Automazione (Laboratorio)

Reti Neurali per l'Identificazione, Previsione e Controllo

Lecture 1:

Introduction to Neural Networks

(Machine Learning)

Silvio Simani

References

Textbook (*suggested*):

- *Neural Networks for Identification, Prediction, and Control*, by Duc Truong Pham and Xing Liu. Springer Verlag; (December 1995). ISBN: 3540199594

- *Nonlinear Identification and Control: A Neural Network Approach*, by G. P. Liu. Springer Verlag; (October 2001). ISBN: 1852333421

Course Overview

1. Introduction
   i. Course introduction
   ii. Introduction to neural network
   iii. Issues in Neural network
2. Simple Neural Network
   i. Perceptron
   ii. Adaline
3. Multilayer Perceptron
   i. Basics
4. Radial Basis Networks
5. Application Examples

Machine Learning

- Improve automatically with experience
- Imitating human learning
  - Human learning

    Fast recognition and classification of complex classes of objects and concepts and fast adaptation
  - Example: neural networks
- Other techniques are based on reasoning or inductive inference (e.g. Decision tree, *Fuzzy logic*)

Disciplines relevant to ML

- Artificial intelligence
- Bayesian methods
- Control theory
- Information theory
- Computational complexity theory
- Philosophy
- Psychology and neurobiology
- Statistics

Machine Learning Definition

A computer program is said to **learn** from *experience* **E** with respect to some class of *tasks* **T** and *performance measure* **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience.

Examples of Learning Problems

Example: Handwriting Recognition:

- T: Recognizing and classifying handwritten words within images
- P: percentage of words correctly classified
- E: a database of handwritten words with given classification.

Issues in *Machine Learning*

- What algorithms can approximate functions well and when?
- How does the number of training examples influence accuracy?
- How does the complexity of hypothesis representation impact it?
- How does noisy data influence accuracy?
- *How do you reduce a learning problem to a set of function approximation* ?

Summary

- *Machine Learning* is useful for data mining, poorly understood domain (face recognition) and programs that must dynamically adapt.
- Draws from many diverse disciplines
- Learning problem needs well-specified task, performance metric and training experience
- Involve searching space of possible hypotheses. Different learning methods search different hypothesis space, such as numerical functions, *neural networks*, decision trees, symbolic rules.

TOPICS IN NEURAL NETWORKS

LECTURE 2:
INTRODUCTION

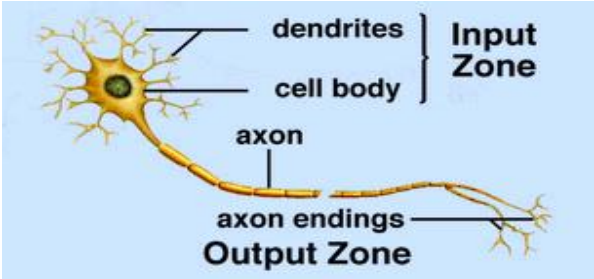
Lecture Outline

1. Introduction
   i. Course introduction
   ii. Introduction to neural network
   iii. Issues in Neural network
2. Simple Neural Network
   i. Perceptron
   ii. Adaline
3. Multilayer Perceptron
   i. Basics
   ii. Dynamics
4. Genetic Algorithms
5. Radial Basis Networks
6. Application Examples

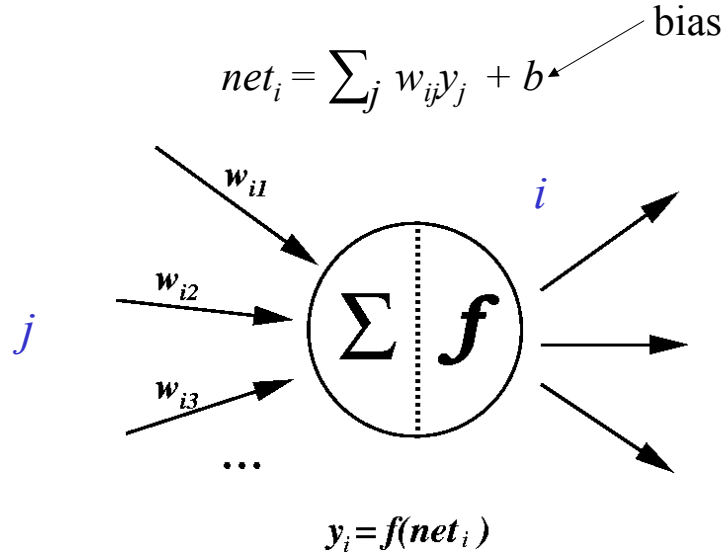
INTRODUCTION TO NEURAL NETWORKS

Brain

- 10^{11} neurons (processors)
- On average 1000-10000 connections



Artificial Neuron



Artificial Neuron

- Input/Output Signal may be
  - Real value
  - Unipolar {0, 1}
  - Bipolar {-1, +1}
- Weight : w_{ij} -- strength of connection.

Note that w_{ij} refers to the weight from unit j to unit i (not the other way round).

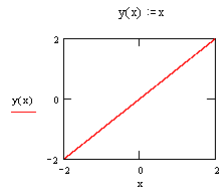
Artificial Neuron

- The bias b is a constant that can be written as $w_{i0}y_0$ with $y_0 = b$ and $w_{i0} = 1$ such that

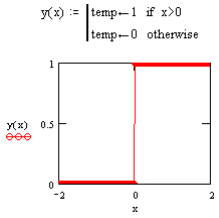
$$net_i = \sum_{j=0}^{n} w_{ij} y_j$$

- The function f is the unit's activation function. In the simplest case, f is the identity function, and the unit's output is just its net input. This is called a linear unit.
- Other activation functions are : step function, sigmoid function and Gaussian function.

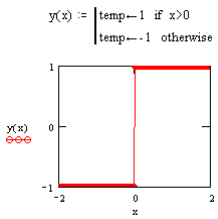
Activation Functions



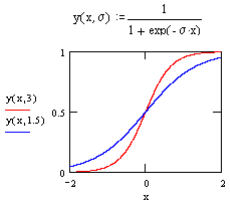
Identity function



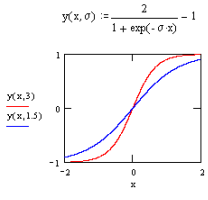
Binary Step function



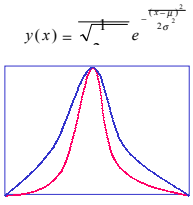
Bipolar Step function



Sigmoid function

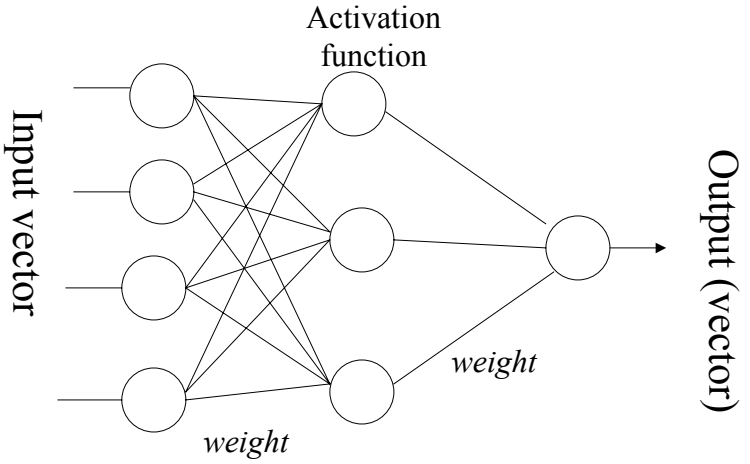


Bipolar Sigmoid function



Gaussian function

Artificial Neural Networks (ANN)



Historical Development of ANN…

- William James (1890) : Describes in words and figures simple distributed networks and Hebbian learning
- McCulloch & Pitts (1943) : Binary threshold units that perform logical operations (they proof universal computation)
- Hebb (1949) : formulation of a physiological (local) learning rule
- Roseblatt (1958) : The perceptron– a first real learning machine
- Widrow & Hoff (1960) : ADALINE and the Widrow-Hoff supervised learning rule

Historical Development of ANN

- Kohonen(1982) : Self-organizing maps
- Hopfield(1982) :Hopfield Networks
- Rumelhart, Hinton & Williams (1986) : Back-propagation & MultiLayer Perceptron
- Broomhead & Lowe (1988) : Radial basis functions (RBF)
- Vapnik (1990) ---support vector machine

When should ANN Solution be Considered ?

•The solution to the problem cannot be explicitly described by an algorithm, a set of equations, or a set of rules.

•There is some evidence that an input-output mapping exists between a set of input and output variables.

•There should be a large amount of data available to train the network.

Problems which can lead to poor performance ?

- The network has to distinguish between very similar cases with a very high degree of accuracy.
- The train data does not represent the ranges of cases that the network will encounter in practice.
- The network has a several hundred inputs.
- The main discriminating factors are not present in the available data. E.g. trying to assess the loan application without having knowledge of the applicant's salaries.
- The network is required to implement a very complex function.

Applications of Artificial Neural Networks

- Manufacturing : fault diagnosis, fraud detection
- Retailing : fraud detection, forecasting, data mining
- Finance : fraud detection, forecasting, data mining
- Engineering : fault diagnosis, signal/image processing
- Production : fault diagnosis, forecasting
- Sales &Marketing : forecasting, data mining.

