Levenberg-Marquardt Learning for CNN

RBF and Convolutional Neural Networks

Files

MATLAB Drive > demo_fsolve_MLP_learning.m

```matlab

function demo_mlp_learning()
syms c    % adaptable parameters in an MLP network
M=20;d=2;
% initialization
c0=rand(1,M*(d+1)+M+1)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
z=rand(100,2)*2*pi-pi;
y=g_sin(z(:,1),z(:,2));
plot3(z(:,1),z(:,2),y,'.');

h = g_hat2(z(:,1),z(:,2),c0,M);
```

```
% 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,M);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options);
mean((learning_mlp(c_zero,z(:,1),z(:,2),y,M)).^2)

end
```

```
% a nonlinear system for MLP_learning
function F = learning_mlp(c,x1,x2,y,M)
A = [x1 x2 ones(length(x1),1)];
d=2;W=[];
%W=reshape(c(1:M*(d+1)),d+1,M);
for j=1:M
   W=[W c((j-1)*(d+1)+1:j*(d+1))'];
end
v =tanh(A*W);
F = v*c(M*(d+1)+1:end-1)'+ones(length(x1),1)*c(end);
F =F-y;
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
```

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

```
function demo_mlp_learning()
syms c   % adaptable parameters in an MLP network
M=20;d=2;
% initialization
c0=rand(1,M*(d+1)+M+1)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
z=rand(100,2)*2*pi-pi;
y=g_sin(z(:,1),z(:,2));
plot3(z(:,1),z(:,2),y,'.');

h = g_hat2(z(:,1),z(:,2),c0,M);

% 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,M);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options);
mean((learning_mlp(c_zero,z(:,1),z(:,2),y,M)).^2)
mean((g_hat2(z(:,1),z(:,2),c_zero,M)-y).^2)

% testing phase
z_test=rand(100,2)*2*pi-pi;
y_test=g_sin(z_test(:,1),z_test(:,2));
mean((g_hat2(z_test(:,1),z_test(:,2),c_zero,M)-y_test).^2)
end
```

```
% testing phase
z_test=rand(100,2)*2*pi-pi;
y_test=g_sin(z_test(:,1),z_test(:,2));
mean((g_hat2(z_test(:,1),z_test(:,2),c_zero,M)-y_test).^2)
```

data with noises

- Training data
  - paired predictors and desired targets
  - A desired target is the response of a target function to a predictor
  - A target could be added with a white noise
- Testing data
  - paired predictors and desired targets oriented from the target function that generates paired training data

Excercise

- Revise your Levenberg-Marquardt learning
  - Learning built-in parameters of MLP subject to training data
  - Report the mean square error for training
  - Verifying trained MLP with testing data
  - Report the mean square error for testing
  - Compare the two mean square errors

Large-scaled training data

- Levenberg-Marquardt learning should be verified with large scaled training data
- Enlarge training data
- Training time versus data size

Architecture

- MLP : multilayer perceptrons
- BRF : radial basis functions
- Convolutional neural networks
- Revise your Levenberg-Marquardt learning for RBF neural networks
- Revise your Levenberg-Marquardt learning for Convolutional neural networks

RBF Network function

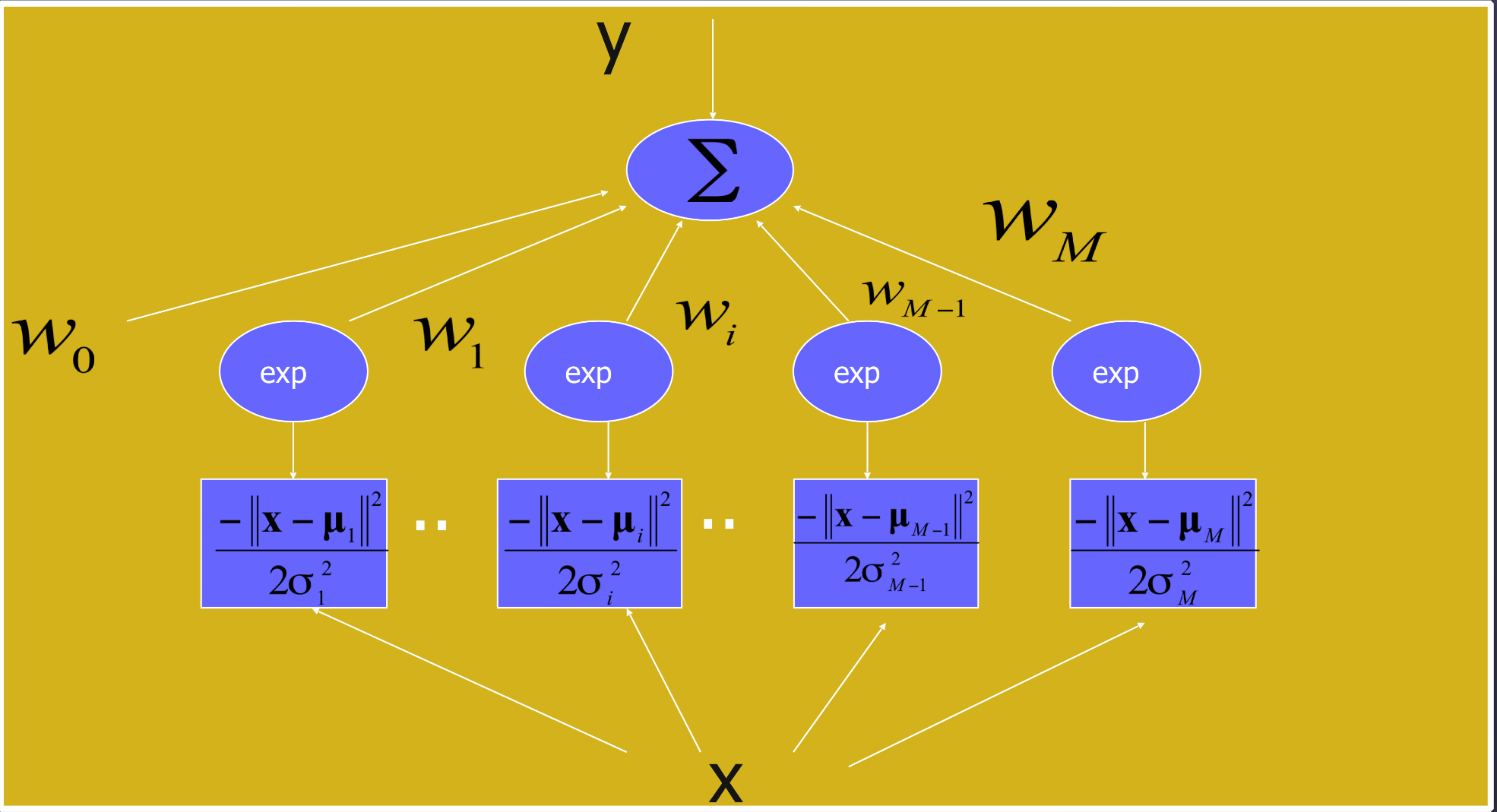
$$y(t \mid \theta) = G(\mathbf{x}[t] \mid \theta)$$

