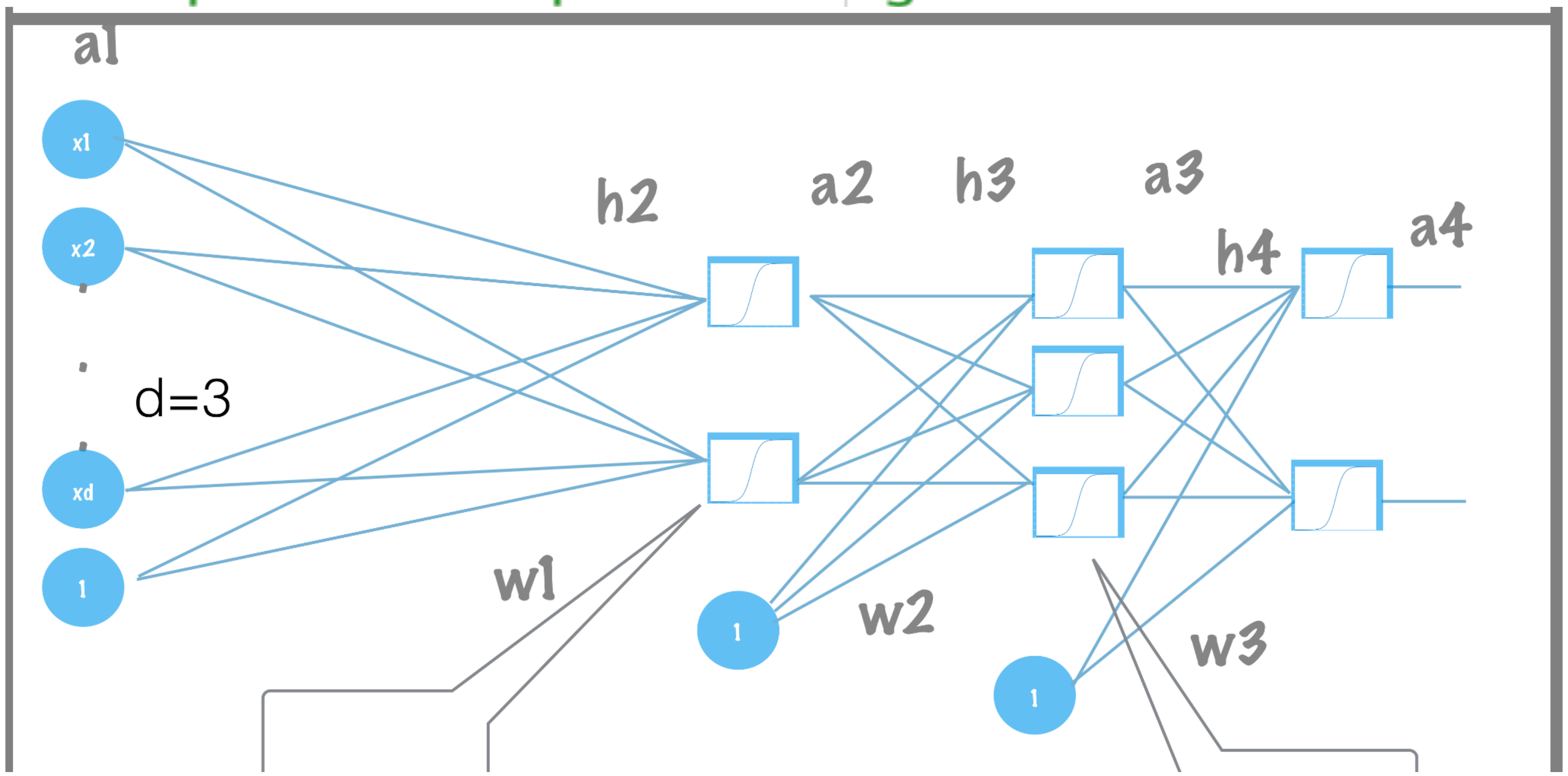
Object-oriented Matlab programming for deep learning

