Machine Learning and Computational Intelligence Lecture 9

Sanjeeb Prasad Panday, PhD

Associate Professor

Dept. of Electronics and Computer Engineering

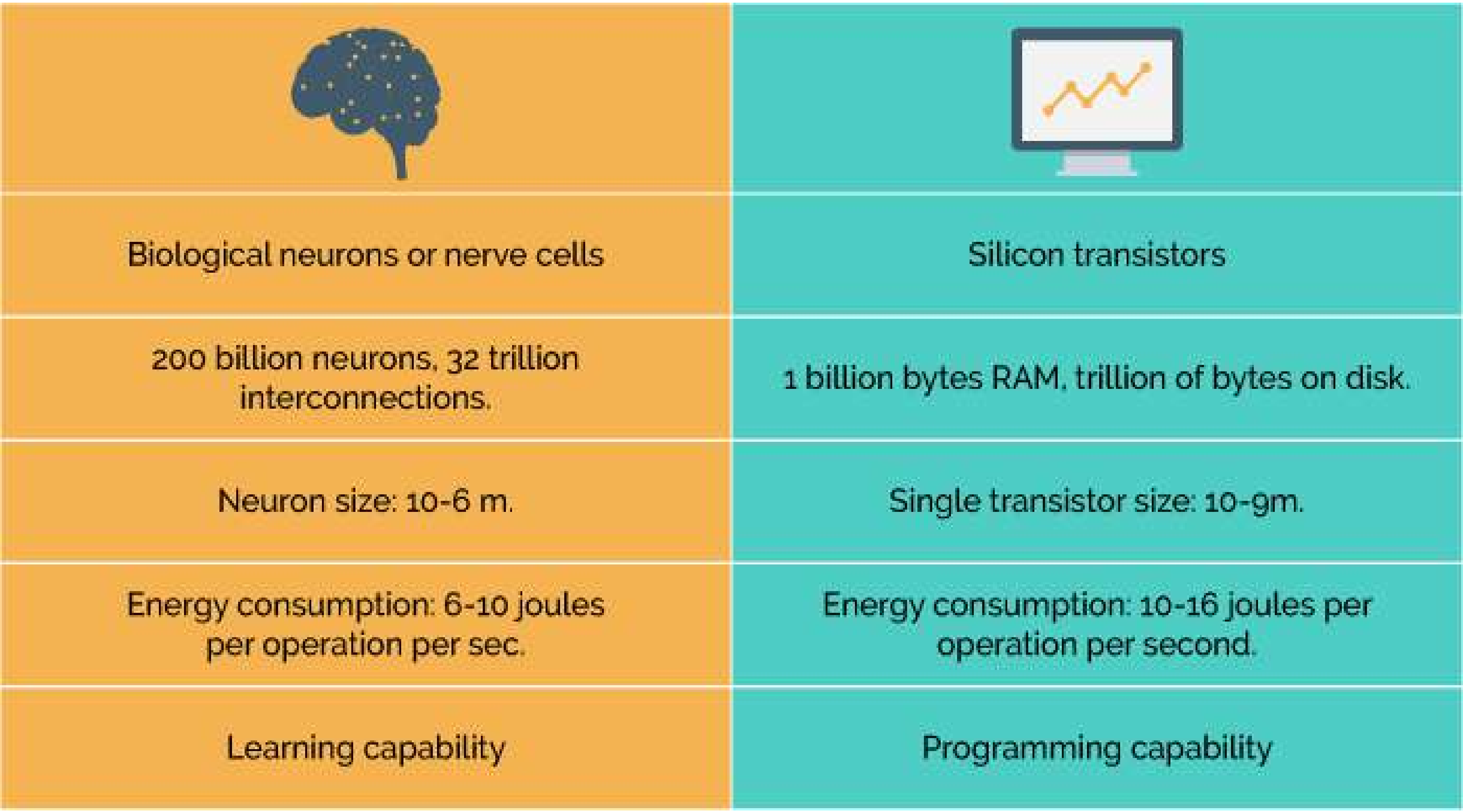
Director (ICTC)

IOE, TU

Human Brain

- Computers and the Brain: A Contrast
  - Arithmetic: 1 brain = 1/10 pocket calculator
  - Vision: 1 brain = 1000 super computers
  - Memory of arbitrary details: computer wins
  - Memory of real-world facts: brain wins
  - A computer must be programmed explicitly
  - The brain can learn by experiencing the world

Other Comparisons



Biological neurons or nerve cells	Silicon transistors
200 billion neurons, 32 trillion interconnections.	1 billion bytes RAM, trillion of bytes on disk.
Neuron size: 10-6 m.	Single transistor size: 10-9m.
Energy consumption: 6-10 joules per operation per sec.	Energy consumption: 10-16 joules per operation per second.
Learning capability	Programming capability

Computer Operations

Traditionally computers execute a sequence of instructions to accomplish a set task

This is a powerful technique if you know the ‘algorithm’

It’s not very useful if you don’t !!

There are many interesting tasks where the algorithm is either unknown or unclear

- Recognizing handwriting - Pattern recognition
- Playing table tennis - Interacting with the environment
- Balancing activities - Optimization

The Question

Humans find these tasks relatively simple

- We learn by example
- The brain is responsible for our ‘computing’ power

If a machine were constructed using the fundamental building blocks found in the *brain* could it *learn* to do ‘difficult’ tasks ???

Definition

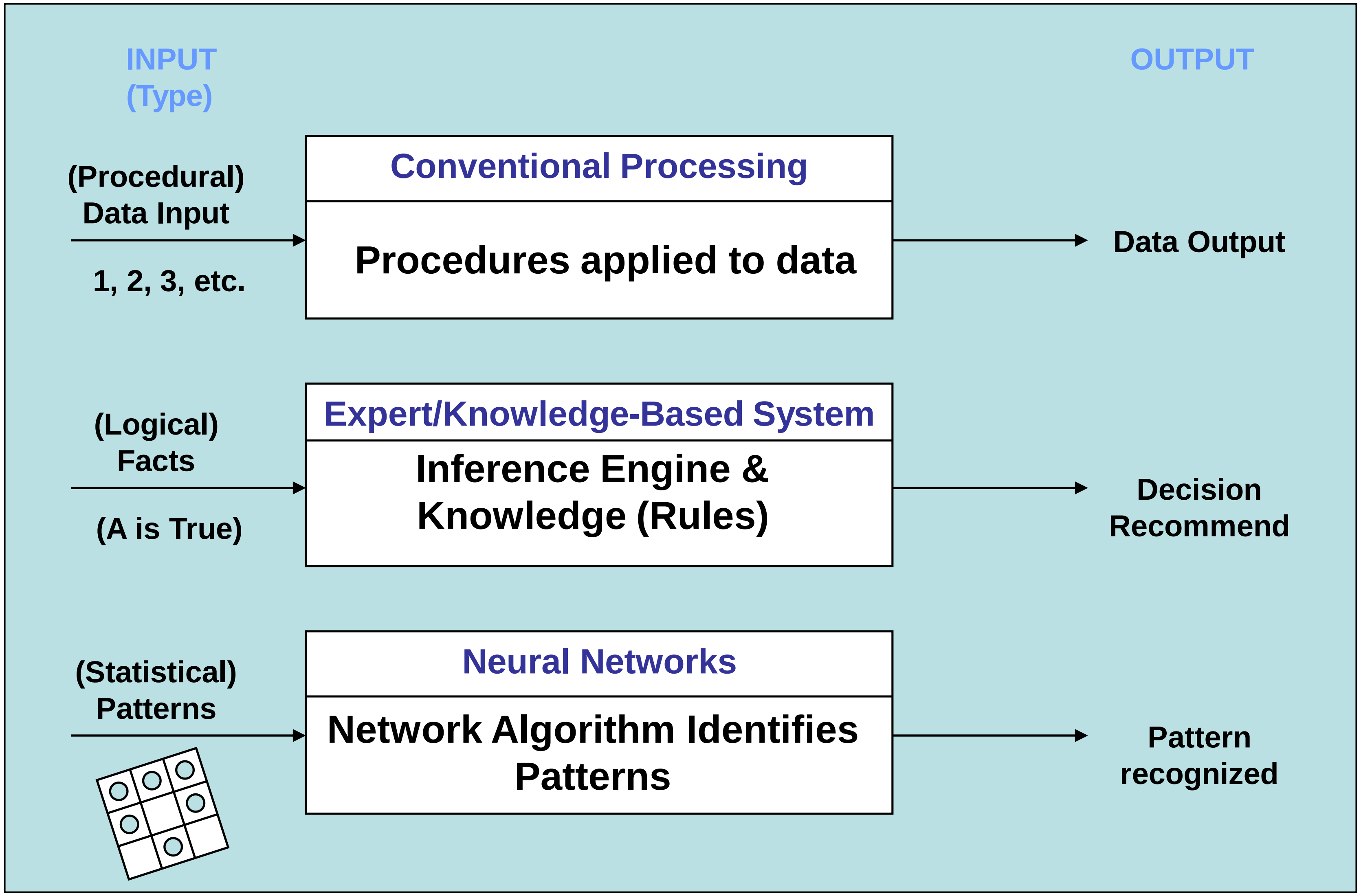
- “. . . Neural nets are basically mathematical models of information processing . . .”
- “. . . (neural nets) refer to machines that have a structure that, at some level, reflects what is known of the structure of the brain . . .”
- “A neural network is a massively parallel distributed processor . . . “

Neural Net Concept

- Artificial Neural Systems are called:
  - neurocomputers
  - neural networks
  - parallel distributed processors
  - connectionists systems

- Basic Philosophy
  - large number of simple “neuron-like” processors which execute global or distributed computation

Processing Comparisons



History of NN

- Creation:
  1890: William James - defined a neuronal process of learning
  1911: Ramon y Cajal (1911) introduced the idea of a neuron (brain cell)

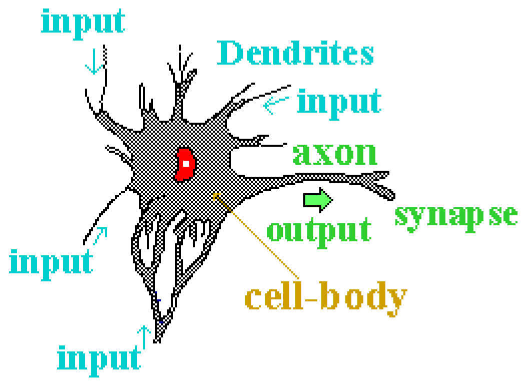
  Promising Technology:
  1943: McCulloch and Pitts - earliest mathematical models
  1954: Hebb and IBM research group - earliest simulations
  1958: Frank Rosenblatt - The Perceptron

- Disenchantment:
  1969: Minsky and Papert - perceptrons have severe limitations

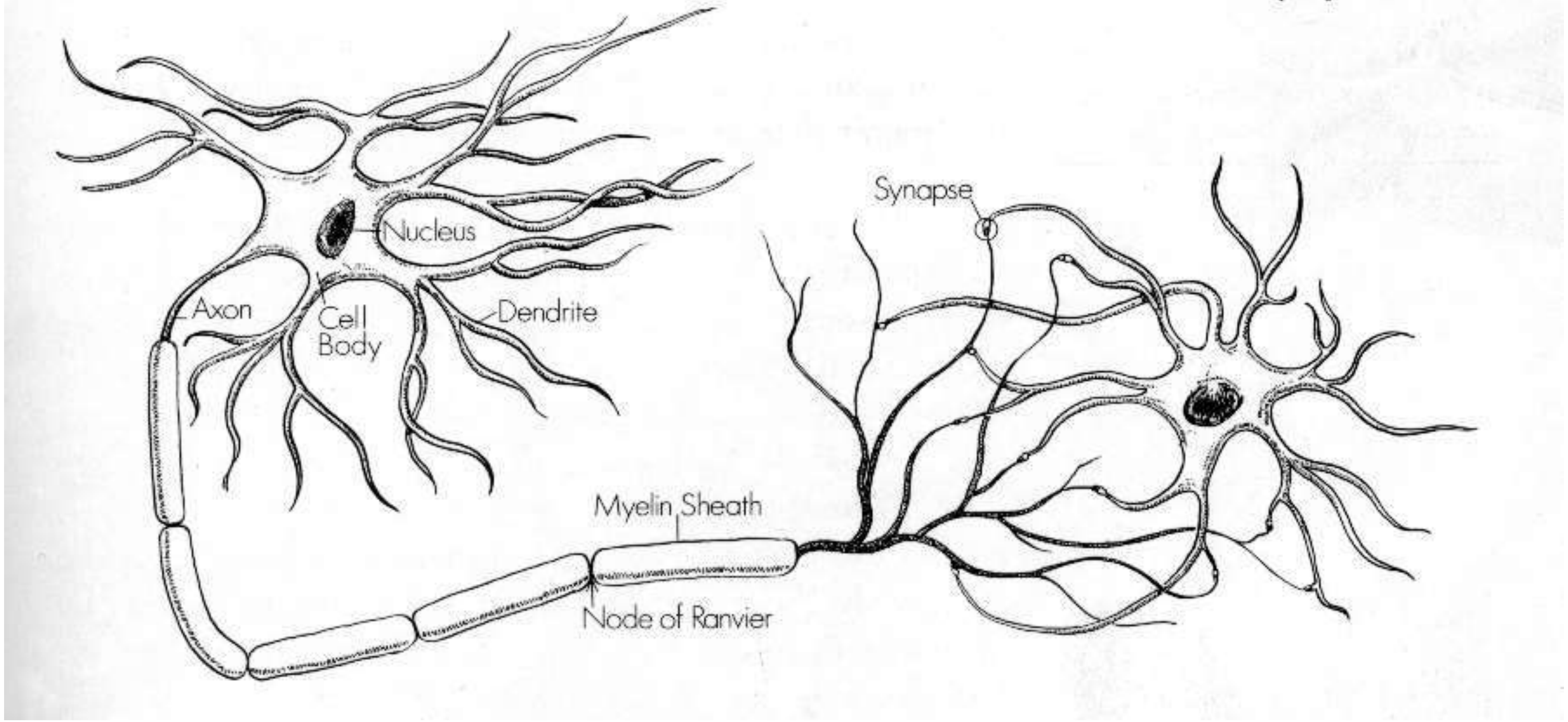
- Re-emergence:
  1985: Multi-layer nets that use back-propagation
  1986: PDP Research Group - multi-disciplined approach

Biological Neurons

- A neuron consists of two main parts:
  - Axon
    - one per neuron
    - excites up to 10^4 other neurons
    - all or nothing output signal
  - Dendrites
    - 1 to 10^4 per neuron



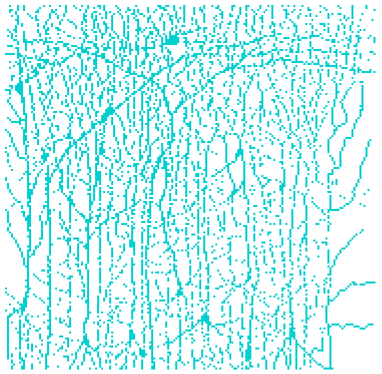
Two Neurons



Two neurons in synaptic contact.

Properties of the Brain

- Architectual
  - **80,000 neurons per square mm**
  - **10^{11} neurons - 10^{15} connections**
  - **Most axons extend less than 1 mm (local connections)**
- Operational
  - **Highly complex, nonlinear, parallel computer**
  - **Operates at millisecond speeds**


A typical network of neurons

Interconnectedness

- Each neuron may have over a thousand synapses
- Some cells in cerebral cortex may have 200,000 connections
- Total number of connections in the brain “network” is astronomical—greater than the number of particles in known universe

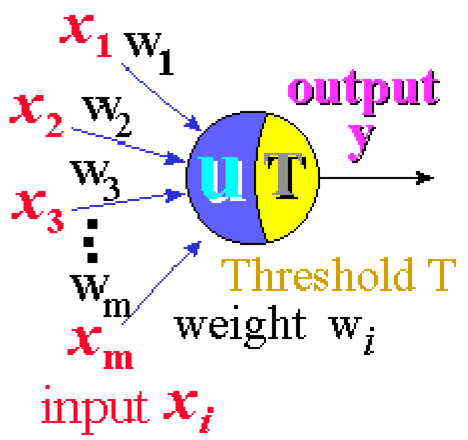
Artificial Neuron

- An artificial neuron is designed to mimic the first-order characteristics of a biological neuron

A set of inputs, x_i

Each input is multiplied by the corresponding weight, w_i

All of the weighted inputs are summed to determine the activation level



An activation function is applied to determine the neuron output

Mathematical Structure

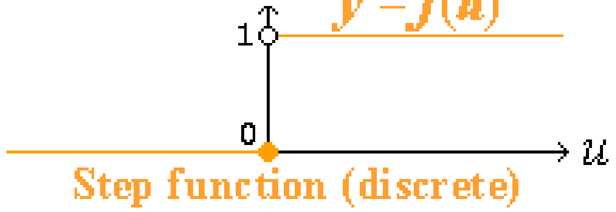
- A neuron is a simple sum and compare device

$$\mathbf{u = x_1w_1 + x_2w_2 + \ldots x_nw_n - T}$$

threshold

$$\mathbf{y = f(u)}$$

One possible function for y is a step function:

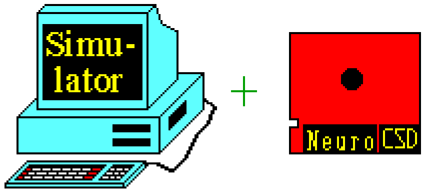


If the weighted sum of the inputs is greater than the threshold, then the neuron "fires"

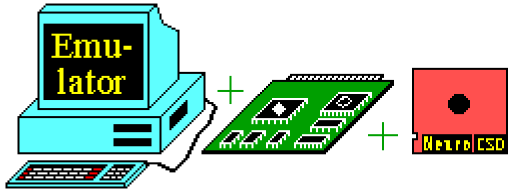
Applications of Neural Nets

Applications 1

- There are a variety of applications (both research and commercial) of neural nets



Simulations
special software that implements NN on standard computers



Hardware emulation
special hardware coprocessors

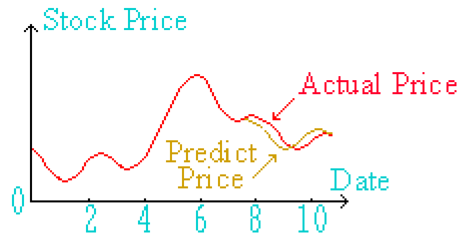
Applications 2

- Diagnosis
  - Closest to pure concept learning and classification
  - Some ANNs can be post-processed to produce probabilistic diagnoses
- Prediction and Monitoring
  - *aka* prognosis (sometimes forecasting)
  - Predict a continuation of (typically numerical) data
- Decision Support Systems
  - *aka* recommender systems
  - Provide assistance to human “subject matter” experts in making decisions
    - Design (manufacturing, engineering)
    - Therapy (medicine)
    - Crisis management (medical, economic, military, computer security)
- Control Automation
  - Mobile robots
  - Autonomic sensors and actuators
- Many, Many More (ANNs for Automated Reasoning, etc.)

Applications 3

- Other applications include:

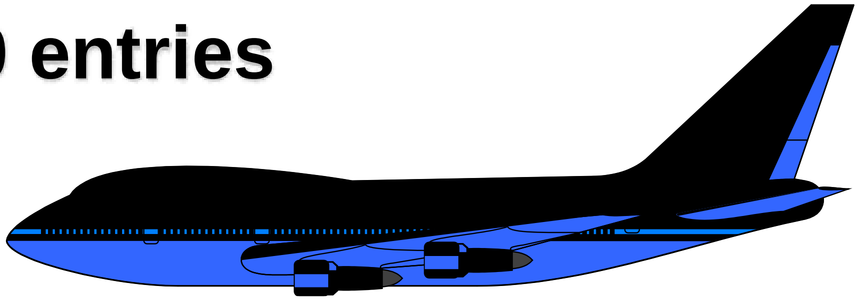
Stock Price Prediction:
Data for past stock prices are learned by neural networks, and their trends for one or a few days ahead are predicted.



Signature Check:
For checking a signature of a person, first we train the neural networks to recognize that signature by learning several writing features such as strength of the pen, direction, and any special point of that sample. After learning, the neural network can check whether a signature belongs to the same person or not.

Applications 4

- The *Boeing Airplane Company* uses an ART-1 neural network system (NIRS) for the identification and retrieval of 2-D and 3-D representations of engineering designs.
- Avoids redesign of existing parts and tools
- Production solid model data base > 55,000 entries
- 2-D data base > 95,000 entries



Applications 5

- Credit Card Fraud Detection
  - **The probability of fraud is calculated with a neural network with each card transaction. When the probability of fraud reaches a critical threshold, the case is sent to one of the retailer's fraud analysts for action.**

Review – Advantage of the Brain

Inherent Advantages of the Brain:

"distributed processing and representation"

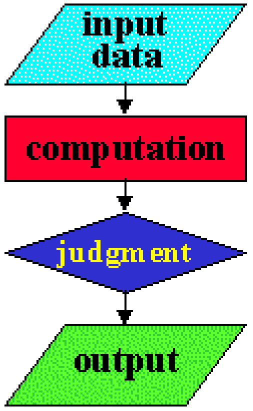
- Parallel processing speeds
- Fault tolerance
- Graceful degradation
- Ability to generalize

Features of Neural Nets

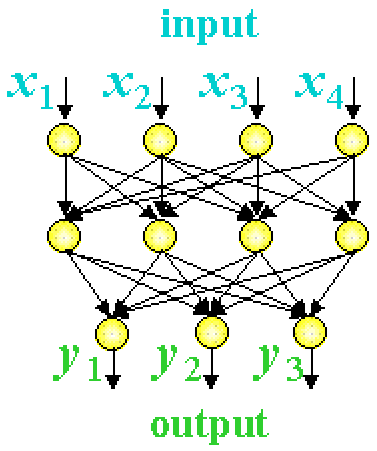
Neural Net Characteristics 1

- Neural nets have many unique features:

Massive Parallel Processing:



Sequential Information Processing (Computer)



Parallel Information Processing (Neural Network)

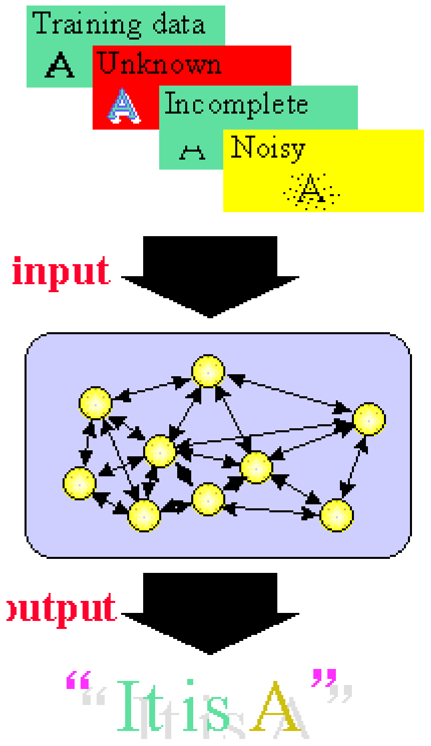
Neural Nets update in parallel

Computers calculate in sequence

Neural Net Characteristics 2

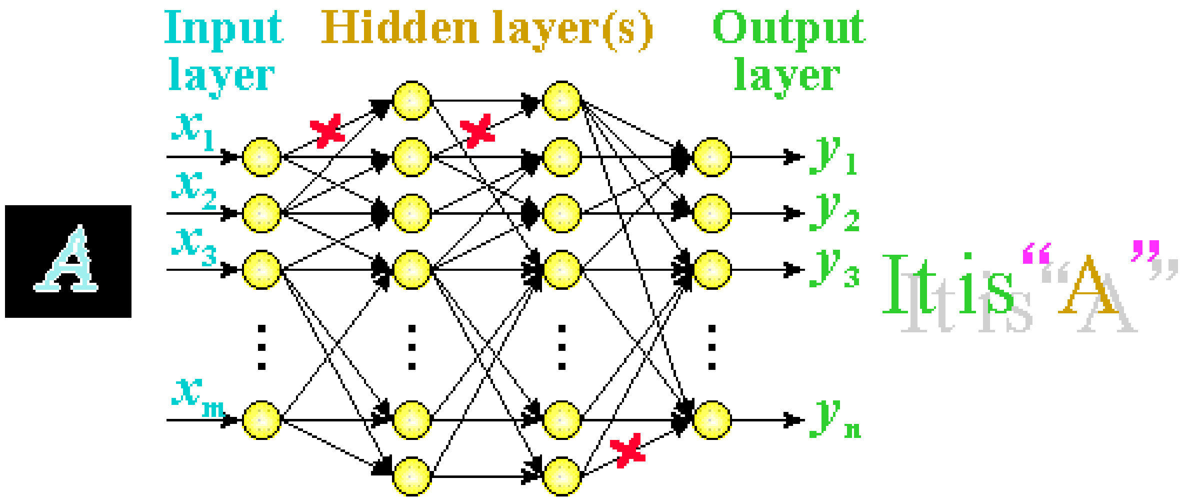
- Neural nets can learn:

Once a neural net is trained, it can handle unknown, incomplete or noisy data



Neural Net Characteristics 3

- Unlike computer memory, information is distributed over the entire network, so a failure in one part of the network does not prevent the system for producing the correct output.



Learning in Neural Nets

- One of the powerful features of neural networks is learning.
- Learning in neural networks is carried out by adjusting the connection weights among neurons.
- It is similar to a biological nervous system in which learning is carried out by changing synapses connection strengths, among cells.
- Learning may be classified into 2 categories:
  (1) Supervised Learning
  (2) Unsupervised Learning

Introduction to Learning by Induction
Learning by Example

- Learning by induction is a general learning strategy where a concept is learned by drawing inductive inferences from a set of facts

- AI systems that learn by example can be viewed as searching a concept space by means of a decision tree

- The best known approach to constructing a decision tree is called ID3

Iterative Dichotomizer 3 - developed by J. Ross Quinlan in 1975

ID3 Process

- ID3 begins with a set of examples (known facts)
- It ends with a set of rules in the form of a decision tree which may then be applied to unknown cases
- It works by recursively splitting the example set into smaller sets in the most efficient manner possible.

