利用卷积神经网络对MINST分类

本文是参照UFDLF中的教程写的。

一. cost function

一般有两种cost function: **均方误差**(MSE)和**交叉熵**(cross entropy)，本文实现的卷积神经网络采用**交叉熵**。

均方误差

$$C(W,B)=\frac{1}{2m}\sum_{x}\|y(x)-a\|^2 \ (1)$$

其中W为网络中的权值，B为神经元的偏置，m为样本数，x为样本，$y(x)$为样本对应的标签，a为样本对应的输出。

交叉熵

$$C(W,B)=-\sum_{x}logP(y(x)=k)(2)$$
$$k=0,1,\dots,C-1$$

其中C为目标类的个数。本文的卷积神经网络采用的cost function是交叉熵。

二. 结构

包含3层(不包括输入层)：

- 第一层：卷积层(convolutional layer)。该层直接与输入图像相连，我用的数据集是MNIST，所以输入图像大小为28x28，用来卷积的filter size为9x9，步长为1，共20个filter，因此卷积层的大小为20(height 28 - 9 + 1) x 20(width) x 20(feature map)。
- 第二层：池化层(pooling layer)。该层的输入为卷积层的输出，我用的是average pooling，而不是max pooling。采用的filter大小为2x2，每次subsample不重叠，因此步长为2。池化层可以大幅减少数据规模，减少参数的数量并减少计算量。一个2x2的filter可以减少上一层75%的输入。池化层的大小为20个10x10。
- 第三层：输出层。该层与池化层之间为全连接，共10个神经元对应10个类，输出为该类的概率。这两层相连的作用实际上就是一次softmax分类。在一般的softmax分类中，输入为整个图像，而在这里的输入为池化层的输出。

三. error backpropagation

feedforward过程没什么说的，接下来重点说error backpropagation过程。

1. 输出层的δ
   根据定义,第l层的δ为

$$\delta^L=\nabla_{z^L}C \qquad (3)$$

   其中z^L为最后一层的输入，C为cost function。由(3)可知，δ与每个神经元一一对应。接下来进行推导，为了简化，只考虑一个样本,且该样本属于类k。

$$
\begin{aligned}
\delta_i^L &= \frac{\partial C}{\partial z_i^L} \\
&= \frac{\partial}{\partial z_i^L}\left[-\log(P(y(x)=k))\right] \\
&= -\frac{1}{P(y(x)=k)} \cdot \frac{\partial}{\partial z_i^L}\left[P(y(x)=k)\right] \\
&= -\frac{1}{P(y(x)=k)} \cdot \frac{\partial}{\partial z_i^L}\left(\frac{e^{z_k^L}}{\sum_n e^{z_n^L}}\right) \\
&= -\frac{1}{P(y(x)=k)} \cdot \left[\frac{\partial e^{z_k^L}}{\partial z_i^L} \cdot \frac{1}{\sum_n e^{z_n^L}} - \frac{e^{z_k^L}}{\sum_n e^{z_n^L}} \cdot \frac{e^{z_i^L}}{\sum_n e^{z_n^L}}\right] \\
&= -\frac{1}{P(y(x)=k)}\left[1\{k=i\} \cdot P(y(x)=k) - P(y(x)=k) \cdot P(y(x)=i)\right] \\
&= -(1\{k=i\} - P(y(x)=i))
\end{aligned}
$$

其中，n为该层神经元的个数，这里为10。写成向量形式为

$$\delta = -(e(k) - y(x)) \qquad (4)$$

其中$e(k)$为一个C维列向量，C为类的个数，只有第k个元素 $(k = 0, 1, \ldots, C-1)$ 为1，其余为0，$y(x)$为网络的输出。

2. 池化层的delta

这一步，δ要从最后一层传播到池化层,这两层之间是全连接的。下面推导两个全连接层之间的误差传播公式：

$$
\begin{aligned}
\delta_i^l &= \frac{\partial C}{\partial z_i^l} \\
&= \sum_n \frac{\partial C}{\partial z_n^{l+1}} \cdot \frac{\partial z_n^{l+1}}{\partial z_i^l} \qquad (5) \\
&= \sum_n \delta_n^{l+1} \cdot \frac{\partial\left(\sum_j w_{nj}^{l+1} a_j^l + b_n\right)}{\partial z_i^l} \\
&= \sum_n \delta_n^{l+1} \cdot w_{ni}^{l+1} \cdot \frac{\partial a_i^l}{\partial z_i^l} \\
&= \sum_n \delta_n^{l+1} \cdot w_{ni}^{l+1} \cdot \frac{\partial f(z_i^l)}{\partial z_i^l}
\end{aligned}
$$

