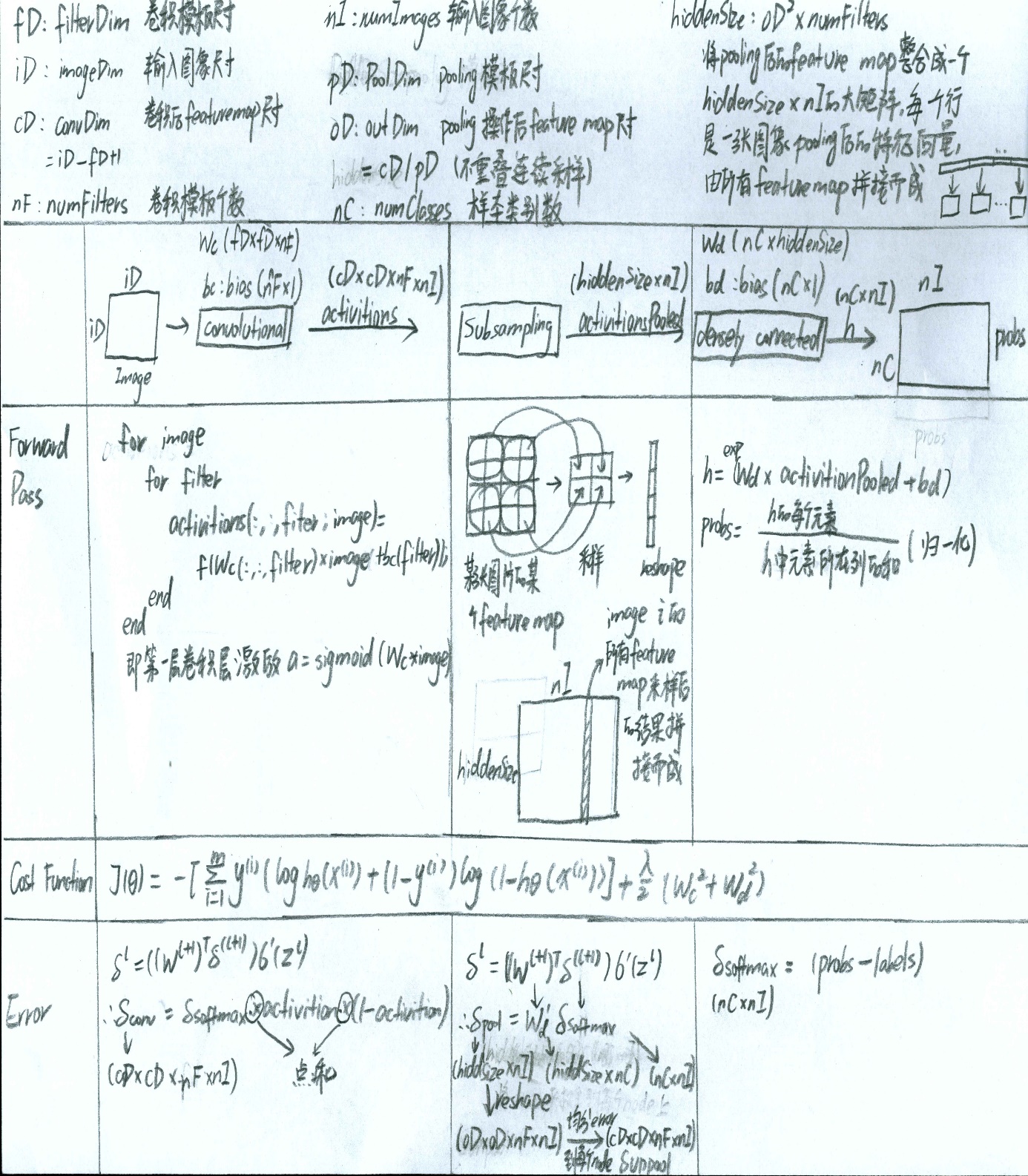
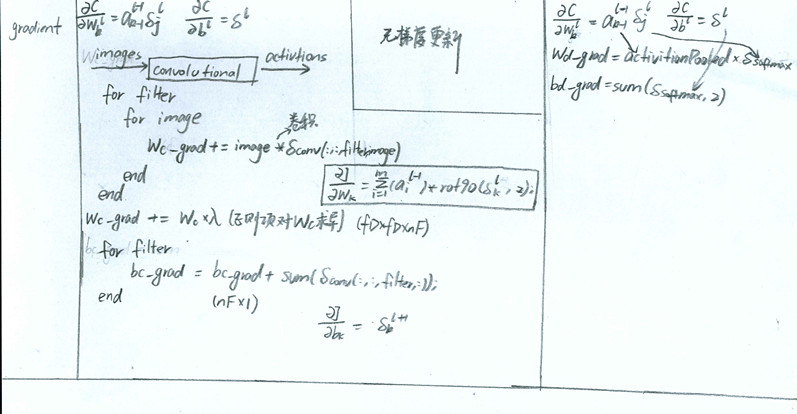
这个exercise需要完成cnn中的forward pass,cost,error和gradient的计算。需要弄清楚每一层的以上四个步骤的原理，并且要充分利用matlab的矩阵运算。大概把过程总结了一下如下图所示：