DATA PRE-PROCESSING

Neural networks very rarely operate on the raw data. An initial pre-processing stage is essential. Some examples are as follows:

- Feature extraction of images: For example, the analysis of X-rays requires pre-processing to extract features which may be of interest within a specified region.
- Representing input variables with numbers. For example "+1" is the person is married, "0" if divorced, and "-1" if single. Another example is representing the pixels of an image: 255 = bright white, 0 = black. To ensure the generalization capability of a neural network, the data should be encoded in form which allows for interpolation.

DATA PRE-PROCESSING

- CONTINUOUS VARIABLES
  - A continuous variable can be directly applied to a neural network. However, if the dynamic range of input variables are not approximately the same, it is better to normalize all input variables of the neural network.

Example of Normalized Input Vector

- Input vector : $(2\ 4\ 5\ 6\ 10\ 4)^t$
- Mean of vector : $\mu = \frac{1}{6}\sum_{i=1}^{6} x_i = 5.167$
- Standard deviation : $\sigma = \sqrt{\frac{1}{6-1}\sum_{i=1}^{6}(x_i - \mu)^2} = 2.714$
- Normalized vector : $x_N = \frac{x_i - \mu}{\sigma} = (-1.17\ \ -0.43\ \ -0.06\ \ 0.31\ \ 1.78\ \ -0.43)^t$
- Mean of normalized vector is zero
- Standard deviation of normalized vector is unity

SIMPLE NEURAL NETWORKS

LECTURE 3:

SIMPLE PERCEPTRON

Outlines

- The Perceptron
- Linearly separable problem
- Network structure
- Perceptron learning rule
- Convergence of Perceptron

THE PERCEPTRON

- The perceptron was a simple model of ANN introduced by Rosenblatt of MIT in the 1960' with the idea of learning.

- Perceptron is designed to accomplish a simple pattern recognition task: after learning with real value training data

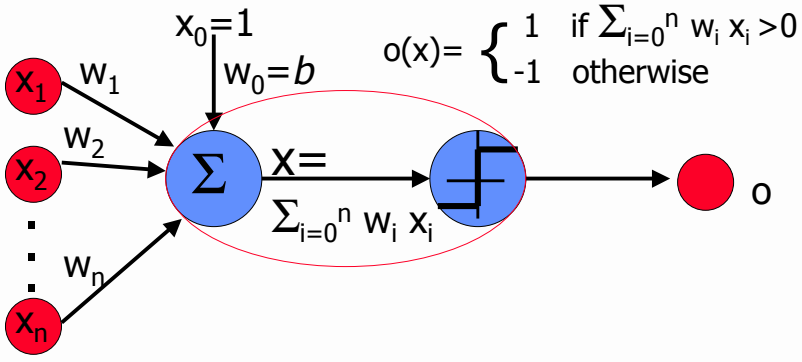
  $\{ \underline{x(i)}, d(i), \ i = 1,2, \ldots, p\}$ *where* $d(i) = 1$ *or* -1

- For a new signal (pattern) $\underline{x(i+1)}$, the perceptron is capable of telling you to which class the new signal belongs

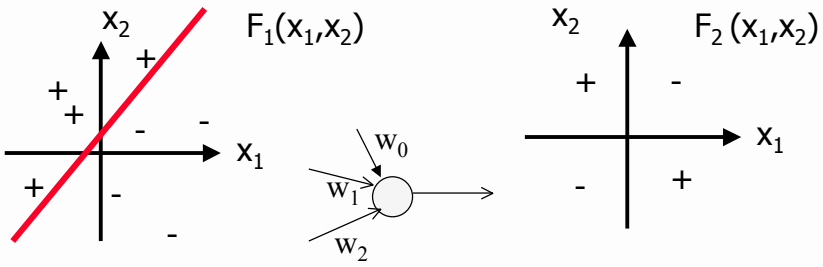
$\underline{x(i+1)}$ → perceptron = 1 or −1

Perceptron

- Linear threshold unit (LTU)



Decision Surface of a Perceptron



- Perceptron is able to represent some useful functions
- $F_1(x_1,x_2)$ choose weights w_0=-1.5, w_1=1, w_2=1
- But functions that are not linearly separable,

  e.g. $F_2(x_1,x_2)$ are not representable

Mathematically the Perceptron is

$$y = f(\sum_{i=1}^{m} w_i x_i + b) = f(\sum_{i=0}^{m} w_i x_i)$$

We can always treat the bias *b* as another weight with inputs equal 1

where f is the hard limiter function i.e.

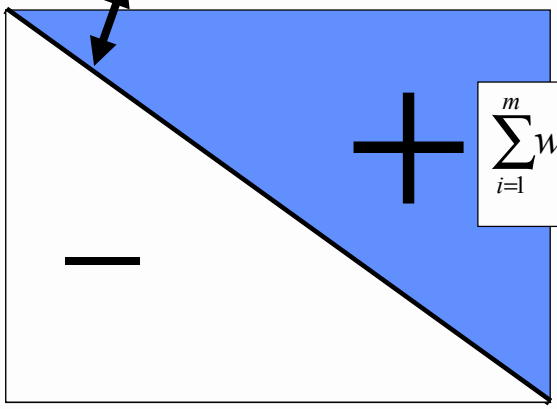
$$y = \begin{cases} 1 \; if \; \sum_{i=1}^{m} w_i x_i + b > 0 \\ -1 \; if \; \sum_{i=1}^{m} w_i x_i + b \leq 0 \end{cases}$$

Why is the network capable of solving linearly separable problem ?

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

$$\sum_{i=1}^{m} w_i x_i + b < 0$$

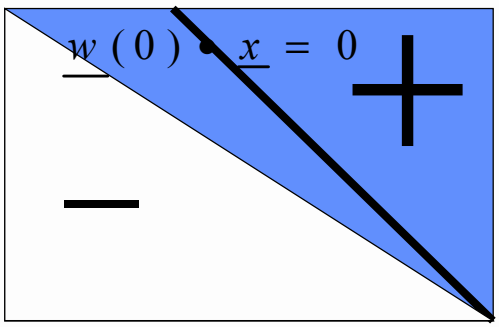
$$\sum_{i=1}^{m} w_i x_i + b > 0$$



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

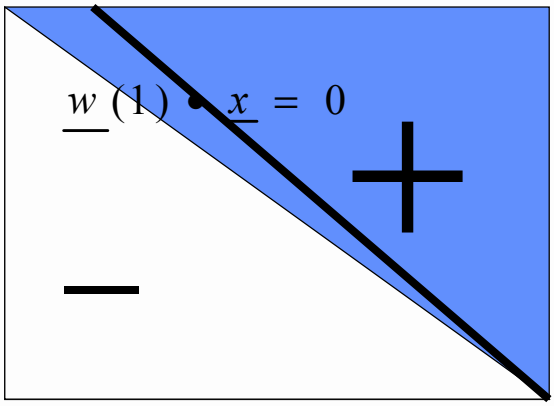
Let *t* be the time, at *t = 0, we have*



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

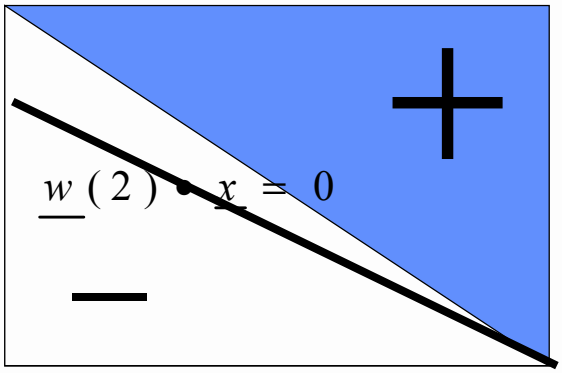
Let *t* be the time, at *t = 1*



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

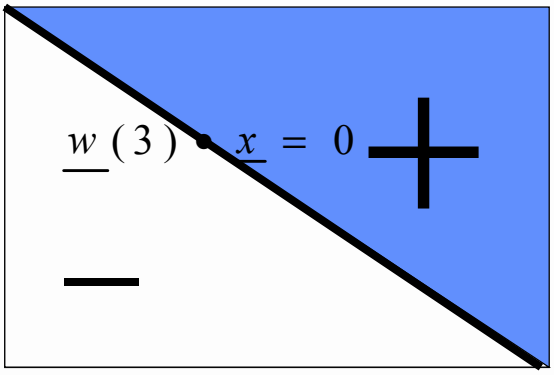
Let *t* be the time, at *t = 2*



Learning rule

An algorithm to update the weights $\underline{w}$ so that finally the input patterns lie on both sides of the line decided by the perceptron

Let *t* be the time, at *t = 3*



In Math

$$d(t) = \begin{cases} +1 \text{ if } x(t) \text{ in class } + \\ -1 \text{ if } x(t) \text{ in class } - \end{cases}$$

Perceptron learning rule

$$\underline{w}(t+1) = \underline{w}(t) + \eta(t)[d(t) - sign(\underline{w}(t) \bullet \underline{x}(t))]\ \underline{x}(t)$$

Where $\eta(t)$ is the learning rate >0,

$$sign(x) = \begin{cases} +1 \text{ if } x>0 \\ \\ -1 \text{ if } x<=0, \end{cases}$$ hard limiter function

NB : d(t) is the same as d(i) and x(t) as x(i)

In words:

- If the classification is right, do not update the weights
- If the classification is not correct, update the weight towards the opposite direction so that the output move close to the right directions.

