MLP(multilayer perceptrons) learning by the Levenberg-Marquardt method

Mathematical modeling and neural architecture

$$y = tanh(c_1x_1 + c_2x_2 + c_3) + tanh(c_4x_1 + c_5x_2 + c_6) + 0$$

Linear Combination of input attributes

Mathematical modeling for nonlinear transformation

$$y = tanh(c_1x_1 + c_2x_2 + c_3) + tanh(c_4x_1 + c_5x_2 + c_6) + 0$$

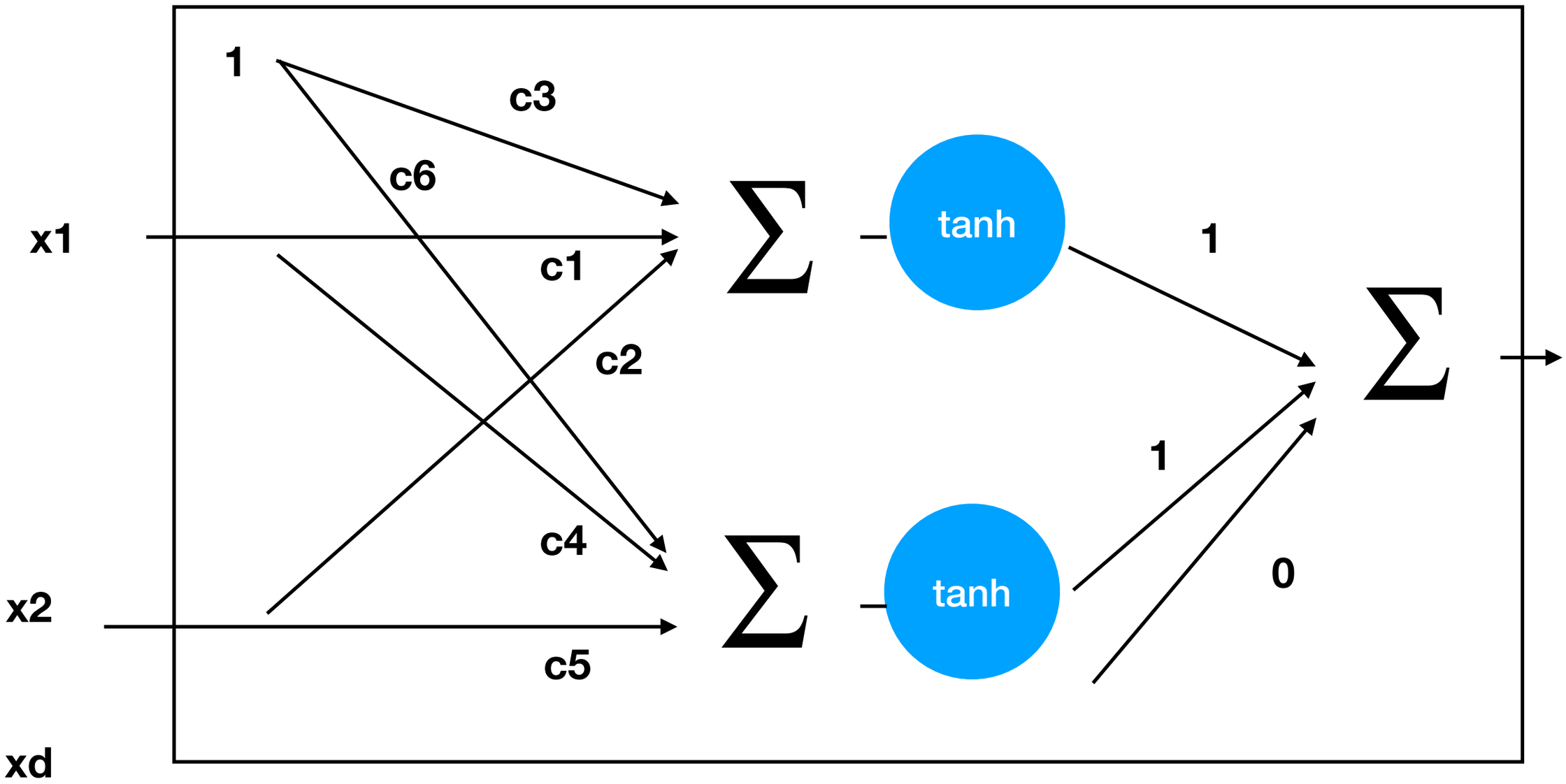
Nonlinear transformation of linear combination of input attributes

Posterior weights

$$y = 1 \times tanh(c_1 x_1 + c_2 x_2 + c_3) + 1 \times tanh(c_4 x_1 + c_5 x_2 + c_6) + 0$$