Review – Learning by Example

- Learning by induction is a general learning strategy where a concept is learned by drawing inductive inferences from a set of facts
- AI systems that learn by example can be viewed as searching a concept space by means of a decision tree
- The best known approach to constructing a decision tree is called ID3

Review – Example of Learning

- User is observed to file email messages as follows:-

Message	Domain	Size	Type	Rank
No.1	Internal	Small	Personal	Important
No.2	Internal	Medium	Mailing List	Normal
No.3	Internal	Small	Personal	Important
No.4	World	Small	Personal	Important
No.5	World	Large	Mailing List	Normal
No.6	IE	Small	Mailing List	Normal
No.7	IE	Small	Personal	Important
No.8	World	Large	Mailing List	Normal
No.9	IE	Large	Personal	Normal
No.10	IE	Medium	Personal	Normal

Learn users behaviour from examples in order to automate this process.

Background

- We are interested in datasets consisting of a number of *records*
- Each record has:
  - A set of *attribute-value pairs*
    E.g. “Color: green, Temperature: 120^{o}C”
  - A *classification*
    E.g. “Risk: Low”
- We assume that the classification is *dependent* on the attributes
  - We are assuming that the dataset “makes sense”

Inductive Learning

- Interesting problem: Can we find the mapping between the attributes and the classifications only using the dataset?

- This is the purpose of an *inductive learning algorithm*

- In practice, we can only find an *approximation* to the true mapping
  - Only a hypothesis, but still useful:
    - Classification of new records (prediction)
    - Identifying patterns in the dataset (data mining)

Classification Problem

- We model each record as a pair (*x*, *y*), where *x* is the set of attributes and *y* is the classification
- We are trying to find a function *f* which models the data:
  - We want to find *f* such that *f*(*x*) = *y* for as many records in the dataset as possible
- To make things easier, we will use a fixed set of attributes and discrete classes

Choosing a Classifier

- Some issue to consider:
  - Construction time
  - Runtime performance
  - Transparency
  - Updateability
  - Scalability

- All these rest on how *f* is represented by the model

Common Classifiers

- Analytical (mathematical) models
- Nearest-neighbor classifiers
- Neural networks
- Genetic programming
- Genetic algorithms (classifier systems)
- Decision trees

Decision Trees

- What is a Decision Tree?
  - it takes as input the description of a situation as a set of attributes (features) and outputs a yes/no decision (so it represents a Boolean function)
  - each leaf is labeled "positive” or "negative", each node is labeled with an attribute (or feature), and each edge is labeled with a value for the feature of its parent node

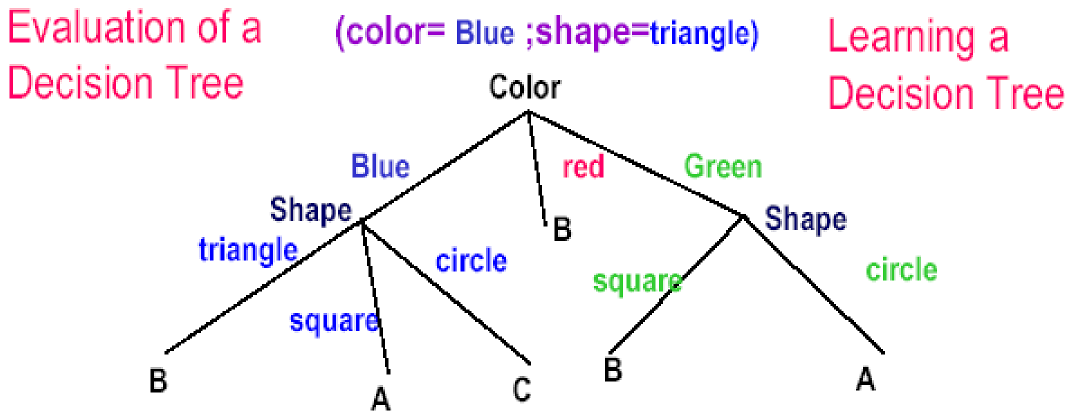
- Attribute-value language for examples
  - in many inductive tasks, especially learning decision trees, we need a representation language for examples
    - each example is a finite feature vector
    - a concept is a decision tree where nodes are features

History

- Independently developed in the 60’s and 70’s by researchers in ...
  - Statistics: L. Breiman & J. Friedman - CART (Classification and Regression Trees)
  - Pattern Recognition: Uof Michigan - AID, G.V. Kass - CHAID   (Chi-squared Automated Interaction Detection)
  - AI and Info. Theory: R. Quinlan - ID3, C4.5 (Iterative Dichotomizer)

Decision Tree Representation

- Decision Trees are classifiers for instances represented as features vectors. (color= ;shape= ;label=)
  - Nodes are tests for feature values;
  - There is one branch for each value of the feature
  - Leaves specify the categories (labels)
  - Can categorize instances into multiple disjoint categories



Example

- Problem: How do you decide to leave a restaurant?

- Variables:

Alternative	Whether there is a suitable alternative restaurant nearby.
Bar	Is there a comfortable bar area?
Fri/Sat	True on Friday or Saturday nights.
Hungary	How hungry is the subject?
Patrons	How many people in the restaurant?
Price	Price range.
Raining	Is it raining outside?
Reservation	Does the subject have a reservation?
Type	Type of Restaurant.
Stay?	Stay or Go

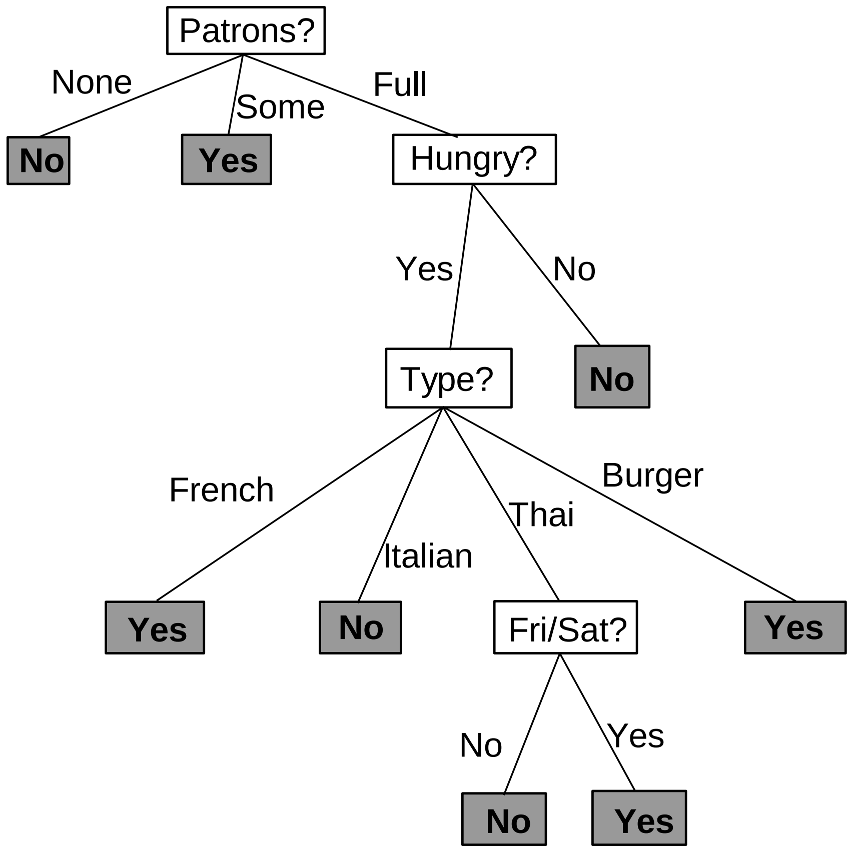
Method

- Gather test data using the variables and observe the decision.

Case	Alt.	Bar	Fri.	Hun	Pat	Price	Rain	Res	Type	Est.	Stay?
X1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
X2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
X3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
X4	Yes	No	Yes	Yes	Full	$	No	No	Thai	Oct-30	Yes
X5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
X6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
X7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
X8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
X9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
X10	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	Oct-30	No
X11	No	No	No	No	None	$	No	No	Thai	0-10	No
X12	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Result

- **Very good Decision Tree**
  - Classifies all examples correctly
  - Very few nodes



How?

Given: Set of examples with Pos. & Neg. classes

Problem: Generate a Decision Tree model to classify a separate (validation) set of examples with minimal error

Approach: *Occam's Razor* - produce the simplest model that is consistent with the training examples -> narrow, short tree. Every traverse should be as short as possible

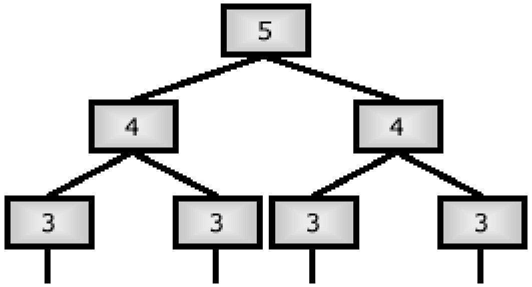
Formally: Finding the absolute simplest tree is intractable, but we can at least try our best

Exhaustive Search

- We could try to build every possible tree and the select the best one

- How many trees are there for a specific problem?
  - Given a dataset with 5 different binary attributes
  - How many trees are there in this case?

Answer



- The resulting tree must contain at least 1 node, and can be at most 5 levels deep
  - We can choose any of the 5 attributes at the root node
  - In the next level, we have 4 attributes to chose from, for each branch
  - This gives 5 x (4 x 4) x (3 x 3 x 3 x 3) x … x 1
  - This is 5 x 42 x 34 x 28 x 1 = 1658880 (for a tree of depth 5)
  - The total is 1659471 for all possible trees (depths 5,4,3,2&1)

Producing the Best DT

Heuristic (strategy) 1: Grow the tree from the top down. Place the *most important variable* test at the root of each successive subtree

The *most important variable:*

- the variable (predictor) that gains the most ground in classifying the set of training examples
- the variable that has the most significant relationship to the response variable
- to which the response is most dependent or least independent

ID3 Process

- ID3 begins with a set of examples (known facts)
- It ends with a set of rules in the form of a decision tree which may then be applied to unknown cases
- It works by recursively splitting the example set into smaller sets in the most efficient manner possible.

Algorithm

- Given a set of training examples, E, which fall into one of two classes (+ or -)

If all the examples of E are negative then create a negative node and stop

If all the examples of E are positive then create a positive node and stop

Otherwise use a criterion to select an attribute A for a new decision node

Use A to partition E into subsets and apply the algorithm to each subset

ID3 uses entropy as the criterion for selecting the next attribute

Aside: Entropy

- Entropy is the basis of Information Theory
- Entropy is a measure of randomness, hence the smaller the entropy the greater the information content
- Given a message with n symbols in which the *i*th symbol has probability p_i of appearing, the entropy of the message is:

$$\boxed{H = -\sum_{i=1}^{n} (p_i \log_2 p_i)}$$

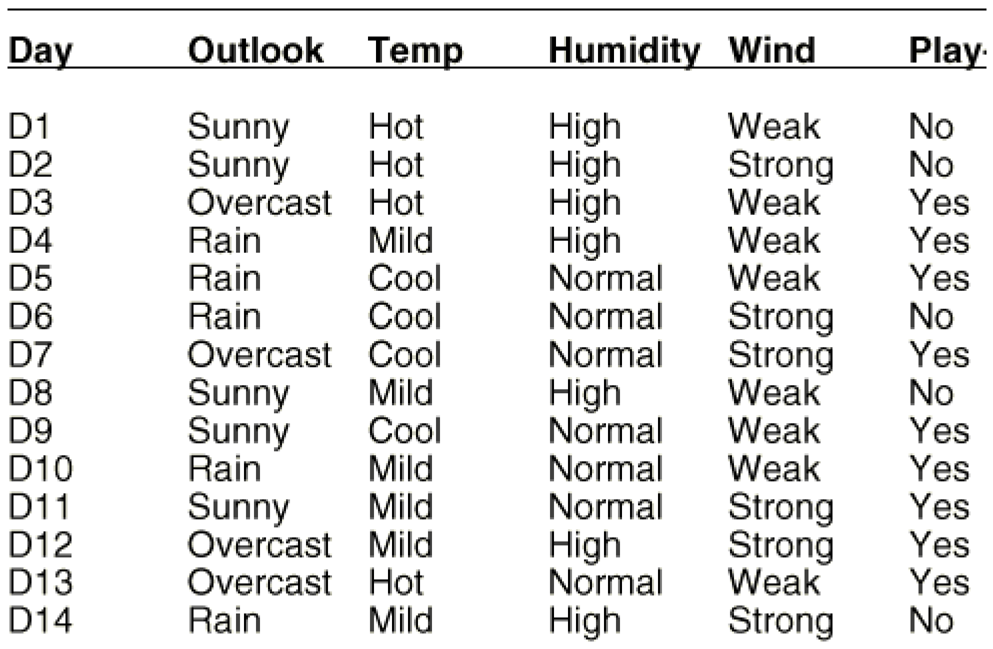
Entropy in ID3

- For a selected attribute value $A = v_i$, determine the number of examples with v_i that are negative (N_n) and the number of examples with v_i that are positive (N_p).

$$\boxed{H = -\sum_{i=1}^{n}\left[N_p \log_2\left(\frac{N_p}{N_p + N_n}\right) + N_n \log_2\left(\frac{N_n}{N_p + N_n}\right)\right]}$$

Example

- Learn when it is best to play tennis
- Observe the weather for 14 days and note when tennis was played:



Day	Outlook	Temp	Humidity	Wind	Play
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

So, what is the general rule?

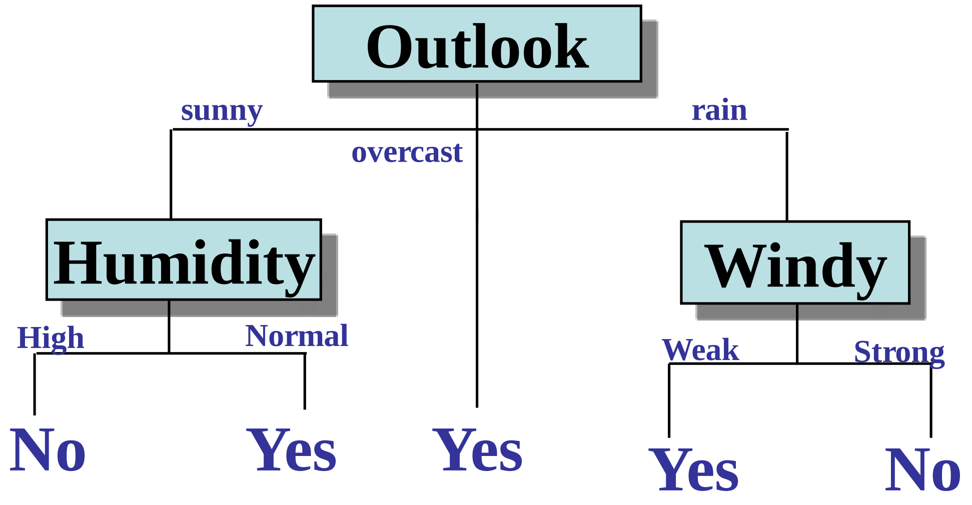
Constructing the Decision Tree

- Because the examples are neither all positive or all negative, the best discriminator must be selected to become the root node (the one with the smallest entropy)

Attribute	Values	N_n	N_p	Entropy
Outlook	Sunny	3	2	9.7096
	Overcast	0	4	
	Rain	2	3	
Temp	Hot	2	2	12.7549
	Mild	2	4	
	Cool	1	3	
Humidity	High	4	3	11.0383
	Normal	1	6	
Windy	Weak	2	6	12.4902
	Strong	3	3	

Constructing the Next Level

- Now look for the minimum entropy among the three outlook cases:



Finished - what happened to Temp?

For the Sunny cases only:

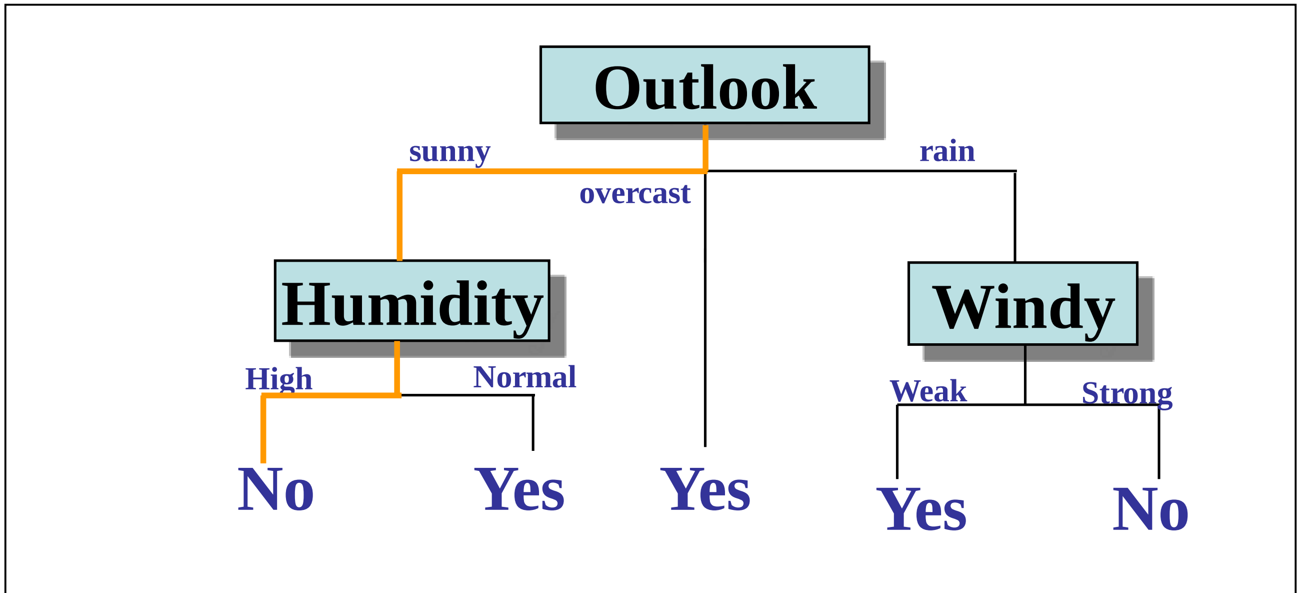
Temp	Hot	2	0	
	Mild	1	1	2.0
	Cool	0	1	
Humidity	High	3	0	0
	Low	0	2	
Windy	Weak	2	1	4.75
	Strong	1	1	

For the Rain cases only:

Temp	Hot	0	0	
	Mild	1	2	4.75
	Cool	1	1	
Humidity	High	1	1	4.75
	Low	1	2	
Windy	Weak	0	3	0
	Strong	2	0	

Rules

- The rules for playing tennis are given by the decision tree:



If the outlook is sunny and the humidity is high
Then do not play tennis

Exercise and Demonstration

- C:\Users\Dell\Deci_Tree.ipynb
- C:\Users\Dell\ ID3.ipynb
- C:\Users\Dell\ ID3_Diabetics.ipynb

Decision Trees

- a decision tree consists of
  - **Nodes:**
    - test for the value of a certain attribute
  - **Edges:**
    - correspond to the outcome of a test
    - connect to the next node or leaf
  - **Leaves:**
    - terminal nodes that predict the outcome

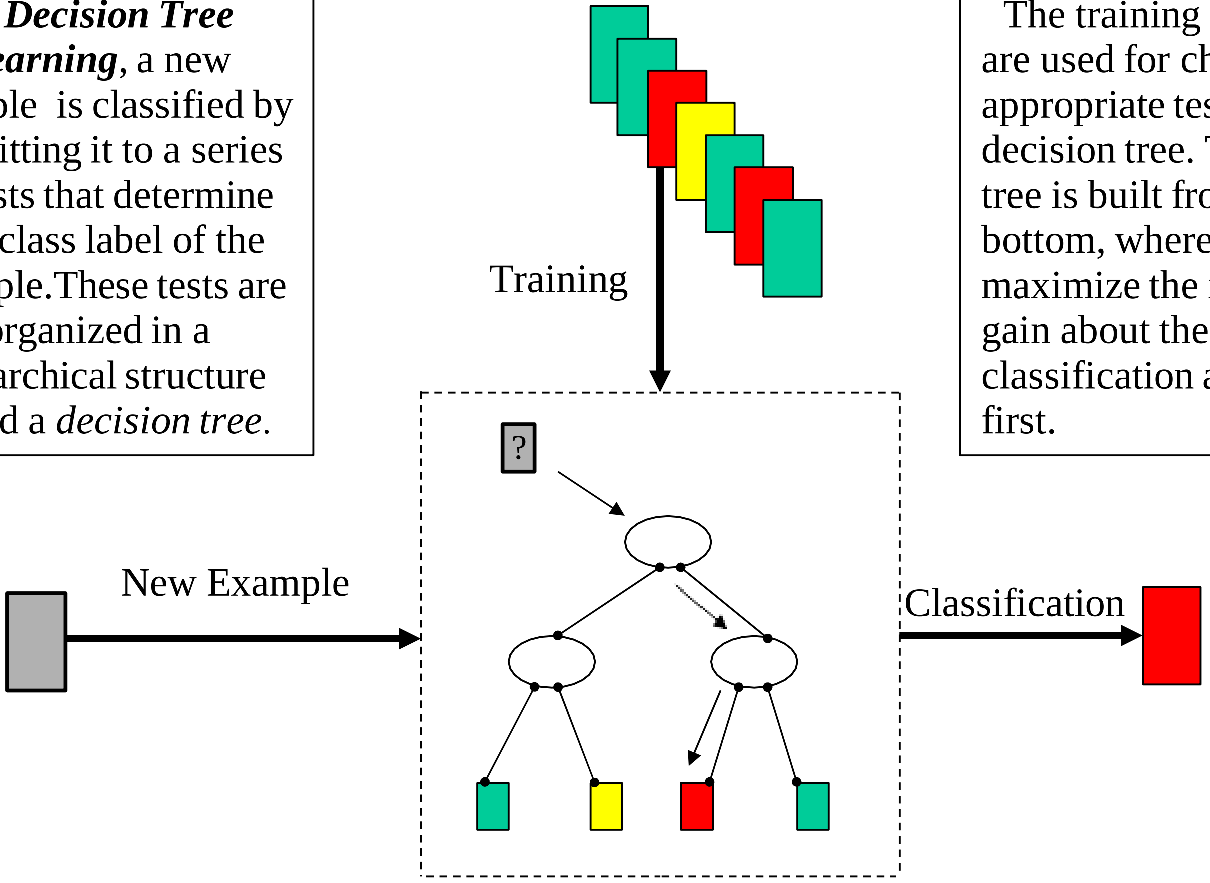
to classifiy an example:

1. start at the root
2. perform the test
3. follow the edge corresponding to outcome
4. goto 2. unless leaf
5. predict that outcome associated with the leaf

SOLL IHR NEUES AUTO SEINEN PREIS WERT SEIN?
NEIN
JA
DM EURO
Genau das Wichtige.

Decision Tree Learning

In ***Decision Tree Learning***, a new example is classified by submitting it to a series of tests that determine the class label of the example.These tests are organized in a hierarchical structure called a *decision tree*.



The training examples are used for choosing appropriate tests in the decision tree. Typically, a tree is built from top to bottom, where tests that maximize the information gain about the classification are selected first.

LARGE EXAMPLE

SHOULD WE PLAY TENNIS?

