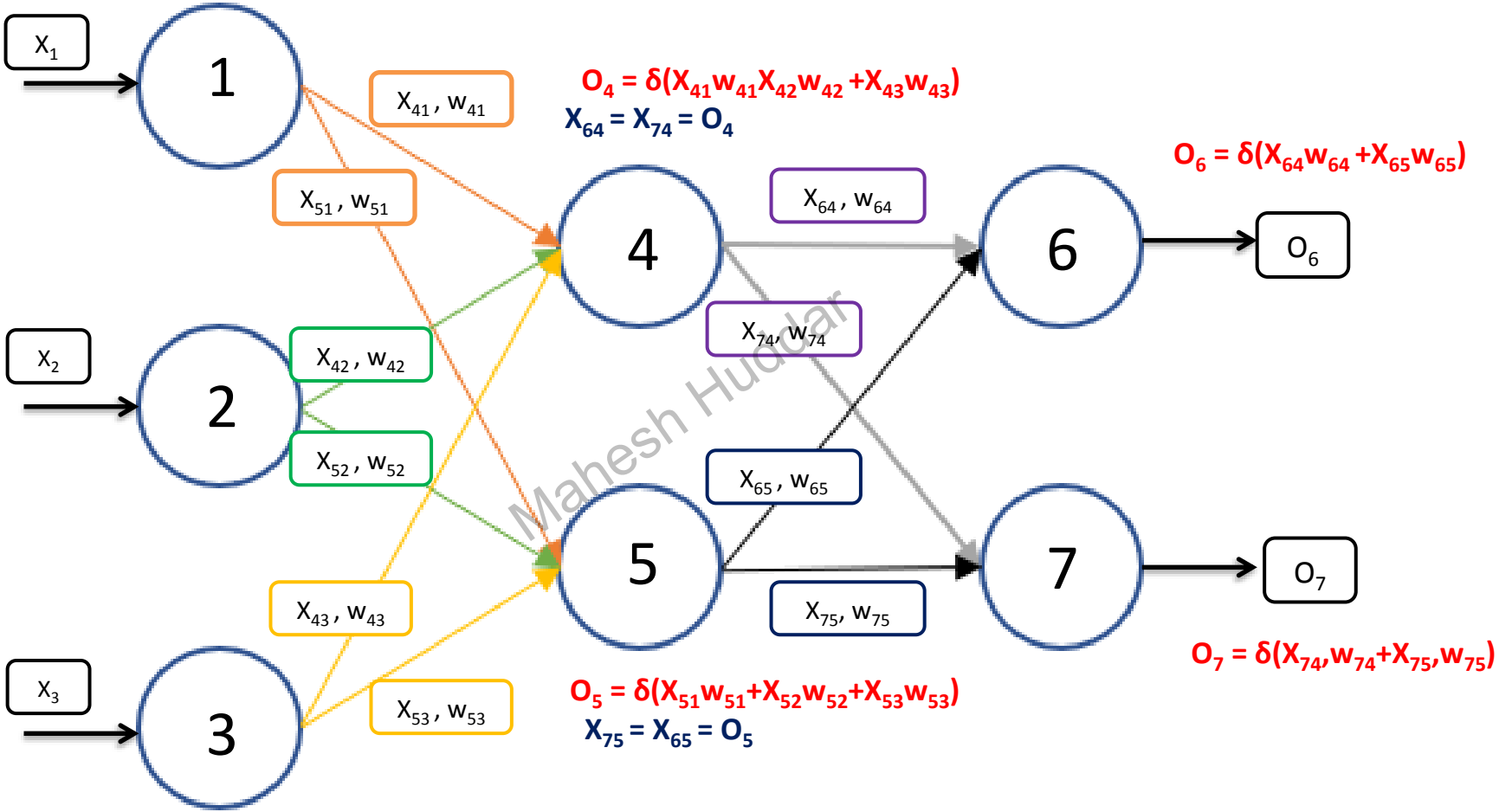
Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Back Propagation Algorithm

- 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

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

Derivation of Back Propagation Algorithm

- To derive the equation for updating weights in back propagation algorithm, we use Stochastic gradient descent rule.
- Stochastic gradient descent involves iterating through the training examples one at a time, for each training example **d** descending the gradient of the error ***Ed*** with respect to this single example.
- In other words, for each training example **d** every weight ***wji*** is updated by adding to it Δw_{ij}.
- That is,

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

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

Derivation of Back Propagation Algorithm

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

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- where ***Ed*** is the error on training example **d**, that is half the squared difference between the target output and the actual output over all output units in the network,

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- Here outputs is the set of output units in the network, $\mathbf{t_k}$ is the target value of unit k for training example **d**, and $\mathbf{o_k}$ is the output of unit k given training example **d**.