Linear combination of results of nonlinear transformation

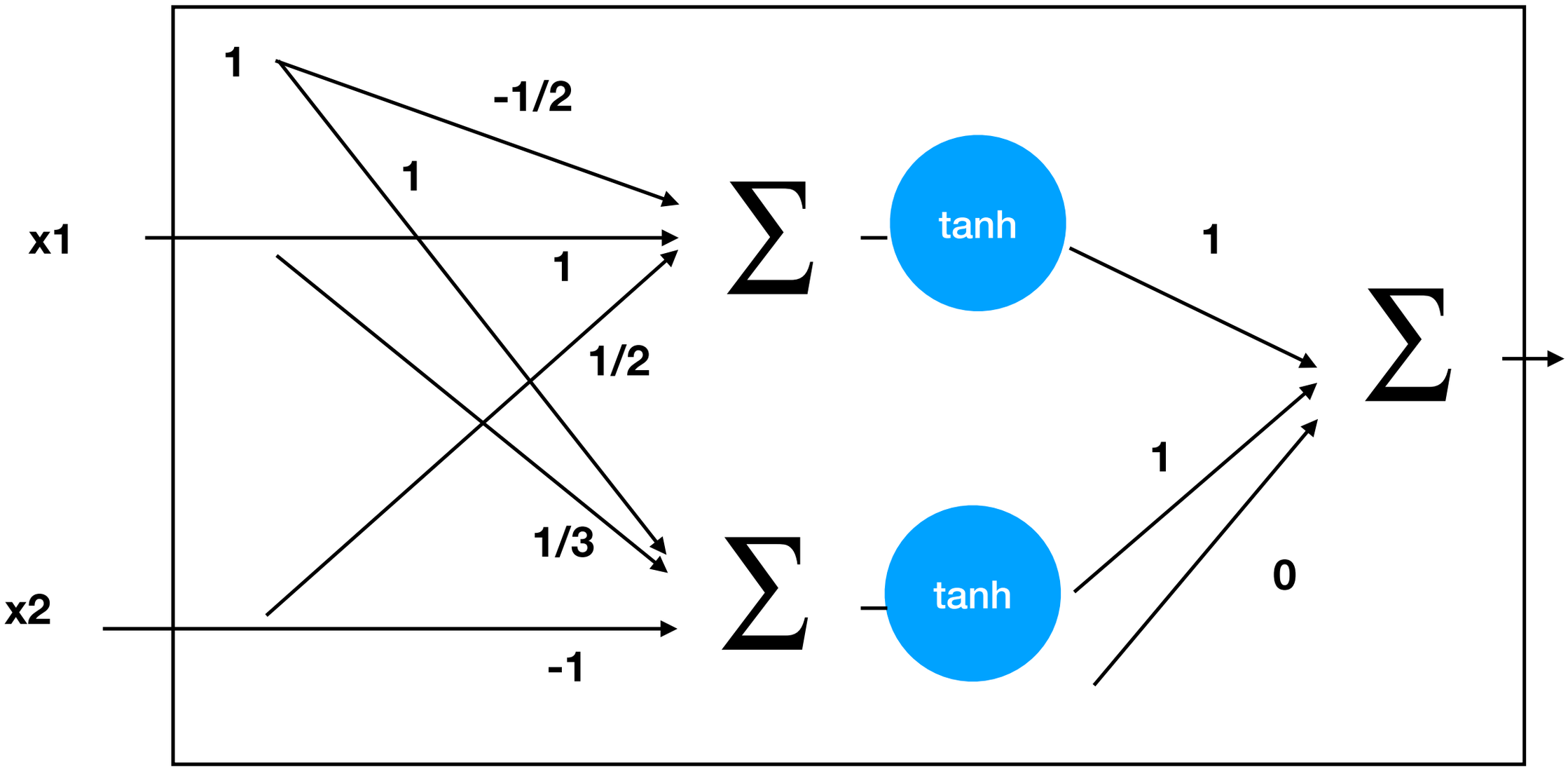
Neural Architecture: Perceptrons 認知機



h = g_hat(x1,x2,c)

g_hat is a parametric function
c: adaptable parameters

Perceptrons with fixed weights



h = g_hat(x1,x2,c)

g_hat is a parametric function
c: adaptable parameters

Target function

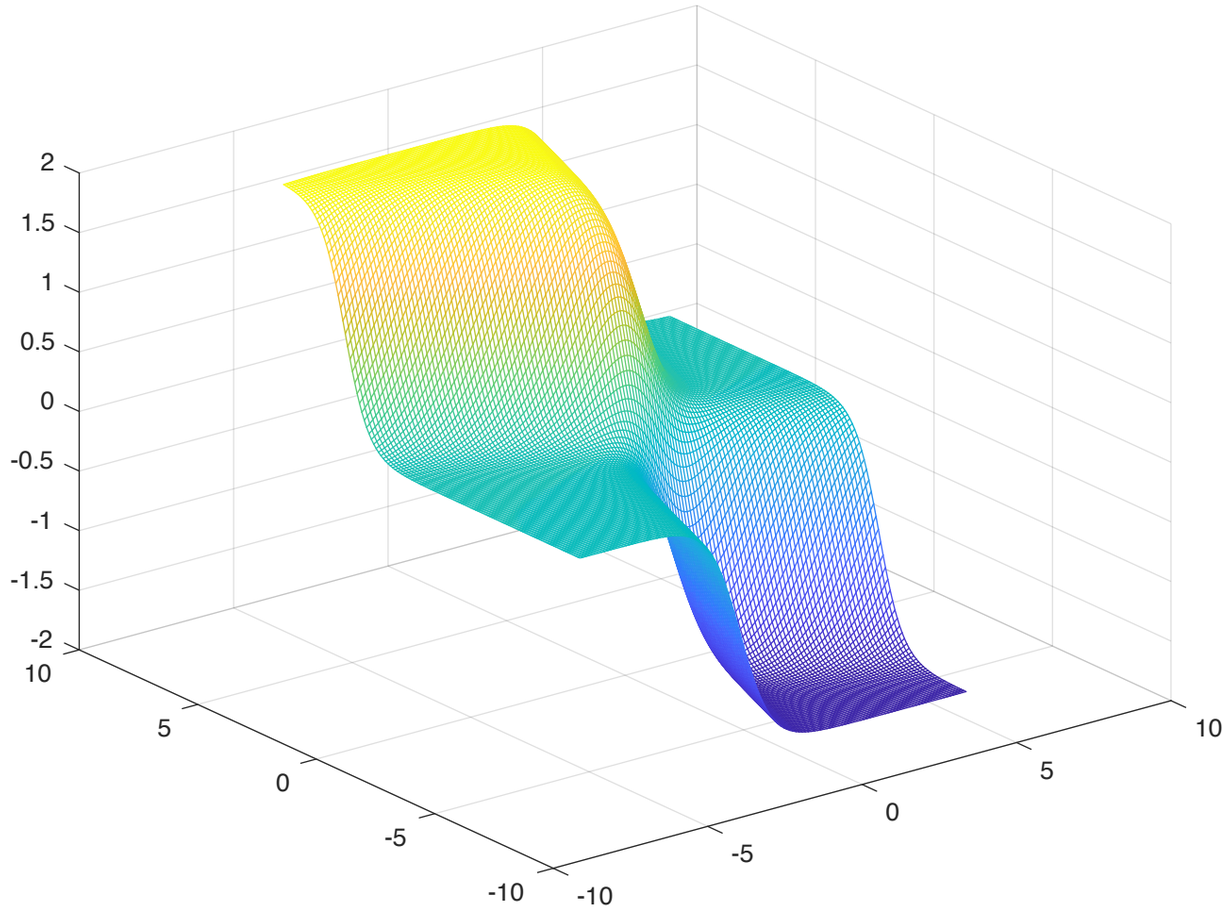
```
function h = g(x1,x2)
      C1=[1 1/2 -1/2]'; %weight
      C2=[1/3 -1 1]'; %weight
      A = [x1 x2 ones(length(x1),1)];
      h = tanh(A*C1)+tanh(A*C2); %activation function
end
```

