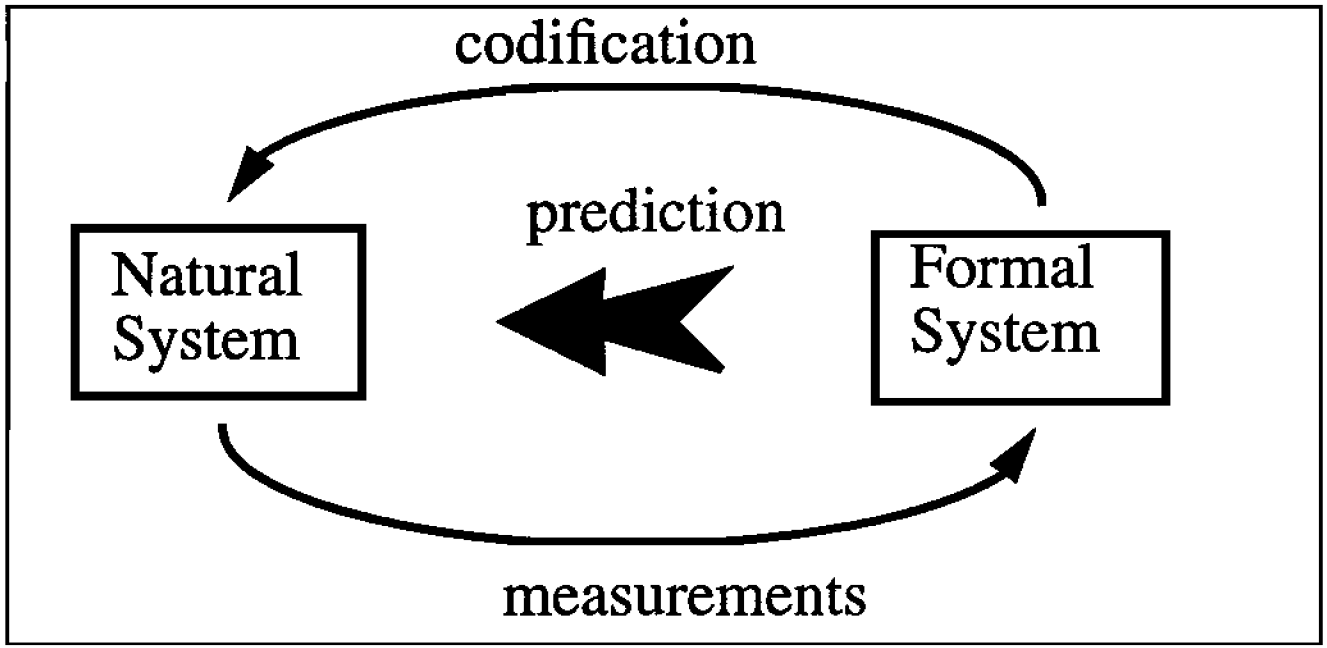
Building Experimental models



Methods: Deduction

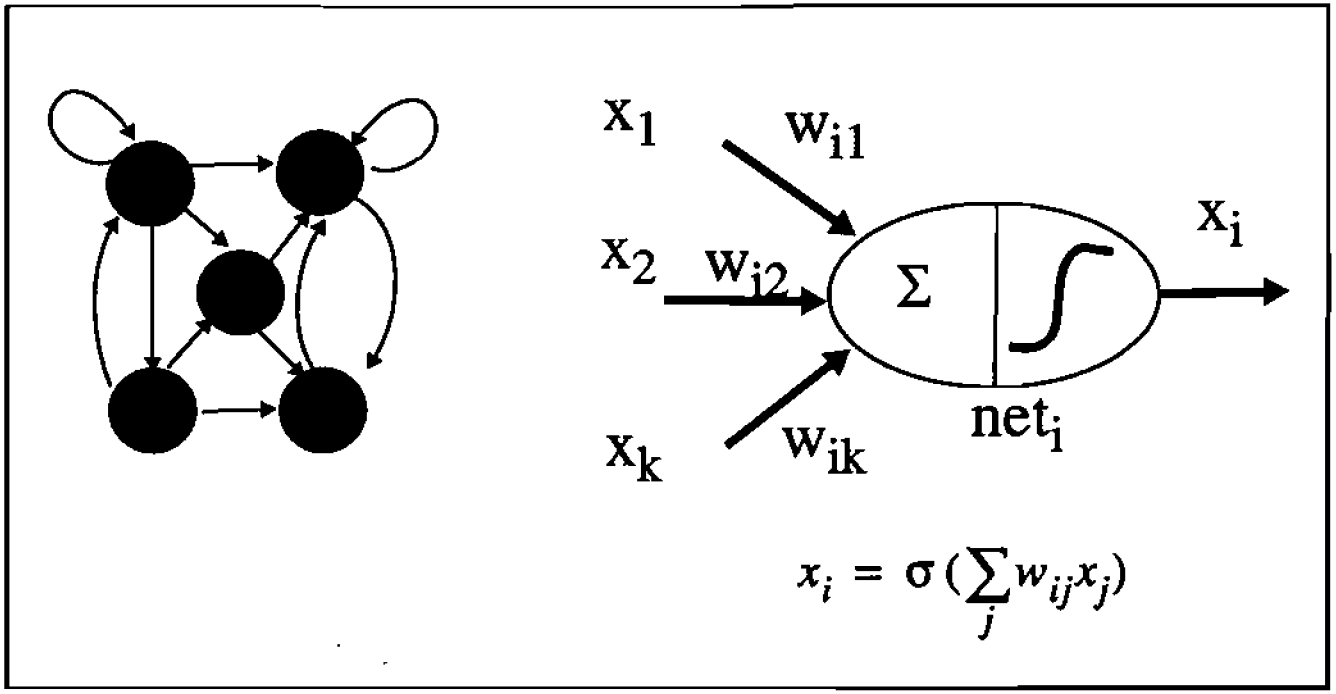
Induction

Information Systems

Information Input	Structured	Unstructured
Numeric	Mathematics Statistics	Artificial Neural Networks
Symbolic	Artificial Intelligence	?

Alternate Computers: The connectionist approach

Artificial neural networks are machines created from sets of simple, normally nonlinear processing elements (PEs), which are highly interconnected by means of adaptive weights.

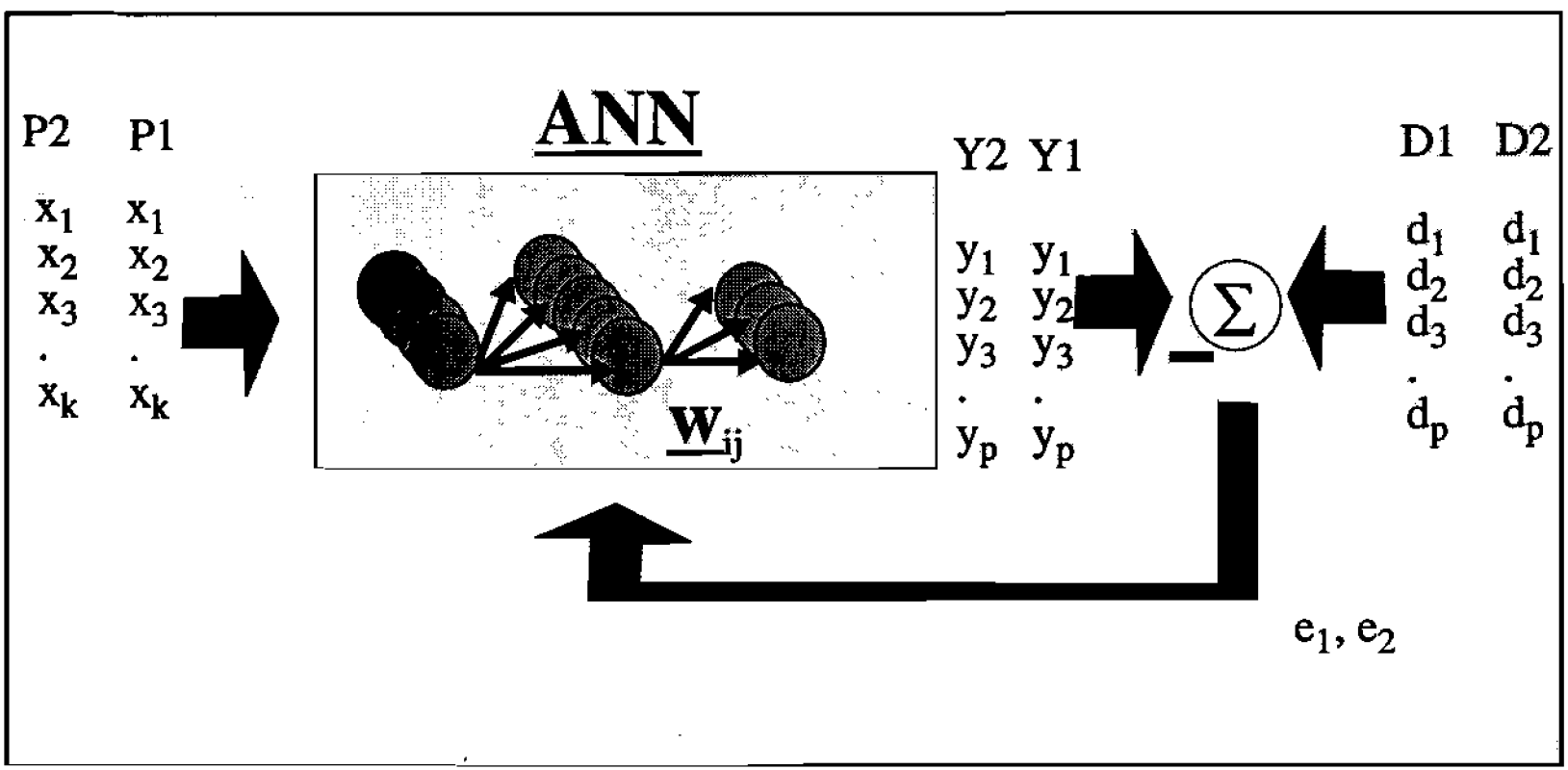


The Style of Neurocomputation

Get training and desired data.

Choose Topology

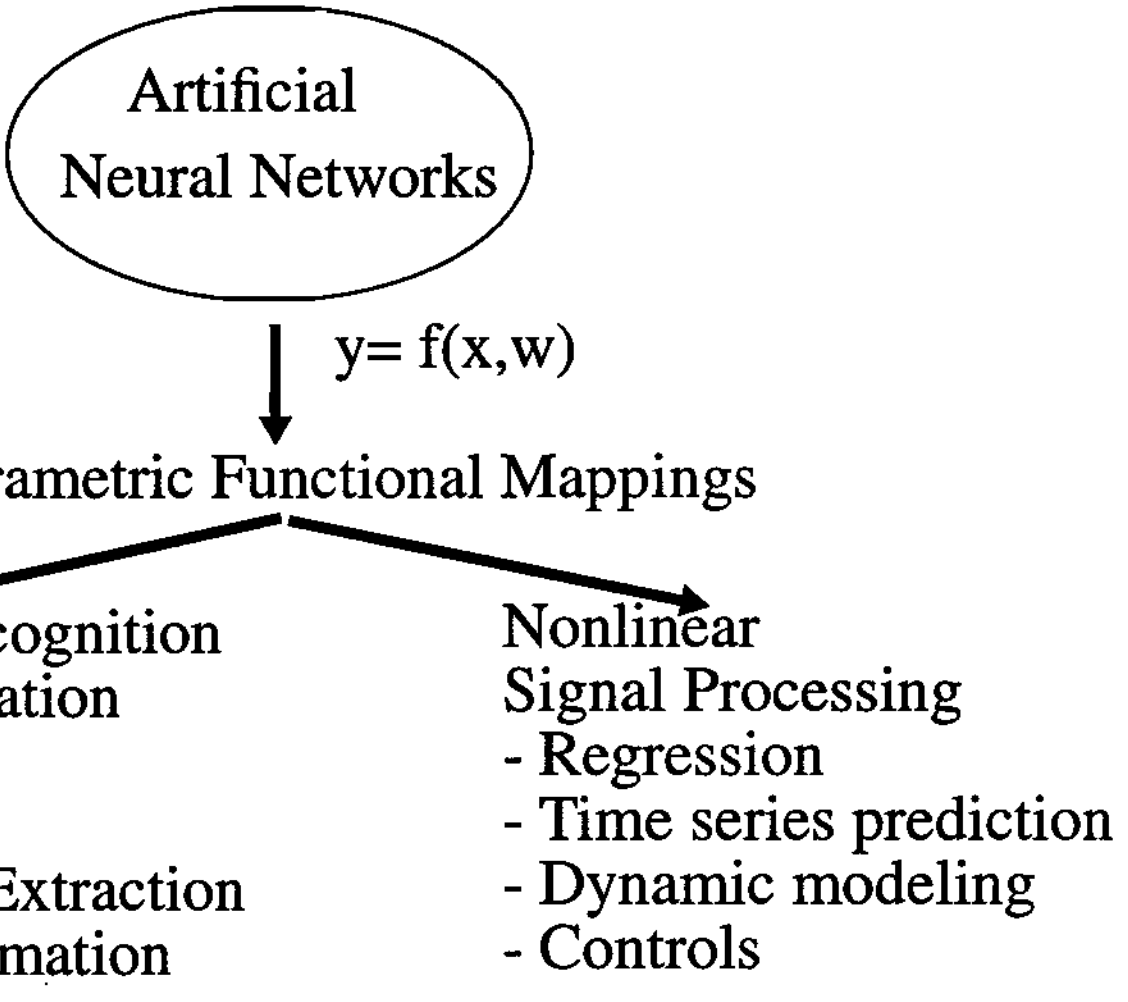
Apply Training rules.



