

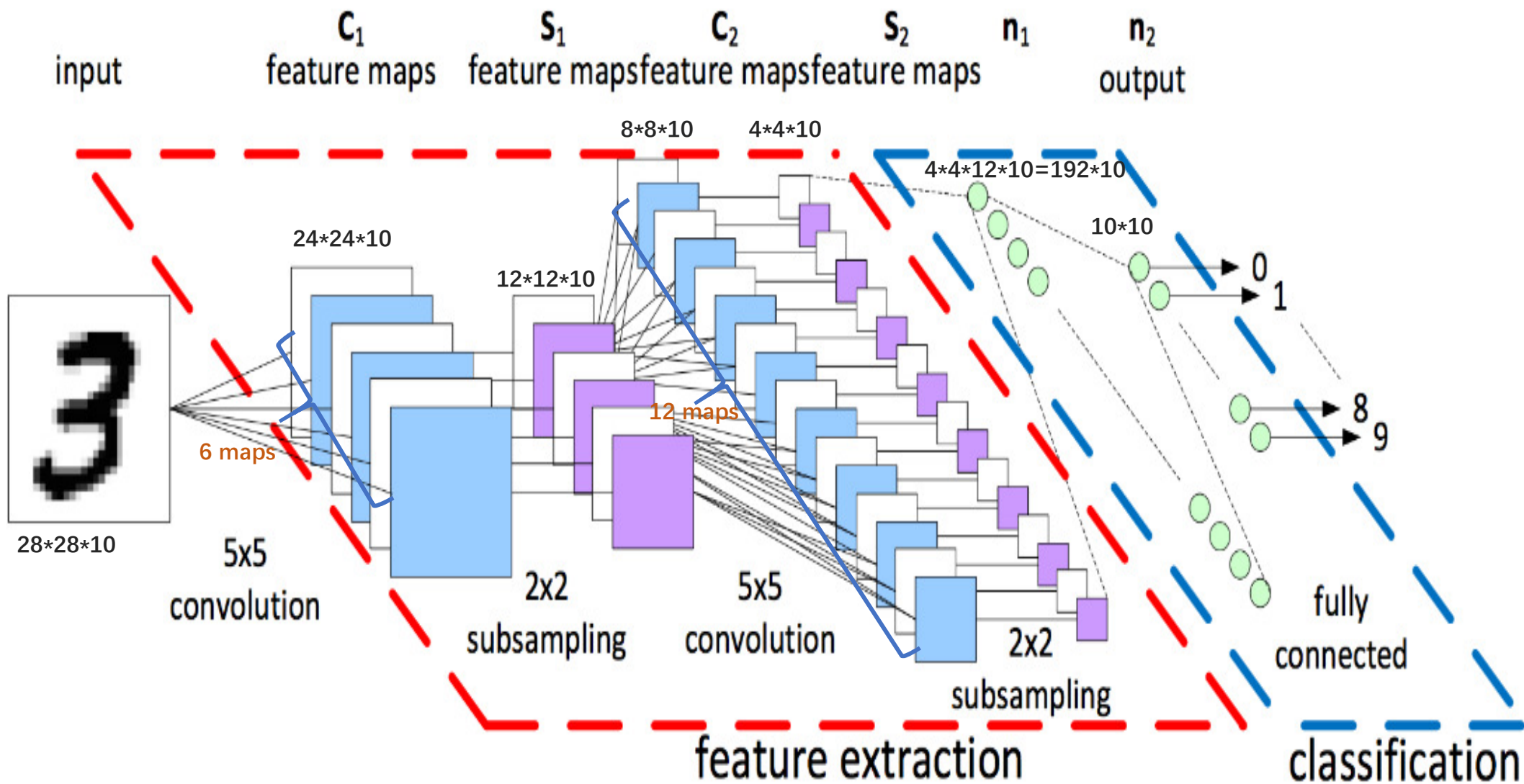
深度网络基本功

第四讲

Convolutional Neural Network编程实践

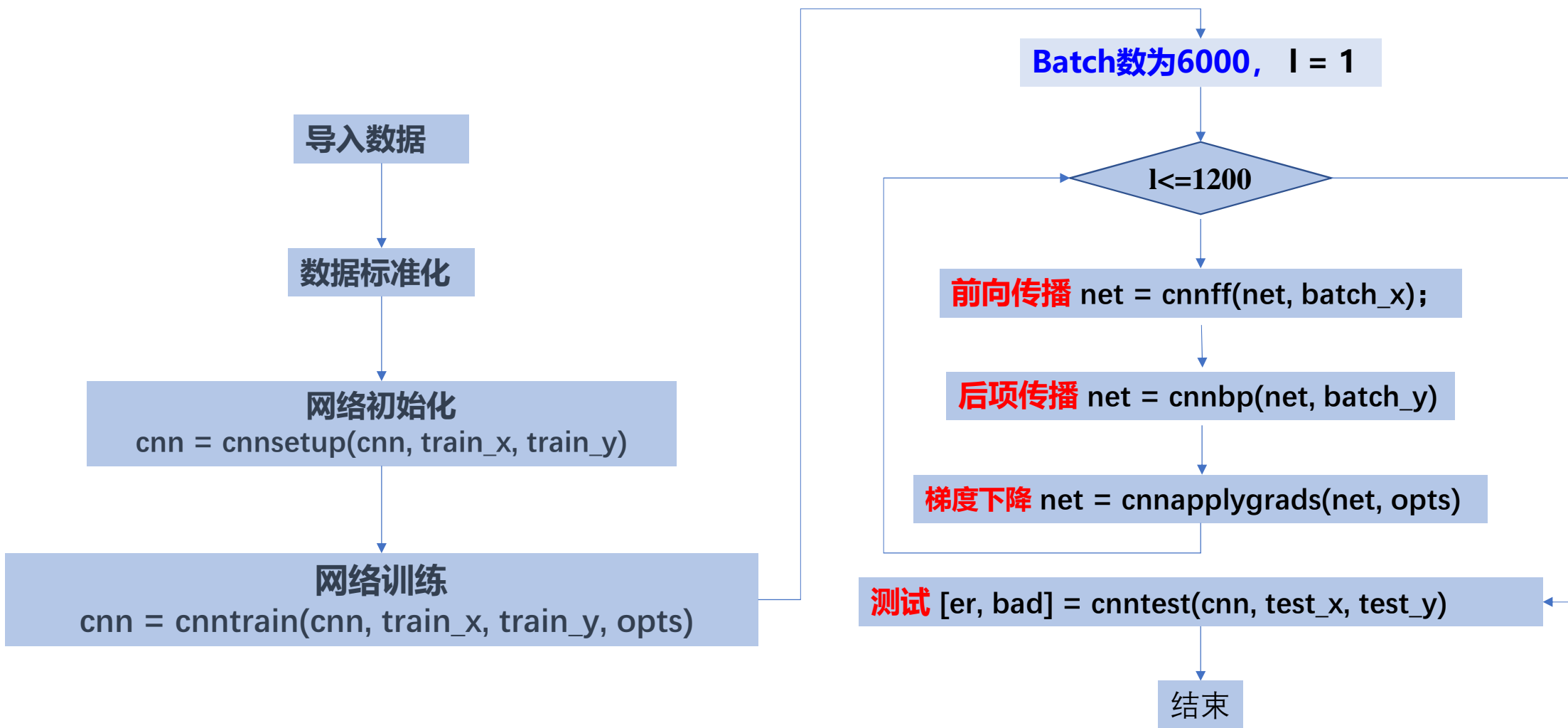
武德安

电子科技大学



Architecture

function test_example_CNN逻辑图







test_example_CNN () : 主程序

```
function test_example_CNN
load mnist_uint8; %读取数据
% 把图像的灰度值变成0~1, 因为本代码采用的是sigmoid激活函数
train_x = double(reshape(train_x',28,28,60000))/255;
test_x = double(reshape(test_x',28,28,10000))/255;
train_y = double(train_y');
test_y = double(test_y');

%% 卷积网络的结构为 6c-2s-12c-2s, 生成28*28*60000的tensor (60000万个样本, 原数据转置后每一个样本为一列)
% 1 epoch 会运行大约200s, 错误率大约为11%。而 100 epochs 的错误率大约为1.2%。

rand('state',0) %指定状态使每次运行产生的随机结果相同

cnn.layers = {
    struct('type','i') % 输入层
    struct('type','c','outputmaps',6,'kernelsize',5) % 卷积层, 输出的
    feature map 为6个, 卷积定义5*5
    struct('type','s','scale',2) % pooling层, 下采样率=2
    struct('type','c','outputmaps',12,'kernelsize',5) % 卷积层, 输出的
    feature map 为12个, 卷积定义5*5
    struct('type','s','scale',2) % pooling层下采样率=2
};
%
```

工作区	
名称 ▼	值
 train_y	60000x10 uint8
 train_x	60000x784 uint8
 test_y	10000x10 uint8
 test_x	10000x784 uint8

MATLAB help: reshape(X,M,N,P,...) or reshape(X,[M,N,P,...]) returns an N-D array with the same elements as X but reshaped to have the size M-by-N-by-P-by-.... The product of the specified dimensions, M*N*P*..., must be the same as NUMEL(X).

```
opts.alpha = 1; % 梯度下降的步长
```

```
opts.batchsize = 50; % 每次批处理50张图
```

```
opts.numepochs = 1; % 所有图片循环处理一次
```

```
cnn = cnnsetup(cnn, train_x, train_y); % 初始化CNN
```

```
cnn = cnnttrain(cnn, train_x, train_y, opts); % 训练CNN
```

```
[er, bad] = cnntest(cnn, test_x, test_y); % 测试CNN
```

```
%plot mean squared error
```

```
figure; plot(cnn.rL);
```

```
assert(er<0.12, 'Too big error');
```

cnnsetup(net, x, y): CNN初始化

```
function net = cnnsetup(net, x, y)
```

```
    assert(~isOctave() || compare_versions(OCTAVE_VERSION, '3.8.0', '>='), ['Octave 3.8.0 or greater is required for CNNs as  
there is a bug in convolution in previous versions. See http://savannah.gnu.org/bugs/?39314. Your version is '  