Derivation of Back Propagation Algorithm

Notation Used:

x_{ji} = the ith input to unit j

w_{ji} = the weight associated with the ith input to unit j

$net_j = \sum_i w_{ji} X_{ji}$ (the weighted sum of inputs for unit j)

o_j = the output computed by unit j

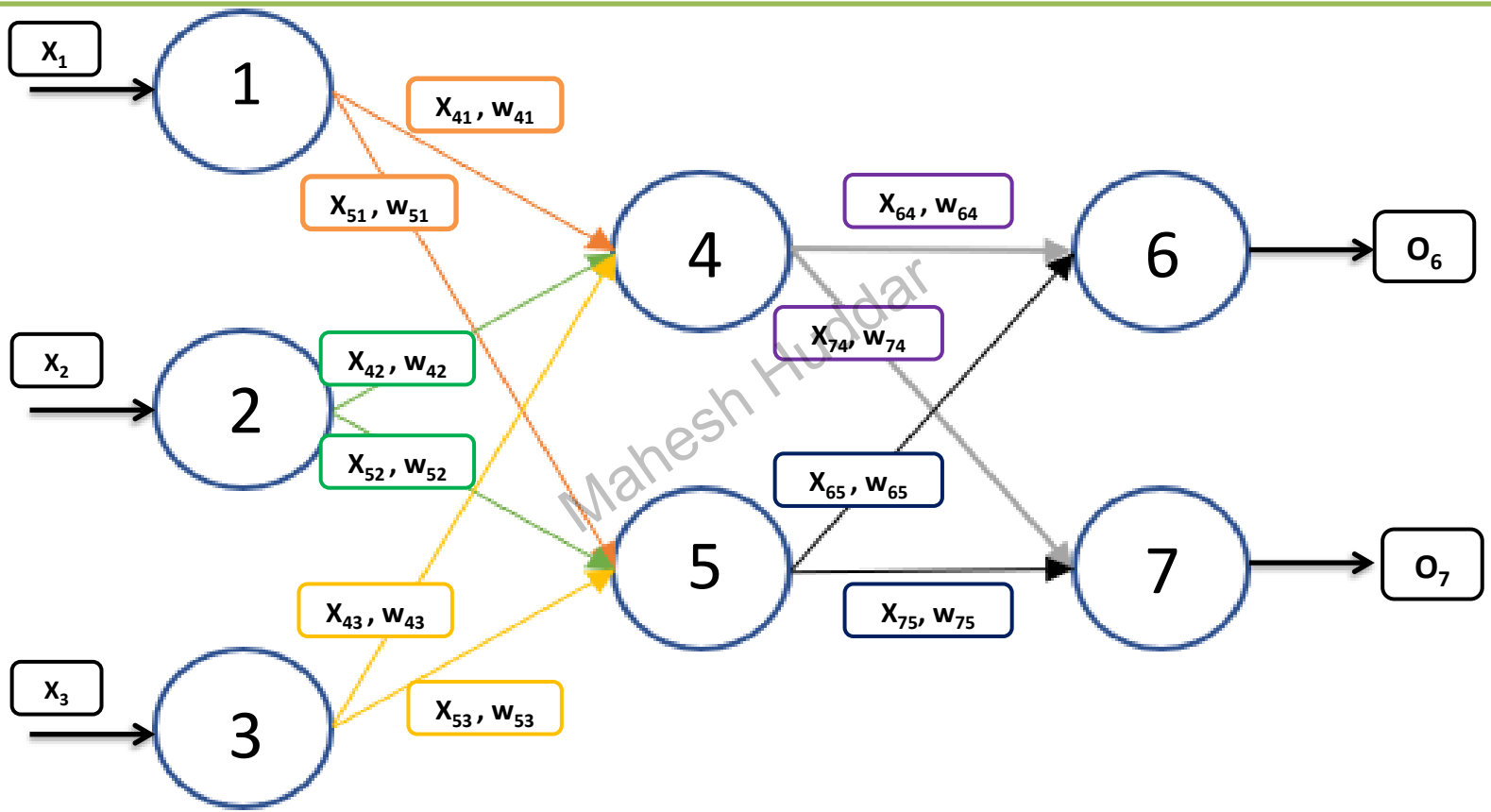
t_j = the target output for unit j

σ = the sigmoid function

outputs = the set of units in the final layer of the network

Downstream(j) = the set of units whose immediate inputs include the output of unit j