Artificial Neural Networks

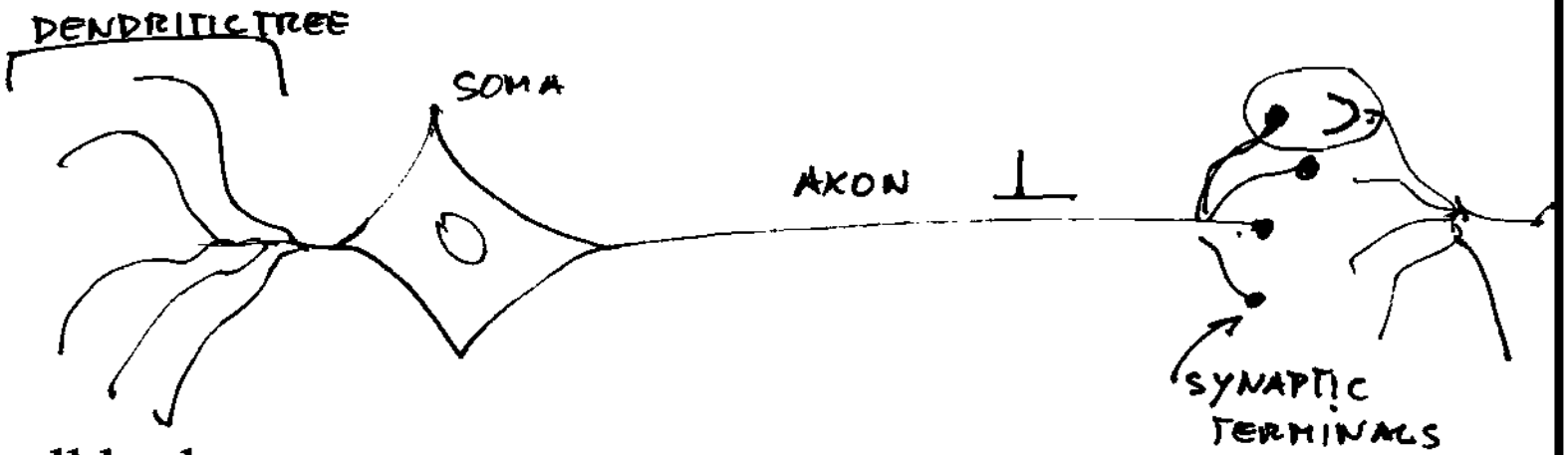
Training / PE and topology	Supervised	Unsupervised
Linear and layered	Linear Regression Linear filters	PCA Kohonen
Nonlinear and layered	Percpetron Multilayer perceptrons (MLPs)	MLP
Nonlinear and recurrent	Recurrent nets time lagged nets	Hopfield

Applications



Biological Neurons

Human brain contains 10^{10} nerve cells called neurons



Soma- cell body

Axon - output. Fiber connected to the soma, which connects to other neurons through synapses.

Dendrite - inputs. Highly branching tree of fibers (10^3-10^4 per neuron).

Synapse - Contacts between axon and dendrites. They modulate the potentials in the dendrites.

How it works: Soma receives electric inputs from other neurons through the dendritic tree. When the potential reaches a threshold the cell fires, and an action potential propagates along the axon and is transmitted through the synapses to other neurons.

The firing rate of a neuron is at most 300 pulses a second. The transmission is digital (all or nothing). But at the dendritic tree the potentials are analog.

Neurons are amazingly efficient in terms of power consumption. The all brain uses ~25 W of power .....

There are between 25-50 different types of neurons. The architecture of the brain is also very different among brain areas.

It is believed that the wiring dictates the function.

Artificial Neural Network models

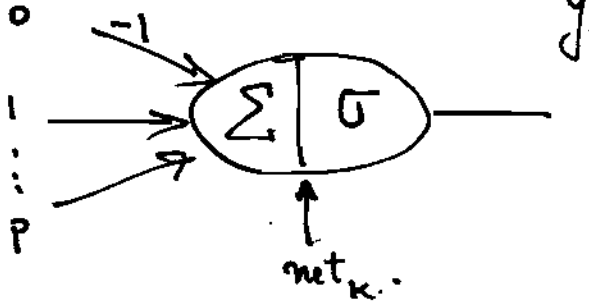
McCulloch-Pitts Model

Modelled biological neuron by a multi-input nonlinear device with weighted interconnections w_{kj}.

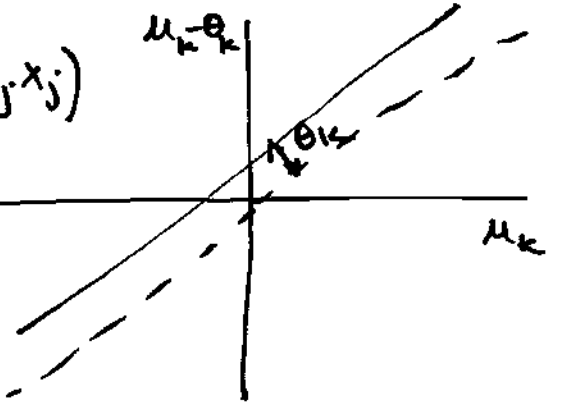
$$u_k = \sum_{j=1}^{p} w_{kj} x_j$$

$$y_k = \sigma(u_k - \theta_k)$$

Dendritic tree sums up contributions, Soma is represented by a non-linearity (threshold). Nonlinearity was a hard limiter. Output was 1 if the input was above threshold (otherwise was zero). Synaptic weights are fixed.



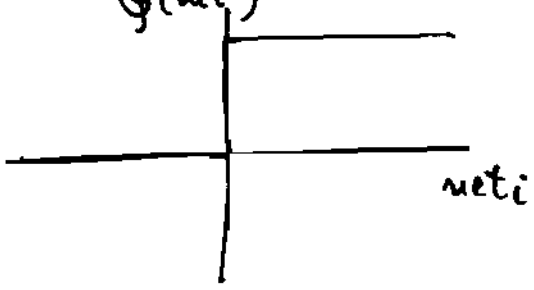
$$y_k = \sigma\left(\sum_{j=0}^{p} w_{kj} x_j\right)$$



Other possible nonlinearities

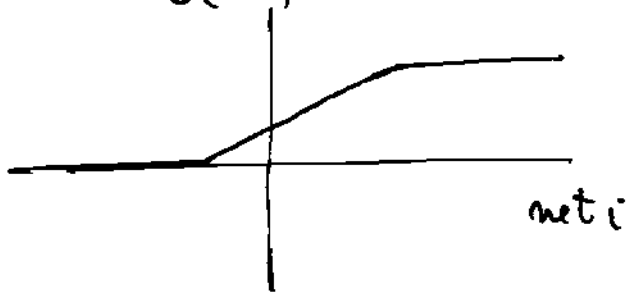
THRESHOLD

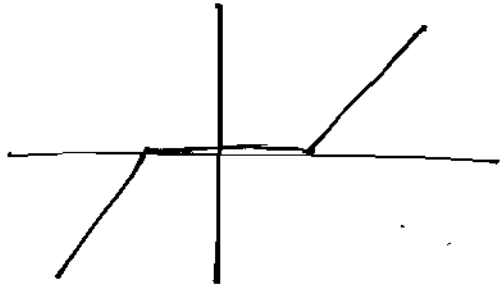
$\varphi(net)$



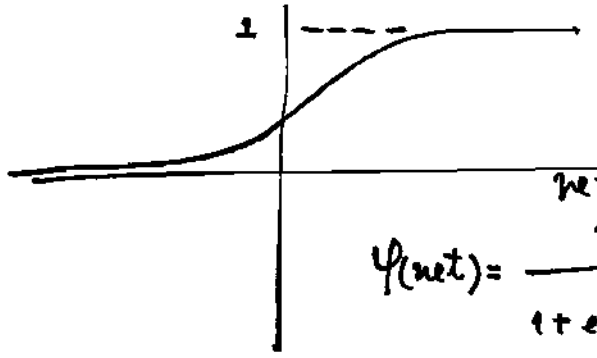
PIECEWISE LINEAR

$\sigma(net)$



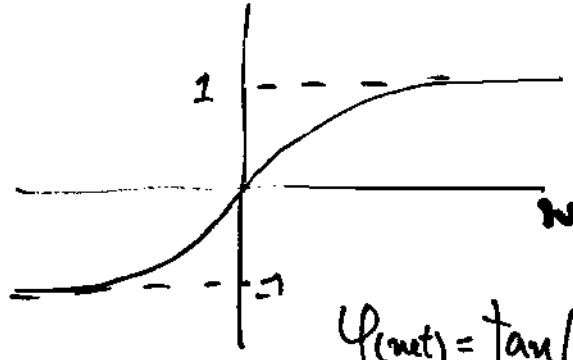


SIGMOID **LOGISTIC**



$$\varphi(net) = \frac{1}{1+\exp(-a\,net)}$$

HIPERBOLIC TANGENT



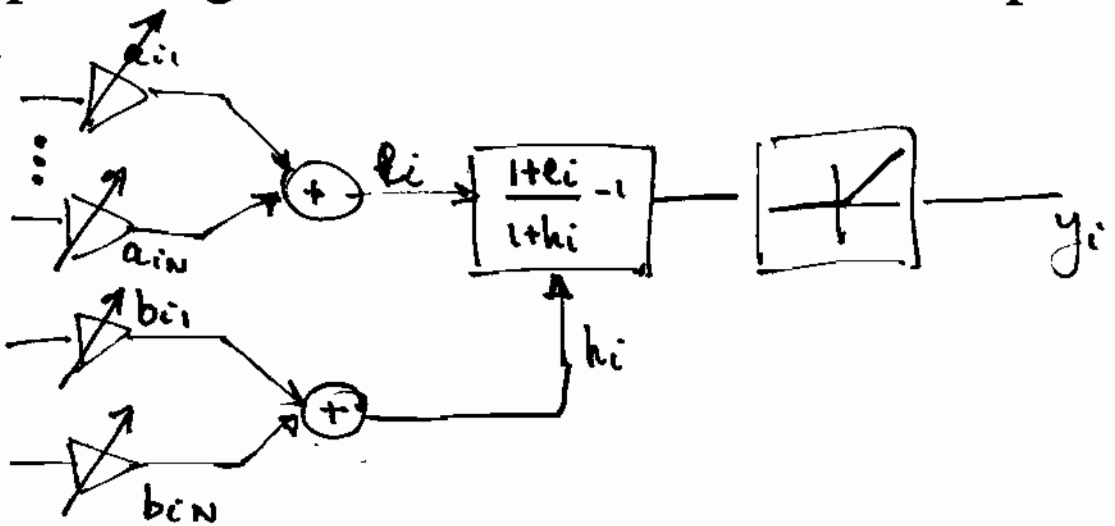
$$\varphi(net) = \tanh\left(\frac{net}{2}\right) = \frac{1-\exp(-net)}{1+\exp(-net)}$$

Fukushima Model

Fukushima proposed a different implementation. He separated the excitatory and the inhibitory inputs, combining them in such that they suppress each other

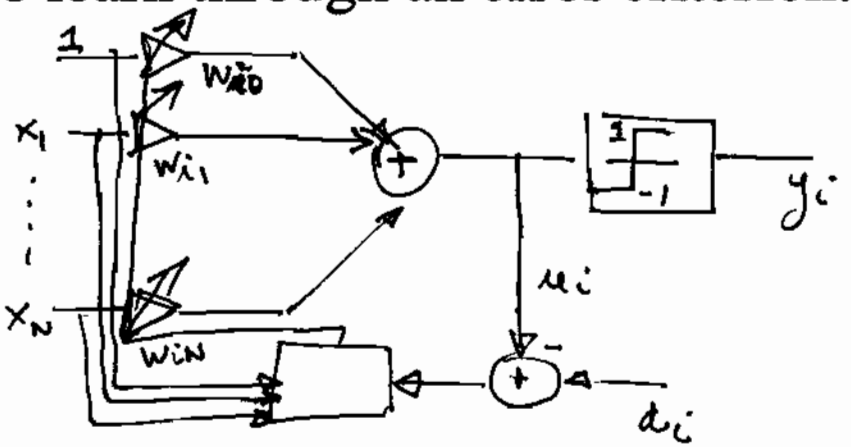
$$y_i = \sigma\left[\frac{1+\sum_{j=1}^{N} a_{ij}x_j}{1+\sum_{j=1}^{N} b_{ij}x_j} - 1\right]; \quad \sigma(a_i) = \begin{cases} a_i & a_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The synaptic weights are variable, and can be adapted using Hebbian learning.



Adaline

The adaline (adaptive linear network) is a linear neuron that has the ability to learn through an error criterion.

