Artificial Neural Network

Lecture Module 22

Neural Networks

- Artificial neural network (ANN) is a machine learning approach that models human brain and consists of a number of artificial neurons.
- Neuron in ANNs tend to have fewer connections than biological neurons.
- Each neuron in ANN receives a number of inputs.
- An activation function is applied to these inputs which results in activation level of neuron (output value of the neuron).
- Knowledge about the learning task is given in the form of examples called training examples.

Contd..

- An Artificial Neural Network is specified by:
  - **neuron model**: the information processing unit of the NN,
  - **an architecture**: a set of neurons and links connecting neurons. Each link has a weight,
  - **a learning algorithm**: used for training the NN by modifying the weights in order to model a particular learning task correctly on the training examples.
- The aim is to obtain a NN that is trained and generalizes well.
- It should behaves correctly on new instances of the learning task.

Neuron

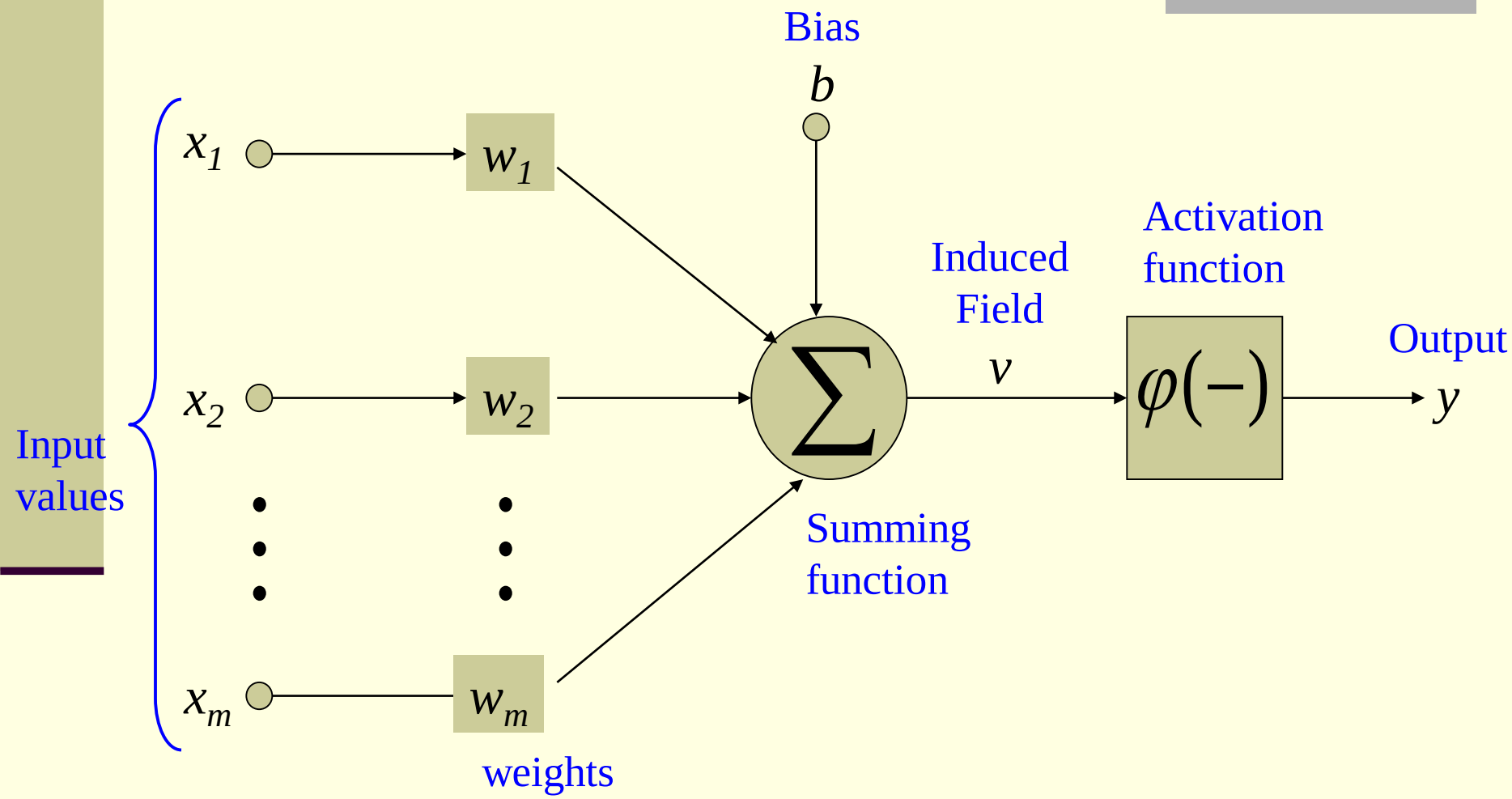
- The neuron is the basic information processing unit of a NN. It consists of:
  1. A set of links, describing the neuron inputs, with weights $W_1, W_2, \ldots, W_m$
  2. An adder function (linear combiner) for computing the weighted sum of the inputs:
     (real numbers)

$$\mathbf{u} = \sum_{j=1}^{\mathbf{m}} \mathbf{w_j x_j}$$

  3. Activation function φ for limiting the amplitude of the neuron output. Here 'b' denotes bias.

$$\mathrm{y} = \varphi(\mathrm{u} + b)$$

The Neuron Diagram



Bias of a Neuron

- The bias ***b*** has the effect of applying a transformation to the weighted sum ***u***

  v = u + b

- The bias is an external parameter of the neuron. It can be modeled by adding an extra input.
- ***v*** is called **induced field** of the neuron

$$v = \sum_{j=0}^{m} w_j x_j$$

$$w_0 = b$$

Neuron Models

- The choice of activation function φ determines the neuron model.

Examples:

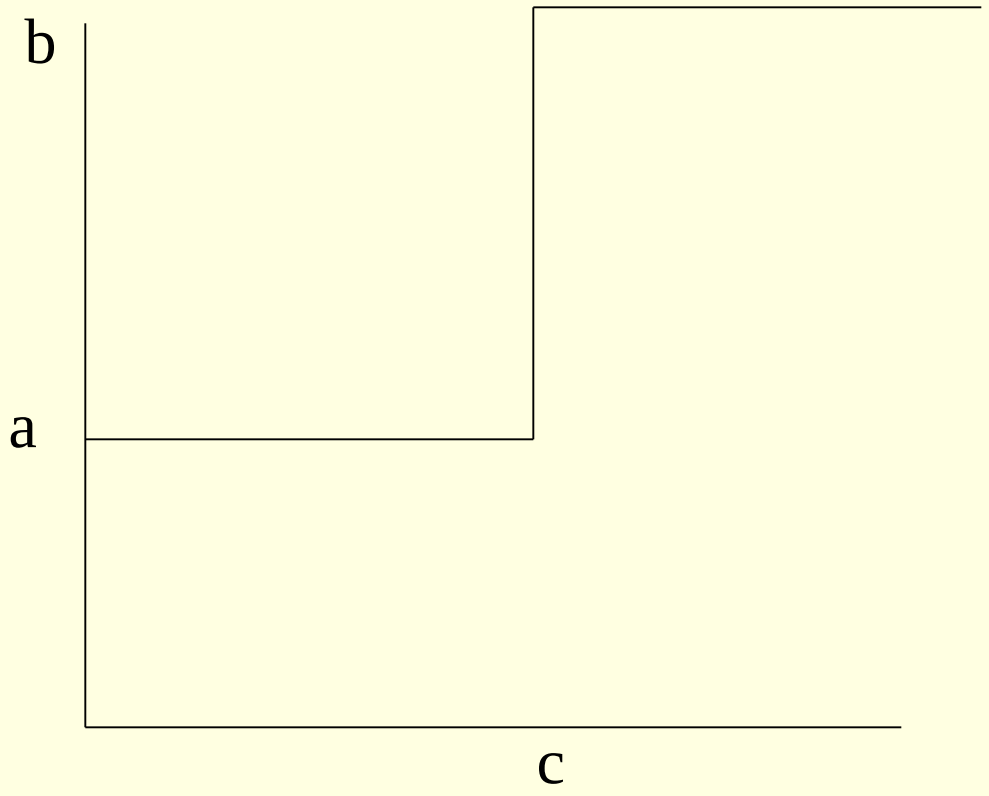
- step function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > c \end{cases}$$

- ramp function: $$\varphi(v) = \begin{cases} a & \text{if } v < c \\ b & \text{if } v > d \\ a + ((v-c)(b-a)/(d-c)) & \text{otherwise} \end{cases}$$

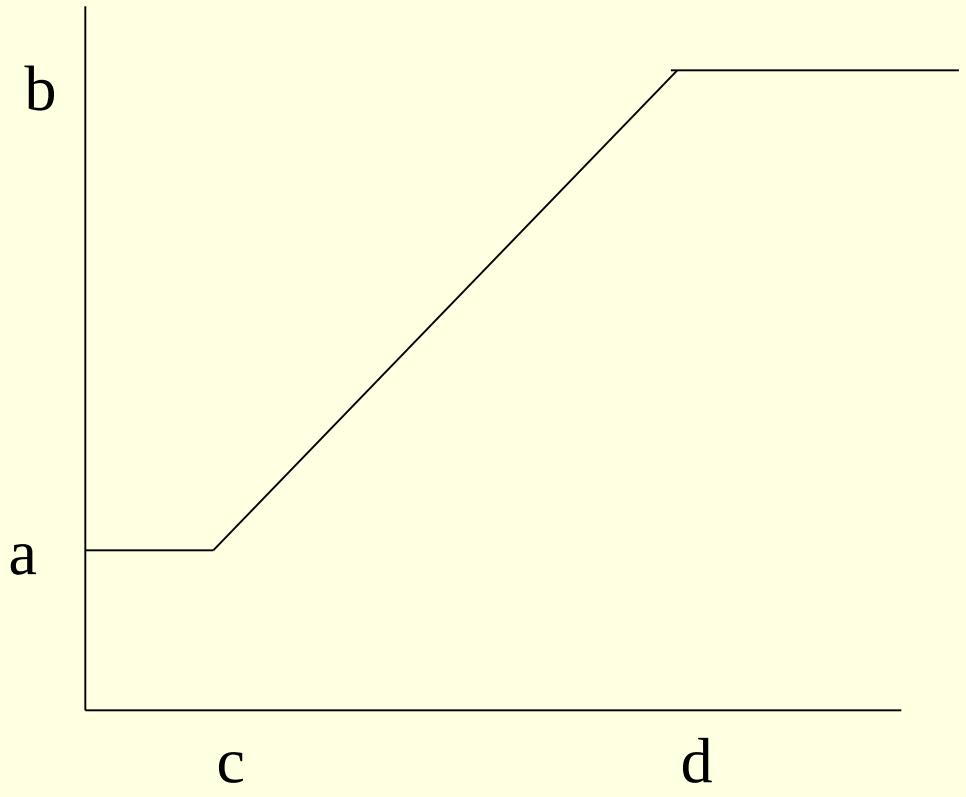
- sigmoid function with z,x,y parameters $$\varphi(v) = z + \frac{1}{1+\exp(-xv+y)}$$

