CS 561 Artificial Intelligence Lecture #15-18

Learning from Examples

Rashmi Dutta Baruah

Department of Computer Science & Engineering

IIT Guwahati

Outline

- Learning
- Supervised Learning
- Naïve Bayes Example
- Artificial Neural Networks: An Introduction
  - Biological and artificial neuron
- ANN: activation functions, interconnections, learning rule
- Multilayer feedforward network and backpropagation learning

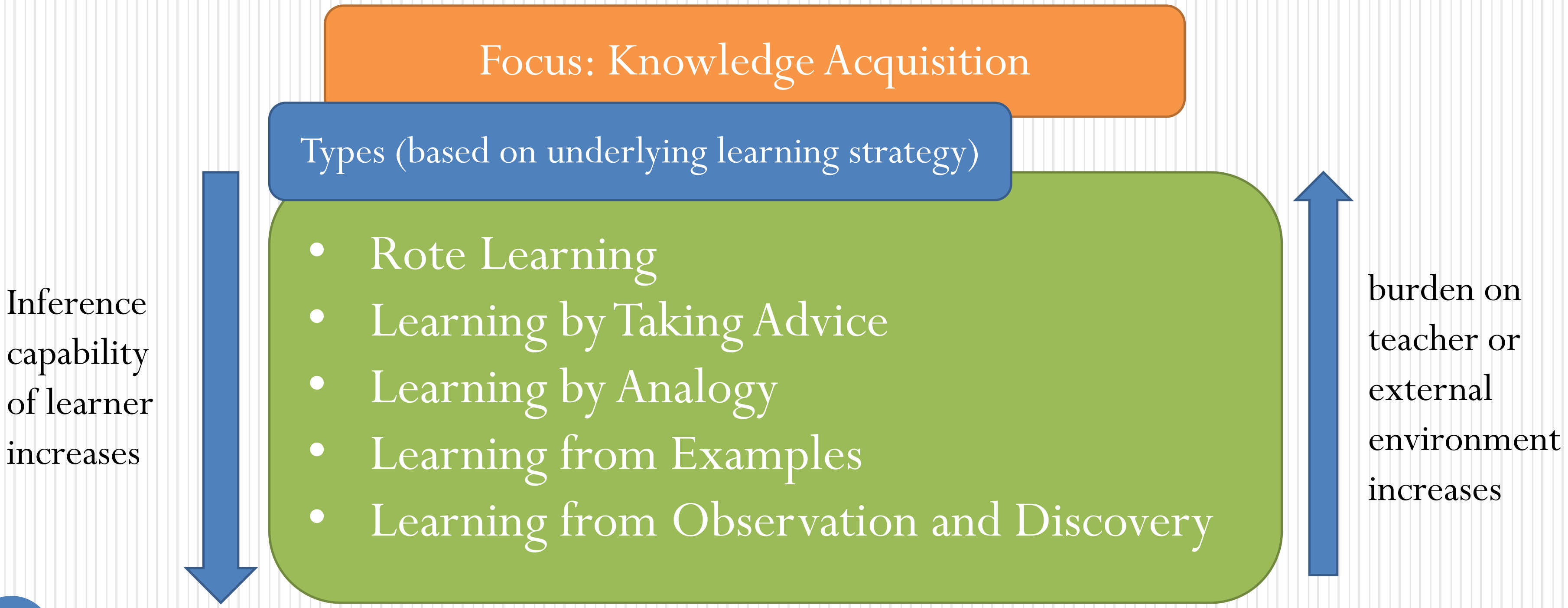
Learning

- What is Learning?
- the cognitive process of acquiring skill or knowledge
- it has several aspects[1]
  - acquisition of new declarative knowledge
  - development of motor or cognitive skills through instruction or practice
  - organization of new knowledge into general
  - effective representations
  - discovery of new facts and theories through observation and experimentation
- Machine learning: study and computer modelling of learning process in their multiple manifestations

Machine Learning: A Historical and Methodological Analysis, J.G. Carbonell, R.S. Michalski, T.M. Mitchell, AI Magazine, Vol. 4, No. 3 (1983)

Learning

- Human learning is a mixture of both knowledge acquisition and skill refinement.
- Machine learning focuses on the knowledge acquisition aspect.


Focus: Knowledge Acquisition
Types (based on underlying learning strategy)
Rote Learning
Learning by Taking Advice
Learning by Analogy
Learning from Examples
Learning from Observation and Discovery
Inference capability of learner increases
burden on teacher or external environment increases

Learning: Knowledge Acquisition

- Rote Learning
  - learning by being programmed, constructed, or modified by an external entity (explicit programming)
  - learning by memorization of facts and data, no inferences are drawn
- Learning from taking advice
  - knowledge is acquired by a teacher or other organized source in the form of advice or instruction
  - learner needs to transform the knowledge from the input language to an internally-usable representation so that it could be integrated with the prior knowledge for effective use.
  - learner needs to perform some inference, but burden remains with the teacher who must provide the knowledge

Learning: Knowledge Acquisition

- Learning by Analogy
  - learner generalizes from its own experiences (teacher not involved)
  - learner transforms and augments existing knowledge applicable in one domain to perform a similar task in related domain
  - Learner requires more inference compared to rote learning or learning from taking advice
    - Fact or skill analogous in relevant parameters must be retrieved from memory, then appropriately transformed, applied to the new situation, and stored for future use.

Learning: Knowledge Acquisition

- Learning from Examples
  - given a set of examples and counter examples of a concept, the learner induces a general concept description that describes all of the positive examples and none of the counter examples
  - **source** of the examples can be a teacher (*Supervised learning*), or the learner itself, or external environment.
- Learning from Observation and Discovery
  - learning without a teacher (*Unsupervised learning*)
  - includes a variety of process, such as creating classifications of given observations, or discovering relationships or laws governing a given system
  - learner is not provide with a set of instances exemplifying a concept, nor is provided with an *oracle* for classifying internally generated instances

Learning: knowledge acquisition

Types (based on type of representation of knowledge (or skill) acquired by the learner)

- Parameters in algebraic expression
- Decision trees
- Formal grammars
- Production rules
- Formal logic-based expressions and related formalisms
- Graphs and Networks
- Taxonomies
- …

Learning

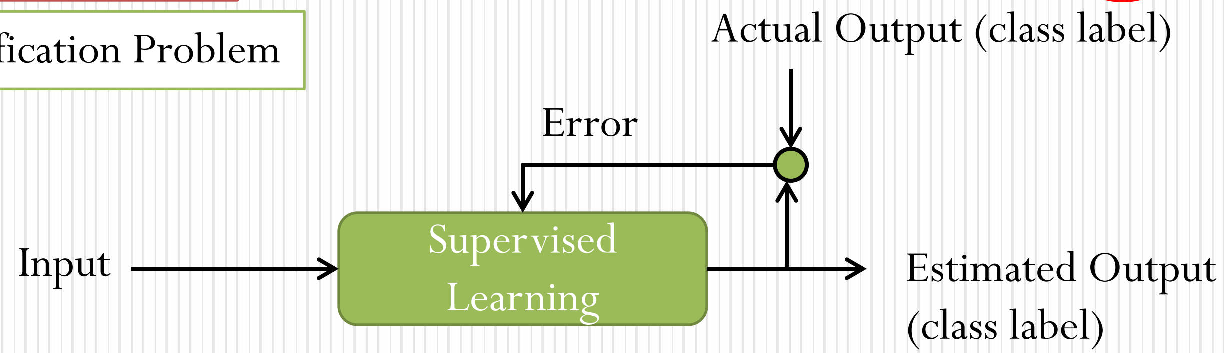
- Machine learning approaches are traditionally divided into three broad categories, depending on the type of the "signal" or "feedback" available to the learning system (agent):
  - Supervised learning: correct answers for each example
  - Unsupervised learning: correct answers not given (no explicit feedback)
  - Reinforcement learning: occasional rewards or punishments

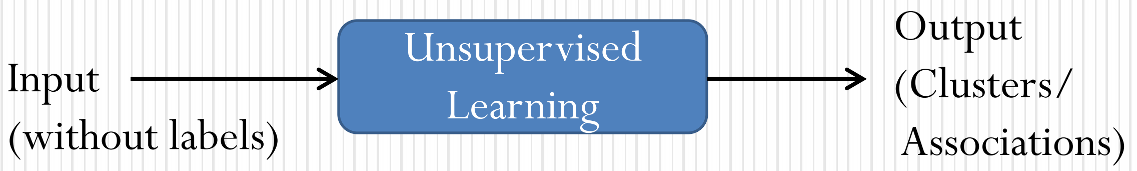
Loan Defaulter ?

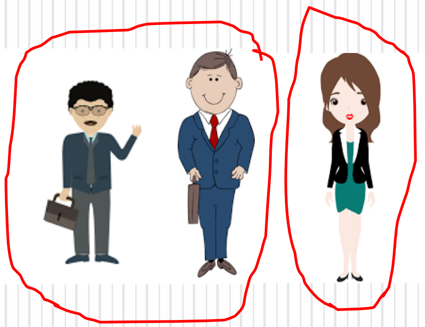
Classification Problem

Income Yearly	Age	Gender	Own House	Defaulter
1200000	40	MALE	YES	NO
100000	30	MALE	NO	YES

Not available for unsupervised learning







Supervised Learning

- Given a training set of N example input-output pairs
$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$$

where each y_i was generated by an unknown function $y = f(x)$

- The task of Supervised learning is to discover a function h (hypothesis) that approximates the true function f.
- Accuracy of the hypothesis is measured using a test set of examples that are distinct from the training set.
- A hypothesis generalizes well if it correctly predicts the value of y for unseen examples.
- y discrete : the learning problem is classification (such as sunny, cloudy, or rainy)
- y continuous: the learning problem is called regression (such as tomorrow's temperature)

Classification Problem

Loan Defaulter ?

Income Yearly	Age	Gender	Own House	Defaulter
1200000	40	MALE	YES	NO
1000000	30	MALE	NO	YES
1000000	50	FEMALE	NO	YES
⋮	⋮	⋮	⋮	⋮
800000	50	MALE	YES	NO

$\mathbf{x}_i$
Vector of feature values

y_i

Regression Problem

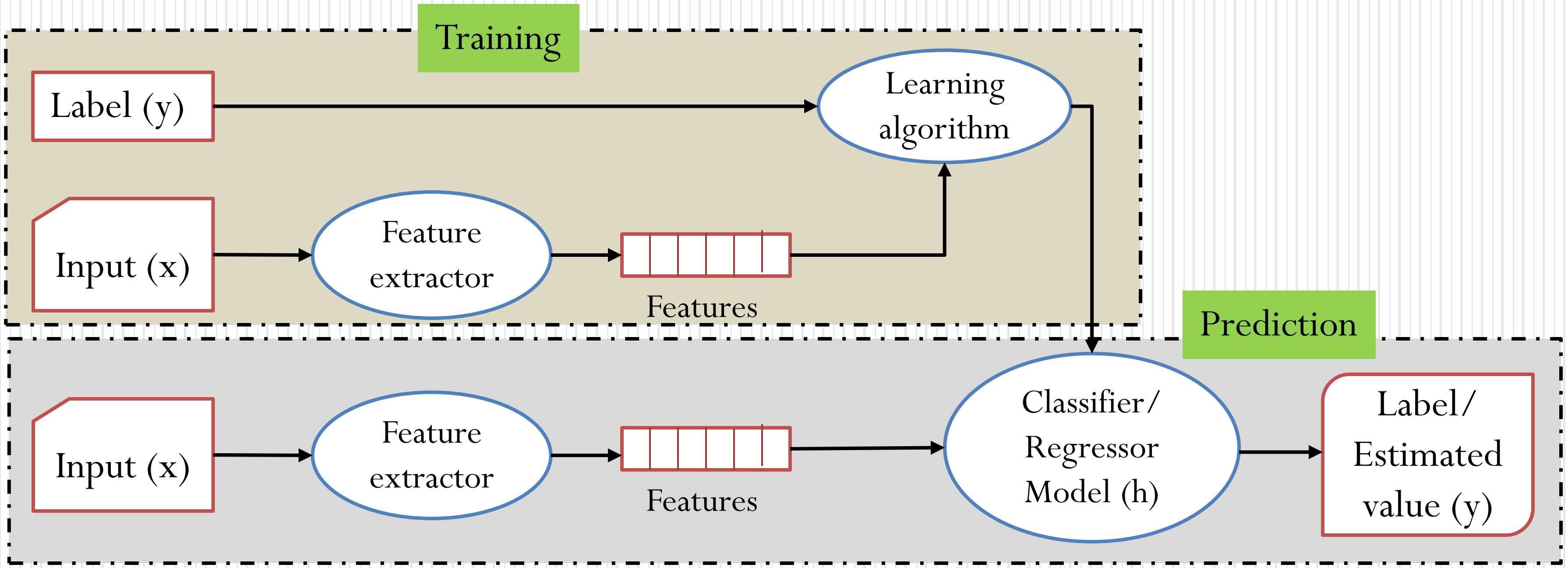
Yearly household energy consumption?