Neural Networks

```
classdef perceptrons
    %   Numerical Analysis of AM NDHU
    %   CopyRight Reserved
    %   Dr. Jiann-Ming Wu
    %   object-oriented programming
```

Neural networks

```
properties
    layers; % number of layers in a network
    w;      % weight matrices n cells
    nL;     % linearity and nonlinearity of transformations of layers
    a;      % activation of  input, hidden layers and output
    u;      % gradient of output with respect to activations
    v;      % gradient of output with respect to stimuli
    gW;     % gradient of output with respect to weight matrices
end
```



Methods

```
function obj=perceptrons(layers,w,nL)

function obj=ff(obj,x)

function draw(obj)

function obj=cal_uv(obj)

function obj=cal_gW(obj)

function err_g=gradient_check(obj,x)
```

```
function obj=perceptrons(layers,w,nL)
% This method initiates a neural network.
% w collects all weight matrices and represents them in cells.

% The length of w must be (layers - 1), where variable layers denotes the nur
% all layers, including input, hidden and output layers.

% nL specifies nonlinearity of each hidden layer and output layer
% nL equals w in length
```

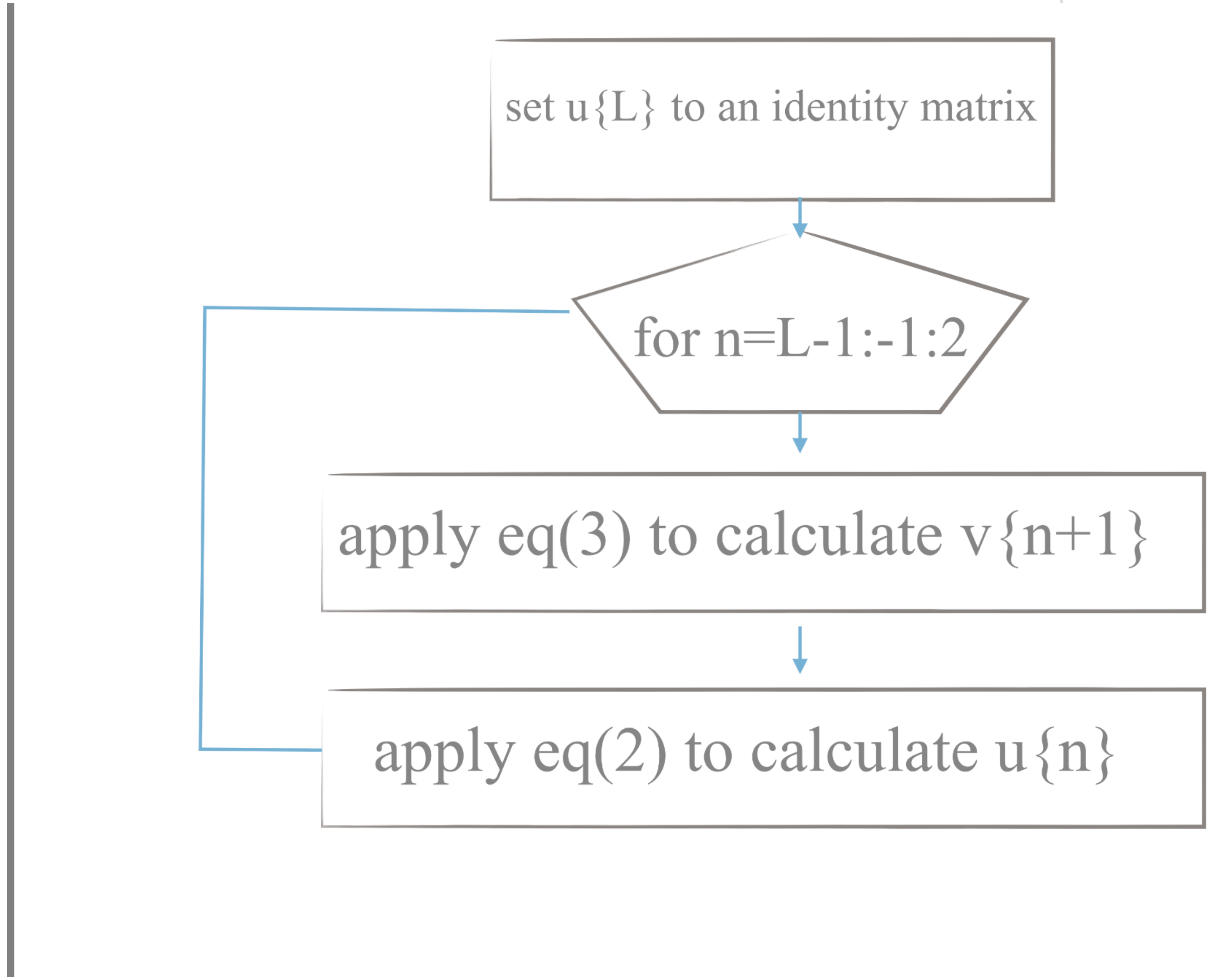
```
function obj=ff(obj,x)
% x is a matrix. Each row is an input pattern.
% This method translates x layer-by-layer forward to generate
% activations.
% a{1} denotes input and a{end} denotes output
%
```

