MLP learning by solving a nonlinear system

Using the Levenberg-Marquardt method

Target function

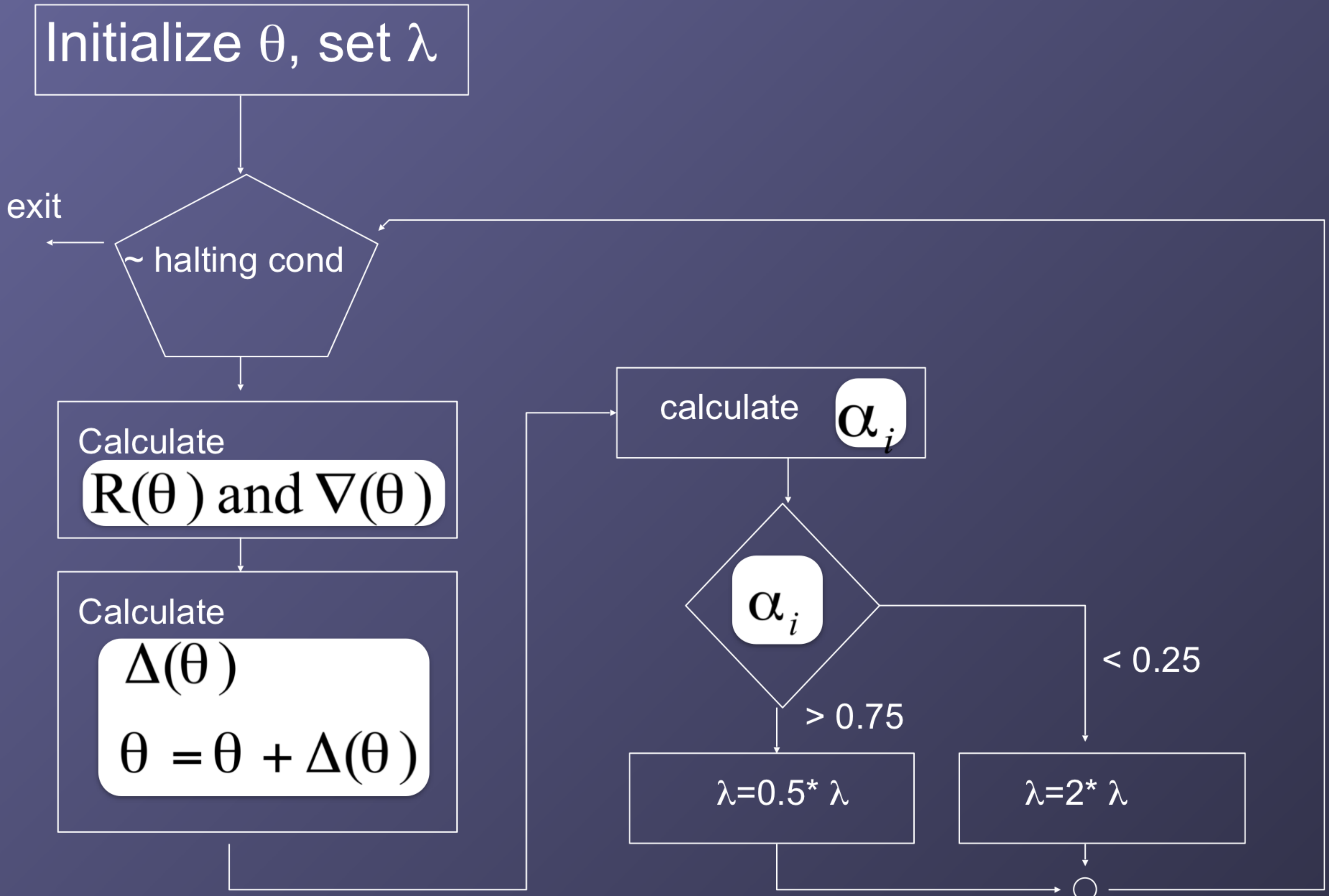
```
function h = g(x1,x2)
C1=[1 1/2 -1/2]'; %weight
C2=[1/3 -1 1]'; %weight
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*C1)+tanh(A*C2); %activation function
end
```