Derivation of Back Propagation Algorithm



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Derivation of Back Propagation Algorithm

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

Where

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- To begin, notice that weight ***wji*** can influence the rest of the network only through ***netj.*** Therefore, we can use the chain rule to write,

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$net_j = \sum_i w_{ji} X_{ji}$$

$$\frac{\partial net_j}{\partial w_{ji}} = x_{ji}$$

- Our remaining task is to derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

Derivation of Back Propagation Algorithm

To derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$

We consider two cases in turn:

- Case 1, where unit j is an output unit for the network, and
- Case 2, where unit j is an internal unit of the network.

Derivation of Back Propagation Algorithm

Case 1: Training Rule for Output Unit Weights

- Just as ***wji*** can influence the rest of the network only through net_j, net_j can influence the network only through ***oj.*** Therefore, we can invoke the chain rule again to write,

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}2(t_j - o_j)\frac{\partial(t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

$$\frac{\partial o_j}{\partial (net_j)} = \frac{\partial \sigma(net_j)}{\partial (net_j)}$$

$$= \sigma(net_j)\,(1 - \sigma(net_j))$$

$$= o_j\,(1 - o_j)$$

$$\frac{\partial \sigma(x)}{\partial (x)} = \sigma(x)\,(1 - \sigma(x))$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\,o_j(1 - o_j)$$

Derivation of Back Propagation Algorithm

Case 1: Training Rule for Output Unit Weights

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

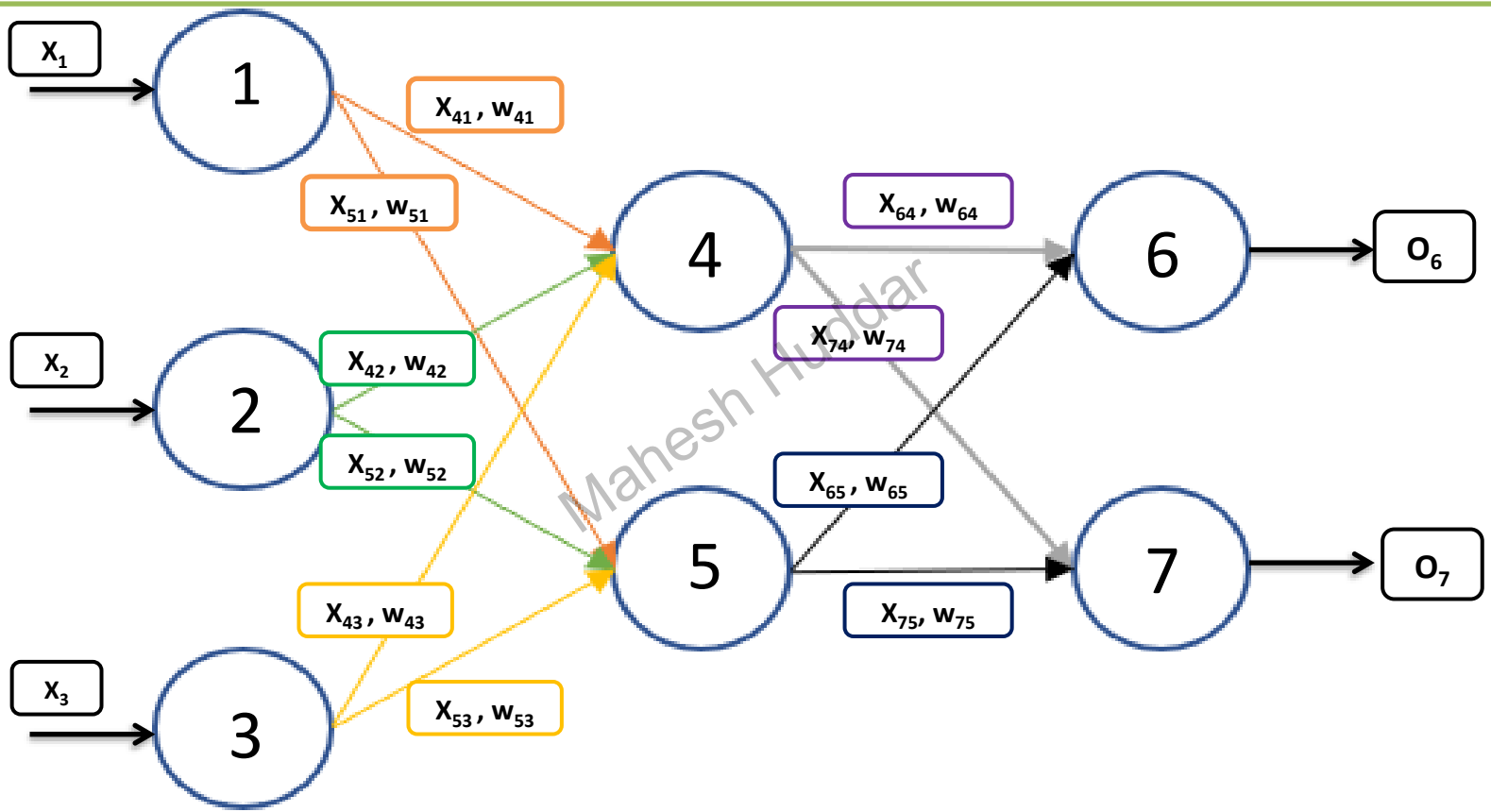
$$\Delta w_{ji} = \eta (t_j - o_j) o_j(1 - o_j) x_{ji}$$