- High dimension and nonlinear target function

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 performs a parametric function

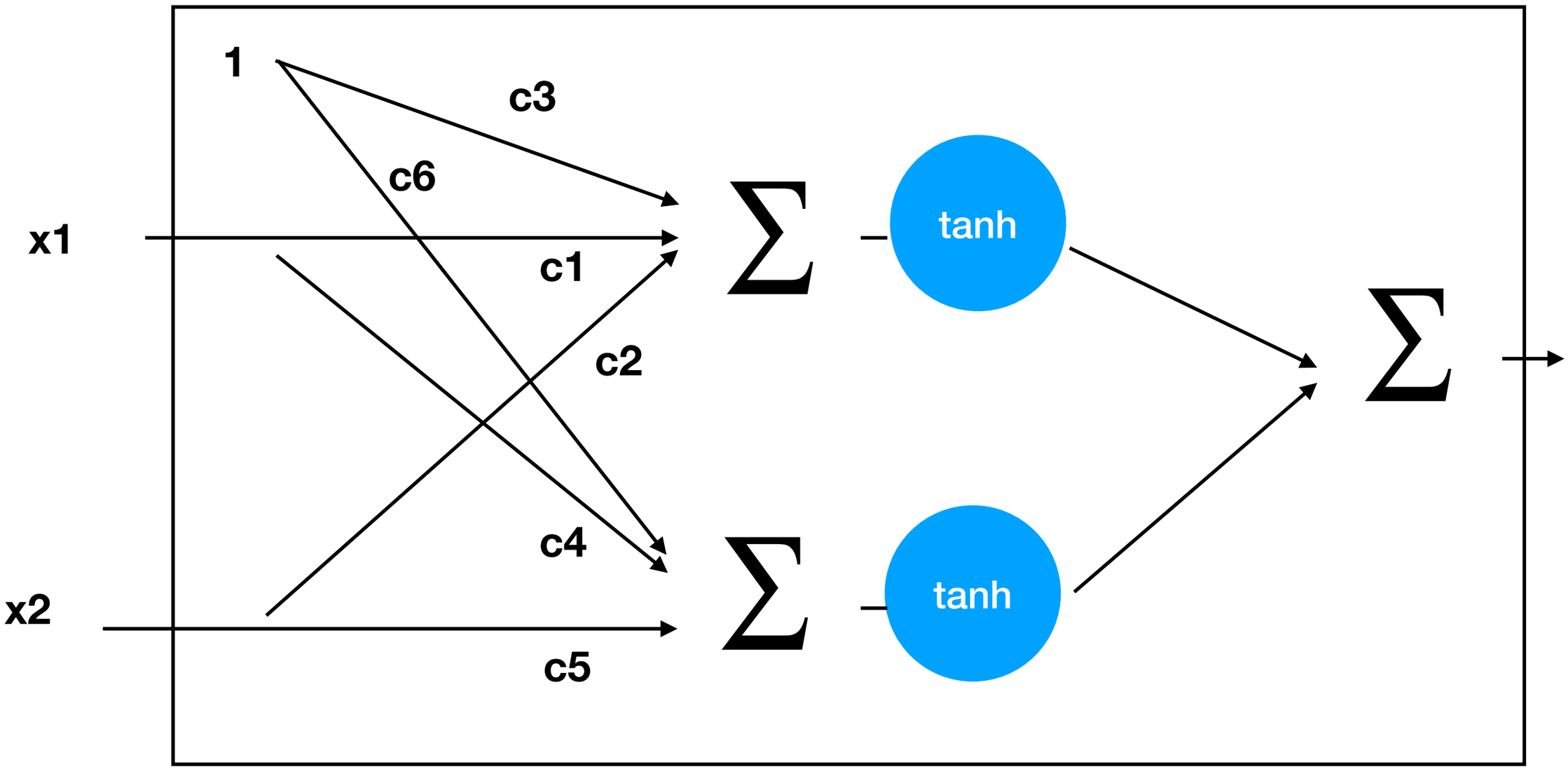
```
function h = g_hat(x1,x2,c)
    A = [x1 x2 ones(length(x1),1)];
    h = tanh(A*c(1:3)')+tanh(A*c(4:6)');
end
```

Adaptable parameters

11.12

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

Multilayer perceptrons



h = g_hat(x1,x2,c)

