Function approximation

- Linear Networks vs Nonlinear Networks
- Multilayer Neural Networks
- Projective basis functions

Hyper-plane fitting: Data creation

$$y[t] = f(x_1[t], x_2[t]) + n[t]$$
$$= 2 * x_1[t] - x_2[t] + n[t]$$

$\{\mathbf{x}[t] = (x_1[t], x_2[t])\}_t$ is a uniform sample

n[t] is noise

Exercise

- Create paired data, $\{(x[t],y[t])\}_t$ using the rule

$$y[t] = a_1 * x_1[t] + a_2 x_2[t] + b + n[t]$$

where $a=[a_1\ a_2]$ and b are constant

- Plot data and the hyper-plane in a 3D figure

Fitting hyper-plane

- Input

$$x = \begin{pmatrix} x_1[1] & x_1[2] & \cdots & x_1[N] \\ x_2[1] & x_2[2] & & x_2[N] \end{pmatrix}$$

$$y = \begin{pmatrix} y[1] & y[2] & \cdots & y[N] \end{pmatrix}$$

Fitting hyper-plane

- Form data matrix

$$A = \begin{pmatrix} x_1[1] & x_1[2] & & x_1[N] \\ x_2[1] & x_2[2] & \cdots & x_2[N] \\ 1 & 1 & & 1 \end{pmatrix}$$

- Form c=[x*y';sum(y)]

