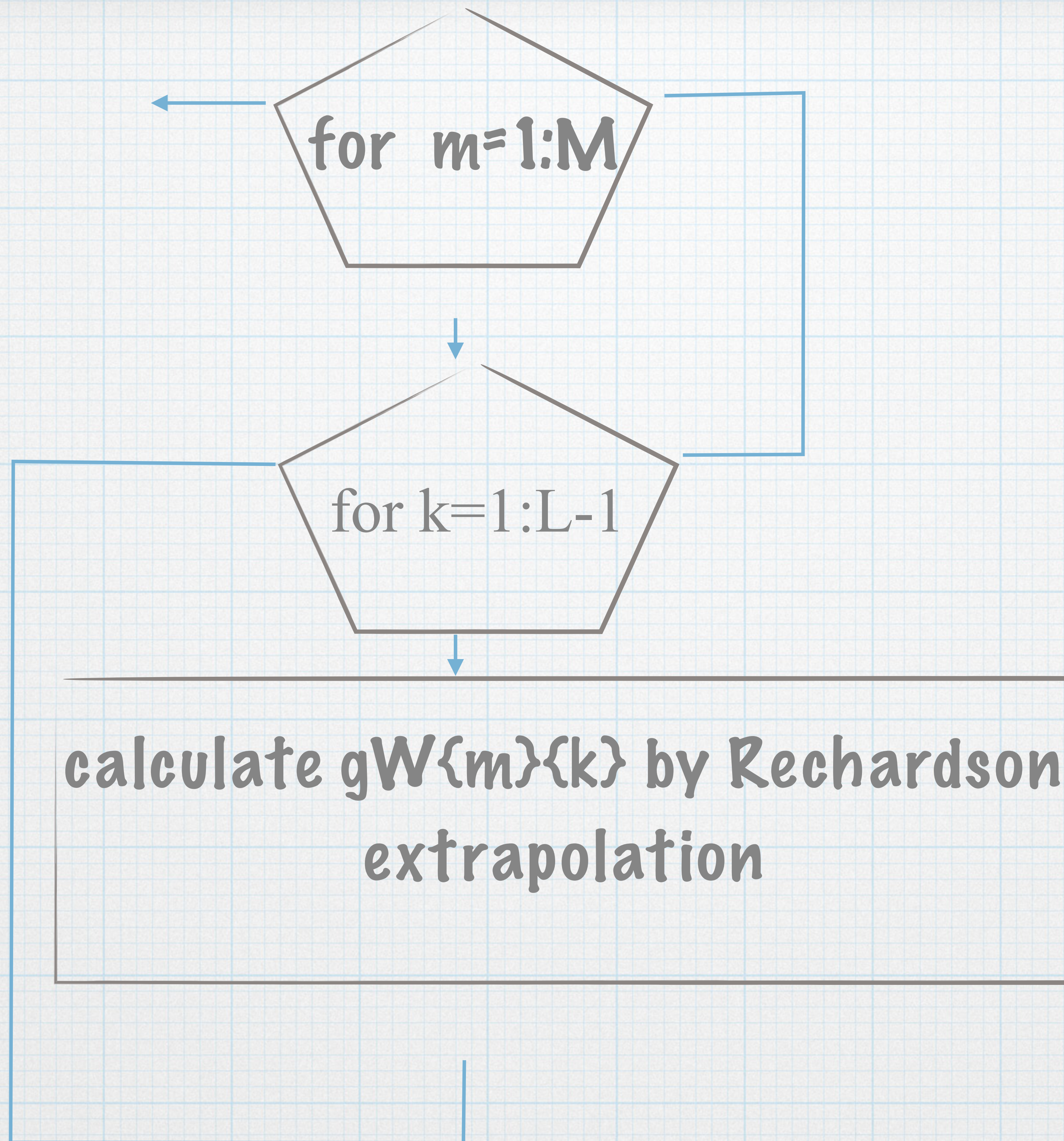
Least mean square method of learning deep neural networks

$$\frac{dy}{dW_{m2}[i,j]} = ?$$


for m=1:M
for k=1:L-1
calculate gW{m}{k} by Rechardson extrapolation

Richardson Extrapolation

set small z

for
each element in

Perturb it by adding z,-z,z/2,-z/2 respectively and determine four corresponding output y{m}, denoted by f1,f2,f3,f4