$$= w_0 + \sum_{m=1}^{M} w_m \exp\left(-\frac{\left\|\mathbf{x}[t] - \boldsymbol{\mu}_m\right\|^2}{2\sigma_m^2}\right)$$

Network parameter

$$\theta = \{\mathrm{w}_\mathrm{i}\}_i \cup \{\boldsymbol{\mu}_i\}_i \cup \{\sigma_i\}_i$$

RBF Network


y
Σ
w_0
w_1
w_i
w_{M-1}
w_M
exp
exp
exp
exp
$\frac{-\|\mathbf{x}-\boldsymbol{\mu}_1\|^2}{2\sigma_1^2}$
..
$\frac{-\|\mathbf{x}-\boldsymbol{\mu}_i\|^2}{2\sigma_i^2}$
..
$\frac{-\|\mathbf{x}-\boldsymbol{\mu}_{M-1}\|^2}{2\sigma_{M-1}^2}$
$\frac{-\|\mathbf{x}-\boldsymbol{\mu}_M\|^2}{2\sigma_M^2}$
x

iPad 上午11:16 134.208.26.59 30%

matlab conv2 & convn

https://www.mathworks.com/help/matlab/ref/conv2.html?requestedDomain=www.mathworks.com&requestedDomain=www.mathworks.com

In applications such as image processing, it can be useful to compare the input of a convolution directly to the output. The `conv2` function allows you to control the size of the output.

Create a 3-by-3 random matrix A and a 4-by-4 random matrix B. Compute the full convolution of A and B, which i by-6 matrix.

```
A = rand(3);
B = rand(4);
Cfull = conv2(A,B)
```

```
Cfull =

    0.7861    1.2768    1.4581    1.0007    0.2876    0.0099
    1.0024    1.8458    3.0844    2.5151    1.5196    0.2560
    1.0561    1.9824    3.5790    3.9432    2.9708    0.7587
    1.6790    2.0772    3.0052    3.7511    2.7593    1.5129
    0.9902    1.1000    2.4492    1.6082    1.7976    1.2655
    0.1215    0.1469    1.0409    0.5540    0.6941    0.6499
```