**STEP 1：Implement CNN Objective**

STEP 1a: Forward Propagation

Forward Propagation主要是为了计算输入图片经过神经网络后的输出，这个网络有三层：convolution->pooling->softmax(dense connected)，卷积层对于每个图像用所有的模板进行卷积；pooling层对卷积层输出的feature map采样；softmax层根据pooling层输出的feature预测图像的分类结果。其中convolution和pooling操作在[之前](http://www.cnblogs.com/sunshineatnoon/p/4527475.html" \t "_blank)就实现过了。具体的过程可以参见上图中Forward Pass中每层的具体操作。代码如下：

复制代码

%%% YOUR CODE HERE %%%

%调用之前已经实现的函数

activations = cnnConvolve(filterDim, numFilters, images, Wc, bc);%sigmoid(wx+b)

activationsPooled = cnnPool(poolDim, activations);

% Reshape activations into 2-d matrix, hiddenSize x numImages,

% for Softmax layer

%将activationsPooled从outDim\*outDim\*numFilters\*numImages拼接成hiddenSize\*numImages的大矩阵

activationsPooled = reshape(activationsPooled,[],numImages);

%% Softmax Layer

% Forward propagate the pooled activations calculated above into a

% standard softmax layer. For your convenience we have reshaped

% activationPooled into a hiddenSize x numImages matrix. Store the

% results in probs.

% numClasses x numImages for storing probability that each image belongs to

% each class.

probs = zeros(numClasses,numImages);

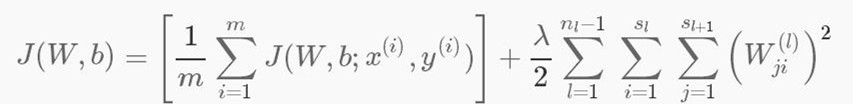
%%% YOUR CODE HERE %%%

h = exp(bsxfun(@plus,Wd \* activationsPooled,bd));

probs = bsxfun(@rdivide,h,sum(h,1));

STEP 1b: Calculate Cost

计算梯度下降要优化的目标函数，主要分为两部分，一部分是由于分类器输出结果和真实结果的差异引起的误差函数，另一部分是对权重w的正则约束。第一部分可以参考[softmax regression](http://www.cnblogs.com/sunshineatnoon/p/4524365.html" \t "_blank)中对损失函数的计算，第二部分就是对Wc和Wd的所有项求平方和。类似下面的公式，不过第一项中的J是softmax的cross entropy损失函数。最后要对第一项除以图像的总数，这是十分重要的，一开始我没有除，最后得到的算法是发散的，原因可能是第一项数值过大，直接把正则项的影响给忽略了。



代码：

%%% YOUR CODE HERE %%%

logp = log(probs);

index = sub2ind(size(logp),labels',1:size(probs,2));

ceCost = -sum(logp(index));

wCost = lambda/2 \* (sum(Wd(:).^2)+sum(Wc(:).^2));

cost = ceCost/numImages + wCost;

STEP 1c: Backpropagation

BP算法首先要计算各层的对最终误差的贡献delta。

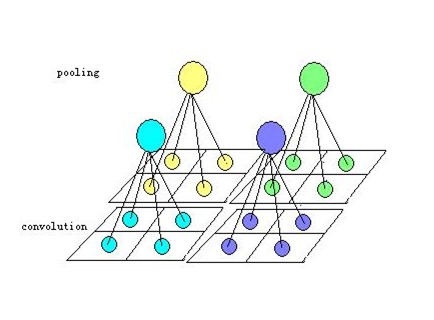
softmax层：这一层的误差最容易计算，只要用ground truth减去神经网络的输出probs就可以了：

output = zeros(size(probs));

output(index) = 1;

DeltaSoftmax = probs - output;

pool层：这一层首先根据公式δl＝ Wδl+1\* f'(zl)（pool层没有f'(zl)这一项）计算该层的error，此时得到一个hiddenSize\*numImages的矩阵，首先利用reshape函数把error还原成一个convDim\*convDim\*numFilters\*numImages的矩阵，在pooling操作时，pooling层一个节点的输入是conv层2\*2个节点的输出（假设poolDim=2）如下图所示：



所以pooling层的这个节点要将自己的error在这2\*2个节点中平均分（因为使用的是mean pooling）,[UFLDL上面提示了可以用kron这个函数来实现](http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionalNeuralNetwork/)，这样如上图所示，就可以通过pooling层一个2\*2的filter对应的error计算得到convolution层一个4\*4的filter对应的error了。代码如下：

DeltaPool = reshape(Wd' \* DeltaSoftmax,outputDim,outputDim,numFilters,numImages);

DeltaUnpool = zeros(convDim,convDim,numFilters,numImages);

for imNum = 1:numImages

for FilterNum = 1:numFilters

unpool = DeltaPool(:,:,FilterNum,imNum);