- Determine a hyper-plane by solving

$$(A * A')\begin{pmatrix} a_1 \\ a_2 \\ b \end{pmatrix} = c$$

$$d = inv(A * A') * c$$

$$a_1 = d(1); a_2 = d(2); b = d(3)$$

- Plot the hyper-plane
- Calculate the fitting error

$$E(a_1, a_2, b)$$

$$= \frac{1}{N}\sum_t (y[t] - (a_1 * x_1[t] + a_2 x_2[t] + b))^2$$

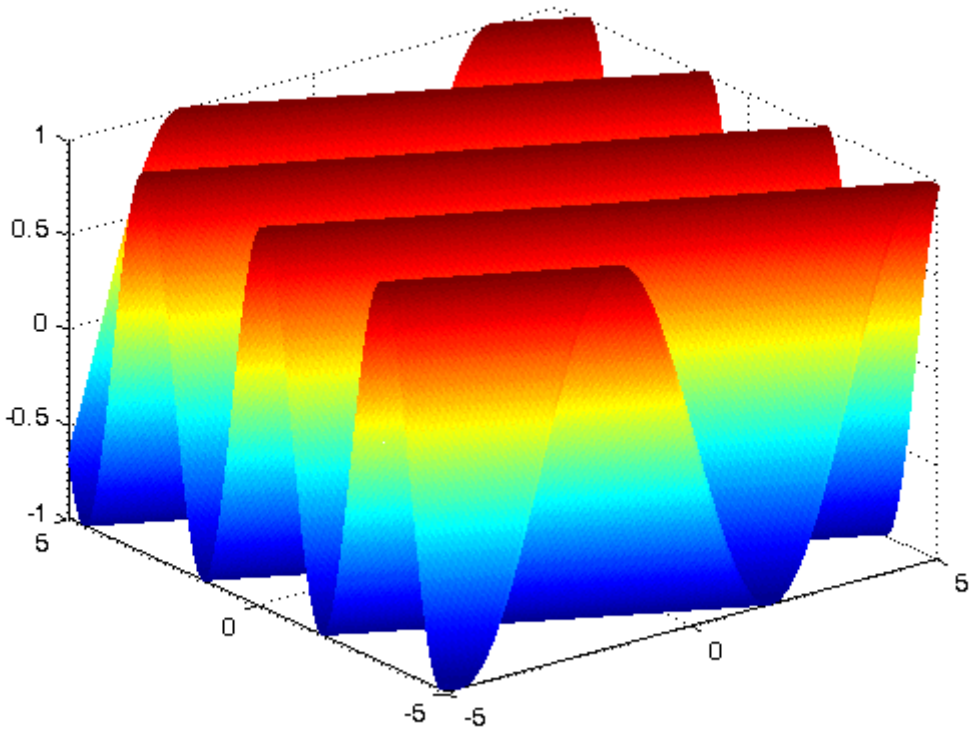
- Create paired data by the rule

$$y[t] = \cos(a_1 * x_1[t] + a_2 x_2[t] + b) + n[t]$$

- Fit hyper-plane
- Output the fitting error