set corresponding element in gW{m}{k} to extrapolation determined by f1-f4

```
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
    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)
                % calculate f1 f2 f3 f4
                obj.w{k}(i,j)=W_k(i,j)+z;
                obj =obj.ff(x);f1=obj.a{L}(m);
                obj.w{k}(i,j)=W_k(i,j)-z;
                obj=obj.ff(x);f2=obj.a{L}(m);
                obj.w{k}(i,j)=W_k(i,j)+z/2;
                obj=obj.ff(x);f3=obj.a{L}(m);
                obj.w{k}(i,j)=W_k(i,j)-z/2;
                obj=obj.ff(x);f4=obj.a{L}(m);
                g1=(f1-f2)/(2*z);g2=(f3-f4)/z;
                RE_gW(i,j)=g2+(g2-g1)/3;
                obj.w{k}(i,j)=W_k(i,j);
            end
        end
        err_g=err_g+sum(sum(abs(obj.gW{m}{k}-RE_gW')));
    end
end
```

Deep perceptrons

a1 h2 a2 h3 a3 h4 a4

x1
x2
xd
1

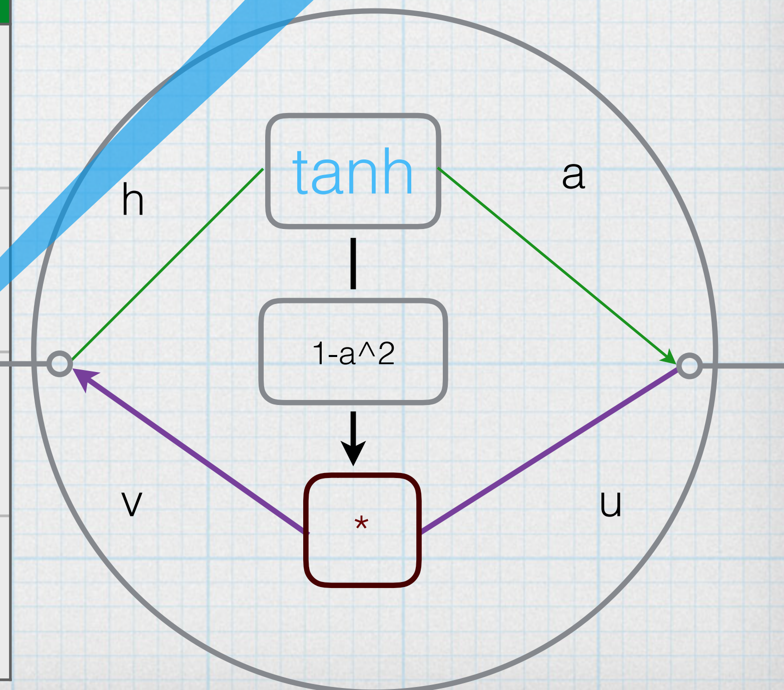
d=3

w1 w2 w3

1
1

batch input

Size	u{m}	v{m}	backward. w'	h	W	a	
4	nx2	nx2		nx2		nx2	Output
3	nx3	nx3	3x2	nx3	2x4	nx3	
2	nx2	nx2	2x3	nx2	3x3	nx2	
1			3x2		2x4	nx3	