Function construction
Deep composition
Extremely high input dimension
Learning and adaption
LIIII

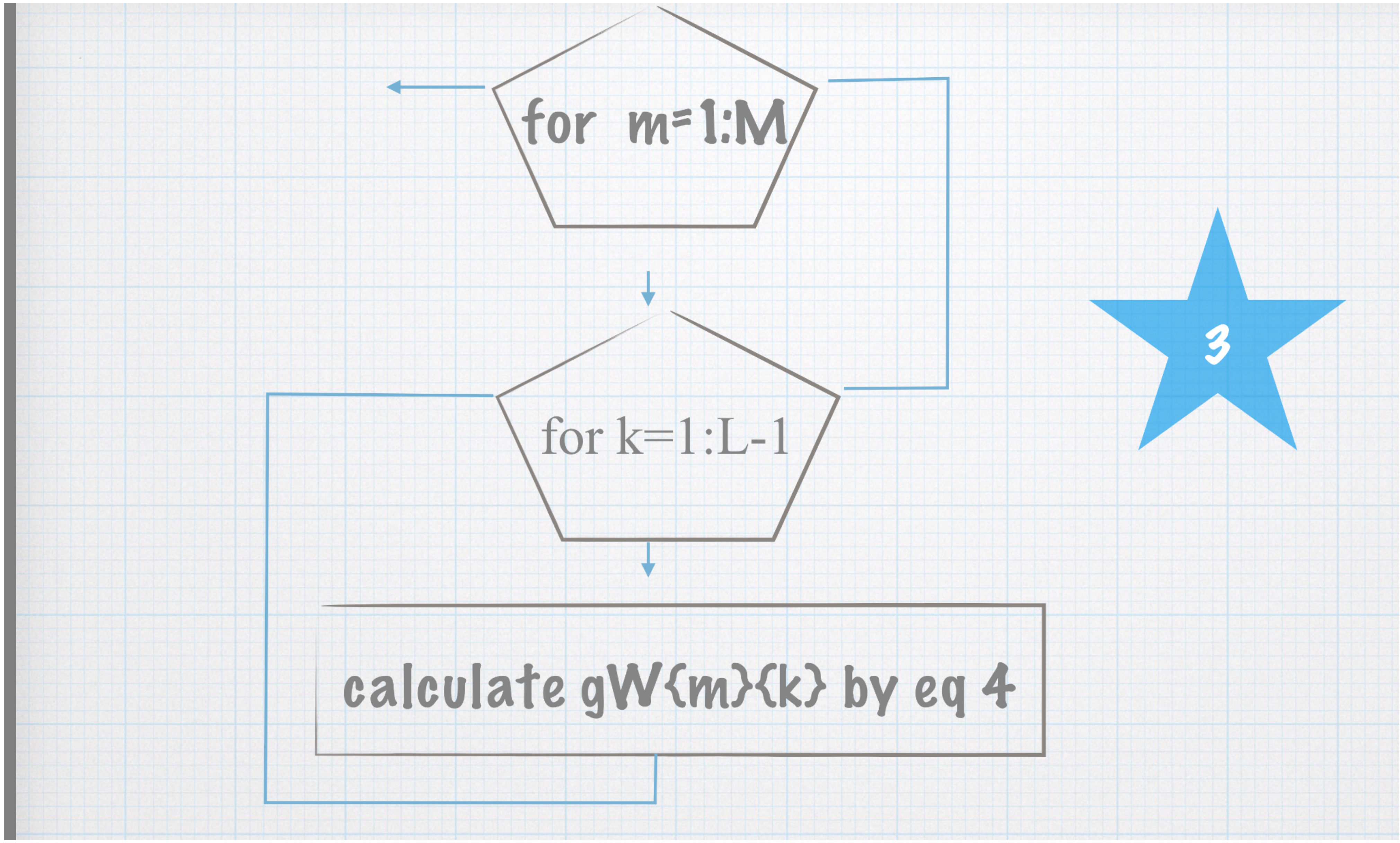
```matlab
function draw(obj)
% This method draws network functions by mesh
% It is asserted that dimension of input equals 2
% Activations will be reserved when existing
```

```matlab
function draw(obj)
  % This method draws network functions by mesh
  % It is asserted that dimension of input equals 2
  % Activations will be reserved when existing

    [d(1) d(2)]=size(obj.w{1});
    assert(d(1)==3,'exceed 2-dimensional inputs');
    tag = 0;
    if ~isempty(obj.a)
        ba=obj.a;
        tag = 1;
    end
    range=pi;
    L=obj.layers;
    x1=-range:0.1:range;
    x2=x1;
    M=size(obj.w{L-1},2);

    for i=1:length(x1)

        obj=obj.ff([x1(i)*ones(length(x1),1) x2']);
        y=obj.a{L};
        for m=1:M
        C{m}(i,:)=y(:,m)';
        end
    end
    for m=1:M
    mesh(x1,x2,C{m}'); hold on
    end
    if tag==1
        obj.a=ba;
    end
end
```

```
function obj=cal_uv(obj)
    %
    % calculate u{n} and v{n} for all n
    % according to flowchart 2 based on current activations
    % a{1} could contain a batch of input data. This method
    % realizes backpropagation. There are multiple output
    % components. u and v are two-ary cells, where entries of cells correspond
    % to output components and layers respectively
```



```matlab
function obj=cal_gW(obj)
    % This method calculates gradient of output with respect to w.
    % It is asserted that properties a and v are provided.
    % Here a{1} contains single data.
    % flowchart 3
```