myOctaveVersion]); %判断版本
```

```
    inputmaps = 1; % 由于网络的输入为1张特征图，因此inputmaps为1
```

```
    mapsize = size(squeeze(x(:, :, 1))); %squeeze():除去x中为1的维度，即得到28*28, mapsize=28*28
```

Matlab help: floor(X) rounds the elements of X to the nearest integers towards minus infinity.

```
    for l = 1 : numel(net.layers) % 网络层数 net.layer=5
```

```
        if strcmp(net.layers{l}.type, 's') % 如果是pooling层
```

```
            mapsize = mapsize / net.layers{l}.scale; % pooling之后图的大小，若为subsampling层（pooling层），图像长宽则各除2，  
            %相当于下采样，这里net.layers{3}.scale= net.layers{5}.scale=2
```

```
            assert(all(floor(mapsize)==mapsize), ['Layer ' num2str(l) ' size must be integer. Actual: ' num2str(mapsize)]);  
            %保证除2后为整数
```

```
            for j = 1 : inputmaps
```

```
                net.layers{l}.b{j} = 0; % 偏置项, net.layers{1}.b{1} = 0, net.layers{2}.b{1} = 0, ..., net.layers{l}.b{6} = 0
```

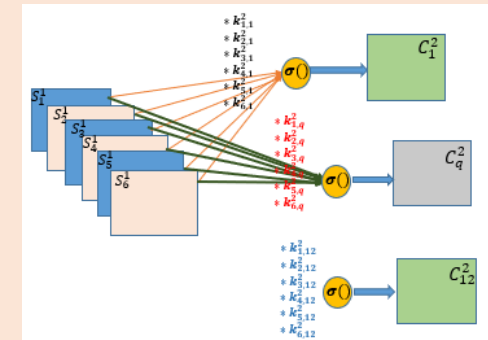
```
            end
```

```
    end
```

```

if strcmp(net.layers{l}.type, 'c') % 如果是卷积层
    mapsize = mapsize - net.layers{l}.kernelsize + 1; % 卷积后图像大小为 原图像长宽-卷积核长宽+1
    fan_out = net.layers{l}.outputmaps * net.layers{l}.kernelsize ^ 2; % 上一层连到该层的权值参数总个数
    % outputmaps (输出特征图的个数) net.layers{2}.outputmaps =6, 卷积的长宽为5*5, 总共参数个数为6*5*5=150
    % net.layers{4}.outputmaps =12, 卷积的长宽为5*5, 总共参数个数为12*5*5=300
    for j = 1 : net.layers{l}.outputmaps % 第l层特征图的数量net.layers{2}.outputmaps =6, net.layers{4}.outputmaps =12
        fan_in = inputmaps * net.layers{l}.kernelsize ^ 2; % 上一层连到该层的第j个特征图的权值参数, inputmaps=1, 总参数个数为
        1*5*5=25
        for i = 1 : inputmaps % 上一层特征图的数量
            net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5) * 2 * sqrt(6 / (fan_in + fan_out)); % 初始化权值, 见论文Understanding
the difficulty of training deep feedforward neural networks, 生成5*5的初始卷积核
        end
        net.layers{l}.b{j} = 0; % 偏置项
    end
    inputmaps = net.layers{l}.outputmaps; % 用该层的outputmaps更新inputmaps的值并为下一层所用
end
end
end