iPad 上午11:16 30%
134.208.26.59

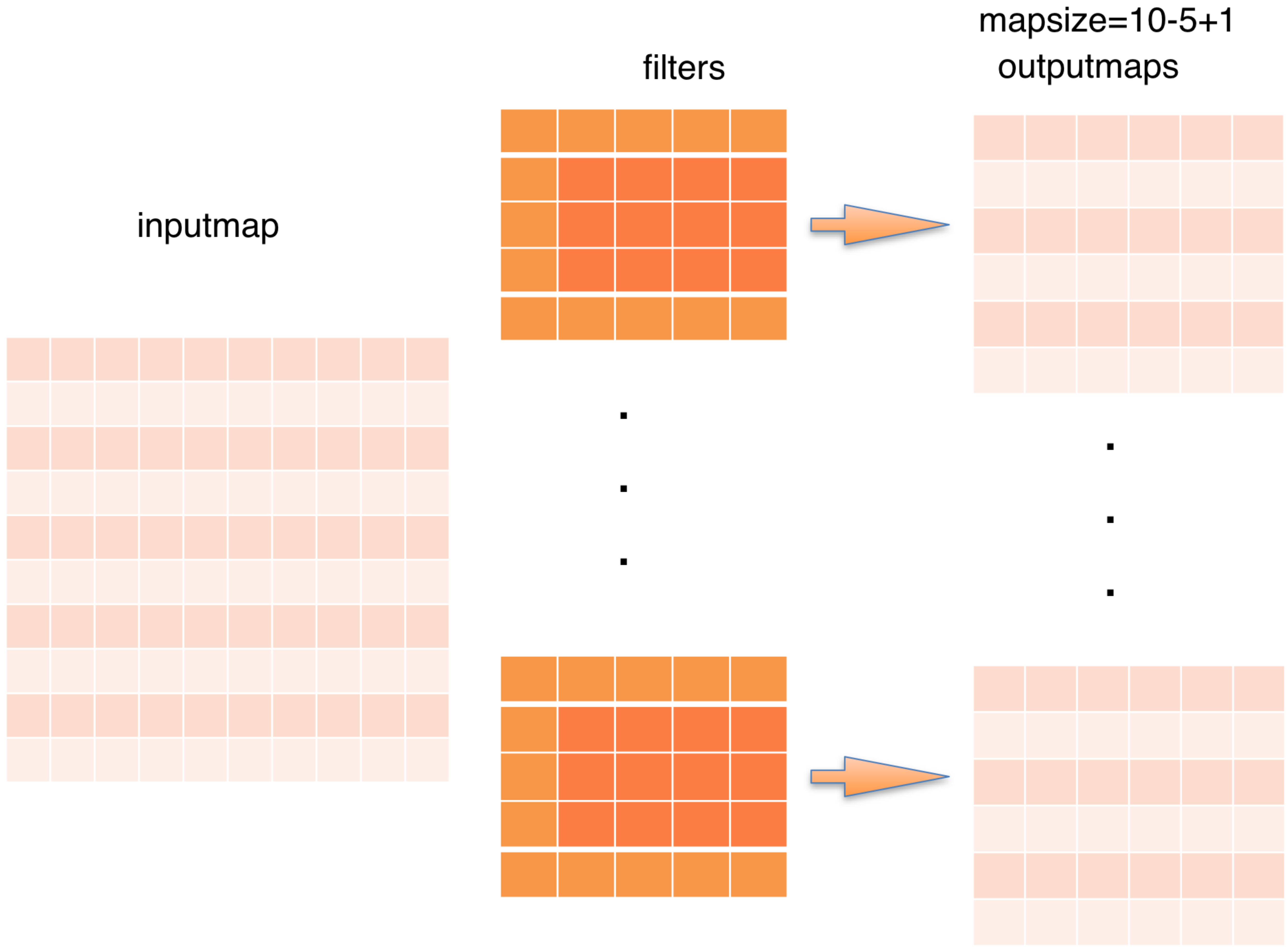

```
B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> C=conv2(A,B)

C =

    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.3333    0.6667    1.0000    1.0000    0.6667    0.3333
    0.2222    0.4444    0.6667    0.6667    0.4444    0.2222
    0.1111    0.2222    0.3333    0.3333    0.2222    0.1111

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
```

iPad 上午11:17 29%
134.208.26.59
w 新網頁 plasticity... tobermory... Perceptro... 134.208.2... Turing ma... Marvin Mi... Files - Dro... Multilayer... 134.208...

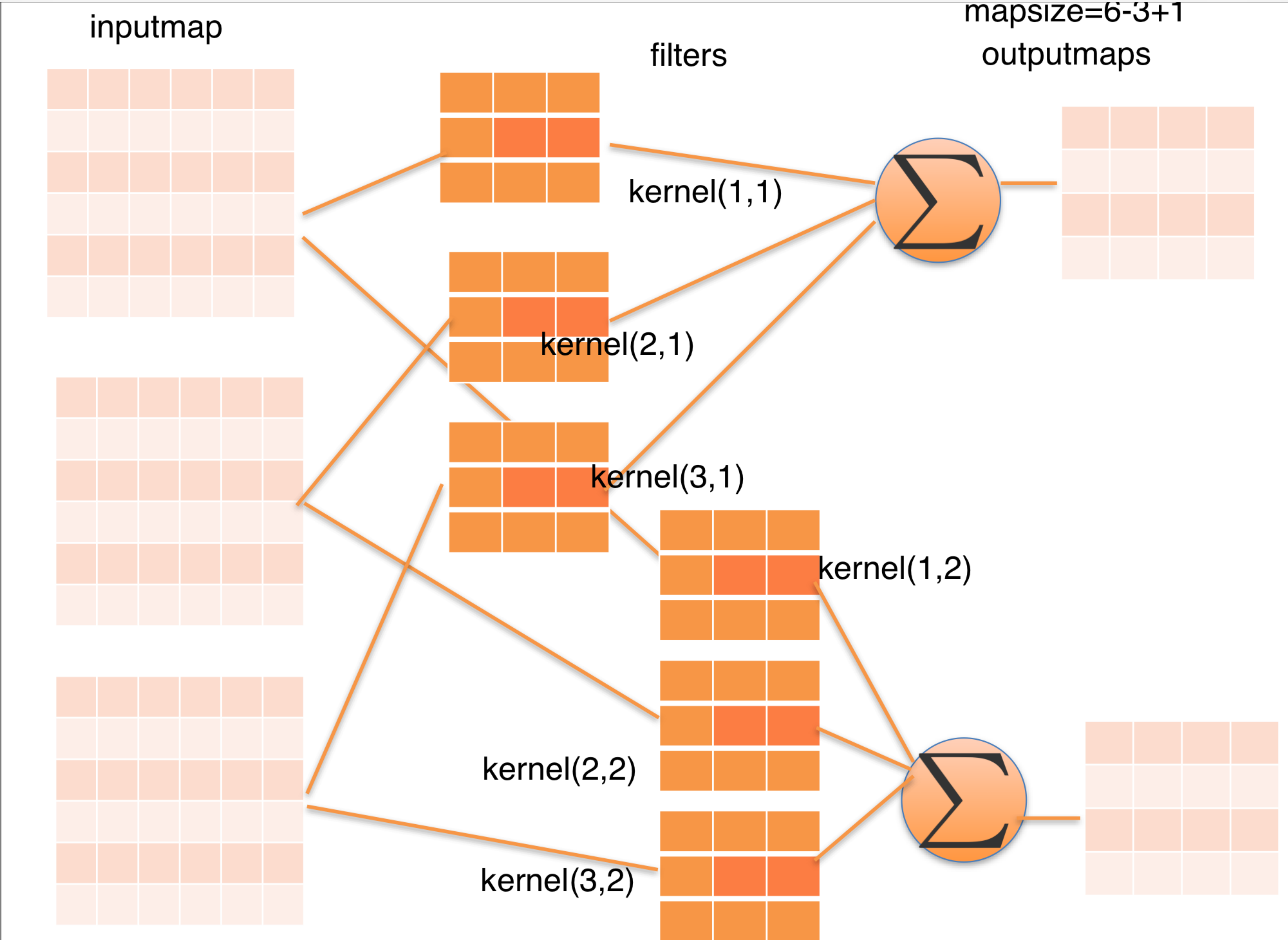
```
>> C=conv2(B,A,'valid')

C =

    1.0000    1.0000
    1.0000    1.0000

>> B

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