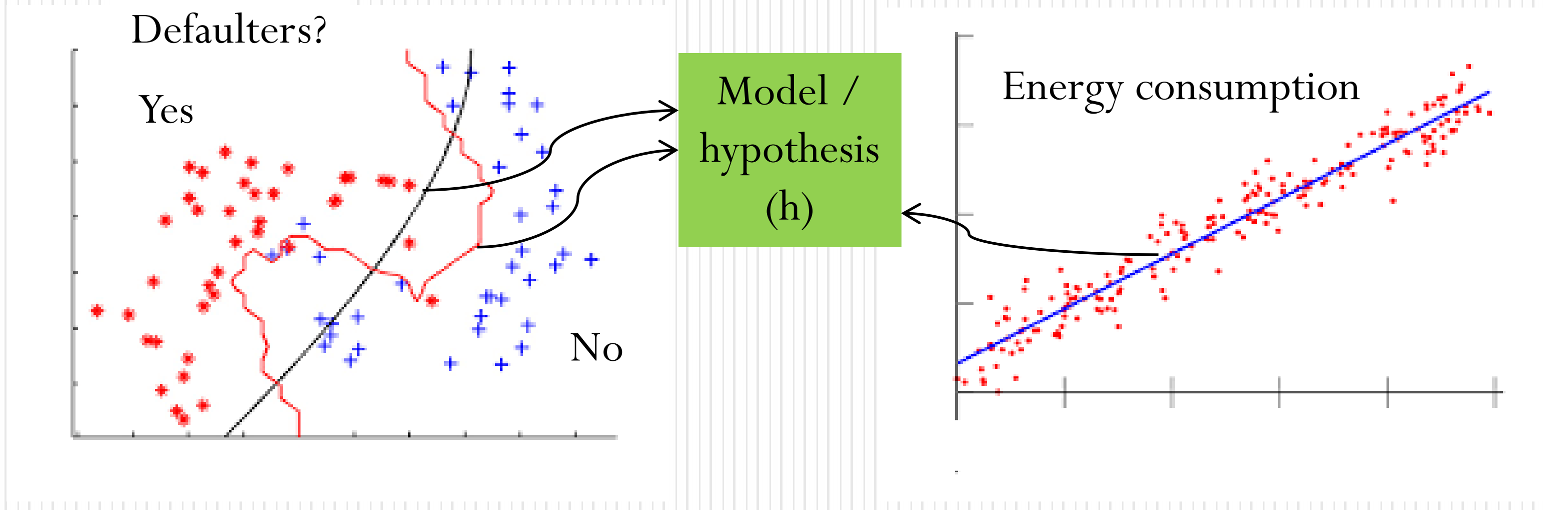
Family size	Yearly Income (Rs.)	Own House	No. of Appliances	Avg. Consumption (Kwh)
2	1000000	YES	5	6
6	2000000	NO	3	10
3	800000	NO	3	5.5
⋮	⋮	⋮	⋮	⋮
4	1000000	YES	4	6.8

Training set

Test set

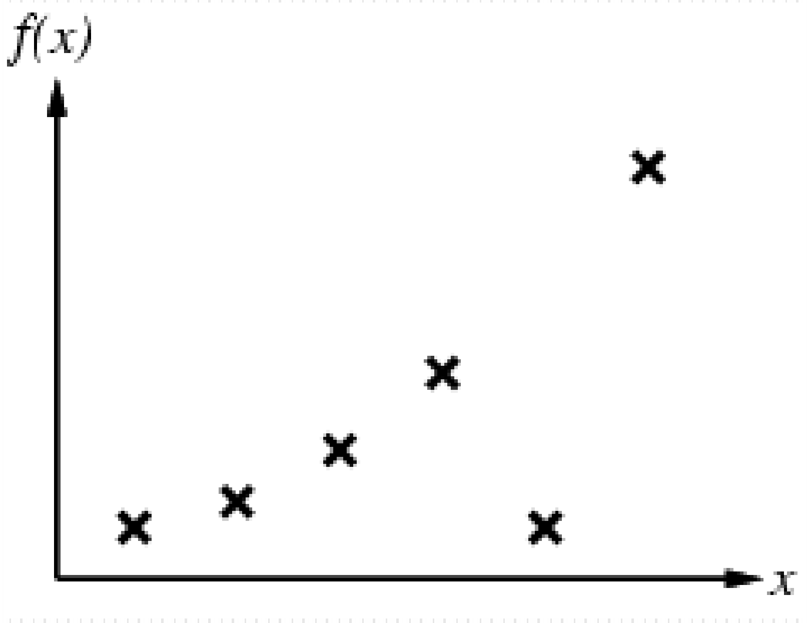
Supervised Learning





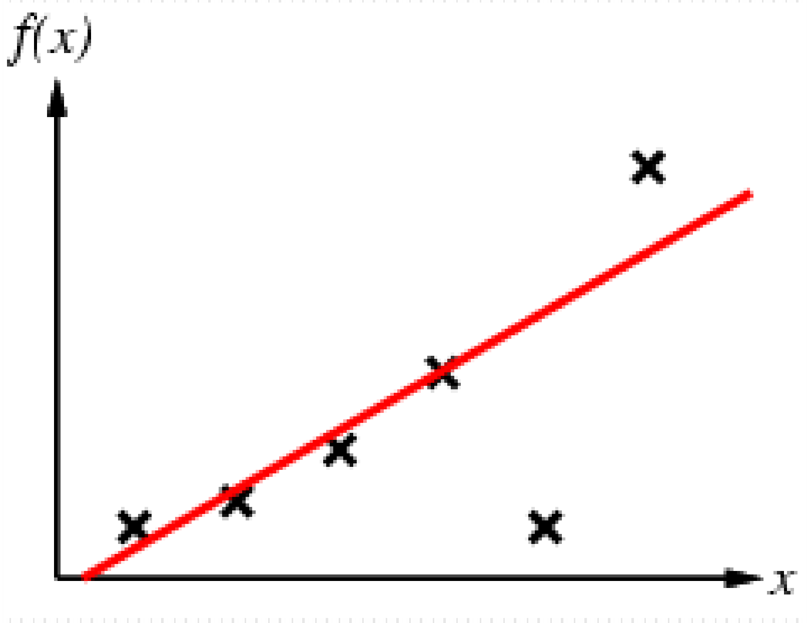
Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)



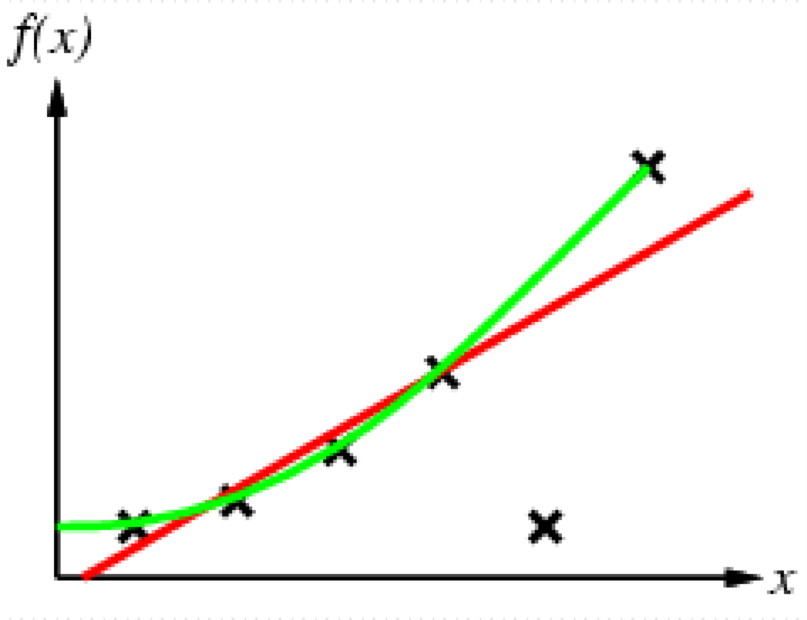
Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)



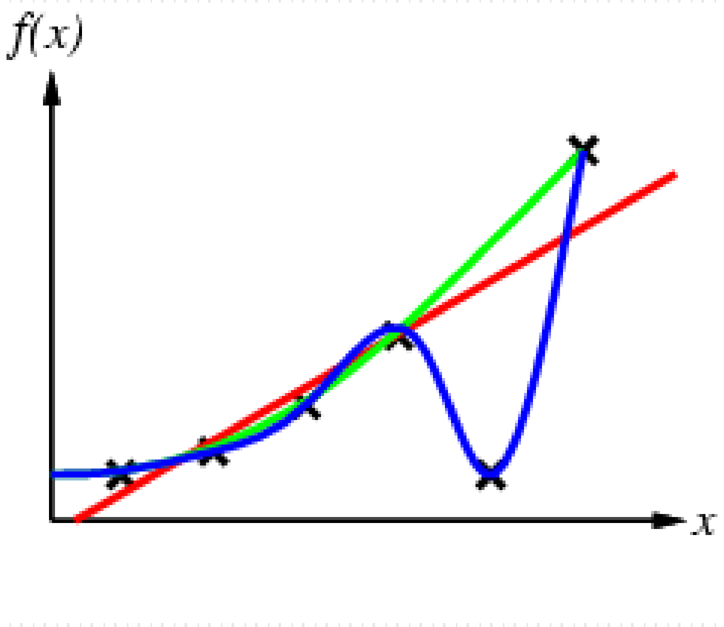
Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)



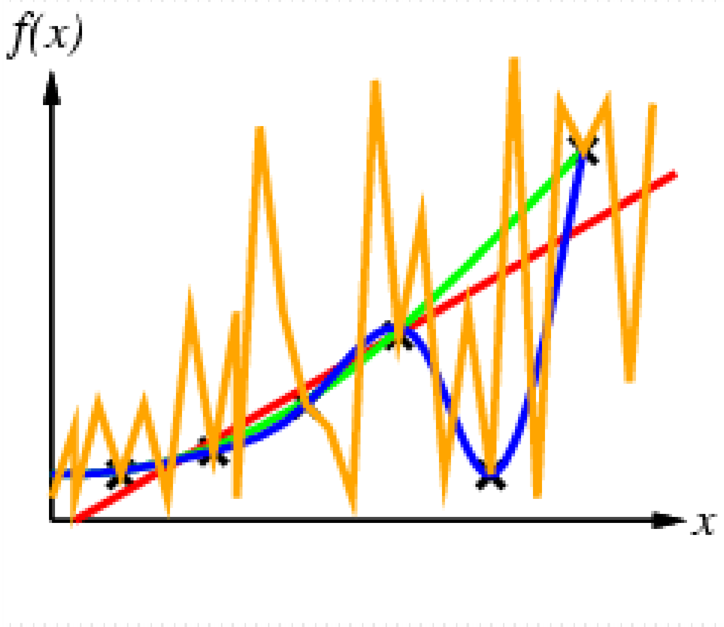
Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)



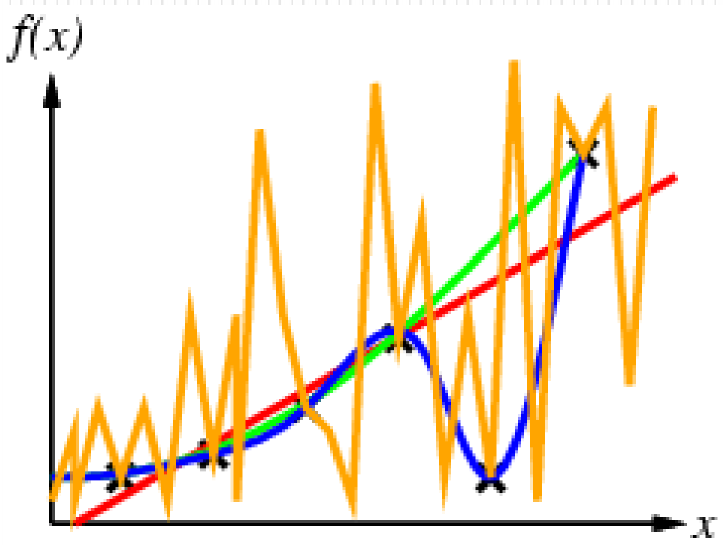
Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)



Supervised Learning

- Construct/adjust *h* to agree with *f* on training set
- (*h* is consistent if it agrees with *f* on all examples)