- **Example:** *PlayTennis*
  - Four attributes used for classification:
    - *Outlook* = {Sunny,Overcast,Rain}
    - *Temperature* = {Hot, Mild, Cool}
    - *Humidity* = {High, Normal}
    - *Wind* = {Weak, Strong}
  - One predicted (target) attribute (binary)
    - *PlayTennis* = {Yes,No}
  - Given 14 Training examples
    - 9 positive
    - 5 negative

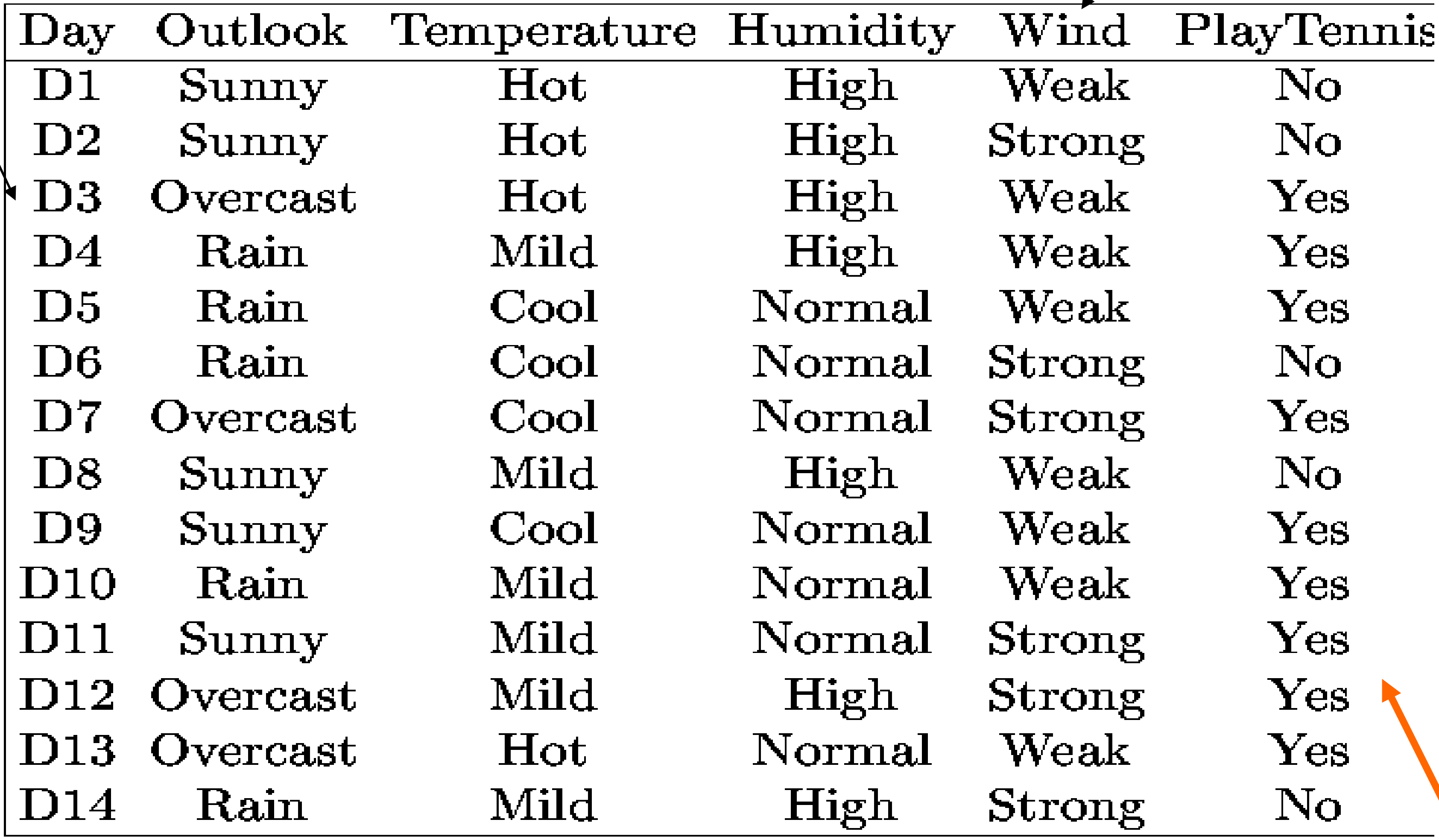
TENNIS

Training Examples

Training Examples

Object, sample, example

Attribute, variable, property

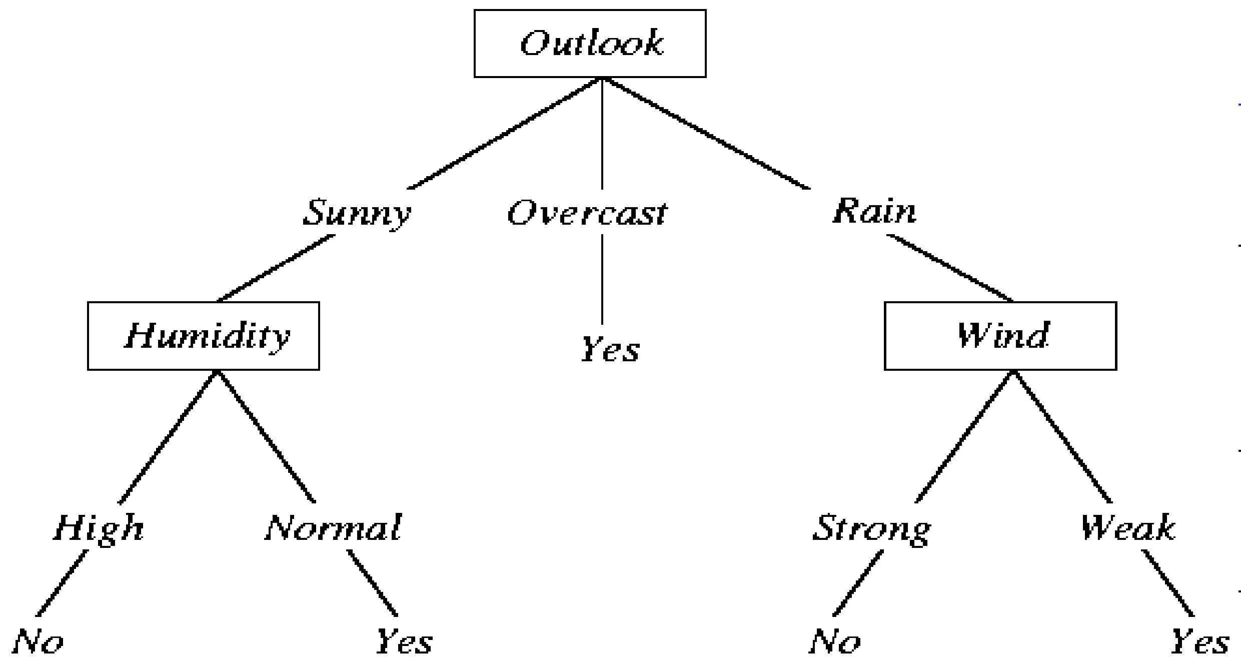


Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal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

Shall we play tennis today? (Tennis 1)

decision

Decision Tree for *PlayTennis*



- **Decision trees do classification**
  - Classifies instances into one of a discrete set of possible categories
  - **Learned function** represented by tree
  - Each ***node in tree*** is test on some attribute of an instance
  - Branches represent *values of attributes*
  - Follow the tree from root to leaves to find the output value.

Shall we play tennis today?

- The tree itself forms hypothesis
  - Disjunction (OR's) of conjunctions (AND's)
  - Each path from root to leaf forms conjunction of constraints on attributes
  - Separate branches are disjunctions
- Example from *PlayTennis* decision tree:

$$(\text{Outlook=Sunny} \wedge \text{Humidity=Normal})$$

$$\vee$$

$$(\text{Outlook=Overcast})$$

$$\vee$$

$$(\text{Outlook=Rain} \wedge \text{Wind=Weak})$$

- **Types of problems decision tree learning is good for:**
  - Instances represented by attribute-value pairs
    - For algorithm in book, attributes take on a small number of discrete values
    - Can be extended to real-valued attributes
      - (numerical data)
    - Target function has **discrete output values**
    - Algorithm in book assumes Boolean functions
    - Can be extended to multiple output values

- Hypothesis space can include disjunctive expressions.
  - In fact, hypothesis space is complete space of finite discrete-valued functions
- Robust to imperfect training data
  - classification errors
  - errors in attribute values
  - missing attribute values

- Examples:
  - Equipment diagnosis
  - Medical diagnosis
  - Credit card risk analysis
  - Robot movement
  - Pattern Recognition
    - face recognition
    - hexapod walking gates

- What is a **greedy search**?
  - At each step, make decision which makes greatest improvement in whatever you are trying optimize.
  - Do not backtrack (unless you hit a dead end)
  - This type of search is likely not to be a globally optimum solution, but generally works well.
- What are we really doing here?
  - At each node of tree, make decision on which **attribute** best classifies training data at that point.
  - Never backtrack (in ID3)
  - Do this for each branch of tree.
  - End result will be tree structure representing a ***hypothesis which works best for the training data***.

Information Theory Background

Information Theory Background

- If there are n equally probable possible messages, then the probability p of each is 1/n
- Information conveyed by a message is -log(p) = log(n)
- Eg, if there are 16 messages, then log(16) = 4 and we need 4 bits to identify/send each message.
- In general, if we are given a probability distribution

  P = (p1, p2, .., pn)
- the information conveyed by distribution (aka Entropy of P) is:

  I(P) = -(p1*log(p1) + p2*log(p2) + .. + pn*log(pn))

Question?

How do you determine *which attribute best classifies data*?

Answer: Entropy!

- ***Information gain:***
  - Statistical quantity measuring how well an attribute classifies the data.
    - Calculate the information gain for each attribute.
    - Choose attribute with greatest information gain.

- **But how do you measure information?**
  - **Claude Shannon in 1948 at Bell Labs established the field of information theory.**
  - **Mathematical function, *Entropy*, measures** information content **of *random process*:**
    - **Takes on largest value when events are equiprobable.**
    - **Takes on smallest value when only one event has non-zero probability.**
  - **For two states:**
    - **Positive examples and Negative examples from set S:**

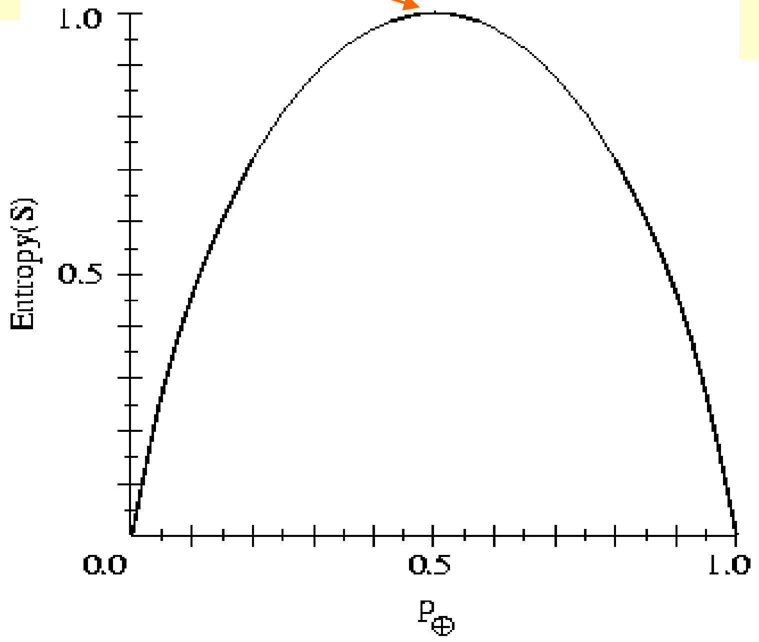
$$H(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

Entropy of set S denoted by H(S)

Entropy

Largest entropy

Entropy



- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

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

Boolean functions with the same number of ones and zeros have largest entropy

- **But how do you measure information?**
  - **Claude Shannon in 1948 at Bell Labs established the field of information theory.**
  - **Mathematical function, *Entropy*, measures** information content **of *random process*:**
    - **Takes on largest value when events are equiprobable.**
    - **Takes on smallest value when only one event has non-zero probability.**
  - **For two states:**
    - **Positive examples and Negative examples from set S:**

$$\boxed{H(S) = - p_{+}\log_{2}(p_{+}) - p_{-}\log_{2}(p_{-})}$$

Entropy

= Measure of order in set S

- In general:
  - For an ensemble of random events: $\{A_1, A_2, \ldots, A_n\}$, occurring with probabilities: $\boldsymbol{z} = \{P(A_1), P(A_2), \ldots, P(A_n)\}$

$$\boxed{H = -\sum_{i=1}^{n} P(A_i)\log_2(P(A_i))}$$

$$\boxed{\text{(Note:} \quad 1 = \sum_{i=1}^{n} P(A_i) \quad \text{and} \quad 0 \le P(A_i) \le 1 \quad)}$$

If you consider the self-information of event, *i*, to be: $-\log_2(P(A_i))$
Entropy is weighted average of information carried by each event.

Does this make sense?

–Does this make sense?

– If an event conveys information, that means it's a surprise.

– If an event always occurs, $P(A_i)=1$, then it carries no information. $-log_2(1) = 0$

– If an event rarely occurs (e.g. $P(A_i)=0.001$), it carries a lot of info. $-log_2(0.001) = 9.97$

– **The less likely the event, the more the information it carries** since, for $0 \leq P(A_i) \leq 1$, $-log_2(P(A_i))$ increases as $P(A_i)$ goes from 1 to 0.

  - (Note: ignore events with $P(A_i)=0$ since they never occur.)

- What about entropy?
  - Is it a good measure of the information carried by an ensemble of events?
  - If the events are equally probable, the entropy is maximum.

1) For N events, each occurring with probability *1/N*.

$$H = -\sum(1/N)\log_2(1/N) = -\log_2(1/N)$$

This is the maximum value.

(e.g. For N=256 (ascii characters) $-\log_2(1/256) = 8$
number of bits needed for characters.
Base 2 logs measure information in bits.)

This is a good thing since an ensemble of equally probable events is as uncertain as it gets.

(Remember, **information corresponds to surprise** - ***uncertainty***.)

– **2)** *H* is a continuous function of the probabilities.

- That is always a good thing.

– **3)** If you sub-group events into compound events, the entropy calculated for these compound groups is the same.

- That is good since the uncertainty is the same.

- ***It is a remarkable fact that the equation for entropy shown above (up to a multiplicative constant) <u>is the only function</u> which satisfies these three conditions.***

- **Choice of base 2 log corresponds to choosing units of information.(BIT's)**
- ***Another remarkable thing:***
  - ***This is the same definition of entropy used in <u>statistical mechanics</u> for the measure of disorder.***
  - ***Corresponds to macroscopic thermodynamic quantity of Second Law of Thermodynamics.***

- The concept of a quantitative measure for information content plays an important role in many areas:
- For example,
  - Data communications (channel capacity)
  - Data compression (limits on error-free encoding)
- Entropy in a message corresponds to ***minimum number of bits needed to encode that message***.
- In our case, for a set of training data, the entropy measures the number of bits needed to encode classification for an instance.
  - Use probabilities found from entire set of training data.
  - **Prob(Class=Pos) = Num. of positive cases / Total case**
  - **Prob(Class=Neg) = Num. of negative cases / Total cases**

We want to select the variables (attributes) that will lead to the smallest tree. **It means, maximizing the information gain**.

- *Information **gain*** is our metric for how well one attribute **A** ^{i} classifies the training data.
- Information gain for a particular attribute =
  Information about target function,
  given the value of that attribute.
  (conditional entropy)
- Mathematical expression for information gain:

$$\boxed{Gain(S, A_i) = \underbrace{H(S)}_{\text{entropy}} - \sum_{v \in Values(A_i)} P(A_i = v)\underbrace{H(S_v)}_{\text{Entropy for value v}}}$$

- **ID3 algorithm (for boolean-valued function)**
  - Calculate the entropy for all training examples
    - positive and negative cases
    - p_+ = #pos/Tot $\quad$ p_- = #neg/Tot
    - $H(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$
  - Determine which ***single* attribute** best classifies the training examples using information gain.
    - For each attribute find:

$$\boxed{Gain(S, A_i) = H(S) \quad - \sum_{v \in Values(A_i)} P(A_i = v) H(S_v)}$$

    - Use attribute with greatest information gain **as a root**

GOING BACK TO TENNIS EXAMPLE

Using Gain Ratios to find the best order of variables in every subtree

Using Gain Ratios to find the best order of variables in every subtree

- The notion of Gain introduced earlier favors attributes that have a large number of values.
  - If we have an attribute D that has a distinct value for each record, then Info(D,T) is 0, thus Gain(D,T) is maximal.
- To compensate for this Quinlan suggests using the following ratio instead of Gain:

  GainRatio(D,T) = Gain(D,T) / SplitInfo(D,T)

- SplitInfo(D,T) is the information due to the split of T on the basis of value of categorical attribute D.

  SplitInfo(D,T) = I(|T1|/|T|, |T2|/|T|, .., |Tm|/|T|)

where {T1, T2, .. Tm} is the partition of T induced by value of D.

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

14 cases

9 positive cases

- Step 1: *Calculate **entropy*** for all cases:

$N_{Pos} = 9$ $N_{Neg} = 5$ $N_{Tot} = 14$

entropy $\rightarrow$ $H(S) = -(9/14)*\log_2(9/14) - (5/14)*\log_2(5/14) = 0.940$

- Step 2: Loop over all attributes, calculate gain:
  - Attribute = ***Outlook***
    - Loop over values of *Outlook*

Want to select best separation of values for all selected attributes.
Approximate this by selecting an attribute with the highest information gain.

Outlook = Sunny

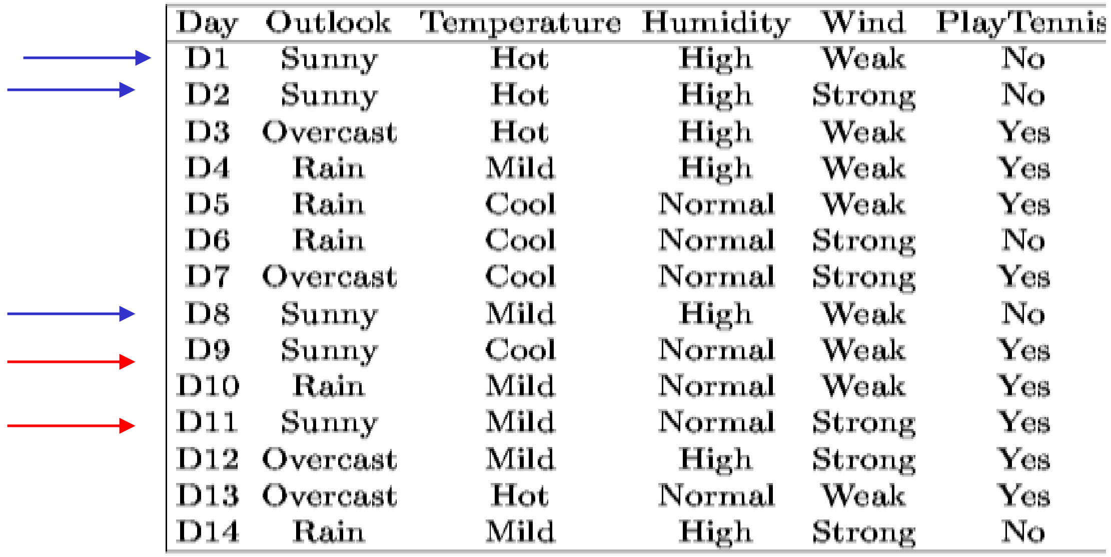
$N_{Pos} = 2$ $N_{Neg} = 3$ $N_{Tot} = 5$

$H(\text{Sunny}) = -(2/5)*\log_2(2/5) - (3/5)*\log_2(3/5) = 0.971$

Outlook = Overcast

$N_{Pos} = 4$ $N_{Neg} = 0$ $N_{Tot} = 4$

$H(\text{Sunny}) = -(4/4)*\log_2 4/4) - (0/4)*\log_2(0/4) = 0.00$



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

Outlook = Rain

$N_{Pos} = 3$ $N_{Neg} = 2$ $N_{Tot} = 5$

$H(Sunny) = -(3/5)*\log_2(3/5) - (2/5)*\log_2(2/5) = 0.971$

- Calculate **Information Gain** for attribute Outlook

$$\begin{aligned} Gain(S,Outlook) = H(S) &- N_{Sunny}/N_{Tot}*H(Sunny) \\ &- N_{Over}/N_{Tot}*H(Overcast) \\ &- N_{Rain}/N_{Tot}*H(Rainy) \end{aligned}$$

$Gain(S,Outlook) = 9.40 - (5/14)*0.971 - (4/14)*0 - (5/14)*0.971$

$Gain(S,Outlook) = 0.246$

– **Attribute = *Temperature***

- (Repeat process looping over {Hot, Mild, Cool})

$Gain(S,Temperature) = 0.029$

– Attribute = *Humidity*

- (Repeat process looping over {High, Normal})

  Gain(*S*,*Humidity*) = 0.029

– Attribute = *Wind*

- (Repeat process looping over {Weak, Strong})

  Gain(*S*,*Wind*) = 0.048

Find attribute with greatest information gain:

Gain(*S*,*Outlook*) = 0.246, Gain(*S*,*Temperature*) = 0.029

Gain(*S*,*Humidity*) = 0.029, Gain(*S*,*Wind*) = 0.048

$\therefore$ ***Outlook is root node of tree***

- **Iterate algorithm to find attributes which best classify training examples under the values of the root node**
- **Example continued**
  - **Take three subsets:**
    - ***Outlook* = Sunny (N$_{\text{Tot}}$ = 5)**
    - ***Outlook* = Overcast (N$_{\text{Tot}}$ = 4)**
    - ***Outlook* = Rainy (N$_{\text{Tot}}$ = 5)**
  - **For each subset, repeat the above calculation looping over all attributes <u>other than *Outlook*</u>**

– For example:

- *Outlook* = Sunny (N_{Pos} = 2, N_{Neg}=3, N_{Tot} = 5) H=0.971
  - *Temp* = Hot (N_{Pos} = 0, N_{Neg}=2, N_{Tot} = 2) H = 0.0
  - *Temp* = Mild (N_{Pos} = 1, N_{Neg}=1, N_{Tot} = 2) H = 1.0
  - *Temp* = Cool (N_{Pos} = 1, N_{Neg}=0, N_{Tot} = 1) H = 0.0

  Gain(S_{Sunny},*Temperature*) = 0.971 - (2/5)*0 - (2/5)*1 - (1/5)*0

  Gain(S_{Sunny},*Temperature*) = 0.571

  Similarly:

  Gain(S_{Sunny},*Humidity*) = 0.971

  Gain(S_{Sunny},*Wind*) = 0.020

  ∴ Humidity classifies *Outlook*=Sunny instances best and is placed as the node under Sunny outcome.

– Repeat this process for *Outlook* = Overcast &Rainy

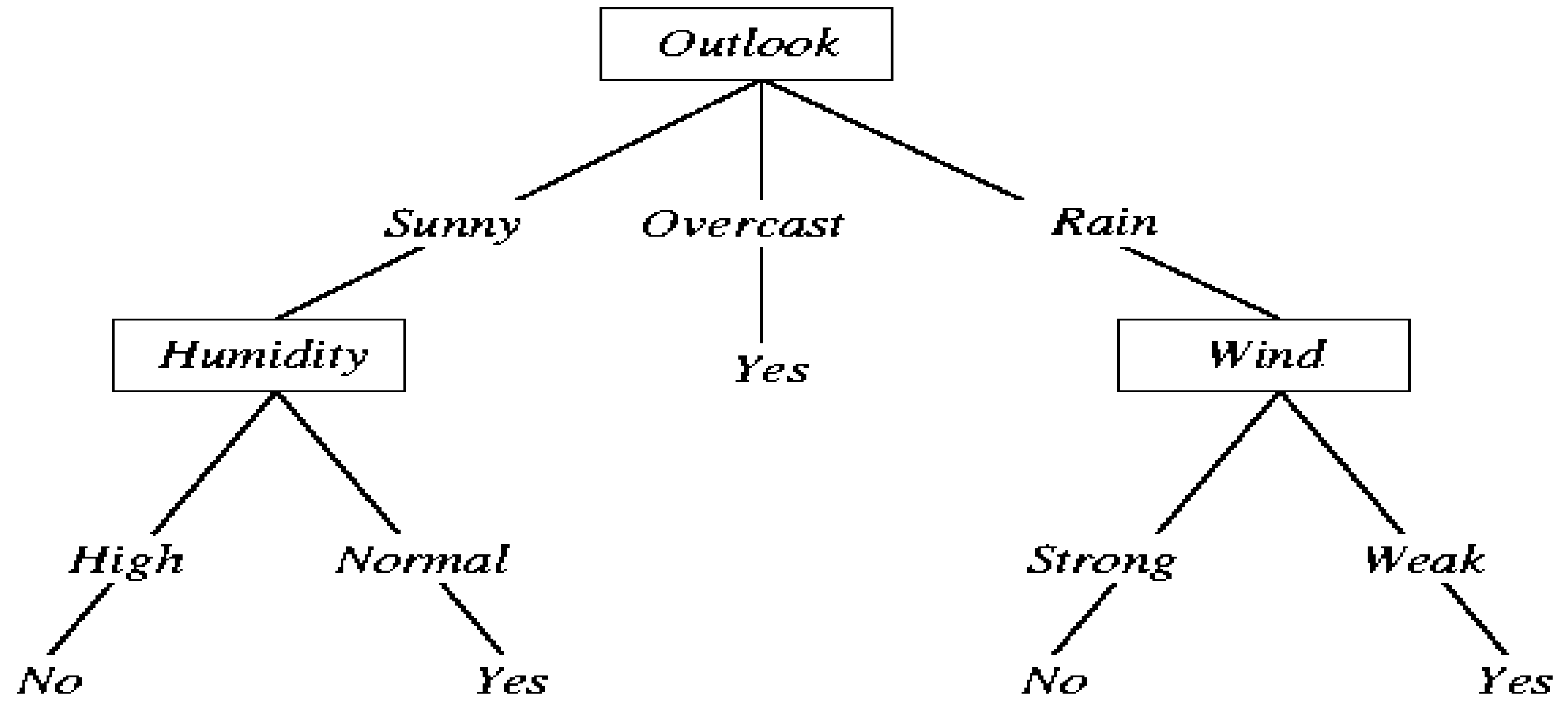
- **Important:**
  - Attributes are excluded from consideration if they appear higher in the tree
- Process continues for each new leaf node until:
  - Every attribute has already been included along path through the tree

  or

  - Training examples associated with this leaf all have same target attribute value.