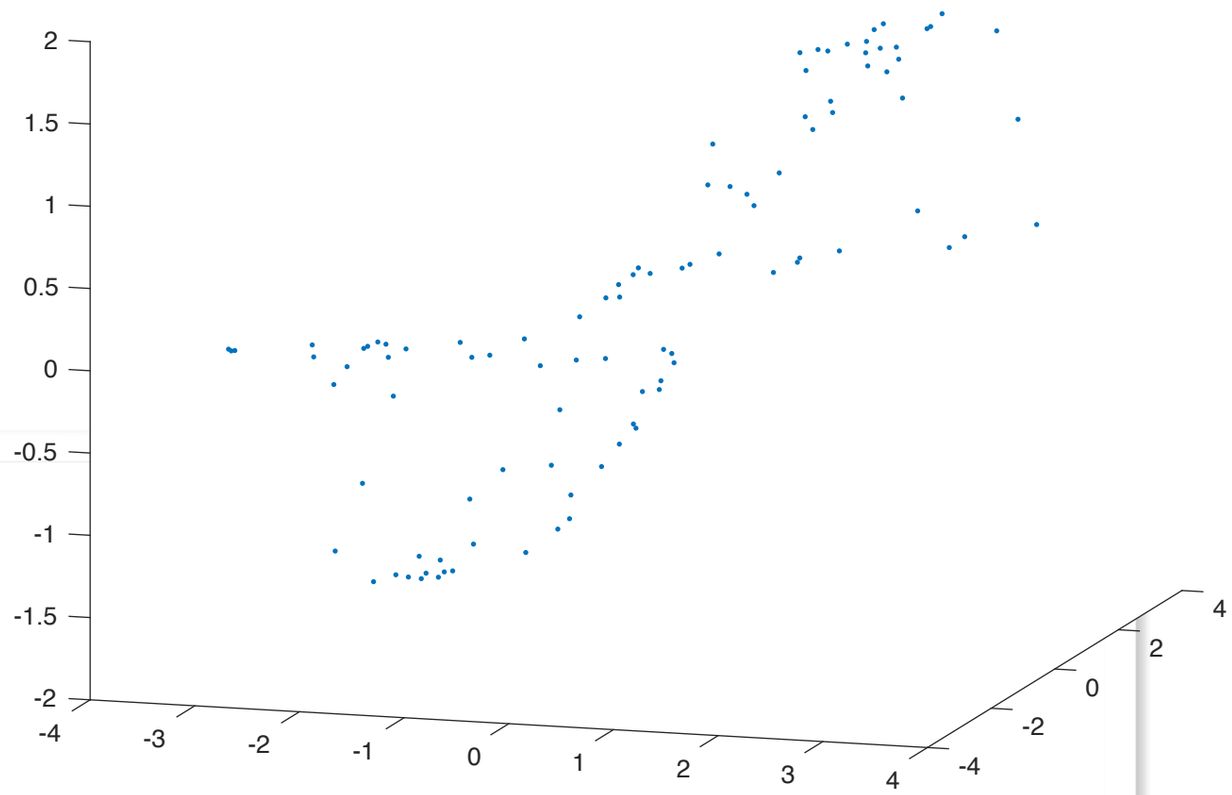
g_hat is a parametric function
c: adaptable parameters

Data Driven Learning and Testing

Training and Testing Data Sets

training data set

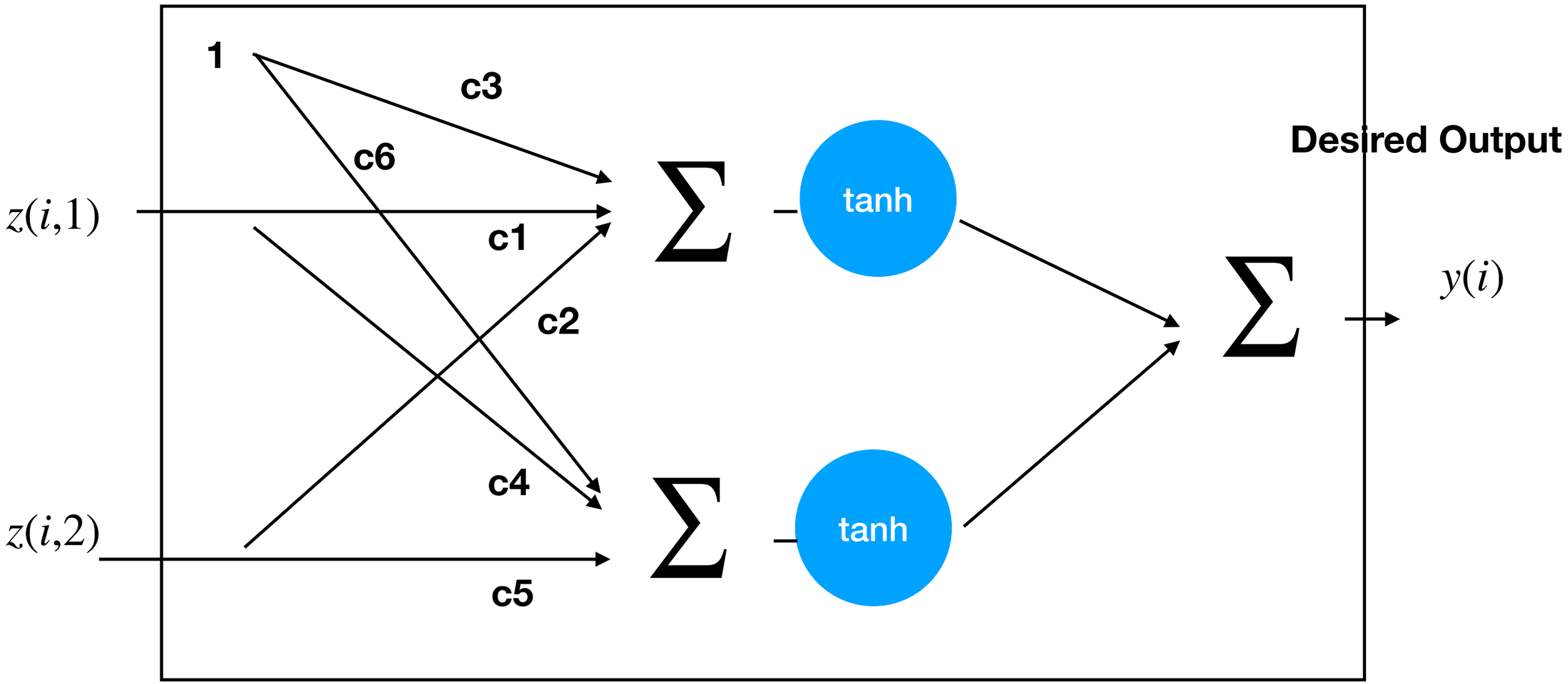
```
z=rand(50,2)*2*pi-pi;
y=g(z(:,1),z(:,2));
plot3(z(:,1),z(:,2),y,'.');
```