```



里面inputmaps是上一层的feature map数。outputmaps当前层的feature map数。其中有一行代码是
`net.layers{l}.k{i}{j} = (rand(net.layers{l}.kernelsize) - 0.5) * 2 * sqrt(6 / (fan_in + fan_out));`
 这一句应该就是初始化卷积核了。这里是随机生成的一个在某个范围内的kernelsize*kernelsize的卷积核。其中i和j分别对应inputmaps和outputmaps。也就是说是为每一个连接初始化了一个卷积核。

% 'onum' is the number of labels, that's why it is calculated using size(y, 1). If you have 20 labels so the output of the network will be 20 neurons.

% 'fvnum' is the number of output neurons at the last layer, the layer just before the output layer.

% 'ffb' is the biases of the output neurons.

% 'ffW' is the weights between the last layer and the output neurons. Note that the last layer is fully connected to the output layer, that's why the size of the weights is (onum * fvnum)

%尾部单层感知机（全连接层）的参数设置

fvnum = prod(mapsize) * inputmaps; % prod函数用于求数组元素的乘积, fvnum = 4*4*12 = 192, 是全连接层的输入数量

onum = size(y, 1); %输出节点的数量

net.ffb = zeros(onum, 1); % 最后一层bias的设置

net.ffW = (rand(onum, fvnum) - 0.5) * 2 * sqrt(6 / (onum + fvnum)); %初始化权重

end

权重设置为: $\text{random}(-1,1) / \sqrt{\frac{6}{(\text{输入神经元数量} + \text{输出神经元数量})}}$

$$k_{1,p}^1 \sim U \left(\pm \sqrt{\frac{6}{(1+6) \times 5^2}} \right)$$
$$k_{p,q}^2 \sim U \left(\pm \sqrt{\frac{6}{(6+12) \times 5^2}} \right)$$
$$W \sim U \left(\pm \sqrt{\frac{6}{192+10}} \right)$$

cnnttrain.m

cnnttrain(net, x, y, opts)训练模块

该函数用于训练CNN。

生成随机序列，每次选取一个batch（50）个样本进行训练。

批训练：计算50个随机样本的梯度，求和之后一次性更新到模型权重中。

在批训练过程中调用：

Cnnff.m 完成前向过程

Cnnbp.m 完成误差传导和梯度计算过程

Cnnapplygrads.m 把计算出来的梯度加到原始模型上去

%net为网络,x是数据, y为训练目标, opts (options)为训练参数

function net = cnnttrain(net, x, y, opts) %m为图片样本的数量, size(x) = 28*28*60000

m = size(x,3); % m=600000

%batchsize为批训练时，一批所含图片样本数

nunbatches = m / opts.batchsize; %分批训练，得到训练批次batchsize=10, nunbatches = 60000/10=6000

net.rl = []; %rL是最小均方误差的平滑序列，绘图要用

for i = 1 : opts.numepochs %训练迭代 波数为1

%显示训练到第几个epoch，一共多少个epoch

disp(['epoch' num2str(i) '/' num2str(opts.numepochs)]);

%Matlab自带函数randperm(n)产生1到n的整数的无重复的随机排列，可以得到无重复的随机数。

%生成m（图片数量）1~n整数的随机无重复排列，用于打乱训练顺序

kk = randperm(n);

```

for l = 1 : numbatches %训练每个batch, 1: 60000
    %得到训练信号，一个样本是一层x(:, :, sampleOrder)，每次训练，取10个样本
    batch_x = x(:, :, kk((l-1) * opts.batchsize + 1 : l * opts.batchsize)); %3层tensor 28*28*10
    batch_y = y(:, kk((l-1) * opts.batchsize + 1 : l * opts.batchsize)); %教师信号，一个样本是一列

    %NN信号前传导过程
    net = cnnff (net, batch_x);
    %计算误差并反向传导，计算梯度
    net = cnnbp (net, batch_y);
    %应用梯度，模型更新
    net = cnnapplygrads(net, opts);
    %net.L为模型的costFunction,即最小均方误差mse
    %net.rL是最小均方误差的平滑序列
    if isempty(net.rL)
        net.rL(1) = net.L;
    end
    net.rL(end+1) = 0.99 * net.rL(end) + 0.01 * net.L;
end
end

```

cnnff(net, x): 前向传播

```
function net = cnnff(net, x)
```

```
    n = numel(net.layers); % 网络层数 net.layer=n=5
```

```
    net.layers{1}.a{1} = x;
```

```
    inputmaps = 1;
```

```
    for l = 2 : n % 除输入层以外的每一层到第5层
```

```
        if strcmp(net.layers{l}.type, 'c') % 卷积层
```

```
            % !!below can probably be handled by insane matrix operations
```

```
            for j = 1 : net.layers{l}.outputmaps % 该层的每一个特征图第二层为6, 第四层为12
```

```
                % 该层的输出: 上一层图片大小-kernel+1,
```

```
                % size(前层特征图)-size(Kernelsize, 5*5)+1=size(本层特征图) , z = zeros([28,28,10]-[4,4,0]) = zeros([24,24,10])
```

```
                z = zeros(size(net.layers{l-1}.a{1}) - [net.layers{l}.kernelsize - 1 net.layers{l}.kernelsize - 1 0]);
```

```
                for i = 1 : inputmaps % 对于每一个输入特征图 (本例中为全连接)
```

```
                    % 每个特征图的卷积都相加, convn()为matlab自带卷积函数
```