- High dimension and nonlinear target function

The Levenberg-Marquardt Method

Levenberg-Marquardt learning

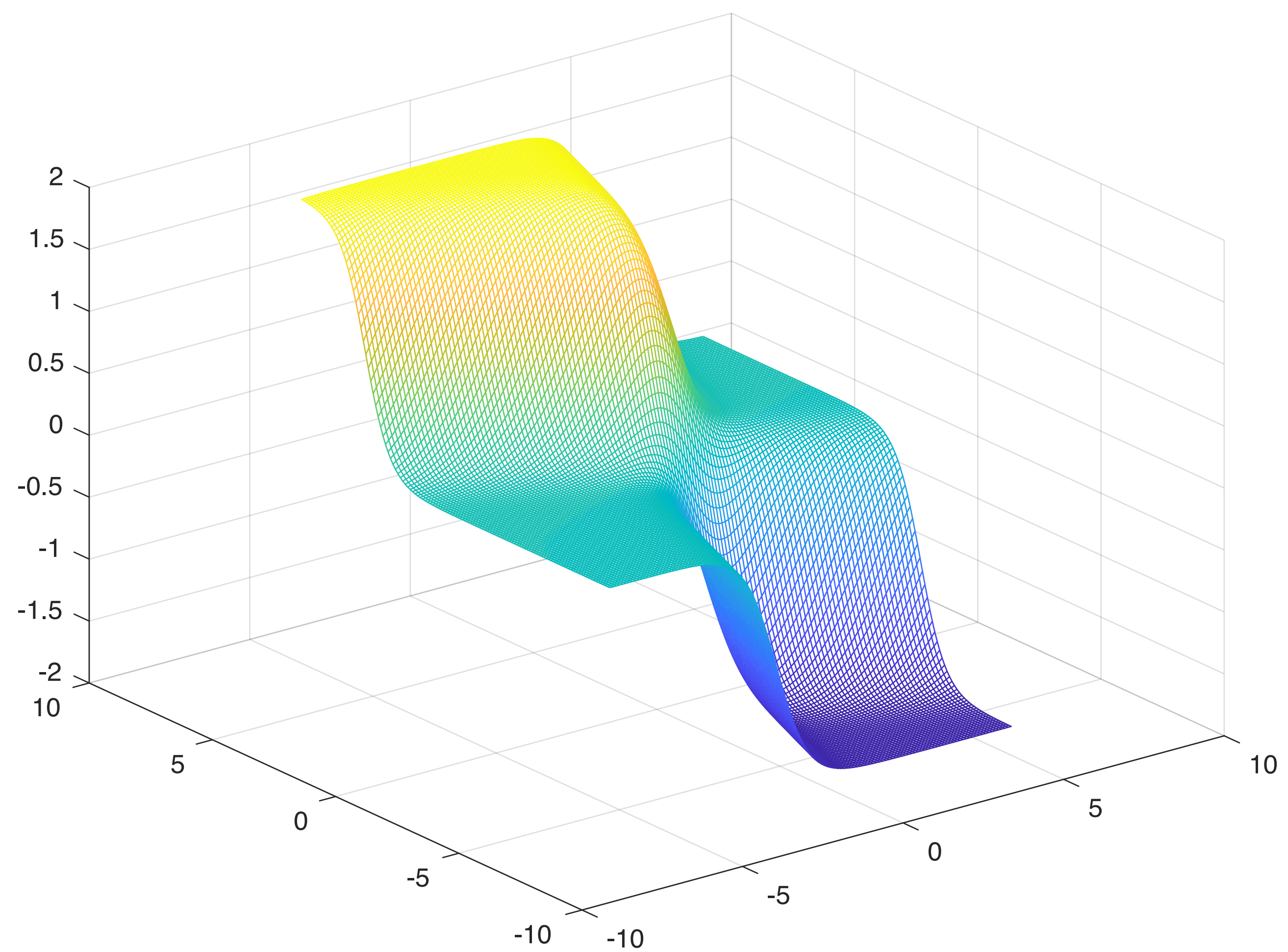
- Multi-layer neural networks
- MLP (Multilayer Perceptrons)
- RBF (Radial Basis Functions)
- Newton-Gauss method
- Levenberg-Marquardt method


Initialize θ, set λ
exit
~ halting cond
Calculate
R(θ) and ∇(θ)
Calculate
Δ(θ)
θ = θ + Δ(θ)
calculate α_i
α_i
< 0.25
> 0.75
λ=0.5* λ
λ=2* λ

demo_plot2d

```
function demo_plot2d()

range=2*pi;
x1=-range:0.1:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=g(x1(i)*ones(length(x2),1),x2');
end
mesh(x1,x2,C);
end
function h = g(x1,x2)
C1=[1 1/2 -1/2]'; %weight
C2=[1/3 -1 1]'; %weight
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*C1)+tanh(A*C2); %activation function
end
```

```
function h = g(x1,x2)
C1=[1 1/2 -1/2]'; %weight
C2=[1/3 -1 1]'; %weight
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*C1)+tanh(A*C2); %activation function
end
```