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

$z_i = (z(i,1), z(i,2))^T$ denotes an input

y_i denotes the correspondent output



$$g(x;c) = tanh(c_1x_1 + c_2x_2 + c_3) + tanh(c_4x_1c_5x_2 + c_6)$$

Radial basis functions

$$g(x;d) = d_4exp(-\frac{(d_1 - x_1)^2 + (d_2 - x_2)^2}{d_3^2}) + d_8exp(-\frac{(d_5 - x_1)^2 + (d_6 - x_2)^2}{d_7^2})$$

g.m | demo_plot2d.m | g_hat.m | generateTraningData.m | learning_mlp.m | demo_

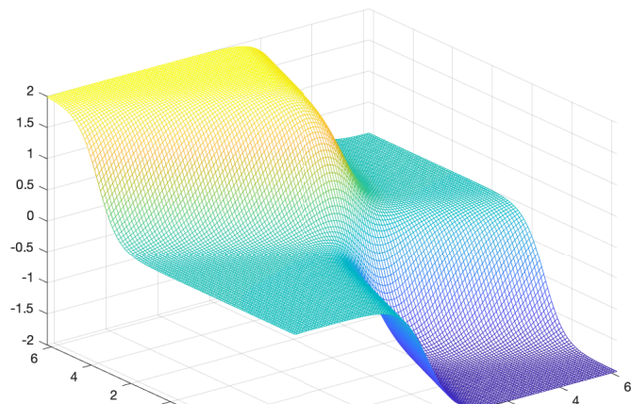
```
1  function h = g(x1,x2)
2      C1=[1 1/2 -1/2]'; %weight
3      C2=[1/3 -1 1]'; %weight
4      A = [x1 x2 ones(length(x1),1)];
5      h = tanh(A*C1)+tanh(A*C2); %activation functior
6  end
```

1

g.m | demo_plot2d.m | g_hat.m | generateTraningData.m | learning_mlp.m | demo_

```
1  function demo_plot2d()
2
3      range=2*pi;
4      x1=-range:0.1:range;
5      x2=x1;
6      for i=1:length(x1)
7          C(i,:)=g(x1(i)*ones(length(x2),1),x2');
8      end
9      mesh(x1,x2,C);
10 end
```

2



Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/codes/demo_MLP_learning2023

g.m | demo_plot2d.m | g_hat.m | generateTraningData.m | learning_mlp.m | den

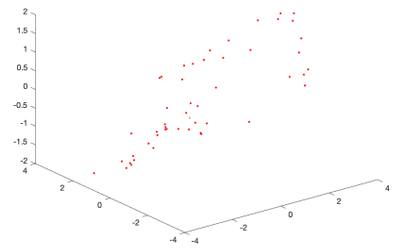
```
1  z=rand(50,2)*2*pi-pi;
2  y=g(z(:,1),z(:,2));
3  plot3(z(:,1),z(:,2),y,'.');
4
5  z_test=rand(50,2)*2*pi-pi;
6  y_test=g(z_test(:,1),z_test(:,2));
7  figure
8  plot3(z_test(:,1),z_test(:,2),y_test,'r.');
```

3

Workspace

Name

- y
- y_test
- z
- z_test



Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/codes/demo_MLP_learning20

+1 | g.m | demo_plot2d.m | g_hat.m | generateTraningData.m | learning_mlp.m | d

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

4A

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2023-I NA數值分析/codes/demo_MLP_learning2023/learning_ml...

+1 | g.m | demo_plot2d.m | g_hat.m | generateTraningData.m | learning_mlp.m | demo_mlp_learning.m

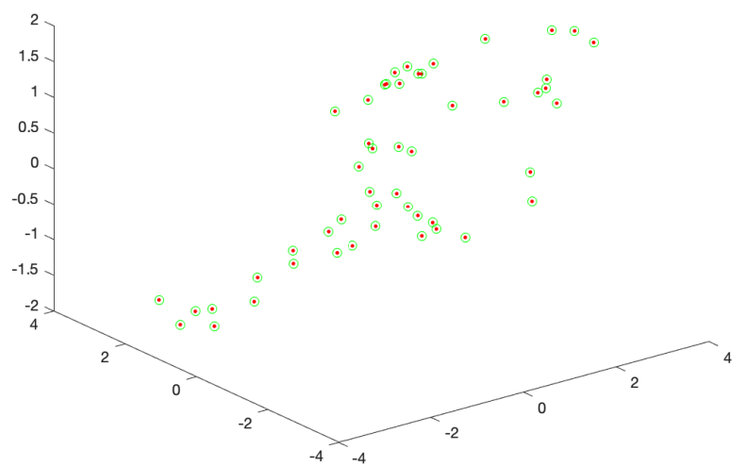
```
1   function F = learning_mlp(c,x1,x2,y)
2         A = [x1 x2 ones(length(x1),1)];
3         F = tanh(A*c(1:3)')+tanh(A*c(4:6)')-y; %activation function
4   end
```