- Gaussian function: $$\varphi(v) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\left(\frac{v-\mu}{\sigma}\right)^2\right)$$

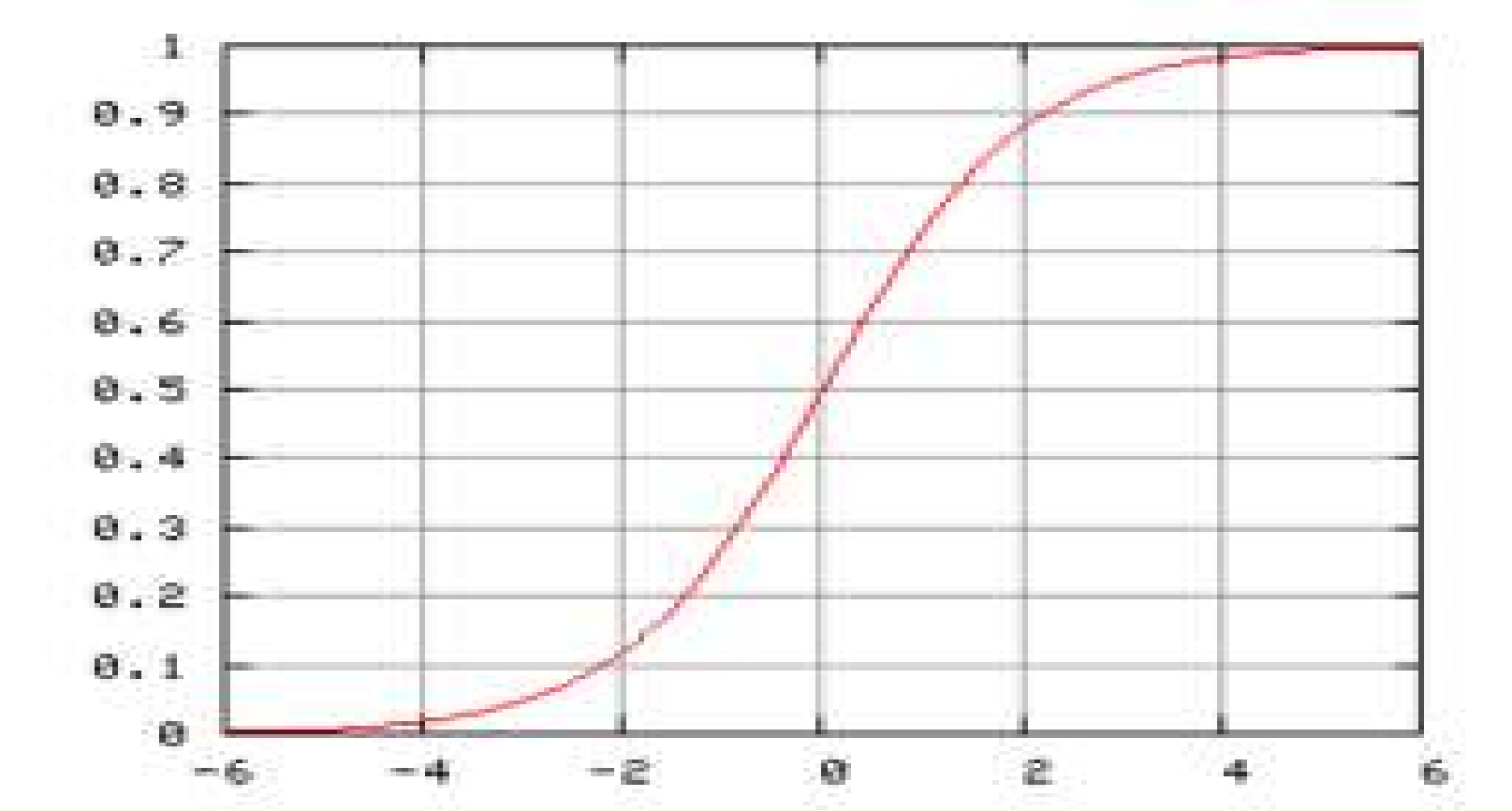
Step Function



Ramp Function

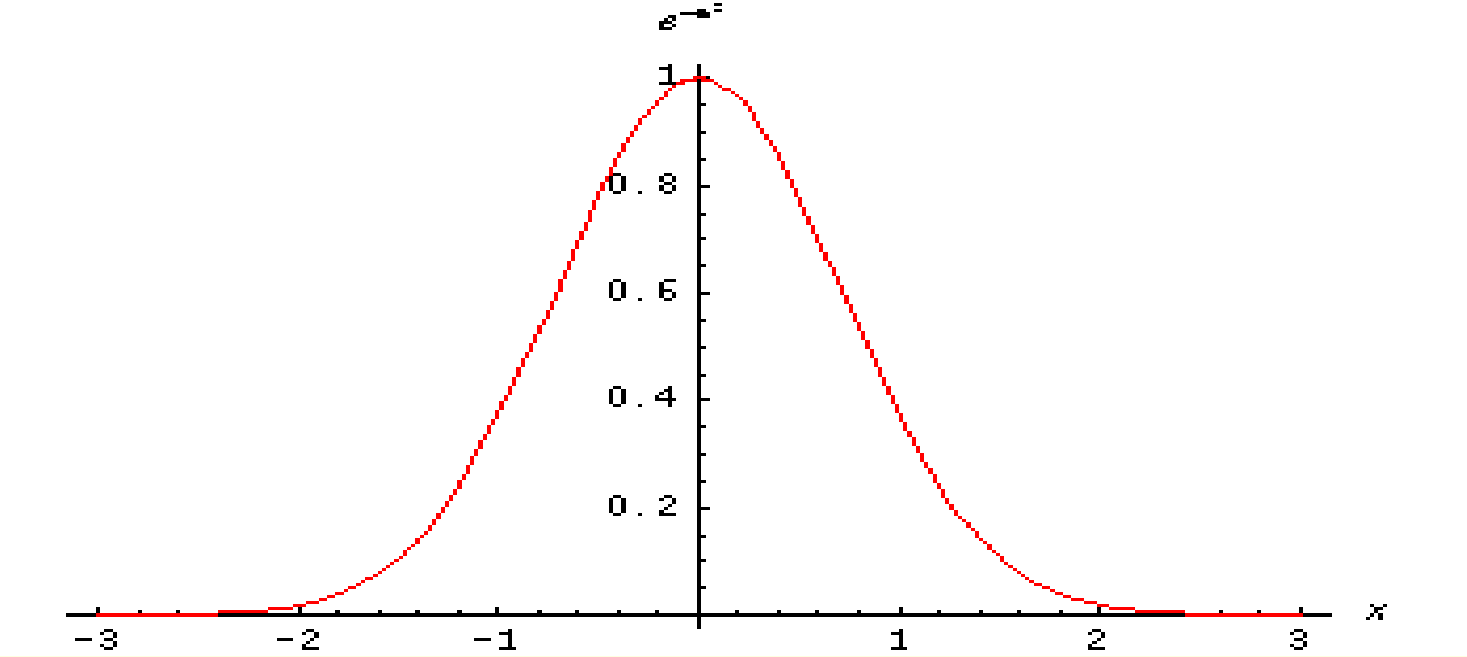


Sigmoid function



• The **Gaussian function** is the probability function of the normal distribution. Sometimes also called the frequency curve.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2},$$

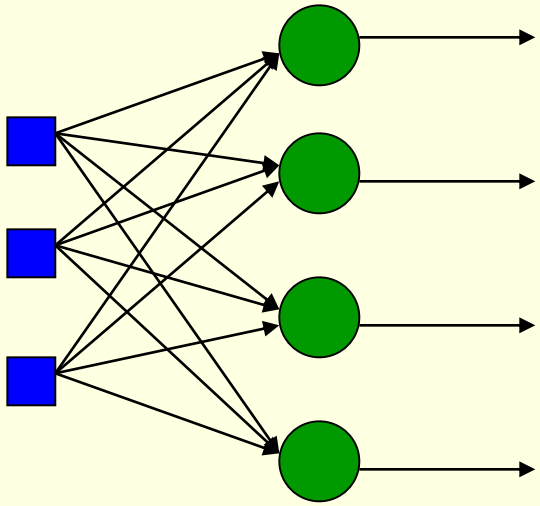


Network Architectures

- Three different classes of network architectures
  - single-layer feed-forward
  - multi-layer feed-forward
  - recurrent
- The architecture of a neural network is linked with the learning algorithm used to train

Single Layer Feed-forward

Input layer of source nodes



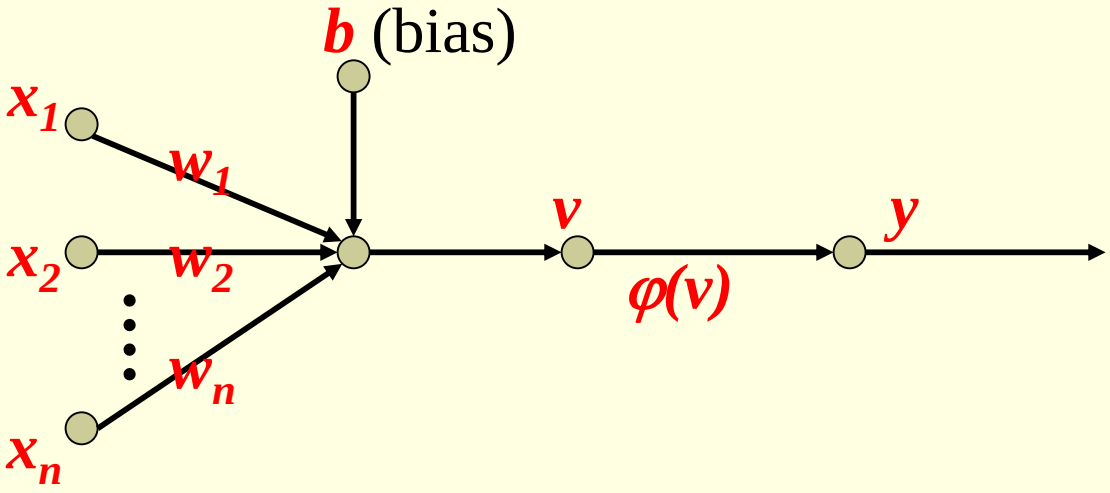
Output layer of neurons

Perceptron: Neuron Model

(Special form of single layer feed forward)