```
    inputmaps = 1;
    mapsize = size(squeeze(x(:, :, 1)));

    for l = 1 : numel(net.layers)   %  layer
        if strcmp(net.layers{l}.type, 's')
            mapsize = mapsize / net.layers{l}.scale;
            assert(all(floor(mapsize)==mapsize), ['Layer ' num2str(l) ' size must be integer. Actual: ' num2str(mapsize)]);
            for j = 1 : inputmaps
                net.layers{l}.b{j} = 0;
            end
        end
        if strcmp(net.layers{l}.type, 'c')
            mapsize = mapsize - net.layers{l}.kernelsize + 1;
            fan_out = net.layers{l}.outputmaps * net.layers{l}.kernelsize ^ 2;
            for j = 1 : net.layers{l}.outputmaps  %  output map
                fan_in = inputmaps * net.layers{l}.kernelsize ^ 2;
                for i = 1 : inputmaps  %  input map
                    net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5) * 2 * sqrt(6 / (fan_in + fan_out));
                end
                net.layers{l}.b{j} = 0;
            end
            inputmaps = net.layers{l}.outputmaps;
        end
    end
```

iPad 上午11:17 29%
134.208.26.59



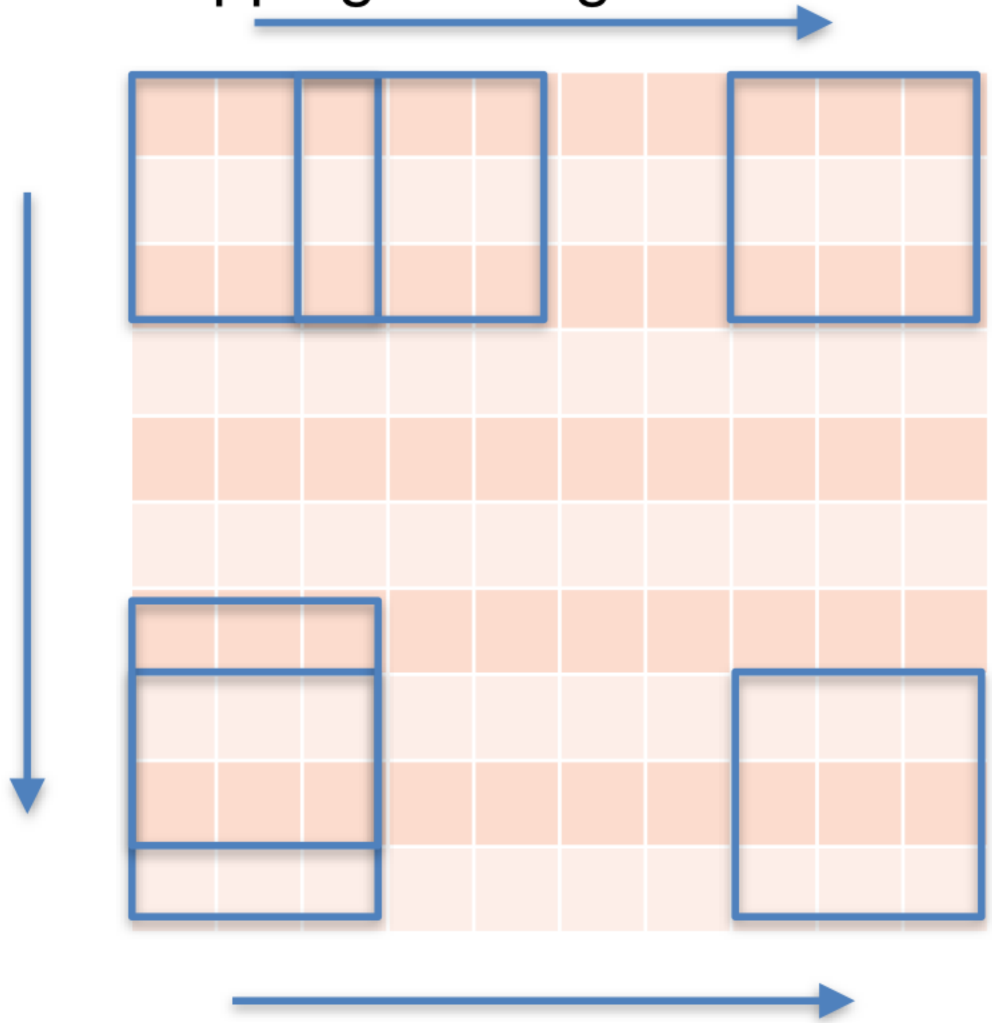
iPad 上午11:18 29%
134.208.26.59

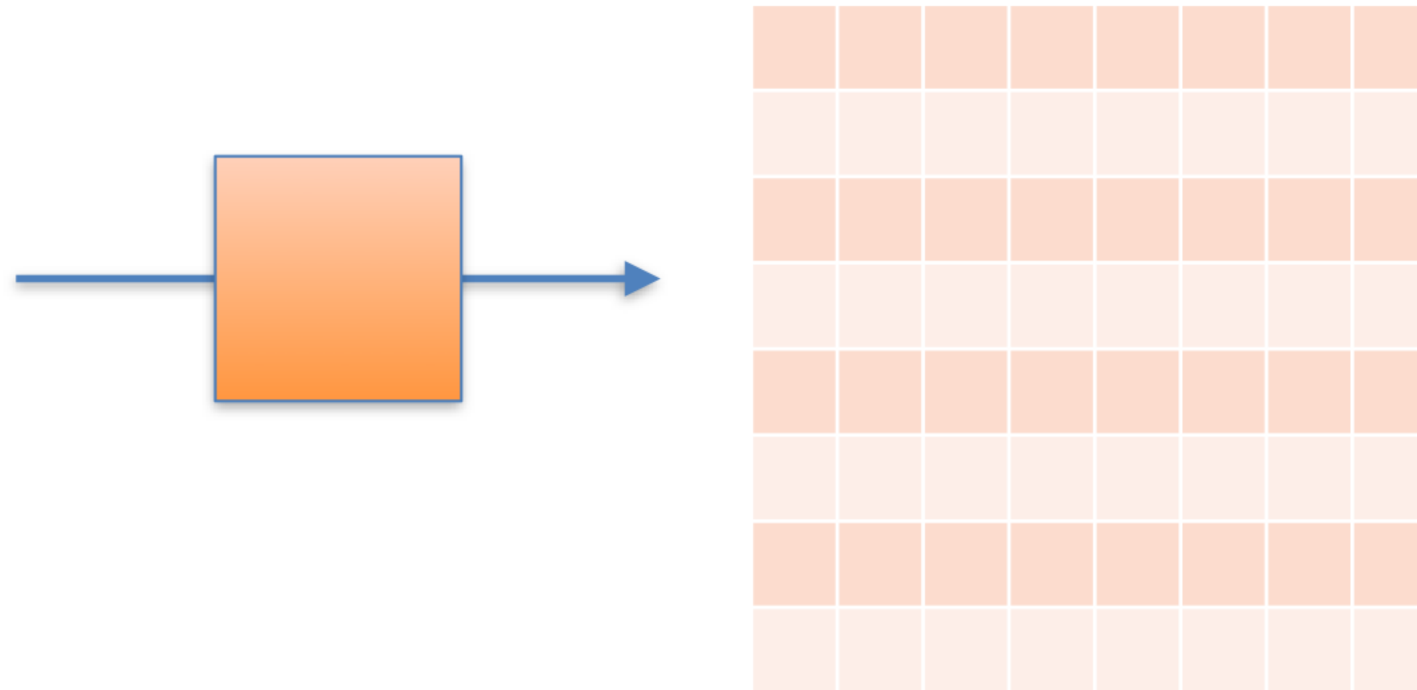


iPad 上午11:18 29%
134.208.26.59
w 新網頁 plasticity... tobermory... Perceptro... 134.208.2... Turing ma... Marvin Mi... Files - Dro... Multilayer... 134.208...

2D Convolution

overlapping moving of filters



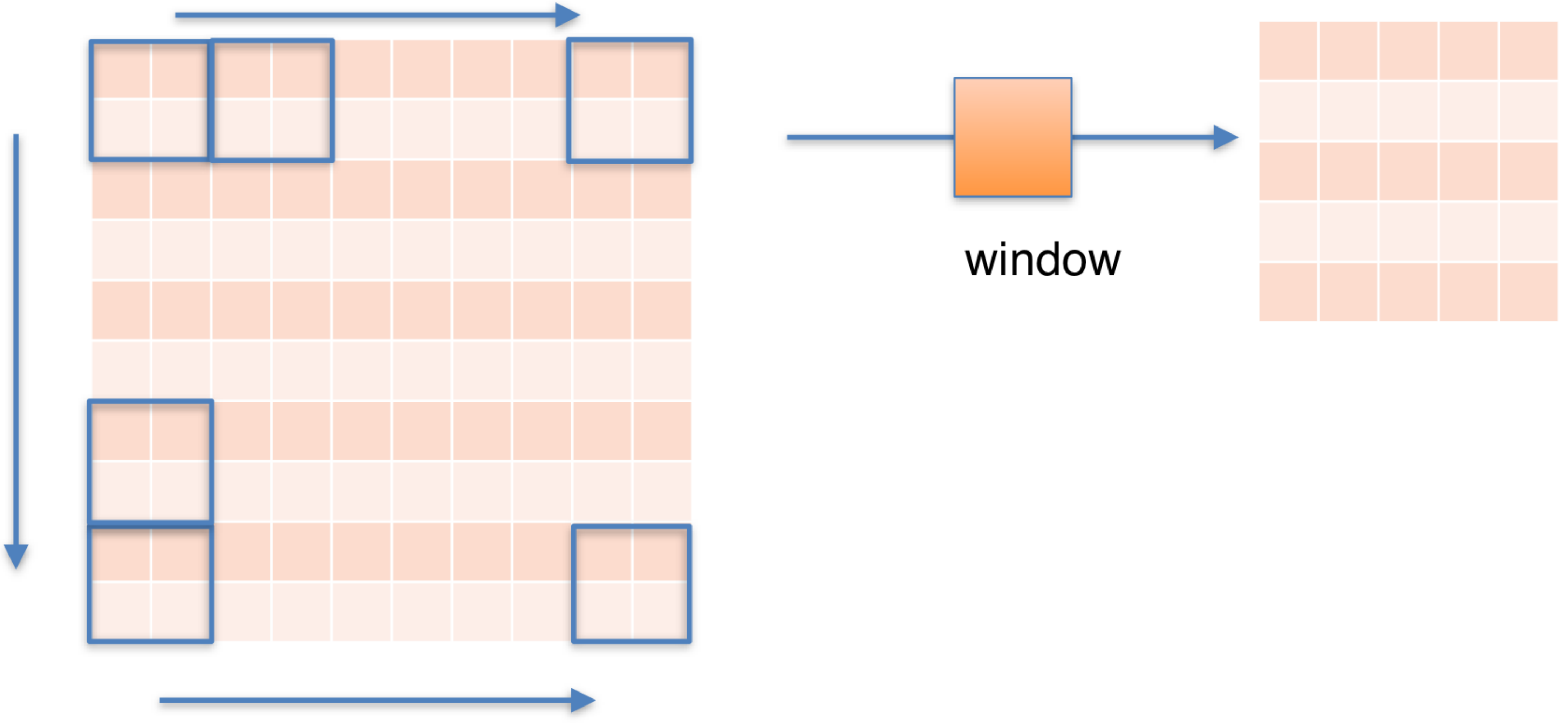