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

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j(1 - o_j)$$

$$\delta_j = (t_j - o_j) o_j(1 - o_j)$$

Derivation of Back Propagation Algorithm



Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar

Derivation of Back Propagation Algorithm

Case 2: Training Rule for Hidden Unit Weights

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) \, o_j(1 - o_j)$$

$$\delta_j = (t_j - o_j) \, o_j(1 - o_j)$$

$$\frac{\partial net_k}{\partial o_j} = \frac{\partial x_{kj} w_{kj}}{\partial o_j} = \frac{\partial o_j w_{kj}}{\partial o_j}$$

$$\frac{\partial o_j}{\partial (net_j)} = \frac{\partial \sigma(net_j)}{\partial (net_j)}$$

$$= \sigma(net_j)\,(1 - \sigma(net_j))$$

$$= o_j \, (1 - o_j)$$

Derivation of Back Propagation Algorithm

Case 2: Training Rule for Hidden Unit Weights

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \; o_j(1 - o_j)$$

$$\Delta w_{ji} = \eta \; o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \; w_{kj} \; x_{ji}$$

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

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \; w_{kj}$$

Neural Network Representations - ALVINN

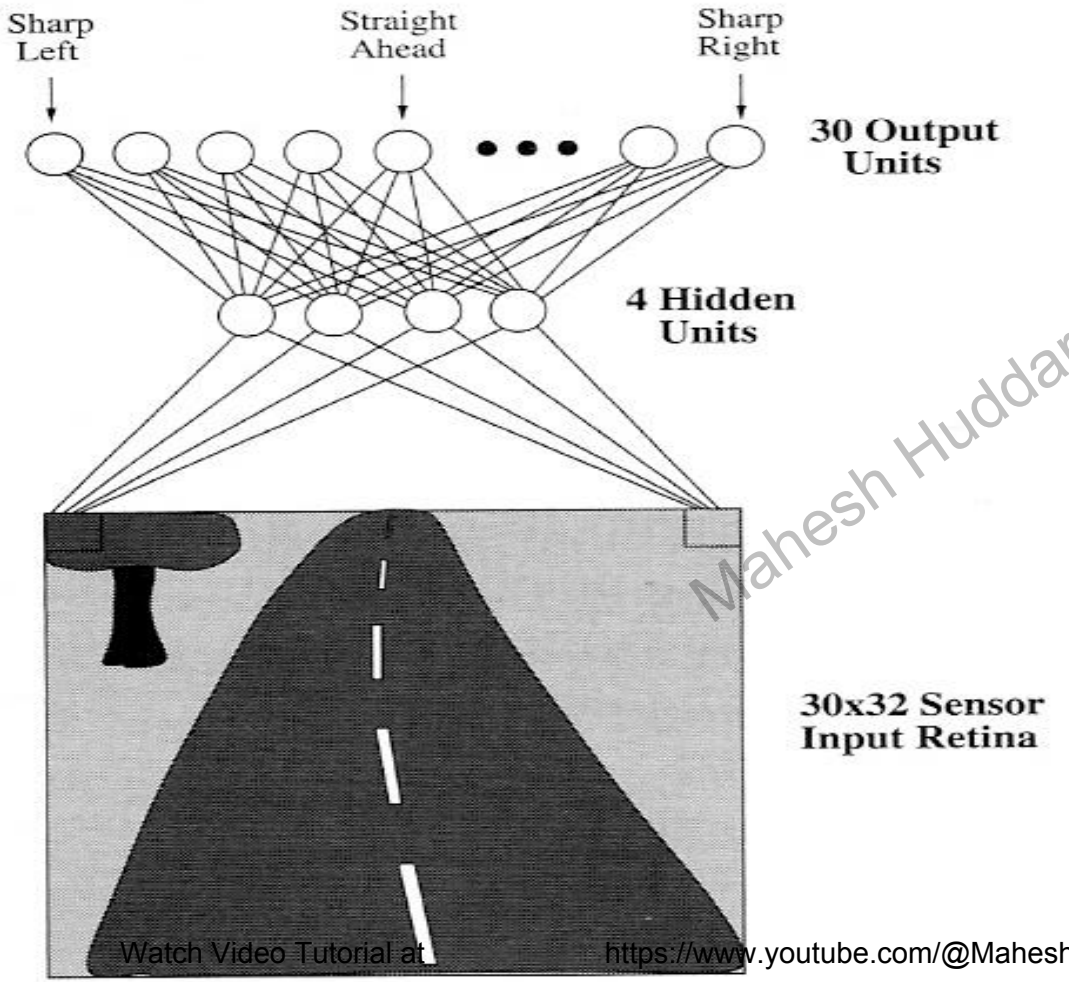
- **A** prototypical example of **ANN** learning which uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways.
- The input to the neural network is a 30 x 32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle.