Perceptron convergence theorem (Rosenblatt, 1962)

Let the subsets of training vectors be linearly separable. Then after finite steps of learning we have

lim $\underline{w}(t)$ **=** $\underline{w}$ **which correctly separate the samples.**

The idea of proof is that to consider $||\underline{w}(t+1)-\underline{w}||-||\underline{w}(t)-\underline{w}||$ which is a decrease function of t

Summary of Perceptron learning …

Variables and parameters

$\underline{x}(t)$ = *(m+1) dim.* input vectors at time t
$= (b, x_1(t), x_2(t), \ldots, x_m(t))$

$\underline{w}(t)$ = *(m+1) dim.* weight vectors
$= (1, w_1(t), \ldots, w_m(t))$

b = bias
$y(t)$ = actual response
$\eta(t)$ = learning rate parameter, a +ve constant < 1
$d(t)$ = desired response

Summary of Perceptron learning …

Data $\{ (\underline{x}(i), d(i)), i=1,\ldots,p\}$

✓ Present the data to the network once a point

✓ could be cyclic :
$(\underline{x}(1), d(1)), (\underline{x}(2), d(2)),\ldots, (\underline{x}(p), d(p)),$
$(\underline{x}(p+1), d(p+1)),\ldots$

✓ *or randomly*

(Hence we mix time t with i here)

Summary of Perceptron learning (algorithm)

1. Initialization Set $\underline{w}(0)=0$. Then perform the following computation for time step t=1,2,...

2. Activation At time step t, activate the perceptron by applying input vector $\underline{x}(t)$ and desired response $d(t)$

3. Computation of actual response Compute the actual response of the perceptron

$$y(t) = sign\ (\ \underline{w}(t) \cdot \underline{x}(t)\ )$$

where ***sign*** is the sign function

4. Adaptation of weight vector Update the weight vector of the perceptron

$$\underline{w}(t+1) = \underline{w}(t) + \eta(t)\ [\ d(t) - y(t)\ ]\ \underline{x}(t)$$

5. Continuation

Questions remain

Where or when to stop?

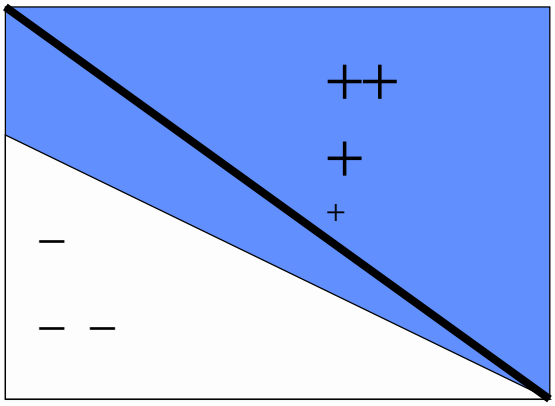
By minimizing the generalization error

For training data $\{(\underline{x}(i), d(i)), i=1,...p\}$

How to define training error after t steps of learning?

$$E(t)= \sum^{p}_{i=1} [d(i)-sign(\underline{w}(t) \cdot \underline{x}(i)]^2$$

After learning t steps



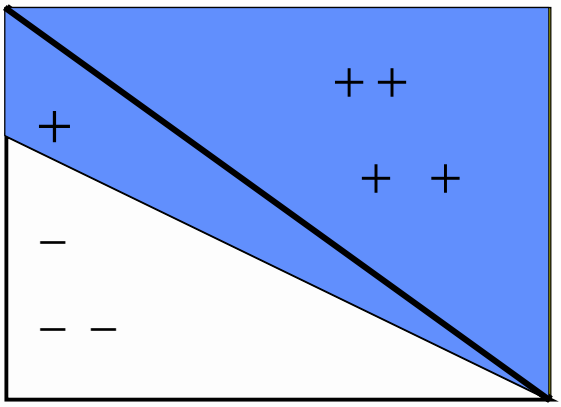
E(t) = 0

How to define generalization error?

For a new signal {$\underline{x}$(t+1),d(t+1)}, we have

$$E_g = [d(t+1) - sign(\underline{x}(t+1) \bullet \underline{w}(t))]^2$$

After learning t steps



We next turn to ADALINE learning,
from which we can understand
the learning rule, and more general the
Back-Propagation (BP) learning

SIMPLE NEURAL NETWORK

LECTURE 4:
ADALINE LEARNING

Outlines

- ADALINE
- Gradient descending learning
- Modes of training

Unhappy over Perceptron Training

- When a perceptron gives the right answer, no learning takes place
- Anything below the threshold is interpreted as 'no', even it is just below the threshold.
- It might be better to train the neuron based on how far below the threshold it is.

ADALINE

•ADALINE is an acronym for ADAptive LINear Element (or ADAptive LInear NEuron) developed by Bernard Widrow and Marcian Hoff (1960).

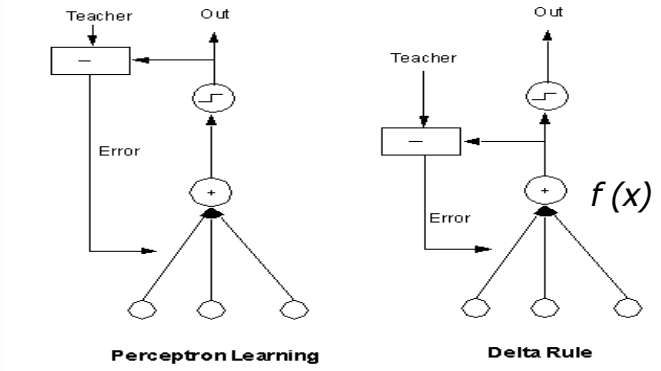
• There are several variations of Adaline. One has threshold same as perceptron and another just a bare linear function.

•The Adaline learning rule is also known as the least-mean-squares (LMS) rule, the delta rule, or the Widrow-Hoff rule.

• It is a training rule that minimizes the output error using (approximate) gradient descent method.

• Replace the step function in the perceptron with a continuous (differentiable) function *f*, e.g the simplest is linear function

• With or without the threshold, the Adaline is trained based on the output of the function *f* rather than the final output.



After each training pattern $\underline{x}(i)$ is presented, the correction to apply to the weights is proportional to the error.

$$E(i,t) = \tfrac{1}{2} [\, d(i) - f(\underline{w}(t) \cdot \underline{x}(i)) \,]^2 \qquad i=1,\ldots,p$$

N.B. If f is a linear function $f(\underline{w}(t) \cdot \underline{x}(i)) = \underline{w}(t) \cdot \underline{x}(i)$

Summing together, our purpose is to find $\underline{W}$ which minimizes

$$E(t) = \sum_i E(i,t)$$

General Approach gradient descent method

To find g

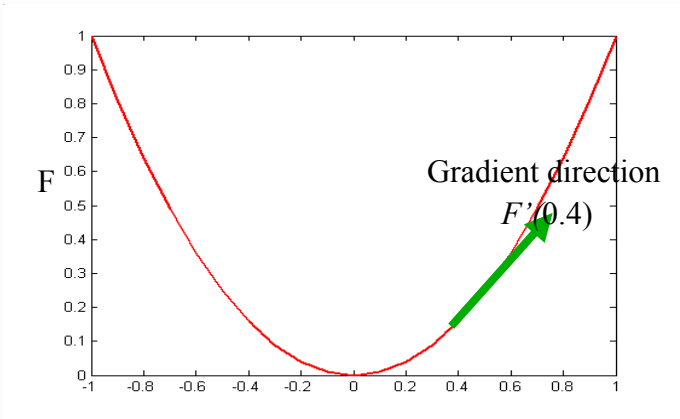
$$\underline{w}(t+1) = \underline{w}(t)+g(E(\underline{w}(t)))$$

so that $\underline{w}$ automatically tends to the global minima of E(w).

$$\underline{w}(t+1) = \underline{w}(t)- E'(\underline{w}(t))\eta(t)$$

(see figure below)

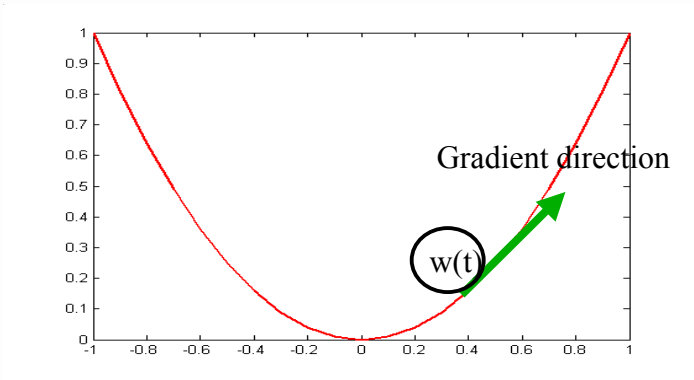
- Gradient direction is the direction of uphill for example, in the Figure, at position 0.4, the gradient is uphill (F is E, consider one dim case)



- In gradient descent algorithm, we have

$$\underline{w}(t+1) = \underline{w}(t) - F'(w(t))\,\eta(\tau)$$

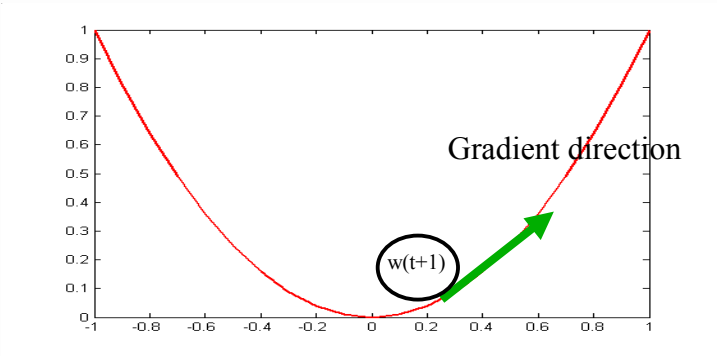
therefore the ball goes downhill since – F'(w(t)) is downhill direction