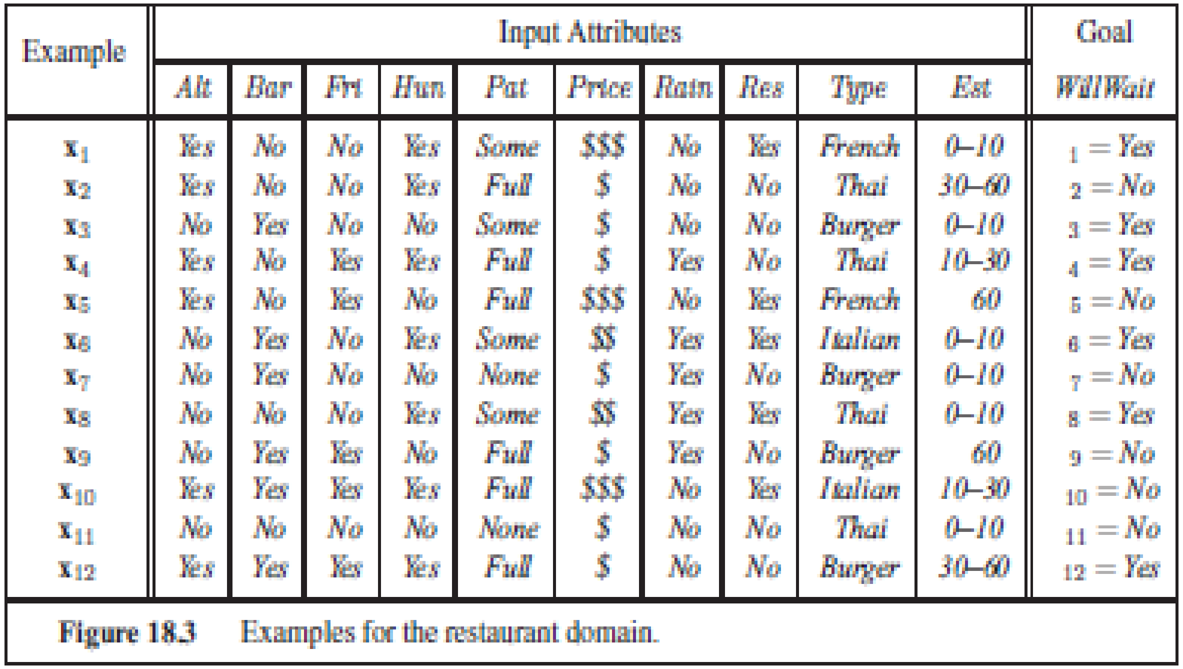


How do we choose from among multiple consistent hypothesis?

- Ockham's razor: prefer the simplest hypothesis consistent with data

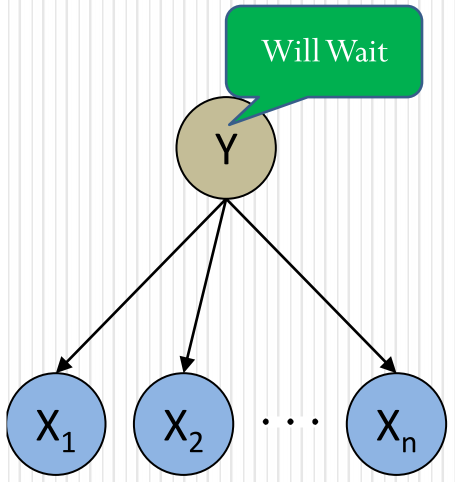
Classification using Naïve Bayes

- Naïve Bayes: Assume all features are independent effects of the label
- Structure known, learn the parameters
- Inference: given an instance determine the class label



Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
x_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$_1$ = *Yes*
x_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$_2$ = *No*
x_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$_3$ = *Yes*
x_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$_4$ = *Yes*
x_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*60*	$_5$ = *No*
x_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$_6$ = *Yes*
x_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$_7$ = *No*
x_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$_8$ = *Yes*
x_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*60*	$_9$ = *No*
x_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$_{10}$ = *No*
x_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$_{11}$ = *No*
x_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$_{12}$ = *Yes*

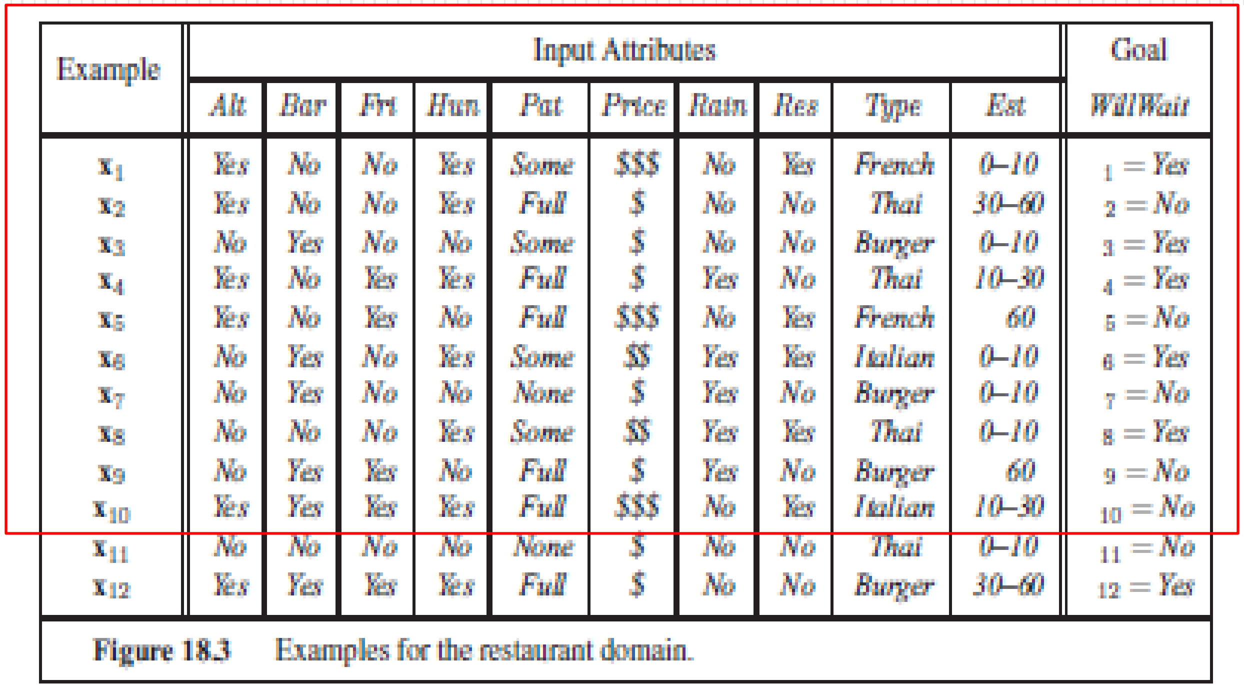
Figure 18.3 Examples for the restaurant domain.



Classification using Naïve Bayes

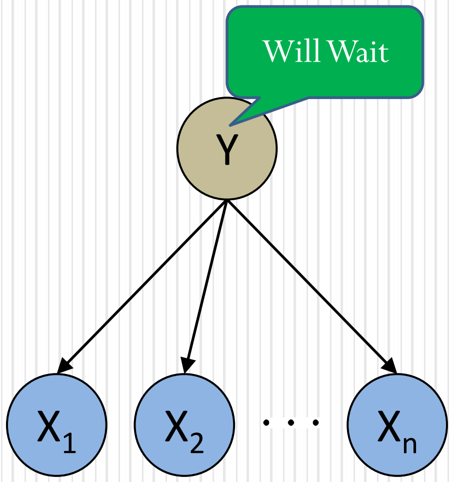
$$\mathbf{P}(C \mid x_1, \ldots, x_n) = \alpha\, \mathbf{P}(C) \prod_i \mathbf{P}(x_i \mid C)\,.$$

- Learn $P(C)$ and $P(x_i \mid C)$ from the examples
- $P(WillWait = Yes), P(Alt = Yes|Willwait = Yes), P(Alt = Yes|WillWait = No)$



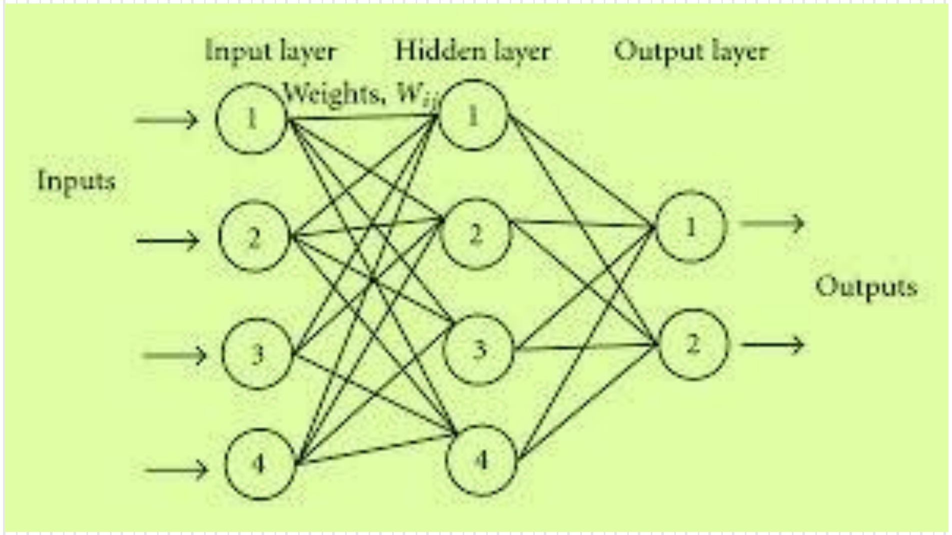
Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$_1 =$ *Yes*
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$_2 =$ *No*
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$_3 =$ *Yes*
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$_4 =$ *Yes*
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*60*	$_5 =$ *No*
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$_6 =$ *Yes*
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$_7 =$ *No*
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$_8 =$ *Yes*
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*60*	$_9 =$ *No*
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$_{10} =$ *No*
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$_{11} =$ *No*
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$_{12} =$ *Yes*

Figure 18.3 Examples for the restaurant domain.

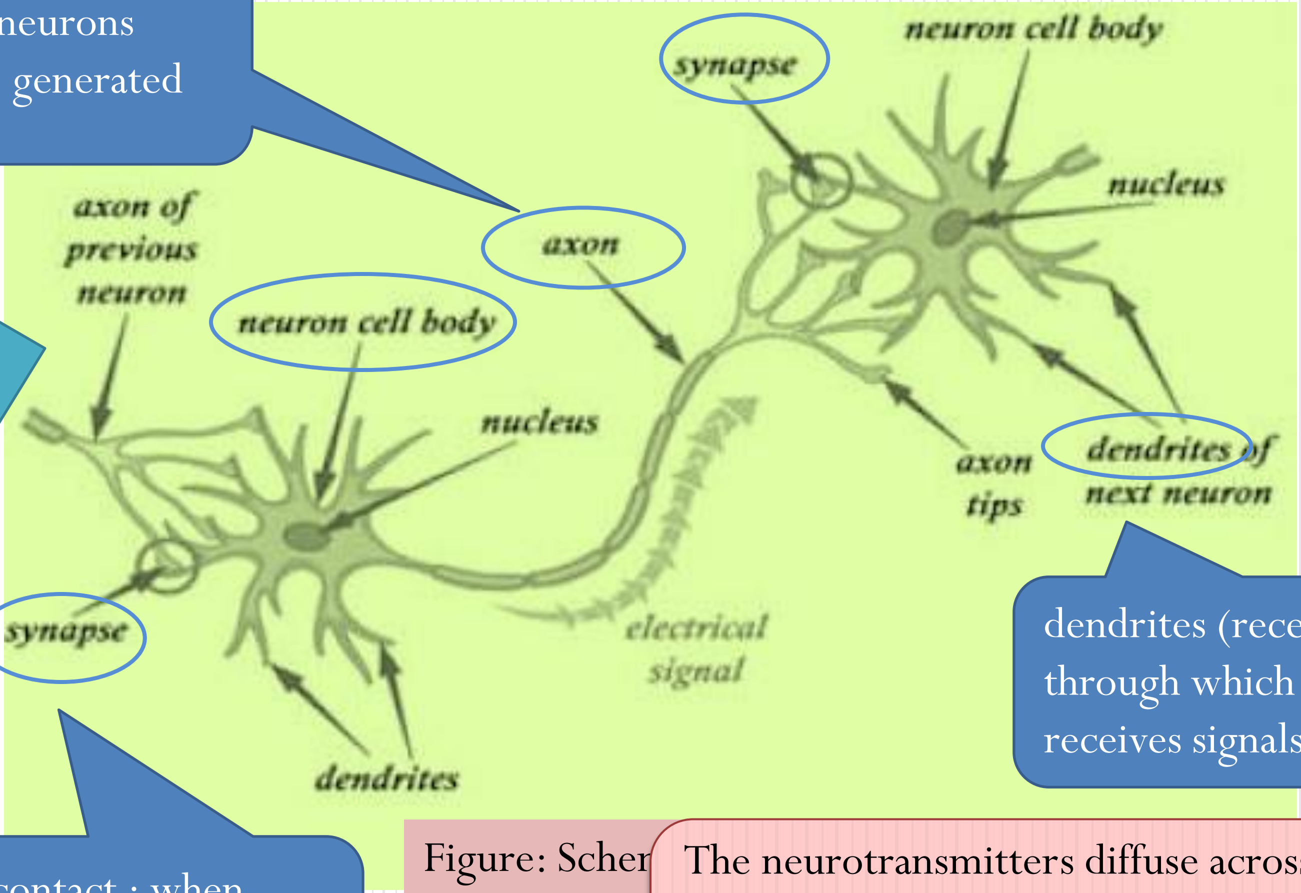


Artificial Neural Networks : Introduction

- Artificial Neural Network (ANN)
  - information processing paradigm inspired by the way biological nervous systems (brain) process information
  - made up of a number of simple, highly interconnected processing elements that tries to emulate the biological neural networks



Biological Neurons



axons (transmitter) : through which neurons transmit signals generated by its cell body.

nucleus and plasma: contains information about the hereditary traits , produces material needed by the neuron

dendrites (receivers) : through which neurons receives signals (impulses)

axon-dendrite contact : when impulse reaches it releases chemical called neurotransmitters

Figure: Scher

The neurotransmitters diffuse across the synaptic gap to enhance or inhibit (depending on the type of synapse) the receptor neuron's own tendency to emit electrical impulses.