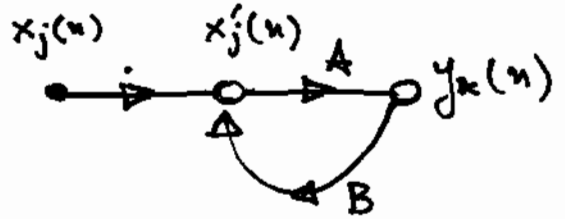


The adaline has n+1 inputs, and auxiliary output (the desired signal) and two outputs: A linear one that is fedback and a nonlinear as output. It has a bias term. The coefficients are modified such the mean square error between the desired and the linear output is minimized.

The adaptation mechanism proposed by Widrow and Hoff is amazingly simple.

Neurons with feedback

Biology uses and abuses of feedback (sometimes positive...).



$$y_k(n) = \frac{A}{1-AB} x_j(n)$$

$$\begin{cases} B = z^{-1} \\ A = w \end{cases}$$

$$\frac{A}{1-AB} = \frac{w}{1-wz^{-1}} = w \sum_{l=0}^{\infty} w^l z^{-l}$$

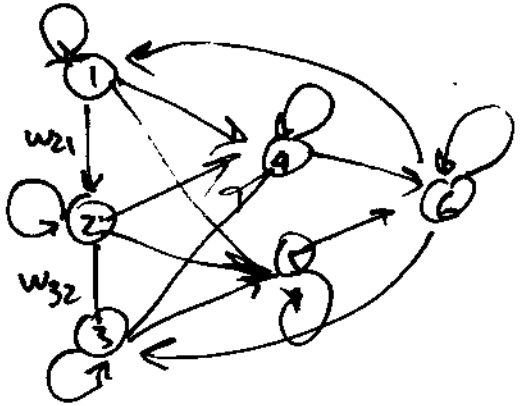
$|w| < 1 \rightarrow$ CONVERGENT

$|w| \geq 1 \rightarrow$ DIVERGENT

One of the applications of feedback is as a short-term memory mechanism (recency gradients).

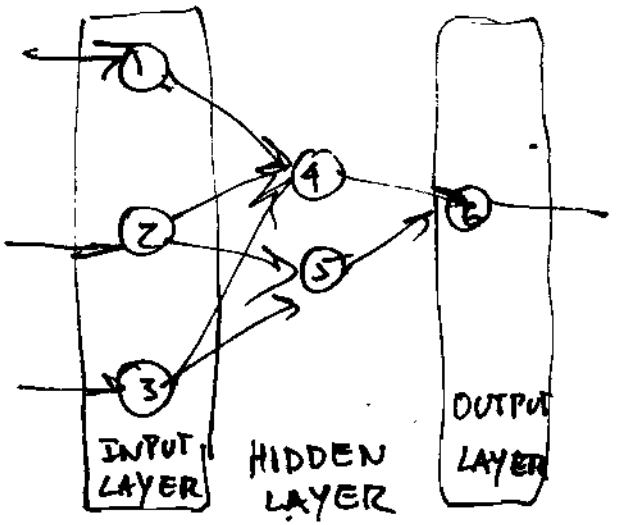
Topologies

Fully connected networks



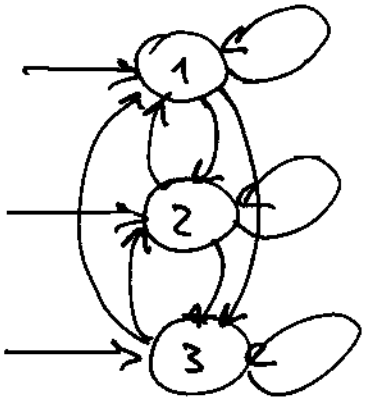
$$\begin{bmatrix} w_{11} & \cdots & w_{61} \\ \vdots & & \vdots \\ w_{16} & & w_{66} \end{bmatrix}$$

Special cases: Layered networks are very popular



$$\begin{bmatrix} 0 & 0 & 0 & w_{41} & w_{51} & 0 \\ & & & w_{42} & w_{52} & 0 \\ 1 & & 1 & w_{43} & w_{53} & 0 \\ 1 & & 1 & 0 & 0 & w_{64} \\ 1 & & 1 & 0 & 0 & \\ 0 & & 0 & 0 & 0 & w_{65} \\ & & & & & 0 \end{bmatrix}$$

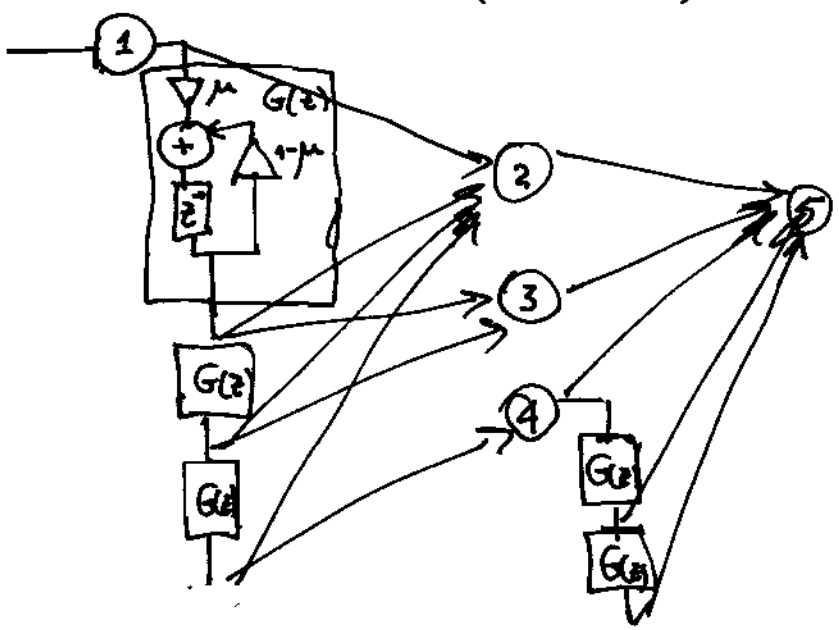
One layer recurrent networks



Time lagged recurrent networks(TLRNs)

G(z)

LOCALLY RECURRENT MEMORY STRUCTURE

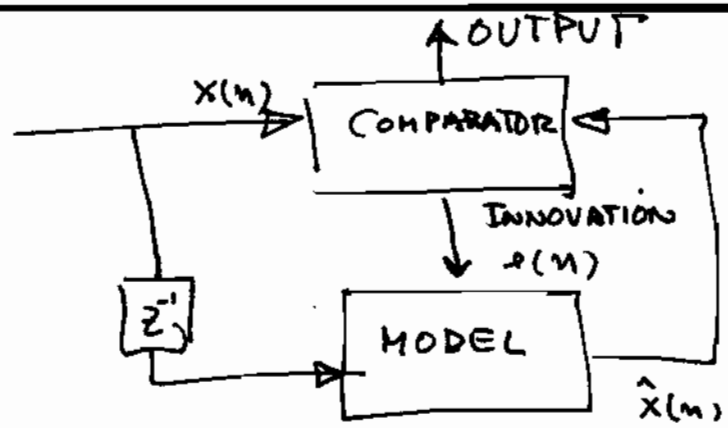


Adaptation and Learning

Learning has spatial and temporal aspects. In stationary environments, the temporal aspect can be forgotten. We can gather all the information apriori in a training set, and use supervised learning to train the free parameters of the learning system. This is the common way to use ANNs.

However, in nonstationary environments, we can not get a priori all the conditions. So the system or some of its parts must be adapted during learning. For this unsupervised methods seem our only hope.

In control theory this was understood a long time ago (Kahlman). The idea then was to find how badly was the system operating, and then correct the operating point. Predictors were utilized



to generate a correcting signal (innovation) that would change the operating point of the system through an additive gain term. This idea gives rise to systems that are continually adapting, and unfortunately is seldom applied in neurocomputing. Unsupervised learning does the same job.

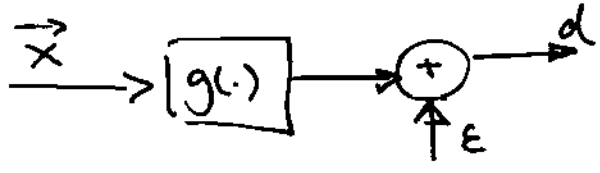
$$\begin{pmatrix}\text{STATE}\\ \text{ESTIMATE}\end{pmatrix} = \begin{pmatrix}\text{PREVIOUS}\\ \text{STATE ESTIMATE}\end{pmatrix} + \text{GAIN} \cdot (\text{INNOVATION})$$

Statistical Nature of Learning

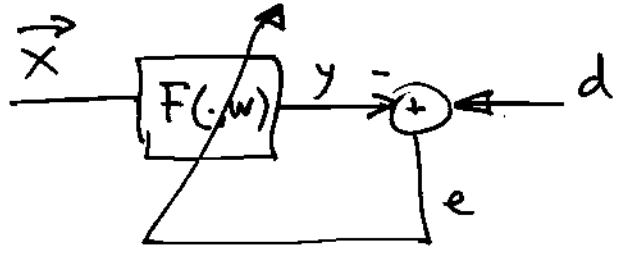
Learning is intrinsically stochastic. Not only due to the environment but also due to the algorithms of learning.

Suppose we observe the environment and we collect signals x (independent variables). We also know that there is a dependent variable d associated with the phenomenon we are observing. We normally do not have a precise knowledge of the relation of d and x

$$d = g(x) + \varepsilon$$



$$\begin{cases} E[\varepsilon | x] = 0 \\ E[\varepsilon g(x)] = 0 \end{cases}$$



However, we know that e is uncorrelated with g() and x. Now let us use a physical system (ANN) to "learn" the unknown function g().

We will approximate it by $y = F(x,w)$

We can adapt the free parameters of F by minimizing the MSE

$$J(w) = \frac{1}{2} E[e^2] = E[(d - F(x,w))^2]$$

and we can write

$$J(w) = \frac{1}{2} E[(d - g(x))^2] + \frac{1}{2} E[(g(x) - F(x,w))^2]$$

Now note that the first term is independent of the free parameters w of the model, so the minimum is obtained by minimizing

$$\min_w \{ E[(g(x) - F(x,w))^2] \} = \min_w \{ E[d|x] - F(x,w))^2 \}$$

We can also show that the regressive model is the best approximation in the LS sense of d

$$J(w) \geq \frac{1}{2} E[(d - E[d|x])^2]$$

Another interesting problem is how the training set (the choices of x) affect our model F. Writing $D = \{(x_1,d_1), \ldots (x_N,d_N)\}$

$$E_D[(E[d|x] - F(x,D))^2] =$$

$$= (E_D[F(x,D)] - E[d|x])^2 + E_D[(F(x,D) - E_D[F(x,D)])^2]$$

we can see that:

1. The term $E_D[\bar{F}(x,D) - E[d|x]]$

represent s the BIAS of the estimator.

2. The term $E_D[(F(x,D) - E_D[F(x,D)])^2]$

represents the variance of the estimator.

To get a good estimator both the variance and bias must be small. ANNs learn from example so they can have small bias (finite set of examples, so can reduce the error practically to zero); however we can not control the variance well (it would need an infinite set of examples). This bias/variance dilemma is present in any learning system.

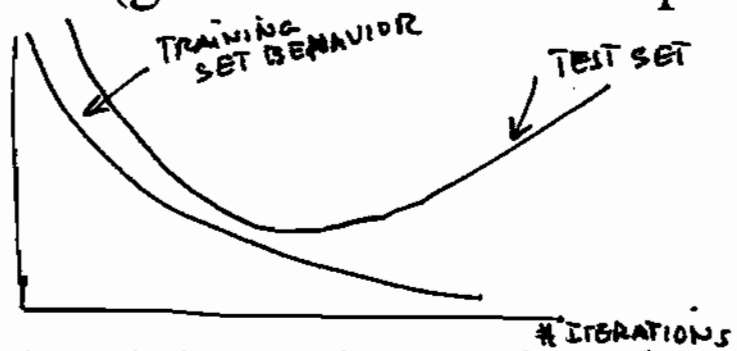
We can control the variance if we purposefully allow bias (i.e. constraint the architecture, or preprocessing).

This is the purpose of the theory of learning: to define the optimal

conditions of the learning system given the input data.

The book presents a summary of very difficult (because abstract) concepts of the learning theory developed by Vapnik.

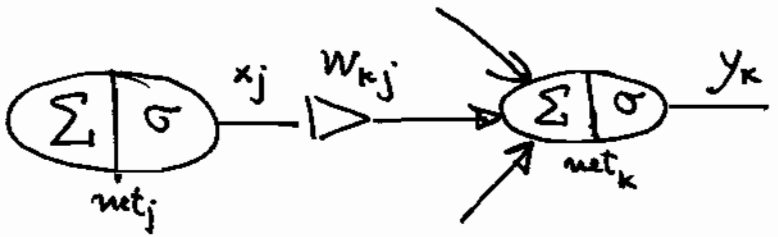
For a long time it was known that the number of training iterations and the relation size of net/training set affected the generalization of learning systems (generalization is the performance on the training set).



Vapnik formulated these observations by using the theory of empirical risk minimization. Basically he found that there is a dimension (Vapnik-Chervonenkis dimension) of the learning system (# degrees of freedom) below which we can specify an upper bound for the generalization error.

Learning Processes

Definition: Process by which free parameters are adapted through an interaction with the environment.