4B

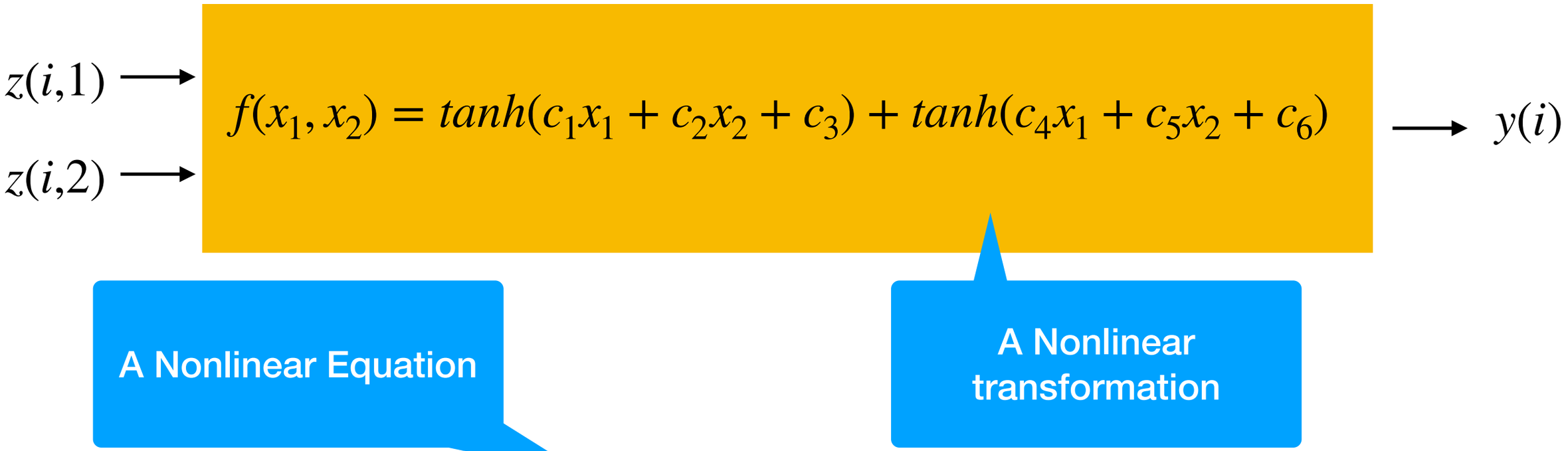
```
function demo_mlp_learning()
syms c   % adaptable parameters in an MLP network

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

generateTraningData
% 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);
maeTraining = mean(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))
maeTesting = mean(abs(learning_mlp(c_zero,z_test(:,1),z_test(:,2),y_test)))
fprintf('maeTraining %12.10f\n',maeTraining)
fprintf('maeTesting %12.10f\n',maeTesting)
figure(1);hold on
plot3(z(:,1),z(:,2),g_hat(z(:,1),z(:,2),c_zero),'go')
figure(2);hold on
plot3(z_test(:,1),z_test(:,2),g_hat(z_test(:,1),z_test(:,2),c_zero),'go')
end
```

5

```
Command Window

maeTraining =

  1.7079e-11


maeTesting =

  1.4374e-11

maeTraining 0.0000000000
maeTesting 0.0000000000
```



Approximating a target function by solving a nonlinear system



$$f(z_1[i], z_2[i]) = tanh(c_1z_1[i] + c_2z_2[i] + c_3) + tanh(c_4z_1[i] + c_5z_2[i] + c_6) = y(i)$$

N = 50
Nonlinear Equations

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \cdots & \cdots & & \cdots \\ \frac{\partial f_n(\mathbf{x})}{\partial x_1} & \frac{\partial f_n(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_n(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

$$f(z_1[1], z_2[1]) = tanh(c_1 z_1[1] + c_2 z_2[1] + c_3) + tanh(c_4 z_1[1] + c_5 z_2[1] + c_6) = y(1)$$

$$f(z_1[2], z_2[2]) = tanh(c_1 z_1[2] + c_2 z_2[2] + c_3) + tanh(c_4 z_1[2] + c_5 z_2[2] + c_6) = y(2)$$

.

.

$$f(z_1[i], z_2[i]) = tanh(c_1 z_1[i] + c_2 z_2[i] + c_3) + tanh(c_4 z_1[i] + c_5 z_2[i] + c_6) = y(i)$$

.

.

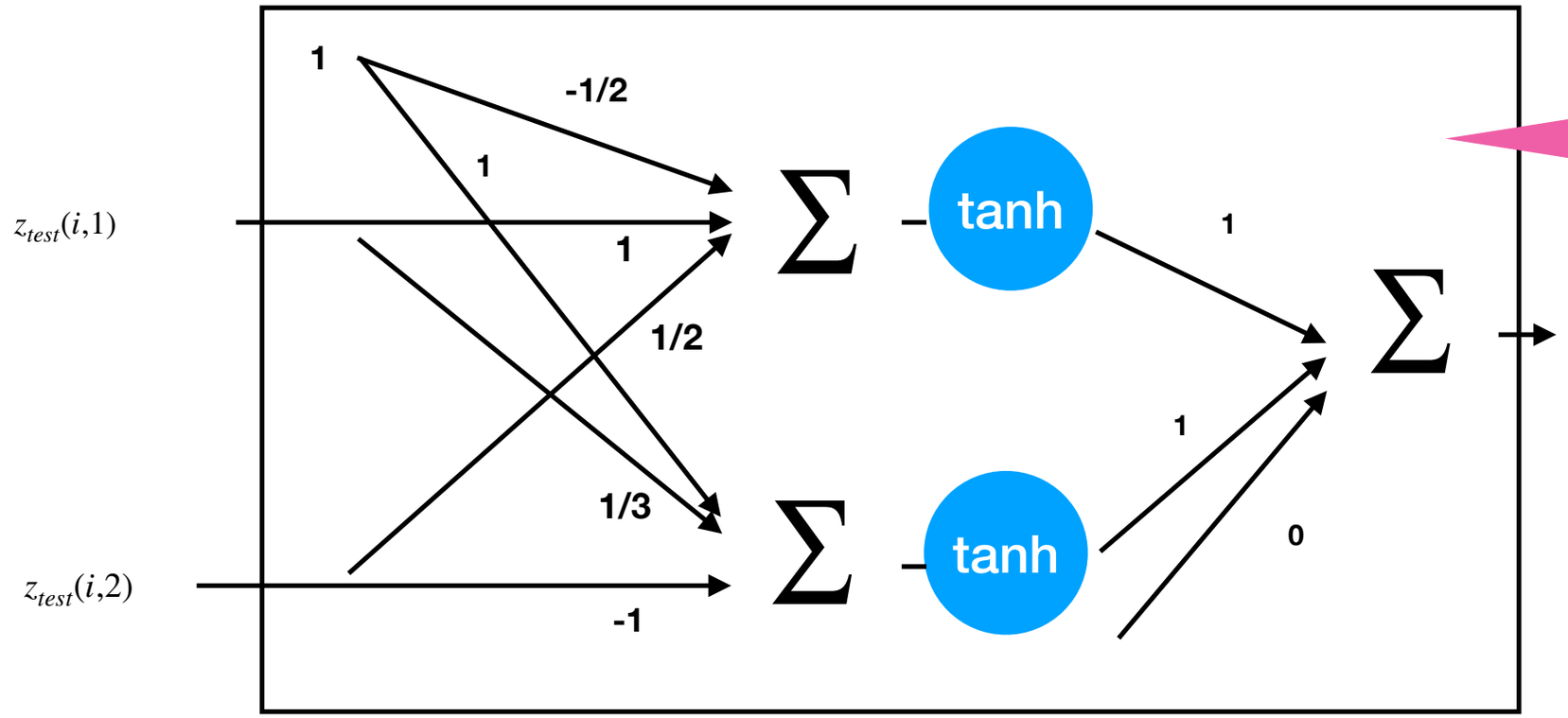
.