- The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories.
- A perceptron uses a **step function** that returns +1 if weighted sum of its input ≥ 0 and -1 otherwise

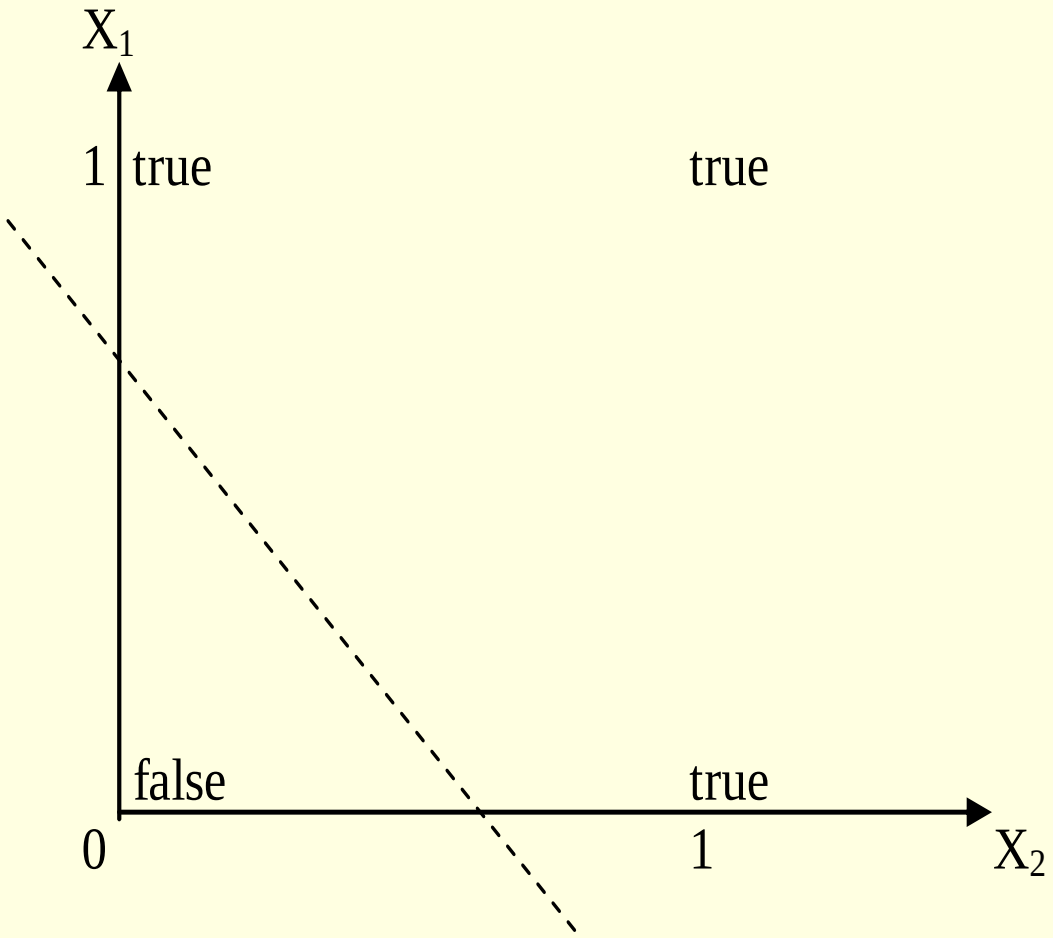
$$\varphi(v) = \begin{cases} +1 \text{ if } v \geq 0 \\ -1 \text{ if } v < 0 \end{cases}$$



Perceptron for Classification

- The perceptron is used for binary classification.
- First train a perceptron for a classification task.
  - Find suitable weights in such a way that the training examples are correctly classified.
  - Geometrically try to find a hyper-plane that separates the examples of the two classes.
- The perceptron can only model linearly separable classes.
- When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.
- Given training examples of classes C_1, C_2 train the perceptron in such a way that :
  - *If the output of the perceptron is +1 then the input is assigned to class C_1*
  - *If the output is -1 then the input is assigned to C_2*

Boolean function OR – Linearly separable



Learning Process for Perceptron

- Initially assign random weights to inputs between -0.5 and +0.5
- Training data is presented to perceptron and its output is observed.
- If output is incorrect, the weights are adjusted accordingly using following formula.

$$wi \leftarrow wi + (a^* xi^* e),$$ where 'e' is error produced and 'a' ($-1 < a < 1$) is learning rate

  - 'a' is defined as 0 if output is correct, it is +ve, if output is too low and –ve, if output is too high.
  - Once the modification to weights has taken place, the next piece of training data is used in the same way.
  - Once all the training data have been applied, the process starts again until all the weights are correct and all errors are zero.
  - Each iteration of this process is known as an epoch.

Example: Perceptron to learn OR function

- Initially consider w1 = -0.2 and w2 = 0.4
- Training data say, x1 = 0 and x2 = 0, output is 0.
- Compute y = Step(w1*x1 + w2*x2) = 0. Output is correct so weights are not changed.
- For training data x1=0 and x2 = 1, output is 1
- Compute y = Step(w1*x1 + w2*x2) = 0.4 = 1. Output is correct so weights are not changed.
- Next training data x1=1 and x2 = 0 and output is 1
- Compute y = Step(w1*x1 + w2*x2) = - 0.2 = 0. Output is incorrect, hence weights are to be changed.
- Assume a = 0.2 and error e=1

  wi = wi + (a * xi * e) gives w1 = 0 and w2 =0.4
- With these weights, test the remaining test data.
- Repeat the process till we get stable result.

Perceptron: Limitations

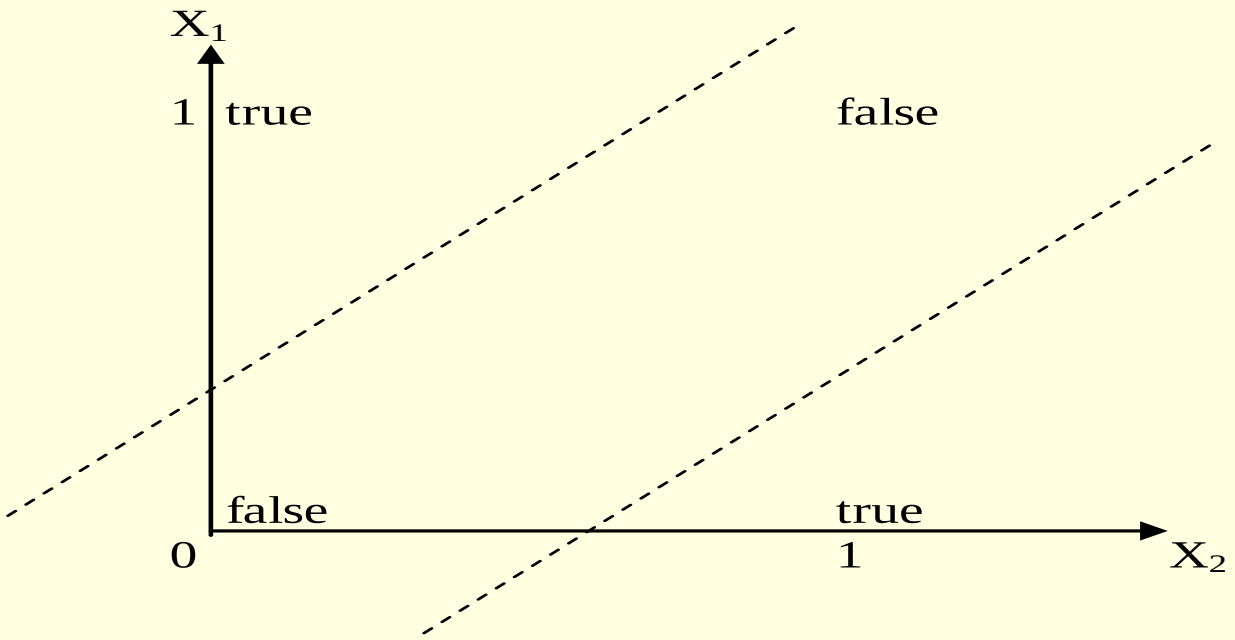
- The perceptron can only model linearly separable functions,
  - those functions which can be drawn in 2-dim graph and single straight line separates values in two part.
- Boolean functions given below are linearly separable:
  - AND
  - OR
  - COMPLEMENT
- It cannot model XOR function as it is non linearly separable.
  - When the two classes are not linearly separable, it may be desirable to obtain a linear separator that minimizes the mean squared error.

XOR – Non linearly separable function

- A typical example of non-linearly separable function is the XOR that computes the logical **exclusive or.**.
- This function takes two input arguments with values in {0,1} and returns one output in {0,1},
- Here 0 and 1 are encoding of the truth values **false** and **true**,
- The output is **true** if and only if the two inputs have different truth values.
- XOR is non linearly separable function which can not be modeled by perceptron.
- For such functions we have to use multi layer feed-forward network.

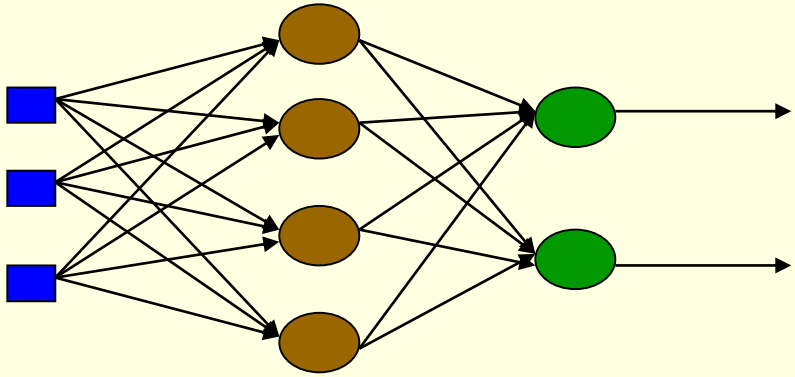
Input		Output
X_1	X_2	X_1 XOR X_2
0	0	0
0	1	1
1	0	1
1	1	0

These two classes (true and false) cannot be separated using a line. Hence XOR is non linearly separable.



Multi layer feed-forward NN (FFNN)

- FFNN is a more general network architecture, where there are hidden layers between input and output layers.
- Hidden nodes do not directly receive inputs nor send outputs to the external environment.
- FFNNs overcome the limitation of single-layer NN.
- They can handle non-linearly separable learning tasks.