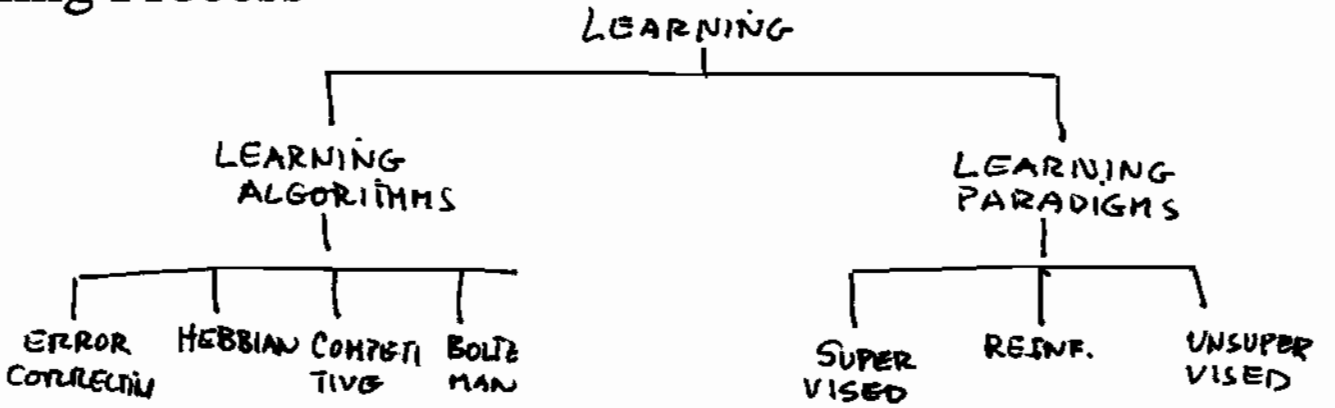


$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

A learning algorithm is the rule used to update the weights.

Learning processes can be studied looking at the algorithms of learning and at the form of interaction with the environment (learning paradigms).

Learning Process

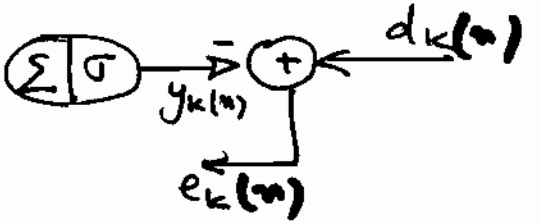


Learning Rules: error correction learning

This is the learning rule found in adaptive signal processing. It is highly useful for engineering applications because normally we know what the desired response is.

Assume $d_k(n)$ is known (target response for PE k), The actual response is $y_k(n)$. Normally we can expect a nonzero difference

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



The ultimate purpose is to minimize this difference with respect to some norm (error criterion). Commonly mean square error is used, because it gives equations easy to solve.

$$J = E\left[\frac{1}{2}\sum_k e_k^2(n)\right] \Rightarrow \simeq \frac{1}{2}\sum_{\forall k} e_k^2(n)$$

E is a statistical operator (expectation), and sum is over all the output PEs.

One can minimize J with respect to w using the method of gradient descent.

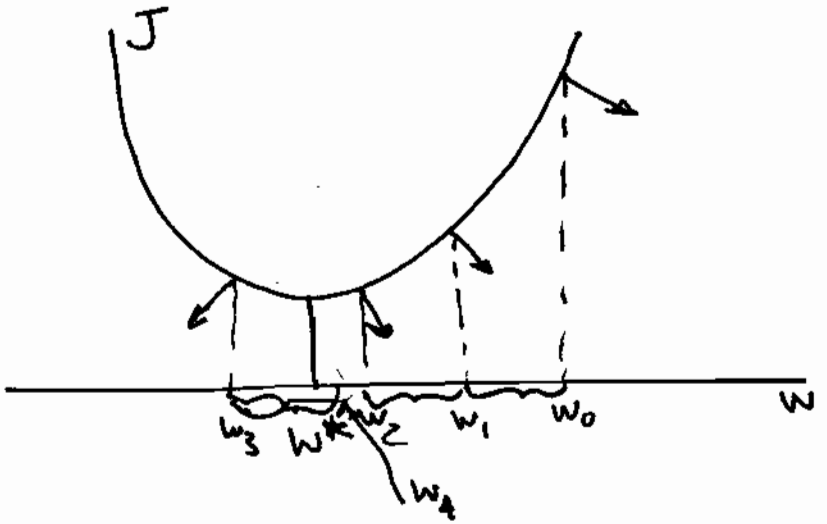
Normally we substitute the E[] by its instantaneous value and let the "noise" be reduced through the iterative presentation of patterns. This yields the LMS algorithm

$$\Delta w_{kj}(n) = \eta\, e_k(n)\, x_j(n)$$

η is he learning rate parameter and affects the speed (and stability) of learning.

When the net is linear, and the network feedforward (as in the tap delay line), we can show that there is a single minimum of the performance surface. When the net is nonlinear this can not be guaranteed.

Since gradient descent only uses the local information, it can be trapped in local minima.



Hebbian learning

Hebb's postulated in 1949 a far reaching principle that he observed in the nervous system. "when two neurons repeatedly excited each other, it was easier to induce a response in one of them by the other".

We can translate mathematically this observation in different forms

$$\Delta w_{kj}(n) = F(y_k(n), x_j(n))$$

The ingredients are that the process is time dependent, local, interactive, and involved conjunction (correlation). Normally we use the form

$$\Delta w_{kj}(n) = \eta\, y_k(n)\, x_j(n)$$

which emphasizes the correlation nature of Hebbian learning (linear). Notice also that if nothing else is done, the weight will increase exponentially.

Normally we impose a nonlinear forgetting factor, function of the weight and activity

$$\Delta w_{kj}(n) = \alpha\, y_k(n)\left[c\, x_j(n) - w_{kj}(n)\right]$$

This is called the generalized activity product rule, but others are possible (a simple threshold may suffice in some cases). Covariance learning also works well and has links to DSP.

Notice that the Hebbian learning principle can be applied to both unsupervised or supervised methods.

Competitive learning

Here the learning principle is very different. Processing elements compete among themselves for being active. Only one PE ends up active (winner take all). This rule has also been shown to be biological plausible, and one of the principles of self-organization.

There are three basic elements in competitive learning:

An unspecified asymmetry (due to different initial conditions, or noise). A limit in the PE response. And a mechanism to create competition. One such rule is

$$\Delta w_{ji} = \begin{cases} \eta (x_i - w_{ji}) & \text{IF } PE_j \text{ wins} \\ 0 & \text{IF } PE_j \text{ loses} \end{cases} \qquad \sum_i w_{ji} = 1 \quad \forall j$$

This rule has the net effect of making the weight vector approach the input. A simple net to implement this rule has lateral inhibitory connections between the output PEs.

Boltzman Learning

The principles here are once again very different. The PE only has 2 states (1/-1), and is assumed randomly switching between these two states, according to a given pdf (Boltzman distribution).

$$W_{j+/-} = \frac{1}{1 + e^{-\frac{\Delta E_j}{T}}}$$

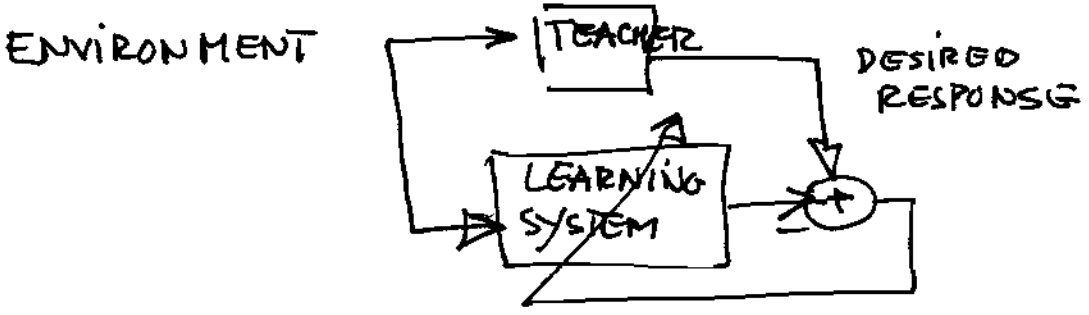
There are two types of functional groups the visible units and the

hidden units. The visible units can operate on the clamped mode (i.e. with the inputs applied). There is also a free running condition, where no inputs are applied. The difference between the correlation between two neurons in the clamped and free running condition can be used to modify the network weights.

Supervised learning

This learning paradigm requires the existence of a desired or target response. We can think that there is a teacher during learning. The teacher presents to the learning machine pairs of input-outputs, which embody the knowledge about the unknown environment.

From this, the learning machine can derive an error, that can be used to modify the internal parameters, such that it will mimic the teacher. Error correcting algorithms are the most common.



The most famous supervised learning rule is the backpropagation algorithm (BP). LMS is a special case of BP. Supervised learning is

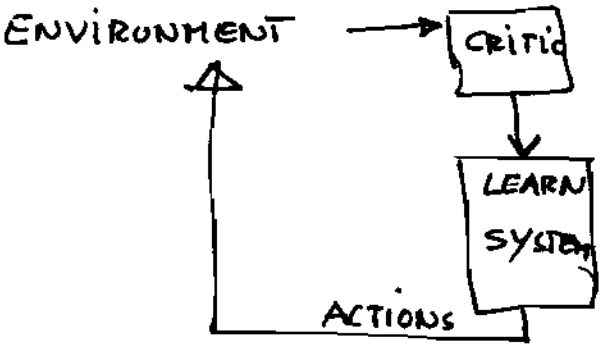
slow and scales worse than linear with the number of free parameters. So it seems that it has its intrinsic limitations.

Reinforcement Learning

In reinforcement learning the system learns by doing things. It is also the learning of an input-output map, but through the maximization of a scalar performance index (like a global yes or no answer). There are actions and reinforcement after every action. The problem is that the system does not know explicitly why (structural credit assignment) and when (temporal credit assignment) did the right thing. The discovery of this policy becomes the purpose of learning. In so doing the input to output map is learned.

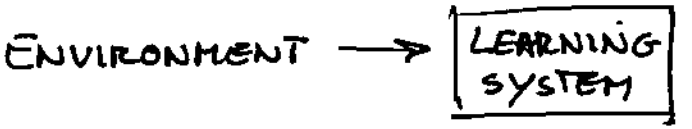
This is an highly interesting learning paradigm that is not yet fully understood. The adaptive heuristic critic is one of the best examples

of this learning paradigm.



Unsupervised learning

In unsupervised learning the system extracts knowledge directly from the environment. So only if there is redundancy in it the system will learn. Internally the learning system has some constraint that specifies (or rates) the quality of the internal representations. Learning proceeds to optimize this quality measure.



List of learning tasks:

approximation, association, pattern classification, prediction, control,

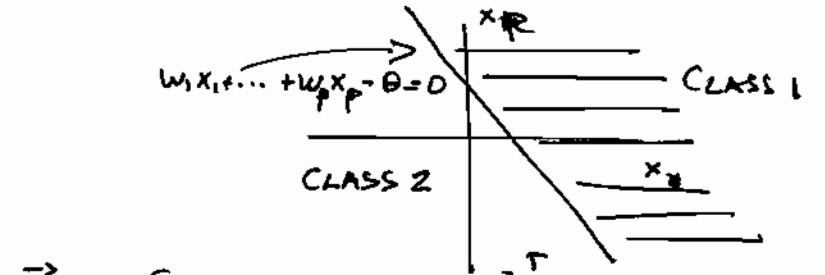
Perceptron

Is one of the earliest neural networks developed by Rosenblatt (58). The perceptron is nothing but a network built from McCulloch-Pitts PEs (one per class). However, unlike the work of M-P, the perceptron has ADAPTIVE weights.

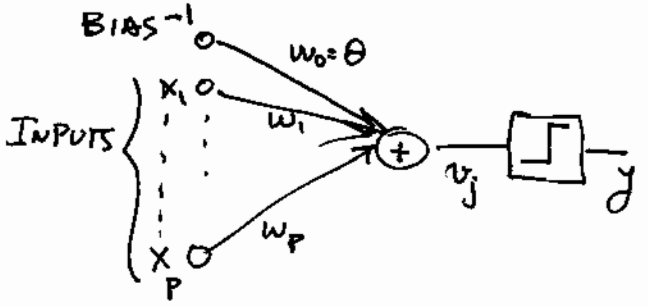
The perceptron was used as a pattern recognition device. It was able to recognize letter presented in a retina. It was insensitive to distortions and rotations of the characters (to a certain extent), what was considered exceptional at the time.

THe importance of the perceptron was the existence of the perceptron convergence theorem, which guaranteed convergence in a finite number of steps, provided the input patterns were linearly separable (the only ones that the machine could classify).

The perceptron PE



Let us write it as



$$\vec{x}(n) = [-1, x_1(n) \ldots x_p(n)]^T$$

$$\vec{w}(n) = [\theta(n), w_1(n), \ldots w_p(n)]^T$$

$$v(n) = \vec{w}^T(n)\vec{x}(n)$$

Note that this device is able to locate a hyperplane in the input space.

The perceptron convergence theorem has a very simple statement:

Present a pattern. If the output is correct do nothing. If the output is wrong and pattern is class 1, go to the elements that are active, and increase their weights until the correct response is obtained. Decrease the weights for class 2 under the same conditions. You only need to do this a FINITE number of times to classify all the patterns (if the problem is linearly separable).