其中w_{ij}^l为第l-1层的第j个神经元，与第l层的第i个神经元之间的权值，f(x)为激活函数，本实验中池化层的激活函数为f(x) = x。写成向量形式为

$$\delta^l = \left(W^{l+1}\right)^T \delta^{l+1} \bigodot f'\left(z^l\right) \qquad (6)$$

其中，符号$\bigodot$为对应元素分别相乘。

3. 卷积层的δ

以2x2的池化filter为例，每个池化层的神经元对应卷积层中的4个，本实验的池化层是没有重叠的，所以每个池化层神经元均对应不同的卷积层神经元,因此（**5**）中的求和符号就不需要了。

$$
\begin{aligned}
\delta_i^l &= \frac{\partial C}{\partial z_i^l} \\
&= \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_i^l} \\
&= \delta_k^{l+1} \cdot \frac{\partial \frac{1}{4}\left(f(z_i^l) + f(z_j^l) + f(z_m^l) + f(z_n^l)\right)}{\partial z_i^l} \\
&= \frac{1}{4}\,\delta_k^{l+1} \cdot f'(z_i^l)
\end{aligned}
$$

向量形式为

$$\delta^l = \frac{1}{pooldim^2} \cdot upsample\left(\delta^{l+1}\right) \bigodot f'(z^l)$$

本实验中激活函数f(x)为sigmoid函数,pooldim为2。
例：卷积层经sigmoid函数激活后的矩阵A为

$$\begin{matrix} 0.1000 & 0.2000 & 0 & 0.1000 \\ 0.9000 & 0 & 0.1000 & 0.3000 \\ 0.4000 & 0.1000 & 0.1000 & 0 \\ 0 & 0 & 0 & 0.1000 \end{matrix}$$

池化层的δ为

$$\begin{matrix} 3 & 1 \\ 0 & 2 \end{matrix}$$

则可求得卷积层的δ'为

$$\begin{matrix} 0.0750 & 0.1500 & 0 & 0.0250 \\ 0.6750 & 0 & 0.0250 & 0.0750 \\ 0 & 0 & 0.0500 & 0 \\ 0 & 0 & 0 & 0.0500 \end{matrix}$$

求最后结果时可以利用matlab中的kron函数计算，该函数可计算两矩阵的Kronecker product。代码为

```
kron(delta, ones(2,2)) .* A / (poolDim)^2
```

四. 求梯度

1. 对输出层到池化层的权值W_{ij}^l, b_i^l
   这两层是全连接的，所以

$$\begin{aligned} \frac{\partial C}{\partial W_{ij}^l} &= \frac{\partial C}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial W_{ij}^l} \\ &= \delta_i^l \cdot a_j^{l-1} \end{aligned}$$

$$\begin{aligned} \frac{\partial C}{\partial b_i^l} &= \frac{\partial C}{\partial z_i^l} \cdot \frac{\partial z_i^l}{\partial b_i^l} \\ &= \delta_i^l \end{aligned}$$

2. 输入层与卷积层之间的权值W_{ij}^l, b^l
   令卷积层的δ为Δ,卷积层的输入为Z,w为某权值,则

$$\frac{\partial C}{\partial w} = sum\{\Delta \bigodot \frac{\partial Z}{\partial w} \bigodot \nabla_Z C\} \quad (7)$$

   sum{·}为矩阵中所有值相加。

$$\frac{\partial C}{\partial b} = sum\{\Delta \bigodot \nabla_Z C\} \qquad (8)$$

   例：

   $\Delta = \begin{matrix} 1 & 2 \\ 0 & 1 \end{matrix}$, 输入图像$A = \begin{matrix} 2 & 0 & 1 \\ 1 & 2 & 3 \\ 1 & 1 & 1 \end{matrix}$, 某filter为$F = \begin{matrix} x & y \\ p & q \end{matrix}$,偏置为b则

$$\frac{\partial C}{\partial x} = 1 * 2 * f'(2x + p + 2q + b) + 2 * 0 * f'(y + 2p + 3q + b) + 0 * 1 * f'(x + 2y + p + q + b) + 1 * 2 * f'(2x + $$

$$\frac{\partial C}{\partial b} = 1 * f'(2x+p+2q+b) + 2 * f'(y+2p+3q+b) + 0 * f'(x+2y+p+q+b) + 1 * f'(2x+3y+p+q+b$$