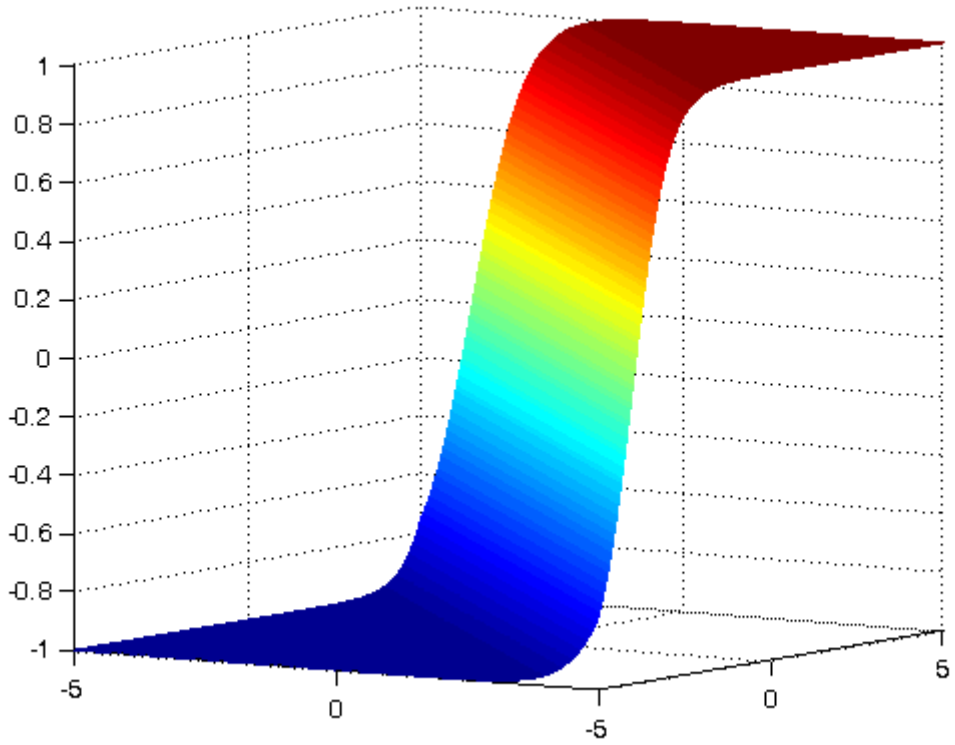
Post-cos projection

- $$y = \cos(a_1 * x_1 + a_2 x_2 + b)$$



Post-tanh projection

$$y = \tanh(a_1 * x_1 + a_2 x_2 + b)$$



Weighted post-tanh projections

- Plot

$$y = r_1 \tanh(x_1 + x_2 + 1) + r_2 \tanh(0.5 * x_1 - x_2)$$

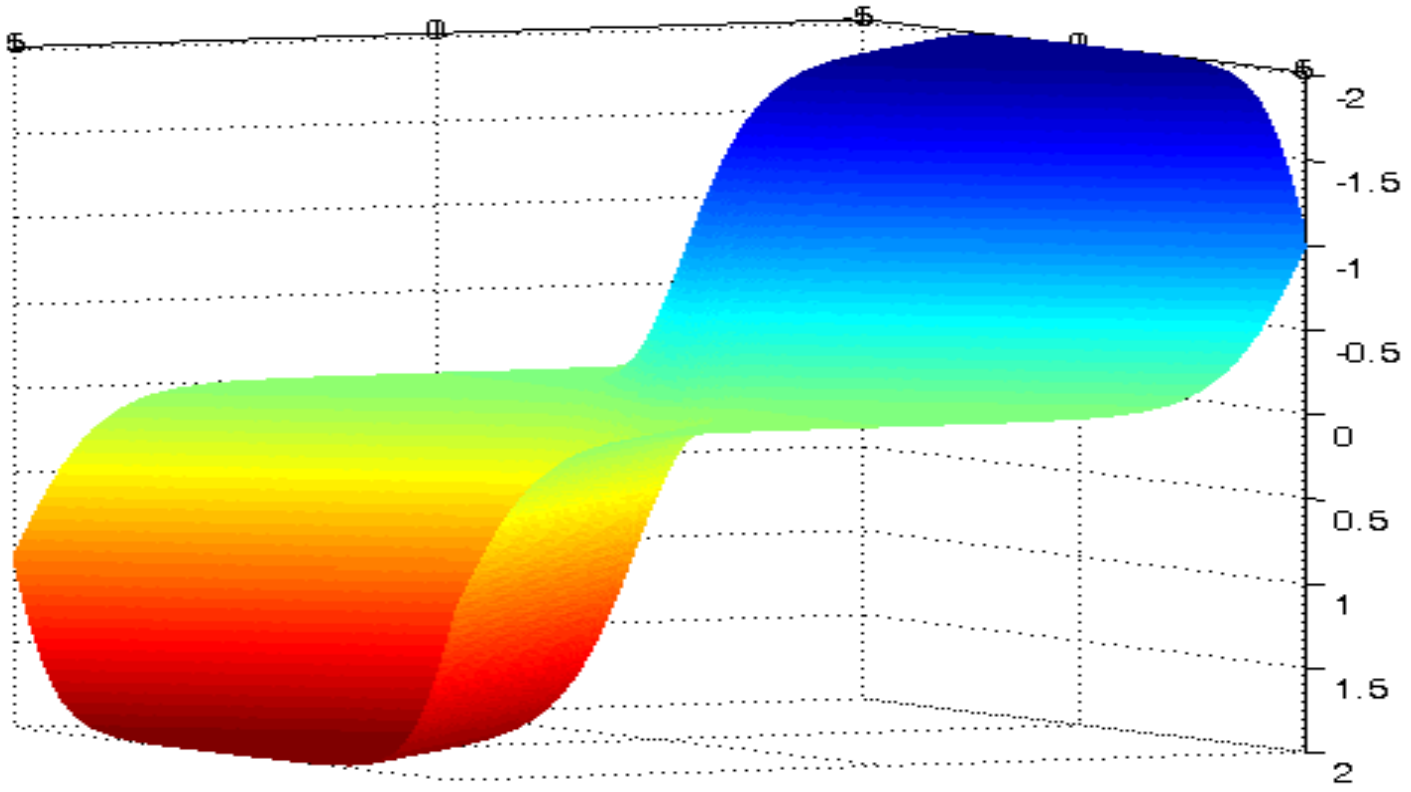
Projective basis function

- M=2

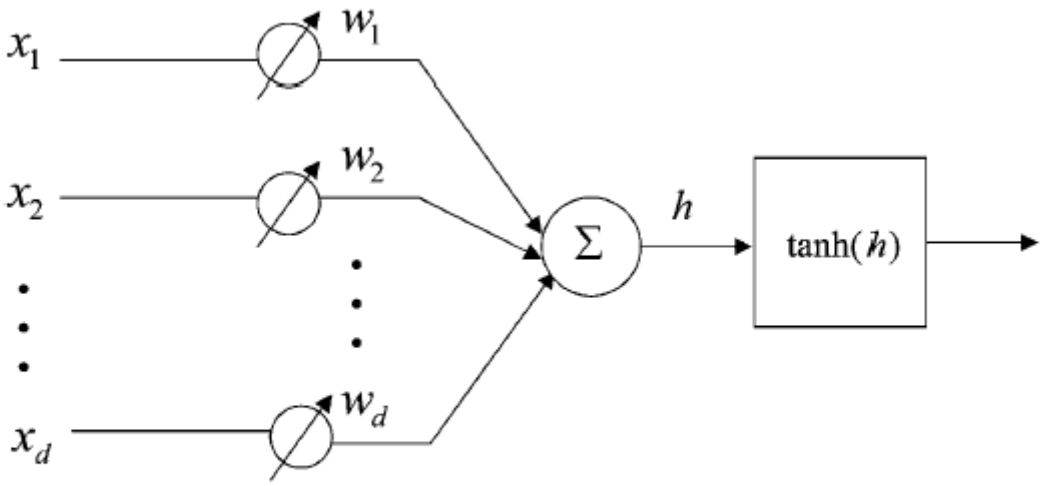
$$y = F(\mathbf{x};\theta) = \sum_{m=1}^{2} r_m \tanh(\mathbf{a}_m^T \mathbf{x} - b_m)$$