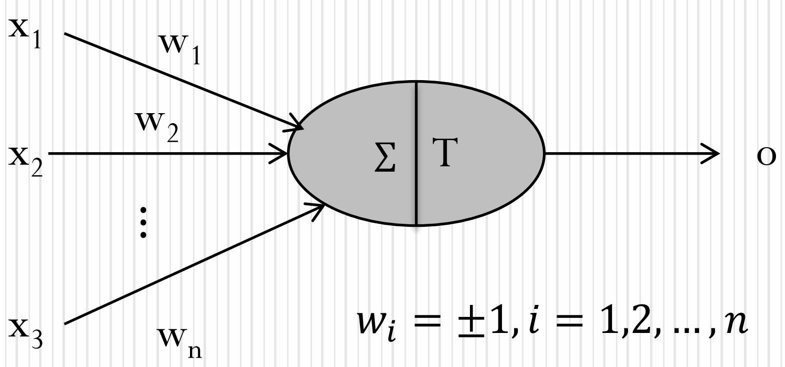
Biological Neurons

- Neuron responds to the total of its inputs aggregated within short time interval called the *period of latent summation*.
- Neuron generates a pulse response and sends it to its axon only if conditions necessary for firing are fulfilled.
- Incoming impulses can be *excitatory* if they cause the firing, or *inhibitory* if they hinder the firing of the response.
- Firing occurs when excitation exceeds the inhibition by the amount called the *threshold* of the neuron.

ANN incorporates the two fundamental components of biological Neural Nets:
Neurons (Nodes) and Synapses (Weights)

Artificial Neurons

- The first formal definition of an artificial neuron model was given by McCulloch and Pits (1943).
  - proposed a binary threshold unit as a computational model for an artificial neuron.



$w_i = \pm 1, i = 1,2, \ldots, n$

Firing rule for this model is:

$$o^{k+1} = \begin{cases} 1 \; if \; \sum_{i=1}^{n} w_i x_i^k \geq T \\ 0 \; if \; \sum_{i=1}^{n} w_i x_i^k < T \end{cases}$$

$k = 0,1,2, \ldots$ denotes discrete $-$ time instant

  - The neuron computes a weighted sum of its n input signals and generates an output of 1 if this sum is above threshold T.

Artificial Neurons

- Positive weights correspond to *excitatory* synapses while negative weights model *inhibitory* ones.

- T is the neuron’s threshold value, which needs to be exceeded by the weighted sum of signals for the neuron to fire.

- Although the model is simplistic, it can perform the basic logic operations NOT, OR, and AND, provided its weights and thresholds are appropriately selected.

McCulloch-Pitts model

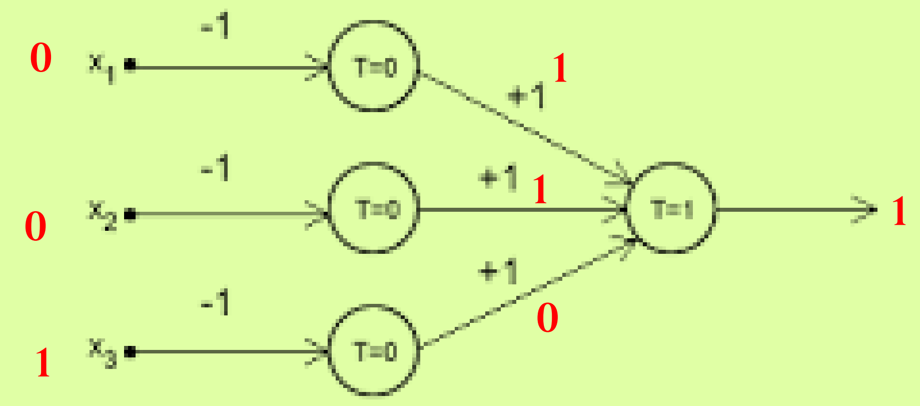
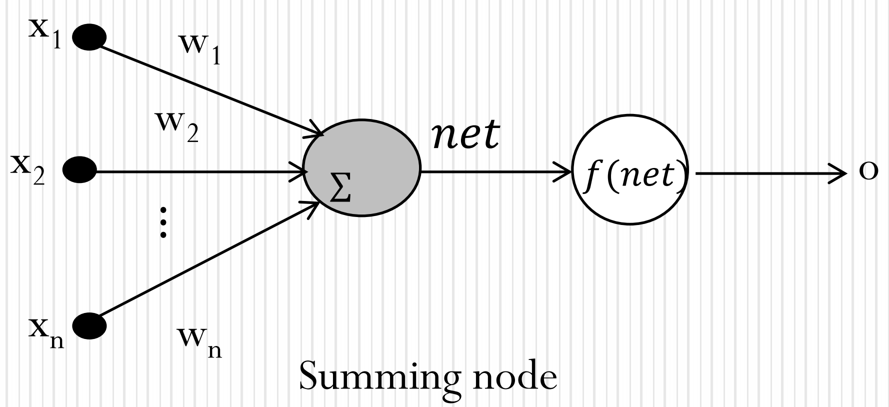


Figure: NAND gate with McCulloch-Pitts neuron model

- Features of McCulloch-Pitts model
  - Allows binary 0,1 states only
  - Weights and the neurons' thresholds are fixed in the model
  - Just a primitive model

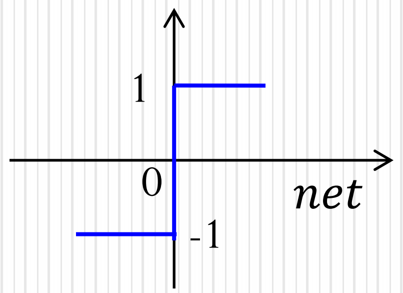
The Perceptron

- The Perceptron, introduced by Frank Rosenblatt (1958), can be considered as single layer network
- weights can be determined analytically or by a learning rule
- the weights and thresholds are not all identical
- Weights can be positive or negative



Firing rule for this model is:

$$f(net) = \begin{cases} 1 \; if \; net > 0 \\ -1 \; if \; net < 0 \end{cases}$$



XOR Problem

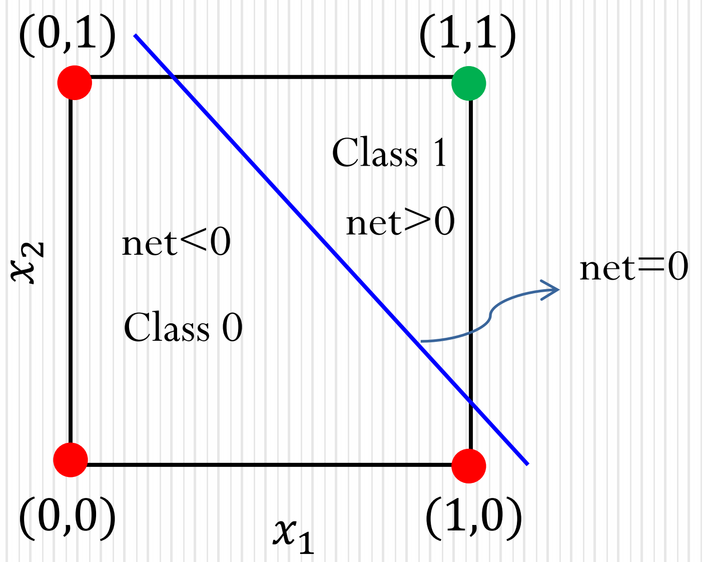


Fig. two-input AND operation

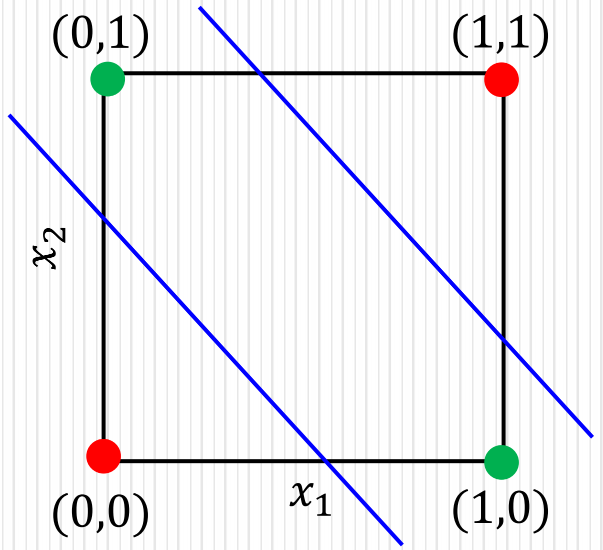
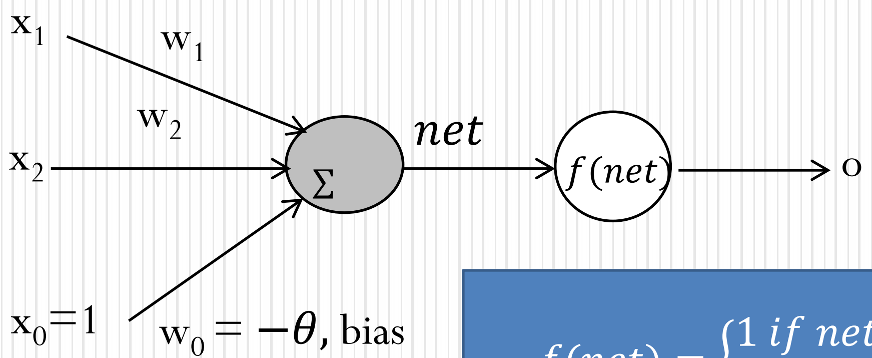


Fig. two-input XOR operation



$$f(net) = \begin{cases} 1 \ if \ net > 0 \\ 0 \ if \ net < 0 \end{cases}$$

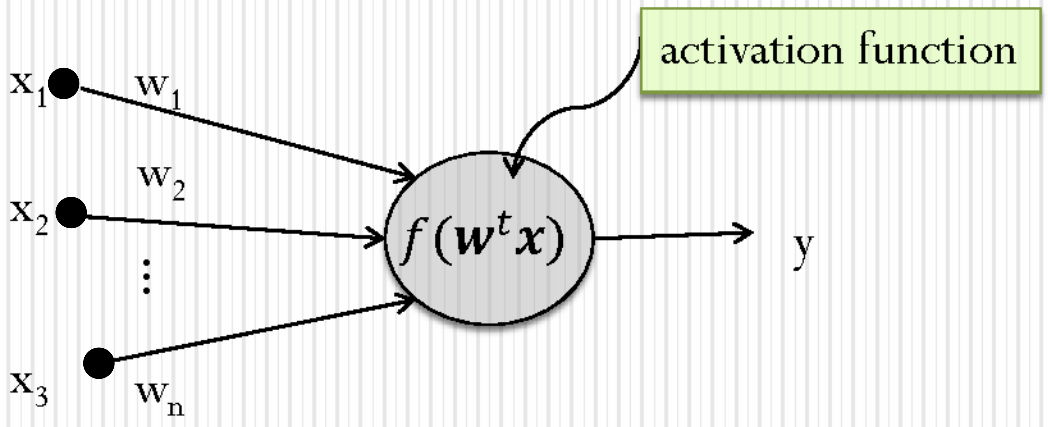
$$net = w_1 x_1 + w_2 x_2 - \theta$$

Linear in components of $\mathbf{x}$

by finding suitable coefficients of the line (w_1, w_2 , θ) we can get the linear model for AND function, but NOT possible for XOR