Square error

$$E(W) = \sum_t \sum_m e_m^2[t]$$

$$e_m[t] = y_m[t] - y_m(t|\theta)$$

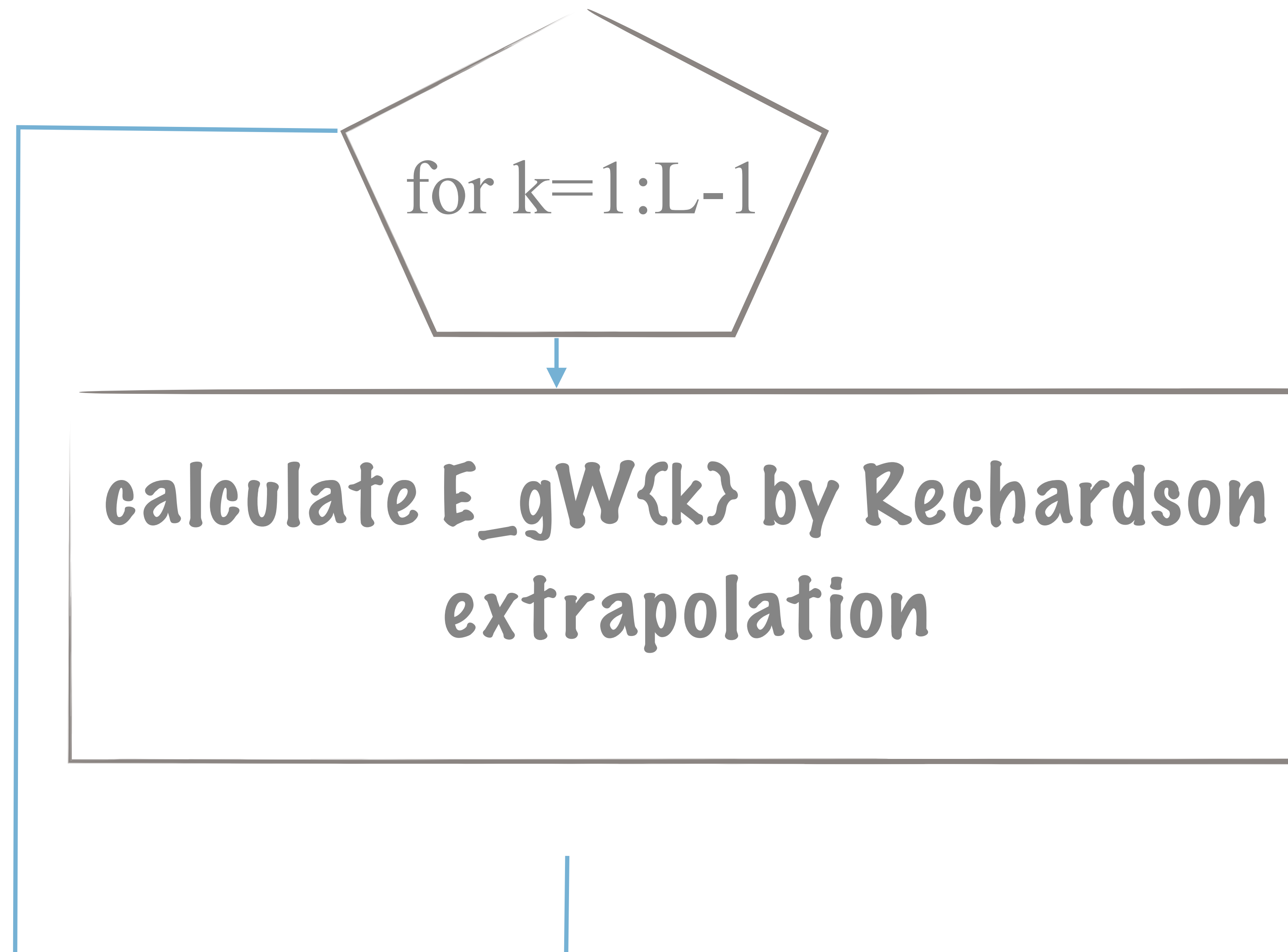
$y_m[t]$ denotes the m*th* component of the desired network output in response to x[t].

$y_m(t|\theta)$ denotes the real network output in response to x[t]

```
function obj=cal_se(obj,y)
    % calculate square error
    assert(~isempty(obj.a),'empty a')

    L=obj.layers;
    M=size(obj.w{L-1},2);
    obj.e=y-obj.a{L};
    obj.se=sum(sum(obj.e.^2));
end
```

Richardson Extrapolation for approximating gradient of square error with respect to weight matrices