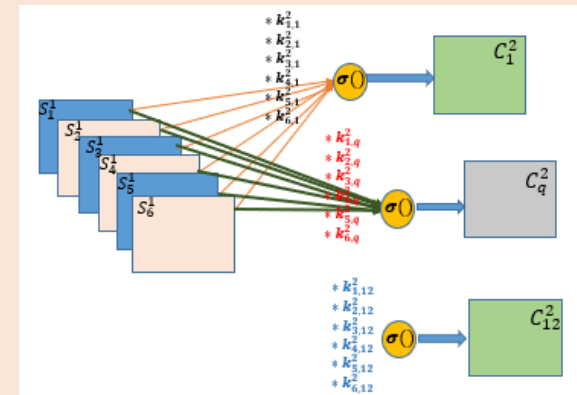
```
                    z = z + convn(net.layers{l-1}.a{i}, net.layers{l}.k{i}{j}, 'valid');
```

```
                end
```

```
            % 加入偏置项, 并通过sigmoid函数 (现今一般采用RELU) 6个a 24*24*50的卷积特征
```

```
            net.layers{l}.a{j} = sigm(z + net.layers{l}.b{j});
```

$$C_p^1 = \sigma(I * k_{1,p}^1 + b_p^1)$$



% 用该层的outputmaps更新inputmaps的值并为下一层所用，第一次输出为6个卷积特征z

```
inputmaps = net.layers{l}.outputmaps;
```

```
elseif strcmp(net.layers{l}.type, 's')
```

% mean-pooling, 平均池化

```
for j = 1 : inputmaps
```

%%这子采样是怎么做到的？卷积一个 $[0.25, 0, 25; 0.25, 0, 5]$ 的卷积核，然后降采样。这里有重复计算

```
z = convn(net.layers{l - 1}.a{j}, ones(net.layers{l}.scale) / (net.layers{l}.scale ^ 2), 'valid'); % !! replace with variable
```

% 取出有效的mean-pooling矩阵，即每隔net.layers{l}.scale提取一个值，

$\text{ones}(2)/4$	=	0.2500	0.2500
		0.2500	0.2500

%这里设置scale长度的步长,窗移一》scale，但是这里有计算浪费

```
net.layers{l}.a{j} = z(1 : net.layers{l}.scale : end, 1 : net.layers{l}.scale : end, :); %每隔scale, 降采样
```

```
end
```

```
end
```

```
end
```

% 把最后一层展开变成一行向量方便操作

```
net.fv = [];
```

```
for j = 1 : numel(net.layers{n}.a) %numel(net.layers{n}.a)=12
```

```
    %fv每次拼合入subFeaturemap2[j],[包含50个样本]
```

```
    sa = size(net.layers{n}.a{j}); % size(sa) = [4, 4, 10];
```

```
    %reshape(A,m,n)%把矩阵A改变形状，编程m行n列元素个数不变，原矩阵按列排成一队，再按行排成若干队)
```

```
    %把所有的向量拼合成为一个列向量fv， [net.fv; reshape(net.layers{numLayers}.a{j}, 4*4, 10)];
```

```
    % net.fv相当于教程中的a，其中a=f(z)，size(net.fv)=[192 10] = [4*4*12 10]
```

```
    net.fv = [net.fv; reshape(net.layers{n}.a{j}, sa(1) * sa(2), sa(3))];
```

```
end
```

```
% 加上权值和偏置项并通入sigmoid(多类别神经网络的输出一般采用softmax形式，损失函数使用cross entropy )
```

```
net.o = sigm(net.ffW * net.fv + repmat(net.ffb, 1, size(net.fv, 2)));
```

%ffW为10*192 在cnnsetup里面进行的随机初始化，net.ffW [10 192] net.fv为：[192 10]的矩阵

End

```
%注释：net.ffb = zeros(onum, 1);% 最后一层bias的设置,
```

```
onum=10, fvnum =4*4*12=192
```

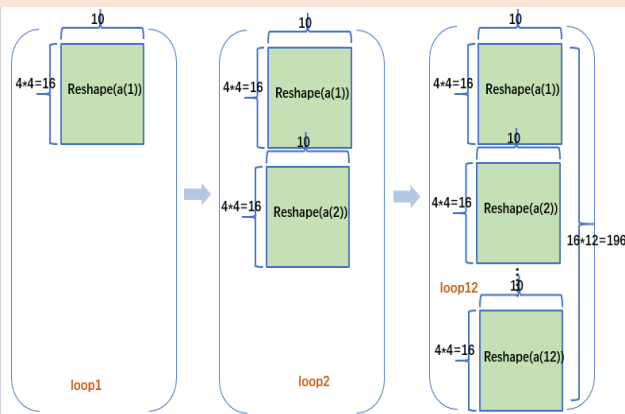
```
%注释： net.ffW = (rand(onum, fvnum) - 0.5) * 2 * sqrt(6 /  
(onum + fvnum)); %初始化权重
```