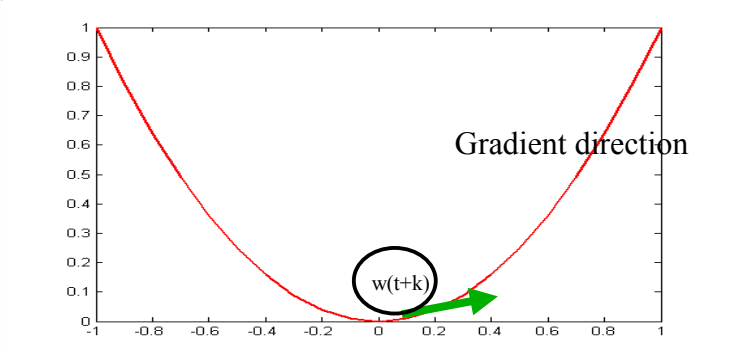
- In gradient descent algorithm, we have

$$w(t+1) = w(t) - F'(w(t))\,\eta(\tau)$$

therefore the ball goes downhill since – F'(w(t)) is downhill direction



- Gradually the ball will stop at a local minima where the gradient is zero



- **In words**

Gradient method could be thought of as a ball rolling down from a hill: the ball will roll down and finally stop at the valley

Thus, the weights are adjusted by

$$w_j(t+1) = w_j(t) + \eta(t) \Sigma \; [d(i) - f(\underline{w}(t) \cdot \underline{x}(i))] \; x_j(i) \; f'$$

This corresponds to gradient descent on the quadratic error surface E

When f' =1, we have the perceptron learning rule (we have in general f'>0 in neural networks). The ball moves in the right direction.

Comparison Perceptron and Gradient Descent Rules

- Perceptron learning rule guaranteed to succeed if
  - Training examples are linearly separable
  - Sufficiently small learning rate η
- Linear unit training rule uses gradient descent guaranteed to converge to hypothesis with minimum squared error given sufficiently small learning rate η
  - Even when training data contains noise
  - Even when training data not separable by Hyperplane

Summary of Previous Lectures

Perceptron

$$\underline{W}(t+1)= \underline{W}(t)+\eta(t) [d(t) - \text{sign} (\underline{w}(t) . \underline{x})] \underline{x}$$

Adaline (Gradient descent method)

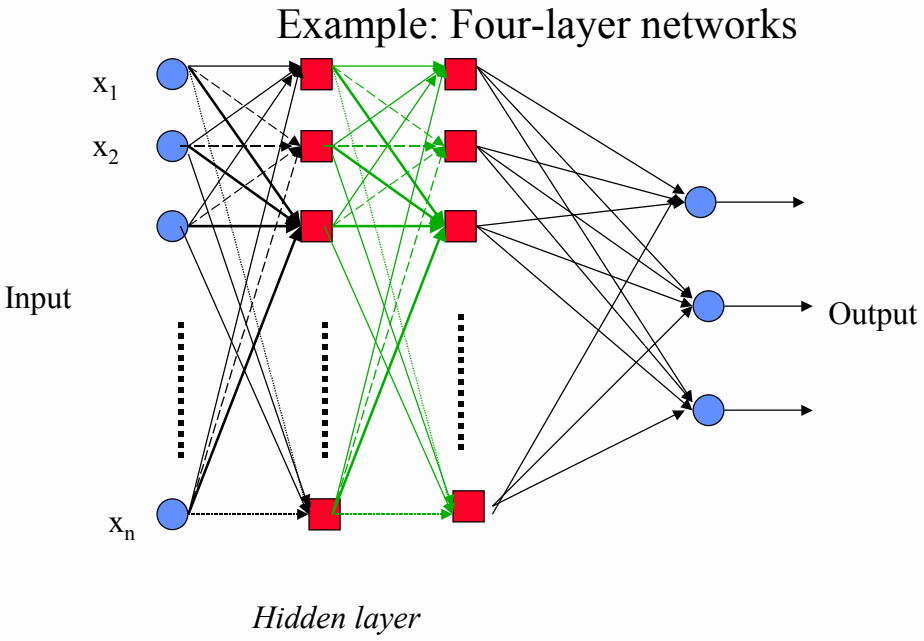
$$\underline{W}(t+1)= \underline{W}(t)+\eta(t) [d(t) - f(\underline{w}(t) . \underline{x})] \underline{x}\, f'$$

Multi-Layer Perceptron (MLP)

Idea: Credit assignment problem

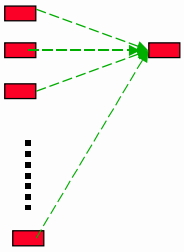
- Problem of assigning 'credit' or 'blame' to individual elements involving in forming overall response of a learning system (hidden units)

- In neural networks, problem relates to dividing which weights should be altered, by how much and in which direction.



Properties of architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 2 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units



Each unit is a perceptron

$$y_i = f\left(\sum_{1}^{m} w_{ij} x_j + b_i\right)$$

BP (Back Propagation)

↘

gradient descent method

+

multilayer networks

LECTURE 5
MULTILAYER PERCEPTRON I

BACK PROPAGATING LEARNING

BP learning algorithm

Solution to credit assignment problem in MLP

Rumelhart, Hinton and Williams (1986)

BP has two phases:

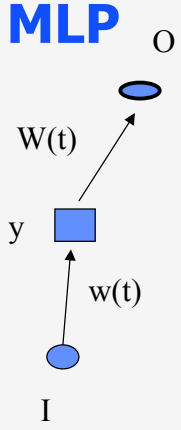
Forward pass phase: computes 'functional signal', feedforward propagation of input pattern signals through network

Backward pass phase: computes 'error signal', propagation of error (difference between actual and desired output values) backwards through network starting at output units

BP Learning for Simplest MLP

Task : Data {I, d} to minimize

$$E = (d - o)^2 /2$$
$$= [d - f(W(t)y(t))]^2 /2$$
$$= [d - f(W(t)f(w(t)I))]^2 /2$$



Error function at the output unit

Weight at time t is w(t)and W(t),
intend to find the weight w and W at time t+1

Where y = f(w(t)I), output of the hidden unit

Forward pass phase

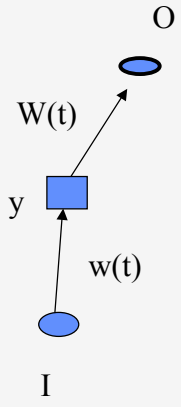
Suppose that we have w(t), W(t) of time t

For given input I, we can calculate

y = f(w(t)I)

and

o = f (W(t) y)
= f (W(t) f(w(t) I))

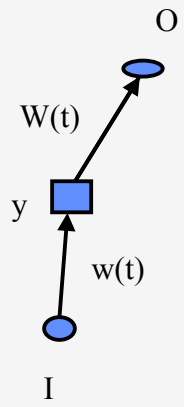


Error function of output unit will be

$$E = (d - o)^2 /2$$

Backward pass phase

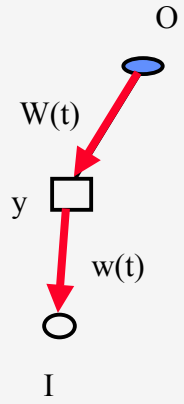
$$W(t+1) = W(t) - \eta \frac{dE}{dW(t)}$$
$$= W(t) - \eta \frac{dE}{df}\frac{df}{dW(t)}$$
$$= W(t) + \eta(d-o) f'(W(t)y)y$$



E = $(d - o)^2 / 2$ o = f (W(t) y)

Backward pass phase

$$W(t+1) = W(t) - \eta \frac{dE}{dW(t)}$$
$$= W(t) - \eta \frac{dE}{df}\frac{df}{dW(t)}$$
$$= W(t) + \eta(d-o) f'(W(t)y)y$$
$$= W(t) + \eta\Delta y$$



where $\Delta = (d-o) f'$

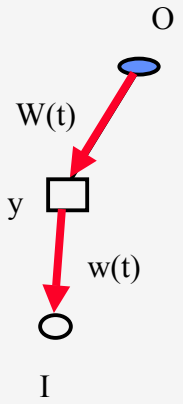
Backward pass phase

$$w(t+1) = w(t) - \eta \frac{dE}{dw(t)}$$

$$= w(t) - \eta \frac{dE}{dy}\frac{dy}{dw(t)}$$

$$= w(t) + \eta (d - o) f'(W(t) y) W(t) \frac{dy}{dw(t)}$$

$$= w(t) + \eta \Delta W(t) f'(w(t) I) I$$

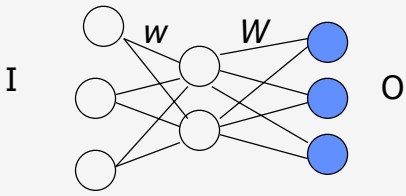


o = f (W(t) y)
= f (W(t) f(w(t) I))