- The network output is the direction in which the vehicle is steered. The ANN is trained to mimic the observed steering commands of a human driving the vehicle for approximately 5 minutes.
- ALVINN has used its learned networks to successfully drive at speeds up to 70 miles per hour and for distances of 90 miles on public highways

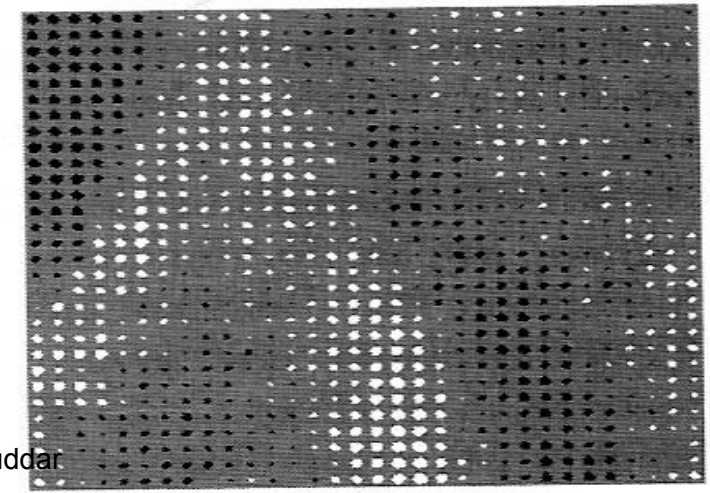
Neural Network Representations - ALVINN

- Figure illustrates the neural network representation used in one version of the ALVINN system, and illustrates the kind of representation typical of many ANN systems.
- The network is shown on the left side of the figure, with the input camera image depicted below it. Each node (i.e., circle) in the network diagram corresponds to the output of a single network ***unit,*** and the lines entering the node from below are its inputs.
- As can be seen, there are four units that receive inputs directly from all of the 30 x 32 pixels in the image. These are called "hidden“ units because their output is available only within the network and is not available as part of the global network output.
- Each of these four hidden units computes a single real-valued output based on a weighted combination of its 960 inputs.
- These hidden unit outputs are then used as inputs to a second layer of 30 "output" units.
- Each output unit corresponds to a particular steering direction, and the output values of these units determine which steering direction is recommended most strongly.