```
>> C=conv2(B,A,'valid')

C =

    1.0000    1.0000
    1.0000    1.0000

>> B

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1

>> A

A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

```
A =

    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111
    0.1111    0.1111    0.1111

>>
```

Down Sampling

Non-overlapping moving sampling



```
function demo_mlp_learning()
syms c   % adaptable parameters in an MLP network
M=3;
% initialization
c0=rand(M,M)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
A=ones(3,3)/9;
B=ones(4,4);
y=conv2(A,B);
size(g_hat(A,B))
size(y)

% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_cnn(c,B,y);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options)
mean(mean((learning_cnn(c_zero,B,y)).^2))
end

% a nonlinear system for MLP_learning
function F = learning_cnn(c,B,y)
F = conv2(c,B);
F =F-y;
end

function h = g_hat(c,B)
   h=conv2(c,B);
end
```

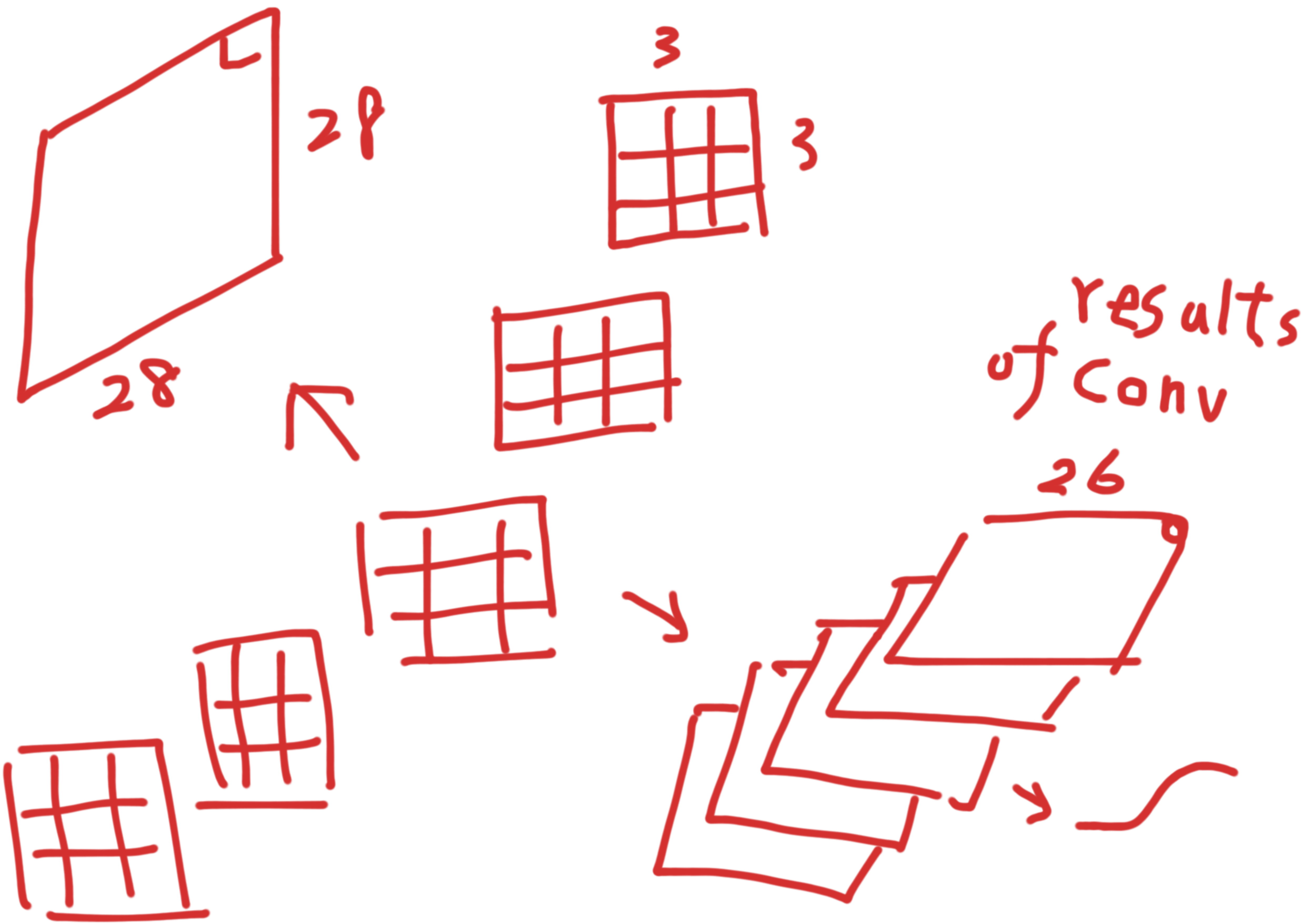
```
function demo_mlp_learning()
syms c   % adaptable parameters in an MLP network
M=3;
% initialization
c0=rand(M,M)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
A=ones(3,3)/9;
B=ones(4,4);
y=conv2(A,B);
size(g_hat(A,B))
size(y)

% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_cnn(c,B,y);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options)
mean(mean((learning_cnn(c_zero,B,y)).^2))
end

% a nonlinear system for MLP_learning
function F = learning_cnn(c,B,y)
F = conv2(c,B);
F =F-y;
end
```