五.代码

只给出需要自己写的部分的代码：
cnnCost.m

```
function [cost, grad, preds] = cnnCost(theta,images,labels,numClasses,...
                                filterDim,numFilters,poolDim,pred)
% Calcualte cost and gradient for a single layer convolutional
% neural network followed by a softmax layer with cross entropy
% objective.
%
% Parameters:
%  theta      -  unrolled parameter vector
%  images     -  stores images in imageDim x imageDim x numImges
%                array
%  numClasses -  number of classes to predict
%  filterDim  -  dimension of convolutional filter
%  numFilters -  number of convolutional filters
%  poolDim    -  dimension of pooling area
%  pred       -  boolean only forward propagate and return
%                predictions
%
%
% Returns:
%  cost       -  cross entropy cost
%  grad       -  gradient with respect to theta (if pred==False)
%  preds      -  list of predictions for each example (if pred==True)


if ~exist('pred','var')
    pred = false;
end;


imageDim = size(images,1); % height/width of image
numImages = size(images,3); % number of images

%% Reshape parameters and setup gradient matrices

% Wc is filterDim x filterDim x numFilters parameter matrix
% bc is the corresponding bias

% Wd is numClasses x hiddenSize parameter matrix where hiddenSize
% is the number of output units from the convolutional layer
% bd is corresponding bias
[Wc, Wd, bc, bd] = cnnParamsToStack(theta,imageDim,filterDim,numFilters,...
                        poolDim,numClasses);

% Same sizes as Wc,Wd,bc,bd. Used to hold gradient w.r.t above params.
Wc_grad = zeros(size(Wc));
Wd_grad = zeros(size(Wd));
bc_grad = zeros(size(bc));
bd_grad = zeros(size(bd));

%%======================================================================
%% STEP 1a: Forward Propagation
%  In this step you will forward propagate the input through the
%  convolutional and subsampling (mean pooling) layers.  You will then use
%  the responses from the convolution and pooling layer as the input to a
%  standard softmax layer.

%% Convolutional Layer
%  For each image and each filter, convolve the image with the filter, add
%  the bias and apply the sigmoid nonlinearity.  Then subsample the
%  convolved activations with mean pooling.  Store the results of the
%  convolution in activations and the results of the pooling in
```

```
%  activationsPooled.  You will need to save the convolved activations for
%  backpropagation.
convDim = imageDim-filterDim+1; % dimension of convolved output
outputDim = (convDim)/poolDim; % dimension of subsampled output

% convDim x convDim x numFilters x numImages tensor for storing activations
activations = zeros(convDim,convDim,numFilters,numImages);

% outputDim x outputDim x numFilters x numImages tensor for storing
% subsampled activations
activationsPooled = zeros(outputDim,outputDim,numFilters,numImages);

%%% YOUR CODE HERE %%%
activations = cnnConvolve(filterDim, numFilters, images, Wc, bc);
activationsPooled = cnnPool(poolDim, activations);
% Reshape activations into 2-d matrix, hiddenSize x numImages,
% for Softmax layer
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);
output = exp(Wd * activationsPooled + repmat(bd, 1, numImages));
probs = bsxfun(@rdivide, output, sum(output));
%%% YOUR CODE HERE %%%

%%======================================================================
%% STEP 1b: Calculate Cost
%  In this step you will use the labels given as input and the probs
%  calculate above to evaluate the cross entropy objective.  Store your
%  results in cost.

%%% YOUR CODE HERE %%%
%cost = 0; % save objective into cost
idx = sub2ind(size(probs), labels', 1:numImages);
cost = -sum(log(probs(idx))) / numImages;

% Makes predictions given probs and returns without backproagating errors.
if pred
    [~,preds] = max(probs,[],1);
    preds = preds';
    grad = 0;
    return;
end;

%%======================================================================
%% STEP 1c: Backpropagation
%  Backpropagate errors through the softmax and convolutional/subsampling
%  layers.  Store the errors for the next step to calculate the gradient.
%  Backpropagating the error w.r.t the softmax layer is as usual.  To
%  backpropagate through the pooling layer, you will need to upsample the
%  error with respect to the pooling layer for each filter and each image.
%  Use the kron function and a matrix of ones to do this upsampling
%  quickly.

%%% YOUR CODE HERE %%%
%Wd: numClasses x hiddenSize
%Wc: filterDim x filterDim x numFilters parameter matrix
%delta_L: numClasses x numImages
%delta_pooling: hiddenSize x numImages
%delta_conv: convDim x convDim x numFilters x numImages
%images: imageDim x imageDim x numImges
e = zeros(numClasses, numImages);
idx = sub2ind(size(e), labels', 1:numImages);
e(idx) = 1;
delta_L = probs - e;
delta_pooling = reshape((Wd' * delta_L) .* ones(size(activationsPooled)), outputDim, outputDi
```

```
m, numFilters, numImages);
delta_conv = zeros(convDim, convDim, numFilters, numImages);

for i = 1 : numFilters
    for j = 1 : numImages
        delta_conv(:, :, i, j) = kron(delta_pooling(:, :, i, j), ones(poolDim, poolDim)) / (poolDim)^2;
    end
end

%%======================================================================
%% STEP 1d: Gradient Calculation
%  After backpropagating the errors above, we can use them to calculate the
%  gradient with respect to all the parameters.  The gradient w.r.t the
%  softmax layer is calculated as usual.  To calculate the gradient w.r.t.
%  a filter in the convolutional layer, convolve the backpropagated error
%  for that filter with each image and aggregate over images.

%%% YOUR CODE HERE %%%
for i = 1 : numFilters
    for j = 1 : numImages
        Wc_grad(:, :, i) =  Wc_grad(:, :, i) + conv2(images(:, :, j), rot90(delta_conv(:, :, i, j)...
            .* activations(:, :, i, j) .* (1 - activations(:, :, i, j)), 2), 'valid');
        bc_grad(i) = bc_grad(i) + sum(sum(delta_conv(:, :, i, j)...
            .* activations(:, :, i, j) .* (1 - activations(:, :, i, j))));
    end
    Wc_grad(:, :, i) = Wc_grad(:, :, i) / numImages;
    bc_grad(i) = bc_grad(i) / numImages;
end

for i = 1 : numImages
    Wd_grad = Wd_grad + delta_L(:, i) * activationsPooled(:, i)';
end
Wd_grad = Wd_grad / numImages;

bd_grad = sum(delta_L, 2) / numImages;

%% Unroll gradient into grad vector for minFunc
grad = [Wc_grad(:) ; Wd_grad(:) ; bc_grad(:) ; bd_grad(:)];
end
```