Approximating function

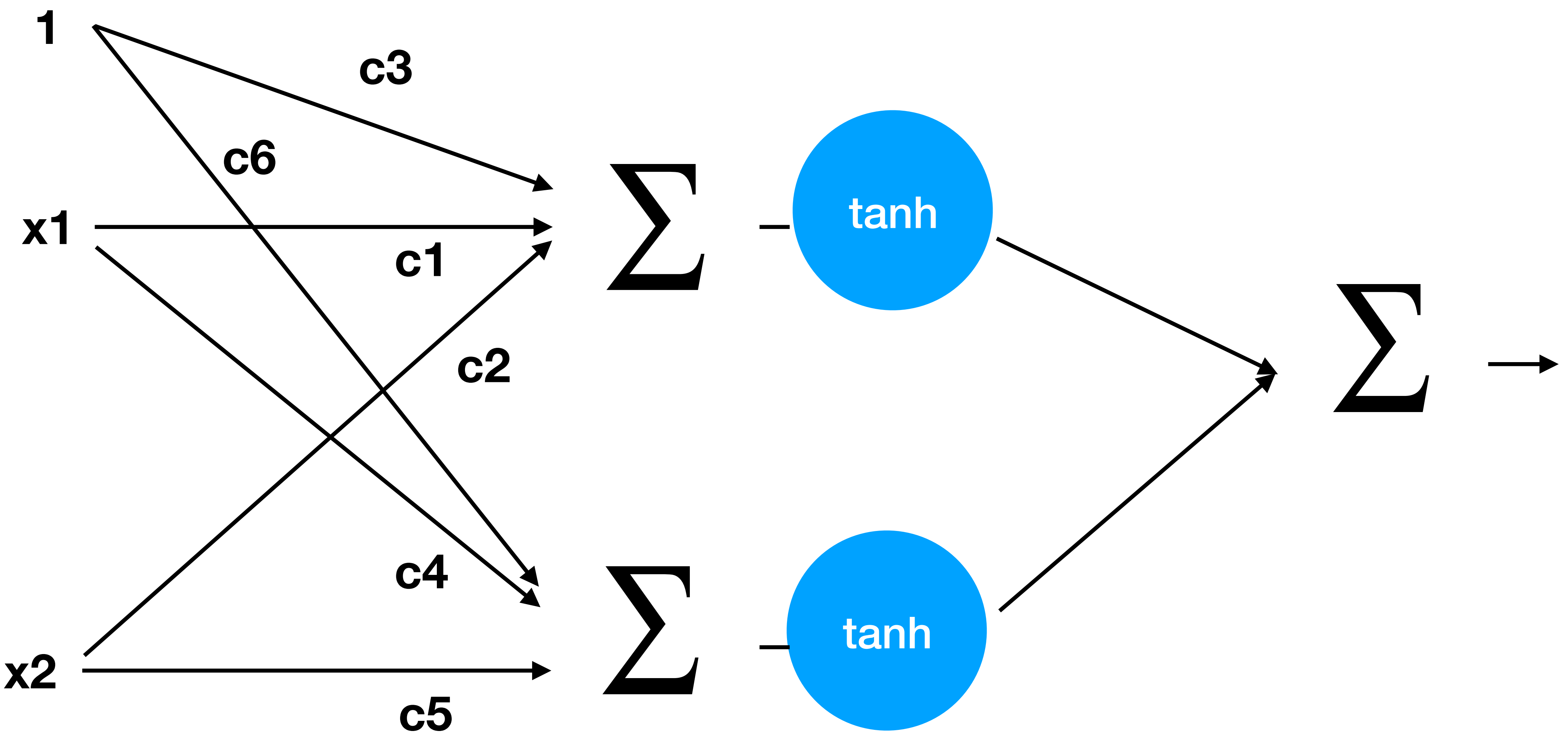
- A multilayer neural network

Adaptable parameters

```
function h = g_hat(x1,x2,c)
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*c(1:3)')+tanh(A*c(4:6)');
end
```