Mathematically this means:

1. If correct $w(n+1) = w(n)$

$$\begin{cases} w^T(n)\,x(n) \geq 0 & x(n) \in C_1 \\ w^T(n)\,x(n) < 0 & x(n) \in C_2 \end{cases}$$

2. If incorrect response

$$w(n+1) = w(n) - \eta(n)\,x(n) \qquad w^T(n)\,x(n) \geq 0 \qquad x(n) \in C_2$$

$$w(n+1) = w(n) + \eta(n)\,x(n) \qquad w^T(n)\,x(n) < 0 \qquad x(n) \in C_1$$

$\eta(n)$ → LEARNING RATE OR STEP SIZE PARAMETER

Here we will cover only the case of a constant step size. $\eta(n) = \eta > 0$

Proof:

Initial condition w(0)=0. Assume that $w^T(n)x(n) < 0$ is negative and pattern is class 1 (wrong). Then we have to increase the weights, $w(n+1) = w(n) + \eta\,x(n) = w(n) + x(n)$

Assume a solution w0, ($w_0^T x(n) > 0$)

$$\alpha = \min_{x(n) \in X_1} w_0^T x(n)$$

Then we get a positive constant

$$w_o^T w(n+1) = w_o^T [x_1 + \cdots + x(n)]$$

Now multiply by the row vector $w_o^T w(n+1) \geq n\alpha$

we conclude that there is a lower bound

which can be rewritten in terms of the Cauchy-Schwartz inequality

$$\|w(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|w_o\|^2}$$

But note that we can rewrite $w(n+1) = w(n) + x(n)$

as

$$\|w(k+1)\|^2 = \|w(k)\|^2 + \|x(k)\|^2 + 2w^T(k)x(k)$$

which means that

$$\|w(k+1)\|^2 - \|w(k)\|^2 \leq \|x(k)\|^2 \qquad k = 1, \cdots n$$

$$\|w(n+1)\|^2 \leq \sum_{k=1}^{n} \|x(k)\|^2 \leq n\beta$$

So there is a contradiction which can only be solved if

$$n_{max} = \frac{\beta \|w_o\|^2}{\alpha^2}$$

But this caps the number of iteration, and we proved the theorem.

This rule can be put in more recognizable form if we write it

$$y(n) = \text{sgn}\left(w^T(n)\,x(n)\right)$$

where

$$d(n) = \begin{cases} +1 & x(n) \in C_1 \\ -1 & x(n) \in C_2 \end{cases}$$

$$\vec{w}(n+1) = \vec{w}(n) + \eta\left(d(n) - y(n)\right)\vec{x}(n)$$

This means that the perceptron learning rule is a quantized form of error correction learning.

In fact Shynk showed that we can put it as an instantaneous estimate of a gradient descent procedure

$$J = -E[e(n)\,v(n)]$$

$$\begin{cases} \dfrac{\partial J(n)}{\partial w(n)} \simeq -\left[d(n) - y(n)\right]\dfrac{\partial v(n)}{\partial w(n)} \Rightarrow \boxed{\Delta w(n) = -\eta\,\nabla_w J(n)} \\ \dfrac{\partial v(n)}{\partial w(n)} = x(n) \end{cases}$$

Maximum Likelihood Gaussian Classifier

The perceptron is a very efficient implementation of a linear classifier, i.e. a system that does pattern recognition by linear discriminant

functions.

The book compares it with the maximum likelihood classifier, a method that treats the parameters as unknown but fixed. It defines the conditional p.d.f. (x observation, w unknown parameters) $f(\vec{x}|w_j)$

The MLE of $\vec{w}$ is the value that maximizes the likelihood $f(\vec{x}|\vec{w})$

So consider for a two class problem the observation $x = [x_1, x_2, \ldots x_p]^T$ with mean m and covariance C

$$\mu = E[x] \qquad C = E((x-\mu)(x-\mu)^T)$$

$$f(x) = \frac{1}{(2\pi)^{\frac{p}{2}}(\det C)^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^T C^{-1}(x-\mu)\right]$$

The log likelihood for the classes can be written

$$\text{CLASS 1}: \mu_1, C \qquad \ln f(x|C_i) = -\frac{p}{2}\ln(2\pi) - \frac{1}{2}\ln(\det C) - \frac{1}{2}x^T C^{-1}x + \mu_i C^{-1}x - \frac{1}{2}\mu_i^T C^{-1}\mu_i$$

$$\text{CLASS 2}: \mu_2, C \qquad \Longrightarrow \quad l_i(x) = \mu_i^T C^{-1}x - \frac{1}{2}\mu_i^T C^{-1}\mu_i$$

$$l = l_1(x) - l_2(x) = (\mu_1 - \mu_2)^T C^{-1}x - \frac{1}{2}(\mu_1^T C^{-1}\mu_1 - \mu_2^T C^{-1}\mu_2)$$

or

$$\ell = \hat{w}^T x - \hat{\theta} = \sum_{i=1}^{p} \hat{w}_i x_i - \hat{\theta}$$

where w is the maximum likelihood estimate of w defined by

$$\hat{w} = C^{-1}(\mu_1 - \mu_2)$$

$$\hat{\theta} = \frac{1}{2}(\mu_1^T C^{-1} \mu_1 - \mu_2^T C^{-1} \mu_2)$$

Notice that this approach requires the knowledge of the a priori p.d.f. (Gaussians), assumes equal priors for the classes, but they can overlap.

In the perceptron classes can not overlap (otherwise they are linearly separable), but one gets a classifier without no further assumptions.

Least Mean Square (LMS) Error Algorithm

This is probably one of the better known adaptation procedures, which became a standard. His inventor was Widrow (60), and it is also known as the delta rule or the Widrow-Hoff algorithm. It is the basis of adaptive linear filtering, and was one of the inspirations for backpropagation.

LMS is a steepest descent procedure that uses the present sample to estimate the gradient of the performance surface. How such a noisy estimate converges is amazing.... It is an unbiased estimate of the gradient and the noise is filtered out during the iterations.

Wiener-Hopf Optimal filtering

Suppose we have an input x, a set of weights w that form an output

$$y = \sum_{k=1}^{p} w_k x_k$$

and we want to find the weight vector W such that y is as close to a desired value d in the mean square error sense.

$e = d - y$ $\quad J = \frac{1}{2} E[e^2]$ $\quad E \rightarrow$ EXPECTATION OPERATOR

We can rewrite J

$$J = \frac{1}{2} E[d^2] - E\left[\sum_{k=1}^{p} w_k x_k d\right] + \frac{1}{2} E\left[\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k x_j x_k\right]$$

and call

$r_d = E[d^2]$ → MEAN SQUARE VALUE OF d

$r_{dx} = E[d\,x_k]$ $\quad k = 1, 2, \ldots p$ CROSSCORRELATION d, x_k

$r_x(j,k) = E[x_j x_k]$ $\quad j, k = 1, 2, \ldots p$ AUTOCORRELATION OF x

J is the error performance surface. $\quad J = \frac{1}{2} r_d - \sum_{k=1}^{p} w_k r_{dx}(k) + \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k r_x(j,k)$

J is minimized when its derivative is zero

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k} = 0$$

or

$$\nabla_{w_k} J = -r_{dx}(k) + \sum_{j=1}^{p} w_j r_x(j,k) = 0$$

which gives

$$\sum_{j=1}^{p} w_j r_x(j,k) = r_{dx}(k) \qquad [RW = P]$$

This is the Wiener -Hopf solution, and the filter the Wiener filter.

Steepest Descent

Instead of solving for the solution analytically, we can search the performance surface. Since it is a paraboloid it only has a single minimum.

We can use the direction of the gradient to orient the search, which gives

$$\Delta w_k(n) = -\eta \nabla_{w_k} J(n) \qquad k = 1, 2, \ldots p$$

$$w_k(n+1) = w_k(n) - \eta \nabla_{w_k} J(n)$$

Instead of the E operator we can use time averages for ergodic processes. $E \rightarrow \mathcal{E}$ $\quad \mathcal{E}_{TOTAL}(n) = \sum_{i=1}^{n} \mathcal{E}(i) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$

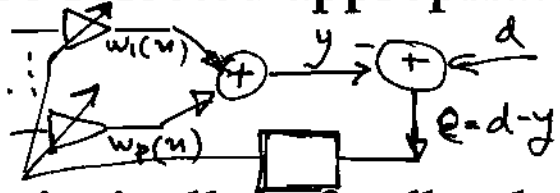
The LMS algorithm

Instead of using averages, the LMS algorithm uses the instantaneous values of the parameters to update the weights. This gives

$$\begin{cases} \hat{R}_x(j,k;n) = x_j(n)\cdot x_k(n) \\ \hat{R}_{dx}(k;n) = x_k(n)\cdot d(n) \end{cases}$$

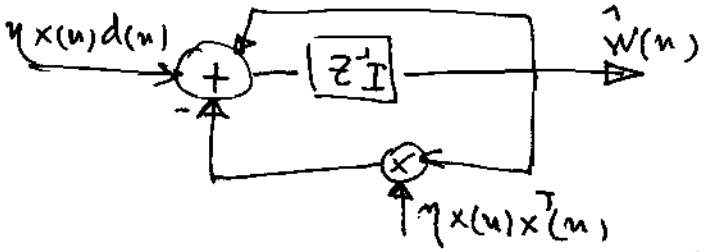
$$\hat{w}_k(n+1) = \hat{w}_k(n) + \eta\left[d(n) - y(n)\right]x_k(n) \quad k = 1, 2, \ldots p$$

Notice that only two multiplication per weight are necessary. The step size parameter must be selected appropriately otherwise it will diverge.



The LMS adaptation is intrinsically a feedback process

$$\hat{W}(n+1) = \left[I - \eta\, x(n)x^T(n)\right]W(n) + \eta\, x(n)d(n)$$



To have convergence in the mean we require that

$$0 < \eta < \frac{2}{\lambda_{MAX}}$$

λ_{MAX} → MAX EIGENVALUE OF R_x

To have convergence in the mean square we require that

$$0 < \eta < \frac{2}{\text{TOTAL INPUT POWER}}$$

Learning curve

It is the graph of J as a function of iteration.

$$M \simeq \eta \, \text{tr}[R]$$

Misadjustment is normalized excess MSE. The settling time is number of iteration needed for convergence. Misadjustment is proportional to step size and the settling time is inversely proportional. So there is a conflict.

The best way is to use a variable step size (annealing schedule). Start large and then reduce it. Moody and Darken proposed the following schedule

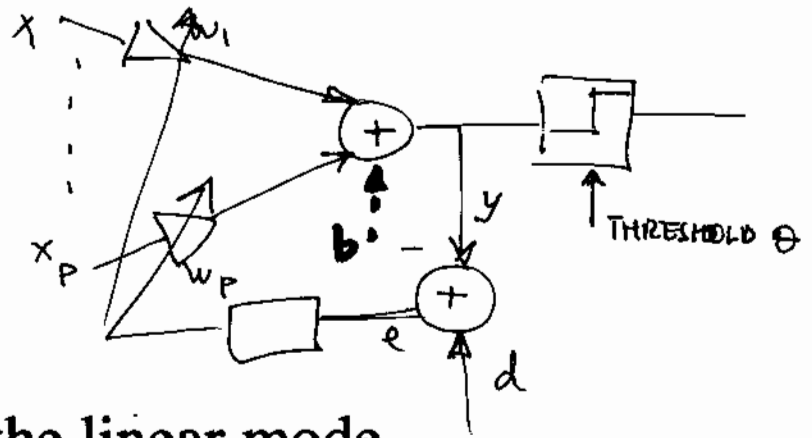
$$\eta(n) = \frac{\eta_0}{1 + \frac{n}{\tau}}$$

η_0, τ → PROBLEM DEPENDENT

But this is still more of an art then a science....

ADALINE (adaptive linear element)

It is the linear system followed by a nonlinearity



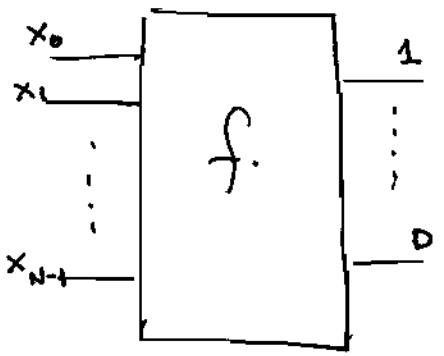
But adapted in the linear mode.

Linear Machines

Let us now take the pattern recognition view of the same problem. Assume we have a set of **D** patterns x, with N components

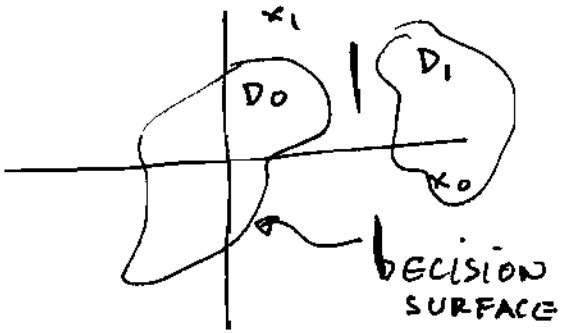
$$\{x_0, x_1, \ldots x_{N-1}\}$$

Each pattern is a point in a N dimensional space R^N called the pattern space



$$f : R^N \rightarrow \{1, 2, \ldots D\}$$

A classifier is a MAPPER from R^N to the subset of integers {1,2,...,D}, the classes.