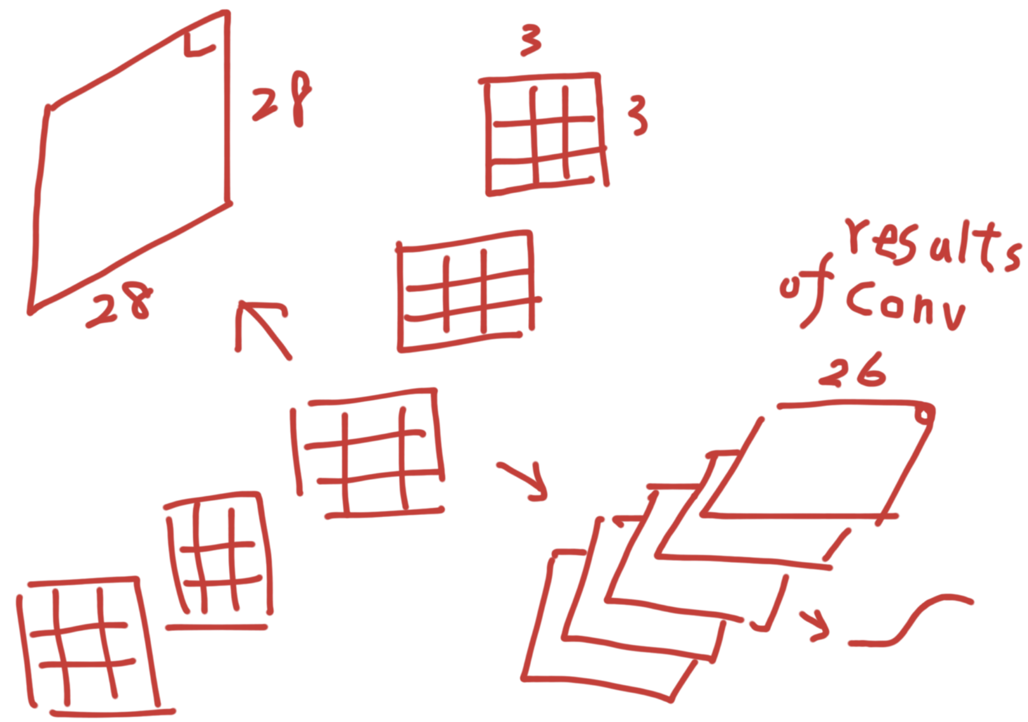
$$A = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$$

$$\downarrow$$

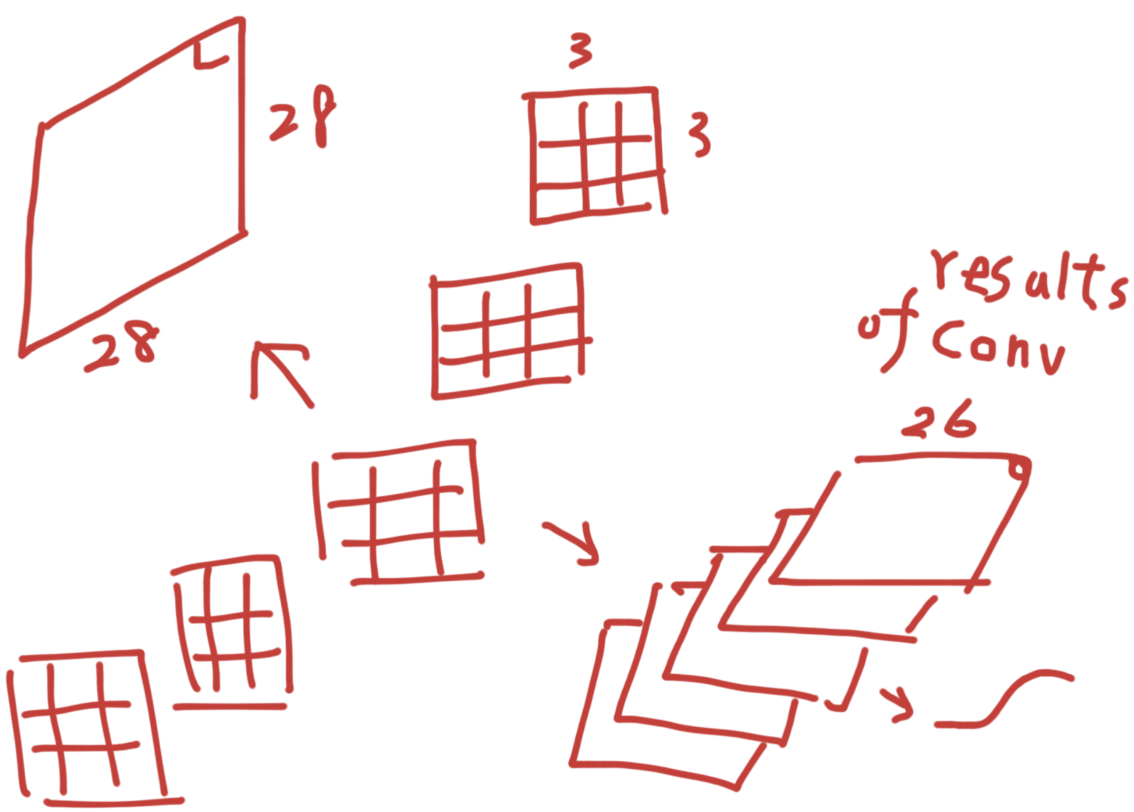
$$B \rightarrow \boxed{\text{conv2}} \rightarrow y$$