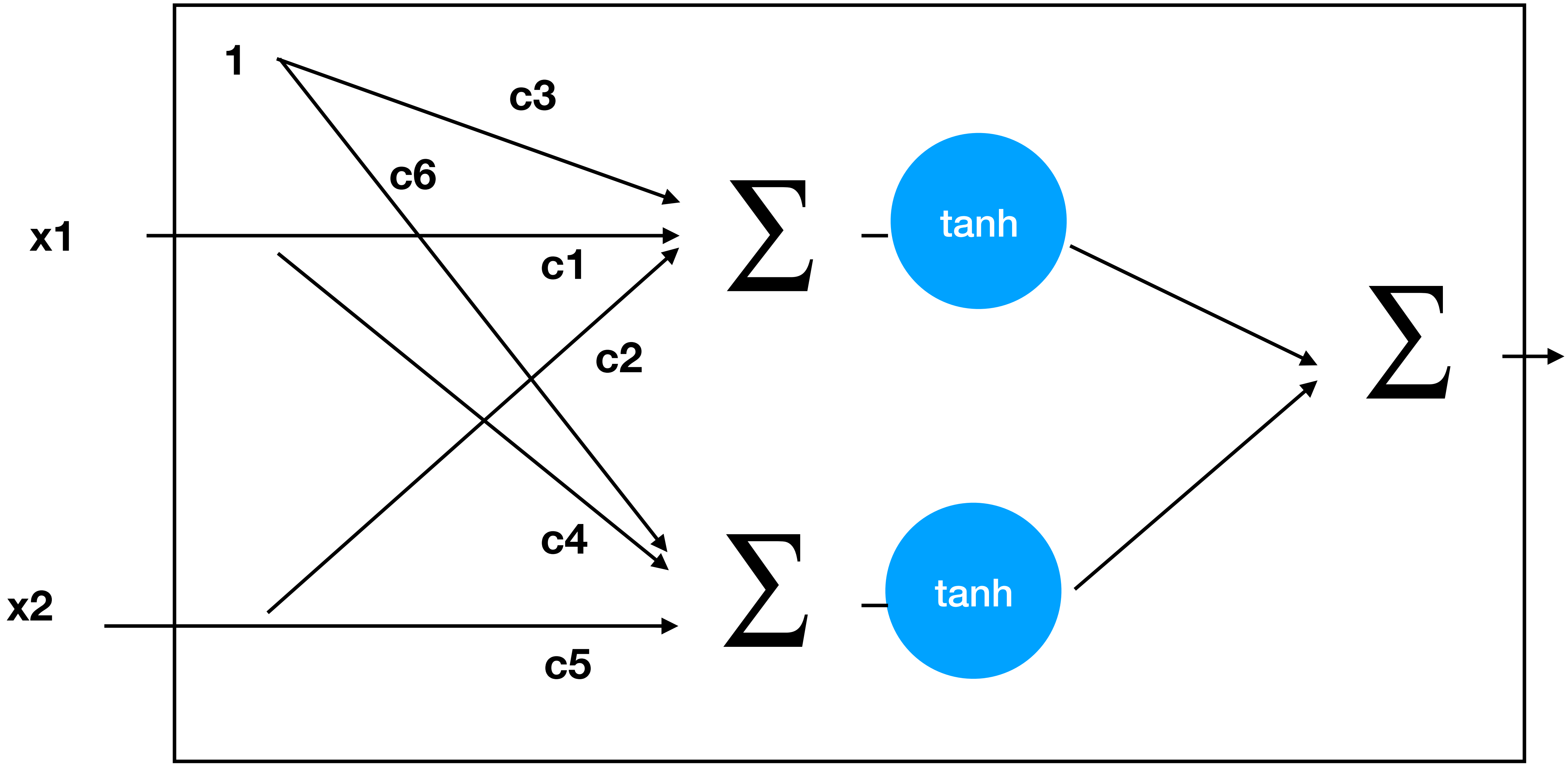
- Approximating function g by g_hat
- The problem is how to estimate adaptable parameters in c.

Multilayer perceptrons



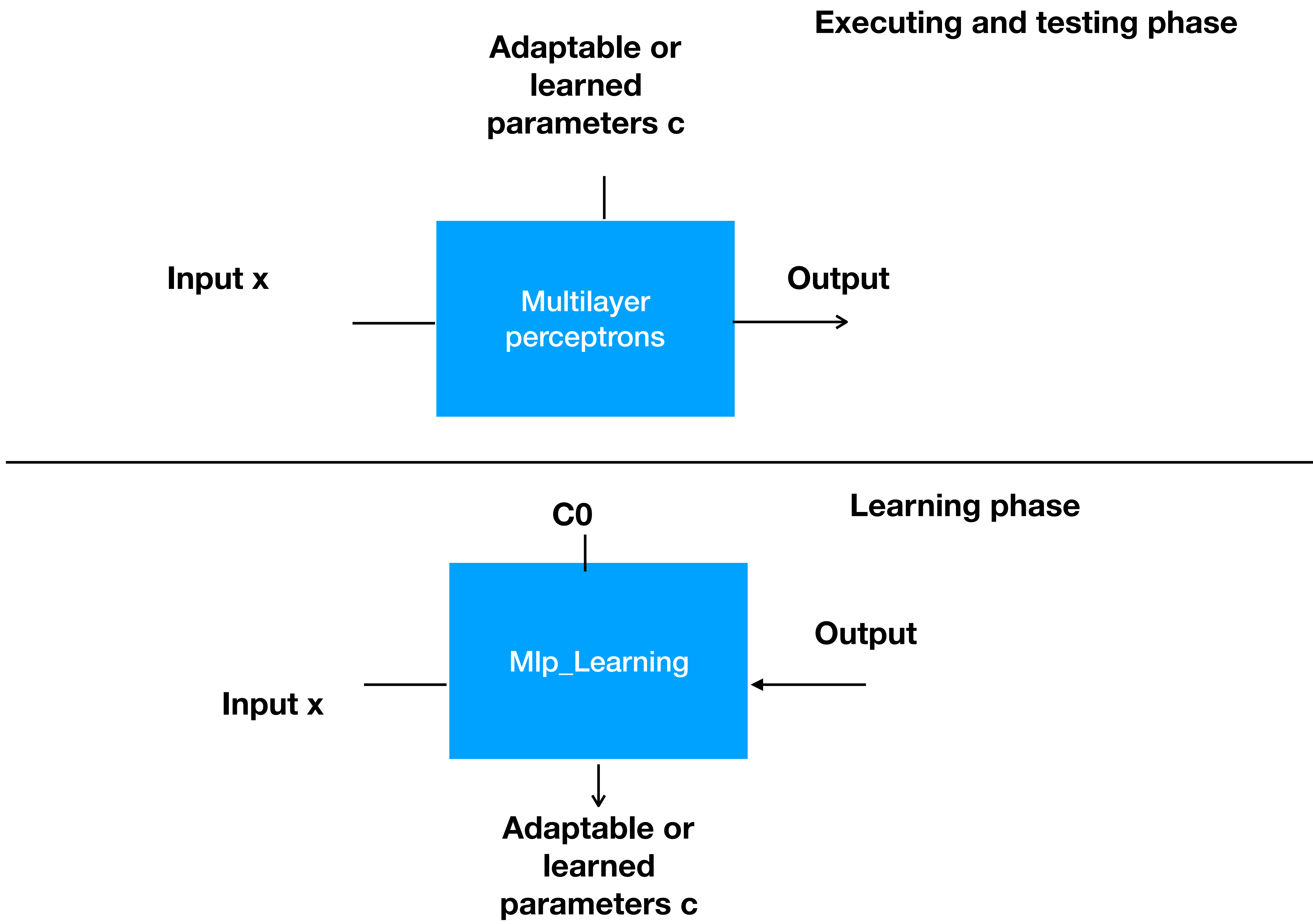
h = g_hat(x1,x2,c)