$$\theta = \{r_m, \mathbf{a}_m, b_m\}_m$$

Weighted post-tanh projections

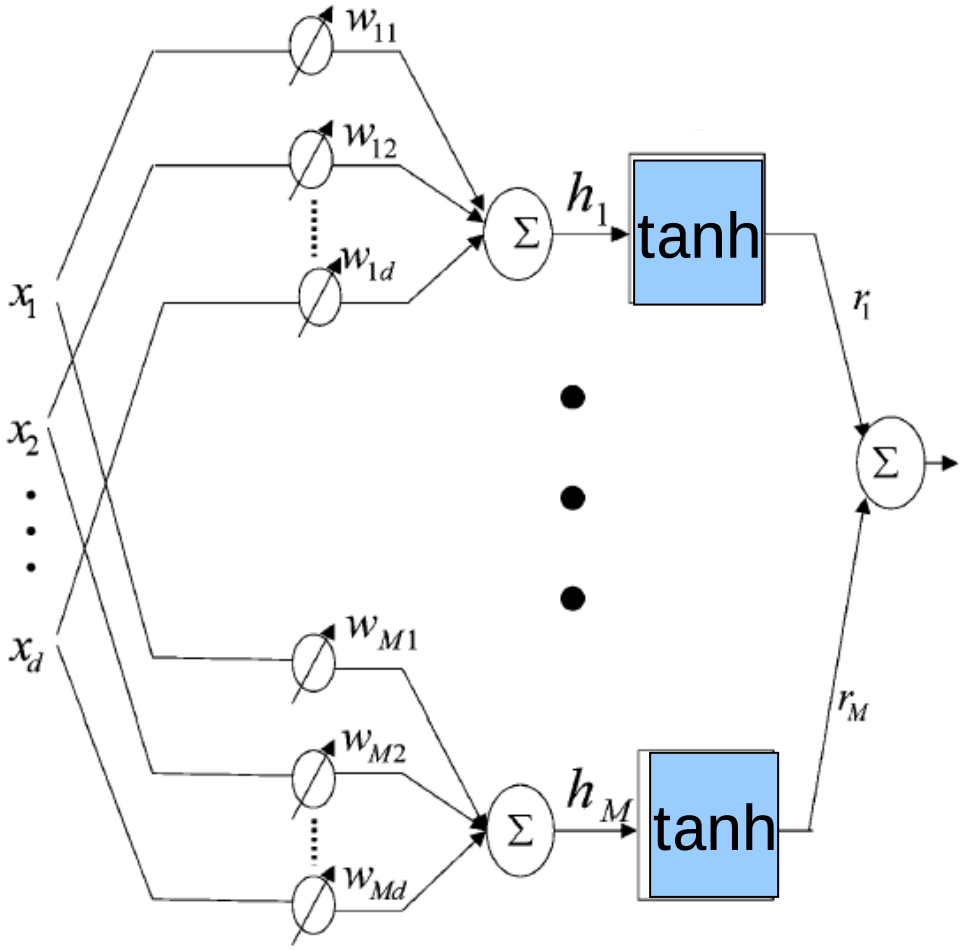


Post-tanh projection

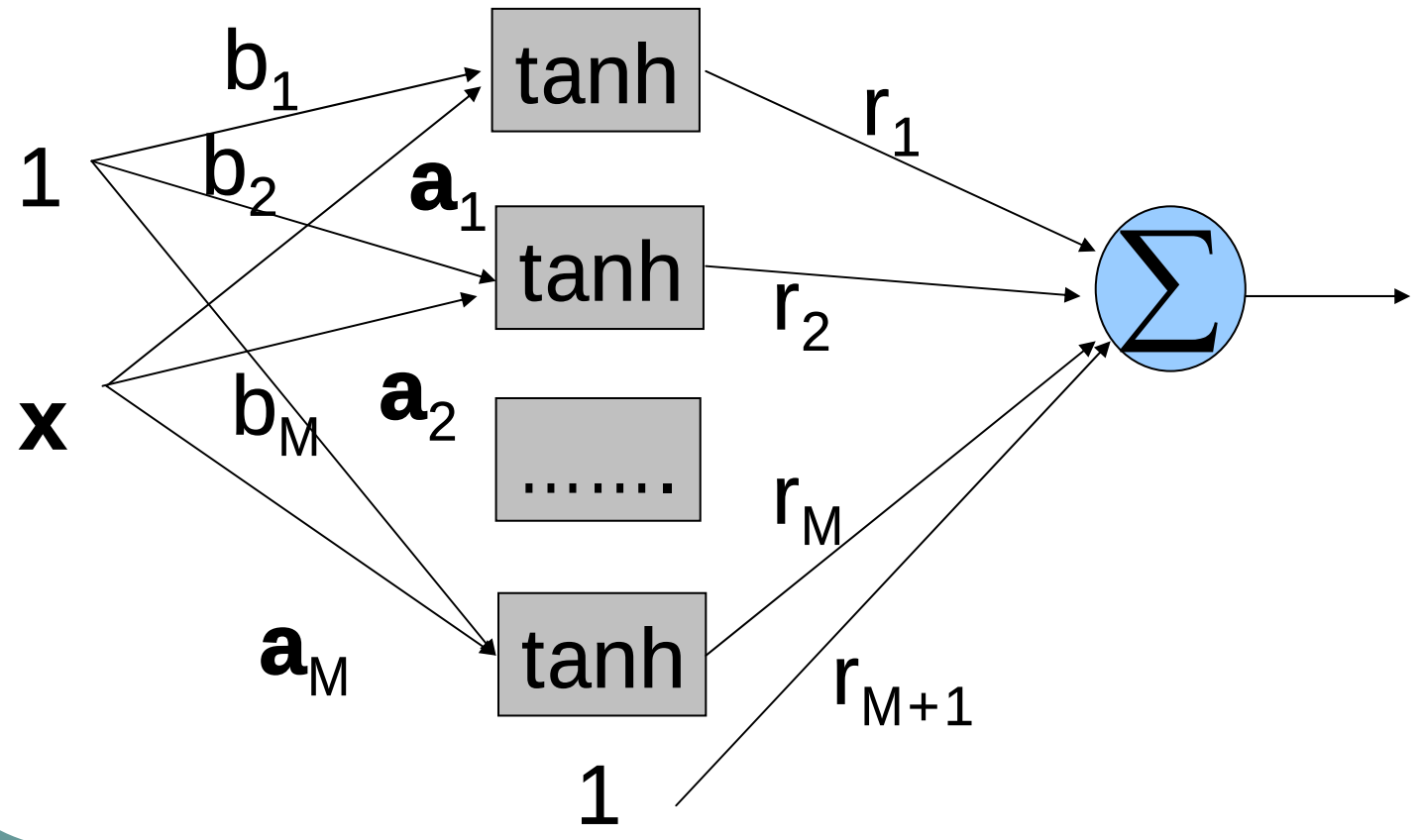


Perceptron.

MLP networks



MLP Networks



Network function

$$f(\mathbf{x};\theta) = \sum_{m=1}^{M} r_m \tanh(\mathbf{a}_m^T \mathbf{x} + b_m) + r_0$$

$$\theta = \{\mathbf{a}_m\} \cup \{b_m\} \cup \{r_m\}$$

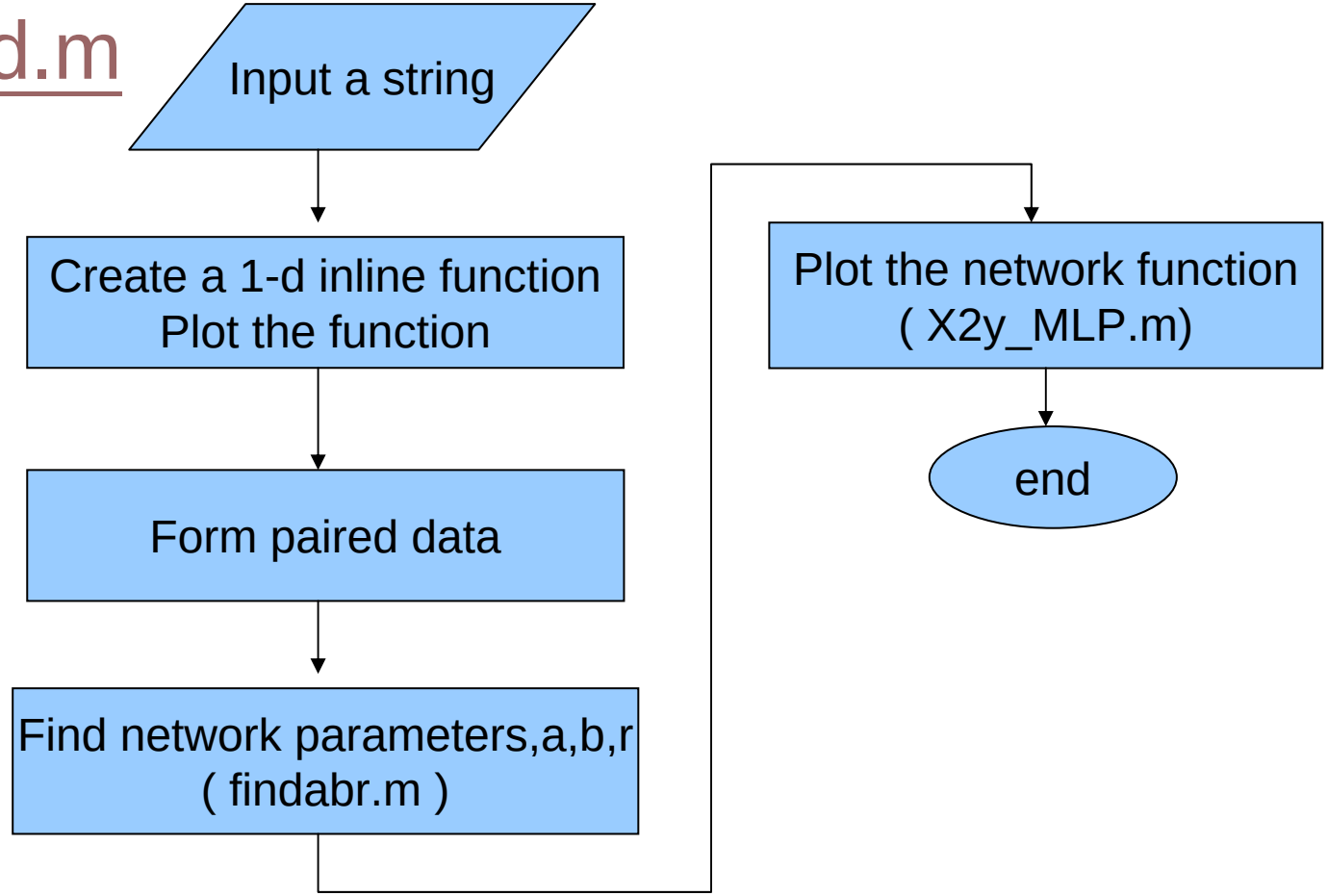
Install NNSYSID

1. Install The NNSYSID Toolbox
2. Download findabr.m x2y_MLP.m

   (Levenberg Marquardt method)
3. Set path to recruit the directory of where NNSYSID is installed