Input layer

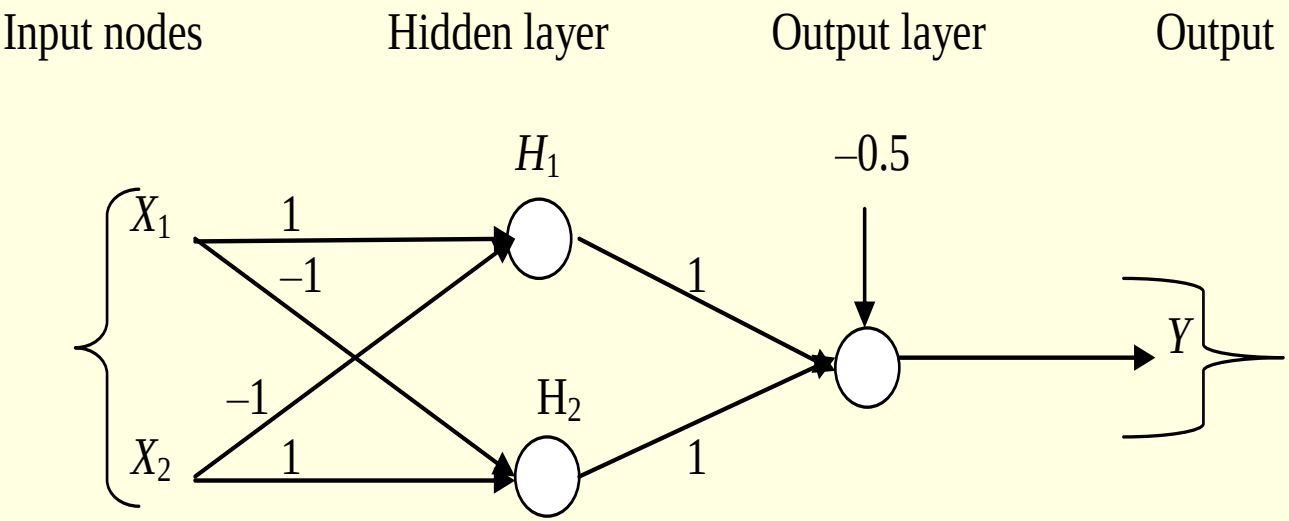
Output layer

Hidden Layer

3-4-2 Network

FFNN for XOR

- The ANN for XOR has two hidden nodes that realizes this non-linear separation and uses the sign (step) activation function.
- Arrows from input nodes to two hidden nodes indicate the directions of the weight vectors (1,-1) and (-1,1).
- The output node is used to combine the outputs of the two hidden nodes.



Inputs		Output of Hidden Nodes		Output	X_1 XOR X_2
X_1	X_2	H_1	H_2	Node	
0	0	0	0	–0.5 → 0	0
0	1	–1 → 0	1	0.5 → 1	1
1	0	1	–1 → 0	0.5 → 1	1
1	1	0	0	–0.5 → 0	0

Since we are representing two states by 0 (false) and 1 (true), we will map negative outputs (–1, –0.5) of hidden and output layers to 0 and positive output (0.5) to 1.

FFNN NEURON MODEL

- The classical learning algorithm of FFNN is based on the gradient descent method.
- For this reason the activation function used in FFNN are continuous functions of the weights, differentiable everywhere.
- The activation function for node i may be defined as a simple form of the **sigmoid function** in the following manner:

$$\varphi(Vi) = \frac{1}{1 + e^{(-A*Vi)}}$$

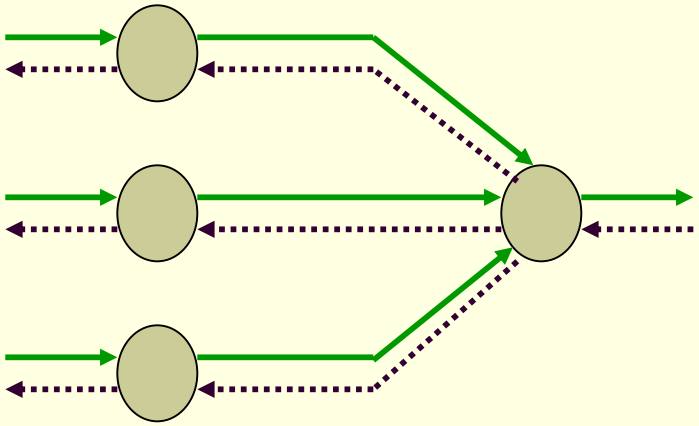
where $A > 0$, $V_i = \sum W_{ij} * Y_j$, such that W_{ij} is a weight of the link from node i to node j and Y_j is the output of node j.

Training Algorithm: Backpropagation

- The Backpropagation algorithm learns in the same way as single perceptron.
- It searches for weight values that minimize the total error of the network over the set of training examples (training set).
- Backpropagation consists of the repeated application of the following two passes:
  - **Forward pass**: In this step, the network is activated on one example and the error of (each neuron of) the output layer is computed.
  - **Backward pass**: in this step the network error is used for updating the weights. The error is propagated backwards from the output layer through the network layer by layer. This is done by recursively computing the local gradient of each neuron.

Backpropagation

- Back-propagation training algorithm



Network activation
Forward Step

Error propagation
Backward Step

- Backpropagation adjusts the weights of the NN in order to minimize the network total mean squared error.

Contd..

- Consider a network of three layers.
- Let us use i to represent nodes in input layer, j to represent nodes in hidden layer and k represent nodes in output layer.
- w_{ij} refers to weight of connection between a node in input layer and node in hidden layer.
- The following equation is used to derive the output value Yj of node j

$$\mathbf{Yj} = \frac{1}{1+e^{-X_j}}$$

where, $X_j = \sum x_i \,.\, w_{ij} - \theta_j$, $1 \leq i \leq n$; n is the number of inputs to node j, and θ_j is threshold for node j

Total Mean Squared Error

- The error of output neuron *k* after the activation of the network on the *n-th* training example (x(n), d(n)) is:

$$e_k(n) = d_k(n) - y_k(n)$$

- The network error is the sum of the squared errors of the output neurons:

$$\mathbf{E(n)} = \sum \mathbf{e_k^2(n)}$$

- The total mean squared error is the average of the network errors of the training examples.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^{N} E(n)$$

Weight Update Rule

- The Backprop weight update rule is based on the gradient descent method:
  - It takes a step in the direction yielding the maximum decrease of the network error E.
  - This direction is the opposite of the gradient of E.
- Iteration of the Backprop algorithm is usually terminated when the sum of squares of errors of the output values for all training data in an epoch is less than some threshold such as 0.01

$$w_{ij} = w_{ij} + \Delta w_{ij} \qquad \Delta \mathrm{w}_{\mathrm{ij}} = \text{-}\eta \frac{\partial E}{\partial \mathrm{w}_{\mathrm{ij}}}$$

Backprop learning algorithm (incremental-mode)

n=1;

initialize **weights** randomly;

while (stopping criterion not satisfied or n <max_iterations)

for each example (**x**,*d*)

- run the network with input x and compute the output y

- update the weights in backward order starting from those of the output layer:

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

with Δw_{ji} computed using the (generalized) Delta rule

end-for

n = n+1;

end-while;

Stopping criterions

- Total mean squared error change:
  - Back-prop is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small (in the range [0.1, 0.01]).
- Generalization based criterion:
  - After each epoch, the NN is tested for generalization.
  - If the generalization performance is adequate then stop.
  - If this stopping criterion is used then the part of the training set used for testing the network generalization will not used for updating the weights.

NN DESIGN ISSUES

- Data representation
- Network Topology
- Network Parameters
- Training
- Validation

Data Representation

- Data representation depends on the problem.
- In general ANNs work on continuous (real valued) attributes. Therefore symbolic attributes are encoded into continuous ones.
- Attributes of different types may have different ranges of values which affect the training process.
- Normalization may be used, like the following one which scales each attribute to assume values between 0 and 1.

$$x_i = \frac{x_i - \min_i}{\max_i - \min_i}$$

for each value x_i of i^{th} attribute, min_i and max_i are the minimum and maximum value of that attribute over the training set.

Network Topology

- The number of layers and neurons depend on the specific task.
- In practice this issue is solved by trial and error.
- Two types of adaptive algorithms can be used:
  - start from a large network and successively remove some neurons and links until network performance degrades.
  - begin with a small network and introduce new neurons until performance is satisfactory.

Network parameters

- How are the weights initialized?
- How is the learning rate chosen?
- How many hidden layers and how many neurons?
- How many examples in the training set?

Initialization of weights

- In general, initial weights are randomly chosen, with typical values between -1.0 and 1.0 or -0.5 and 0.5.
- If some inputs are much larger than others, random initialization may bias the network to give much more importance to larger inputs.
- In such a case, weights can be initialized as follows:

$$w_{ij} = \pm \frac{1}{2N} \sum_{i=1,\ldots,N} \frac{1}{|x_i|}$$

For weights from the input to the first layer

$$w_{jk} = \pm \frac{1}{2N} \sum_{i=1,\ldots,N} \frac{1}{\varphi(\sum w_{ij} x_i)}$$

For weights from the first to the second layer

Choice of learning rate

- The right value of η depends on the application.
- Values between 0.1 and 0.9 have been used in many applications.
- Other heuristics is that adapt η during the training as described in previous slides.