Multilayer perceptrons



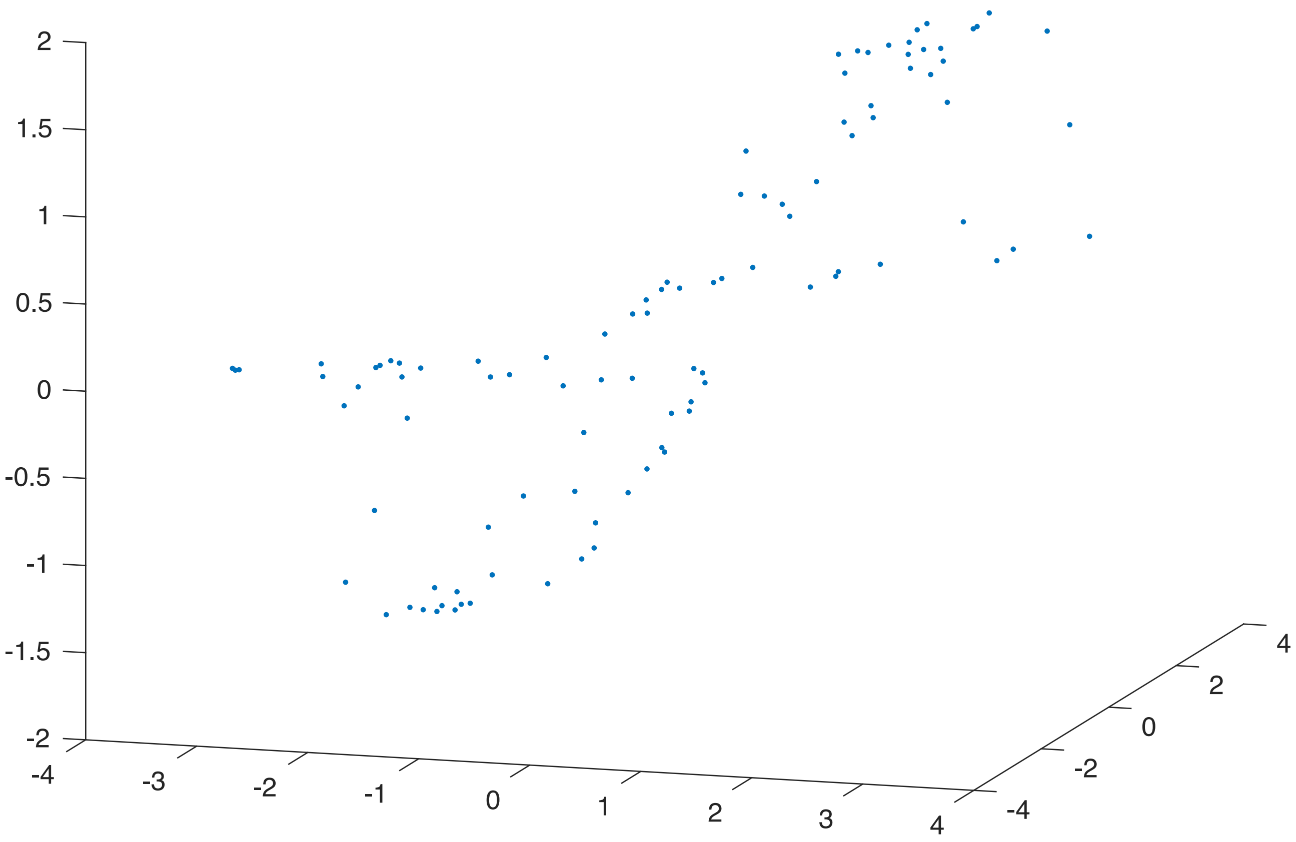
h = g_hat(x1,x2,c)