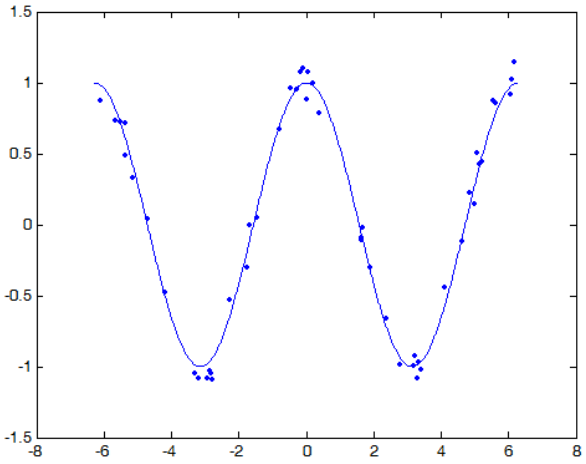
1-dimensional function approximation

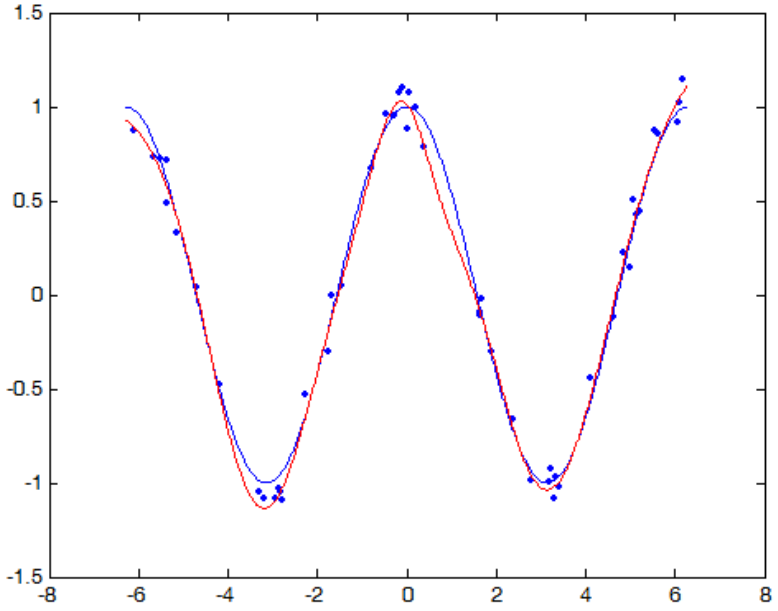
fa1d.m



Example fa1d

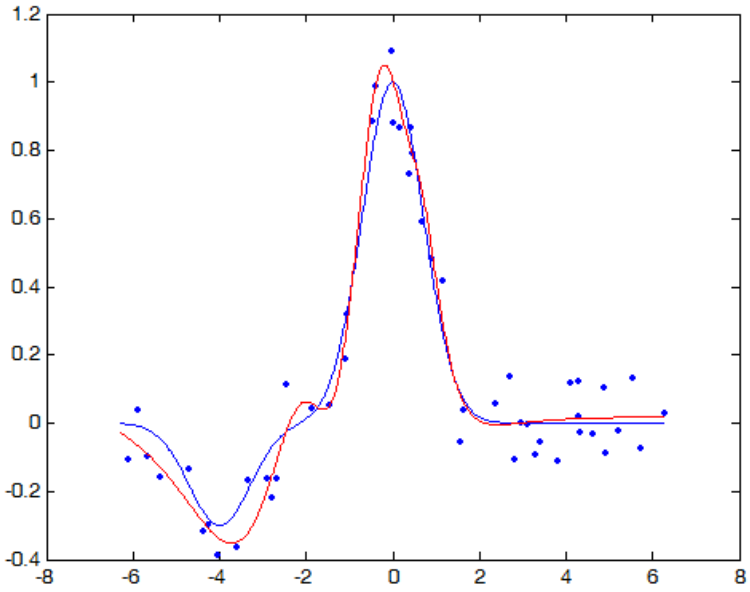
input a 1D function: x.^2+cos(x) :cos(x)
keyin sample size:50
keyin the number of hidden units:21





Example

>> fa1d
input a 1D function: x.^2+cos(x) :exp(-x.^2)-0.3*exp(-(x+4).^2)
keyin sample size:50
keyin the number of hidden units:21



Function approximation

- Unknown Target Function, F
  - Predictors to targets
  - Mapping R^d to R
- Paired data, $(x_i\, y_i)$
  - x_i belongs R^d, representing a predictor
  - All x_i are a sample from the function domain
  - $y_i = F(x_i) + n$
  - n denotes noise

Data driven function approximation

- Given paired data, find an approximating function f to minimize

$$\mathrm{E}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i}(y_i - f(\mathbf{x}_i;\boldsymbol{\theta}))^2$$

- E denotes the mean square error of approximating desired targets by network outputs

Traditional methods

- Numerical methods focuses on one-dimensional target function
  - Polynomial interpolation
  - Spline interpolation

High dimensional target functions

Table 1. Target functions

$f_1(\mathbf{x}) = \sin(x_1 + x_2)$	
$f_2(\mathbf{x}) = x_1^2 + x_2^2$	
$f_3(\mathbf{x}) = 0.5x_1^2 - 0.9x_2^2$	
$f_4(\mathbf{x}) = \exp(-0.05x_1^2 - 0.09x_2^2)$	noise
$f_5(\mathbf{x}) = \sin([1,-1]^T x) + \exp(-x^T A x)$	$A = \begin{bmatrix} 0.08 & 0.015 \\ 0.02 & 0.075 \end{bmatrix}$
$f_6(\mathbf{x}) = \tanh(0.8x_1 + 0.2x_2) + \tanh(0.3x_1 - 0.9x_2)$	
$f_7(\mathbf{x}) = 0.5\sin(x_1 + x_2) + 0.2x_1 - 0.2x_2$	
$f_8(\mathbf{x}) = \exp(-(\mathbf{x}-\mathbf{w}_1)^T A(\mathbf{x}-\mathbf{w}_1)) + \exp(-(\mathbf{x}-\mathbf{w}_2)^T B(\mathbf{x}-\mathbf{w}_1))$	$B = \begin{bmatrix} 0.12 & -0.02 \\ 0.035 & 0.075 \end{bmatrix}$
$f_9(\mathbf{x}) = f_8(\mathbf{x}) + 0.5sin(x_1 + 0.3x_2) + 0.5sin(0.2x_1 - 0.8x_2)$	