Training

- Rule of thumb:
  - the number of training examples should be at least five to ten times the number of weights of the network.
- Other rule:

$$N > \frac{|W|}{(1-a)}$$

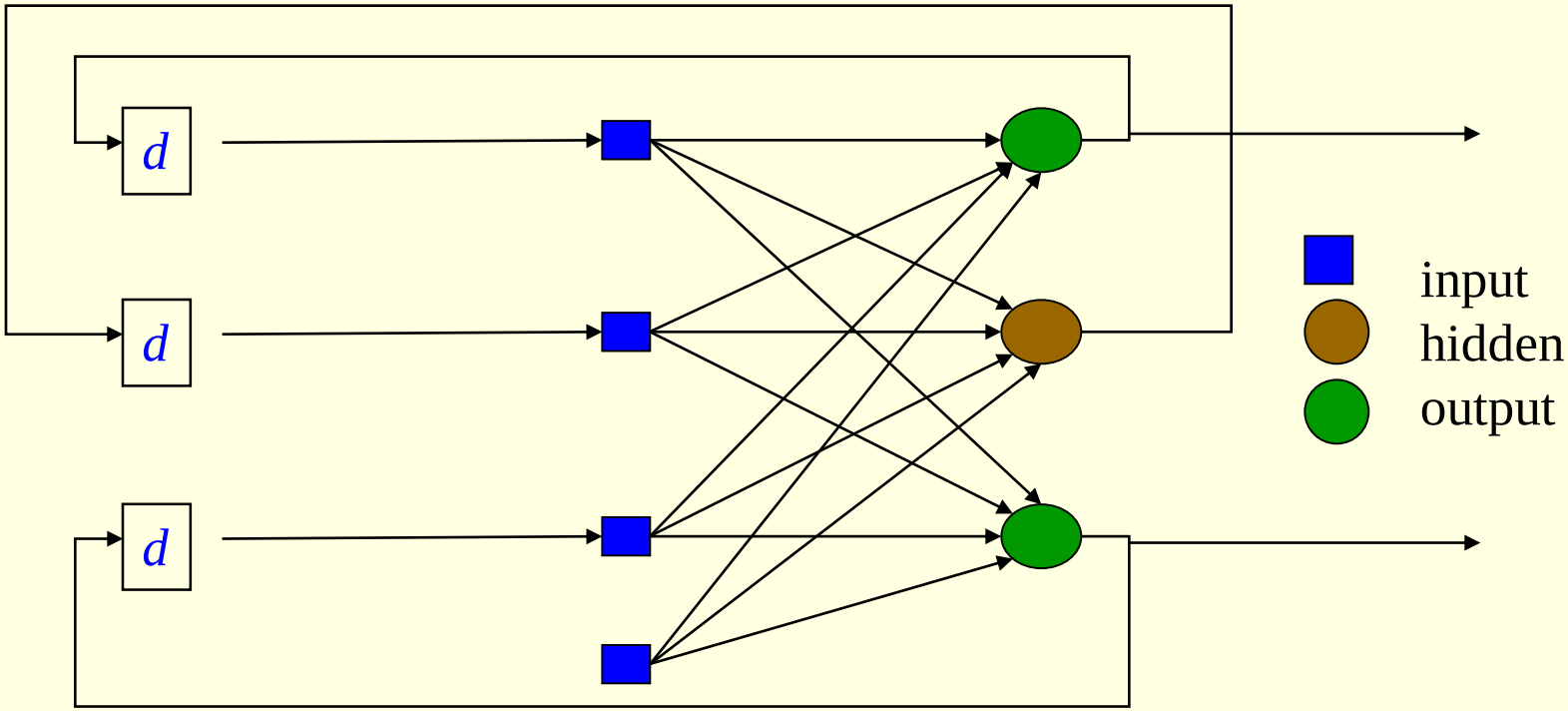
|W|= number of weights
a=expected accuracy on test set

Recurrent Network

- **FFNN** is acyclic where data passes from input to the output nodes and not vice versa.
  - Once the FFNN is trained, its state is fixed and does not alter as new data is presented to it. It does not have memory.
- **Recurrent network** can have connections that go backward from output to input nodes and models dynamic systems.
  - In this way, a recurrent network's internal state can be altered as sets of input data are presented. It can be said to have memory.
  - It is useful in solving problems where the solution depends not just on the current inputs but on all previous inputs.
- Applications
  - predict stock market price,
  - weather forecast

Recurrent Network Architecture

- Recurrent Network with ***hidden neuron***: unit delay operator *d* is used to model a dynamic system



Learning and Training

- During learning phase,
  - a recurrent network feeds its inputs through the network, including feeding data back from outputs to inputs
  - process is repeated until the values of the outputs do not change.
- This state is called equilibrium or stability
- Recurrent networks can be trained by using back-propagation algorithm.
- In this method, at each step, the activation of the output is compared with the desired activation and errors are propagated backward through the network.
- Once this training process is completed, the network becomes capable of performing a sequence of actions.

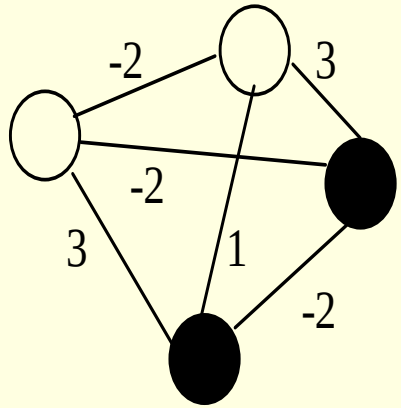
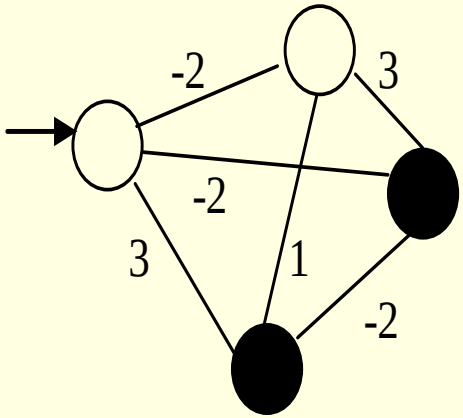
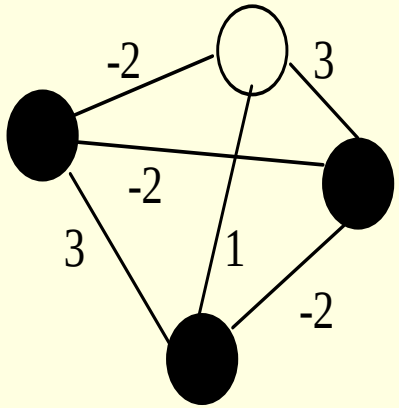
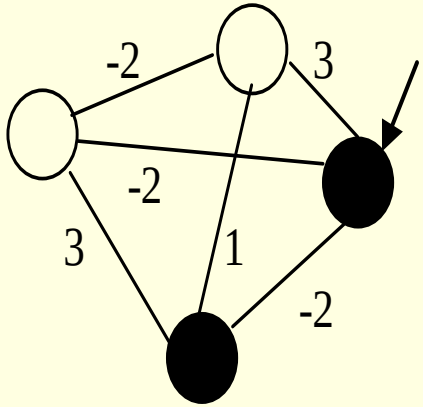
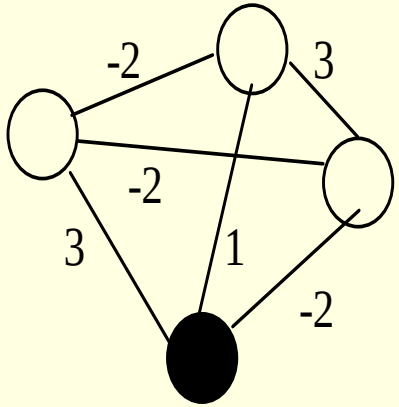
Hopfield Network

- A **Hopfield network** is a kind of recurrent network as output values are fed back to input in an undirected way.
  - It consists of a set of N connected neurons with weights which are symmetric and no unit is connected to itself.
  - There are no special input and output neurons.
  - The activation of a neuron is binary value decided by the sign of the weighted sum of the connections to it.
  - A threshold value for each neuron determines if it is a firing neuron.
  - A firing neuron is one that activates all neurons that are connected to it with a positive weight.
  - The input is simultaneously applied to all neurons, which then output to each other.
  - This process continues until a stable state is reached.

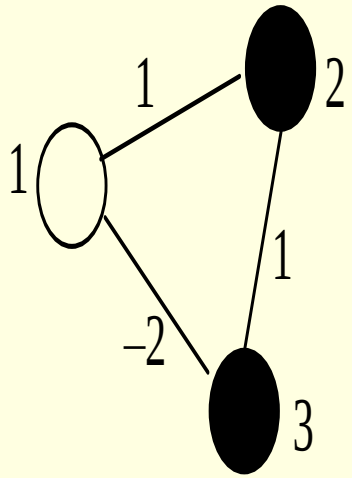
Activation Algorithm

Active unit represented by 1 and inactive by 0.

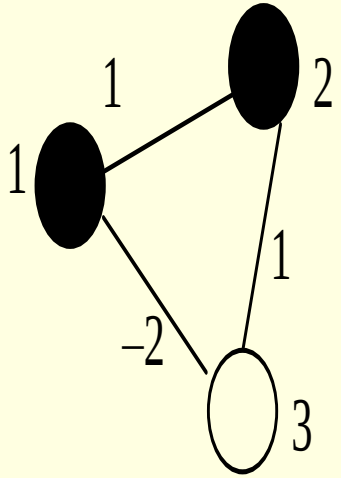
- *Repeat*
  - Choose any unit randomly. The chosen unit may be active or inactive.
  - For the chosen unit, compute the sum of the weights on the connection to the active neighbours only, if any.
    - If sum > 0 (threshold is assumed to be 0), then the chosen unit becomes active, otherwise it becomes inactive.
  - If chosen unit has no active neighbours then ignore it, and status remains same.
- *Until* the network reaches to a stable state

Current State	→ Selected Unit from current state	Corresponding New State
		 Sum = 3 – 2 = 1 > 0; activated
Here, the sum of weights of active neighbours of a selected unit is calculated.		 Sum = –2 < 0; deactivated

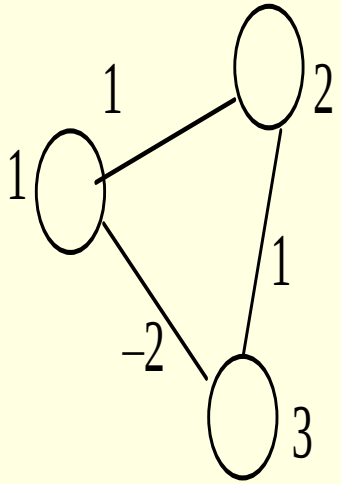
Stable Networks



X = [0 1 1]



X =[1 1 0]



X =[0 0 0]

Weight Computation Method

- Weights are determined using training examples.