Richardson Extrapolation

set small z

for each element in w{k}

Perturb it by adding z,-z,z/2,-z/2 respectively and determine four corresponding square error, denoted by f1,f2,f3,f4

set corresponding element in E_gW{k} to extrapolation determined by f1-f4

use cal_se

gradient of E with respect to W

$$\frac{dE}{dW} = \sum_m \sum_t \frac{de_m^2[t]}{dW} = \sum_m \frac{dE_m}{dW}$$

$$\frac{de_m^2[t]}{dW} = 2e_m[t]\frac{de_m[t]}{dW}$$

$$= -2e_m[t]\frac{dy_m(t|\theta)}{dW}$$

A computation task rises to determine the gradient of E with respect to W, given the gradient of output components with respect to W

$$e_m[t] = y_m[t] - y_m(t|\theta)$$

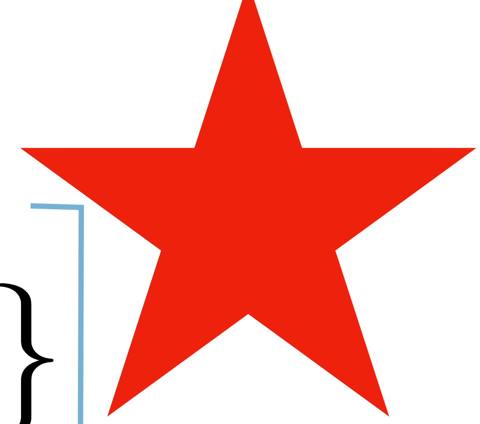
$$\frac{dE_m}{dW} = \sum_t \frac{de_m^2[t]}{dW}$$

$$= \sum_t -2e_m[t]\frac{dy_m(t|\theta)}{dW}$$

$$\frac{dE_m}{dW} = -2\sum_t e_m[t]\frac{dy_m(t|\theta)}{dW}$$

Recall

$$\frac{dE_m}{dW} = -2\sum_t e_m[t]\frac{dy_m(t|\theta)}{dW}$$



$$h\{n+1\} = W\{n\}\begin{bmatrix} a\{n\} \\ 1 \end{bmatrix}$$

y{m} denotes an output component

$$\frac{dy\{m\}}{dW\{n\}} = \frac{dy\{m\}}{dh\{n+1\}}\frac{dh\{n+1\}}{dW\{n\}}$$

denoted by $v_m\{n+1\}$

$$= v_m\{n+1\}\frac{dh\{n+1\}}{dW\{n\}}$$

$$= v_m\{n+1\}\begin{bmatrix} a\{n\} \\ 1 \end{bmatrix}^T$$

eq 4

$$h\{n+1\} = W\{n\} \begin{bmatrix} a\{n\} \\ 1 \end{bmatrix}$$

$$\frac{dy\{m\}}{dW\{n\}} = v\{n+1\} \begin{bmatrix} a\{n\} \\ 1 \end{bmatrix}^T$$

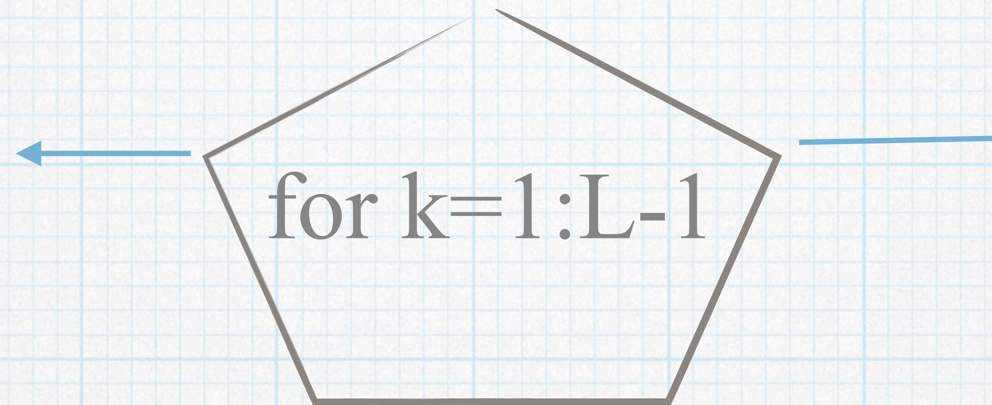
$$\frac{dE_m}{dW_n} = -2 \sum_t e_m[t] \frac{dy_m(t|\theta)}{dW_n}$$