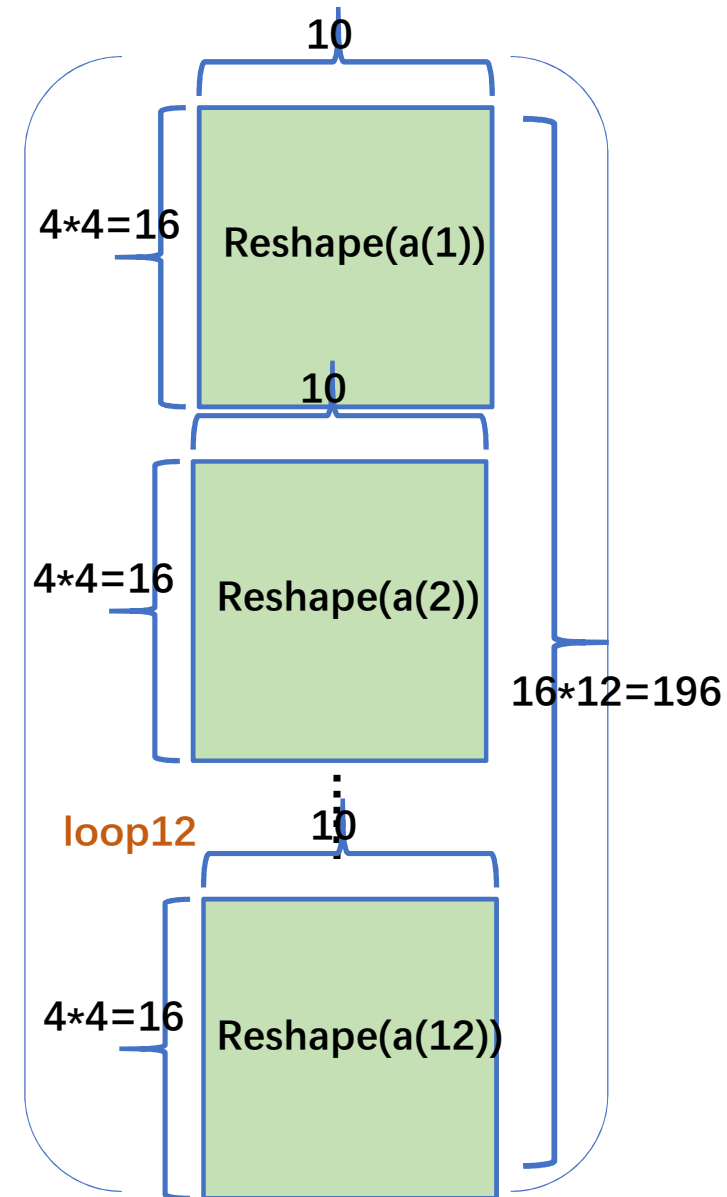
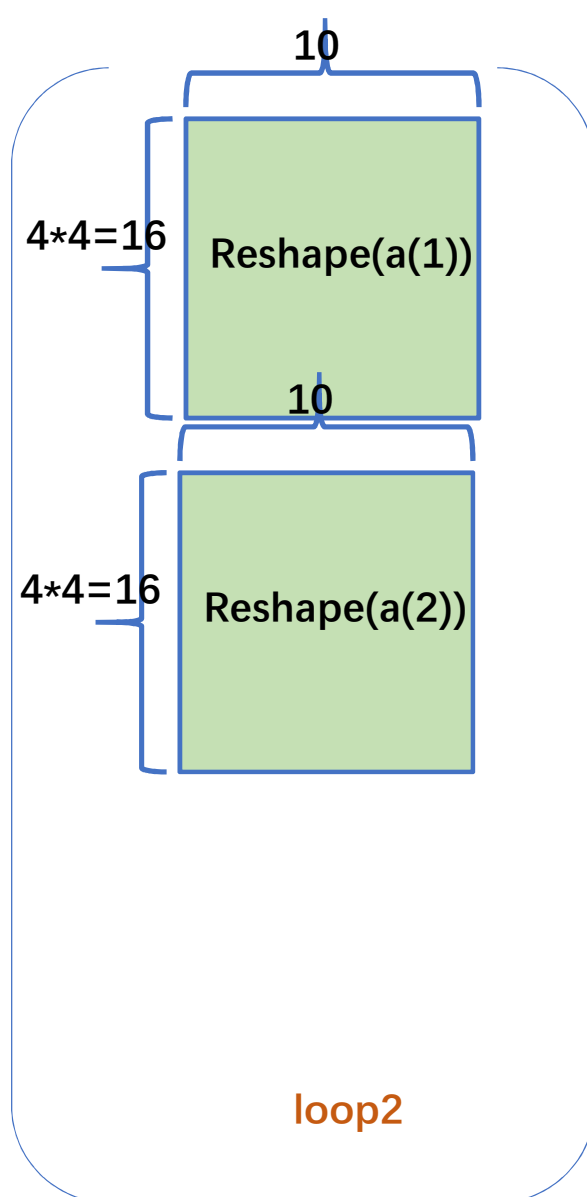
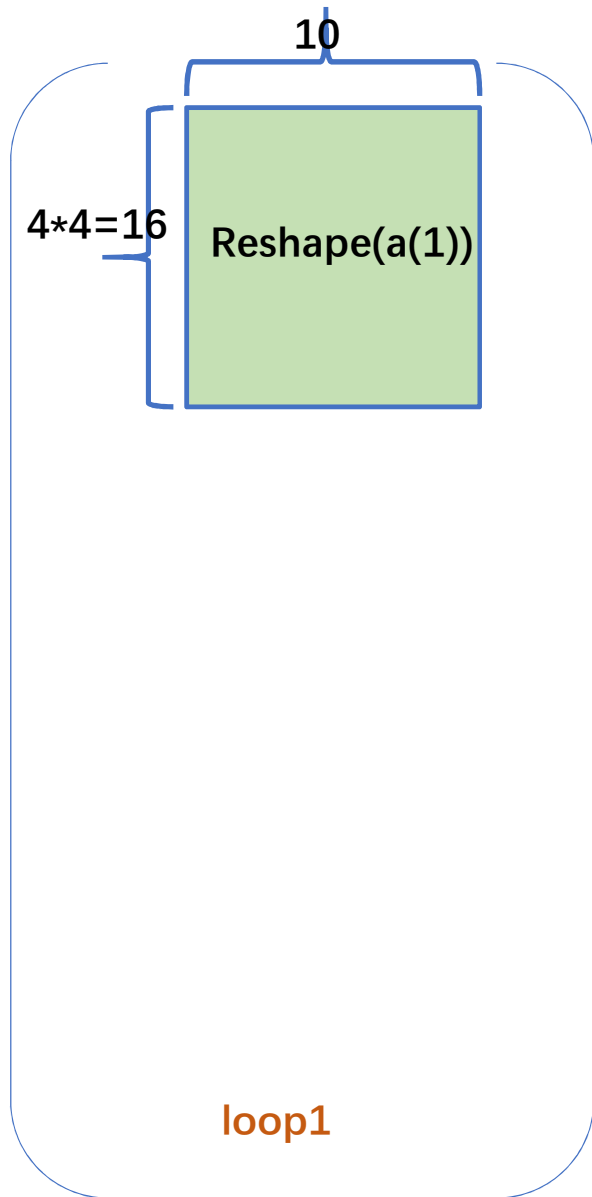
```
A = (1:3)';  
B = repmat(A,1,4)
```

B =			
1	1	1	1
2	2	2	2
3	3	3	3

尾部单层感知机的数据处理

需要把subFeatureMap2连接成为一个(4*4)*12=192的向量，但是由于采用了50样本批训练的方法，subFeatureMap2被拼合成为一个192*10的特征向量fv；Fv作为单层感知机的输入，全连接的方式得到输出层





```
function net = cnnbp(net, y)
```

```
    n = numel(net.layers); %n=5
```

```
    % error
```

```
    net.e = net.o - y;  $(\hat{y}(i) - y(i))$ 
```

```
    % loss function
```

```
    net.L = 1/2* sum(net.e(:) .^ 2) / size(net.e, 2); %均方误差
```

```
    %% backprop deltas
```

```
    net.od = net.e .* (net.o .* (1 - net.o)); % output delta
```

```
    net.fvd = (net.ffW' * net.od); % feature vector delta
```

```
    if strcmp(net.layers{n}.type, 'c') % only conv layers has sigm function n=5
```

```
        net.fvd = net.fvd .* (net.fv .* (1 - net.fv));
```

```
end
```

$$\Delta W(i, j) = \frac{\partial L}{\partial W(i, j)} = (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i)) \cdot f(j)$$