$$W = \sum X_i \,.\, (X_i)^{\mathrm{T}} - M.I, \quad \text{for } 1 \le i \le M$$

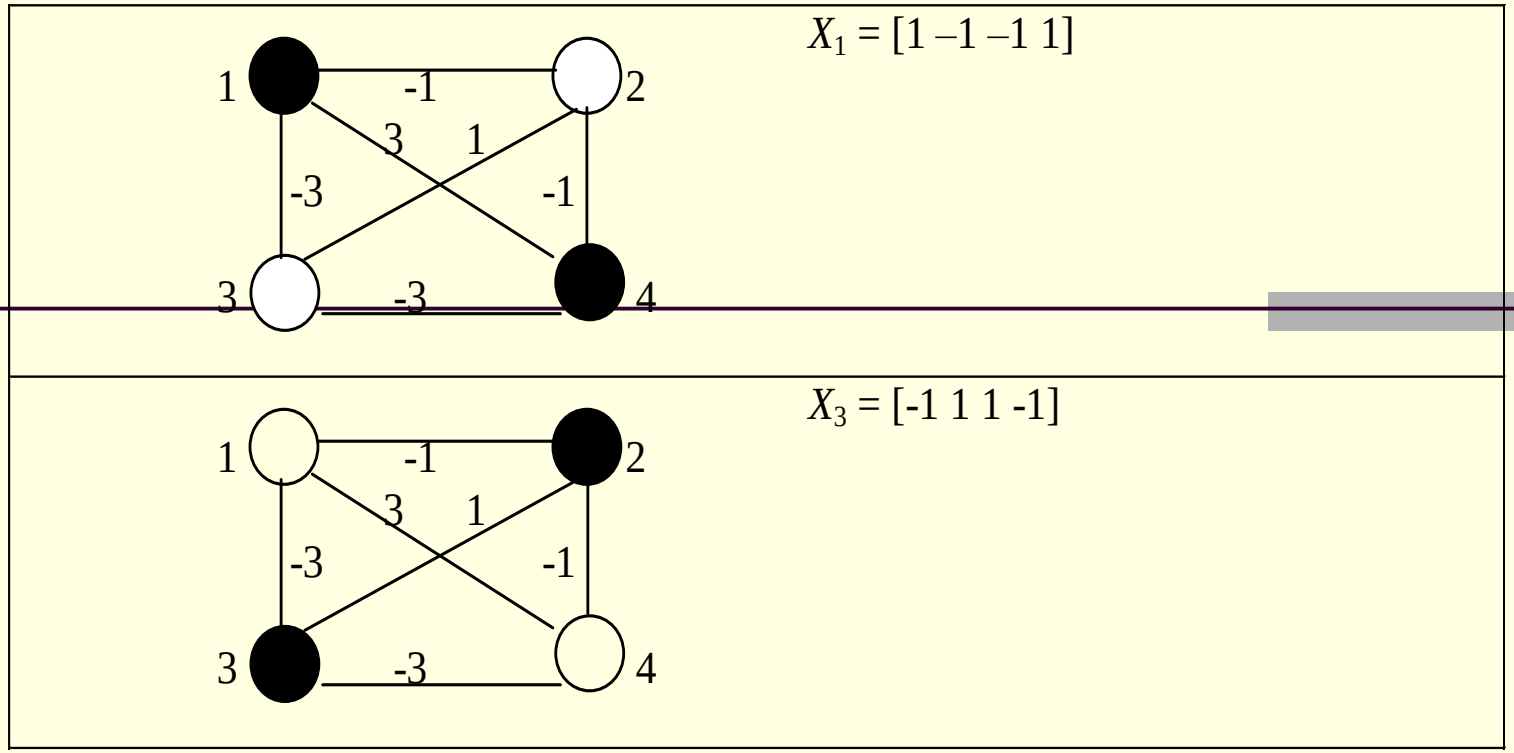
- Here
  - *W* is weight matrix
  - *Xi* is an input example represented by a vector of *N* values from the set {–1, 1}.
    - Here, *N* is the number of units in the network; 1 and -1 represent active and inactive units respectively.
  - $(Xi)^{\mathrm{T}}$ is the transpose of the input *Xi* ,
  - *M* denotes the number of training input vectors,
  - *I* is an *N* × *N* identity matrix.

Example

- Let us now consider a Hopfield network with four units and three training input vectors that are to be learned by the network.
- Consider three input examples, namely, X1, X2, and X3 defined as follows:

$$X_1 = \begin{pmatrix} 1 \\ -1 \\ -1 \\ 1 \end{pmatrix} \qquad X_2 = \begin{pmatrix} 1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \qquad X_3 = \begin{pmatrix} -1 \\ 1 \\ 1 \\ -1 \end{pmatrix}$$

$$W = X_1 \,.\, (X_1)^T + X_2 \,.\, (X_2)^T + X_3 \,.\, (X_3)^T - 3.I$$



Stable positions of the network

$$W = \begin{pmatrix} 3 & -1 & -3 & 3 \\ -1 & 3 & 1 & -1 \\ -3 & 1 & 3 & -3 \\ 3 & -1 & -3 & 3 \end{pmatrix}. - \begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = \begin{pmatrix} 0 & -1 & -3 & 3 \\ -1 & 0 & 1 & -1 \\ -3 & 1 & 0 & -3 \\ 3 & -1 & -3 & 0 \end{pmatrix}$$

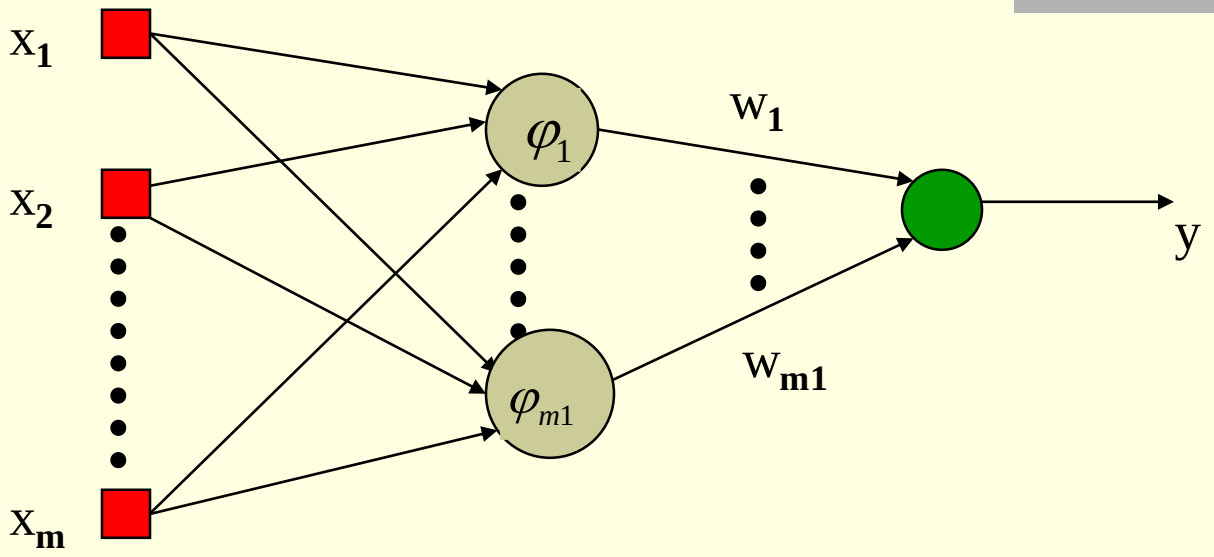
Contd..

- The networks generated using these weights and input vectors are stable, except X2.
- X2 stabilizes to X1 (which is at hamming distance 1).
- Finally, with the obtained weights and stable states (X1 and X3), we can stabilize any new (partial) pattern to one of those

Radial-Basis Function Networks

- A function is said to be a ***radial basis function*** (RBF) if its output depends on the distance of the input from a given stored vector.
  - The RBF neural network has an input layer, a hidden layer and an output layer.
  - In such RBF networks, the hidden layer uses neurons with RBFs as activation functions.
  - The outputs of all these hidden neurons are combined linearly at the output node.
- These networks have a wide variety of applications such as
  - function approximation,
  - time series prediction,
  - control and regression,
  - pattern classification tasks for performing complex (non-linear).

RBF Architecture



- **One hidden layer with RBF activation functions**

$$\varphi_1 \cdots \varphi_{m1}$$

- **Output layer with linear activation function.**

$$y = w_1\varphi_1(\| x - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x - t_{m1} \|)$$

$$\| x - t \| \text{ distance of } x = (x_1, \ldots, x_m) \text{ from center } t$$

Cont...

- Here we require weights, w_i from the hidden layer to the output layer only.
- The weights w_i can be determined with the help of any of the standard iterative methods described earlier for neural networks.
- However, since the approximating function given below is linear w. r. t. w_i, it can be directly calculated using the matrix methods of linear least squares without having to explicitly determine w_i iteratively.

$$Y = f(X) = \sum_{i=1}^{N} w_i \varphi(\|X_i - t_i\|)$$

- It should be noted that the approximate function *f(X)* is differentiable with respect to w_i.

Comparison

RBF NN	FF NN
Non-linear layered feed-forward networks.	*Non-linear layered feed-forward* networks
Hidden layer of RBF is *non-linear*, the output layer of RBF is *linear.*	Hidden and output layers of FFNN are usually *non-linear*.
One *single* hidden layer	May have *more* hidden layers.
Neuron model of the hidden neurons is *different* from the one of the output nodes.	Hidden and output neurons share a *common neuron model*.
Activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit.	Activation function of each hidden neuron in a FFNN computes the *inner product* of input vector and the synaptic weight vector of that neuron

Machine Learning

Machine Learning

- Simon defined the concept of machine learning as adaptive changes in a system that enable a system to do the same task drawn from the same population with greater efficiency when ever the task are repeated
- Ryszard Michalski learning involves constructing and modifying representations of experience

Machine Learning

- Marvin Minsky defined learning as making useful changes to the mind
- Machine learning is a subfield of AI concerned with design and development of algorithms and techniques that allow computers to learn
- There are two types of learning methodologies
  - Inductive methodology: learning from rules and patterns extracted from massive data sets
  - Deductive methodology: deducting new knowledge from existing knowledge

Machine Learning

- Computation analysis of machine learning algorithm and their performance forms a branch of theoretical CS known as *computational learning theory*
- Machine learning can be categorized as
  - Supervised learning : generates a function that maps input to the desired output
  - Unsupervised learning: no labeled examples are available

Machine Learning

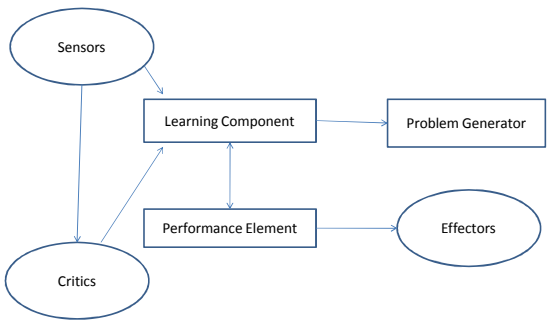


Figure: Components of a Learning System

Machine Learning

- **Learning Components**: Making changes or improvement to the system based on performance
- **Performance element**: Choose the actions (best) that need to be taken
- **Critic**: To inform learning component regarding its performance with respect to a fixed standard

Machine Learning

- **Problem Generator**: it suggests problems or actions that would lead to generation of new examples or experiences, that aid in training
- **Sensors and Effectors**: Sensors are inputs and Effectors (action on the environment) are output

Rote Learning

- It is the process of memorization
- It involves one-to-one mapping from inputs to stored representation and is also known as *learning by memorization*; it uses association-based storage and retrieval
- Caching is an example of rote learning as large pieces of data or computation are stored and recalled when necessary