g_hat is a parametric function
c: adaptable parameters


Executing and testing phase
Adaptable or learned parameters c
Input x
Multilayer perceptrons
Output
Learning phase
C0
Output
Mlp_Learning
Input x
Adaptable or learned parameters c

training data

```
z=rand(50,2)*2*pi-pi;
y=g(z(:,1),z(:,2));
plot3(z(:,1),z(:,2),y,'.');
```



- Two steps
  - Generate a uniform sample from the domain
  - Substitute z(:,1) and z(:,2) to g

testing data

```
z_test=rand(50,2)*2*pi-pi;
y_test=g(z_test(:,1),z_test(:,2));
```

- Two steps
  - Generate another uniform sample from the domain
  - Substitute z_test(:,1) and z_test(:,2) to g to generate y_test
  - Substitute z_test(:,1) and z_test(:,2) to g_hat to generate y_hat for approximating y_test

MLP learning and testing

- MLP learning
  - Train g_hat(x1, x2,c) by data z, y
- Let c_hat denote the learning result
- Test g_hat(x1,x2,c_hat) by z_test and y_test

A nonlinear system

```
function F = learning_mlp(c,x1,x2,y)
A = [x1 x2 ones(length(x1),1)];
F = tanh(A*c(1:3)')+tanh(A*c(4:6)')-y; %activation function
end
```

- Create a nonlinear function
  - F = learning_mlp(c,x1,x2,y)

```
function F = learning_mlp(c,x1,x2,y)
A = [x1 x2 ones(length(x1),1)];
F = tanh(A*c(1:3)')+tanh(A*c(4:6)')-y; %activation function
end
```

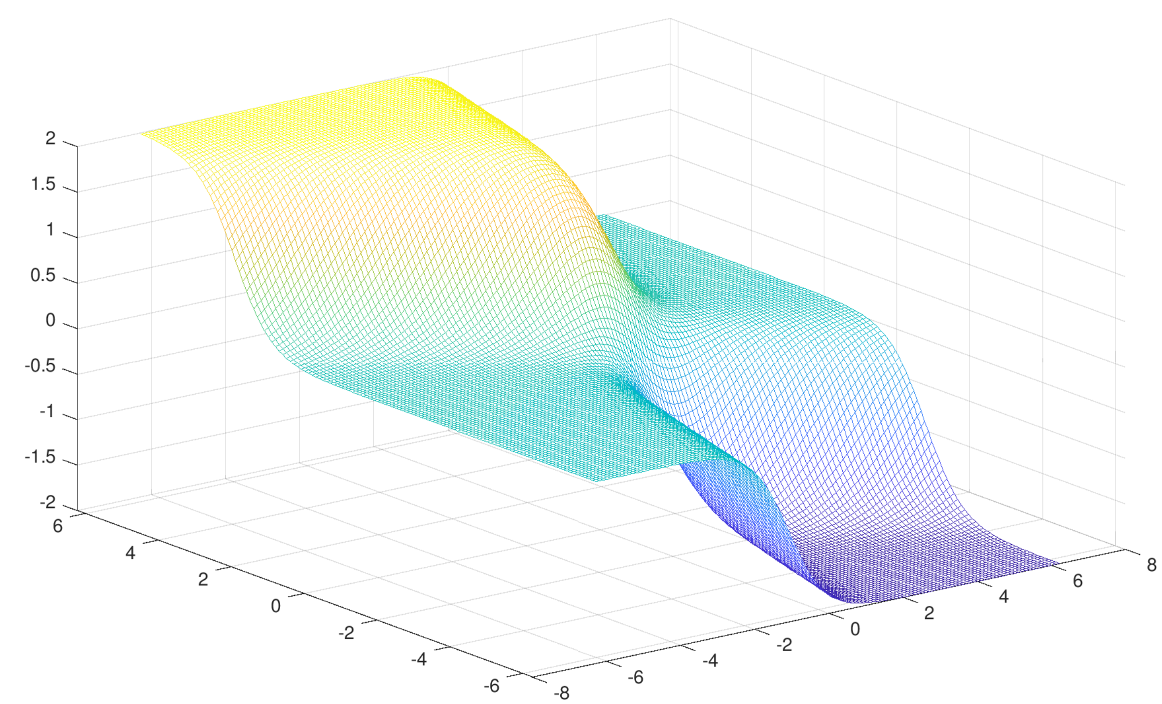
- learning_mlp is a nonlinear system
- adaptable parameters c are unknown
- Set x1 to z(:,1), x2 to z(:,2) and y
  - Let c_zero denote a zero where F is close to zero
  - The output of g_hat(x1, x2,c_zero) well approximates given target y

```
%% Solving a nonlinear system for MLP learning, created by Jiann-Ming Wu
%    Department of Applied Mathematics, National Dong Hwa University
%
%  Using Matlab (R)
%  2018 dec. 3
%  MLP learning is resolved by the Levenberg-Marquardt method
%
```

```
function demo_mlp_learning()
syms c   % adaptable parameters in an MLP network

% initialization
c0=rand(1,6)*2-1;

% preparation of training data
% uniform sampling
% Substitute to the target function g
z=rand(100,2)*2*pi-pi;
y=g(z(:,1),z(:,2));
plot3(z(:,1),z(:,2),y,'.');
```

```
% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_mlp(c,z(:,1),z(:,2),y);
```