Let $\Delta \hat{y}(i) = (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i))$, whose size is 10×1 , then

$$\begin{aligned} \Delta W(i, j) &= \Delta \hat{y}(i) \cdot f(j) \\ \Rightarrow \Delta W &= \Delta \hat{y} \times f^T \end{aligned}$$

$$L = \frac{1}{2} \sum_{i=1}^{10} (\hat{y}(i) - y(i))^2$$

$$\Delta \hat{\mathbf{y}}_{10 \times 1} \triangleq (\hat{\mathbf{y}} - \mathbf{y}) \hat{\mathbf{y}}(1 - \hat{\mathbf{y}})$$

$$\Delta \hat{y}(i) = (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i))$$

$$\Delta f(j) = \frac{\partial L}{\partial f} = \sum_{i=1}^{10} \Delta \hat{y}(i) \cdot W(i, j)$$

$$\Rightarrow \Delta f = W^T \times \Delta \hat{y}$$

注: $\text{size}(\text{net.fvd}) = [192, 10]$, $\text{size}(\text{net.od}) = [10, 10]$, $\text{size}(\text{net.ffW}) = [10, 192]$
 $(\text{net.ffW}')_{192 \times 10} * \text{net.od}_{10 \times 10} = \text{net.fvd}_{192 \times 10}$

% reshape feature vector deltas into output map style

sa = size(net.layers{n}.a{1}); % size(net.layers{5}.a{1})=sa=[4,4,10]

fvnum = sa(1) * sa(2); %fvnum=4*4=16

for j = 1 : numel(net.layers{n}.a) % numel(net.layers{n}.a)=1*12=12

net.layers{n}.d{j} = reshape(net.fvd(((j - 1) * fvnum + 1) : j * fvnum, :), sa(1), sa(2), sa(3));

%reshape(net.fvd(1 : 16, :), 4, 4, 10); reshape(net.fvd(17 : 32, :), 4, 4, 10),...

%reshape(net.fvd(176 : 192, :), 4, 4, 10);

end

for l = (n - 1) : -1 : 1 %n=4,3,2,1

if strcmp(net.layers{l}.type, 'c')

for j = 1 : numel(net.layers{l}.a) % (n=4-> j=1,2,...,12); % (n=2-> j=1,2,...,6);

net.layers{l}.d{j} = net.layers{l}.a{j} .* (1 - net.layers{l}.a{j}) .*

(expand(net.layers{l + 1}.d{j}, [net.layers{l + 1}.scale net.layers{l + 1}.scale 1]) / net.layers{l + 1}.scale ^ 2);

end

$$\Delta b_{10 \times 1} \triangleq \Delta \hat{y}_{10 \times 1}$$

$$\Delta b(i) = \frac{\partial L}{\partial b(i)} = (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i))$$

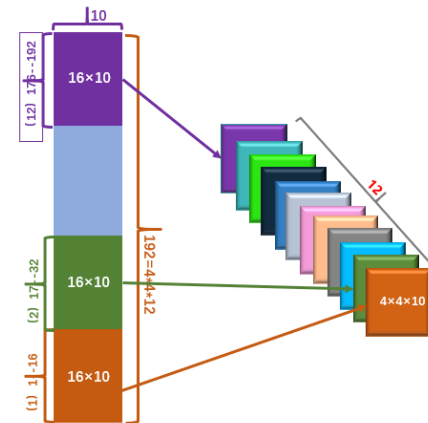
$$\text{expand}([1,2;3,4],[2 \ 2]) = \begin{pmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{pmatrix} \times \frac{1}{4}$$

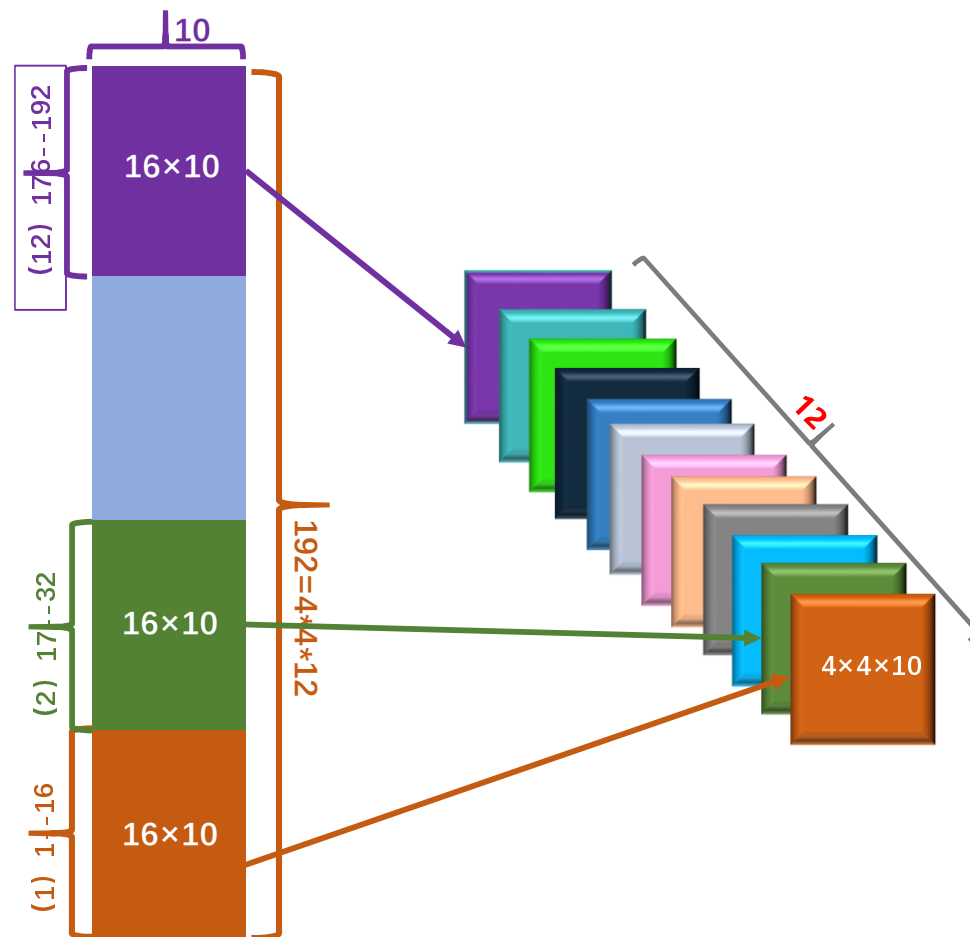
$$\Delta C_q^2(i, j)_{q=1, \dots, 12} \leftarrow \frac{1}{4} \Delta S_q^2([i], [j])_{i, j=1, \dots, 8}$$