Rote Learning

- Rote learning help highlight some issues like
  - Organization: Knowledge should be organized in a manner that is easy to retrieve than re-compute
  - Generalization: Since stored objects are many, we need generalization to manage the problem
  - Stability of environment: Rote learning is not effective when environment changes rapidly
- When to store and when to re-compute has to be decided based on cost-benefit analysis involving amount of storage, cost of computation, likelihood of recall

Learning by taking Advice

First approach

- Taking high-level advice and then converting them into rules (McCarthy, 1959)
- All aspects of advice taking are automated
- The following steps are involved
  - Request
  - Interpret
  - Operationalize
  - Integrate
  - Evaluate

Learning by taking Advice

Second approach

- Developing sophisticated modules: the modules enable the expert to translate expertise into detailed rules

Learning by Parameter Adjustment

- This is a static evaluation function. A slight modification in evaluation function may lead to learning. This is called learning by parameter adjustment
- The function may be represented as

$$\sum W_i * T_i$$

Where W_i are weight of terms and T_i are the terms representing values of features

Learning by Parameter Adjustment

- Initially start with some estimate of correct weights
- Then modify the weights on basis of accumulated experiences
- Increase weights of that appear to be good predictors and decrease the weights for bad predictors

Learning by Analogy

- Analogical reasoning involves recognizing and applying solutions from known problems to a new problem
- They are also called case-based reasoning
- Learning by analogy involves identifying similarities in information stored and transfer knowledge from previous information an apply it to improve solution of the task at hand

Learning by Analogy

- They are two types
  - Transformational analogy: copy solution from a similar old situation to a new situation
  - Derivational analogy: examine the history of problem solution and the steps involved before applying it as a solution to a new situation

Supervised and Unsupervised Learning

- Here the basic aim is to learn an unknown function ***f(x) = y***, where ***x*** is the input example and ***y*** is the desired output
- The model in supervised learning describes the effect of set of observations called input on another set of observations called output
- Here both sets are given and the purpose is to find the function ***f*** that transforms them from input to output

Supervised and Unsupervised Learning

- In unsupervised learning, all input observations are given and no output observations are available
- Learning models with deep hierarchies have are more suitable for unsupervised learning as learning time increases linearly with increasing levels of the learning model

Supervised-Neural Networks

- This model is kind of similar to human brain
- It consists of several simple units called neurons
- These units are connected (work in parallel) and weights are assigned to these connections
- These weights can be modified by some learning method

Supervised-Neural Networks

- Feed-forward network is a kind of supervised learning where training examples are given by an external teacher to train the network
- There are two kinds of inputs called input vectors and output vectors
- Input vectors are associated with input neurons (input layer) and output vectors are associated with output neurons (output layer)

Neural Network

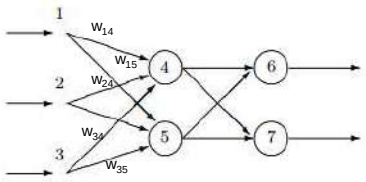


Figure: Nodes 1, 2, 3 are input layer, 4 and 5 are hidden layer and 6, 7 are output layer

Supervised-Neural Networks

- The input vector is provided at input layer to each neuron and desired responses for each neuron at the output layer
- In forward pass, the discrepancies in the desired versus actual responses for each neuron in the output layer is obtained
- These discrepancies can be reduced by adjusting the weights in the network using the learning rule

Supervised Concept Learning

- Here a set of ***(x,y)*** pairs are provided by the teacher where x denotes input and y denotes corresponding output
- Example of ***x*** could be a feature set to identify a concept like family car
- ***y*** could be positive or negative where + denotes that example car is family car and – denote that example car is not a family car

Supervised Concept Learning

- If example is instance of concept (family car), then it is positive example
- If example is not an instance of concept (family car) then it is negative example
- To generalize we have to learn a good estimation function ***f***, if we are given a training set $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\ldots(x_n,y_n)\}$ where x_i is input features and y_i is a label like + or –
- Then, an unknown example (instance) should be correctly classified as + or – by the function ***f***

Supervised Concept Learning

- The training set $\{(x_1, y_1), (x_2, y_2), (x_3, y_3)\ldots(x_n,y_n)\}$ where x_i is input features and y_i is a label like + or – for N examples is defined as ordered pair
  $X = \{(x^k, label^k)\}$, $1<= k <=N$
- For family car concept, if we have considered three features like price, seat and engine and label is 1 if it is family car, otherwise it is 0. Then, an ordered pair of an instance for first car is x^1=(price^1, seat^1, engine^1) and label^1 would be either 1 or 0 and is written as (x^1, label^1). There would be N such ordered pair for N cars examples

Supervised Concept Learning

- In the example price, seat and engine may have their ranges written as

p1<= price <=p2, s1<= seat <=s2 and e1<=engine <= e2

- You get a region C of positive examples data points in space for each tuple ((p1, p2),(s1,s2),(e1,e2))
- The learning algorithm must find an hypothesis ***h*** that approximates C as close as possible
- The hypothesis ***h(x)*** where ***x*** is an instance will have values 1 if example is positive or 0 if example is negative i.e., ***h(x) = 0 or 1***

Supervised Concept Learning

- The error in hypothesis h given set of training example X may be defined as

$$Er(h \mid X) = \sum_{k=1}^{N} 1, \text{ such that } h(x^k) \neq \text{label}^k$$

- This is sum of all the cases for which it could not classify it with the correct label

Unsupervised Learning

- For unsupervised learning, a typical task of classification or regression, uses examples prepared by humans
- Only given input ***s***, the ultimate goal is to estimate the function ***f***
- Examples include clustering and discovery
- Commonly used unsupervised learning algorithms among neural network models include self-organizing map and Adaptive Resonance Theory (ART)

Reinforcement Learning

- In reinforcement learning the decision-making agent receives rewards (feedback/utility) for actions at the end of sequence of steps - function
- It cannot compare the actual result with any desired results
- It only determines if the function is successful based on positive feedback for its actions – maximize utility
- Some applications are checker playing program , cart-pole balancing problem and simulation of triple-inverted-pendulum problem using neural network

Reinforcement Learning

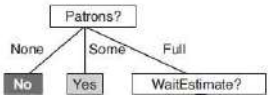
- Learning is achieved by using
  - Utility function: This is feedback (maximize utility value) based on the desirable states after performing some action – requires agent to remember the model of the environment
  - Action-value function: The expected utility is maximized based on the actions taken at a given state – This is also called Q-Learning; agent need not remember the model of the environment
- Learning is difficult as there is no teacher but only a critic. Critic only passes judgment on the result and does not provide examples or guide the actions

Learning Decision Trees (Russell an Norvig)

- Decision tree takes as input a set of attribute values (vector) and returns a single decision value
- Each node in decision tree corresponds to an attribute value. Each node performs a test to determine the value of the attribute
- Each branch is labeled as per the value of that the attribute can take. The values may be discrete or continuous

Example

1. *Alternate*: whether there is a suitable alternative restaurant nearby.
2. *Bar*: whether the restaurant has a comfortable bar area to wait in.
3. *Fri/Sat*: true on Fridays and Saturdays.
4. *Hungry*: whether we are hungry.
5. *Patrons*: how many people are in the restaurant (values are *None*, *Some*, and *Full*).
6. *Price*: the restaurant's price range ($, $$, $$$).
7. *Raining*: whether it is raining outside.
8. *Reservation*: whether we made a reservation.
9. *Type*: the kind of restaurant (French, Italian, Thai, or burger).
10. *WaitEstimate*: the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).



Here Patrons? Is an internal node that perform the test and result can be None, Some or Full

Learning Decision Trees (Russell an Norvig)

- The *goal predicate* is the decision that tree is trying to determine (boolean decision)
- *Boolean decision tree* is an assertion that the goal attribute is true provided the input attributes satisfy a path in the tree leading to a leaf node that is true
- If we consider each path leading to a true state of leaf node (goal) as P_i where i takes values 1, 2, 3 … then

Learning Decision Trees (Russell an Norvig)

- If we consider each path leading to a true state of leaf node (goal) as $Path_i$ where *i* takes values 1, 2, 3 … then goal

  $Goal \Leftrightarrow (Path_1 \vee Path_2 \vee Path_3 \vee \ldots)$
- In the example the goal is *WillWait* that can either be *Yes* or *No*

Induction in Decision Trees

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
x_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
x_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
x_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
x_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
x_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
x_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
x_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
x_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
x_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
x_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
x_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
x_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Table above provides 12 Training examples

- (x_1, x_3, x_4, x_6 …) are positive examples (*WillWait = Yes*)
- (x_2, x_5, x_7, x_9, …) are negative examples (*WillWait = No*)

Induction in Decision Trees

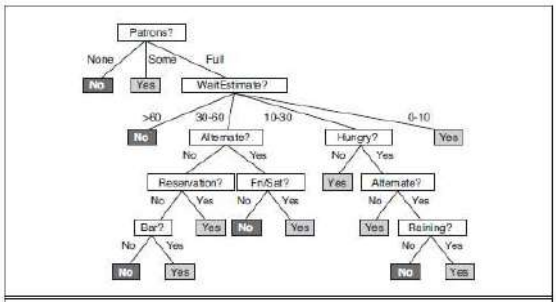


Figure: Decision tree for waiting for a table in a restaurant

Learning Decision Trees (Russell an Norvig)

- A Boolean decision tree consists of an (**x**, *y*) where ***x*** is the vector of list of attributes values for which *y* has a single boolean value
- The decision-tree-learning algorithm using greedy divide-and-conquer strategy i.e., test the most important attribute first

Inducing Decision Trees

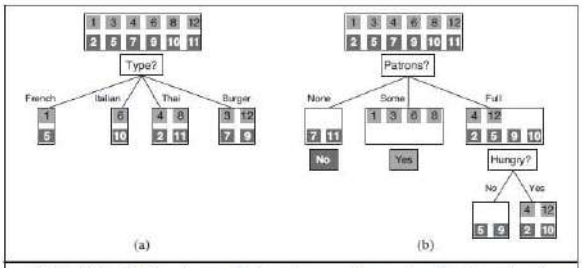
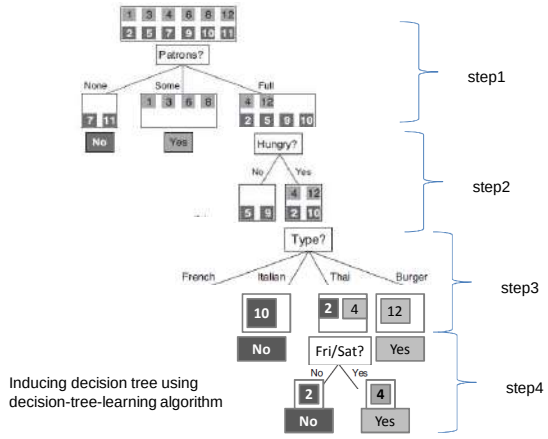
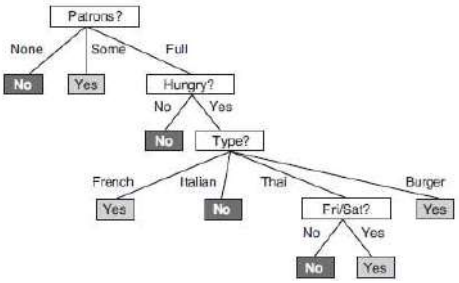


Figure 18.4 Splitting the examples by testing on attributes. At each node we show the positive (light boxes) and negative (dark boxes) examples remaining. (a) Splitting on *Type* brings us no nearer to distinguishing between positive and negative examples. (b) Splitting on *Patrons* does a good job of separating positive and negative examples. After splitting on *Patrons*, *Hungry* is a fairly good second test.