$d_1(x), d_2(x) \ldots, d_D(x)$

The point sets in R^2 are separated from each other by surfaces (hypersurfaces) called decision surfaces. The decision surfaces can be defined implicitly by a set of scalar and single valued functions in D dimensions.

$$d: \quad R^N \longrightarrow R$$

They are called DISCRIMINANT functions, ands they have the property that

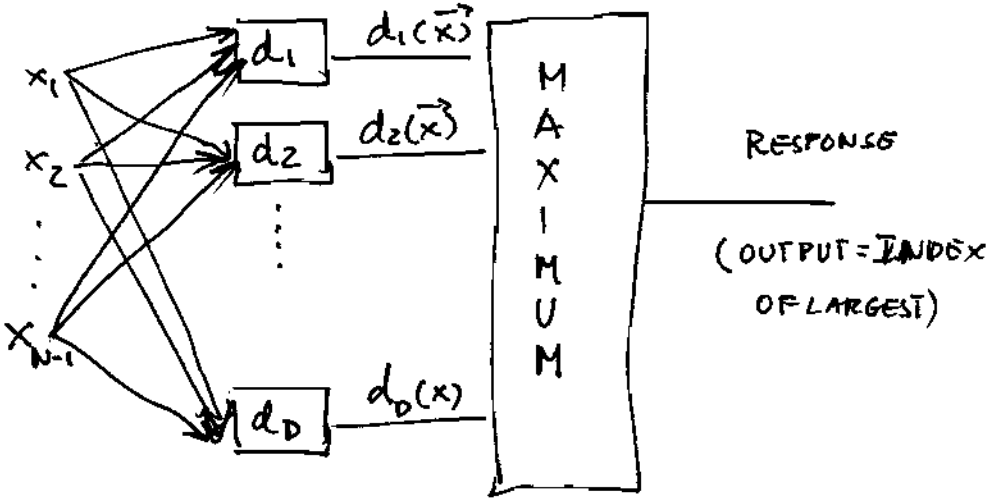
$$d_i(\vec{x}) > d_j(\vec{x}) \qquad \text{for } i=1,\ldots D \qquad i \neq j$$

In region i, $d_i(x)$ is larger than any other function. The equation for the decision surface separating contiguous regions is

$$d(x) = d_i(x) - d_j(x) = 0$$

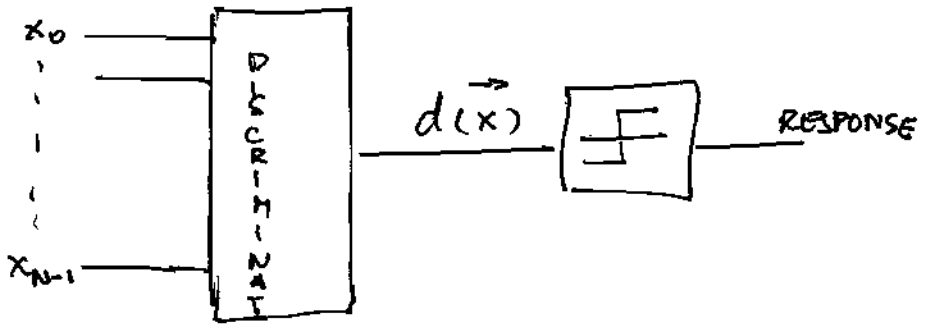
The location and the shape of the decision surface does NOT uniquely specify the discriminant function (just add some constant to all the classes and the result is the same)

How can we implement decision surfaces?



If D=2 the selector will decide which is larger. This is simply a sign computation if we create a discriminant function

$$d(\vec{x}) = 0 \quad \text{WHERE} \quad d(\vec{x}) = d_1(\vec{x}) - d_2(\vec{x})$$



We can see now that the central problem is pattern recognition is the design of discriminant functions.

Discriminant Functions

We saw how our knowledge of the conditionals enabled us to find the optimal discriminant function for the signal constellation.

There are cases where we do not have the knowledge, so we have to GUESS the function, and give it the ability to ADAPT to the data. We can do this by providing a parametric representation of the discriminant function and find the coefficients. This procedure is called

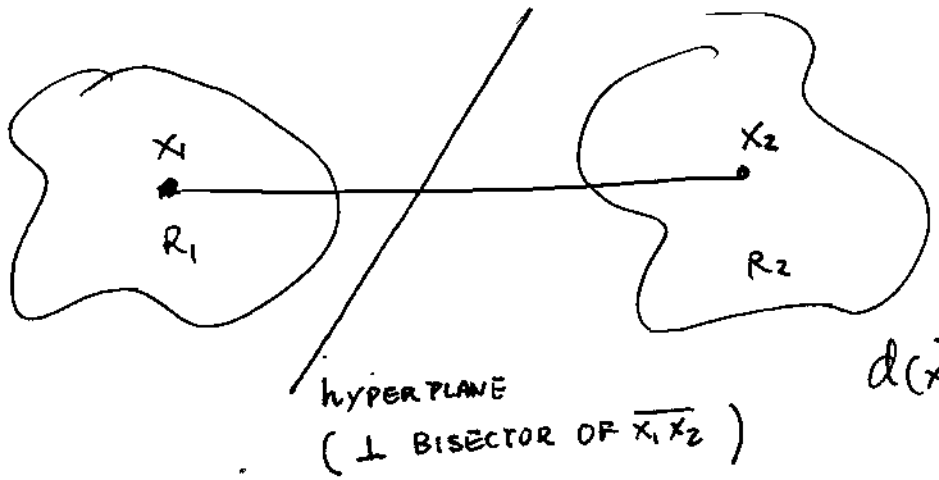
the TRAINING of the classifier.

There are basically TWO methods in trainable classifiers:

parametric

non-parametric

Parametric requires the apriori knowledge of a set of parameters that characterizes each class (the conditionals in the communication example $\bar{x}$, $\bar{\sigma}$), with unknown values.



x_1, x_2 NOT KNOWN

$$d(\vec{x}) = (\bar{x}_1 - \bar{x}_2).\vec{x} + \frac{1}{2}|\bar{x}_2|^2 - \frac{1}{2}|\vec{x}_1|^2$$

Parametric methods will define x_1, x_2.

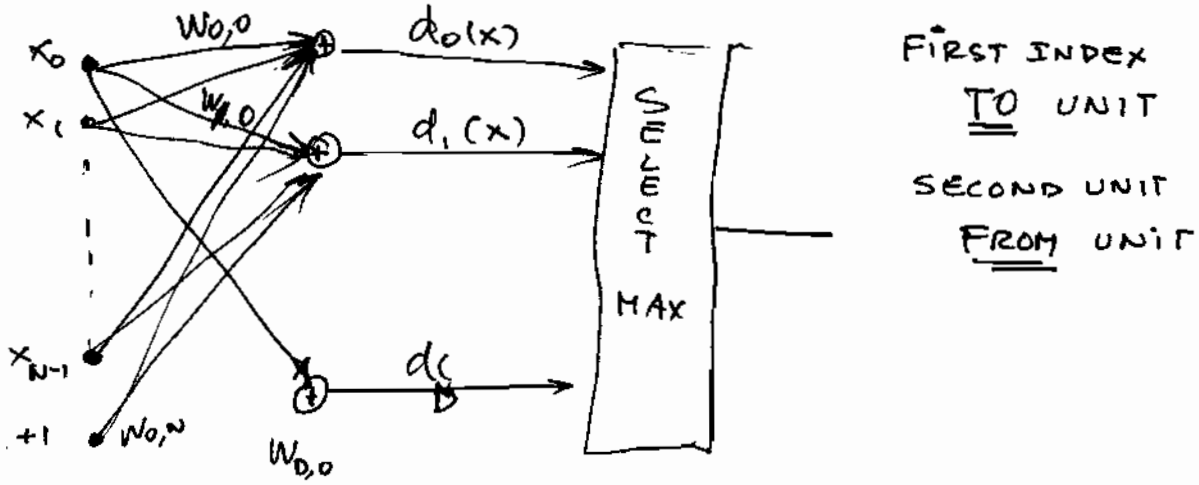
NON-PARAMETRIC methods

No assumption can be made about the characteristics parameters. Functional forms for the discriminant functions will be assumed (linear, quadratic, polynomial, etc.).

Let us assume a linear form

$$\begin{cases} d(\vec{x}) = w_0 x_0 + w_1 x_1 + \cdots + w_{N-1} x_{N-1} + w_N \\ d(\vec{x}) = 0 \Longrightarrow \text{HYPERPLANE} \end{cases}$$

The problem is to find the coefficients which we call the WEIGHTS. A pattern classifier that implements this rule is called a LINEAR MACHINE.



These machines can implement minimum distance classifiers. For D points in R^N, the Euclidean distance is $P_1, \ldots P_D \in R^N$

$$|x - P_i| = \sqrt{(\bar{x} - \bar{P}_i)(\bar{x} - \bar{P}_i)}$$

So minimum distance classifier will place x in the class P_i closest to

X. We just need to compute $|x - p_i|$

and choose i such that value is minimum. An equivalent classifier is obtained if we compare the square distance

$$|x-p_i|^2 = (x-p_i)(x-p_i) = x \cdot x - 2x \cdot p_i + p_i \cdot p_i$$

Since X is a constant, can work only with $d(x) = x \cdot p_i - \frac{1}{2} p_i \, p_i$ $i = 1, \ldots D$

So the conclusion is that in L_2 the distance is a linear machine (linear in x). Comparing with previous example

$$p_i \equiv p_{i1}, p_{i2}, \ldots p_{iD} = w_{ij} \qquad i \rightarrow 0, \ldots N-1 \quad j \rightarrow 1, \ldots D$$

$$w_{i,D+1} = -\frac{1}{2} p_i \cdot p_i$$

The decision surface between two adjacent regions is a surface with equations

$$d_i(x) - d_j(x) = 0$$

We know that the dot product is a matched filter operation (or correlation detector).

For N dimensional space there are N((N-1)/2) such equations, but only a few are in fact used for the classification (regions that are adjacent).

For linear machines, the decision regions are always CONVEX (a straight line passing by 2 pints in the regions always belongs to the region).

A problem is said to be LINEARLY SEPARABLE if to classify a set of K input patterns $X_1,...X_K$, a linear machine can place them in D classes with NO error, i.e.

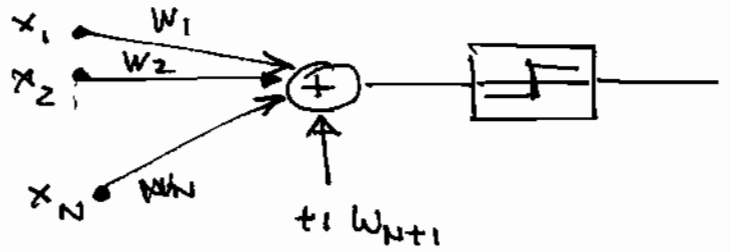
$$d_i(x) > d_j(x) \quad \forall \vec{x} \text{ in region } i \quad j=1,\ldots,D \quad i \neq j$$

A special case is the dichotomy $(D=2)$

$$d(x) > 0$$
$$d(x) < 0$$

We only need a threshold logic unit to implement it.

$$d(x) = w_1 x_1 + \cdots + w_N x_N + w_{N+1}$$



$$d(x) = \sum_{j=1}^{N} w_{ij}\, x_j + w_{N+1}$$

The hyperplane (discriminant function) has an equation

$$\vec{X} \cdot \vec{W} = -w_{N+1}$$

Another form is obtained if P is a point on the hyperplane, and N the normal at P

$$(X - P) \cdot \vec{n} = 0$$

$$\begin{cases} n = \dfrac{\vec{w}}{|\vec{w}|} \\ \vec{n} \cdot \vec{P} = \dfrac{-w_{N+1}}{|\vec{w}|} \end{cases}$$

DISTANCE TO ORIGIN, ΔW

Therefore, we can write the discriminant function

$$\vec{X} \cdot \vec{n} + \Delta W = 0$$

Famous LTU machines - The PERCEPTRON (Rosenblatt 1950).

The Perceptron is a device capable of computing all predicates which are linear in some set ϕ of partial predicates

$$\phi = \{ \varphi_1, \cdots \varphi_m \}$$

A predicate is a function that translates properties

$$\varphi_p(x) = \begin{cases} 1 & \text{IF } P \text{ IS IN } x \\ 0 & \text{OTHERWISE} \end{cases}$$

Since p(x) can be interpreted as True or False, it can be thought as a variable statement where true or false depends on the value of X (the predicate).