$$\Delta C_{q\downarrow}^2(i, j) \leftarrow \Delta C_q^2(i, j) C_q^2(i, j) [1 - C_q^2(i, j)], i, j = 1, \dots, 12$$

Layer 4->12 $\uparrow \Delta C_{q\downarrow}^2(i, j)$

Layer 2->6 $\uparrow \Delta C_p^1(i, j)$





```

elseif strcmp(net.layers{l}.type, 's')%下采样层

    for i = 1 : numel(net.layers{l}.a)%i=1,2,3

        z = zeros(size(net.layers{l}.a{1})); % size(net.layers{3}.a{1})=12*12*10;

        for j = 1 : numel(net.layers{l + 1}.a)% numel(layers{4}.a)=12;

            z = z + convn(net.layers{l + 1}.d{j}, rot180(net.layers{l + 1}.k{i}{j}), 'full');

        end

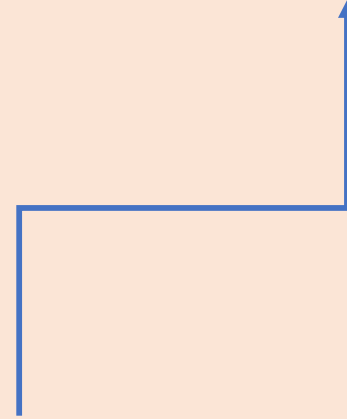
        net.layers{l}.d{i} = z;

    end

end

end

```



$$\begin{aligned}
 \Delta S_p^1(i, j)_{i, j=1, \dots, 12} &= \frac{\partial L}{\partial S_p^1(i, j)} \Leftarrow \sum_{q=1}^{12} \sum_{u=-2}^2 \sum_{v=-2}^2 \Delta C_{q\downarrow}^2(i+u, j+v) k_{pq}^2(u, v) = \sum_{q=1}^{12} \sum_{u=-2}^2 \sum_{v=-2}^2 \Delta C_{q\downarrow}^2(i+u, j+v) \text{Rot}(k_{pq}^2)(-u, -v) \\
 &= \sum_{q=1}^{12} \sum_{u=-2}^2 \sum_{v=-2}^2 \Delta C_{q\downarrow}^2(i - (-u), j - (-v)) \text{Rot}(k_{pq}^2)(-u, -v) = \sum_{q=1}^{12} \Delta C_{q\downarrow}^2 * \text{Rot}(k_{pq}^2)(i, j)
 \end{aligned}$$

```

%%  calc gradients
for l = 2 : n
    if strcmp(net.layers{l}.type, 'c')
        for j = 1 : numel(net.layers{l}.a)
            for i = 1 : numel(net.layers{l - 1}.a)
                fuck = flipall(net.layers{l - 1}.a{i});
                you = conv2(fuck(:, :, 1), net.layers{l}.d{j}(:, :, 1), 'valid')
/ size(net.layers{l}.d{j}, 3);
                dick = sum(sum(net.layers{l}.d{j}(:, :, 1)));
                net.layers{l}.dk{i}{j} = convn(flipall(net.layers{l -
1}.a{i}), net.layers{l}.d{j}, 'valid') / size(net.layers{l}.d{j}, 3);
            end
            net.layers{l}.db{j} = sum(net.layers{l}.d{j}(:)) /
size(net.layers{l}.d{j}, 3);
        end
    end
end
net.dffW = net.od * (net.fv)' / size(net.od, 2);
net.dffb = mean(net.od, 2);

```

```
function X = rot180(X)
```

```
    X = flipdim(flipdim(X, 1), 2);
```

```
end
```

```
end
```

$$\Delta \hat{\mathbf{y}}_{10 \ast 1} \triangleq (\hat{\mathbf{y}} - \mathbf{y}) \odot \hat{\mathbf{y}} \odot (1 - \hat{\mathbf{y}})$$

$$\Delta \mathbf{b}_{10 \ast 1} \triangleq \Delta \hat{\mathbf{y}}_{10 \ast 1}$$

$$\Delta \mathbf{W}_{10 \ast 192} \triangleq \Delta \hat{\mathbf{y}}_{10 \ast 1} (\mathbf{f}^T)_{1 \ast 192}$$

$$\Delta \mathbf{f}_{192 \ast 1} \triangleq (\mathbf{W}^T)_{192 \ast 10} \hat{\mathbf{y}}_{10 \ast 1}$$

$$\{\Delta S_q^2(4 \times 4)\}_{q=1,\dots,12} \leftarrow F^{-1}(\Delta \mathbf{f}_{192 \ast 1})$$

$\Delta C_q^2(1,1); \Delta C_q^2(1,2)$	$= \frac{1}{4} \Delta S_q^2(1,1)$...	$\Delta C_q^2(1,7); \Delta C_q^2(1,8)$	$= \frac{1}{4} \Delta S_q^2(1,4)$
$\Delta C_q^2(2,1); \Delta C_q^2(2,2)$			$\Delta C_q^2(2,7); \Delta C_q^2(2,8)$	
\vdots			\vdots	
$\Delta C_q^2(7,1); \Delta C_q^2(7,2)$	$= \frac{1}{4} \Delta S_q^2(4,1)$...	$\Delta C_q^2(7,7); \Delta C_q^2(7,8)$	$= \frac{1}{4} \Delta S_q^2(4,4)$
$\Delta C_q^2(8,1); \Delta C_q^2(8,2)$			$\Delta C_q^2(8,7); \Delta C_q^2(8,8)$	

$$(i,j) \cdot S_p^1(-(u-i), -(v-j))$$

$$\Delta C_q^2(i,j)_{q=1,\dots,12} \leftarrow \frac{1}{4} \Delta S_q^2([i],[j])_{i,j=1,\dots,8}$$