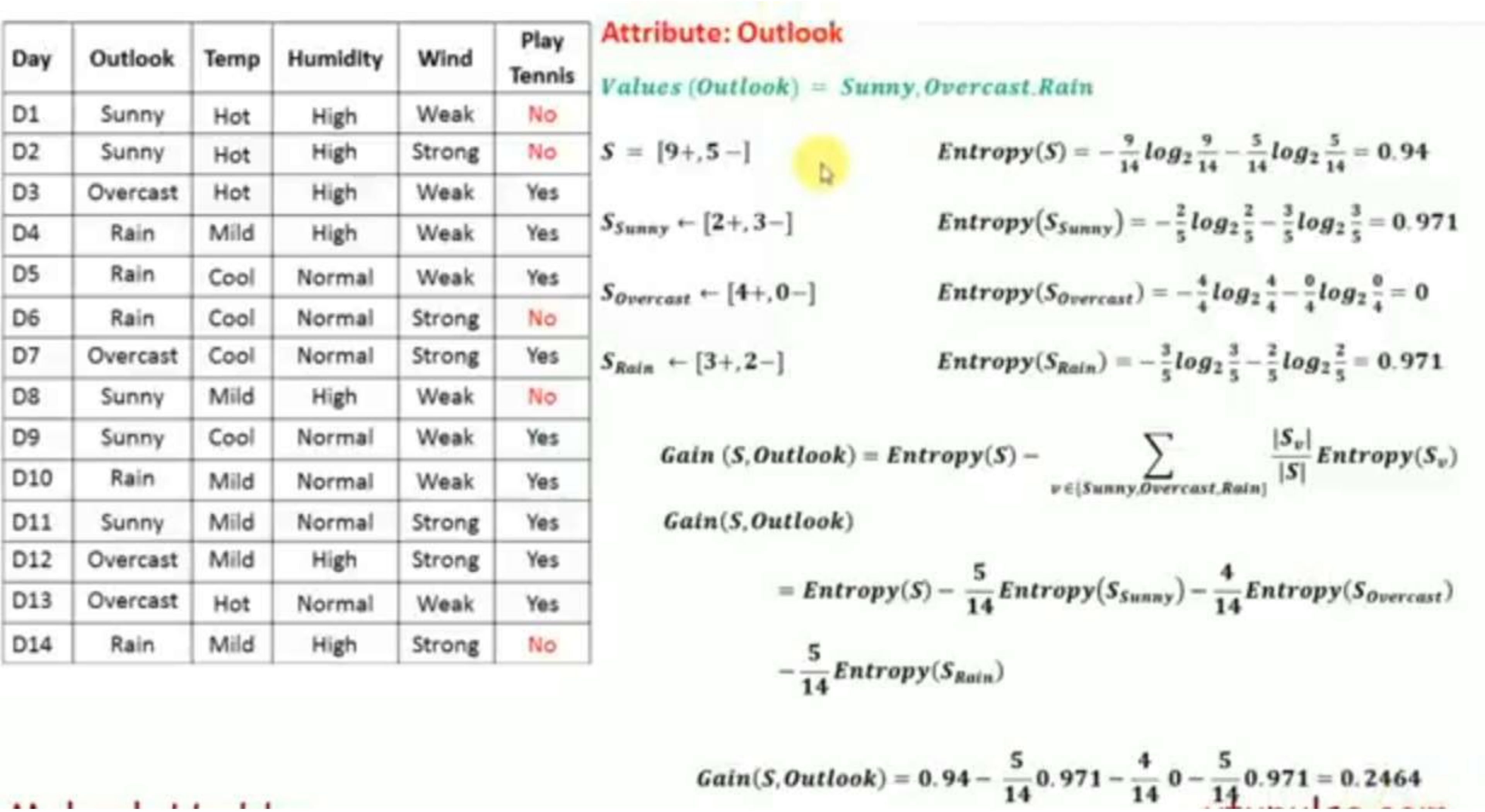
– End up with tree:

Decision Tree for *PlayTennis*



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

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

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

$S_{Hot} \leftarrow [2+, 2-]$ $\quad Entropy(S_{Hot}) = -\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4} = 1.0$

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

$S_{Cool} \leftarrow [3+, 1-]$ $\quad Entropy(S_{Cool}) = -\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8113$

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

$$Gain(S, Temp)$$

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

$$- \frac{4}{14} Entropy(S_{Cool})$$

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

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

$Values\ (Humidity) = High, Normal$

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

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

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

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

$$Gain(S, Humidity)$$

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

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

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

$Values\ (Wind) = Strong, Weak$

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

$S_{Strong} \leftarrow [3+, 3-]$ $\qquad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [6+, 2-]$ $\qquad Entropy(S_{Weak}) = -\frac{6}{8}\log_2\frac{6}{8} - \frac{2}{8}\log_2\frac{2}{8} = 0.8113$

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

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

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

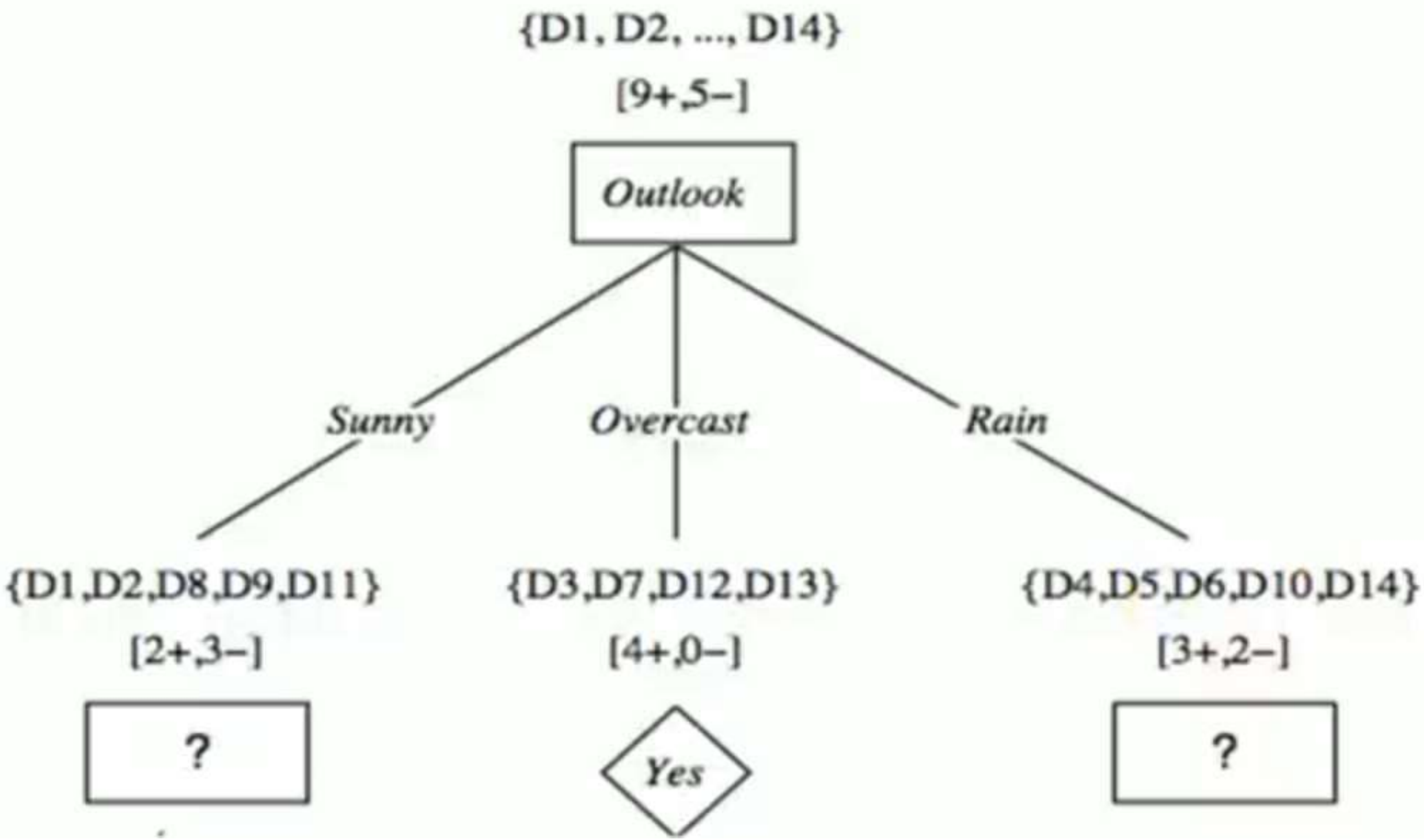
Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak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

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

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

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

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


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
{D1,D2,D8,D9,D11}
[2+,3−]
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
?
Yes
?

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

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

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

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

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

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

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

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

$$= Entropy(S) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild})$$

$$- \frac{1}{5}Entropy(S_{Cool})$$

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

Day	Temp	Humidity	Wind	Play Tennis
DI	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
DI1	Mild	Normal	Strong	Yes

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

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

$S_{high} \leftarrow [0+, 3-]$ $\quad Entropy(S_{High}) = 0.0$

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

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

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

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

Day	Temp	Humidity	Wind	Play Tennis
DI	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
DI1	Mild	Normal	Strong	Yes

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

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

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

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

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

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

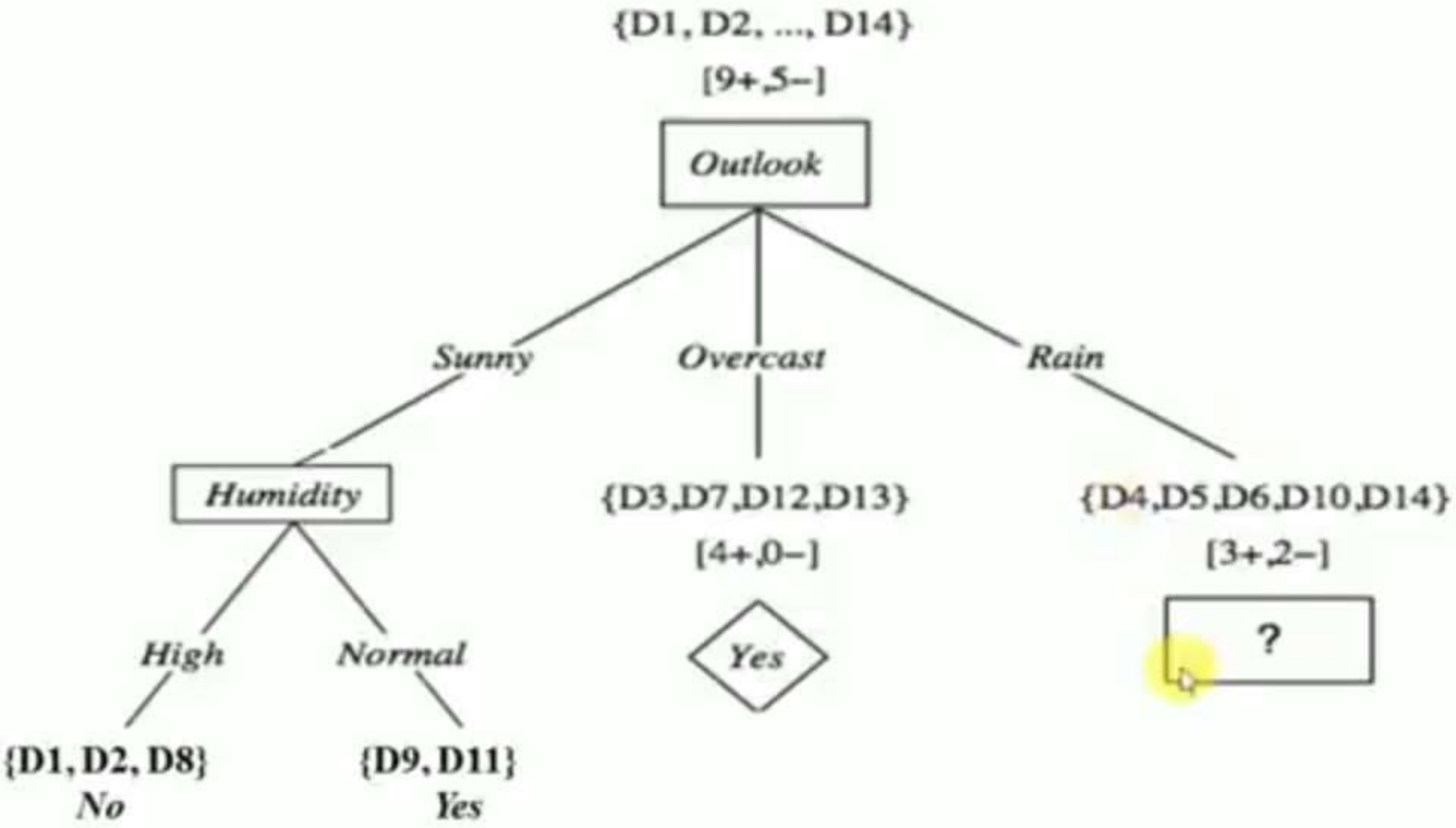
$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5} 1.0 - \frac{3}{5}\ 0.918 = 0.0192$$

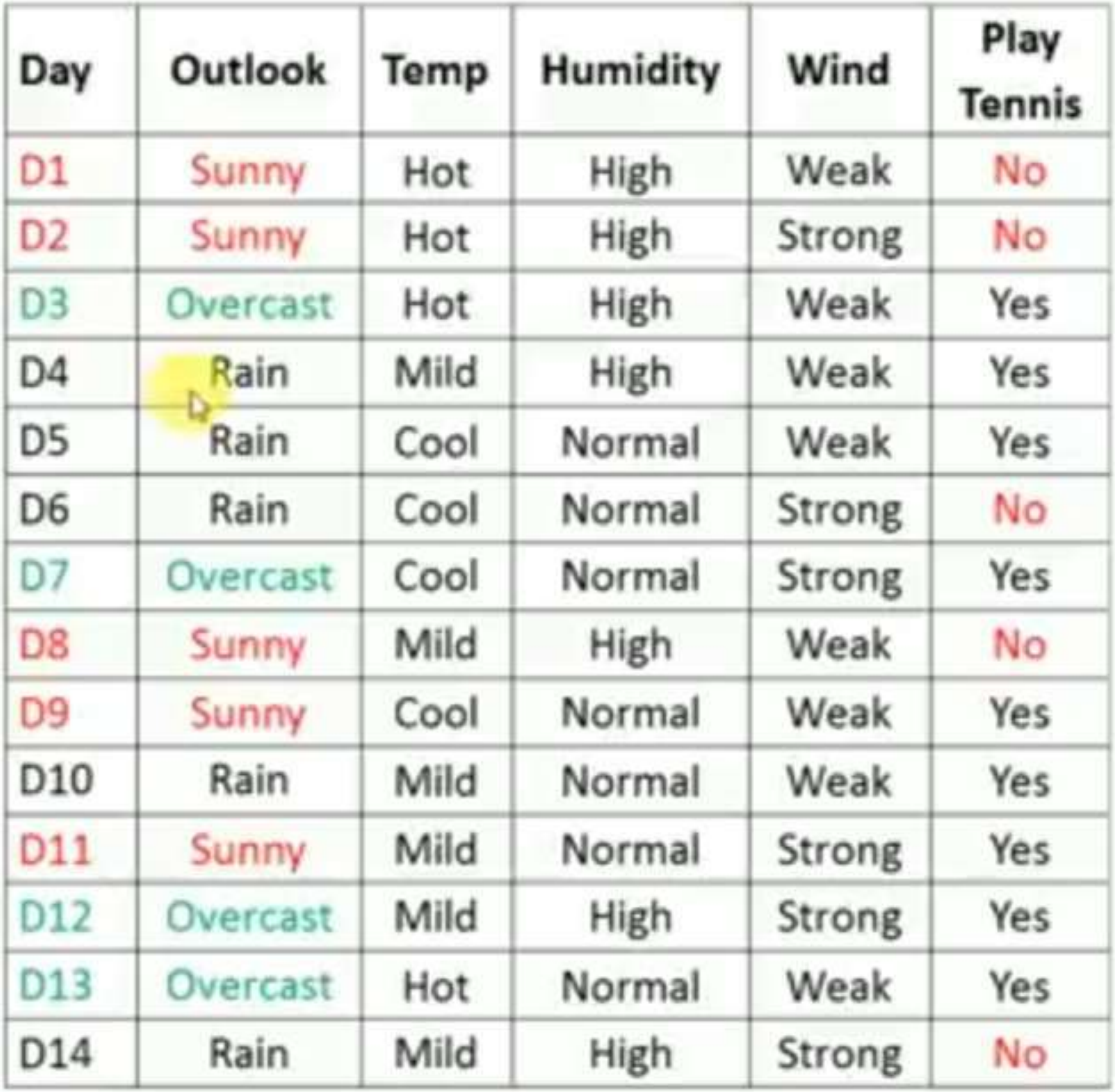
Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

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

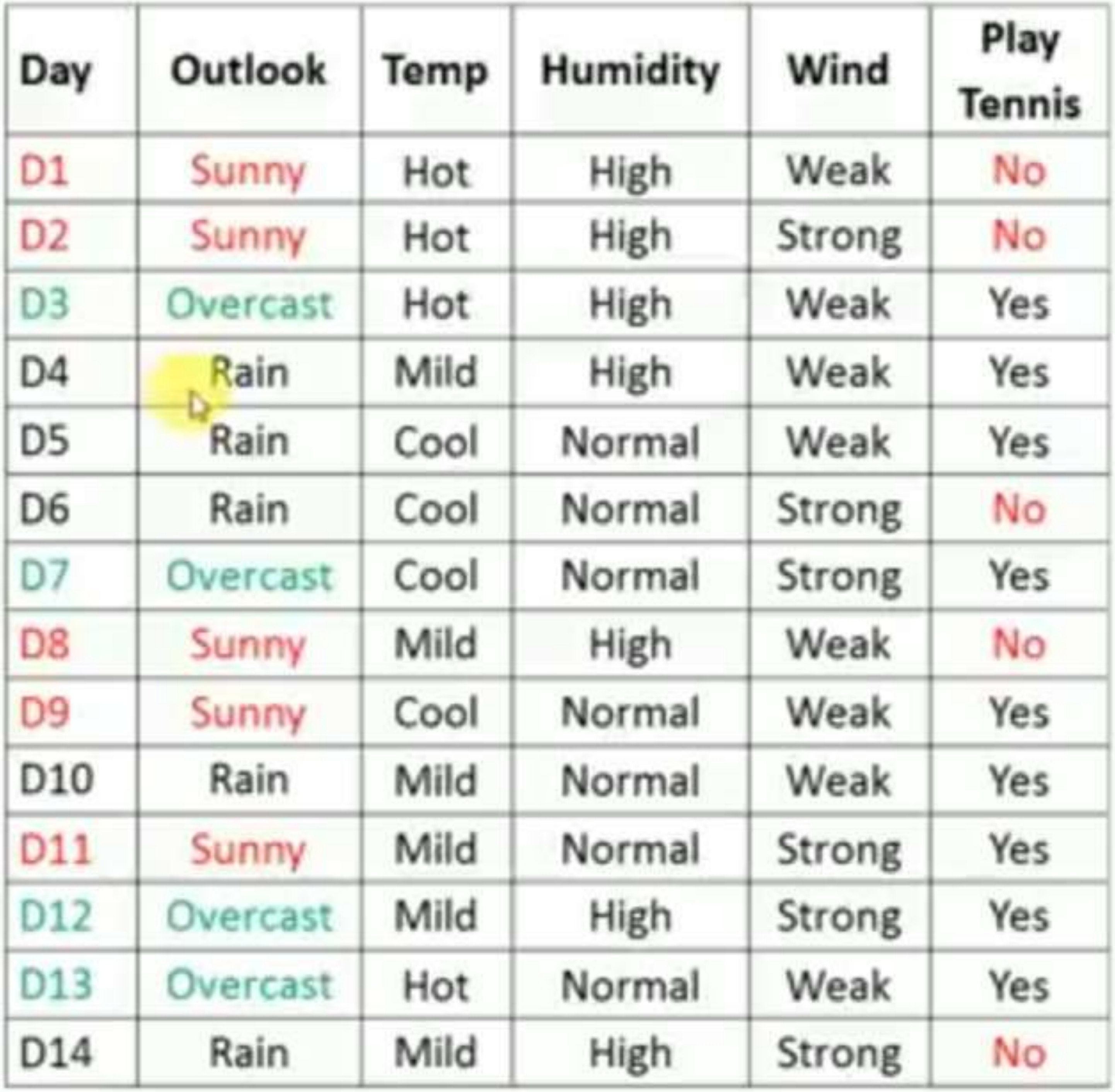
$$Gain(S_{sunny}, Humidity) = 0.97$$

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


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
High
Normal
{D1, D2, D8}
No
{D9, D11}
Yes
{D3,D7,D12,D13}
[4+,0−]
Yes
{D4,D5,D6,D10,D14}
[3+,2−]
?



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	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

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

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

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

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

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

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

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

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

$$- \frac{2}{5}Entropy(S_{Cool})$$

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

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
DI0	Mild	Normal	Weak	Yes
DI4	Mild	High	Strong	No

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

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

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

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

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

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

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

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
DI0	Mild	Normal	Weak	Yes
DI4	Mild	High	Strong	No

Attribute: Wind

Values (wind) = Strong, Weak

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

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

$S_{Weak} \leftarrow [3+, 0-]$ $Entropy(S_{weak}) = 0.0$

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

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

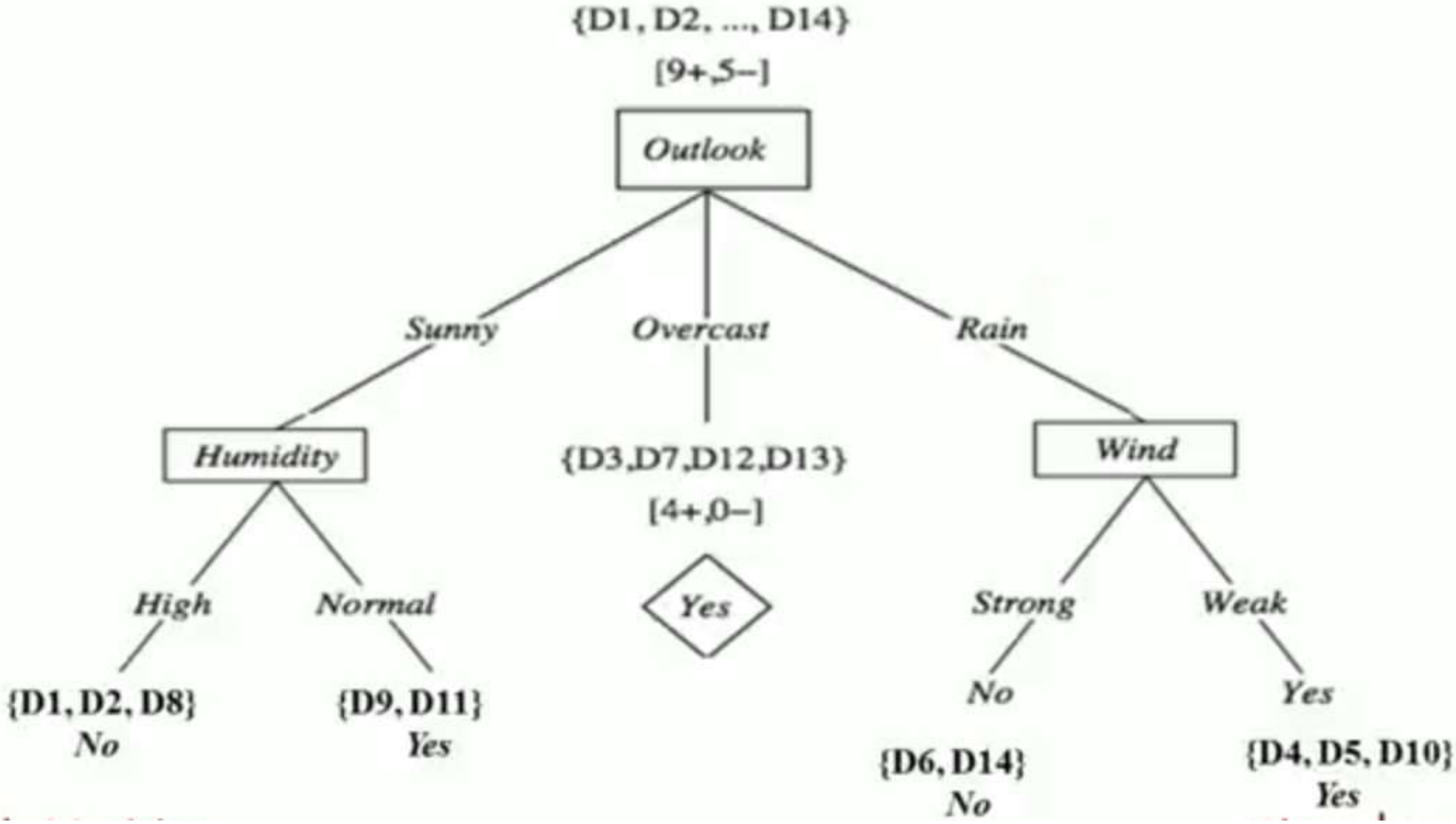
$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\ 0.0 - \frac{3}{5} 0.0 = 0.97$$

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
DI0	Mild	Normal	Weak	Yes
DI4	Mild	High	Strong	No

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

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

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



{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
{D3,D7,D12,D13}
[4+,0−]
Yes
Wind
High
Normal
Strong
Weak
{D1, D2, D8}
No
{D9, D11}
Yes
No
{D6, D14}
No
Yes
{D4, D5, D10}
Yes

- **Note:** In this example data were perfect.
  - No contradictions
  - Branches led to unambiguous *Yes*, *No* decisions
  - If there are contradictions take the majority vote
    - This handles noisy data.
- **Another note:**
  - Attributes are eliminated when they are assigned to a node and never reconsidered.
    - e.g. You would not go back and reconsider *Outlook* under *Humidity*
- **ID3 uses all of the training data at once**
  - Contrast to Candidate-Elimination
  - Can handle noisy data.

CHOOSING RESTAURANT

EXAMPLE

Another Example: Russell’s and Norvig’s Restaurant Domain

- Develop a decision tree to model the decision a patron makes when deciding whether or not to wait for a table at a restaurant.
- Two classes: wait, leave
- Ten attributes: alternative restaurant available?, bar in restaurant?, is it Friday?, are we hungry?, how full is the restaurant?, how expensive?, is it raining?,do we have a reservation?, what type of restaurant is it?, what's the purported waiting time?
- Training set of 12 examples
- ~ 7000 possible cases

A Training Set

alternative restaurant available?

bar in restaurant?,

is it Friday?

are we hungry?

how full is the restaurant?

how expensive?,

is it raining?