$$y = \text{conv2}(A, B)$$


28
28
3
3
results of conv
26



3
3
28
28
results
of conv
26



3
3
28
28
results
of conv
26

```
syms c   % adaptable parameters in an MLP network
M=3;
% initialization
c0=rand(M,M)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
A=ones(3,3)/9;
B=ones(4,4);
y=conv2(A,B);
size(g_hat(A,B))
size(y)

% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_cnn(c,B,y);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options)
mean(mean((learning_cnn(c_zero,B,y)).^2))
end

% a nonlinear system for MLP_learning
function F = learning_cnn(c,B,y)
F = conv2(c,B);
F =F-y;
end

function h = g_hat(c,B)
   h=conv2(c,B);
end
```

```
ans =

     3     3     2


B(:,:,1) =

  Columns 1 through 2

    0.4666    0.0183
   -0.6914   -0.5728
    0.3525    0.6314

  Column 3

   -0.1302
    0.1618
   -0.0351


B(:,:,2) =

  Columns 1 through 2

   -0.4853    0.3251
    0.8508    0.2434
   -0.9067   -0.9273

  Column 3

   -0.2751
   -0.3668
    0.1994


ans =

    30    30   100

>>
```

iPad 上午9:59 95%

Files

MATLAB Drive > demo_fsolve_CNN.m

```matlab
c0=rand(M,M,filter_no)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
A=rand(28,28,100);
B=rand(M,M,filter_no)*2-1
y = cnn_my(A,B);
size(y)

% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_cnn(c,A,y);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options)
mean(mean(mean((learning_cnn(c_zero,A,y)).^2)))
end

% a nonlinear system for CNN_learning
function F = learning_cnn(C,A,y)
filter_no = size(C,3);
    F=0;
    for i=1:filter_no
```

```
measured by the gradient.

<stopping criteria details>


c_zero(:,:,1) =

  Columns 1 through 2

    0.4666    0.0183
   -0.6914   -0.5728
    0.3525    0.6314

  Column 3

   -0.1302
    0.1618
   -0.0351


c_zero(:,:,2) =

  Columns 1 through 2

   -0.4853    0.3251
    0.8508    0.2434
   -0.9067   -0.9273

  Column 3

   -0.2751
   -0.3668
    0.1994


ans =

   1.1785e-26

>>
```

```
function demo_fsolve_CNN()
syms c   % adaptable parameters in an MLP network
M=3;filter_no=2;
% initialization
c0=rand(M,M,filter_no)*2-1;
size(c0)
% preparation of training data
% uniform sampling
% Substitute to the target function g
A=rand(28,28,100);
B=rand(M,M,filter_no)*2-1
y = cnn_my(A,B);
size(y)

% Levenberg-Marquardt method
options = optimoptions('fsolve','Algorithm', 'levenberg-marquardt')
% specification of a nonlinear system for MLP_learning
f = @(c)learning_cnn(c,A,y);
% apply fsolve
% Verification of c_zero
c_zero = fsolve(f,c0,options)
mean(mean(mean((learning_cnn(c_zero,A,y)).^2)))
end
```

```
% a nonlinear system for CNN_learning
function F = learning_cnn(C,A,y)
filter_no = size(C,3);
   F=0;
   for i=1:filter_no
      F=F+tanh(convn(A,C(:,:,i)));
   end
F =F-y;
end

function h = cnn_my(A,B)
   filter_no = size(B,3);
   h=0;
   for i=1:filter_no
      h=h+tanh(convn(A,B(:,:,i)));
   end
end
function h = cnn_hat(C,A)
   filter_no = size(C,3);
   h=0;
   for i=1:filter_no
      h=h+tanh(convn(A,C(:,:,i)));
   end
end
```

Applicability of Levenberg-Marquardt method for learning CNN

- LM method is verified for learning CNN

- A simple CNN is composed of K=2 filters in the first hidden layer. The convolution results are transferred by the tanh function and added to form the final output.

- The input is a 3-ary matrix. The matrix size is 28x28x100. There are 100 28x28 input patterns.