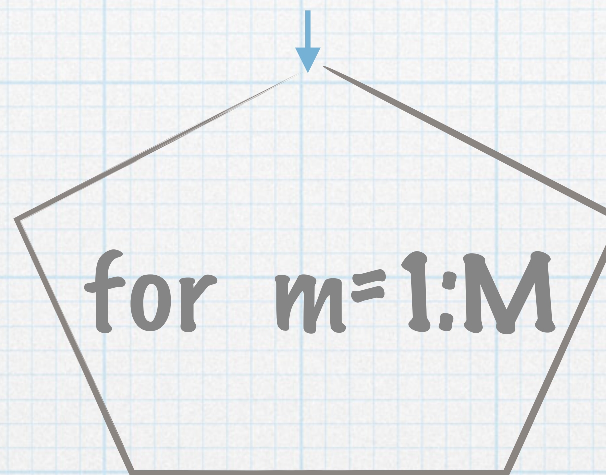

```
[b1 b2]=size(a{k});
a1=[a{k};ones(b1,1)];
e1=e{m}*ones(1,b2+1);
E_gW{m}{k}=-2*v{k+1}*(a1.*e1)
```

batch size = 300

Size	e{m}	u{m}	v{m}	backward . w'	h	W	a	
4	300x1	2x300	2x300		300x2		300x2	Output
3		3x300	3x300	3x2	300x3	2x4	300x3	
2		2x300	2x300	2x3	300x2	3x3	300x2	
1				3x2		2x4	300x3	

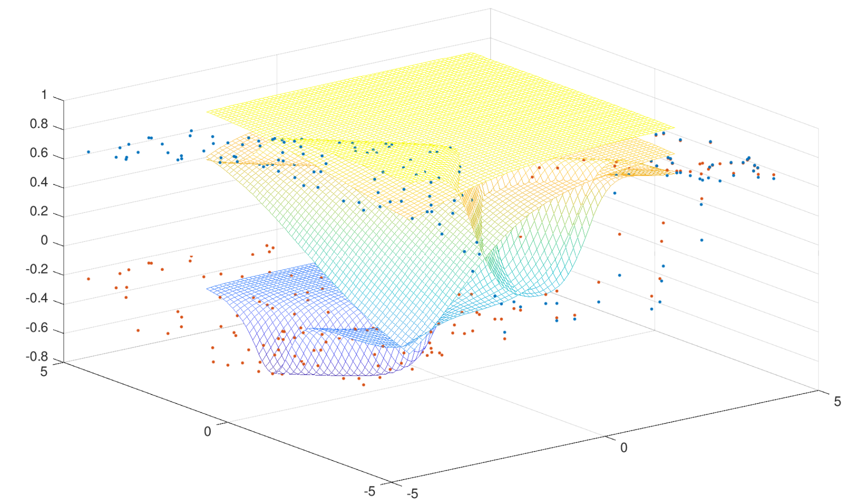
function cal_E_gW

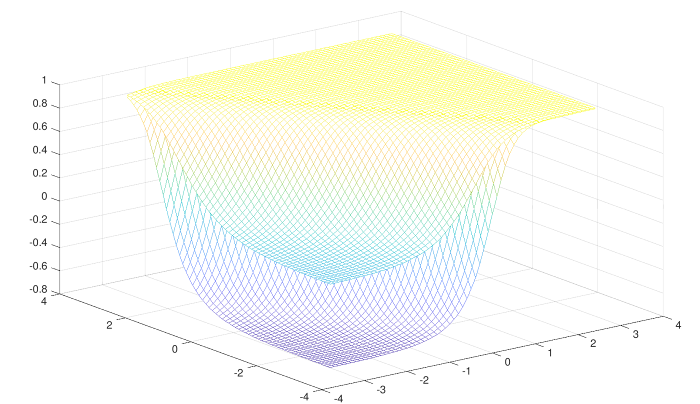





```
if m==1
    set E_gW{k} to the result calculated by eq 6
else
    add the result determined by eq 6 to E_gW{k}
end
```

```
% 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
% consider y as desired output
% randomize weight matrices
for i=1:layers-1
    W{i}=rand(size(w{i}));
end
net2=perceptrons(layers,W,nL);
figure
net2.draw();
net2=net2.ff(x);
y_hat=net2.a{layers};
net2=net2.cal_uv(); % backpropagation
net2=cal_E_gW(net2,y);
err_g=net2.gradient_check2(x,y);
```