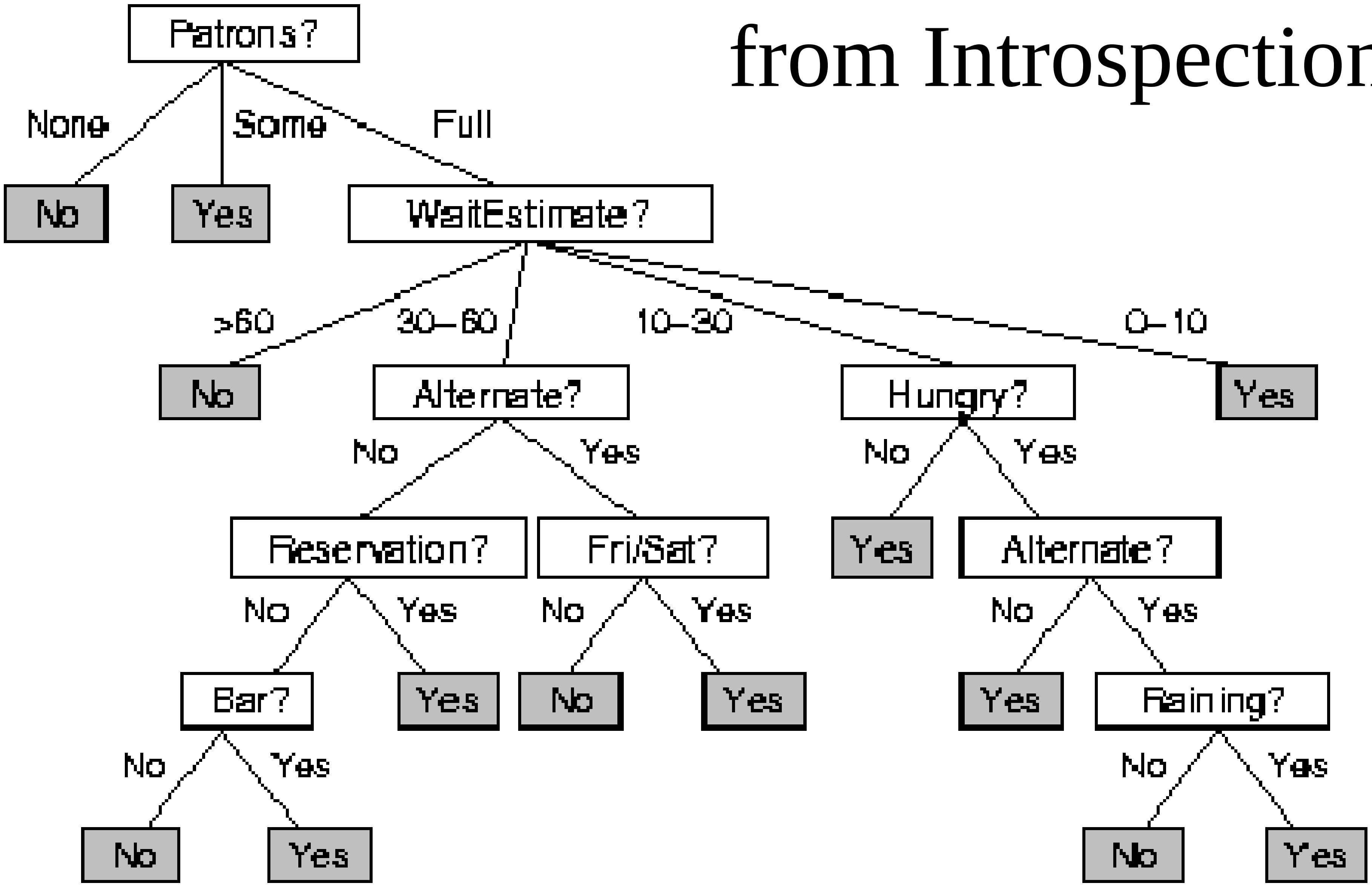
do we have a reservation?

what type of restaurant is it?

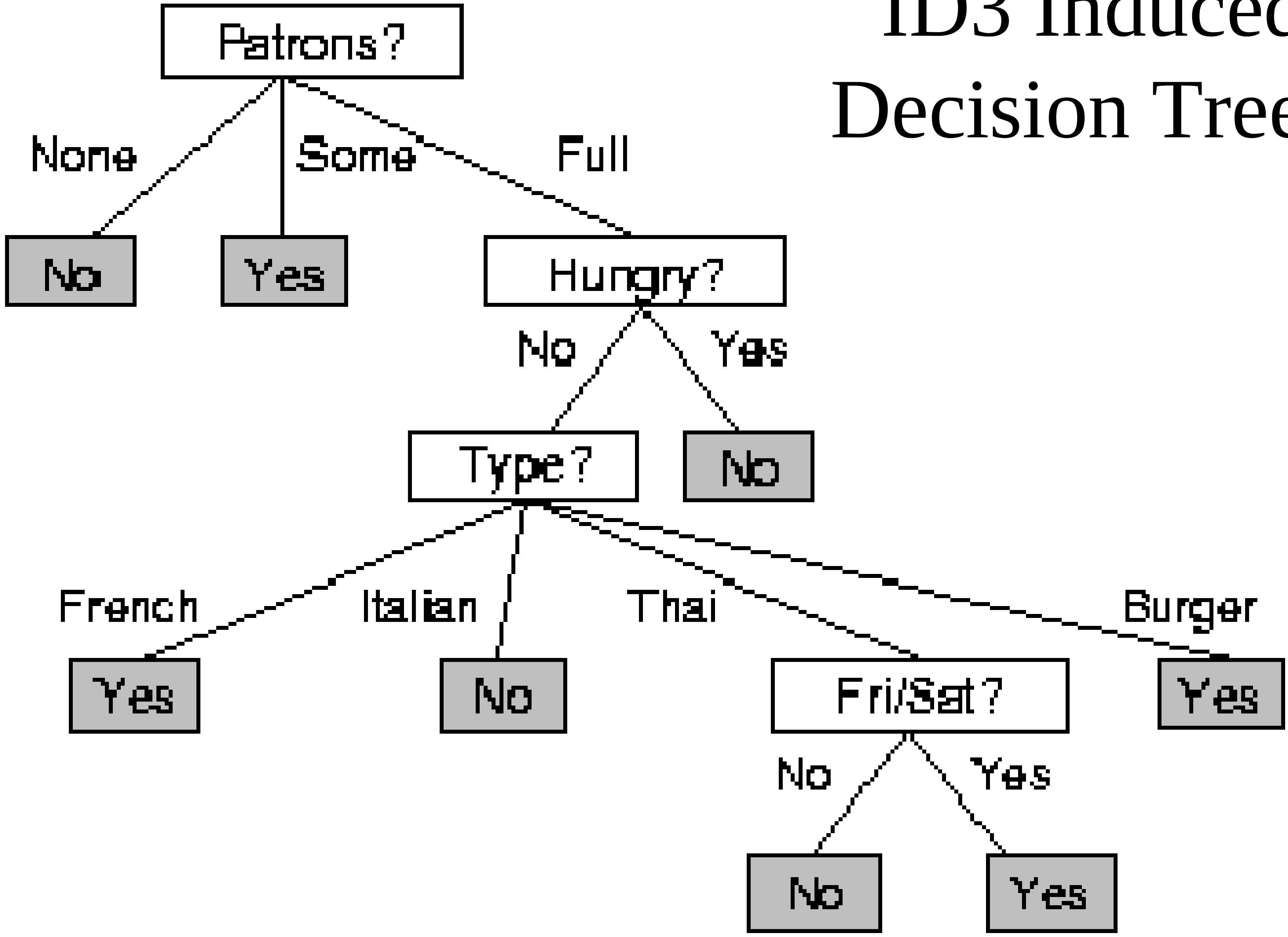
what's the purported waiting time?

Example	Attributes										Shall we wait?
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	
X_1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0–10	Yes
X_2	Yes	No	No	Yes	Full	\$	No	No	Thai	30–60	No
X_3	No	Yes	No	No	Some	\$	No	No	Burger	0–10	Yes
X_4	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10–30	Yes
X_5	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0–10	Yes
X_7	No	Yes	No	No	None	\$	Yes	No	Burger	0–10	No
X_8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0–10	Yes
X_9	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10–30	No
X_{11}	No	No	No	No	None	\$	No	No	Thai	0–10	No
X_{12}	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30–60	Yes

A decision Tree from Introspection



ID3 Induced Decision Tree



ID3

- A greedy algorithm for Decision Tree Construction developed by Ross Quinlan, 1987
- Consider a smaller tree a better tree
- **Top-down construction** of the decision tree by recursively selecting the "best attribute" to use at the current node in the tree, based on the examples belonging to this node.
  - Once the attribute is selected for the current node, generate children nodes, one for each possible value of the selected attribute.
  - Partition the examples of this node using the possible values of this attribute, and assign these subsets of the examples to the appropriate child node.
  - Repeat for each child node until all examples associated with a node are either all positive or all negative.

1. this is not exhaustive method
2. What to do if several attributes have the same information gain.
3. How do we know that we get a globally minimum solution?
4. How do we know that the minimum tree is best?

- **ID3 Algorithm**
  - Top-down, greedy search through space of possible decision trees
    - Remember, decision trees represent hypotheses, so this is a search through hypothesis space.
  - What is top-down?
    - How to start tree?
      - What attribute should represent the root?
    - As you proceed down tree, choose attribute for each successive node.
    - ***No backtracking***:
      - So, algorithm proceeds from top to bottom

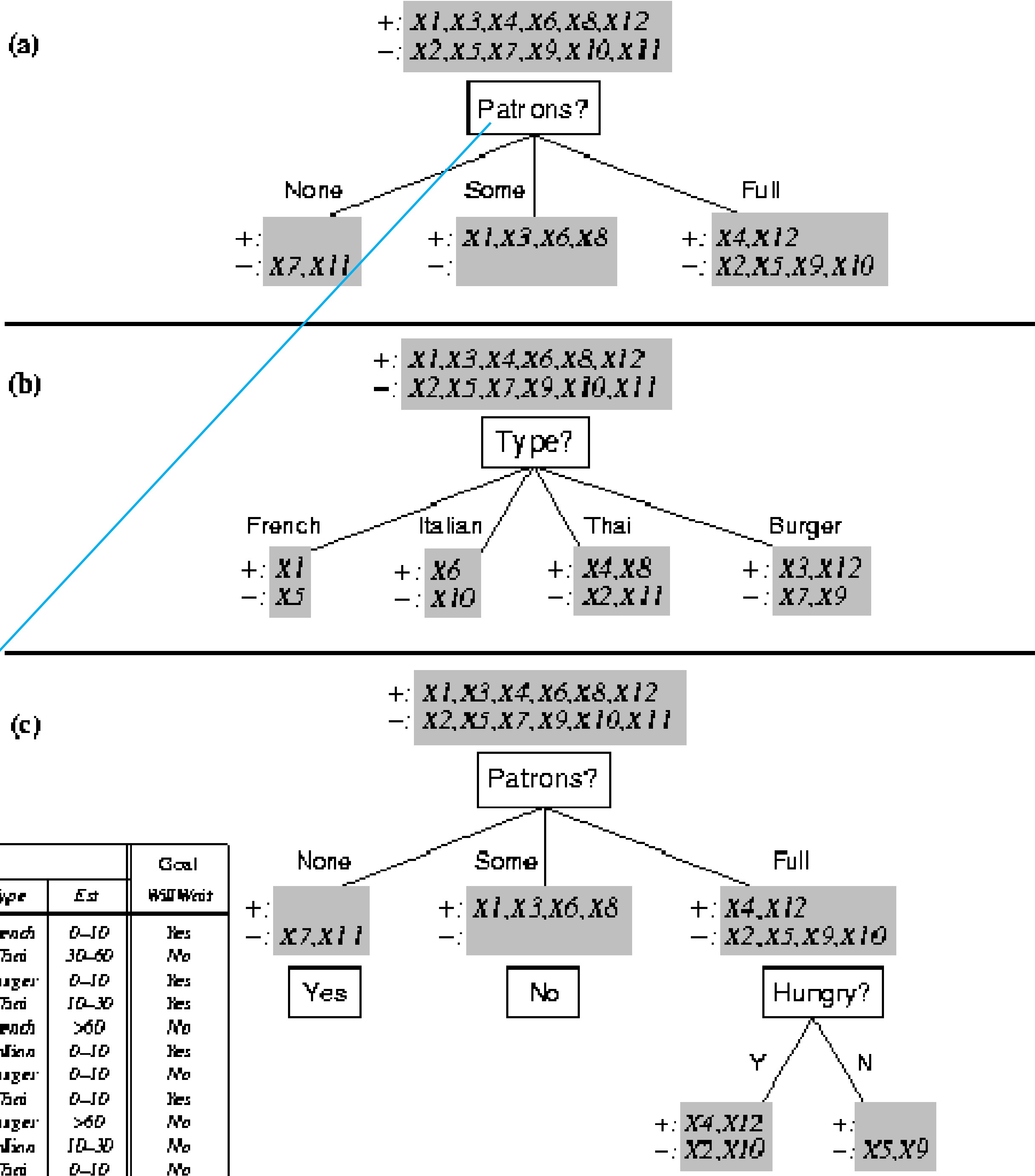
The ID3 algorithm is used to build a decision tree, given a set of non-categorical attributes C1, C2, .., Cn, the categorical attribute C, and a training set T of records.

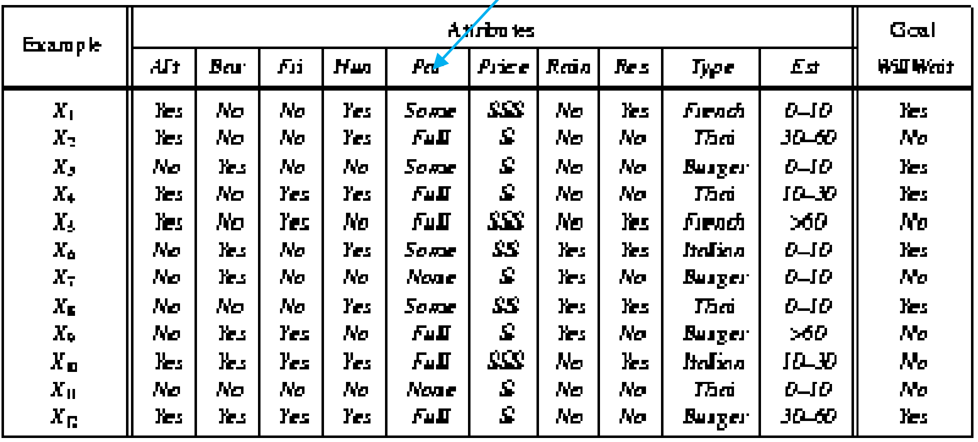
```
function ID3 (R: a set of non-categorical attributes,
             C: the categorical attribute,
             S: a training set) returns a decision tree;
   begin
     If S is empty, return a single node with value Failure;
     If every example in S has the same value for categorical
        attribute, return single node with that value;
     If R is empty, then return a single node with most
       frequent of the values of the categorical attribute found in
       examples S; [note: there will be errors, i.e., improperly
       classified records];
     Let D be attribute with largest Gain(D,S) among R's attributes;
     Let {dj| j=1,2, .., m} be the values of attribute D;
     Let {Sj| j=1,2, .., m} be the subsets of S consisting
        respectively of records with value dj for attribute D;
     Return a tree with root labeled D and arcs labeled
        d1, d2, .., dm going respectively to the trees
recursion ID3(R-{D},C,S1), ID3(R-{D},C,S2) ,.., ID3(R-{D},C,Sm);
   end ID3;
```

Choosing the Best Attribute

- The key problem is choosing which attribute to split a given set of examples.
- Some possibilities are:
  - **Random:** Select any attribute at random
  - **Least-Values:** Choose the attribute with the smallest number of possible values (fewer branches)
  - **Most-Values:** Choose the attribute with the largest number of possible values (smaller subsets)
  - **Max-Gain:** Choose the attribute that has the largest expected information gain, i.e. select attribute that will result in the smallest expected size of the subtrees rooted at its children.
- The ID3 algorithm uses the Max-Gain method of selecting the best attribute.

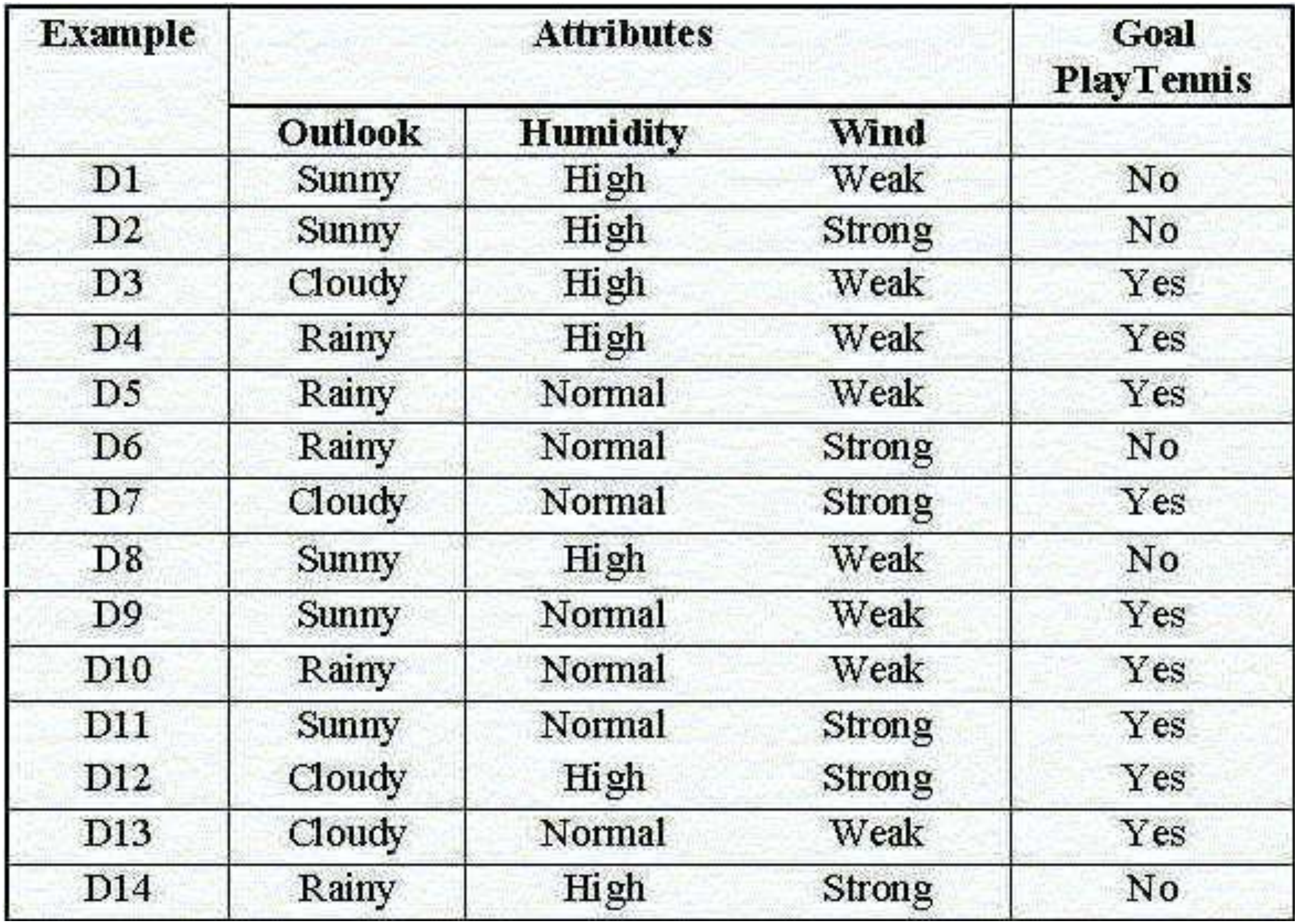
Splitting Examples by Testing Attributes





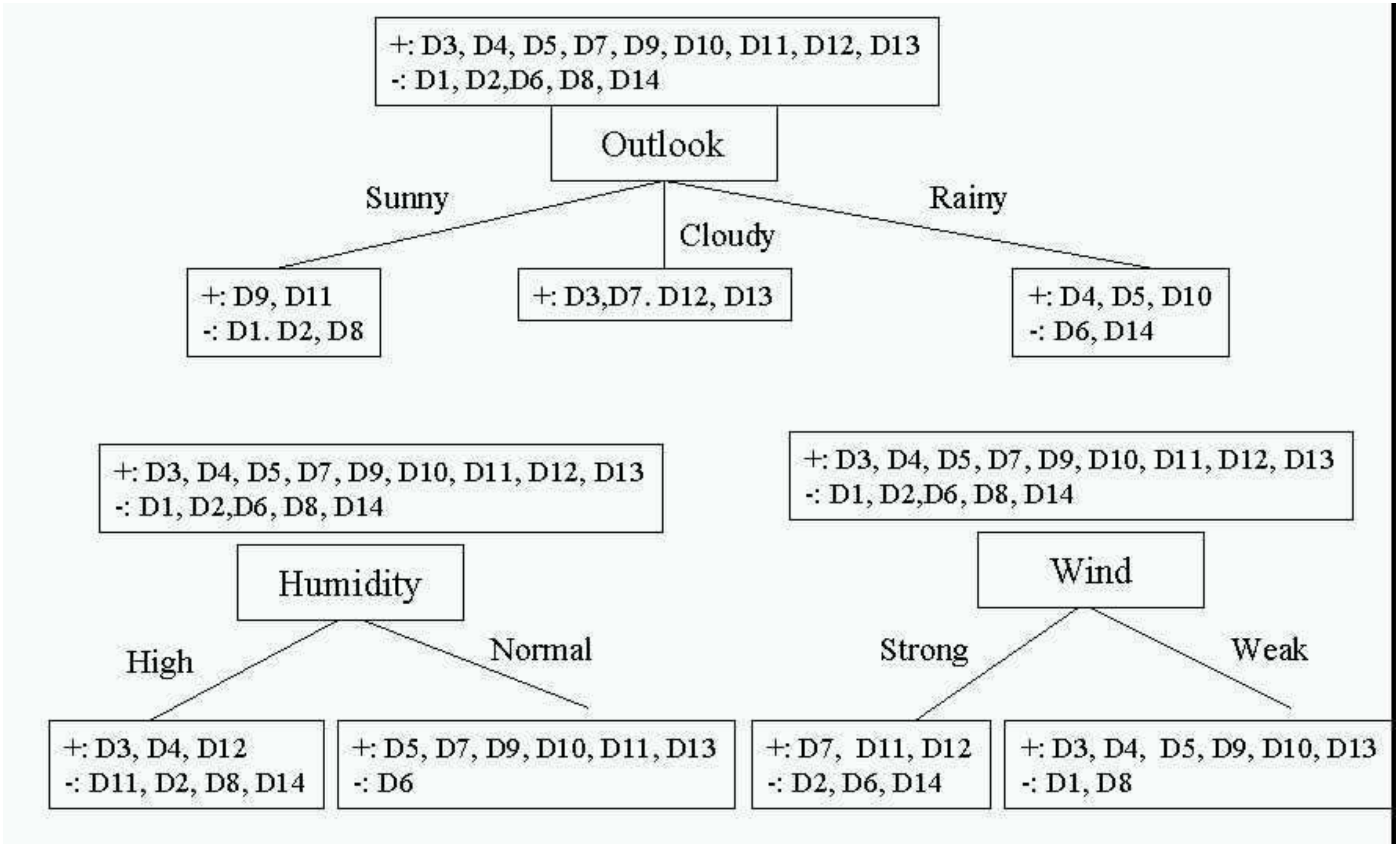
Example	Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0–10	Yes
X_2	Yes	No	No	Yes	Full	\$	No	No	Thai	30–60	No
X_3	No	Yes	No	No	Some	\$	No	No	Burger	0–10	Yes
X_4	Yes	No	Yes	Yes	Full	\$	No	No	Thai	10–30	Yes
X_5	Yes	No	Yes	No	Full	\$\$\$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	\$\$	Yes	Yes	Italian	0–10	Yes
X_7	No	Yes	No	No	None	\$	Yes	No	Burger	0–10	No
X_8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0–10	Yes
X_9	No	Yes	Yes	No	Full	\$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	\$\$\$	No	Yes	Italian	10–30	No
X_{11}	No	No	No	No	None	\$	No	No	Thai	0–10	No
X_{12}	Yes	Yes	Yes	Yes	Full	\$	No	No	Burger	30–60	Yes

Another example: Tennis 2 (simplified former example)

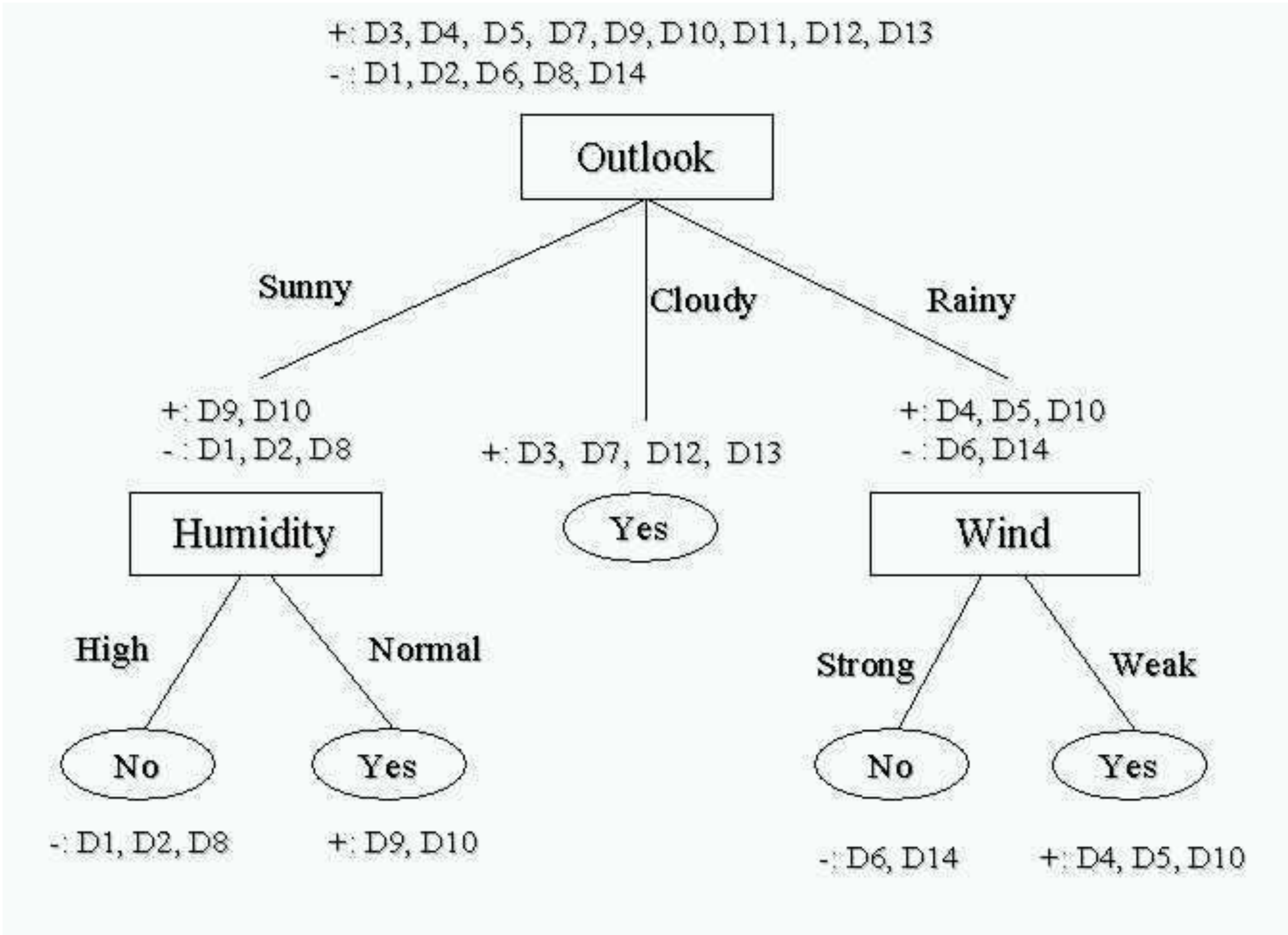


Example	Attributes			Goal PlayTennis
	Outlook	Humidity	Wind	
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Cloudy	High	Weak	Yes
D4	Rainy	High	Weak	Yes
D5	Rainy	Normal	Weak	Yes
D6	Rainy	Normal	Strong	No
D7	Cloudy	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rainy	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Cloudy	High	Strong	Yes
D13	Cloudy	Normal	Weak	Yes
D14	Rainy	High	Strong	No

Choosing the first split



Resulting Decision Tree



1. The entropy is the average number of bits/message needed to represent a stream of messages.
2. Examples:
   1. if P is (0.5, 0.5) then I(P) is 1
   2. if P is (0.67, 0.33) then I(P) is 0.92,
   3. if P is (1, 0) then I(P) is 0.
3. The more uniform is the probability distribution, the greater is its information gain/entropy.