DeltaUnpool(:,:,FilterNum,imNum) = kron(unpool,ones(poolDim))./(poolDim ^ 2);

end

end

convolution层：还是根据公式δl＝ Wδl+1\* f'(zl)来计算：

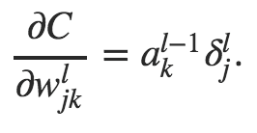
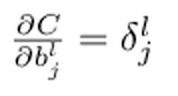
DeltaConv = DeltaUnpool .\* activations .\* (1 - activations);

STEP 1d: Gradient Calculation

整个cnn一共有三层：convolution->pooling->softmax(dense connected)，只有convolution和softmax层有权重，分别为Wc，bc，Wd，bd。那么就要计算目标函数J对他们的导数以便在梯度下降中更新W和b。

Wd和bd的梯度计算：

根据下面两个公式：

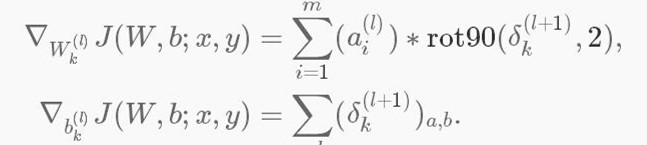
其中al-1对应pooling层的激励（输出）activitonsPooled，δl就是这一层的误差DeltaSoftmax，代码如下：

Wd\_grad = (1./numImages) .\* DeltaSoftmax\*activationsPooled'+lambda\*Wd;

bd\_grad = (1./numImages) .\* sum(DeltaSoftmax,2);

Wc和bc的梯度计算：

还是根据上面两个计算梯度的公式，不过麻烦就麻烦在l-1层其实是输入的图像，所以al-1对应的是输入的图像，那么就得用for循环逐个遍历图像并利用UFLDL上提供的公式计算对应梯度：



这里为了方便，先对所有DeltaConv进行旋转，然后再用for循环依次求出梯度：

%%% YOUR CODE HERE %%%

Wd\_grad = (1./numImages) .\* DeltaSoftmax\*activationsPooled'+lambda\*Wd;

bd\_grad = (1./numImages) .\* sum(DeltaSoftmax,2);

bc\_grad = zeros(size(bc));

Wc\_grad = zeros(filterDim,filterDim,numFilters);

for filterNum = 1:numFilters

error = DeltaConv(:,:,filterNum,:);

bc\_grad(filterNum) = (1./numImages) .\* sum(error(:));

end

%旋转所有DealtaConv

for filterNum = 1:numFilters

for imNum = 1:numImages

error = DeltaConv(:,:,filterNum,imNum);

DeltaConv(:,:,filterNum,imNum) = rot90(error,2);

end

end

for filterNum = 1:numFilters

for imNum = 1:numImages

Wc\_grad(:,:,filterNum) = Wc\_grad(:,:,filterNum) + conv2(images(:,:,imNum),DeltaConv(:,:,filterNum,imNum),'valid');

end

end

Wc\_grad = (1./numImages) .\* Wc\_grad + lambda\*Wc;

**Step 2: Gradient Check**

当时明明我的梯度下降没法收敛，这一步居然通过了=。=

**Step 3: Learn Parameters**

这步比较简单，根据[UFLDL对随机梯度下降的解释](http://ufldl.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/" \t "_blank)，在minFuncSGD中加上冲量的影响就可以了：

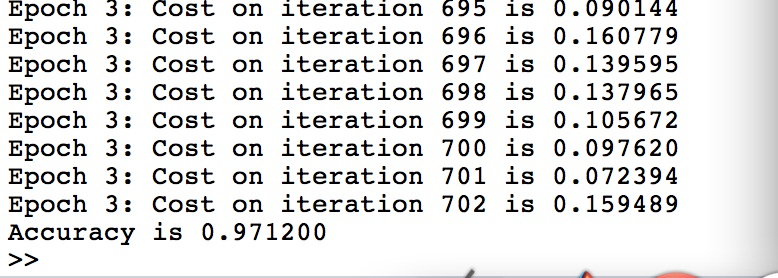
%%% YOUR CODE HERE %%%

velocity = mom\*velocity+alpha\*grad;

theta = theta - velocity;

**Step 4: Test**

运行cnnTrain，最后准确率可以达到97%+



以上就可UFLDL上cnn的实现，最重要的是弄清楚每一层在每一个过程中需要进行的操作，我都总结在文章开头的表格里面了~matlab给我一个很大的感受就是矩阵的demension match，有时候知道公式是什么样的，但是实现起来要考虑矩阵的维度，两个维度match的矩阵才能相乘或者相加，不过好处就是再不知道怎么写代码的时候可以结果维度match来写代码。而且cnn debug起来真的好困难，完全不知道是哪里出了问题=。=

参考：

【1】<http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionalNeuralNetwork/>

【2】<http://blog.csdn.net/lingerlanlan/article/details/41390443>