Artificial Neurons : General representation

- A general representation of artificial neuron.



$$y = f(\boldsymbol{w}^t\boldsymbol{x}) = f\left(\sum_{i=1}^{n} w_i x_i\right) = f(net)$$

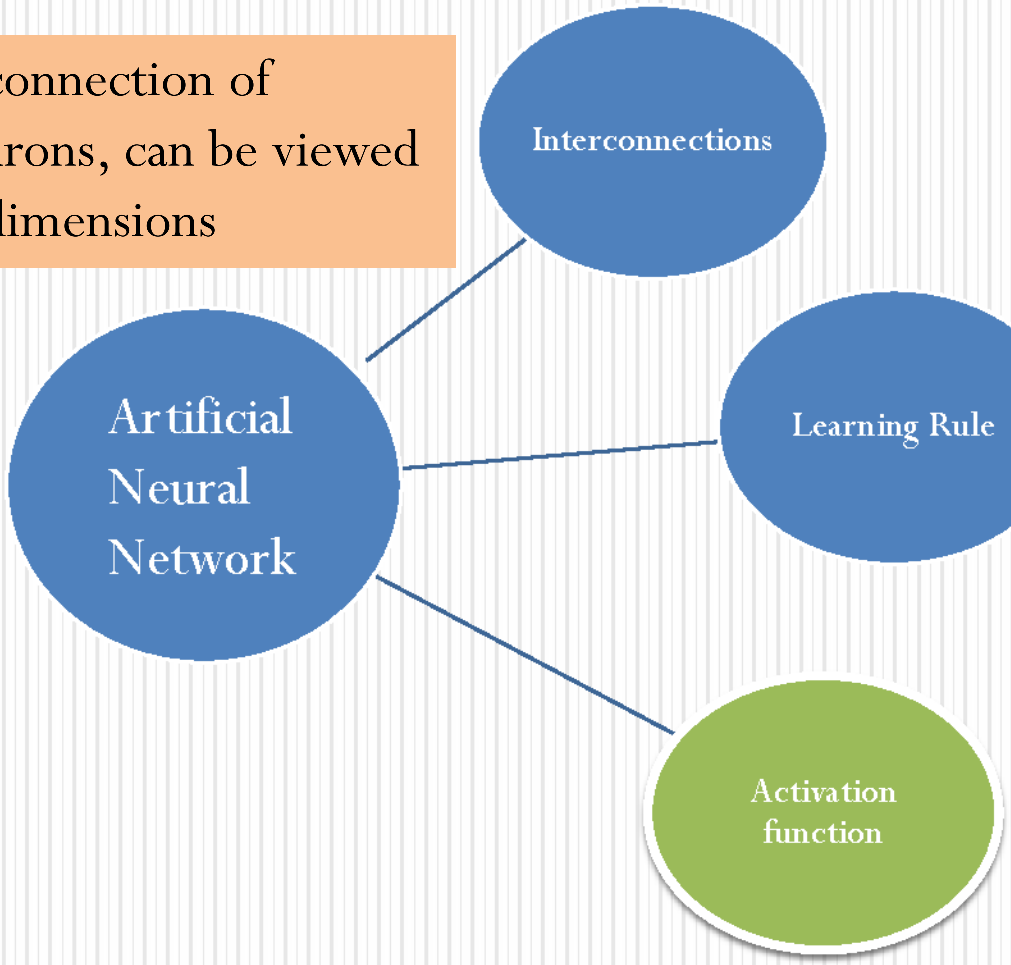
Weight vector $\boldsymbol{w} = [w_1 \quad \dots \quad w_n]^t$

Input vector $\boldsymbol{x} = [x_1 \quad \dots \quad x_n]^t$

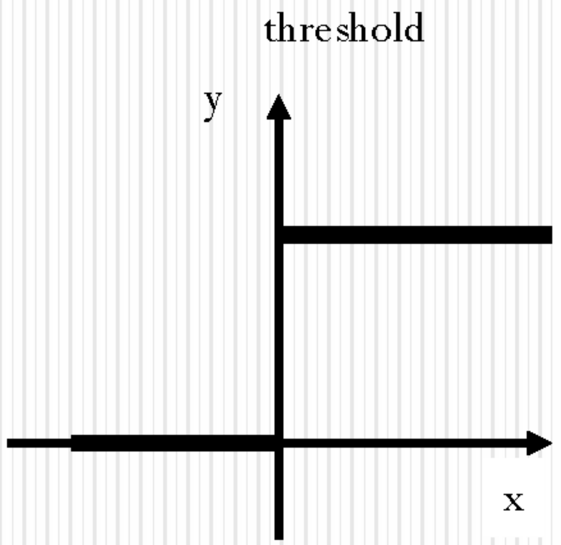
superscript t is denoting transposition

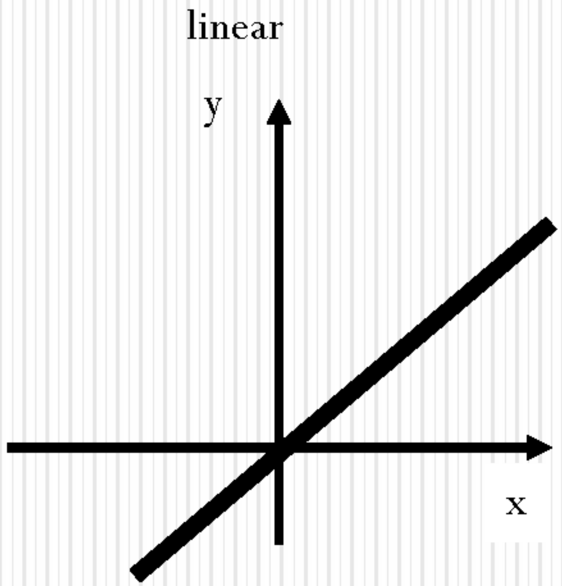
Artificial Neural Network

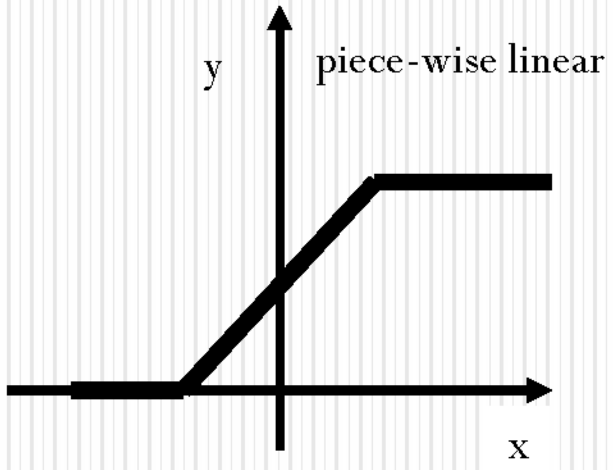
ANN: interconnection of artificial neurons, can be viewed from three dimensions

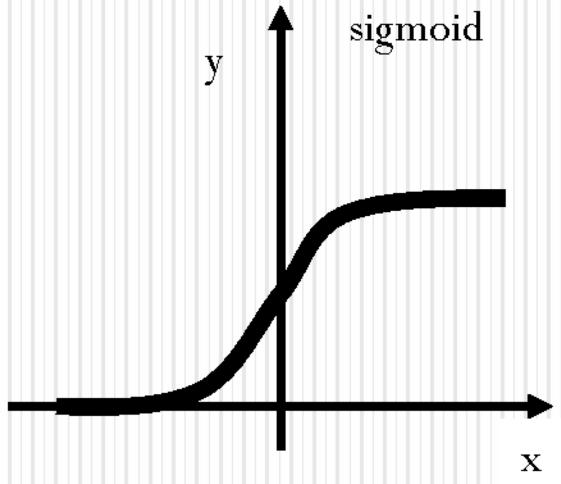


Activation Functions

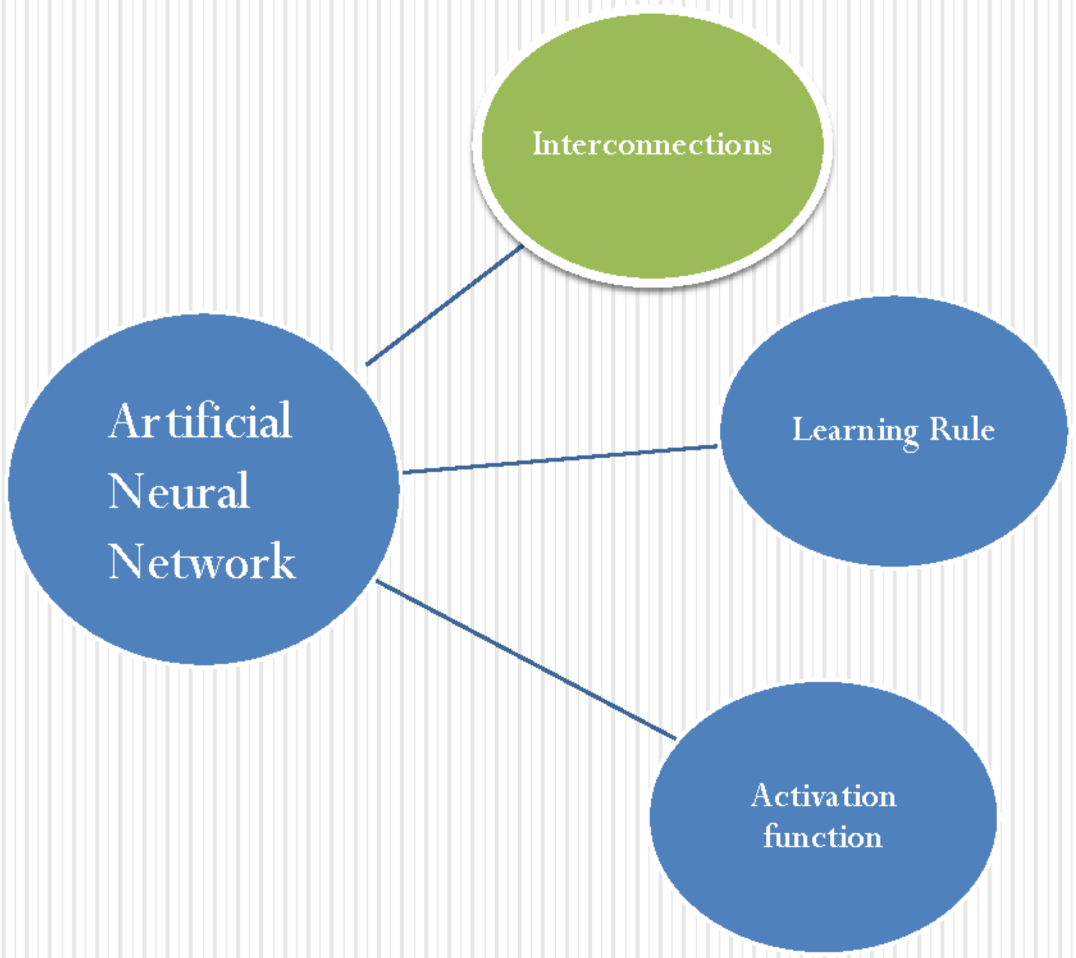








Artificial Neural Network



Network Architecture

- ANNs can be grouped into two categories based on the connection pattern (architecture):
- feedforward network
  - neurons are organized into layers that have unidirectional connections between them
  - lack of feedback (no loops).
- feedback (or recurrent) network
  - can be obtained from feedforward network by connecting the neurons' outputs to their inputs (feedback).