1. Entropy assumes that function with half-ones half-zeros is the most complicated.
2. Statistically it is true.
3. It may be not true for specific functions.
4. We can recognize classes of these specific functions and improve the branching method.

- What is the **hypothesis space** for decision tree learning?
  - Search through space of all possible decision trees
    - from simple to more complex guided by a heuristic: *information gain*
  - The space searched is complete space of finite, discrete-valued functions.
    - Includes disjunctive and conjunctive expressions
- ID3 and similar methods only maintain one current hypothesis
    - In contrast to Candidate-Elimination
  - Not necessarily global optimum
    - attributes eliminated when assigned to a node
    - No backtracking
    - Different trees are possible

- **Inductive Bias:** (restriction vs. preference)
  - **ID3**
    - searches *complete hypothesis space*
    - But, *incomplete search* through this space looking for simplest tree
    - This is called a **preference** (or search) bias
  - **Candidate-Elimination**
    - Searches an *incomplete hypothesis space*
    - But, does a *complete search* finding all valid hypotheses
    - This is called a **restriction** (or language) bias
  - Typically, preference bias is better since you do not limit your search up-front by restricting hypothesis space considered.

How well does the decision tree method work?

Many case studies have shown that decision trees are at least as accurate as human experts.

- A study for diagnosing breast cancer:
  - humans correctly classifying the examples 65% of the time,
  - the decision tree classified 72% correct.
- British Petroleum designed a decision tree for gas-oil separation for offshore oil platforms/
  - It replaced an earlier rule-based expert system.
- Cessna designed an airplane flight controller using 90,000 examples and 20 attributes per example.

- **Summary of ID3 Inductive Bias**
  - **Short trees** are preferred over long trees
    - It accepts the first tree it finds
  - Information gain heuristic
    - Places high information gain attributes near root
    - Greedy search method is an approximation to finding the shortest tree
  - Why would short trees be preferred?
    - Example of **Occam's Razor:**

      Prefer simplest hypothesis consistent with the data.

      (Like Copernican vs. Ptolemic view of Earth's motion)

Review - Disadvantages

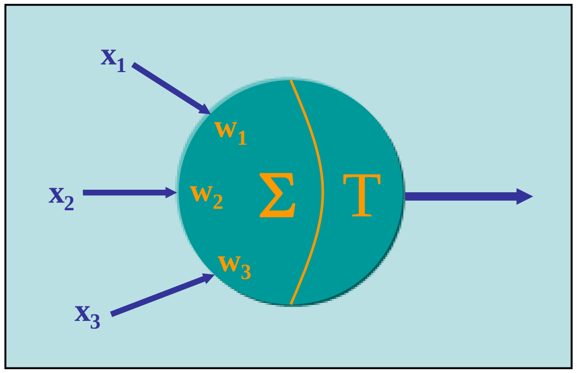
- Only one response variable at a time
- Different significance tests required for nominal and continuous responses
- Discriminate functions are often suboptimal due to orthogonal decision hyperplanes
- No proof of ability to learn arbitrary functions
- Can have difficulties with noisey data

Common NN Features

- “weight” = strength of connection
- threshold = value of weighted input below which no response is produced
- signals may be:
  - real-valued, or
  - binary-valued:
    - “unipolar” {0, 1}
    - “bipolar” {-1, 1}

McCulloch/Pitts Neuron

- One of the first neuron models to be implemented
- Its output is 1 (fired) or 0
- Each input is weighted with weights in the range -1 to + 1
- It has a threshold value, T



The neuron fires if the following inequality is true:

$$x_1w_1 + x_2w_2 + x_3w_3 > T$$

Single Neuron Application

- **GOAL**: Construct an MCP (McCulloch/Pitts) neuron which will implement the function below:

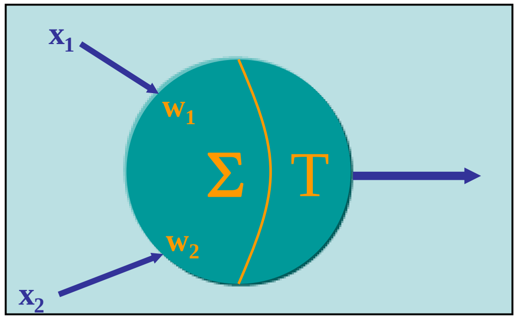
x1	x2	F
0	0	0
0	1	1
1	0	1
1	1	1

What is this function?

PROBLEM: find the threshold, T, and the weights w_1 and w_2 where:

$$\textbf{F is 1 if } x_1w_1 + x_2w_2 > T$$

Each line of the function table places conditions on the unknown values:



$0 < T$	$w_2 > T$
$w_1 > T$	$w_1 + w_2 > T$

w_1 and w_2 = 0.7 T = 0.5 works

Implement AND function using McCulloch–Pitts Neuron

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

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

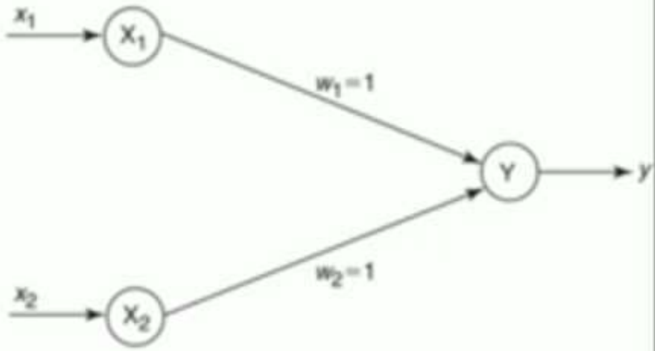
$(1, 1),\ y_{in} = x_1w_1 + x_2w_2 = 1 \times 1 + 1 \times 1 = 2$

$(1, 0),\ y_{in} = x_1w_1 + x_2w_2 = 1 \times 1 + 0 \times 1 = 1$

$(0, 1),\ y_{in} = x_1w_1 + x_2w_2 = 0 \times 1 + 1 \times 1 = 1$

$(0, 0),\ y_{in} = x_1w_1 + x_2w_2 = 0 \times 1 + 0 \times 1 = 0$

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



Implement XOR function using McCulloch–Pitts Neuron

- Consider the truth table for XOR function
- The M–P neuron has no particular training algorithm
- In M-P neuron, only analysis is being performed.
- XOR function cannot be represented by simple and single logic function; it is represented as

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

$$\underline{y = x_1\overline{x_2} + \overline{x_1}x_2}$$

Implement XOR function using McCulloch–Pitts Neuron

$$y = x_1 \overline{x_2} + \overline{x_1} x_2$$

$$y = z_1 + z_2$$

where

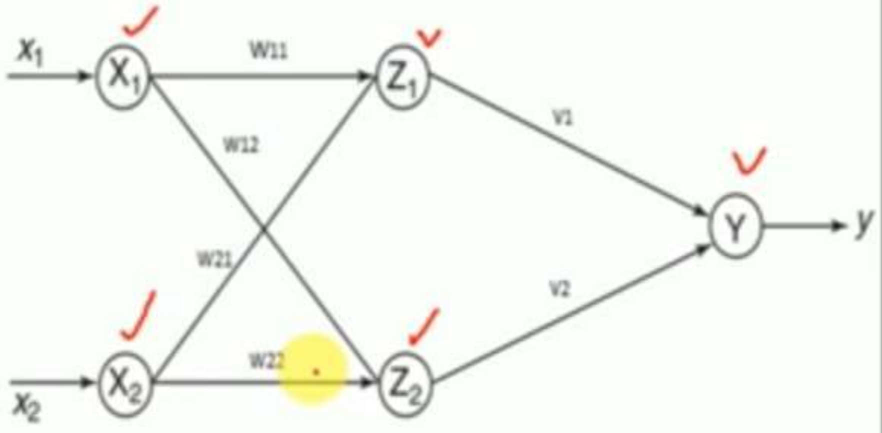
$z_1 = x_1 \overline{x_2}$ (function 1)

$z_2 = \overline{x_1} x_2$ (function 2)

$y = z_1$ (OR) z_2 (function 3)

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

- A single-layer net is not sufficient to represent the XOR function. We need to add an intermediate layer is necessary.



Implement XOR function using McCulloch–Pitts Neuron

- First function $z_1 = x_1 \overline{x_2}$
- The truth table for function z_1

$\theta = 1, 2$

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

- Assume the weights are initialized to

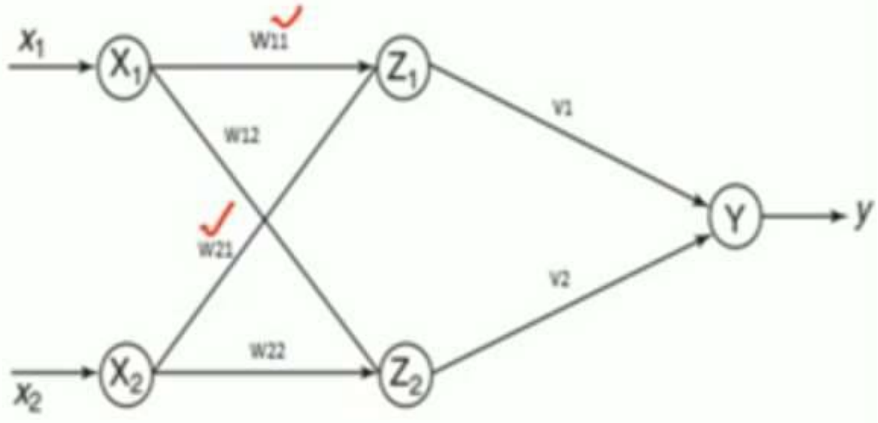
$$w_{11} = w_{21} = 1 \checkmark$$

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

- Calculate the net inputs,

$$(0, 0), z_{1in} = 0 \times 1 + 0 \times 1 = 0 \checkmark$$
$$(0, 1), z_{1in} = 0 \times 1 + 1 \times 1 = 1 \checkmark$$
$$(1, 0), z_{1in} = 1 \times 1 + 0 \times 1 = 1 \checkmark$$
$$(1, 1), z_{1in} = 1 \times 1 + 1 \times 1 = 2 \checkmark$$

- Hence, it is not possible to obtain function z_1 using these weights.



Implement XOR function using McCulloch–Pitts Neuron

- First function $z_1 = x_1 \overline{x_2}$
- The truth table for function z_1
- Assume the weights are initialized to

$$\underline{w_{11} = 1}; \quad \underline{w_{21} = -1}$$

- Calculate the net inputs,

$$\underline{(0, 0)}, z_{1in} = 0 \times 1 + 0 \times -1 = \underline{0} \; ✗$$

$$\underline{(0, 1)}, z_{1in} = 0 \times 1 + 1 \times -1 = \underline{-1} \; ✗$$

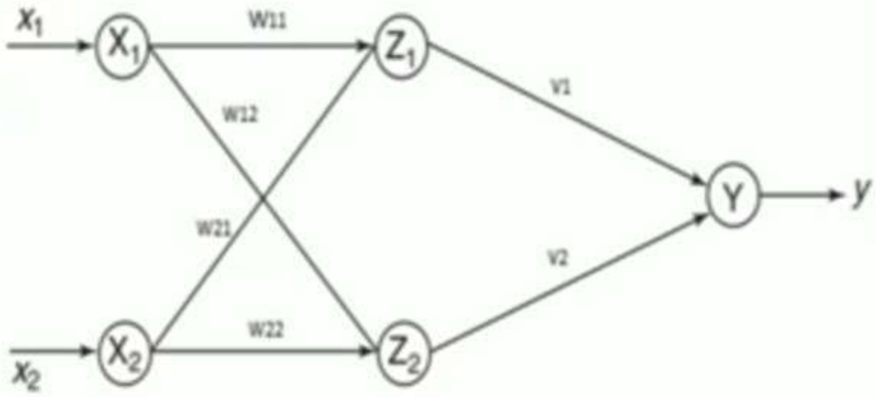
$$\underline{(1, 0)}, z_{1in} = 1 \times 1 + 0 \times -1 = \underline{1} \; ✓$$

$$\underline{(1, 1)}, z_{1in} = 1 \times 1 + 1 \times -1 = \underline{0} \; ✗$$

- If the $\underline{\theta=1}$ then the neuron fires.
- Hence $w_{11} = \underline{1}; \quad w_{21} = \underline{-1}$

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

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0



Implement XOR function using McCulloch–Pitts Neuron

- Second function $z_2 = \overline{x_1} x_2$
- The truth table for function z_2
- Assume the weights are initialized to

$$\underline{w_{12}} = \underline{w_{22} = 1}$$

- Calculate the net inputs,

$\underline{(0, 0)}, z_{2in} = 0 \times 1 + 0 \times 1 = \underline{0}$

$\underline{(0, 1)}, z_{2in} = 0 \times 1 + 1 \times 1 = \underline{1}$

$\underline{(1, 0)}, z_{2in} = 1 \times 1 + 0 \times 1 = \underline{1}$

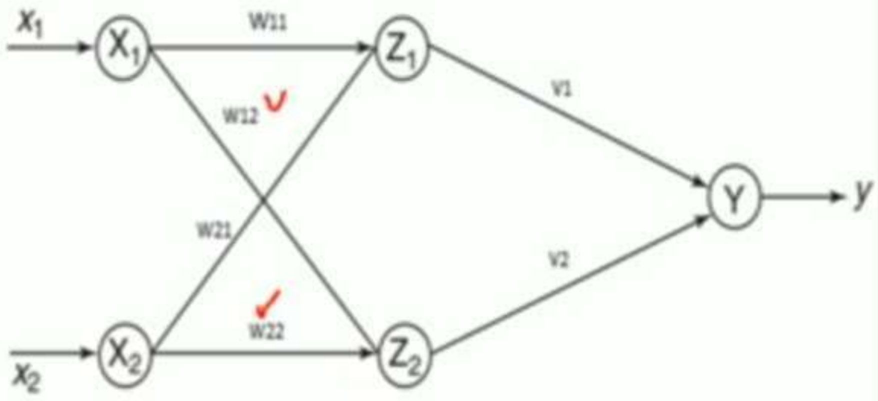
$(\underline{1,} 1), z_{2in} = 1 \times 1 + 1 \times 1 = \underline{2}$

- Hence, it is not possible to obtain function z_2 using these weights.

$\theta = 1$

x_1	x_2	z_2
0	0	0
✓0	1	1
1	0	0
1	1	0

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



Implement XOR function using McCulloch–Pitts Neuron

- Second function $z_2 = \overline{x_1} x_2$
- The truth table for function z_2
- Assume the weights are initialized to

$$w_{12} = w_{22} = 1$$

- Calculate the net inputs,

$(0, 0), z_{2in} = 0 \times 1 + 0 \times 1 = 0$ ✗ ✓

$(0, 1), z_{2in} = 0 \times 1 + 1 \times 1 = 1$ ✓ ✗

$(1, 0), z_{2in} = 1 \times 1 + 0 \times 1 = 1$ ✓ ✗

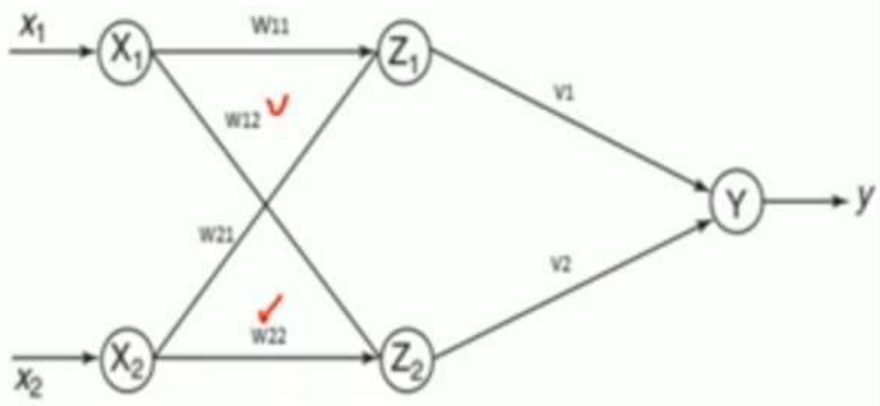
$(1, 1), z_{2in} = 1 \times 1 + 1 \times 1 = 2$ ✓ ✓

- Hence, it is not possible to obtain function z_2 using these weights.

$\theta = 1, 2$

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

x_1	x_2	z_2
0	0	0
✓0	1	1
1	0	0
1	1	0



Implement XOR function using McCulloch–Pitts Neuron

- Second function $z_2 = \overline{x_1} x_2$
- The truth table for function z_2

$\theta = 1$

x_1	x_2	z_2
0	0	0
0	1	1 ✓
1	0	0
1	1	0

- Assume the weights are initialized to

$$w_{12} = -1; \quad w_{22} = 1$$

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

- Calculate the net inputs,

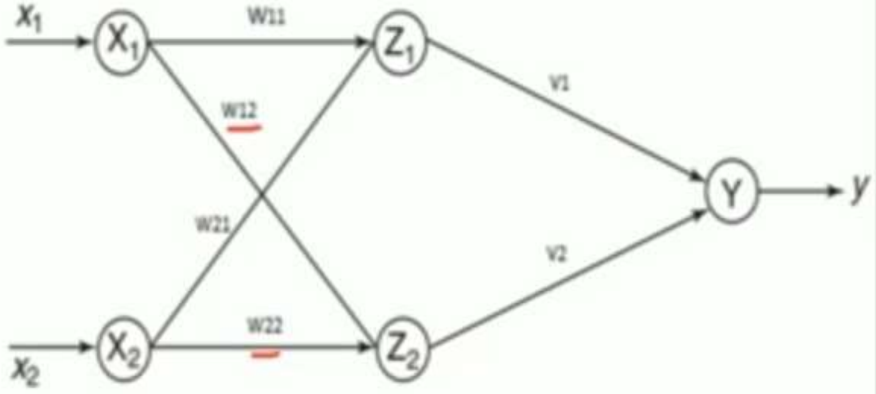
$(0, 0), z_{2in} = 0 \times -1 + 0 \times 1 = 0$ ✗

$(0, 1), z_{2in} = 0 \times -1 + 1 \times 1 = 1$ ✓

$(1, 0), z_{2in} = 1 \times -1 + 0 \times 1 = -1$ ✗

$(1, 1), z_{2in} = 1 \times -1 + 1 \times 1 = 0$ ✗

- If the θ=1 then the neuron fires.
- Hence $w_{12} = -1; \quad w_{22} = 1$



Implement XOR function using McCulloch–Pitts Neuron

- Third function $y = z_1 \text{ (OR) } z_2$
- The truth table for function y

$$y_{in} = z_1 v_1 + z_2 v_2$$

- Assume the weights are initialized to

$$v_1 = v_2 = 1 ✓$$

- Calculate the net inputs, $f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} \geq \theta \\ 0 & \text{if } y_{in} < \theta \end{cases}$

✓ $(0, 0), y_{in} = 0 \times 1 + 0 \times 1 = \underline{0}$ ✓ ✗

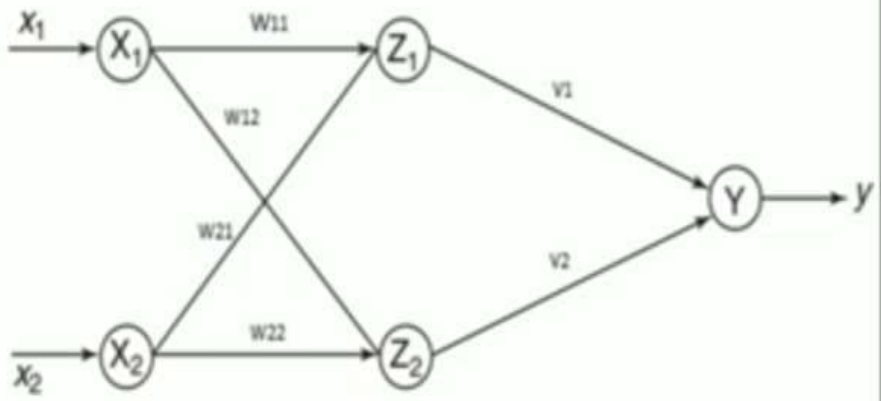
✓ $(0, 1), y_{in} = 0 \times 1 + 1 \times 1 = \underline{1}$ ✓ ✓

✓ $(1, 0), y_{in} = 1 \times 1 + 0 \times 1 = \underline{1}$ ✓ ✓

$(0, 0), y_{in} = 0 \times 1 + 0 \times 1 = \underline{0}$ ✓ ✗

$\theta = 0,1$

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1 ✓	0	1
1	0	1 ✓	1	0
1	1	0	0	0



Implement XOR function using McCulloch–Pitts Neuron

- Third function $y = z_1 \text{ (OR) } z_2$
- The truth table for function y

x_1	x_2	y	z_1	z_2
0	0	0	0	0
0	1	1 ✓	0	1
1	0	1 ✓	1	0
1	1	0	0	0

$$y_{in} = z_1 v_1 + z_2 v_2$$

$\theta = 0, 1$

- Assume the weights are initialized to

$$v_1 = v_2 = 1 \checkmark$$

- Calculate the net inputs,

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

$(0, 0), y_{in} = 0 \times 1 + 0 \times 1 = 0$ ✓ ✗

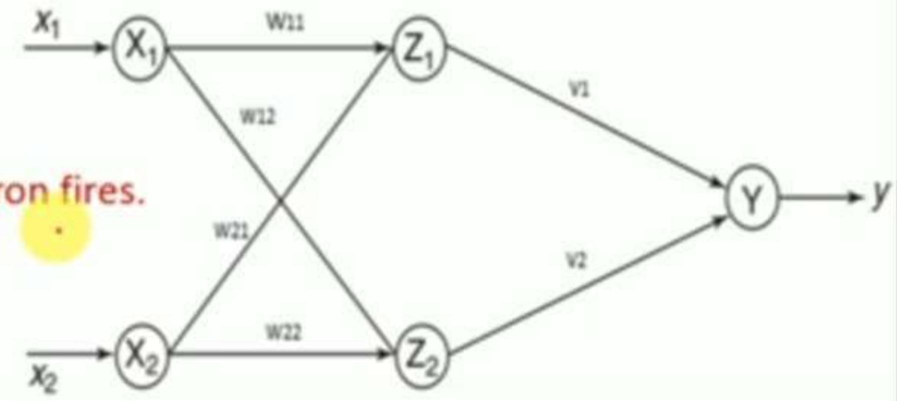
✓ $(0, 1), y_{in} = 0 \times 1 + 1 \times 1 = 1$ ✓

✓ $(1, 0), y_{in} = 1 \times 1 + 0 \times 1 = 1$ ✓

$(0, 0), y_{in} = 0 \times 1 + 0 \times 1 = 0$ ✓ ✗

If the $\theta = 1$ then the neuron fires.