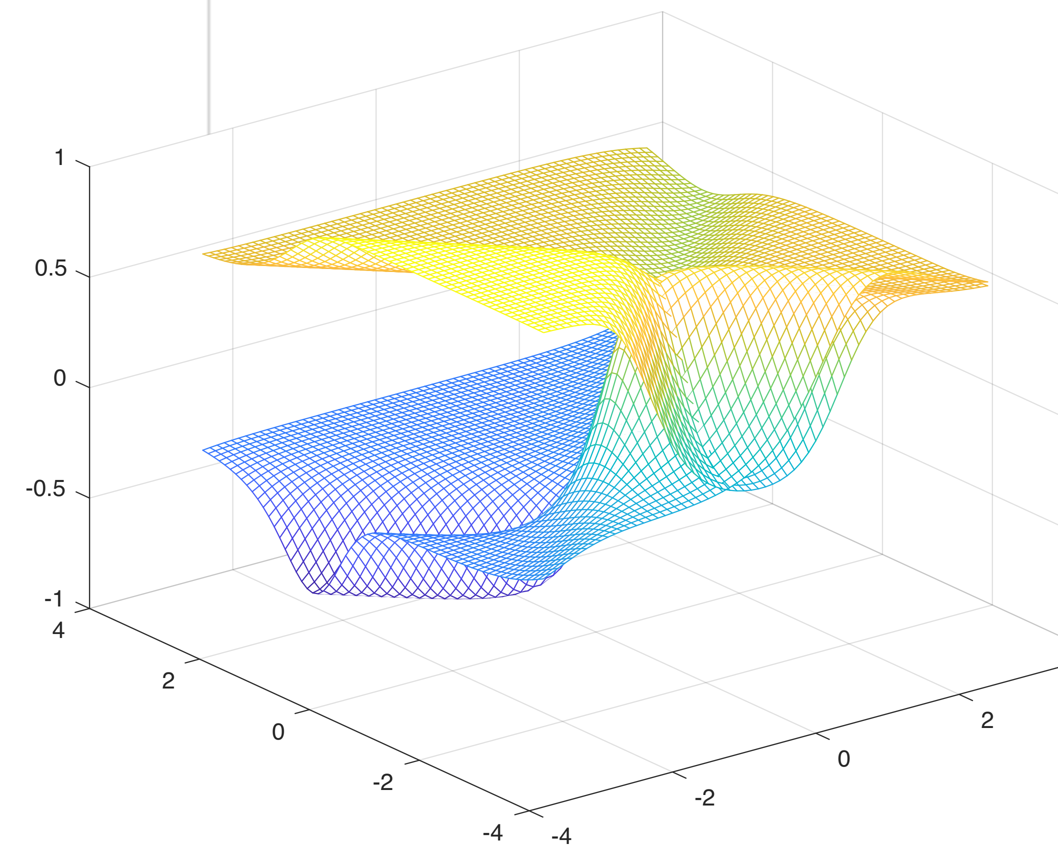
```
function err_g=gradient_check(obj,x)
    % Calculate gradient of output with respect to w by Richardson
    % Extrapolation by flow chart 4.
    % x conbtains single data
```

沒有服務 上午11:08 87%

Files

```
            obj=obj.ff(x);
            obj=obj.cal_uv();
            obj=obj.cal_gW();
            L=obj.layers;
            M=size(obj.w{L-1},2);
            z=0.01; err_g=0;
            for m=1:M
                for k=1:L-1
                    W_k=obj.w{k};
                    RE_gW = zeros(size(W_k));
                    for i=1:size(W_k,1)
                        for j=1:size(W_k,2)
[redacted]
                        end
                    end
                    err_g=err_g+sum(sum(abs(obj.gW{m}{k}-RE_gW')));
                end
            end
```

```
% define a net of deep perceptrons
w{1}=[1 1 2;1 -1 -1]';nL{1}='tanh';
w{2}=[1 -1 1;-1 -1 -1; -2 1.5 0.5]';nL{2}='tanh';
w{3}=[1 1 1/2 1;1 -1 1 -1]';nL{3}='tanh';
layers=4;
net=perceptrons(layers,w,nL);

% draw network functions if input dim is 2
net.draw();
```



```
x=rand(300,2)*10-5;
net=net.ff(x);
y=net.a{layers};
plot3(x(:,1),x(:,2),y,'.')
net=net.cal_uv(); % backpropagation
```

```
net=net.ff(x(10,:));
net=net.cal_uv();
net=net.cal_gW();
```

```
err=0;
for i=1:300
err_g=net.gradient_check(x(i,:));
err=err+err_g;
end
err/300
```