General Three Layer Network

I inputs, *O* outputs, *w* connections between input and hidden units, *W* connections between hidden units and output , y is the activity of hidden unit

net (t) = network input to the unit at time t



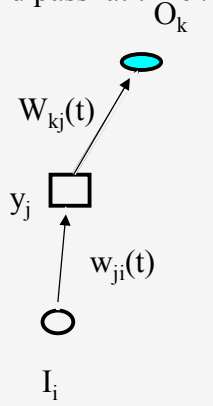
Forward pass

Weights are fixed during forward and backward pass at time *t*

1. Compute values for hidden units

$$net_j(t) = \sum_i w_{ji}(t) I_i(t)$$

$$y_j = f(net_j(t))$$



2. compute values for output units

$$Net_k(t) = \sum_j W_{kj}(t) y_j$$

$$O_k = f(Net_k(t))$$

Backward Pass

Recall delta rule , error measure for pattern n is

$$E(t) = \frac{1}{2}\sum_k (d_k(t) - O_k(t))^2$$

We want to know how to modify weights in order to decrease E where

$$w_{ij}(t+1) - w_{ij}(t) \propto -\frac{\partial E(t)}{\partial w_{ij}(t)}$$

both for hidden units and output units

This can be rewritten as product of two terms using chain rule

$$\frac{\partial E(t)}{\partial w_{ij}(t)} = \frac{\partial E(t)}{\partial net_j(t)} \bullet \frac{\partial net_j(t)}{\partial w_{ij}(t)}$$

both for hidden units and output units

Term A — How error for pattern changes as function of change in network input to unit *j*

Term B — How net input to unit *j* changes as a function of change in weight *w*

Summary

weight updates are local

$$w_{ji}(t+1) - w_{ji}(t) = \eta \delta_j(t) I_i(t)$$
$$W_{kj}(t+1) - W_{kj}(t) = \eta \Delta_k(t) y_j(t)$$

output unit

$$W_{kj}(t+1) - W_{kj}(t) = \eta \Delta_k(t) y_j(t)$$
$$= \eta (d_k(t) - O_k(t)) f'(Net_k(t)) y_j(t)$$

hidden unit

$$w_{ji}(t+1) - w_{ji}(t) = \eta \delta_j(t) I_i(t)$$
$$= \eta f'(net_j(t)) \sum_k \Delta_k(t) W_{kj} I_i(t)$$

Once weight changes are computed for all units, weights are updated at same time (bias included as weights here)

We now compute the derivative of the activation function $f(\ )$.

Activation Functions

• to compute Δ_k and δ_j we need to find the derivative of activation function f

• to find derivative the activation function *f* must be smooth

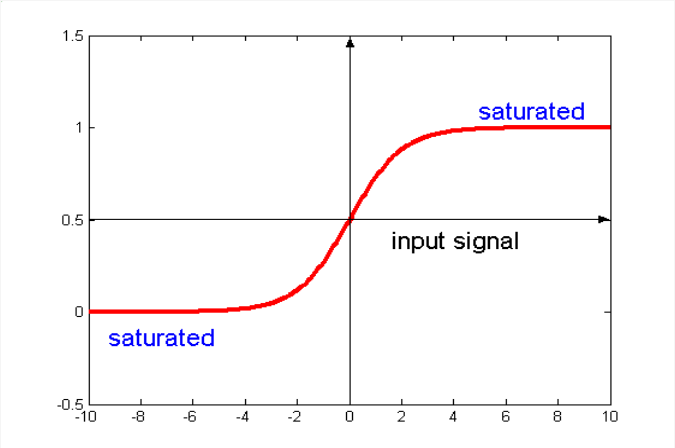
Sigmoidal (logistic) function-common in MLP

$$f\ (net_i(t)) = \frac{1}{1+\exp(-k net_i(t))}$$

where *k* is a positive constant. The sigmoidal function gives value in range of 0 to 1

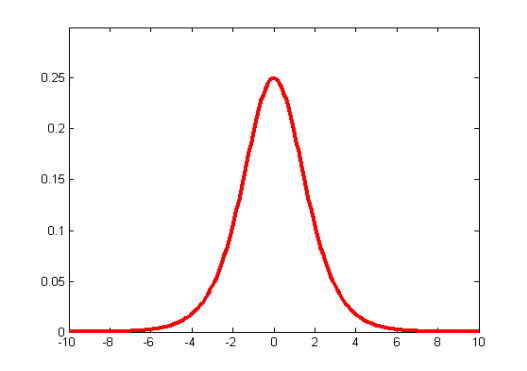
Input-output function of a neuron

Shape of sigmoidal function



Note: when net = 0, f = 0.5

Shape of sigmoidal function derivative



Derivative of sigmoidal function has max at x= 0, is symmetric about this point falling to zero as sigmoidal approaches extreme values

Returning to local error gradients in BP algorithm we have for output units

$$\Delta_i(t) = (d_i(t) - O_i(t))\, f'(Net_i(t)) = (d_i(t) - O_i(t))\, kO_i(t)(1 - O_i(t))$$

For hidden units we have

$$\delta_i(t) = f'(net_i(t)) \sum_k \Delta_k(t) W_{ki} = ky_i(t)(1 - y_i(t)) \sum_k \Delta_k(t) W_{ki}$$

Since degree of weight change is proportional to derivative of activation function, weight changes will be greatest when units receives mid-range functional signal than at extremes

Summary of BP learning algorithm

Set learning rate η

Set initial weight values (incl.. biases): w, W

Loop until stopping criteria satisfied:

present input pattern to input units
compute functional signal for hidden units
compute functional signal for output units

present Target response to output units
compute error signal for output units
compute error signal for hidden units
update all weights at same time
increment n to n+1 and select next I and d

end loop

Network training:

- Training set shown repeatedly until stopping criteria are met
- Each full presentation of all patterns = 'epoch'
- Randomize order of training patterns presented for each epoch in order to avoid correlation between consecutive training pairs being learnt (order effects)

Two types of network training:

- **Sequential mode** (on-line, stochastic, or per-pattern)
  Weights updated after each pattern is presented
- **Batch mode** (off-line or per -epoch)

LECTURE 6
MULTILAYER PERCEPTRON II

DYNAMICS OF MULTILAYER PERCEPTRON

Summary of Network Training

Forward phase: $\underline{I}(t), \underline{w}(t), \underline{net}(t), \underline{y}(t), \underline{W}(t), \underline{Net}(t), \underline{O}(t)$

Backward phase:

output unit

$$W_{kj}(t+1) - W_{kj}(t) = \eta \Delta_k(t) y_j(t)$$
$$= \eta (d_k(t) - O_k(t)) f'(Net_k(t)) y_j(t)$$

hidden unit

$$w_{ji}(t+1) - w_{ij}(t) = \eta \delta_j(t) I_i(t)$$
$$= \eta f'(net_j(t)) \sum_k \Delta_k(t) W_{kj}(t) I_i(t)$$

Network training:

Training set shown repeatedly until stopping criteria are met.

Possible convergence criteria are

- Euclidean norm of the gradient vector reaches a sufficiently small denoted as θ.
- When the absolute rate of change in the average squared error per epoch is sufficiently small denoted as θ.
- Validation for generalization performance : stop when generalization reaching the peak (illustrate in this lecture)

Goals of Neural Network Training

To give the correct output for input training vector (learning)

To give good responses to new unseen input patterns (Generalization)

Training and Testing Problems

• **Stuck neurons**: Degree of weight change is proportional to derivative of activation function, weight changes will be greatest when units receives mid-range functional signal than at extremes neuron. To avoid stuck neurons weights initialization should give outputs of all neurons approximate 0.5

•**Insufficient number of training patterns**: In this case, the training patterns will be learnt instead of the underlying relationship between inputs and output, i.e. network just memorizing the patterns.

•**Too few hidden neurons**: network will not produce a good model of the problem.

•**Over-fitting**: the training patterns will be learnt instead of the underlying function between inputs and output because of too many of hidden neurons. This means that the network will have a poor generalization capability.

Dynamics of BP learning

Aim is to minimise an error function over all training patterns by adapting weights in MLP

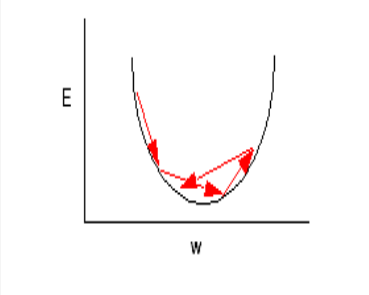
Recalling the typical error function is the mean squared error as follows

$$E(t)= \frac{1}{2}\sum_{k=1}^{p}(d_k(t)-O_k(t))^2$$

The idea is to reduce E(t) to global minimum point.

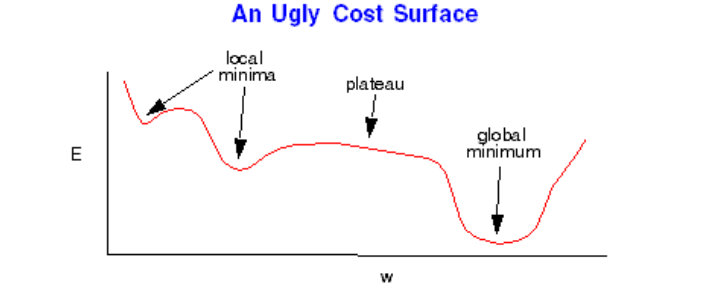
Dynamics of BP learning

In single layer perceptron with linear activation functions, the error function is simple, described by a smooth parabolic surface with a single minimum



Dynamics of BP learning

MLP with nonlinear activation functions have complex error surfaces (e.g. plateaus, long valleys etc.) with no single minimum



For complex error surfaces the problem is learning rate must keep small to prevent divergence. Adding momentum term is a simple approach dealing with this problem.