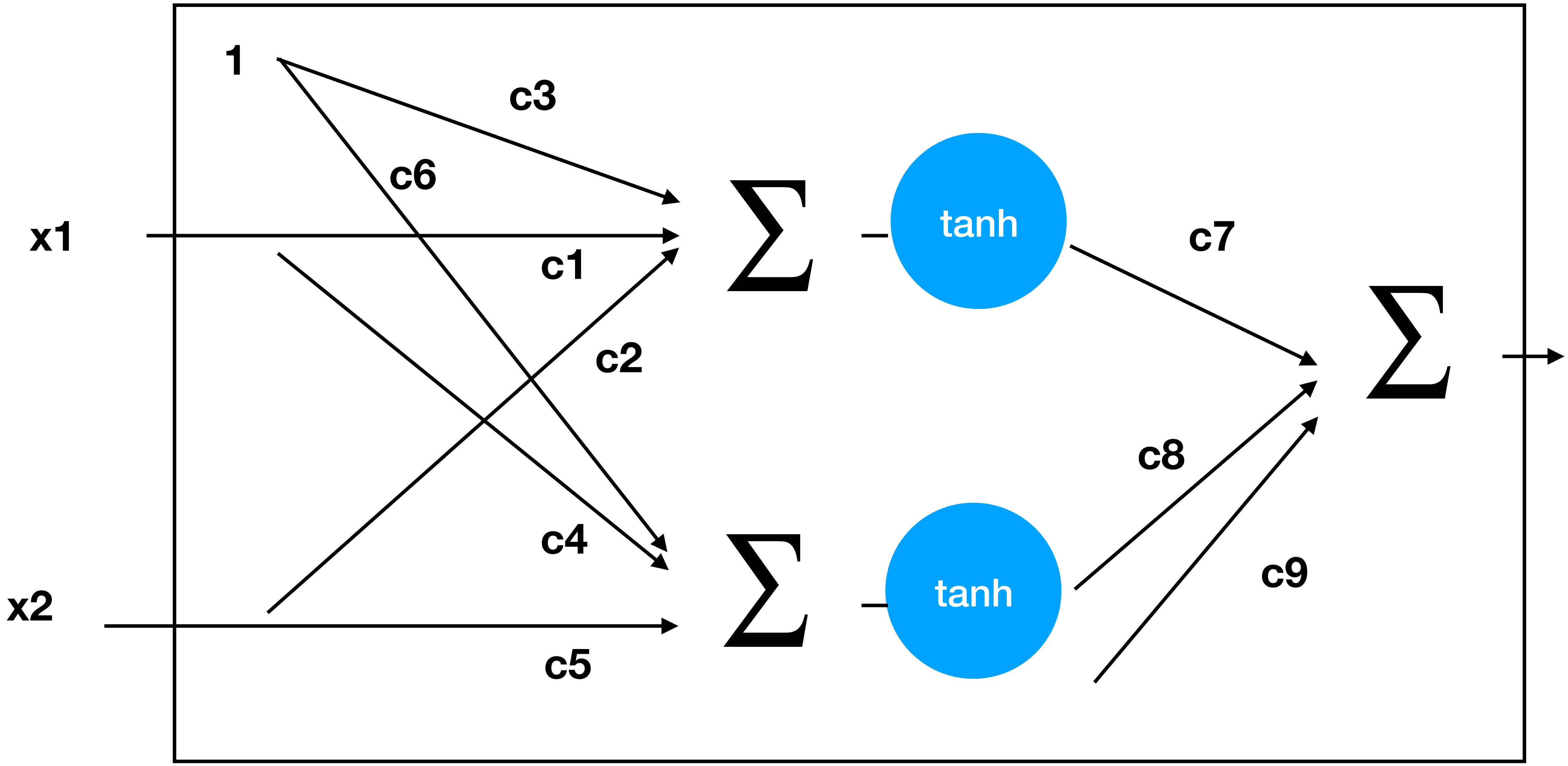
```
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options);
sum(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))

j=2*pi;
x1=-range:0.1:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=g_hat(x1(i)*ones(length(x2),1),x2',c_zero);
end
mesh(x1,x2,C);

end
```



```
% a target function
function h = g(x1,x2)
C1=[1 1/2 -1/2]'; %weight
C2=[1/3 -1 1]'; %weight
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*C1)+tanh(A*C2); %activation function
end
```

```
% a nonlinear system for MLP_learning
function F = learning_mlp(c,x1,x2,y)
A = [x1 x2 ones(length(x1),1)];
F = tanh(A*c(1:3)')+tanh(A*c(4:6)')-y; %activation function
end

% An approximating function
function h = g_hat(x1,x2,c)
A = [x1 x2 ones(length(x1),1)];
h = tanh(A*c(1:3)')+tanh(A*c(4:6)');
end
```