$$f(z_1[N], z_2[N]) = tanh(c_1 z_1[N] + c_2 z_2[N] + c_3) + tanh(c_4 z_1[N] + c_5 z_2[N] + c_6) = y(N)$$

testing data

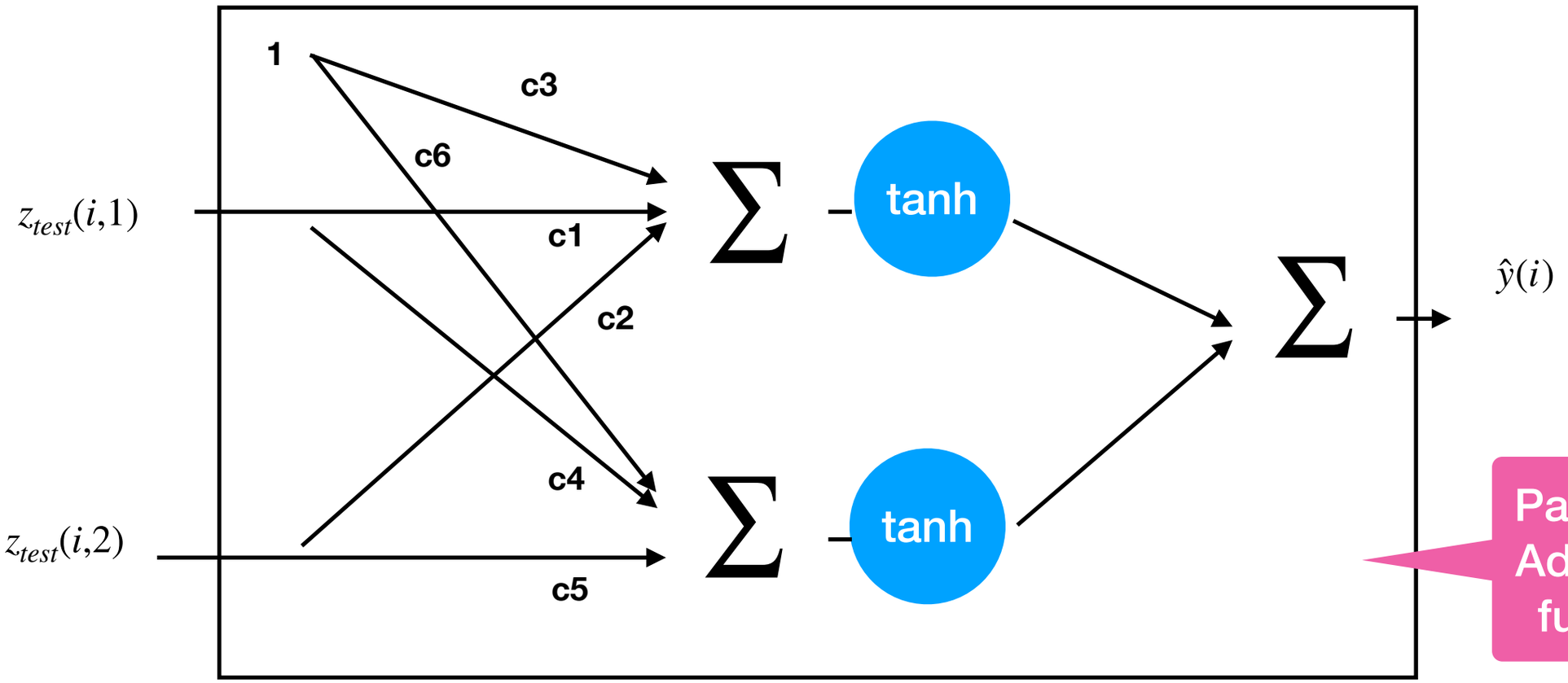
```
z_test=rand(50,2)*2*pi-pi;
y_test=g(z_test(:,1),z_test(:,2));
```

- 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



Target function

Error
$y(i) - \hat{y}(i)$



Parameter Adaptable function

Data driven adaptable learning

goal of learning

Minimizing not only the training mean square error but also the testing mean square error

MLP(multilayer perceptrons) learning and testing

1. MLP learning

Approximating function

   - Train g_hat(x1, x2,c) to optimize c subject to training data z, y

2. Let c_hat denote the learning result

3. Test g_hat(x1,x2,c_hat) by z_test and y_test

A nonlinear system

g_hat(x1,x2,c_hat)

```
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
```