Momentum

•Reducing problems of instability while increasing the rate of convergence

•Adding term to weight update equation can effectively holds as exponentially weight history of previous weights changed

Modified weight update equation is

$$w_{ij}(n+1) - w_{ij}(n) = \eta \delta_j(n) y_i(n) + \alpha [w_{ij}(n) - w_{ij}(n-1)]$$

Effect of momentum term

• If weight changes tend to have same sign, momentum term increases and gradient decrease speed up convergence on shallow gradient

• If weight changes tend to have opposing signs, momentum term decreases and gradient descent slows to reduce oscillations (stabilizes)

• Can help escape being trapped in local minima

$$w^{ij}(n+1) - w^{ij}(n) = \eta \delta^j(n) y^i(n) + \alpha [w^{ij}(n) - w^{ij}(n-1)]$$

Selecting Initial Weight Values

• Choice of initial weight values is important as this decides starting position in weight space. That is, how far away from global minimum

• Aim is to select weight values which produce midrange function signals

• Select weight values randomly from uniform probability distribution

• Normalise weight values so number of weighted connections per unit produces midrange function signal

Convergence of Backprop

Avoid local minumum with fast convergence :

- Add momentum
- Train multiple nets with different initial weights

Nature of convergence

- Initialize weights 'near zero' or initial networks near-linear
- Increasingly non-linear functions possible as training progresses

Use of Available Data Set for Training

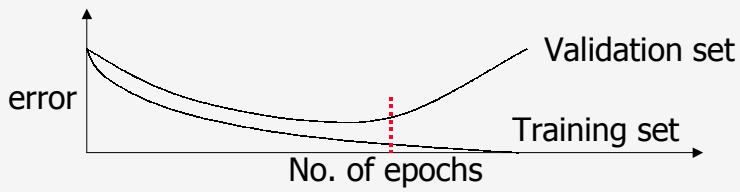
The available data set is normally split into three sets as follows:

- Training set – use to update the weights. Patterns in this set are repeatedly in random order. The weight update equation are applied after a certain number of patterns
- Validation set – use to decide when to stop training only by monitoring the error.
- Test set – Use to test the performance of the neural network. It should not be used as part of the neural network development cycle.

Earlier Stopping - *Good Generalization*

- Running too many epochs may overtrain the network and result in overfitting and perform poorly in generalization

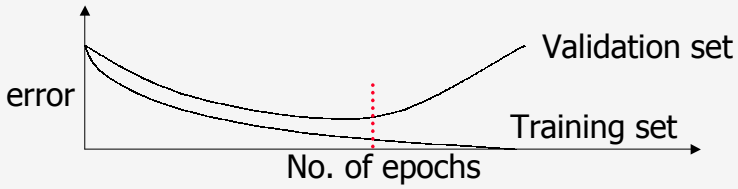
➢ Keep a hold-out validation set and test accuracy after every epoch. Maintain weights for best performing network on the validation set and stop training when error increases beyond this.



Model Selection by *Cross-Validation*

- Too few hidden units prevent the network from learning adequately fitting the data and learning the concept.
- Too many hidden units leads to overfitting.

➢ Similar cross-validation methods can be used to determine an appropriate number of hidden units, layers and nodes.



AN ALTERNATIVE TRAINING ALGORITHM

LECTURE 8 : GENETIC ALGORITHMS

History Background

- Idea of evolutionary computing was introduced in the 1960s by I. **Rechenberg** in his work "***Evolution strategies***" (*Evolutionsstrategie* in original). His idea was then developed by other researchers. **Genetic Algorithms** (GAs) were invented by John **Holland** and developed by him and his students and colleagues. This lead to Holland's book "*Adaptation in Natural and Artificial Systems*" published in 1975.
- In 1992 John **Koza** has used genetic algorithm to evolve programs to perform certain tasks. He called his method "**genetic programming**" (GP). LISP programs were used, because programs in this language can expressed in the form of a "parse tree", which is the object the GA works on.

Biological Background

Chromosome

- All living organisms consist of cells. In each cell there is the same set of **chromosomes**. Chromosomes are strings of DNA and serves as a model for the whole organism. A chromosome consist of **genes**, blocks of DNA. Each gene encodes a particular protein. Basically can be said, that each gene encodes a **trait**, for example color of eyes. Possible settings for a trait (e.g. blue, brown) are called **alleles**. Each gene has its own position in the chromosome. This position is called **locus**.
- Complete set of genetic material (all chromosomes) is called **genome**. Particular set of genes in genome is called **genotype**. The genotype is with later development after birth base for the organism's **phenotype**, its physical and mental characteristics, such as eye color, intelligence etc.

Biological Background

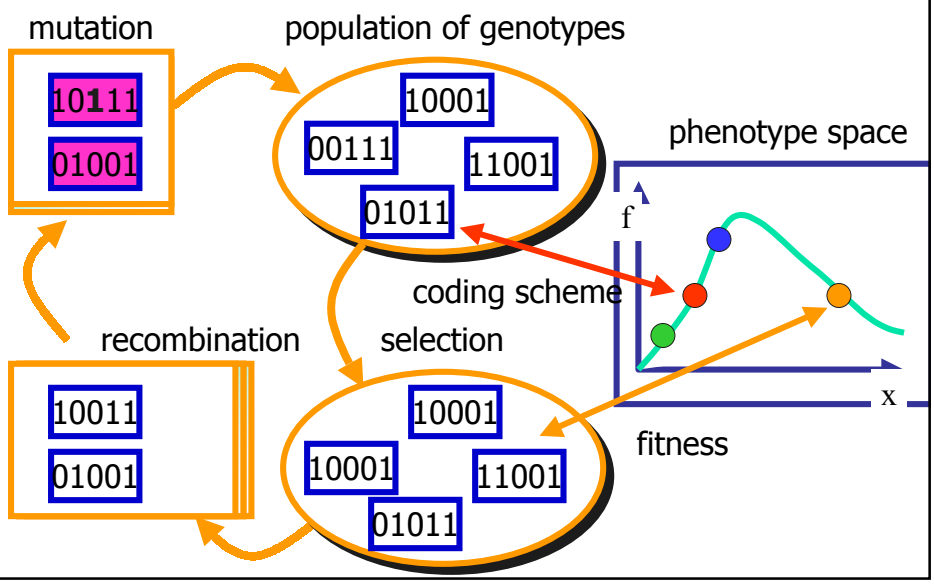
Reproduction

- During reproduction, first occurs **recombination** (or **crossover**). Genes from parents form in some way the whole new chromosome. The new created offspring can then be mutated. **Mutation** means, that the elements of DNA are a bit changed. This changes are mainly caused by errors in copying genes from parents.
- The **fitness** of an organism is measured by success of the organism in its life.

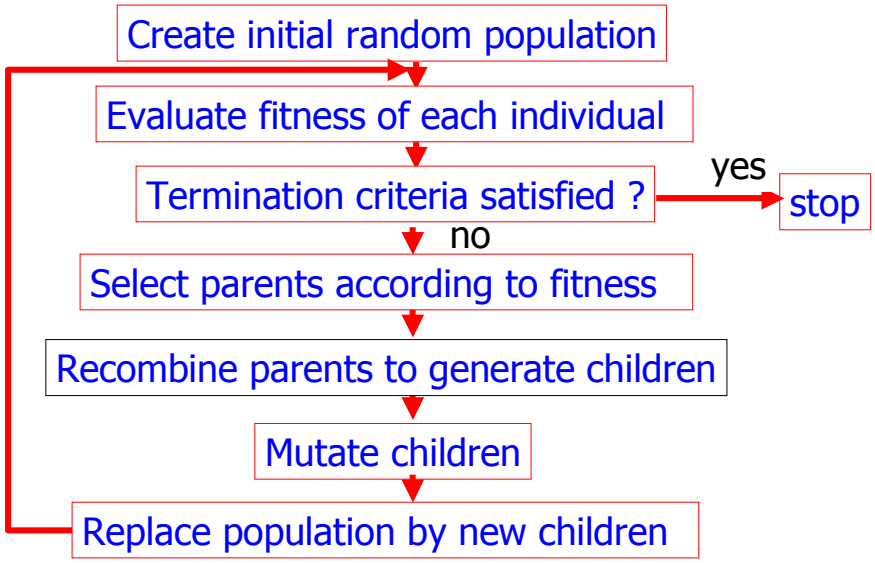
Evolutionary Computation

- Based on evolution as it occurs in nature
  - Lamarck, Darwin, Wallace: evolution of species, survival of the fittest
  - Mendel: genetics provides inheritance mechanism
    - Hence "genetic algorithms"
- Essentially a massively parallel search procedure
  - Start with random population of individuals
  - Gradually move to better individuals

Evolutionary Algorithms



Pseudo Code of an Evolutionary Algorithm



A Simple Genetic Algorithm

- optimization task : find the maximum of f(x)
  for example f(x)=x•sin(x) $x \in [0,\pi]$
- genotype: binary string $s \in [0,1]^5$ e.g. 11010, 01011, 10001
- mapping : genotype $\Longrightarrow$ phenotype
  binary integer encoding: $x = \pi \bullet \Sigma_{i=1}^{n=5} s_i \bullet 2^{n-i-1} / (2^n-1)$

Initial population

genotype	integ.	phenotype	fitness	prop. fitness
11010	26	2.6349	1.2787	30%
01011	11	1.1148	1.0008	24%
10001	17	1.7228	1.7029	40%
00101	5	0.5067	0.2459	6%

Some Other Issues Regarding Evolutionary Computing

- Evolution according to Lamarck
  - Individual adapts during lifetime
  - Adaptations inherited by children
  - In nature, genes don't change; but for computations we could allow this...
- Baldwin effect
  - Individual's ability to learn has positive effect on evolution
    - It supports a more diverse gene pool
    - Thus, more "experimentation" with genes possible

LECTURE 7
RADIAL BASIS FUNCTIONS

RADIAL BASIS FUNCTIONS

Radial-basis function (RBF) networks

So RBFs are functions taking the form

$$\phi\ (\|\ \underline{x}\ -\ \underline{x}_i\ \|)$$

where ϕ is a nonlinear activation function, $\underline{x}$ is the input and $\underline{x}_i$ is the *i'th* position, prototype, *basis* or *centre* vector.