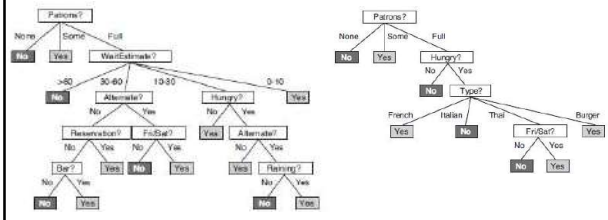


Inducing decision tree using decision-tree-learning algorithm



Final decision tree induced from the training example set

Learning Decision Trees (Russell an Norvig)



Comparison of decision trees before (left) and after (right) learning which is shallower and less nodes than the original tree for the training set provided

Learning Decision Trees (Russell an Norvig)

- The accuracy of the algorithm is measured by splitting the examples into test set and training set first
- Learn the hypothesis using the training set and run the result of the algorithm on the test set
- The resulting graph obtained are called *learning curves (happy graphs)*

Learning Decision Trees (Russell an Norvig)

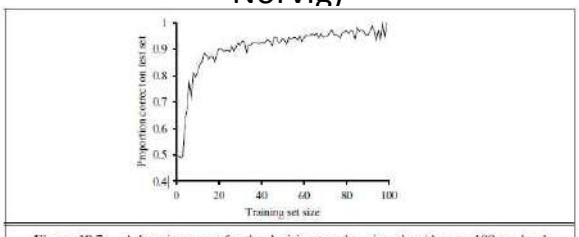


Figure 18.7 A learning curve for the decision tree learning algorithm on 100 randomly generated examples in the restaurant domain. Each data point is the average of 20 trials.

The learning curve shows that as the training set size grows, the accuracy is also increasing

Learning Decision Trees (Russell an Norvig)

- To determine the best variable to expand the information gain is calculated for each attribute
- *Information gain* the expected reduction in entropy for a particular variable
- *Entropy* is nothing but the measure of uncertainty for that attribute – acquisition of information decreases the entropy
- The attribute with the highest information gain is selected as the best attribute to expand

Learning Decision Trees (Russell an Norvig)

- The decision-tree-learning algorithm may sometimes produce trees that may have nodes that is testing irrelevant attributes
- This problem is called *overfitting*
- A technique called decision-tree pruning is carried out after a tree is generated by the algorithm by pruning irrelevant nodes – considering nodes with only leaf nodes
- We remove the nodes that are testing irrelevant attributes

Information Gain

- The probability of boolean random variable that is true with probability q is defined as

$$B(q) = -(q \log_2 q + (1-q)\log_2(1-q))$$

- The entropy of a goal attribute (*WillWait*) is defined as:

$$H(Goal) = B\left(\frac{p}{p+n}\right)$$

where p are positive and n are negative examples

Information Gain

- Expected remaining entropy after testing an attribute is where p_k are postive examples and n_k are negative examples and *d* are the discrete values of the attribute

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$$

- The information gain from testing the attribute A is defined as

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

Information Gain

- Expected remaining entropy after testing an attribute is where p_k are postive examples and n_k are negative examples and *d* are the discrete values of the attribute

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k+n_k}{p+n} B(\frac{p_k}{p_k+n_k})$$

- The information gain from testing the attribute A is defined as

$$Gain(A) = B(\frac{p}{p+n}) - Remainder(A)$$

Information Gain

- Thus the information gain for each attribute can be calculated as follows for the example

$$Gain(Patrons) = 1 - \left[\frac{2}{12}B(\frac{0}{2}) + \frac{4}{12}B(\frac{4}{4}) + \frac{6}{12}B(\frac{2}{6})\right] \approx 0.541 \text{ bits},$$
$$Gain(Type) = 1 - \left[\frac{2}{12}B(\frac{1}{2}) + \frac{2}{12}B(\frac{1}{2}) + \frac{4}{12}B(\frac{2}{4}) + \frac{4}{12}B(\frac{2}{4})\right] = 0 \text{ bits},$$

- Thus, we can see that Gain(Patrons) is more than that of Gain(Type), therefore Patrons is a better attribute than Type

Support Vector Machines (SVM)

- It is the most popular supervised learning framework introduced by Vladimir Vapnik and colleagues in 1995.
- SVM performs classification of data by constructing N-dimensional hyper-plane that optimally separates the data points into two categories (binary classifier)
- A SVM that construct a N-1 dimensional hyper-plane in N dimensional space and separate data points such that there is a maximum separation margin between the two data sets is called a *linear classifier*

SVM-Example

Suppose we have 50 photographs of elephants and 50 photos of tigers.

vs.

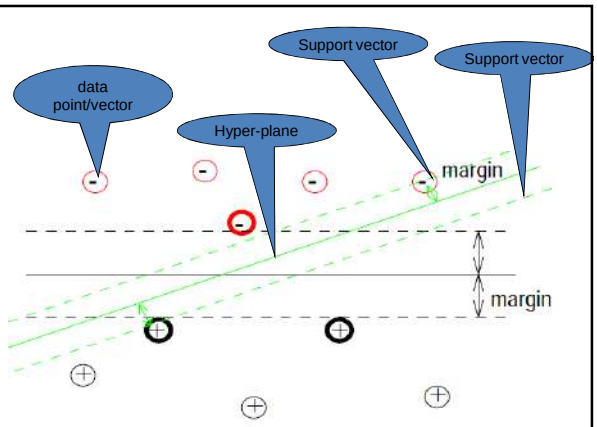
We digitize them into 100 x 100 pixel images, so we have $x \in R^n$ where $n = 10,000$.

Now, given a new (different) photograph we want to answer the question: **is it an elephant or a tiger?** [we assume it is one or the other.]

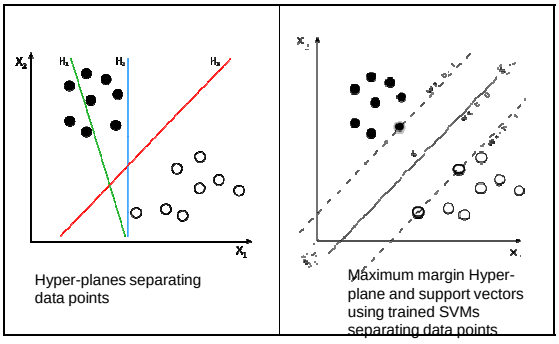
Support Vector Machines (SVM)

Terminology used to describe SVM

- Attribute: a predictor variable is called an attribute
- Feature: a transformed attribute that can be used to define a hyper-plane
- Feature selection: This is a process of choosing the most suitable representation
- Vector: It is set of features used to describe a predictor variable
- Hyper-plane: it separates data points into two categories
- Support vectors: vectors parallel to hyper-plane separating the data points
- Margin: distance between support vectors



SVM-Linear Classifier



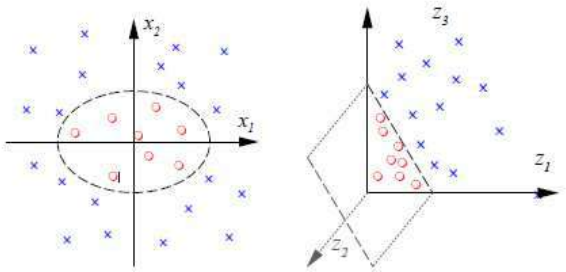
Hyper-planes separating data points

Maximum margin Hyper-plane and support vectors using trained SVMs separating data points

SVM

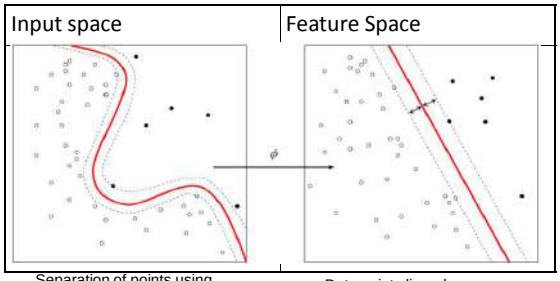
- The distance between the two dashed line is called the *margin*
- Data points in two-dimensional space can be linearly separable by one-dimensional line
- Data points in N-dimensional space can be linearly separable with the help of N-1 dimensional hyper-plane

SVM



Non-linear curve (left) now linearly separable (right) in 3-dimensional (N) feature space using two-dimensional (N-1) hyper-plane

SVM-Non-Linear Classifier



Separation of points using non-linear curve

Data points linearly separable

Figure: Transformation into higher-dimension space

SVM

- The hyper-plane for three –attribute situation for a linearly separable case is represented as

$$y= w_o+w_1x_1+w_2x_2+w_3x_3$$

Where y is the outcome and x_i denote attribute values. If y=+1 for class1 and y=-1 for class2 then solving the above equation give hyper-plane separating the points with maximum distance

SVM

- When a non-linear curve is used to separate points in the *input space*, then SVM uses *kernel function* that transforms the given data into higher dimension so that the data is now linearly separable in the *feature space*
- But transforming input space to feature space increases the number of features and if there are too many feature vector dimensions then it is difficult to generalize this model to other (unseen) data . This is called *over fitting*

SVM

- The maximum margin hyper-plane can be represented as follows in terms of support vectors

$$\sum \alpha_i y_i K(x(i) \bullet x)$$

Where $K(x(i) \bullet x)$ is called the kernel function

- Common examples of kernel functions are
  - Polynomial kernel: $K(x \bullet y) = (xy+1)^d$
  - Gaussian radial basis function: $K(x \bullet y) = \exp\left[-\frac{(x-y)^2}{2\sigma^2}\right]$

  Where *d* is degree of polynomial and σ is bandwidth of gaussian radial basis function

SVM Applications

- SVM has been used successfully in many real-world problems
  - text (and hypertext) categorization
  - image classification
  - bioinformatics (Protein classification, Cancer classification)
  - hand-written character recognition