Fa2D

fa2d.m

Calling findabr →

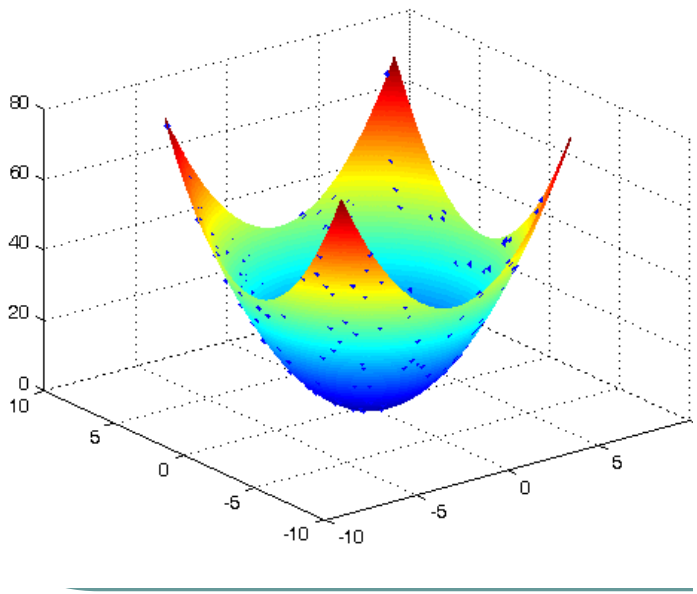
Calling x2y_MLP →

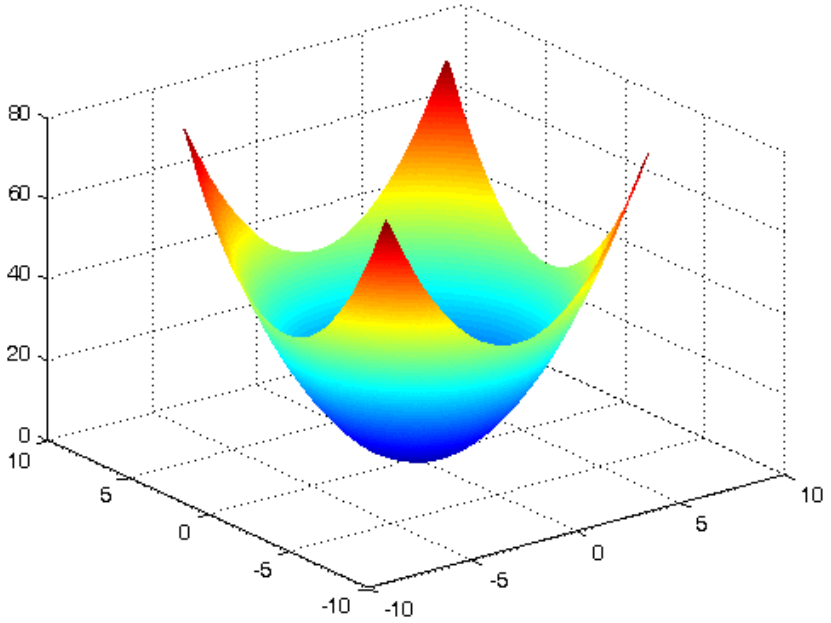
```
M=input('keyin the number of hidden units:');

%%%%%%%%%%%%%%%%%%%%%%%%
[a,b,r]=findabr(x,y,M);
x1=-range:0.02:range;
x2=x1;
for i=1:length(x1)
        x_test = [x1(i)*ones(1,length(x2));x2]';
        y_test=fx(x_test(1,:),x_test(2,:));
        %%%%%%%%%%%%%%%%%%%%%%%%
        y=x2y_MLP(x_test,r,a,b,M);
        C(i,:)=y';
```

example

>> fa2d
input a 2D function: x1.^2+x2.^2+cos(x1) :x1.^2+x2.^2
keyin sample size:50
keyin the number of hidden units:21



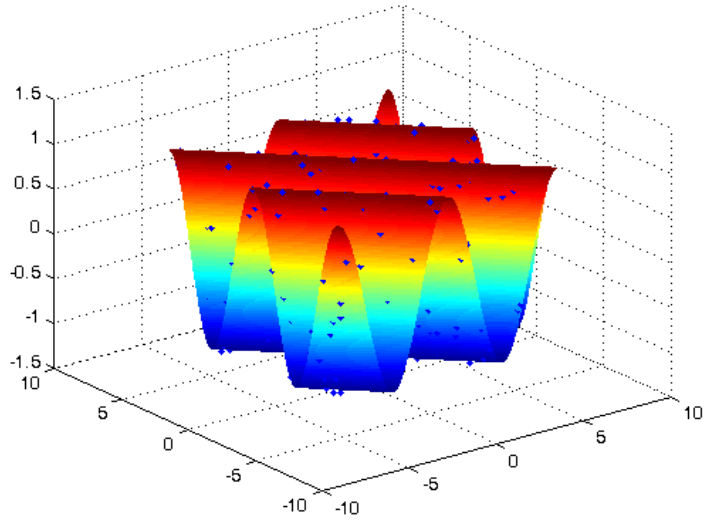


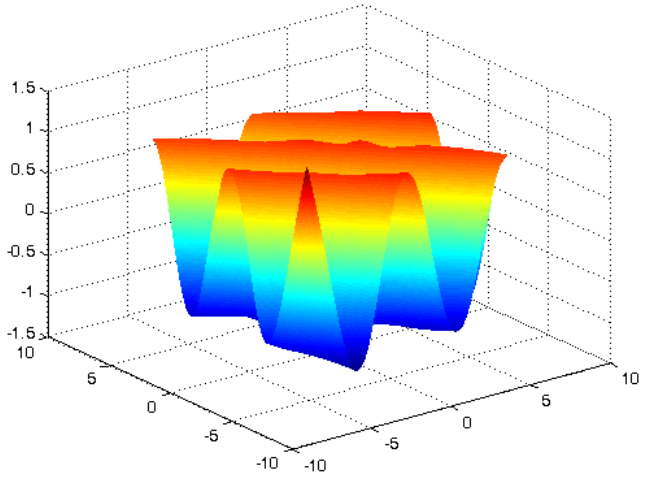
Mesh

```
clear all
fstr=input('input a 2D function: x1.^2+x2.^2+cos(x1) :','s');
fx=inline(fstr);
range=5;
x1=-range:0.02:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=fx(x1(i),x2);
end
mesh(x1,x2,C);
```

Example

>> fa2d
input a 2D function: x1.^2+x2.^2+cos(x1) :cos(x1+x2)
keyin sample size:300
keyin the number of hidden units:31