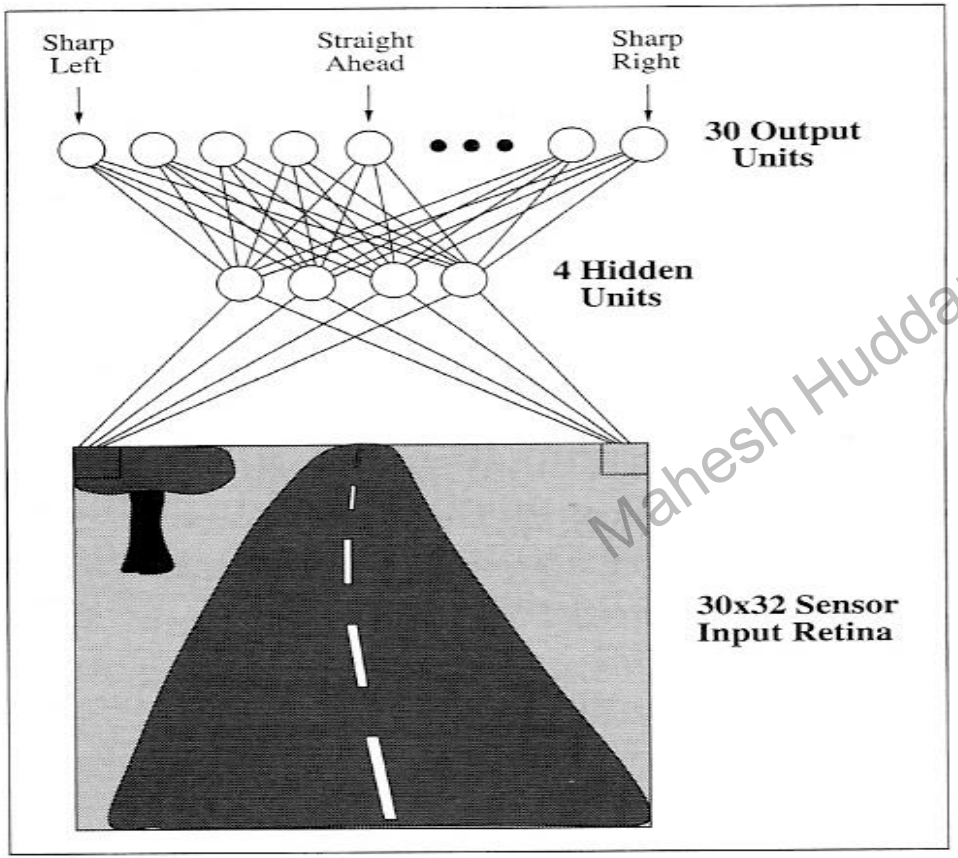
Neural Network Representations - ALVINN

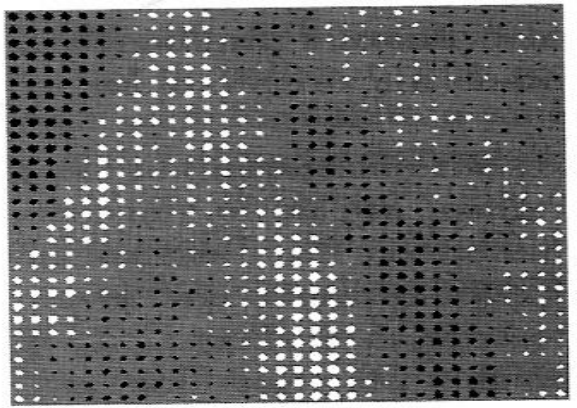
- The diagrams on the right side of the figure depict the learned weight values associated with one of the four hidden units in this ANN.
- The large matrix of black and white boxes on the lower right depicts the weights from the 30 x 32 pixel inputs into the hidden unit.
- Here, a white box indicates a positive weight, a black box a negative weight, and the size of the box indicates the weight magnitude.
- The smaller rectangular diagram directly above the large matrix shows the weights from this hidden unit to each of the 30 output units.





ALVINN





Watch Video Tutorial at https://www.youtube.com/@MaheshHuddar