Substract y
F denotes the error

- 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
```

- function 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 target y

Matlab Programming

```
%% 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,'.');
```

Random initialization of six adaptable parameters

A sample from the input space $(-\pi, \pi) \times (-\pi, \pi)$

y denotes the desired target

Option for solving a nonlinear system

```
% 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);
```

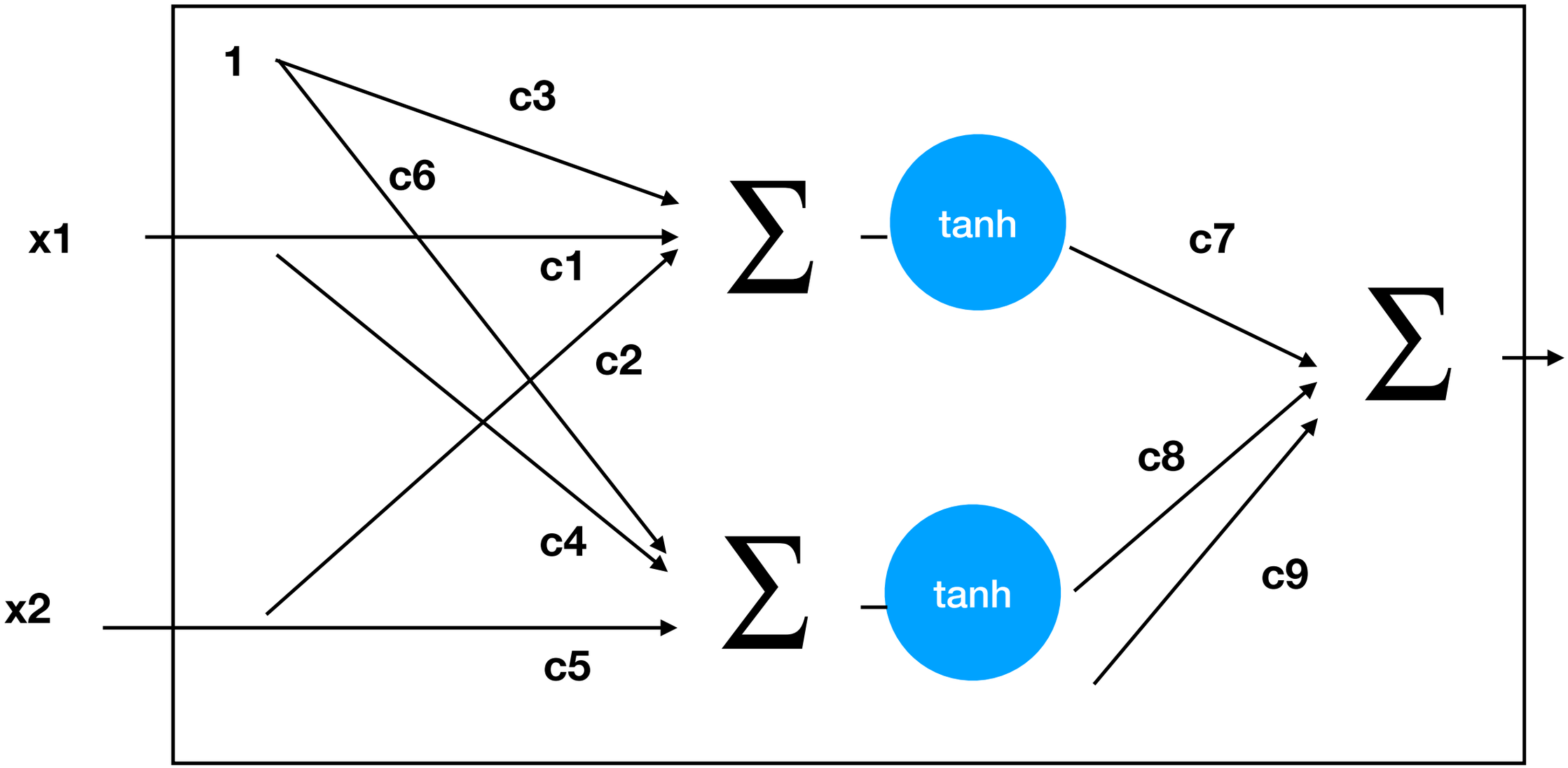
```
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options);
sum(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))
end
```

```
% 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

apple ▸ Desktop ▸ Jiann-Ming Wu ▸ 2022-I NA數值分析 ▸ codes

Editor - /Users/apple/Desktop/Jiann-Ming Wu/2022-I NA數值分析/codes/demo_mlp_learning.m

+1 | demo2.m | demo_sym.m | demo_fsolve_CNN2.m | demo_newton.m | demo_mlp_learning.m

```
17 –        z=rand(100,2)*2*pi-pi;
18 –        y=g(z(:,1),z(:,2));
19 –        plot3(z(:,1),z(:,2),y,'.');
20          % Levenberg-Marquardt method
21 –        options = optimoptions('fsolve','Algorithm', 'levenberg-marquar
22          % specification of a nonlinear system for MLP_learning
23 –        f = @(c)learning_mlp(c,z(:,1),z(:,2),y);
24          % apply fsolve
25          % Verification of c_zero
26 –        c_zero = fsolve(f,c0,options);
27 –        sum(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))
28 –    end
29      % a target function
30      function h = g(x1,x2)
31 –        C1=[1 1/2 -1/2]'; %weight
```

Command Window

```
<stopping criteria details>

ans =

   1.7042e-09
```

Absolute approximating error
sum(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))

```
c_zero = fsolve(f,c0,options);
sum(abs(learning_mlp(c_zero,z(:,1),z(:,2),y)))
h = g_hat(z(:,1),z(:,2),c_zero);
mean((h-y).^2)
plot3(z(:,1),z(:,2),h,'ro')
```

Sum up absolute outputs of the nonlinear system

h approximates desired target
Calculate the mean square approximating error

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) ?