Hence ✓ $v_1 = v_2 = 1$

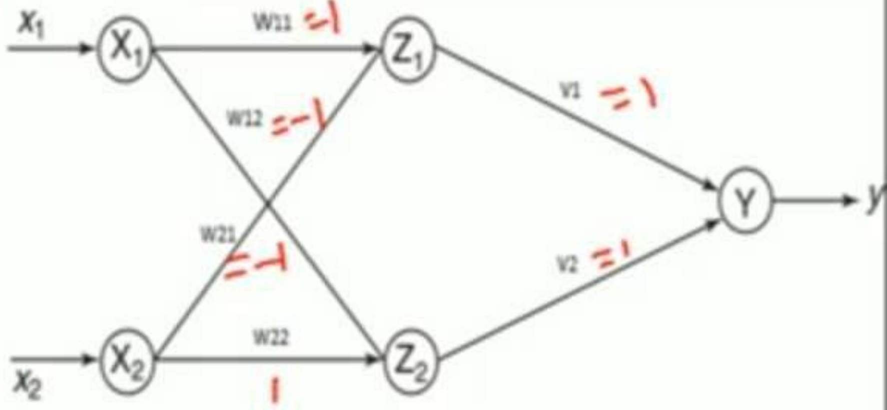


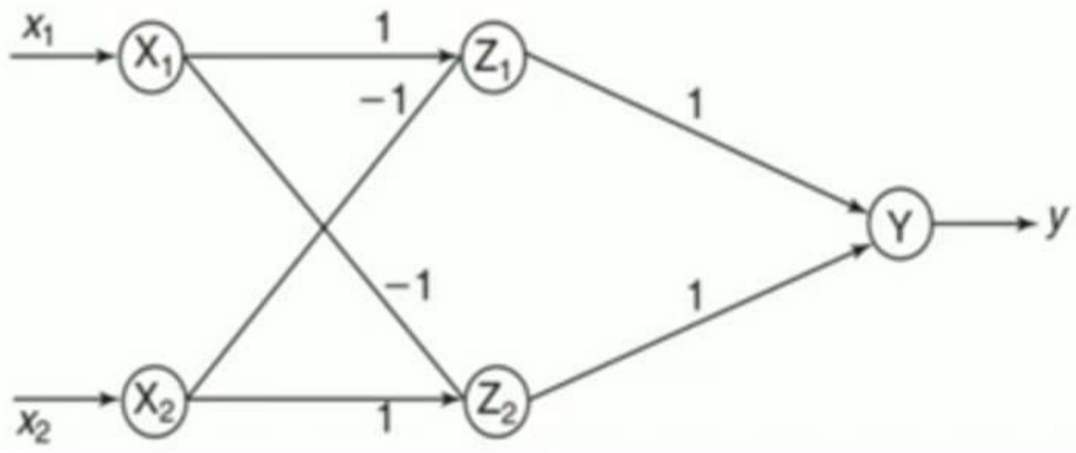
Implement XOR function using McCulloch–Pitts Neuron

$w_{11} = 1; \quad w_{21} = -1$

$w_{12} = -1; \quad w_{22} = 1$

$v_1 = v_2 = 1$





Exercise for McCulloch/Pitts

- **Write a python code for AND gate**
- **Write a python code for OR gate**
- **Write a python code for AND gate using function**
- **Write a python code for OR gate using function**

Exercise for McCulloch/Pitts

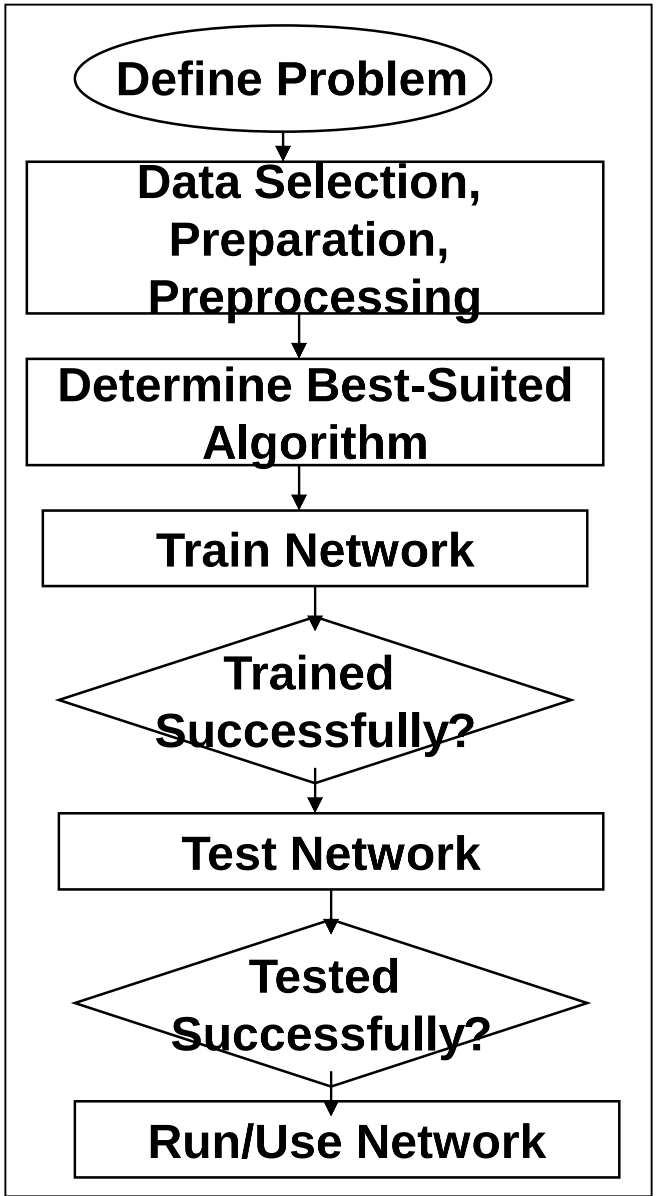
- **Write a python code for OR gate using function to print following:**

  OR Gate

X1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

NN Development

- There is no single methodology for NN development
  - A suggested approach is:



Learning in Neural Nets (again)

- The operation of a neural network is determined by the values of the interconnection weights
  - There is no algorithm that determines how the weights should be assigned in order to solve a specific problems
  - Hence, the weights are determined by a learning process

Hebbian Learning

- The oldest and most famous of all learning rules is Hebb’s postulate of learning:

“When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A’s efficiency as one of the cells firing B is increased”

Application to artificial neurons:

If two interconnected neurons are both “on” at the same time, then the weight between them should be increased

Hebb's Algorithm

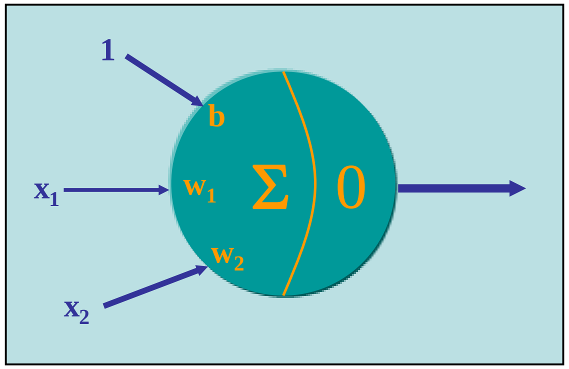
- **Step 0**: initialize all weights to 0
- **Step 1**: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$
- **Step 2**: Set the activation of the output unit to the target value: $y = t$
- **Step 3**: Adjust the weights: $w_i(\text{new}) = w_i(\text{old}) + x_i y$
- **Step 4**: Adjust the bias (just like the weights): $b(\text{new}) = b(\text{old}) + y$

Example

- **PROBLEM**: Construct a Hebb Net which performs like an AND function, that is, only when both features are "active" will the data be in the target class.

- **TRAINING SET** (with the bias input always at 1):

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



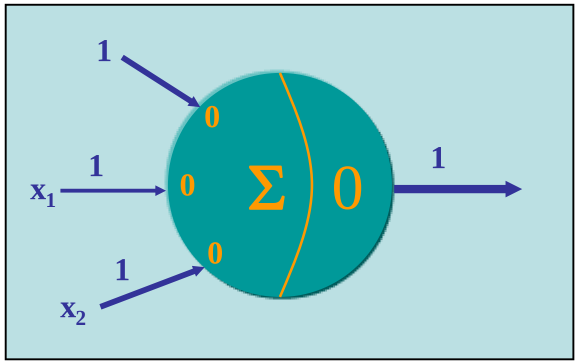
Training – First Input

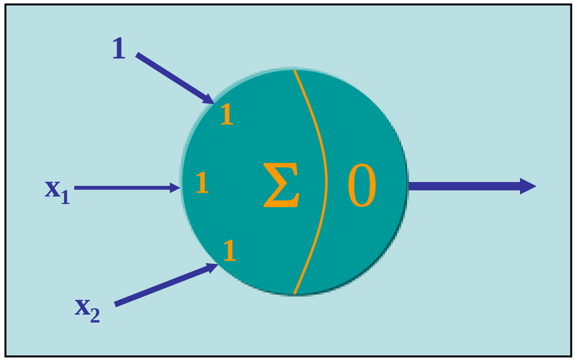
- Initialize the weights to 0

Present the first input (1 1 1) with a target of 1

Update the weights:

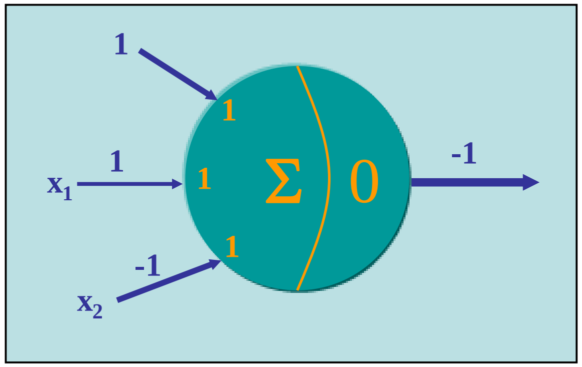
$$w_1(new) = w_1(old) + x_1 t$$
$$= \quad 0 \ + \ 1 = 1$$
$$w_2(new) = w_2(old) + x_2 t$$
$$= \quad 0 \ + \ 1 = 1$$
$$b(new) \quad = b(old) + t$$
$$= \quad 0 \ + \ 1 = 1$$





Training – Second Input

- Present the second input: (1 -1 1) with a target of -1



Update the weights:

$w_1(\text{new}) = w_1(\text{old}) + x_1 t$

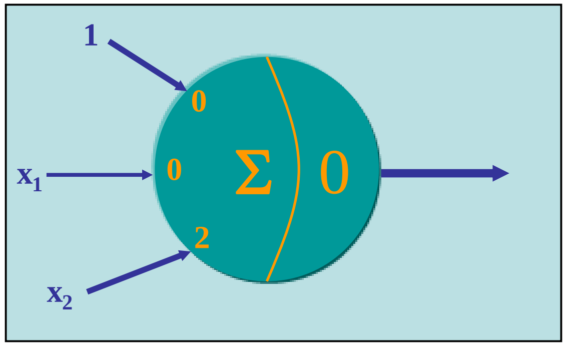
$= 1 + 1(-1) = 0$

$w_2(\text{new}) = w_2(\text{old}) + x_2 t$

$= 1 + (-1)(-1) = 2$

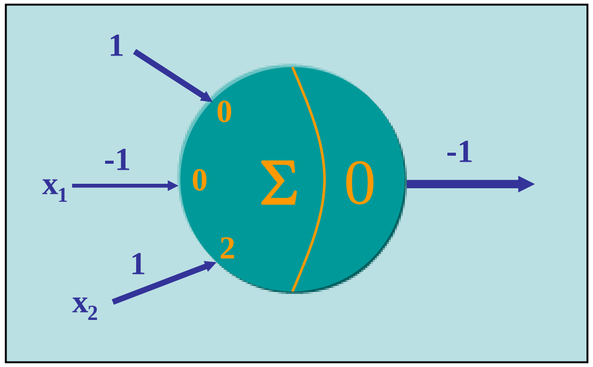
$b(\text{new}) = b(\text{old}) + t$

$= 1 + (-1) = 0$



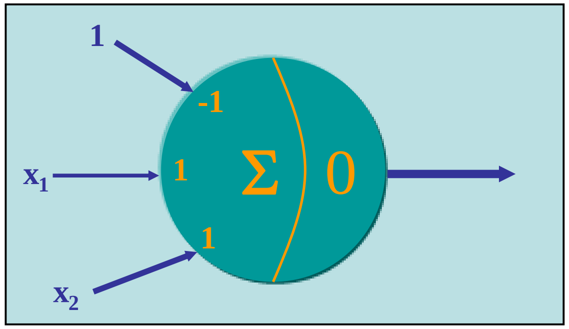
Training – Third Input

- Present the third input: (-1 1 1) with a target of -1



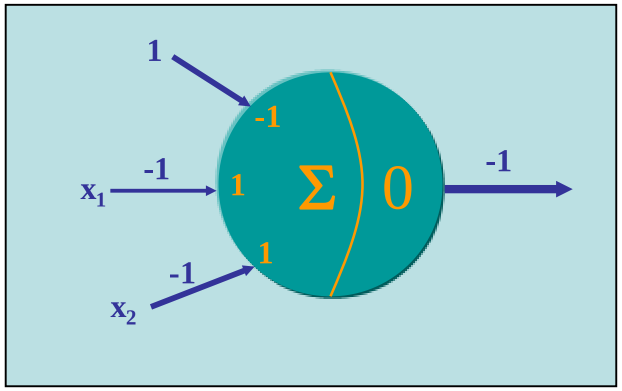
Update the weights:

$$w_1(\text{new}) = w_1(\text{old}) + x_1 t$$
$$= 0 + (-1)(-1) = 1$$
$$w_2(\text{new}) = w_2(\text{old}) + x_2 t$$
$$= 2 + 1(-1) = 1$$
$$b(\text{new}) = b(\text{old}) + t$$
$$= 0 + (-1) = -1$$



Training – Fourth Input

- Present the fourth input: (-1 -1 1) with a target of -1



Update the weights:

$w_1(\text{new}) = w_1(\text{old}) + x_1 t$

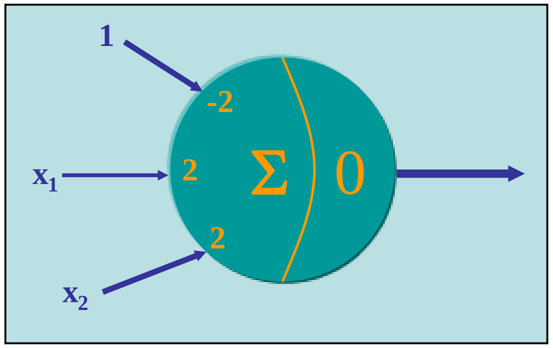
$= 1 + (-1)(-1) = 2$

$w_2(\text{new}) = w_2(\text{old}) + x_2 t$

$= 1 + (-1)(-1) = 1$

$b(\text{new}) = b(\text{old}) + t$

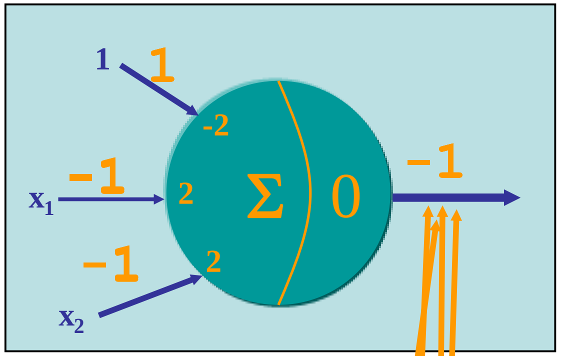
$= -1 + (-1) = -2$



Final Neuron

- This neuron works:

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



$$1*2 + 1*2 + 1*(-2) = 2 > 0$$
$$(-1)*2 + 1*2 +1*(-2) = -2 < 0$$
$$1*2 + (-1)*2 + 1*(-2) = -2 < 0$$
$$(-1)*2 + (-1)*2 + 1 *(-2) = -6 <0$$

Review

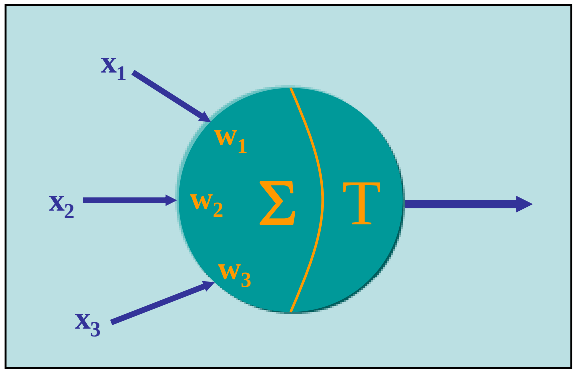
•Features of Neural Nets

•Learning in Neural Nets

•McCulloch/Pitts Neuron and Hebbian Learning

Review – MCP Neuron

- One of the first neuron models to be implemented
- Its output is 1 (fired) or 0
- Each input is weighted with weights in the range -1 to + 1
- It has a threshold value, T



The neuron fires if the following inequality is true:

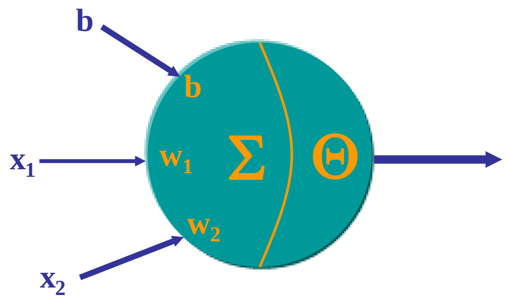
$$x_1w_1 + x_2w_2 + x_3w_3 > T$$

OUTLINE

- Introduction to the Perceptron
- Adaline Network

The Perceptron

- The perceptron was suggest by Rosenblatt in 1958.
- It uses an iterative learning procedure which can be proven to converge to the correct weights for linearly separable data
- It has a bias and a threshold function



Activation function:

$$f(s) = \begin{cases} 1 \text{ if } s > \Theta \\ 0 \text{ if } -\Theta <= s <= \Theta \\ -1 \text{ if } s < -\Theta \end{cases}$$

Perceptron Learning Rule

- Weights are changed only when an error occurs

- The weights are updated using the following:

$$\mathbf{w_i(new) = w_i(old) + \alpha t x_i}$$

t is either +1 or -1
α is the learning rate

If an error does not occur, the weights are not changed

Perceptron Training Algorithm - Single Output Class

T

- Initialize the weights and the bias. Also initialize the learning rate ά (0< ά ≤ 1).
- Until the final stopping condition is false.
  - for each training pair indicated by s:t.
    - Set each input unit i = 1 to n: $x_i = s_i$
    - Calculate the output of the network.

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i \qquad \boxed{y} = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > \theta \\ 0 & \text{if } -\theta \le y_{in} \le \theta \\ -1 & \text{if } y_{in} < -\theta \end{cases}$$

    - Weight and bias adjustment:

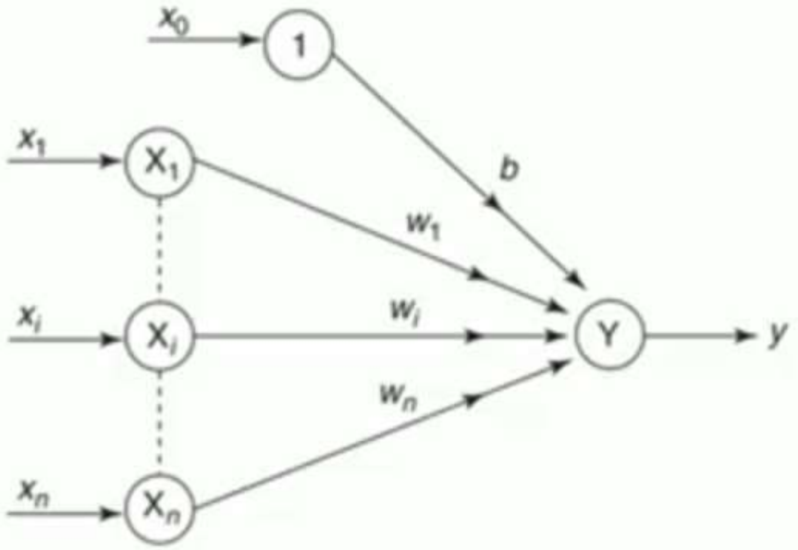
If $y \neq t$, then

$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else we have

$$w_i(\text{new}) = w_i(\text{old})$$
$$b(\text{new}) = b(\text{old})$$

  - Train the network until there is no weight change.



Perceptron Training Algorithm - Multiple Output Class

- Initialize the weights and the bias. Also initialize the learning rate ά (0< ά ≤ 1).
- Until the final stopping condition is false.
  - for each training pair indicated by s : t.
    - Set each input unit i = 1 to n: $x_i = s_i$
    - Calculate the output of the network.

$$y_{in_j} = b_j + \sum_{i=1}^{n} x_i w_{ij} \qquad y_j = f(y_{in_j}) = \begin{cases} 1 & \text{if } y_{in_j} > \theta \\ 0 & \text{if } -\theta \le y_{in_j} \le \theta \\ -1 & \text{if } y_{in_j} < -\theta \end{cases}$$

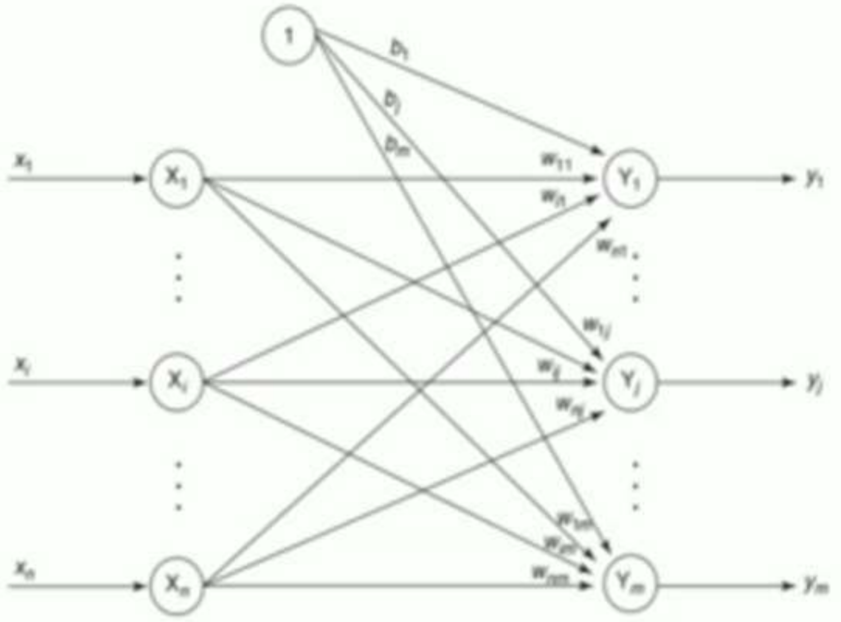
    - Weight and bias adjustment:

If $t_j \neq y_j$, then

$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha t_j x_i$$
$$b_j(\text{new}) = b_j(\text{old}) + \alpha t_j$$

else, we have

$$w_{ij}(\text{new}) = w_{ij}(\text{old})$$
$$b_j(\text{new}) = b_j(\text{old})$$



Perceptron Network Testing Algorithm

- The initial weights to be used here are taken from the training algorithms (the final weights obtained during training).
- For each input vector X to be classified, perform the following
  - Calculate the net input of the unit.
  - Obtain the response of output unit.

$$y_{in} = \sum_{i=1}^{n} x_i w_i$$

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

Perceptron Learning Rule

- In case of the perceptron learning rule, the learning signal is the difference between the calculated output and actual (target) output of a neuron.
- The output "y" is obtained on the basis of the net input calculated and activation function being applied over the net input.

$$\underline{y_{in}} = b + \sum_{i=1}^{n} x_i w_i$$

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

- Weights are updated using the formula

If $\underline{y \neq t}$, then

$$\underline{w}(\text{new}) = w(\underline{\text{old}}) + \underline{\alpha t x} \quad (\alpha - \text{learning rate})$$

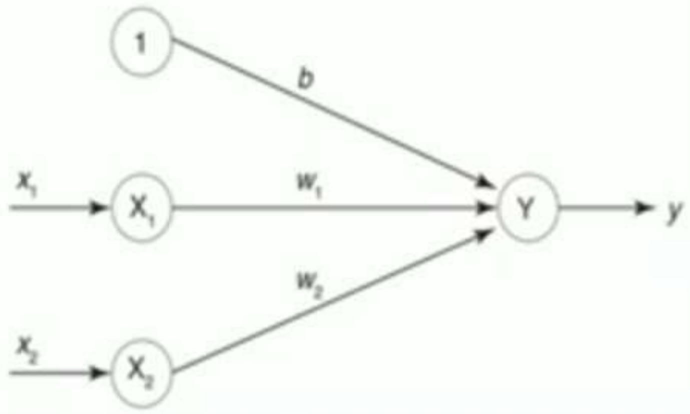
else, we have

$$w(\text{new}) = w(\text{old})$$

AND function using Perceptron Rule Solved Example

- The perceptron network, which uses perceptron learning rule, is used to train the AND function.
- The input patterns are presented to the network one by one.
- When all the four input patterns are presented, then one epoch is said to be completed.
- The initial weights and threshold are set to zero.
- The learning rate a is set equal to 1.

x_1	x_2	t
1	1	1
1	−1	−1
−1	1	−1
−1	−1	−1



AND function using Perceptron Rule Solved Example

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

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

$$\underline{\Delta w_1} = \underline{\alpha t x_1};$$
$$\underline{\Delta w_2} = \alpha t x_2;$$
$$\underline{\Delta b} = \alpha t$$

Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	−1	−1	1	1	−1	1	−1	0	2	0
−1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1