The idea is that points near the centres will have similar outputs (i.e. if $\underline{x} \sim \underline{x}i$ then $f(\underline{x}) \sim f(\underline{x}i)$) since they should have similar properties.

The simplest is the *linear RBF* : $\phi(x) = ||\underline{x} - \underline{x}_i||$

Typical RBFs include

(a) Multiquadrics

$$\phi(r) = (r^2 + c^2)^{1/2}$$

for some c>0

(b) Inverse multiquadrics

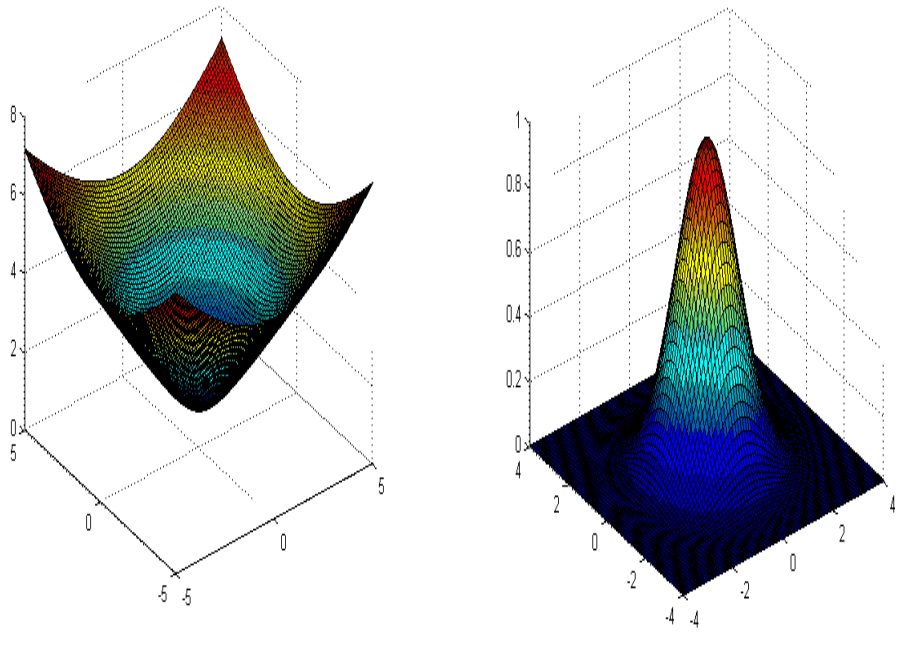
$$\phi(r) = (r^2 + c^2)^{-1/2}$$

for some c>0

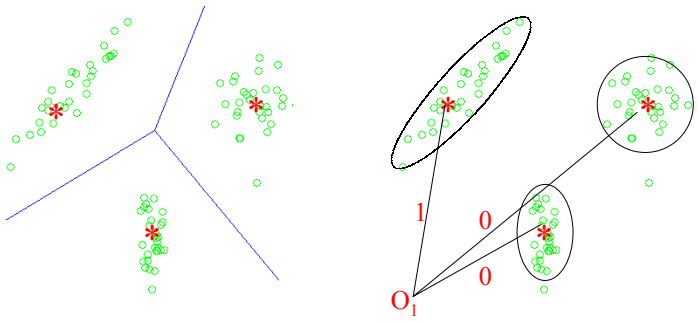
(c)Gaussian

$$\phi(r) = \exp(-\frac{r^2}{2\sigma^2})$$

for some σ >0

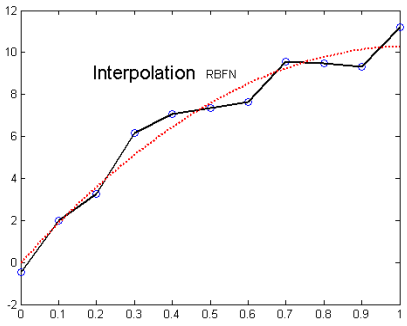


•Idea is to use a weighted sum of the outputs from the basis functions to represent the data.
•Thus centers can be thought of as prototypes of input data.



MLP distributed vs RBF local

Starting point: exact interpolation
Each input pattern x must be mapped onto a target value d



That is, given a set of N vectors $\underline{x}_i$ and a corresponding set of N real numbers, d_i (the targets), find a function F that satisfies the interpolation condition:

$$F(\underline{x}_i) = d_i \quad \text{for } i = 1,\ldots,N$$

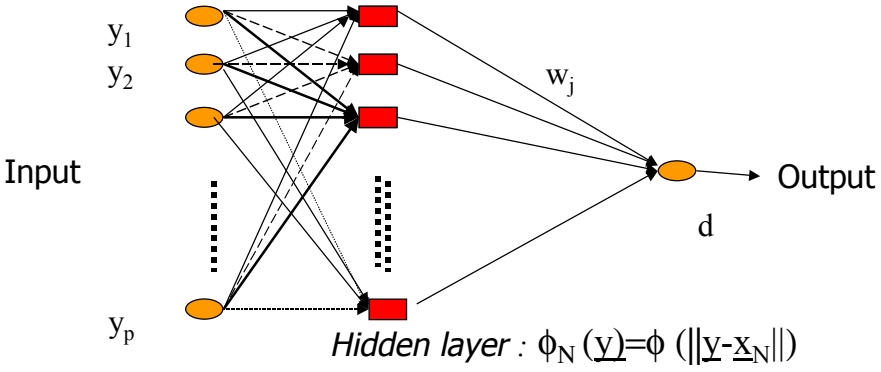
or more exactly find:

$$F(\underline{x}) = \sum_{j=1}^{N} w_j \phi(\|\underline{x} - \underline{x}_j\|)$$

satisfying:

$$F(\underline{x}_i) = \sum_{j=1}^{N} w_j \phi(\|\underline{x}_i - \underline{x}_j\|) = d_i$$

Three-layer networks



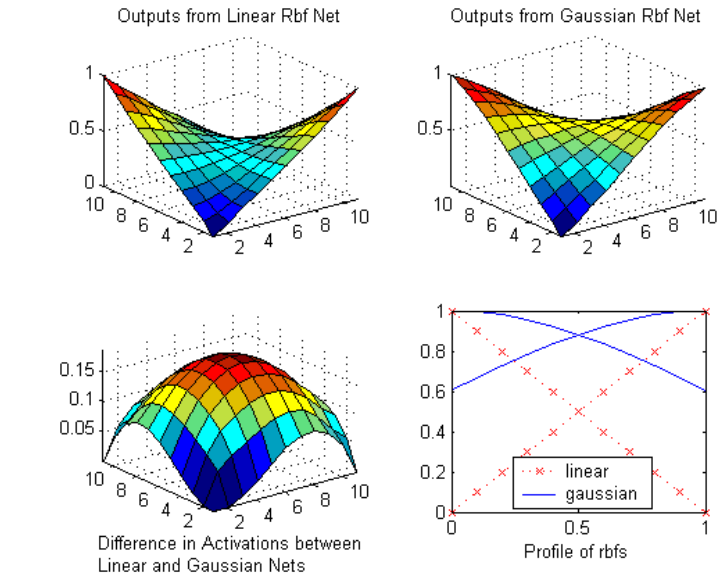
- output = $\Sigma\, w_i\, \phi_i\, (\underline{y})$
- adjustable parameters are weights w_j
- number of hidden units = number of data points
- Form of the basis functions *decided in advance*

To summarize:

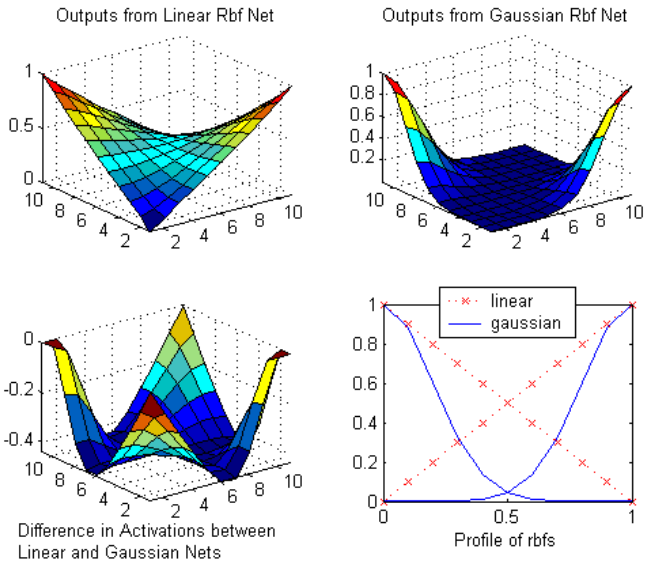
- For a given data set containing N points $(\underline{x}_i, d_i)$, i=1,…,N
- Choose a RBF function ϕ
- Calculate $\phi(\underline{x}_j - \underline{x}_i)$
- Solve the linear equation $\Phi\ \underline{W} = \underline{D}$
- Get the unique solution
- Done

• Like MLP's, RBFNs can be shown to be able to approximate any function to arbitrary accuracy (using an arbitrarily large numbers of basis functions).
• Unlike MLP's, however, RBFNs have the property of 'best approximation' i.e. there exists an RBFN with minimum approximation error.