Add method cal_E_gW

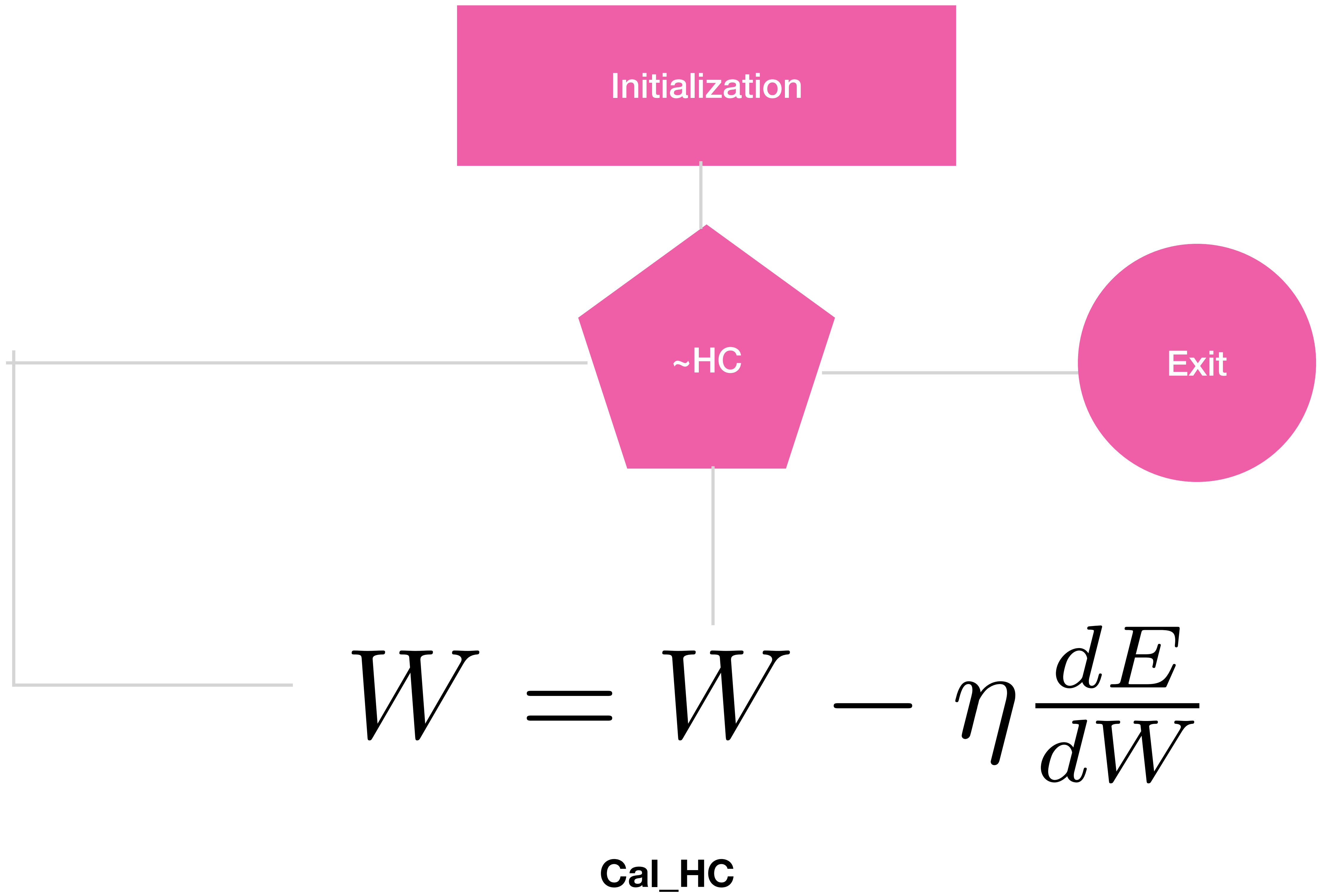
Add method gradient_check2

Gradient descent method

- calculate E_gW{k} for all k
- set w{k} to w{k} - eta * E_gW{k} for all k
- eta is a small positive number

$$W = W - \eta \frac{dE}{dW}$$

$$Wopt = \arg \min_{\{W\}} E(W)$$


Initialization
~HC
Exit
$W = W - \eta \frac{dE}{dW}$
Cal_HC