$$\frac{\partial L}{\partial (u,v)} \Leftarrow \sum_{i=1}^8 \sum_{j=1}^8 \Delta C_{q\sigma}^2(i,j) S_p^1(i-u,j-v) = \sum_{i=1}^8 \sum_{j=1}^8 \Delta C_{q\sigma}^2(i,j) \text{Rot}(S_p^1)(u-i,v-j) = \Delta \mathbf{C}_{q\sigma}^2 * \text{Rot}(S_p^1)(u,v)$$

$$S_p^1(i-u,j-v) = \sum_{i=1}^8 \sum_{j=1}^8 \Delta C_{q\sigma}^2(i,j) \text{Rot}(S_p^1)(u-i,v-j)$$

$$= \Delta \mathbf{C}_{q\sigma}^2 * \text{Rot}(S_p^1)(u,v)$$

$$u,v) = \frac{\partial L}{\partial k_{p,q}^2} \Delta k_{p,q}^2(u,v) = \sum_{i=1}^8 \sum_{j=1}^8 S_{p,rot180}^1(u-i,v-j) \cdot \Delta C_{q,\sigma}^2(i,j)$$

$$\implies \Delta k_{p,q}^2 = S_{p,rot180}^1 * \Delta C_{q,\sigma}^2$$

$$i)) \cdot \hat{y}(i)(1 - \hat{y}(i)) \cdot f(j)$$

$$\Delta \hat{y}(i) = (\hat{y}(i) - y(i)) \cdot \hat{y}(i)(1 - \hat{y}(i))$$

$$\begin{aligned} \Delta k_{pq}^2(u, v)_{u, v=1, \dots, 5} &= \frac{\partial L}{\partial k_{pq}^2(u, v)} \Leftarrow \sum_{i=1}^8 \sum_{j=1}^8 \Delta C_{q\sigma}^2(i, j) S_p^1(i - u, j - v) \\ &= \sum_{i=1}^8 \sum_{j=1}^8 \Delta C_{q\sigma}^2(i, j) \text{Rot}(S_p^1)(u - i, v - j) = \Delta C_{q\sigma}^2 * \text{Rot}(S_p^1)(u, v) \end{aligned}$$

$$\begin{array}{ccc}
\begin{array}{c} \Delta C_q^2(1,1); \Delta C_q^2(1,2) \\ \Delta C_q^2(2,1); \Delta C_q^2(2,2) \\ \vdots \end{array} & = \frac{1}{4} \Delta S_q^2(1,1) \quad \cdots & \begin{array}{c} \Delta C_q^2(1,7); \Delta C_q^2(1,8) \\ \Delta C_q^2(2,7); \Delta C_q^2(2,8) \\ \vdots \end{array} = \frac{1}{4} \Delta S_q^2(1,4) \\
\begin{array}{c} \Delta C_q^2(7,1); \Delta C_q^2(7,2) \\ \Delta C_q^2(8,1); \Delta C_q^2(8,2) \end{array} & = \frac{1}{4} \Delta S_q^2(4,1) \quad \cdots & \begin{array}{c} \Delta C_q^2(7,7); \Delta C_q^2(7,8) \\ \Delta C_q^2(8,7); \Delta C_q^2(8,8) \end{array} = \frac{1}{4} \Delta S_q^2(4,4)
\end{array}$$

$$\begin{array}{ccccccc}
\begin{array}{c} \Delta C_q^2(3,1); \Delta C_q^2(3,2) \\ \Delta C_q^2(4,1); \Delta C_q^2(4,2) \end{array} & = \frac{1}{4} \Delta S_q^2(2,1) & \begin{array}{c} \Delta C_q^2(1,3); \Delta C_q^2(1,4) \\ \Delta C_q^2(2,3); \Delta C_q^2(2,4) \end{array} & = \frac{1}{4} \Delta S_q^2(1,2) & \begin{array}{c} \Delta C_q^2(1,5); \Delta C_q^2(1,6) \\ \Delta C_q^2(2,5); \Delta C_q^2(2,6) \end{array} & = \frac{1}{4} \Delta S_q^2(1,3) & \begin{array}{c} \Delta C_q^2(3,7); \Delta C_q^2(3,8) \\ \Delta C_q^2(4,7); \Delta C_q^2(4,8) \end{array} & = \frac{1}{4} \Delta S_q^2(2,4) \\
\begin{array}{c} \Delta C_q^2(5,1); \Delta C_q^2(5,2) \\ \Delta C_q^2(6,1); \Delta C_q^2(6,2) \end{array} & = \frac{1}{4} \Delta S_q^2(3,1) & \begin{array}{c} \Delta C_q^2(3,3); \Delta C_q^2(3,4) \\ \Delta C_q^2(4,3); \Delta C_q^2(4,4) \end{array} & = \frac{1}{4} \Delta S_q^2(2,2) & \begin{array}{c} \Delta C_q^2(3,5); \Delta C_q^2(1,6) \\ \Delta C_q^2(4,5); \Delta C_q^2(2,6) \end{array} & = \frac{1}{4} \Delta S_q^2(2,3) & \begin{array}{c} \Delta C_q^2(5,7); \Delta C_q^2(5,8) \\ \Delta C_q^2(6,7); \Delta C_q^2(6,8) \end{array} & = \frac{1}{4} \Delta S_q^2(3,4) \\
\begin{array}{c} \Delta C_q^2(5,3); \Delta C_q^2(5,4) \\ \Delta C_q^2(6,3); \Delta C_q^2(6,4) \end{array} & = \frac{1}{4} \Delta S_q^2(3,2) & \begin{array}{c} \Delta C_q^2(7,3); \Delta C_q^2(7,4) \\ \Delta C_q^2(8,3); \Delta C_q^2(8,4) \end{array} & = \frac{1}{4} \Delta S_q^2(4,2) & \begin{array}{c} \Delta C_q^2(5,5); \Delta C_q^2(5,6) \\ \Delta C_q^2(6,5); \Delta C_q^2(6,6) \end{array} & = \frac{1}{4} \Delta S_q^2(3,3) & & \\
& & & & \begin{array}{c} \Delta C_q^2(7,5); \Delta C_q^2(7,6) \\ \Delta C_q^2(8,5); \Delta C_q^2(8,6) \end{array} & = \frac{1}{4} \Delta S_q^2(4,3) & &
\end{array}$$