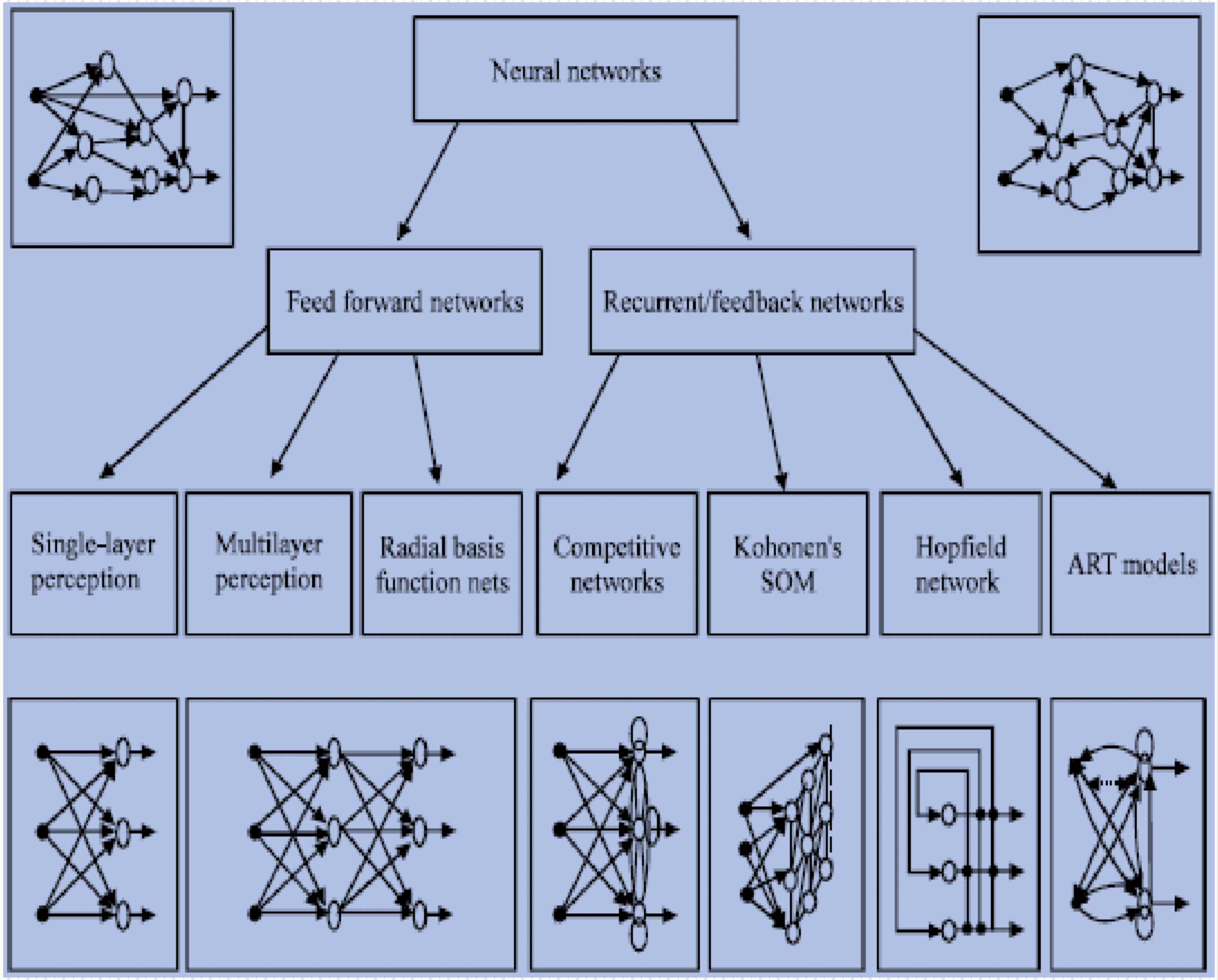
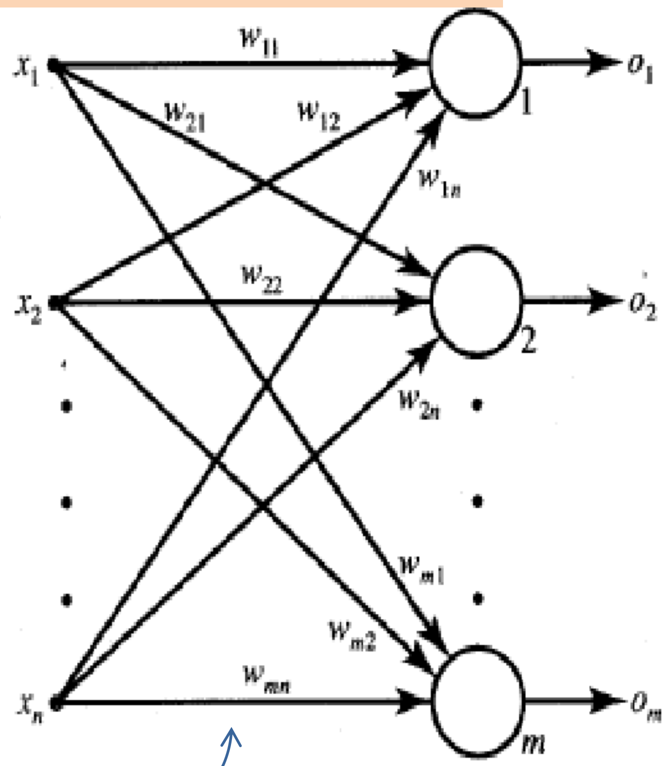


Figure taken from A. K. Jain, Jianchang Mao and K. M. Mohiuddin, "Artificial neural networks: a tutorial," in *Computer*, vol. 29, no. 3, pp. 31-44, Mar 1996.

Single layer Feedforward Network

Inputs connected directly to outputs



Input and output vectors

$$\mathbf{o} = \begin{bmatrix} o_1 & o_2 & \cdots & o_m \end{bmatrix}^T$$

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$$

Weight w_{ij} connects the *i'th* neuron with *j'th* input. Activation rule of ith neuron is

$$net_i = \sum_{j=1}^{n} w_{ij}x_j, \quad \text{for } i = 1, 2, \ldots, m$$

$$o_i = f(net_i)$$

Activation function

Single layer of adaptive weights

Single Layer Feedforward Network

- The mapping of input space to output space can be expressed using matrices :

$$\mathbf{O} = \mathbf{\Gamma}[\mathbf{W}\mathbf{x}]$$

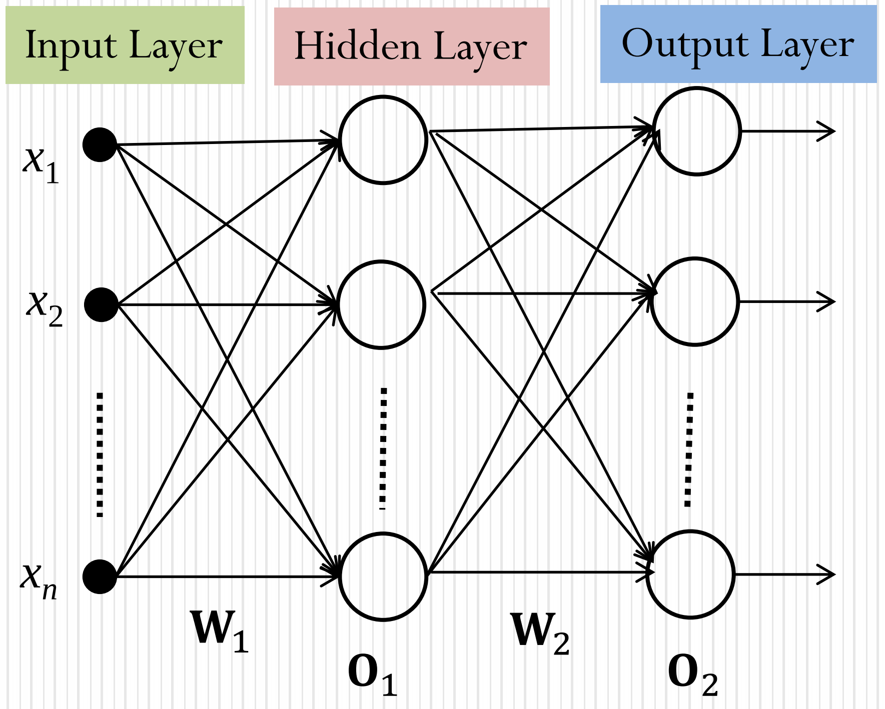
$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$ is the weight matrix

$$\mathbf{x} = [x_1 \quad x_2 .. \quad x_n]^{\mathbf{T}}$$

$$\mathbf{\Gamma}[.] = \begin{bmatrix} f(.) & 0 & \cdots & 0 \\ 0 & f(.) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & f(.) \end{bmatrix}$$

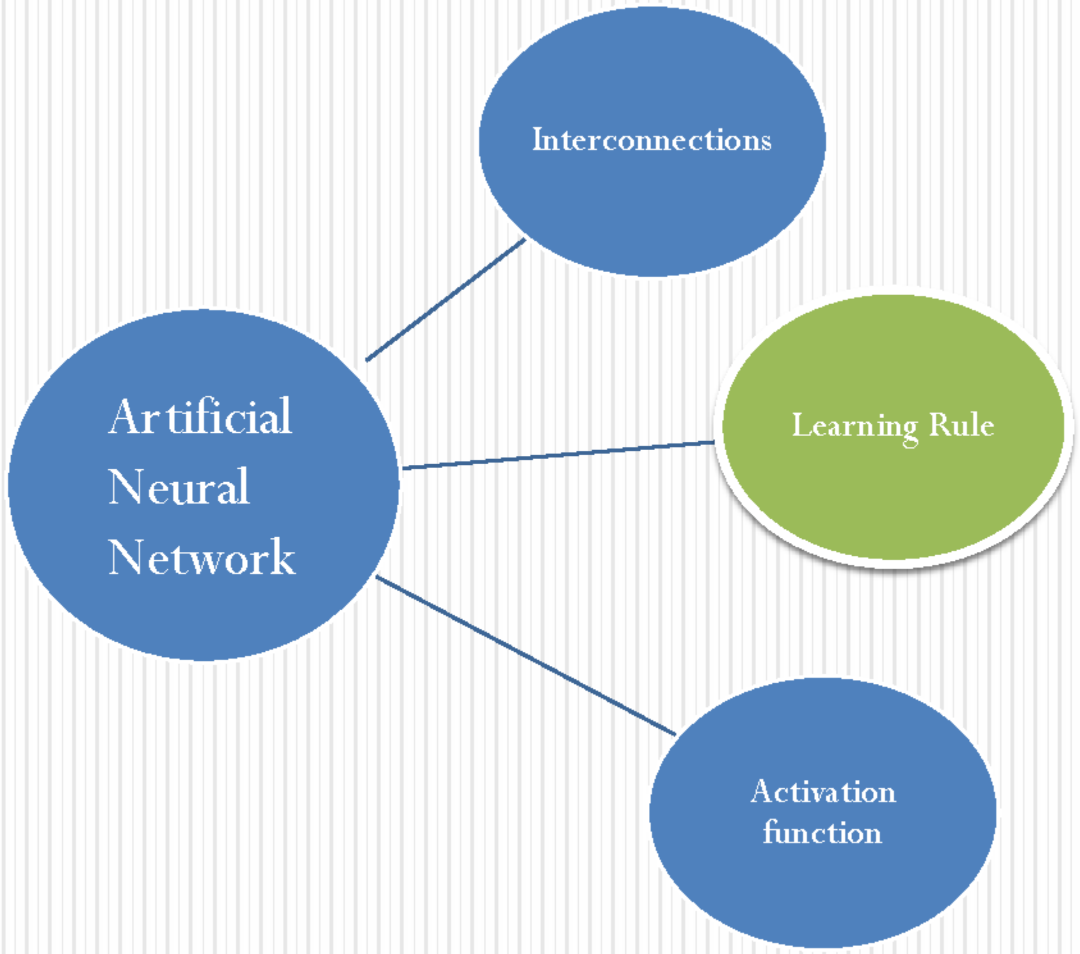
- The nonlinear activation functions $f(.)$ on the diagonal of the matrix operator $\mathbf{\Gamma}$ operated on component-wise on the activation values net of each neuron.

Multilayer feedforward Network



$$\mathbf{W_1} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

Artificial Neural Network



Learning in ANN

- Learning is a process by which a NN adapts itself to a stimulus by making proper parameter adjustments, resulting in the production of desired response
- Two kinds of learning
  - Parameter learning: connection weights are updated
  - Structure Learning: change in network structure
- To design a learning process one must know
  - what information is available to network: learning paradigms (supervised, unsupervised, reinforcement, and hybrid)
  - how network weights (parameters) are updated, i.e. which learning rules govern the update process.
  - A learning algorithm refers to a procedure in which learning rules are used for adjusting the weights.

Supervised Learning

- It assumes the availability of a labeled (i.e., ground-truthed) set of training data made up of N input—output examples:
- Training Sample : $(\boldsymbol{x}_i, d_i), i = 1, .., N$

  $\boldsymbol{x}_i$ = input vector of ith example

  d_i = desired (target) response of ith example, assumed to be scalar for convenience of presentation

  N = sample size
- Given the training sample, the requirement is to compute parameters of the neural network so that the actual output o_i of the neural network due to $\boldsymbol{x}_i$ is close enough to d_i for all i.
  - For example, we may use the mean-square error as the index of performance to be minimized.
  - $E = \frac{1}{N}\sum_{i=1}^{N}(d_i - o_i)^2$

Some Learning rules

- Error correction (Perceptron learning)
- Delta rule
- Widrow-Hoff Learning rule
- Hebbian learning
- Competitive Learning

The weight vector $\mathbf{w_i}$ increases in proportion to the product of input $\mathbf{x}$ and learning signal r.