Multilayer perceptrons



h = g_hat(x1,x2,c)

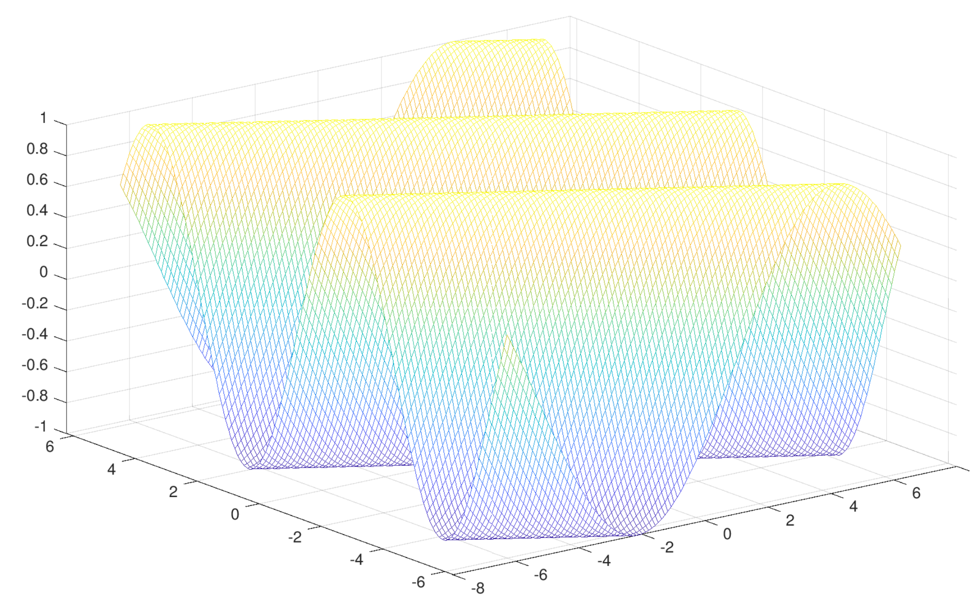
g_hat is a parametric function
c: adaptable parameters

M perceptrons

- How to revise mlp_learning for the case of learning a network of M perceptrons?
- How to add adaptable posterior weights?
- How to approximate a target function, sin(x1+x2) ?

demo_plot2d

```
function demo_plot2d_sin()

range=2*pi;
x1=-range:0.1:range;
x2=x1;
for i=1:length(x1)
        C(i,:)=g_sin(x1(i)*ones(length(x2),1),x2');
end
mesh(x1,x2,C);
end
function h = g_sin(x1,x2)
C1=[1 1/2 -1/2]'; %weight

A = [x1 x2 ones(length(x1),1)];
h = sin(A*C1); %activation function
end
```



Approximating function

- A multilayer neural network

```
function h = g_hat2(x1,x2,c,M)
A = [x1 x2 ones(length(x1),1)];
d=2;
W=reshape(c(1:M*(d+1)),d+1,M);
v = tanh(A*W);
h = v*c(M*(d+1)+1:end);
end
```

Adaptable parameters

- Approximating function g_sin by g_hat
- The problem is how to estimate adaptable parameters in c.

A nonlinear system

```
function F = learning_mlp(c,x1,x2,y)
A = [x1 x2 ones(length(x1),1)];
F = tanh(A*c(1:3)')+tanh(A*c(4:6)')-y; %activation function
end
```

- Create a nonlinear function
  - F = learning_mlp(c,x1,x2,y)