Target function II

TABLE III
TARGET FUNCTIONS

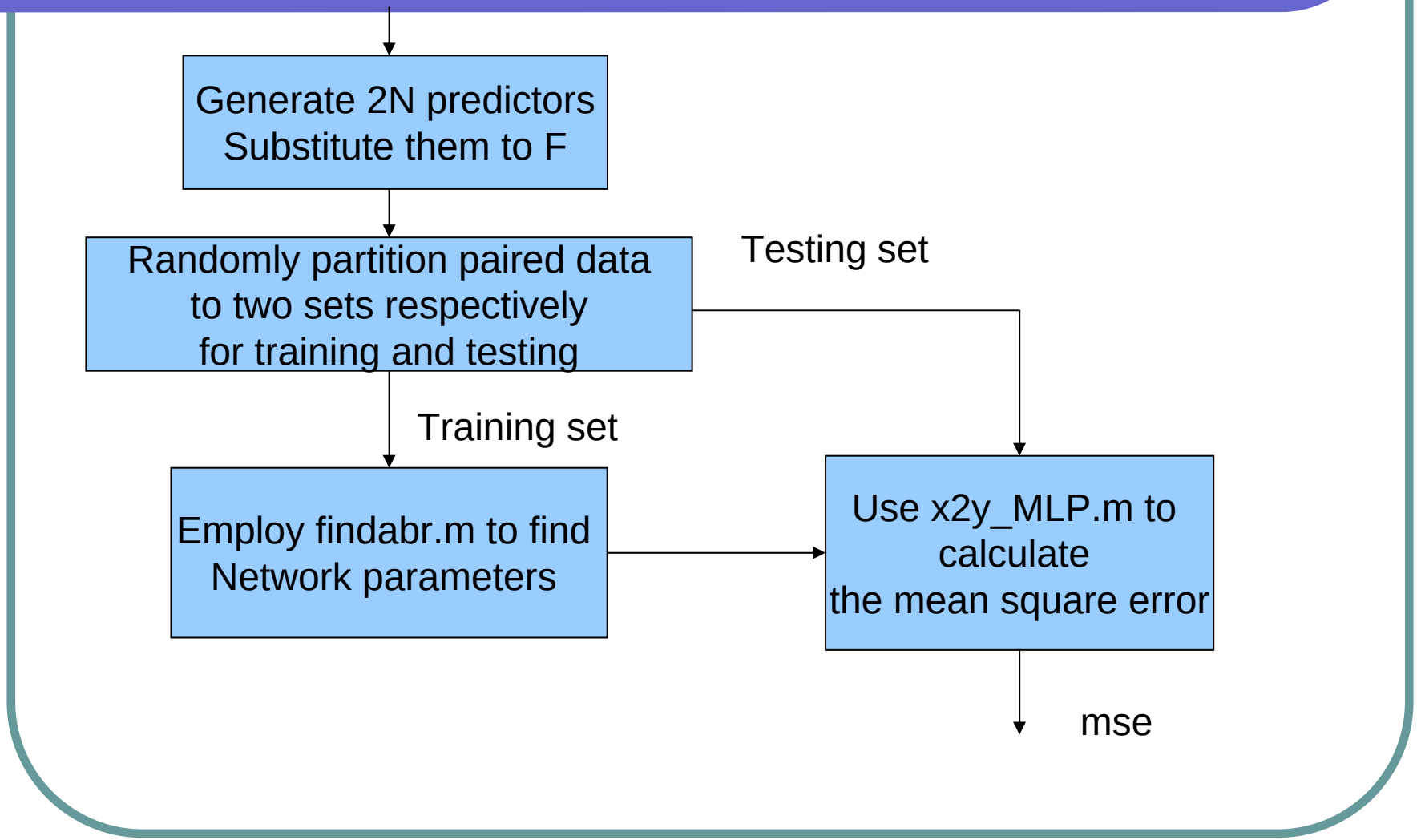
Target function	Description
$y_1(z) = \psi(z) + n$	One-dimensional function with uniform noise $n \in [-0.5, 0.5]$
$y_2(\mathbf{z}) = 0.3z_1 + z_2 - 0.5z_3 + 2z_4 - 0.7z_5$	Single linear projection
$y_3(\mathbf{z}) = \tanh(0.3z_1 + z_2 - 0.5z_3 + 2z_4 - 0.7z_5)$	Single post-nonlinear projection
$y_4(\mathbf{z}) = \tanh(0.8z_1 + 0.2z_2) + \tanh(0.3z_1 - 0.9z_2)$	Sum of two PNL projections
$y_5(\mathbf{z}) = \tanh(0.8z_1 + 0.2z_2 + 0z_3 + 0z_4 + 0z_5)$ $+ \tanh(0.3z_1 - 0.9z_2 + 0z_3 + 0z_4 + 0z_5)$	Sum of two PNL projections with ineffective attributes
$y_6(\mathbf{z}) = 0.5z_1^2 - 0.9z_2^2$	Quadratic function
$y_7(\mathbf{z}) = \exp(-\frac{z_1^2}{\sigma_1} - \frac{z_2^2}{\sigma_2}) \cos(\kappa z_1 + \phi)$	Gabor function