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i)$$

Weight increment is given as

$$\Delta\mathbf{w}_\text{i}(t) = \eta r[\mathbf{w}_\text{i}(t), \mathbf{x}(t), d_i(t)]\mathbf{x}(t)$$

η is positive constant that determines rate of learning

$$\mathbf{w}_i(t+1) = w_i\ (t) + \eta r[\mathbf{w}_\text{i}(t), \mathbf{x}(t), d_i(t)]\mathbf{x}(t)$$

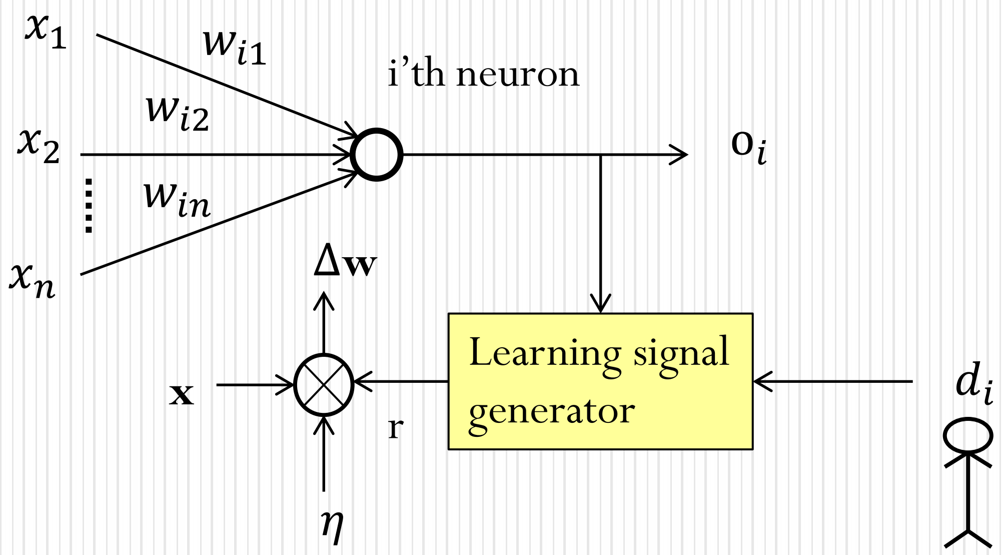
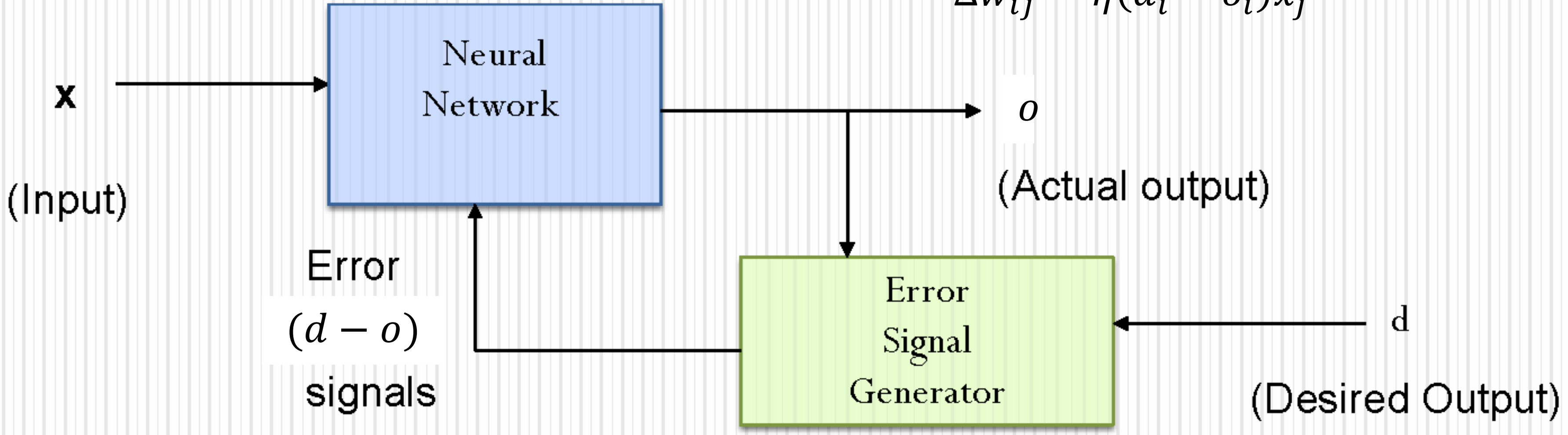


Figure: Illustration for weight learning rules , d_i is provided only for supervised learning

Error correction (Perceptron learning)

in supervised learning paradigm it uses the error signal (d-o) to modify the connection weights to gradually reduce this error

$$r = d_i - o_i$$
$$\Delta w_{ij} = \eta(d_i - o_i)x_j$$



Delta Learning rule

- Delta learning is valid for continuous activation functions and in supervised training mode.
- learning signal for this rule is called delta and defined as

$$r = [d_i - f(\mathbf{w}_i^T\mathbf{x})]f'(\mathbf{w}_i^T\mathbf{x})$$

$f'(\mathbf{w}_i^T\mathbf{x})$ is the derivative of the activation function $f(net)$ computed for $net = \mathbf{w}_i^T\mathbf{x}$

- the learning rule can be derived from the condition of the least squared error between o_i and d_i
- the squared error is given as $$E = \frac{1}{2}(d_i - o_i)^2 = \frac{1}{2}\left(d_i - f(\mathbf{w}_i^T\mathbf{x})\right)^2$$
- the gradient vector with respect to $\mathbf{w}_i$ of the squared error can be given as

$$\nabla\mathbf{E} = -(d_i - o_i)f'(\mathbf{w}_i^T\mathbf{x})\mathbf{x}$$

Delta Learning rule

- the components of the error gradient vector $\mathbf{\nabla E}$ are

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i)f'(\mathbf{w}_i^T\mathbf{x})x_{\boldsymbol{j}} \text{ , for } j = 1,2,\ldots,n$$

- since the minimization of the error requires the weight changes to be in the negative gradient direction, we take

$$\Delta\mathbf{w}_i = -\,\eta\mathbf{\nabla E}$$

$$\Delta w_{ij} = \eta(d_i - o_i)f'(net_i)x_j \text{ , for } j = 1,2,\ldots,n$$

- It parallels the discrete perceptron learning rule and can be called as continuous perceptron rule.
- delta rule can be generalized for multilayer networks

Widrow-Hoff Learning rule

- minimizes the squared error between the desired output (d_i) and the neurons activation value (net_i)

$$r = d_i - \mathbf{w}_i^T \mathbf{x}$$

$$\Delta \mathbf{w}_i = \eta (d_i - \mathbf{w}_i^T \mathbf{x}) \mathbf{x}$$

$$\Delta w_{ij} = \eta \left(d_i - \mathbf{w}_i^T \mathbf{x} \right) x_j \quad \text{for } j = 1,2,\ldots,n$$

- special case of the delta learning rule assuming $f(net) = net$ or activation function is simply the identity function, we obtain $f'(net) = 1$, this rule is sometimes called the LMS (least mean square) learning rule
- weights are initialized at any values in this method.

Hebbian Learning

- implements the interpretation of the *Hebb's postulate of learning (Hebb, 1949)*
  - *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.*
- This rule can be rephrased as (*Stent, 1973; Changeux and Danchin, 1976*)
  - If two neurons on either side of a synapse (connection) are activated simultaneously (i.e. synchronously), then the strength of that synapse is selectively increased.

Neurons that fire together , wire together

Hebbian Learning

- learning signal r is simply the neuron's output $r = f(\mathbf{w}_i^T \mathbf{x})$
- increment $\Delta \mathbf{w}_\mathrm{i}$ of the weight vector becomes

$$\Delta \mathbf{w}_\mathrm{i} = \eta f(\mathbf{w}_i^T \mathbf{x})\mathbf{x}$$

- Single weight is adapted using the following increment

$$\Delta w_\mathrm{ij} = \eta o_i x_j \quad \text{for } j = 1, 2, \ldots, n$$

this states that if the cross-product of output and input, or correlation term $o_i x_j$ is positive, this results in an increase of weight w_ij; otherwise the weight decreases.

Competitive Learning

- used for unsupervised network training
- output neurons of a neural network compete among themselves to become active (fired)
- only a single output is active at one time
- learning based on the premise that one of the neurons (k^{th}) has the maximum response due to input x, this neuron is declared the winner
- only the weights associated with the winning neuron are adjusted

$$\Delta w_{kj} = \eta(x_j - w_{kj})$$

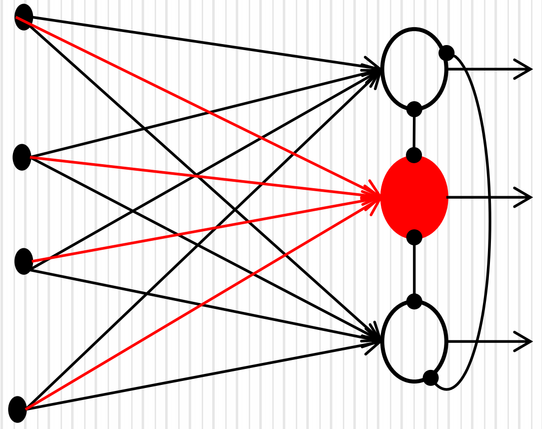
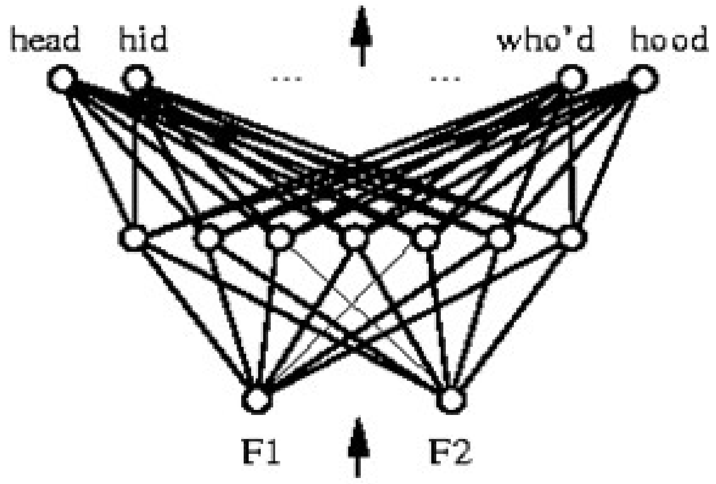


Figure : Competitive unsupervised "winner take-all" learning rule

Multilayer Feedforward Network and Backpropagation Learning



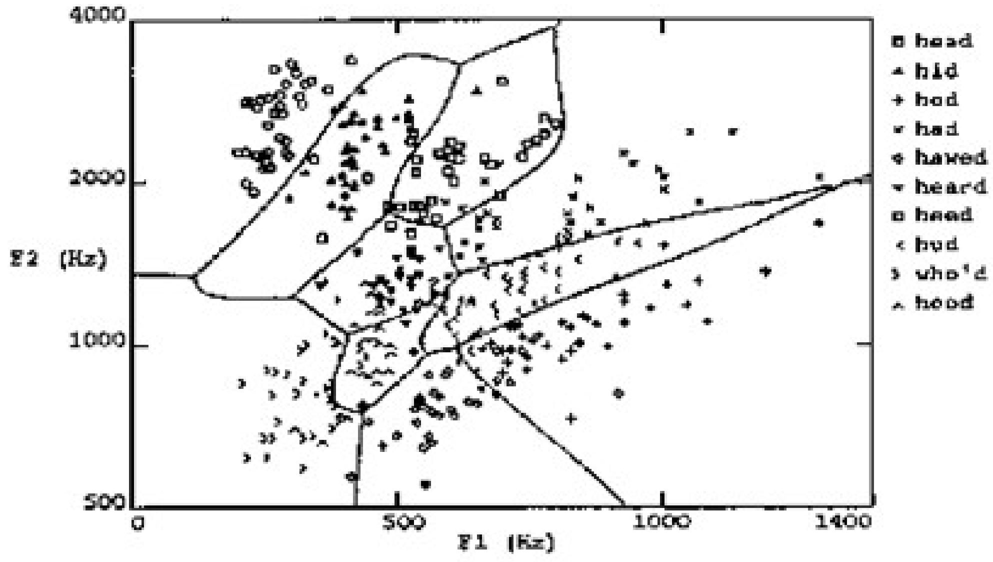


Figure: Decision regions of multilayer feedforward network (source: Machine learning, T.M. Mitchell)

In contrast to single perceptrons, Multilayer networks learnt by back propagation algorithm are capable of expressing non-linear decision surfaces.
The network in figure, trained to recognize 1 of 10 vowel sounds occurring in the context "h_d" (e.g. "had", "hid"). F1 and F2 two input parameters obtained from spectral analysis of the sound

Multilayer Feedforward Network and Backpropagation Learning

- Backpropagation algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections.
- employs gradient descent to attempt to minimize the squared error between the network output values and target (desired) values for these outputs

$$E \quad = \frac{1}{2} \sum_{i \in D} \sum_{k \in outputs} (d_{ki} - o_{kd})^2$$

where $outputs$ is the set of output units in the network, D is the set of training examples, d_{ki} and o_{ki} are the desired and output values associated with k^{th}output unit and training example i.