minFuncSGD.m

```
function [opttheta] = minFuncSGD(funObj,theta,data,labels,...
                        options)
% Runs stochastic gradient descent with momentum to optimize the
% parameters for the given objective.
%
% Parameters:
%  funObj     -  function handle which accepts as input theta,
%                data, labels and returns cost and gradient w.r.t
%                to theta.
%  theta      -  unrolled parameter vector
%  data       -  stores data in m x n x numExamples tensor
%  labels     -  corresponding labels in numExamples x 1 vector
%  options    -  struct to store specific options for optimization
%
% Returns:
%  opttheta   -  optimized parameter vector
%
% Options (* required)
%  epochs*     - number of epochs through data
%  alpha*      - initial learning rate
%  minibatch*  - size of minibatch
%  momentum    - momentum constant, defualts to 0.9


%%======================================================================
%% Setup
assert(all(isfield(options,{'epochs','alpha','minibatch'})),...
```

```
        'Some options not defined');
if ~isfield(options,'momentum')
    options.momentum = 0.9;
end;
epochs = options.epochs;
alpha = options.alpha;
minibatch = options.minibatch;
m = length(labels); % training set size
% Setup for momentum
mom = 0.5;
momIncrease = 20;% what??
velocity = zeros(size(theta));

%%======================================================================
%% SGD loop
it = 0;
for e = 1:epochs

    % randomly permute indices of data for quick minibatch sampling
    rp = randperm(m);

    for s=1:minibatch:(m-minibatch+1)
        it = it + 1;

        % increase momentum after momIncrease iterations
        if it == momIncrease
            mom = options.momentum;
        end;

        % get next randomly selected minibatch
        mb_data = data(:,:,rp(s:s+minibatch-1));
        mb_labels = labels(rp(s:s+minibatch-1));

        % evaluate the objective function on the next minibatch
        [cost grad] = funObj(theta,mb_data,mb_labels);

        % Instructions: Add in the weighted velocity vector to the
        % gradient evaluated above scaled by the learning rate.
        % Then update the current weights theta according to the
        % sgd update rule

        %%% YOUR CODE HERE %%%
        velocity = mom * velocity - alpha * grad;
        theta = theta + velocity;
        fprintf('Epoch %d: Cost on iteration %d is %f\n',e,it,cost);
    end;

    % aneal learning rate by factor of two after each epoch
    alpha = alpha/2.0;

end;

opttheta = theta;

end
```

参考资料：

1.UFLDL

2.Neural networks and deep learning

3.Deep learning：五十一(CNN的反向求导及练习)