Linear predicates

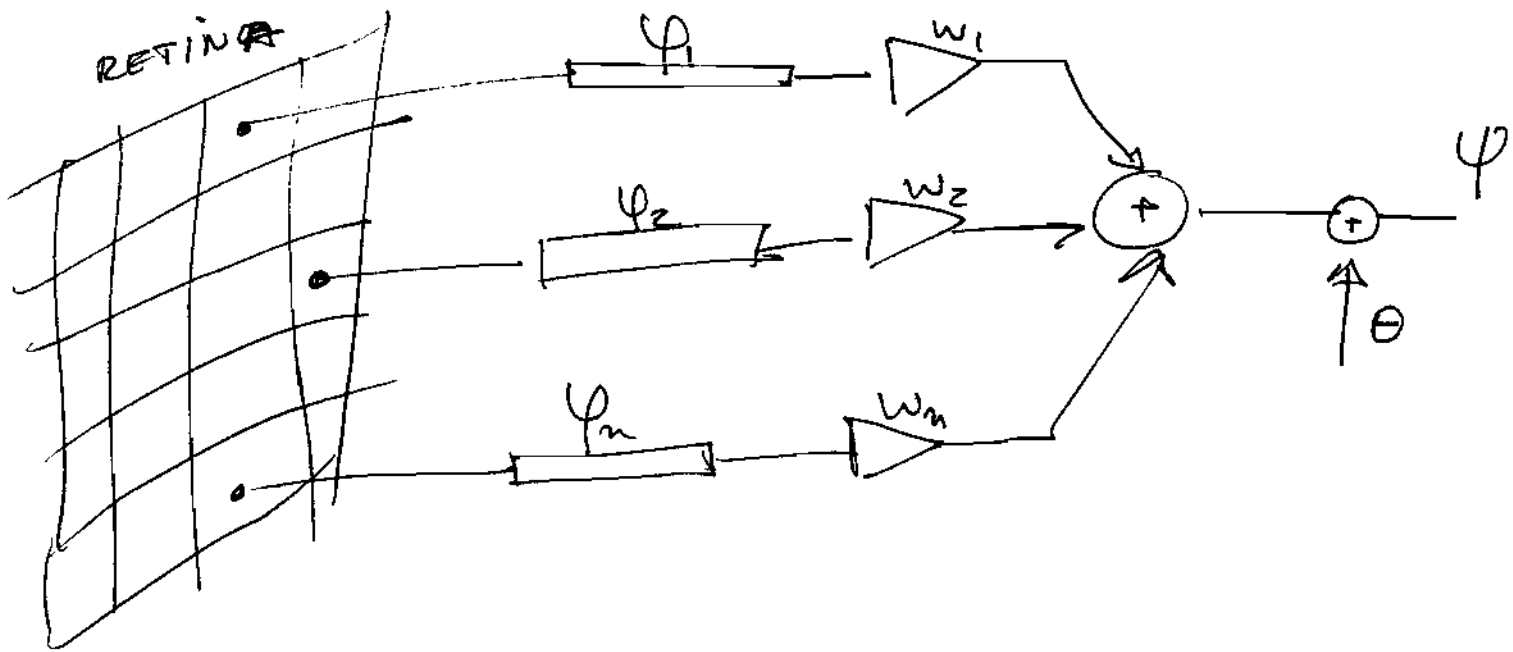
ψ is a linear predicate of ϕ if there exists a number θ and a set of numbers $\{ w_{\varphi_1}, w_{\varphi_2}, \cdots w_{\varphi_n} \}$

such that $\psi(x) = 1$ IFF

$$w_{\varphi_1}\varphi_1(x) + \cdots + w_{\varphi_n}\varphi_n(x) > \theta$$

The number θ is called the threshold and w the weights.

$$\psi(x) = 1 \quad \text{IIF} \quad \sum_i w_{\varphi_i}\varphi_i(x) > \theta$$



Two ideas:

How does this work?

Each predicate of ϕ is providing evidence about whether Ψ is true for any input x. If Ψ is strongly correlated with $\varphi(x)$ we expect w_φ to be positive.

How are we computing Ψ?

In two steps: First is parallel since we compute $\varphi i(x)$ in parallel, them we combine the info in the adder.

Question:

What type of functions can we compute with the perceptron? Not all Ψ, unfortunately. We are using simply linear discriminant functions as we saw.

Not every decision problem can be solved with a linear discriminant function. The complexity of the decision is sometimes called the order of the predicate. All Boolean inequalities of 2 variables (16) are

linear except the exclusive or and the equivalence, which are of order 2 (quadratic). This is bad news, since the Turing machine can compute all of them!!!!

(this argument was used by Minsky &Papert - Perceptrons).

LEARNING

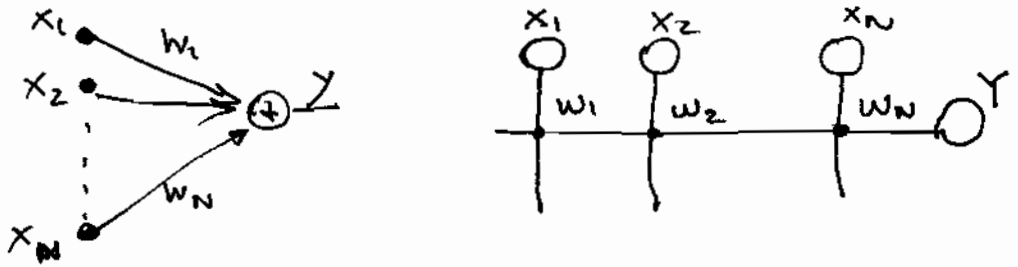
The nice thing about the perceptron is that there is a finite procedure (i.e that reaches convergence in a finite number of steps) that accomplishes training, i.e. that is guaranteed to discovers the $w_{\varphi i}$ to classify any x as long as Ψ is linear.

RULE: Present a pattern. If the response is good, don't touch the weights. If response is wrong, then change the weights of the units that are active, until the desired response is obtained. Keep presenting the inputs and making the adjustments until the desired result is obtained for all patterns. This process converges in a finite number of times. REMEMBER: learning requires a lot of patterns....

Solution will be a vector W*. This can be proved mathematically (see Minsky and Papert, or Nilssen).

VIEW IN VECTOR SPACES

Consider a single output unit, and let us make f()=1.



Each unit receives from N units from previous layer. Associated with each N+1 units there is a scalar value computed by the inner product of w and x.

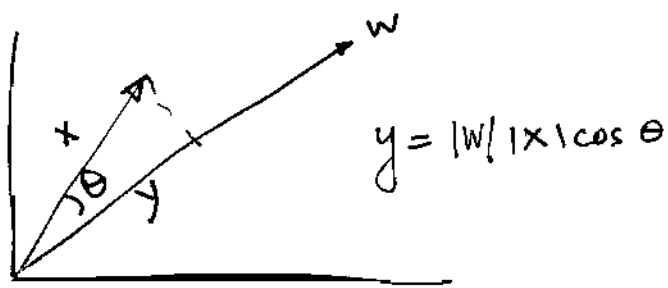
$\vec{X}$ is an activation of input units

$\vec{Y}$ is an activation of the output unit.

$$\vec{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_N \end{bmatrix}$$

Associated with each link there is a weight value w_i. So we have a weight row vector w, and the output is the INNER PRODUCT

$$\vec{Y} = \vec{W} \cdot \vec{X}$$



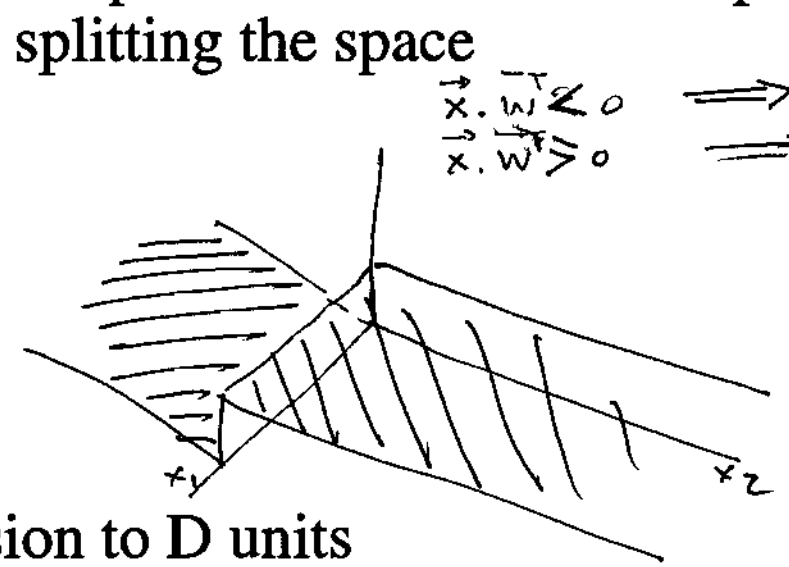
Therefore the output measures HOW CLOSE the input is to the stored weight vector.

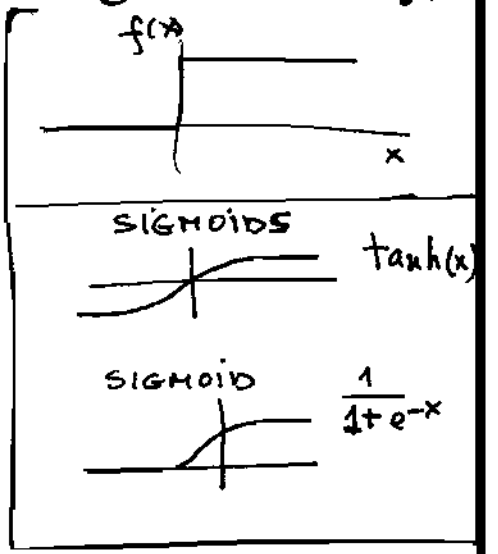
CLOSE TO 90° $y \simeq 0$

CLOSE TO 0 $y \simeq$ MAX, POSITIVE

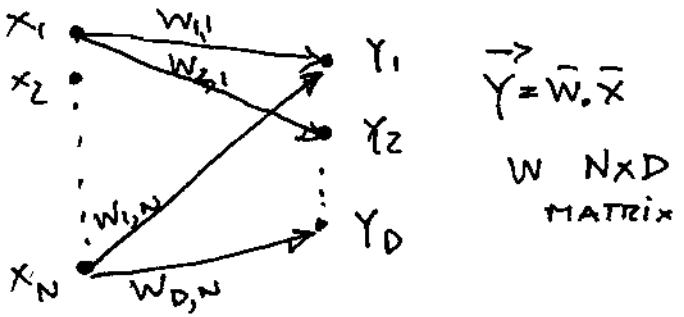
CLOSE TO 180° $y =$ MAX NEGATIVE

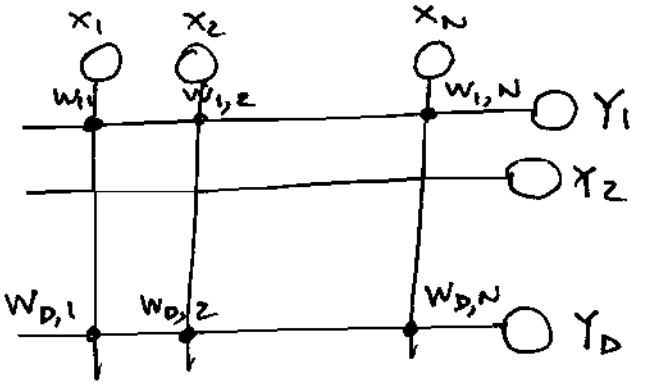
Now if we put a threshold at the output, we are creating a boundary, we are splitting the space





Extension to D units

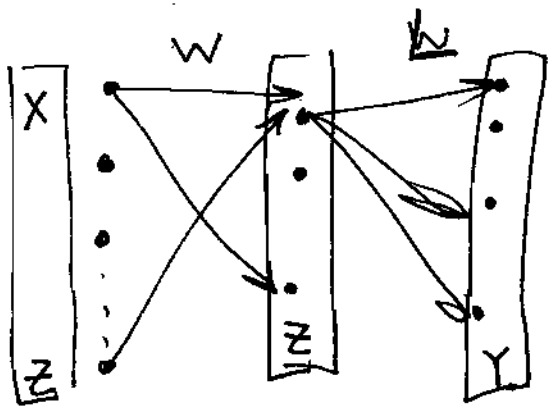




Now W becomes a NxD matrix

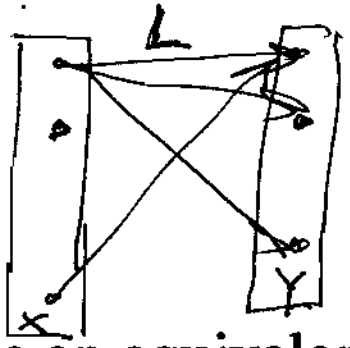
As before each output unit is computing how close the input is to the stored weight vector wi, and this result is displayed as the activation at the unit.

Is there any advantage of creating multilayer networks of linear elements? NO



$$Y = \vec{L} \cdot \vec{z} = \vec{L} \cdot \vec{W} \cdot x$$

$$K = \vec{L} \cdot \vec{W}$$



We can always compute an equivalent one layer system.

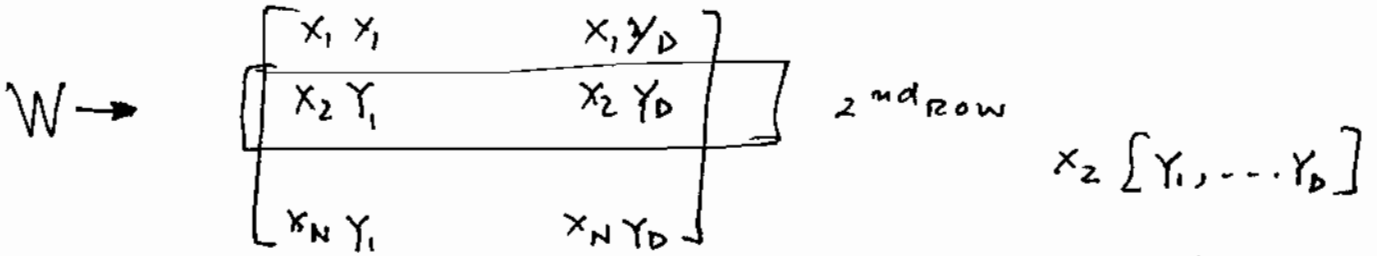
LEARNING AS OUTERPRODUCTS

How can we find the weight matrix for a linear network? Very easily, by computing the OUTER-PRODUCT.