Backpropagation Learning

- the learning problem is to search a large hypothesis space defined by all possible weight values for all the units of the network
- **Gradient descent rule**
  - calculate the direction of steepest descent along the error surface

error surface can have multiple local minima, in case of multilayer networks

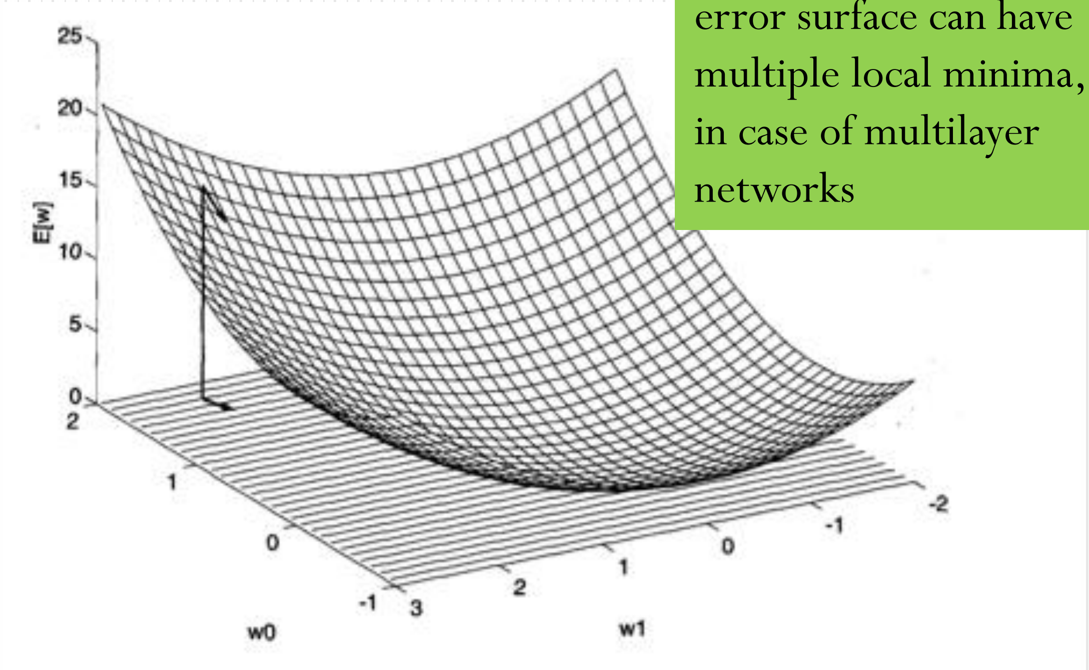


FIGURE 4.4
Error of different hypotheses. For a linear unit with two weights, the hypothesis space H is the w_0, w_1 plane. The vertical axis indicates the error of the corresponding weight vector hypothesis, relative to a fixed set of training examples. The arrow shows the negated gradient at one particular point, indicating the direction in the w_0, w_1 plane producing steepest descent along the error surface.

Figure taken from Machine Learning, T.M. Mitchell, McGraw-Hill

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{w_0}, \frac{\partial E}{w_1}, \dots \frac{\partial E}{w_n}\right]$$

$\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$

$\Delta\mathbf{w} = -\eta \nabla E(\mathbf{w})$, η is learning rate

$w_i = w_i + \Delta w_i$, for each component and where $\Delta w_i = -\eta \dfrac{\partial E}{w_i}$

Multilayer Feedforward Network and Backpropagation Learning

- The training of an MLP is usually accomplished by using a backpropagation (BP) algorithm that involves two phases :
  - **Forward Phase:**
    - parameters of the network are fixed, and the input signal is propagated through the network layer by layer and the error signal is computed.
  - **Backward Phase:**
    - the error signal is propagated through the network in the backward direction, hence the name of the algorithm.
    - adjustments are applied to parameters of the network so as to minimize the error.

Multilayer Feedforward Network and Backpropagation Learning

- Back-propagation learning may be implemented in one of two basic ways, as summarized here:
  - Sequential mode (also referred to as the on-line mode or stochastic mode):
    - adjustments are made to the parameters of the network on an example-by-example basis
  - Batch mode:
    - adjustments are made to the parameters of the network on an epoch-by-epoch basis, where each epoch consists of the entire set of training examples

Backpropagation Learning

- BP Learning with incremental or stochastic gradient descent

x_{ji} − ith input to unit j
w_{ji} − weight associated with the ith input to unit j
$net_j = \sum_i w_{ji} x_{ji}$ − weighted sum of inputs for unit j
o_j − the output computed by unit j
d_j − the target or desired output for unit j
σ − the sigmoid function
$ouputs$ − the set of units in the final layer of the network
$Downstream(j)$ − the set of units whose immediate inputs include the output of unit j

- Stochastic gradient descent iterates through the training examples one at a time, and the weight is updated by adding to it Δw_{ji} for each example

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

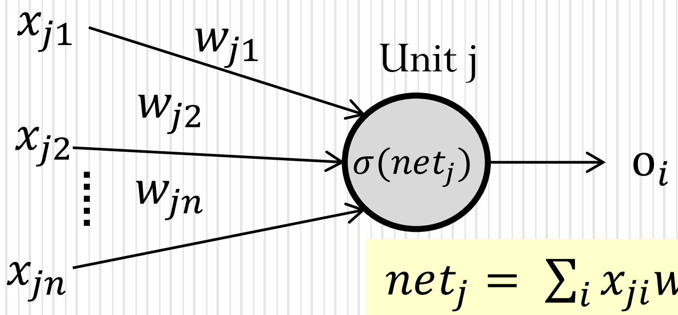
$$E = \frac{1}{2} \sum_{k \in outputs} (d_k - o_k)^2$$

- where E is the error on the training example, summed over all output units in the network

Backpropagation Learning

- For stochastic gradient we need an expression for $\frac{\partial E}{w_{ji}}$
- the weight w_{ji} can influence the rest of the network only through net_j, therefore we can use the chain rule to write

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



$$net_j = \sum_i x_{ji} w_{ji}$$
$$o_j = \sigma(net_j)$$

- now we need to derive expression for $\frac{\partial E}{\partial net_j}$
- two cases: unit j is an output unit and unit j is internal hidden unit

Backpropagation Learning

- Training rule for output unit weights

$$\frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial o_j} . \frac{\partial o_j}{\partial net_j} = (o_k - d_k). f'(net_k)$$

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in ouputs} (d_j - o_j)^2$$

(the derivatives $\frac{\partial}{\partial o_j}(d_j - o_j)^2$ will be zero for all output units k except where k =j)

$$\therefore \frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} (d_k - o_k)^2 = -(o_j - d_j)$$

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

which is a derivative of sigmoid function and is given as $\sigma(net_j)(1 - (net_j))$

$$\therefore \frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j(1 - o_j)$$

$$\boxed{\frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial o_j} . \frac{\partial o_j}{\partial net_j} = -(d_j - o_j) o_j (1 - o_j)}$$

Backpropagation Learning

- Training rule for output unit weights

$$\frac{\partial E}{\partial net_j} = \frac{\partial E}{\partial o_j} . \frac{\partial o_j}{\partial net_j} = -(d_j - o_j)o_j(1 - o_j)$$

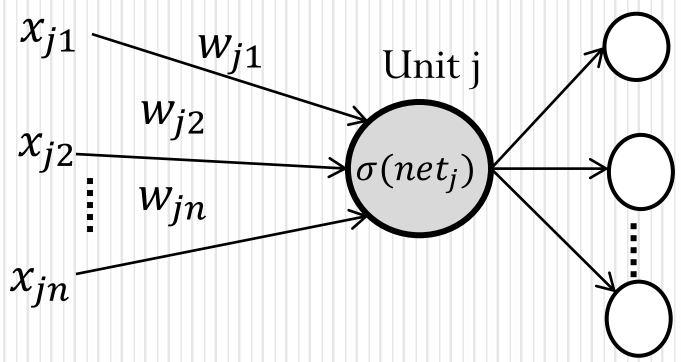
$$let\ \delta_j = -\frac{\partial E}{\partial net_j}$$

$$\boxed{\Delta w_{ji} = -\eta \frac{\partial E}{w_{ji}} = -\eta \frac{\partial E}{net_j} x_{ji} = \eta(d_j - o_j)o_j(1 - o_j)x_{ji}}$$

Backpropagation Learning

- Training rule for hidden unit weights
  - net_j can influence the network outputs (and therefore E) only through the units in the $Downstreams(j)$

$$\frac{\partial E}{\partial net_j} = \sum_{k \in Downstreams(j)} \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial net_j}$$
$$= \sum_k -\delta_k \frac{\partial net_k}{\partial net_j} = \sum_k -\delta_k \frac{\partial net_k}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$
$$= \sum_{k \in Downstreams(j)} -\delta_k \, w_{kj} o_j (1 - o_j)$$



Rearranging terms and using $\delta_j = -\frac{\partial E}{\partial net_j}$, we get

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

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

Example

- Consider a two-layer feedforward ANN with two inputs a and b, one hidden unit c and one output unit d. This network has five weights (w_{ca}, w_{cb}, w_{c0}, w_{d0}, w_{dc}), where w_{x0} represents the threshold weight for unit x. Initialize these weights to the values (0.1, 0.1, 0.1,0.1,0.1), then give their values after each of the first two training iterations of Backpropagation algorithm. Assume learning rate to be 0.3, momentum $\alpha = 0.9$, incremental weight updates, and the following training examples:

A	B	D
1	0	1
0	1	0

Evaluation and Hypothesis selection

- learn a hypothesis that fits the *future data best*
- *Stationarity assumption* on data: the probability distribution over examples remains stationary over time
  - Each example data point (before we see it) is a random variable E_j whose observed value $e_j = (x_j, y_j)$ is sampled from that distribution, and is independent of previous examples

$$\mathbf{P}(E_j | E_{j-1}, E_{j-2}, \ldots) = \mathbf{P}(E_j)$$

  - Each example has an identical prior probability distribution

$$\mathbf{P}(E_j) = \mathbf{P}(E_{j-1}) = \mathbf{P}(E_{j-2}) = \cdots$$

  - Examples that satisfy these assumptions are called independent and identically distributed (i.i.d.)

Evaluation and Hypothesis selection

- Error rate of a hypothesis: proportion of mistakes it makes – proportion of times that $h(x) \neq y$ for an (x, y)
- The hypothesis evaluated by testing it on a set of examples it has not seen (test set)
  - holdout cross-validation
    - randomly split the available data into a training set from which the learning algorithm produces h and a test set on which the accuracy of h is evaluated
    - does not use all available data
  - k-fold cross-validation
    - each example serves as training data as well as test data ,
    - data split into k equal subsets then k rounds of learning is performed
    - each round 1/k of the data is held out as a test set and the remaining examples are used as training data
    - typical values of k = 5 or 10, when k = n it is called as **leave-one-out cross-validation**

Evaluation and Hypothesis selection

- In order to avoid “Peeking” at the test data, the data set is divided into three sets
  - training set, validation set, and test set
- training dataset can be
  - used initially to fit the model parameters (e.g. weights of connections between neurons in artificial neural networks)
- Validation dataset can be used
  - after obtaining the fitted model to predict the output and to get an unbiased evaluation of a model fit on the training dataset, based on the evaluation it can be used to tune the model’s hyperparameters (e.g. the number of hidden units in a neural network).
  - for regularization (in time) by stopping the training when the error on the validation dataset increases and in turn to avoid overfitting
- Finally, test dataset is used to provide an unbiased evaluation of the final model

What did we discuss in L15-18?

- What is learning and what are the various forms of learning?
- Why is learning required?
- What is learning in terms of agent design?
- What is learning from example – supervised learning?
- We then moved to one non-linear learning approach Artificial Neural Network and discussed the following
  - Biological and Artificial Neural Network
  - Mc-culloch and Pitts Model
  - Perceptron
  - ANN: activation function, interconnections, and learning rules
  - Interconnection: feeforward and feedback architecture
  - Learning rules: Perceptron learning, Delta rule, Widrow-Hoff, Hebbian, and competitive rule learning
  - Backpropagation learning in multilayer feedforward network
- How to evaluate and select Hypothesis?