AND function using Perceptron Rule Solved Example

1 + 1 + (-1)

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

0 + 1×0 + 1×0 = 0

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

$\Delta w_1 = \alpha t x_1;$

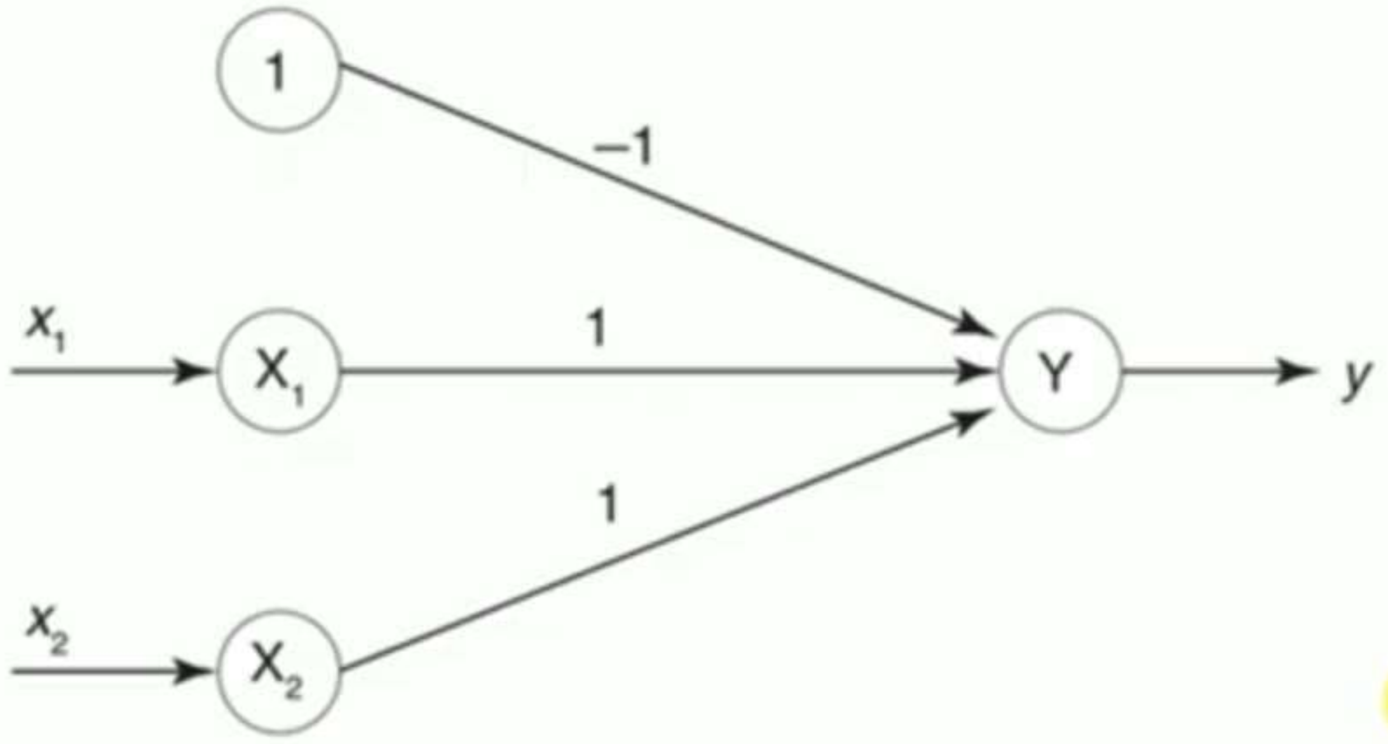
$\Delta w_2 = \alpha t x_2;$

$\Delta b = \alpha t$

$\alpha = 1$

Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1 ✓	x_2 ✓	(t) ✓	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1										
✓ 1	1	1	0 ✓	0 ✓	1	1	1	1	1	1
✓ 1	−1	−1	1	1	−1	1	−1	0	2	0
✓ −1	1	−1	2	1	+1	−1	−1	1	1	−1
✓ −1	−1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2										
✓ 1	1	1 ✓	1	1	0	0	0	1	1	−1
✓ 1	−1	−1 ✓	−1	−1	0	0	0	1	1	−1
✓ −1	1	−1 ✓	−1	−1	0	0	0	1	1	−1
✓ −1	−1	−1 ✓	−3	−1	0	0	0	1	1	−1

AND function using Perceptron Rule Solved Example



Limitations of the Perceptron

The perceptron can only learn to distinguish between classifications if the classes are *linearly separable*.

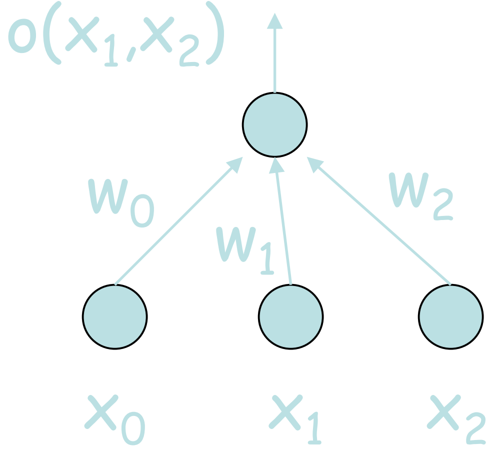
If the problem is not linearly separable then the behaviour of the algorithm is not guaranteed.

If the problem is linearly separable, there may be a number of possible solutions.

The algorithm as stated gives no indication of the quality of the solution found.

XOR Problem

- A perceptron network can not implement an XOR function



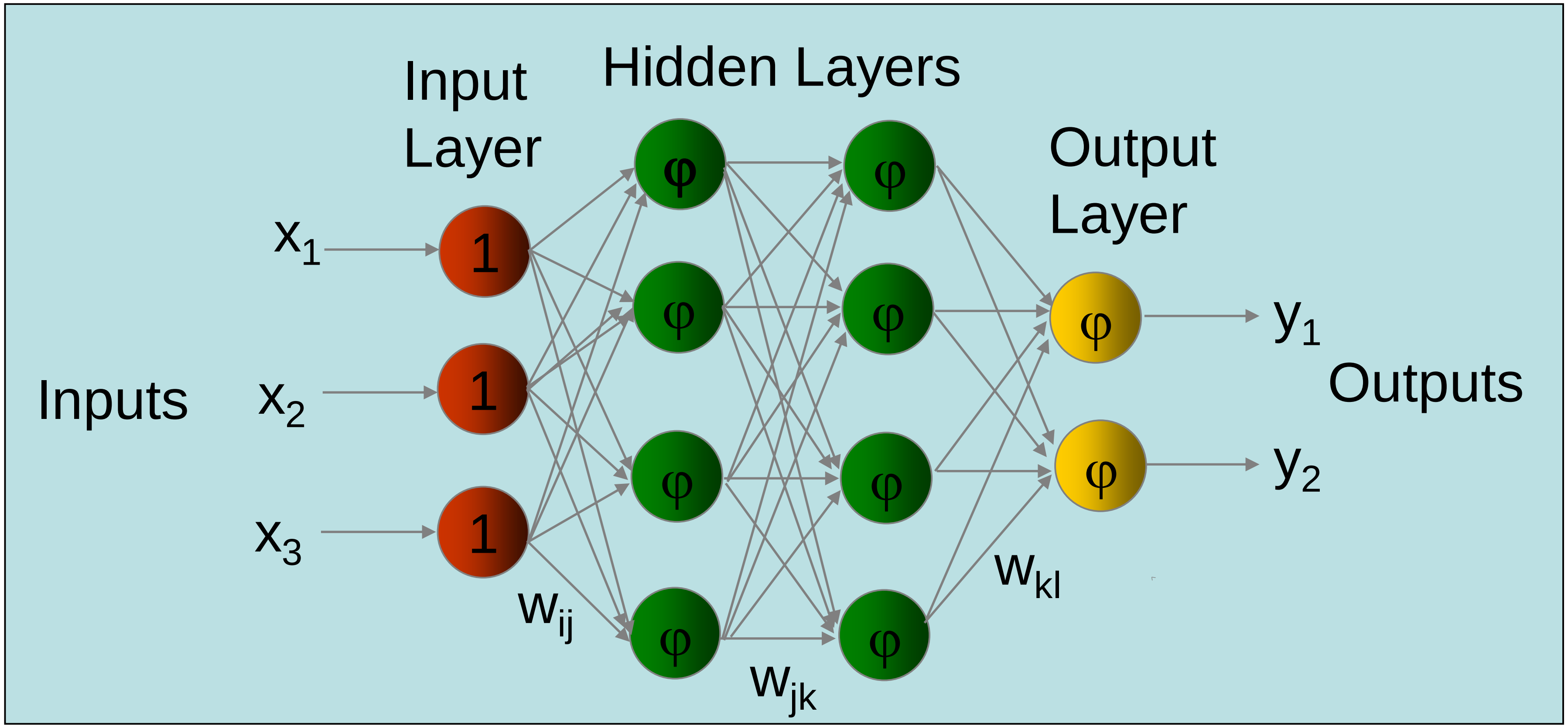
$$\left.\begin{array}{l} w_0 + 0.w_1 + 0.w_2 \le 0 \\ w_0 + 0.w_1 + 1.w_2 > 0 \\ w_0 + 1.w_1 + 0.w_2 > 0 \\ w_0 + 1.w_1 + 1.w_2 \le 0 \end{array}\right\} \mathrm{XOR}(x_1,x_2)$$

INPUTS ARE DIFFERENT – 1 OUT

There is no assignment of values to w_0,w_1 and w_2 that satisfies above inequalities. XOR cannot be represented!

MultiLayer Perceptrons

- Perceptrons can be improved if placed in a multilayered network



Adaline Network

- Variation on the Perceptron Network
  - inputs are +1 or -1
  - outputs are +1 or -1
  - uses a bias input

- Differences
  - trained using the Delta Rule which is also known as the least mean squares (LMS) or Widrow-Hoff rule
  - the activation function, during training is the identity function
  - after training the activation is a threshold function

Adaline Algorithm

- **Step 0: initialize the weights to small random values and select a learning rate, α**
- **Step 1: for each input vector s, with target output, t set the inputs to s**
- **Step 2: compute the neuron inputs**
- **Step 3: use the delta rule to update the bias and weights**
- **Step 4: stop if the largest weight change across all the training samples is less than a specified tolerance, otherwise cycle through the training set again**

Neuron input

$$y\_in = b + \sum x_i w_i$$

Delta rule

$$b(new) = b(old) + \alpha(t - y\_in)$$
$$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$$

The Learning Rate, α

- The performance of an ADALINE neuron depends heavily on the choice of the learning rate
  - if it is too large the system will not converge
  - if it is too small the convergence will take to long

- Typically, α is selected by trial and error
  - typical range: $0.01 < \alpha < 10.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1 < n\alpha < 1.0$
    where n is the number of inputs

Running Adaline

- One unique feature of ADALINE is that its activation function is different for training and running

- When running ADALINE use the following:
  - initialize the weights to those found during training
  - compute the net input
  - apply the activation function

Neuron input

$$y\_in = b + \sum x_i w_i$$

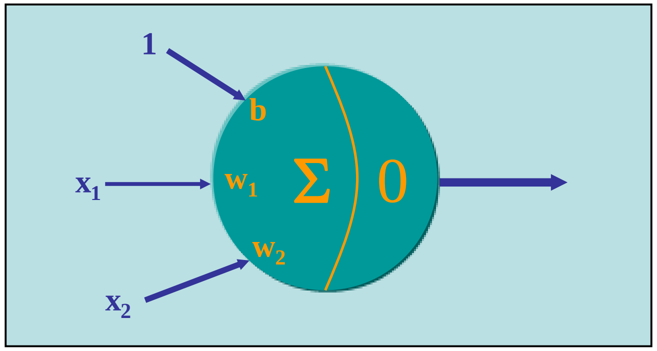
Activation Function

$$y = \begin{cases} 1 \text{ if } y\_in >= 0 \\ -1 \text{ if } y\_in < 0 \end{cases}$$

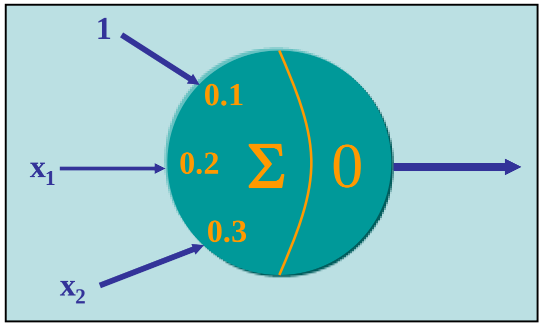
Example – AND function

- Construct an AND function for a ADALINE neuron
  - let $\alpha = 0.1$

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Initial Conditions: Set the weights to small random values:

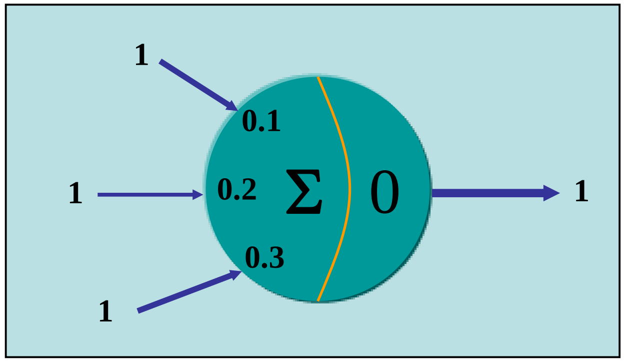


First Training Run

- Apply the input (1,1) with output 1

The net input is:

$y\_in = 0.1 + 0.2*1 + 0.3*1 = 0.6$



The new weights are:

$b = 0.1 + 0.1(1-0.6) = 0.14$
$w_1 = 0.2 + 0.1(1-0.6)1 = 0.24$
$w_2 = 0.3 + 0.1(1-0.6)1 = 0.34$

The largest weight change is 0.04

Neuron input

$y\_in = b + \sum x_i w_i$

Delta rule

$b(new) = b(old) + \alpha(t - y\_in)$
$w_i(new) = w_i(old) + \alpha(t - y\_in)x_i$

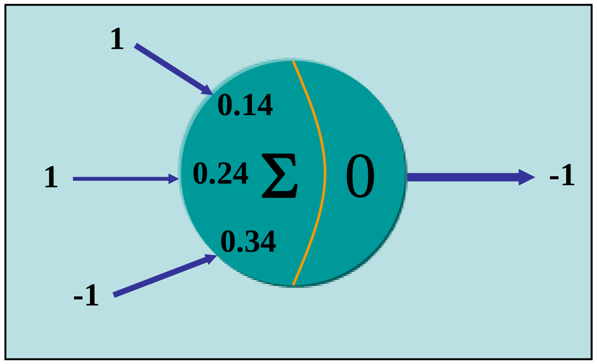
Second Training Run

- Apply the second training set (1 -1) with output -1

The net input is:

$$y\_in = 0.14 + 0.24*1 + 0.34*(-1) = 0.04$$

The new weights are:



$b = 0.14 - 0.1(1+0.04) = 0.04$

$w_1 = 0.24 - 0.1(1+0.04)1 = 0.14$

$w_2 = 0.34 + 0.1(1+0.04)1 = 0.44$

The largest weight change is 0.1

Third Training Run

- Apply the third training set (-1 1) with output -1

The net input is:

$y\_in = 0.04 - 0.14*1 + 0.44*1 = 0.34$

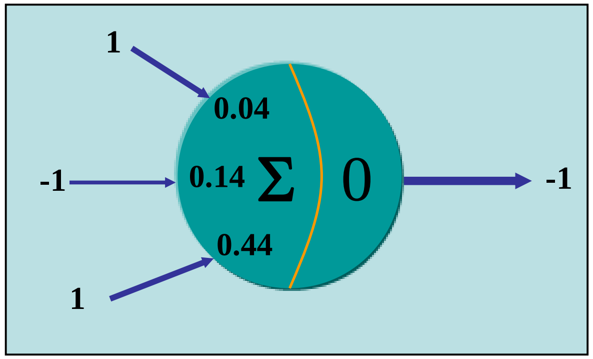
The new weights are:

$b = 0.04 - 0.1(1+0.34) = -0.09$

$w_1 = 0.14 + 0.1(1+0.34)1 = 0.27$

$w_2 = 0.44 - 0.1(1+0.34)1 = 0.31$

The largest weight change is 0.13



Fourth Training Run

- Apply the fourth training set (-1 -1) with output -1

The net input is:

y_in = -0.09 - 0.27*1 - 0.31*1 = -0.67

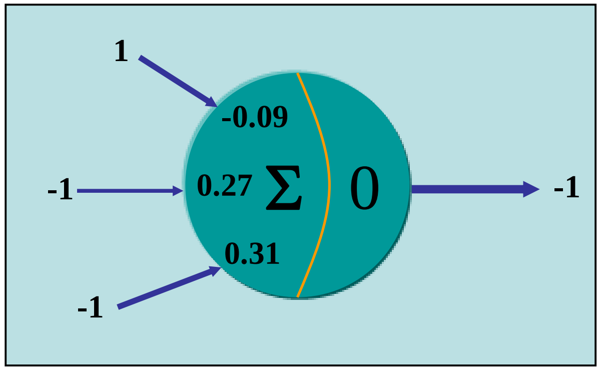
The new weights are:

b = -0.09 - 0.1(1+0.67) = -0.27

w_1 = 0.27 + 0.1(1+0.67)1 = 0.43

w_2 = 0.31 + 0.1(1+0.67)1 = 0.47

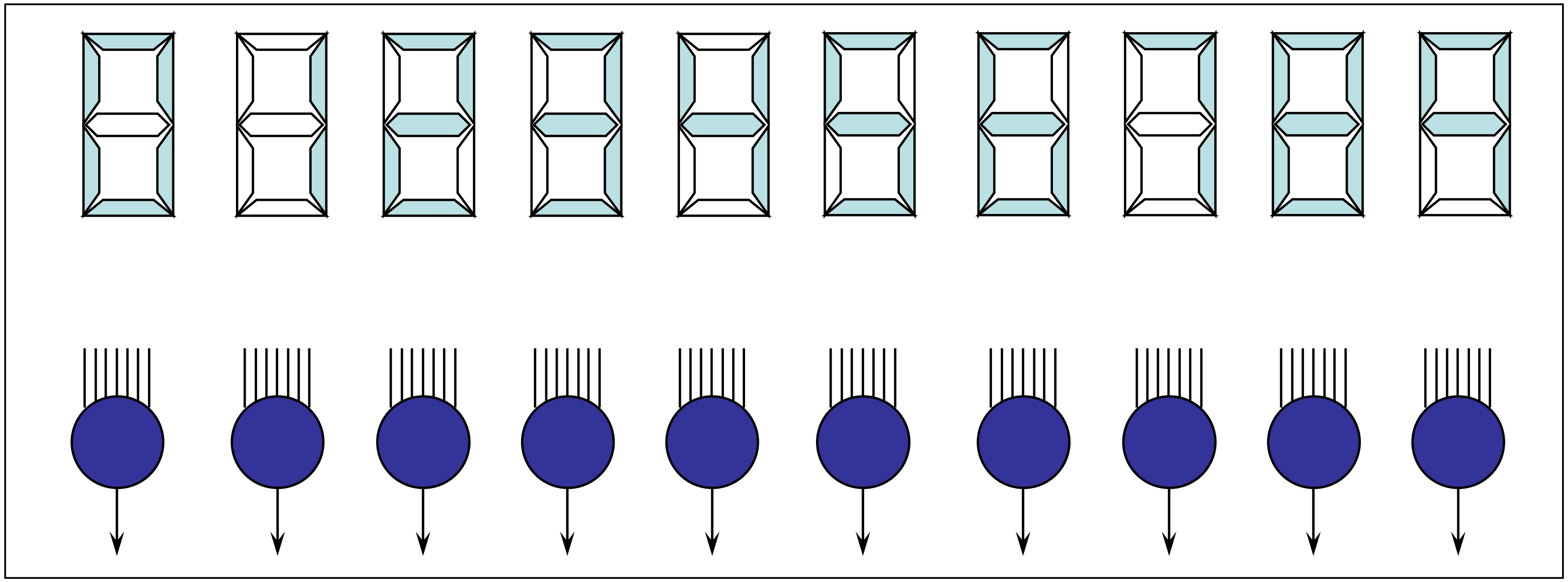
The largest weight change is 0.16



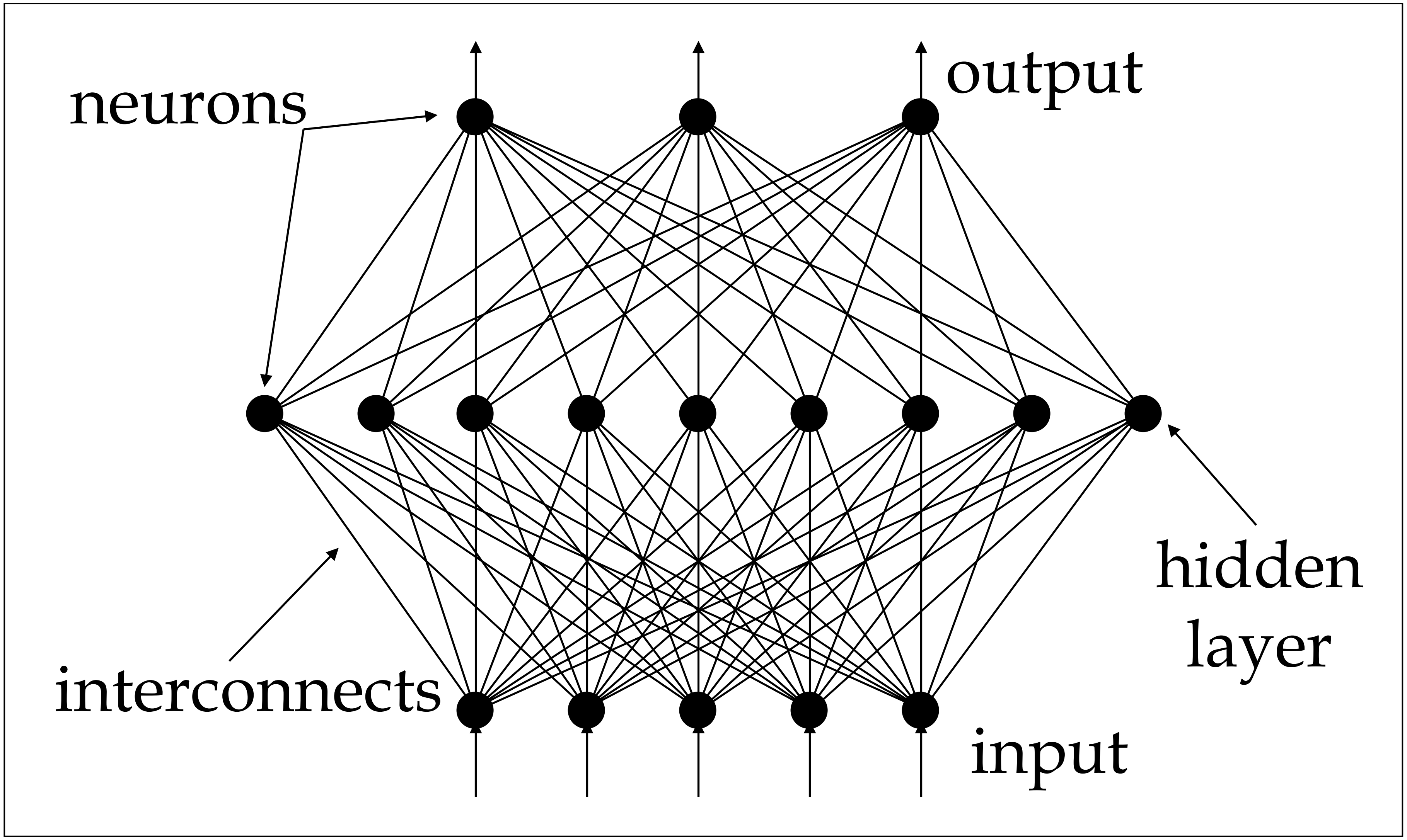
Conclusion

- Continue to cycle through the four training inputs until the largest change in the weights over a complete cycle is less than some small number (say 0.01)
- In this case, the solution becomes: $b = -0.5, w_1 = 0.5, w_2 = 0.5$

Example – Recognize Digits



Multi-Layered Net (again)



Properties

- Patterns of activation are presented at the inputs and the resulting activation of the outputs is computed.

- The values of the weights determine the function computed. A network with one hidden layer is sufficient to represent **every Boolean function**. With real weights **every real valued function** can be approximated with a single hidden layer.

Training

- Given Training Data,
  - input vector set :

    $\{ \boldsymbol{i}_{\boldsymbol{n}} \mid 1 < n < N \}$
  - corresponding output (target) vector set:

    $\{ \boldsymbol{t}_{\boldsymbol{n}} \mid 1 < n < N \}$

- Find the weights of the interconnects using training data to ***minimize the error*** in the test data

Error

- Input, target & response
  - input vector set : { $\boldsymbol{i}_{\boldsymbol{n}}$ | 1 < n < N }
  - target vector set: { $\boldsymbol{t}_{\boldsymbol{n}}$ | 1 < n < N }
  - $\boldsymbol{o}_n$ = neural network output when the input is $\boldsymbol{i}_{\boldsymbol{n}}$

- Error

$$\boxed{E = \frac{1}{2} \sum (o_n - t_n)^2}$$

Error Minimization Techniques

- The error is a function of the
  - fixed training and test data
  - neural network weights

- Find weights that minimize error (Standard Optimization)
  - conjugate gradient descent
  - random search
  - genetic algorithms
  - steepest descent (error backpropagation)

Possible Quiz

What is a limitation of the perceptron?

What are the names of the layers in a multilayer net?

What is the role of training?

SUMMARY

- **Introduction to the Perceptron**
- **Adaline Network**

Thank you for your attention