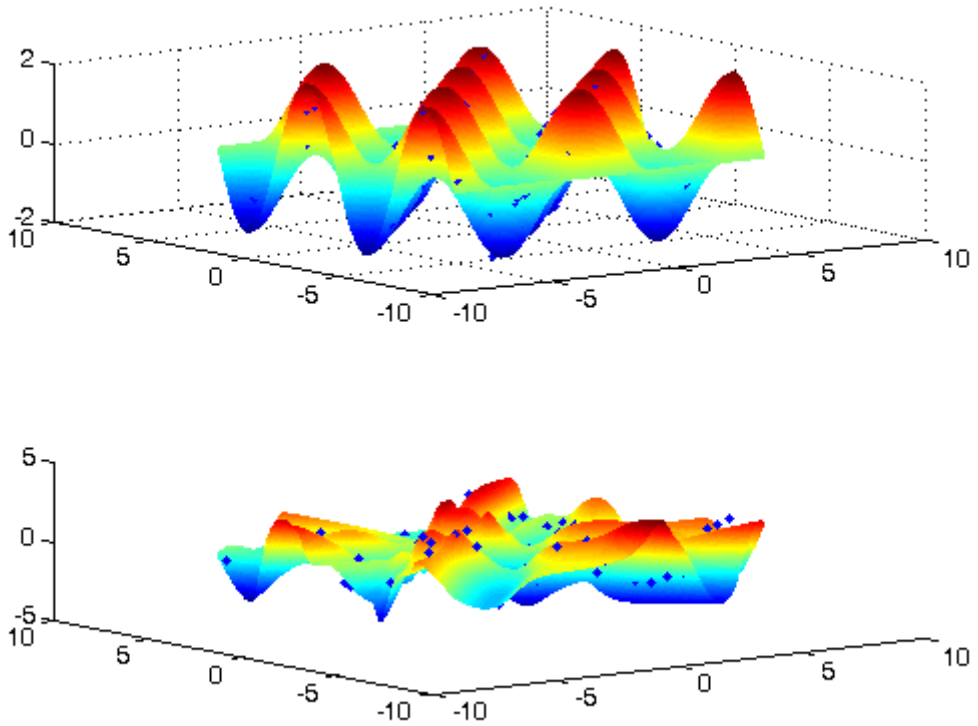
Performance evaluation

TABLE V
PERFORMANCE OF THE RELEVANT LEARNING METHODS FOR APPROXIMATING THE SINUSOIDAL GRATING FUNCTION

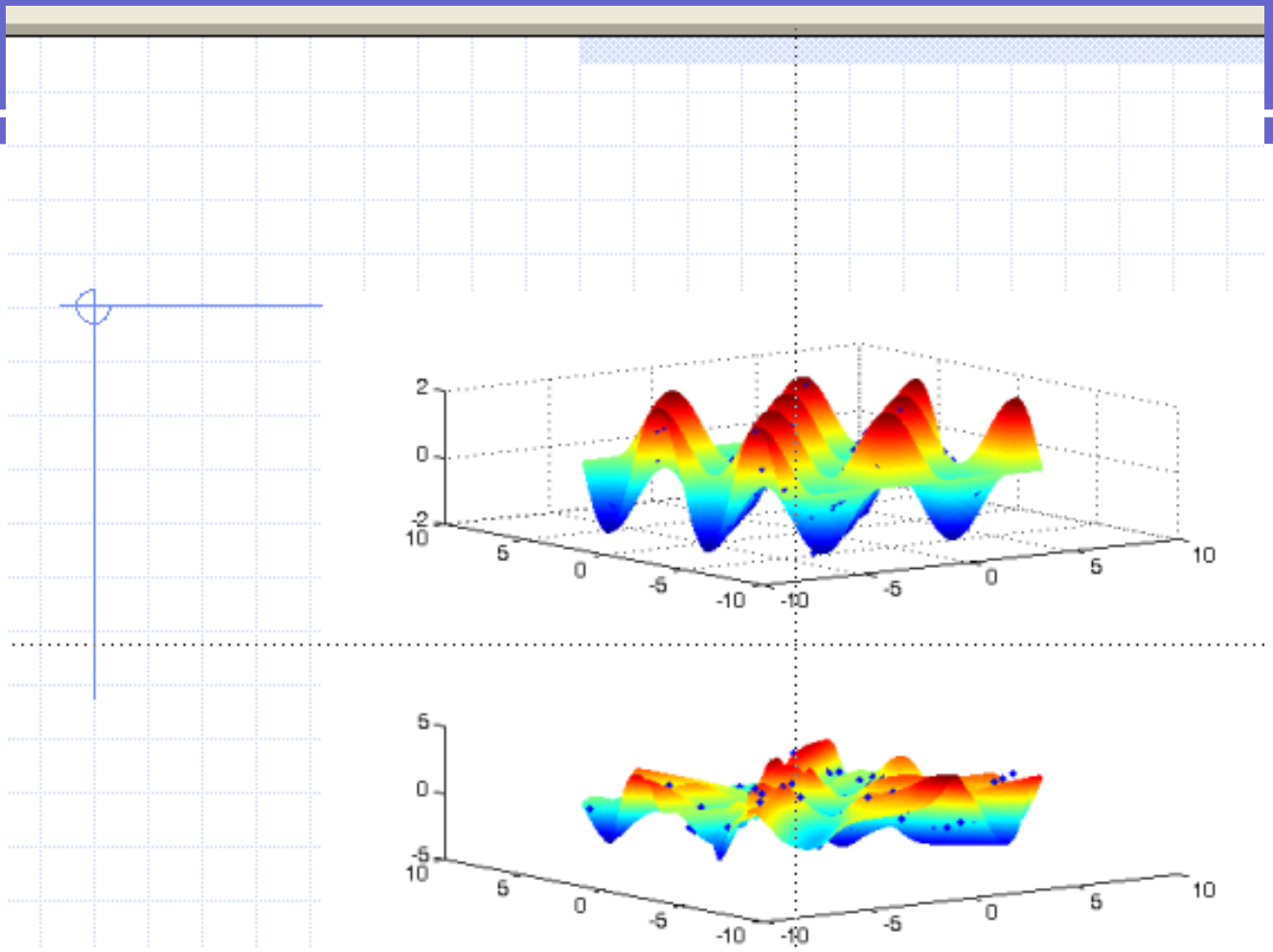
	Mean±std(training)	Mean±std(testing)	CPU time
MLP-Batch(25)	4.70e-1±0	4.73e-1±0	21.2±0.1
MLP-Batch(60)	4.68e-1±0	4.71e-1±0	33.0±0.1
MLP-Recursive(25)	2.51e-1±0	2.59e-1±0	155.3±0.5
MLP-Recursive(60)	3.60e-3±0	3.70e-3±0	186.0±0.5
RBF(25)	7.98e-2±2.02e-2	8.68e-2±2.15e-2	0.6±0.1
RBF(60)	7.90e-3±7.00e-4	8.50e-3±1.10e-3	1.4±0.1

Flow chart


Generate 2N predictors
Substitute them to F
Randomly partition paired data
to two sets respectively
for training and testing
Testing set
Training set
Employ findabr.m to find
Network parameters
Use x2y_MLP.m to
calculate
the mean square error
mse



N=100, M=20, range=2*pi



N=400, M=20, range=2*pi

Function reconstruction

N=600, M=30