By definition the outerproduct of two vectors (assumed column) is

$$Y X^T$$

Notice that the rows of the resulting matrix wi are $\vec{\omega}_i = x_i \bar{Y}$



So if I know Y and X, it is a simple matter to compute the W matrix in the linear case. Just make $W = Y X^T$

$$\bar{Y} = \bar{W}\,\bar{X}$$

$$Y X^T = W X X^T = W \cdot \bar{I} = W$$

(ASSUME X NORMALIZED)

This rule sometimes is called HEBBIAN rule. Present a pattern and modify the weight wij by the product of the i output by the j input

$$W_{ij} = x_j y_i$$

Notice that the rule is LOCAL in space, i.e. all the information to adapt wij is available at the i,j connection.

$$W_{ij}^{NEW} = W_{ij}^{OLD} + (x_j y_i)\mu \qquad \mu \rightarrow \text{SMALL}$$

This linear network has been used as a LINEAR ASSOCIATIVE (content addressable) memory (Wilshaw, Andersen, Canianello).

Properties of these memories:

Totally distributed

Robust

Local Information

But for perfect recall require orthogonal patterns

Number of patterns less than N (at most N orthogonal patterns).

If input vectors are not orthogonal there will be a recall error that can be estimated by the signal to noise expansion

$$W x_k = y_1 (x_1^T x_k) + y_2 (x_2^T x_k) + \cdots + y_D (x_D^T x_k) = y_k (x_k^T x_k) + \sum_{j \neq k} y_j (x_j^T x_k)$$

$$= y_k + \eta$$

In order to choose W to minimize this error, we compute the Pseudo Inverse (Penrose inverse), which is best in terms of the minimum square error.

$$F(w) \equiv \left(\frac{1}{D}\right) \sum_{k=1}^{D} |y_k - W x_k|^2$$

The pseudo inverse of an arbitrary pxq matrix A is defined as the unique matrix A+

$$A A^+ A = A$$

$$A^+ A A^+ = A^+$$

$$A A^+ = (A A^+)^T$$

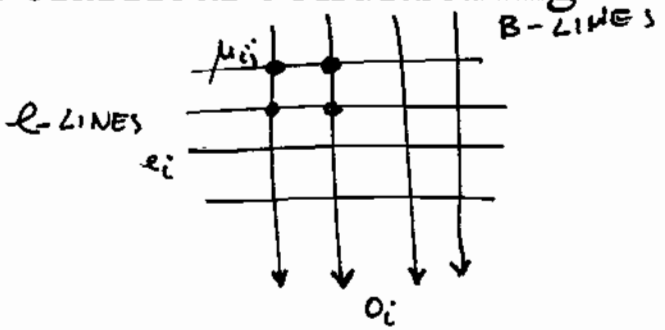
$$A^+ A = (A^+ A)^T$$

The Hebbian weight matrix is the first term of the expansion of the pseudo inverse formula (Kohonen).

Classical Models using distributed memories:

Learning Matrix - Steinbuch, 1963

Model of classical conditioning



$\mu_{ij} \rightarrow$ WEIGHTS

$$o_i = \sum_{i=1}^{n} \mu_{ij} e_j \qquad e_i = \{0, 1\}$$

e lines- stimulus patterns

b lines - classification result

During training, desired "behavior" is forced on the b-lines while stimulus are present in e-lines. In the recall phase only e-lines are active. Learning rule was

e_i \ e_j	0	1
0	*	*
1	*	μ_{ij} increased

CONJUNCTION LEARNING RULE

$(\Delta \mu_{ij})$

Note that there is no corrective action towards a desired response (error).

Analyzing the rule we get (AFTER N PATTERNS)

$$\mu_{ij}^{N} = \Delta\mu \sum_{k=1}^{N} o_i^k e_j^k$$

$$o_i = \sum_{j=1}^{n} \mu_{ij}^{N} e_j = \sum_{k=1}^{N} \alpha_k o_i^k$$

WHERE $\alpha_k = \Delta\mu \sum_{j=1}^{n} e_i e_j$

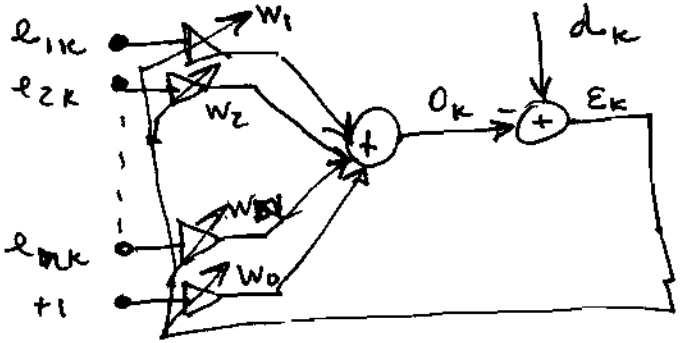
So coefficients are seen to be proportional to the inner product of two patterns. This is the key to associative mappings (correlation). Notice that the memory is distributed, i.e. is robust to degradation (connections destroyed at random).

Other researchers - Willshaw, Caianiello, Kohonen

Classical Learning Systems

Madaline- Widrow 1962

Adaptive Linear element



Notice that the input - output relation is

$$o_k = \sum_{i=1}^{n} w_i e_i + w_0$$

The problem is to adapt the weights such that a desired signal is obtained. Widrow proposed the minimization of the mean square error, using gradient descent learning. He further proposes a rule called the LMS rule that changes weights according to $w_{k+1} = w_k + \mu \varepsilon_k e_k$

LMS is local in space and in time!!!!

MultiLayer Perceptrons

MLPs are feedforward, layered nets. PEs that do not communicate with input or output are called hidden (and hidden layers).

These nets can be trained with the backpropagation algorithm, essentially a steepest descent type of algorithm. It is an extension of LMS in two respects: accepts nonlinear PEs, and can train hidden PEs (no desired signal available to them).

Backprop has been re-invented several times... The first seems to be Bryson and Ho (1965). Werbos comes next in 1974, and Rumelhart/Hinton/Williams brought it to the general public.

Backprop is responsible by the recent interest on neurocomputing, and it brings some improvements to the optimization theory (efficiency).

Notation

Activation signal - signals that propagate from input to output.

Error Signal - signals that propagate from output to input.

n → ITERATION #

i, j, k → PEs

$\mathcal{E}(n)$ → INSTANTANEOUS ERROR

$e_j(n)$ → ERROR SIGNAL AT PE j

$d_j(n)$ → DESIRED SIGNAL AT PE j

$y_j(n)$ → ACTIVATION AT PE j

$net_j(n)$ → LINEAR PART OF PE j

$w_{ji}(n)$ → WEIGHT LINKING PE i, PE j

$\varphi_j(\cdot)$ → NONLINEARITY

$x_i(n)$ → INPUT PATTERN TO PE i

η → LEARNING RATE

Derivation using Chain Rule

Define the instantaneous value of the error of PE j as

$$e_j(n) = d_j(n) - y_j(n) \qquad \frac{1}{2} e_j^2(n)$$

and let the total error at the output nodes be

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in O} e_k^2(n)$$

We want to compute the sensitivity of the error with respect to the weights (gradient descent idea). Consider linear PE first

$$y_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n)$$

To compute it use the CHAIN rule as

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \underbrace{\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}}_{1} \underbrace{\frac{\partial y_j(n)}{\partial w_{ji}(n)}}_{2}$$

1- How the error changes with the output y_j

2- How much changing w_{ji} makes y_j change.

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)}$$

Using

$$\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n) \quad ; \quad \frac{\partial e_j(n)}{\partial y_j(n)} = -1 \Rightarrow \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = -e_j(n)$$

For linear units

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = +y_i(n)$$

So as before

$$\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = +\eta\, e_j(n)\, y_i(n)$$

LOCAL GRADIENT — LOCAL INPUT

If PE is nonlinear, what changes? Not much, as long as nonlinear function is differentiable (smooth)

$$net_j(n) = \sum_{i=0}^{p} w_{ji}(n)\, y_i(n)$$

$$y_j(n) = \varphi_j(net_j(n))$$

Represent net_j, and use chain rule again

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \underbrace{\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}}_{-e_j(n)} \cdot \underbrace{\frac{\partial y_j(n)}{\partial net_j(n)}}_{\varphi_j'(net_j(n))} \cdot \underbrace{\frac{\partial net_j(n)}{\partial w_{ji}(n)}}_{y_i(n)}$$

So we have the final result (Delta Rule)

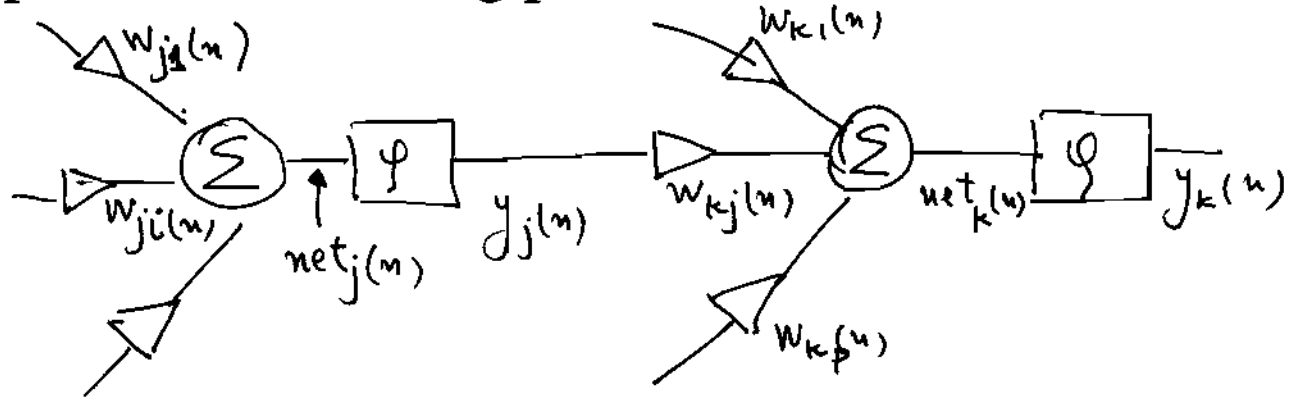
$$\Delta w_{ji}(n) = \eta\, e_j(n)\, \varphi_j'(net_j(n))\, y_i(n)$$

$$= \eta\, \delta_j(n)\, y_i(n)$$

Notice that the same form of the LMS is kept, only now the local gradient includes the effect of the nonlinearity.

Backpropagation

The previous procedure can not be directly applied to multilayer feedforward networks due to the credit assignment problem. What is the error at the output of hidden units? Rumelhart Williams and Hinton proposed the following procedure.



Let us concentrate on computing the sensitivity with respect to jth PE in an internal layer. Assume that we know the error in the layer closer to the output

$$\mathcal{E}(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

AND WANT TO FIND SENSITIVITY W.R.T. j^{th} PE

Differentiating this gives $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$

Now we use the chain rule for the partial derivatives

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = \sum_k e_k(n) \frac{\partial e_k(n)}{\partial net_k(n)} \cdot \frac{\partial net_k(n)}{\partial y_j(n)}$$

Problem is to compute $\frac{\partial net_k(n)}{\partial y_j(n)}$ $\underbrace{\qquad}_{-\varphi_k'(net_k(n))}$

Let us decompose further

$$net_k(n) = \sum_{j=0}^{q} w_{kj}(n)\, y_j(n)$$

and differentiate again

$$\frac{\partial net_k(n)}{\partial y_j(n)} = \sum_{j=0}^{q} w_{kj}(n) \frac{\partial y_j(n)}{\partial y_j(n)} = w_{kj}(n)$$

to obtain

$$\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} = -\sum_k e_k(n)\, \varphi_k'(net_k(n))\, w_{kj}(n) = -\sum_k \delta_k(n)\, w_{kj}(n)$$

$$\Rightarrow \partial_j(n) = \varphi_j'(net_j(n)) \cdot \sum_k \delta_k(n)\, w_{kj}(n)$$

So procedure becomes:

1- Use delta rule to compute errors

$$\Delta w_{ji}(n) = \eta \underbrace{\delta_j(n)}_{\text{LOCAL GRADIENT}} \underbrace{y_i(n)}_{\text{INPUT TO PE}_j}$$

2- If unit is an output unit

$$\delta_j(n) = e_j(n)\,\varphi_j'(net_j(n)) \quad \text{(LOCAL GRADIENT)}$$

3- If unit is an hidden unit

$$\delta_j(n) = \varphi_j'(net_j(n)) \cdot \sum_k \delta_k(n)\, w_{kj}(n) \quad \text{(LOCAL GRADIENT)}$$

During phase 1 activations are propagated to the output (weights are fixed).

During phase 2 output is compared to desired pattern and e_j computed.

During phase 3 the error is propagated back through the network,

one layer at a time. Therefore these are ordered derivatives.

The complexity of the algorithm is O(k), where k is the number of units of the net. This is a substantial savings from previous algorithms.