Large $\sigma = 1$



Small σ = 0.2



Problems with exact interpolation

can produce poor generalisation performance as only data points constrain mapping

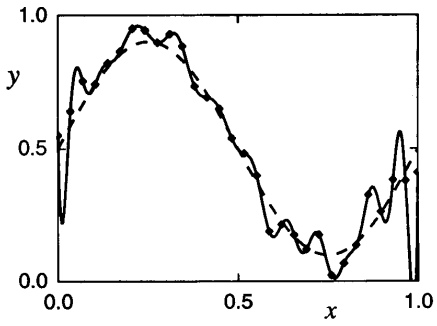
- **Overfitting problem**

Bishop(1995) example

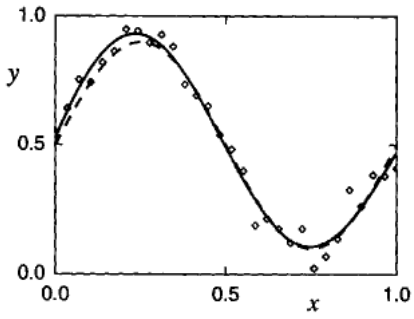
Underlying function $f(x)=0.5+0.4\sin(2\pi x)$
sampled randomly for 30 points

- added Gaussian noise to each data point
- 30 data points, 30 hidden RBF units

Fits all data points but creates oscillations due added noise and unconstrained between data points



All Data Points



5 Basis functions

To fit an RBF to every data point is very inefficient due to the computational cost and is very bad for generalization so:

- Use less RBF's than data points I.e. M<N
- Therefore don't necessarily have RBFs centred at data points
- Can include bias terms
- Can have Gaussian with general covariance matrices but there is a trade-off between complexity and the number of parameters to be found.

APPLICATION EXAMPLES

LECTURE 9:
NONLINEAR IDENTIFICATION, PREDICTION AND CONTROL

Nonlinear System Identification

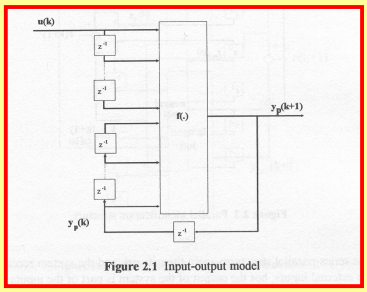
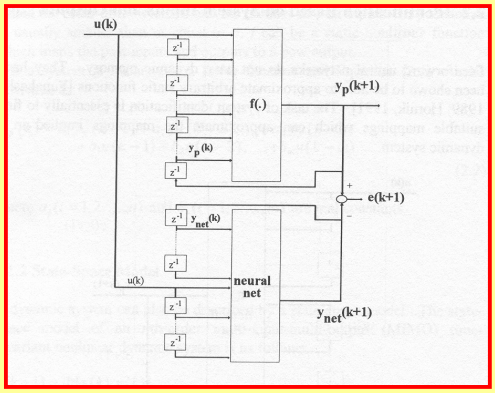
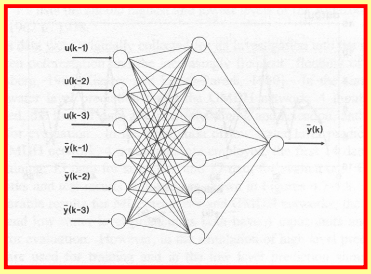


Figure 2.1 Input-output model



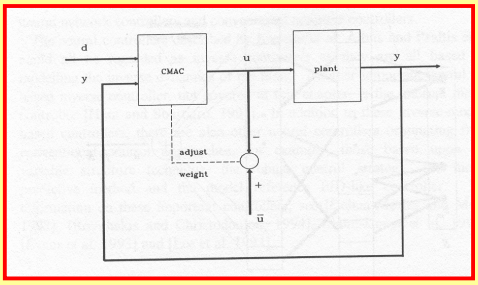


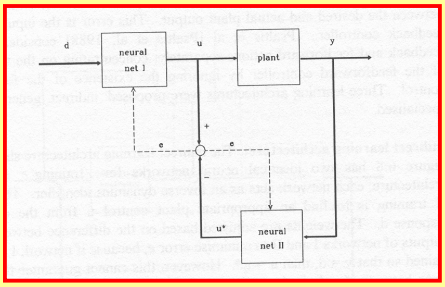
Target function: $y_p(k+1) = f(.)$
Identified function: $y_{NET}(k+1) = F(.)$
Estimation error: $e(k+1)$

Issues in Nonlinear System Identification

- Data-Set Selection and Pre-filtering
- Delays' or Orders' Selection
- Input and Hidden Layers Sets Choice
- Activation Functions' Selection
- Neural Network "Structure" Choice
- Neural Network Test & Validation

Nonlinear System Neural Control

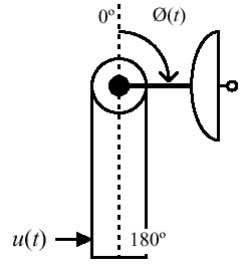




d: *reference/desired response*
y: *system output/desired output*
u: *system input/controller output*
ū: *desired controller input*
u*: *NN output*
e: *controller/network error*

The goal of training is to find an appropriate plant control u *from the desired response* d. *The weights are adjusted based on the difference between the outputs of the networks I & II to minimise* e. *If network I is trained so that* y = d, *then* u = u*. *Networks act as inverse dynamics identifiers.*

Nonlinear System Identification



$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ 9.81\sin x_1 - 2x_2 + u \end{bmatrix}$$

$$x_1 = \emptyset$$

$$x_2 = \frac{d\emptyset}{dt}$$

```
deg2rad = pi/180;
angle = [-20:40:200]*deg2rad;
vel = [-90:36:90]*deg2rad;
force = -30:6:30;
```

```
angle2 = [-20:10:200]*deg2rad;
Pm = [combvec(angle,vel,force);
  [angle2; zeros(2,length(angle2))]];
```

Neural network input generation Pm

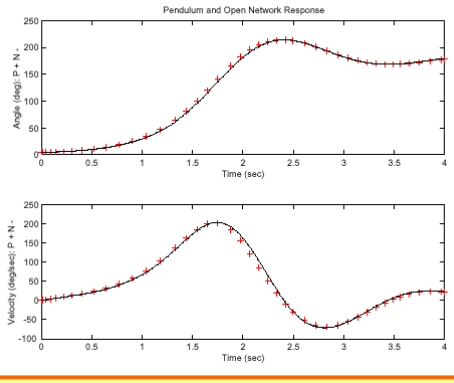
Nonlinear System Identification

```
S1 = 8;
[S2,Q] = size(Tm);
mnet = newff(minmax(Pm),[S1 S2],{'tansig' 'purelin'},'trainlm');
```

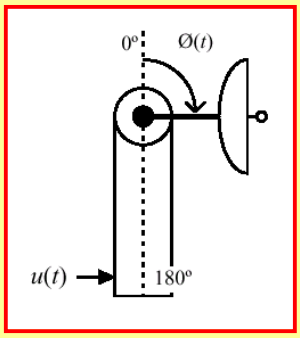
```
mnet.trainParam.goal = (0.0037^2);
mnet = train(mnet,Pm,Tm);
```

Neural network target Tm

Neural network response (angle & velocity)



Model Reference Control



$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ 9.81\sin x_1 - 2x_2 + u \end{bmatrix}$$

$$x_1 = \varnothing$$

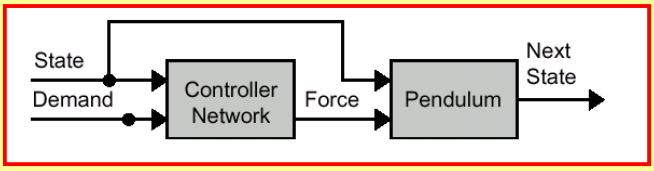
$$x_2 = \frac{d\varnothing}{dt}$$

Antenna arm nonlinear model

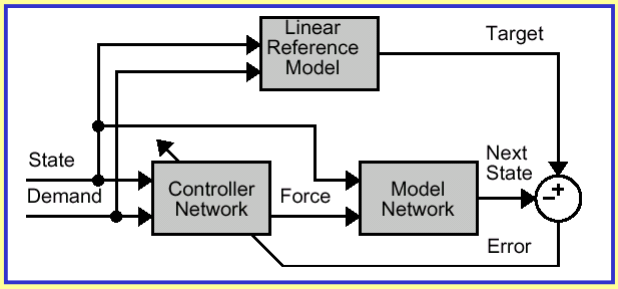
Linear reference model

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -9x_1 - 6x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 9r \end{bmatrix}$$

Model Reference Control

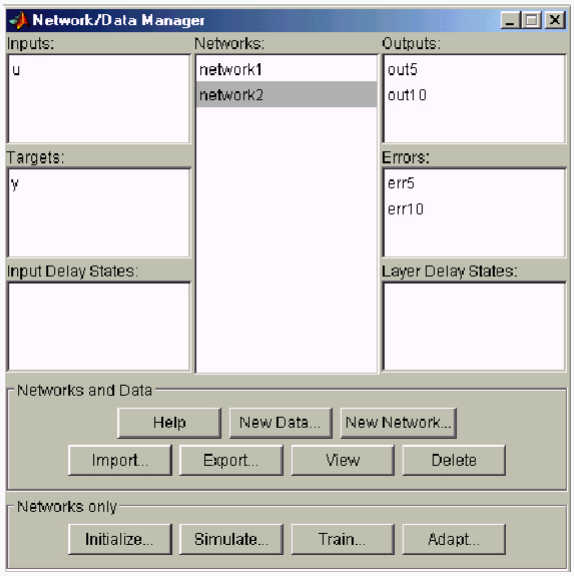


Neural controller + nonlinear system diagram



Neural controller, reference model, neural model

Matlab NNtool GUI (Graphical User Interface)



Model Reference Controller