Lab Assignment 4

Convolutional Neural Networks

Eleftherios Karamoulas - S3261859
Panagiotis Tzafos - S3302148

March 26, 2017

2 Theory questions

1. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting(CS231n lecture). In practice, its use is to downsample its input, reducing the amount of resources that the network needs in order to perform the computations by reducing the dimensions of the input using a filter, a stride and an elementwise activation function. Also, after the downsampling it is obvious that the network will have less parameters. As a result, it will be able to generalize better in new situations and avoid overfitting(small training set error, large error for new examples).

2. Weight sharing is a very important feature, as it can dramatically reduce the number of weights(less computations). The idea behind this is that the number of unique sets of weights can be equal to the number of filters that are going to be used, as we are about use these to every different region of the image we are scanning. This is when the network is interested to examine if a feature exists in the image indepentantly by its position. It can be applied because of the fact that, assuming that a single depth slice(filter) weight configuration is associated with a single feature that our network is looking for(edges, circles etc.), in every single one spatial region that the filter checks, it responsible of tracing these specific features(Not useful when the network has to learn in different positions of the image different features, e.g. a training set of faces, that are centralized).

3. The time is less in ReLu activation function compared to the sigmoid and the tanh activation functions, as it does not involve expensive operations(exponentials, etc.), as it can be implemented by simply thresholding a matrix of activations at zero. The results are better because of the properties that ReLu function provides, like sparse representation(the more activations are ≤ 0 the more sparse is our model and this is generally more beneficial than dense representations that sigmoid and tanh provide) and ReLu function non-saturating form(sigmoid and tanh functions are vanishing gradient, as the absolute value of x in $\alpha = Wx + b$ increases the gradient of sigmoid becomes increasingly small compared to ReLu that the gradient is constant.)

4. (a) From the given data and the formula to compute the output we have that $W = 12, F = 3, P = 0, S = 1$, so output $= \frac{W-F+2*P}{S} + 1 = \frac{12-3+2*0}{1} + 1 = 10$ and because we have 3 filters the total neurons will be 10 x 10 x 3 = 300. The total number of parameters will be 300 x 1 weights for the full depth(grayscale) of the input image + 3 for the 3 biases of each filters = 303.

   (b) The input layer is our image and if we assume that each pixel is one neuron, our input layer consists of 12 x 12 = 144 neurons and each of them will be fully connected to our neurons in the hidden layer which consists of 300 neurons. As a result the total connections between input and hidden layer will be 144 x 300 = 43200 weights + 300 biases = 43500. It is obvious that the number of parameters of the fully-connected layer compared with a convolutional layer is much bigger, and that makes convolutional layer more efficient.

5. This approach to solve the car decision problem is questionable because even though we can use CNN's and get a network that can decide between cars an average solution, the features that characterise the car can be subjective for different customers depending on their needs and comforts.

4 Convolutional Layer

```
function convolvedFeatures = cnnConvolve(filterDim, numFilters, images, W, b)
%cnnConvolve Returns the convolution of the features given by W and b with
%the given images
%
% Parameters:
%  filterDim - filter (feature) dimension
%  numFilters - number of feature maps
%  images - large images to convolve with, matrix in the form
%           images(r, c, image number)
%  W, b - W, b for features from the sparse autoencoder
%         W is of shape (filterDim,filterDim,numFilters)
%         b is of shape (numFilters,1)
%
% Returns:
%  convolvedFeatures - matrix of convolved features in the form
%                      convolvedFeatures(imageRow, imageCol, featureNum, imageNum)

numImages = size(images, 3);
imageDim = size(images, 1);
convDim = imageDim - filterDim + 1;
convolvedFeatures = zeros(convDim, convDim, numFilters, numImages);

% Instructions:
%   Convolve every filter with every image here to produce the
%   (imageDim - filterDim + 1) x (imageDim - filterDim + 1) x numFeatures x numImages
%   matrix convolvedFeatures, such that
%   convolvedFeatures(imageRow, imageCol, featureNum, imageNum) is the
%   value of the convolved featureNum feature for the imageNum image over
%   the region (imageRow, imageCol) to (imageRow + filterDim - 1, imageCol + filterDim -
     1)
%
% Expected running times:
%   Convolving with 100 images should take less than 30 seconds
%   Convolving with 5000 images should take around 2 minutes
%   (So to save time when testing, you should convolve with less images, as
%   described earlier)

%%% Add code here
    for i=1:numImages
```

```
        im = images(:,:,i);
        for j=1:numFilters
            cw = rot90(W(:,:,j),2);
            cf = conv2(im,cw,'valid') + b(j);
            convolvedFeatures(:,:,j,i) = 1./(1+exp(-cf));
        end
    end
end
```

Listing 1: cnnConvolve.m

5 Mean Pooling Layer

```
function pooledFeatures = cnnPool(poolDim, convolvedFeatures)
%cnnPool Pools the given convolved features
%
% Parameters:
%  poolDim - dimension of pooling region
%  convolvedFeatures - convolved features to pool (as given by cnnConvolve)
%                      convolvedFeatures(imageRow, imageCol, featureNum, imageNum)
%
% Returns:
%  pooledFeatures - matrix of pooled features in the form
%                   pooledFeatures(poolRow, poolCol, featureNum, imageNum)
%

numImages = size(convolvedFeatures, 4);
numFilters = size(convolvedFeatures, 3);
convolvedDim = size(convolvedFeatures, 1);
pooledFeatures = zeros(convolvedDim / poolDim, ...
        convolvedDim / poolDim, numFilters, numImages);
% Instructions:
%   Now pool the convolved features in regions of poolDim x poolDim,
%   to obtain the
%   (convolvedDim/poolDim) x (convolvedDim/poolDim) x numFeatures x numImages
%   matrix pooledFeatures, such that
%   pooledFeatures(poolRow, poolCol, featureNum, imageNum) is the
%   value of the featureNum feature for the imageNum image pooled over the
%   corresponding (poolRow, poolCol) pooling region.
%
%   Use mean pooling here.

%%% Add code here
    for i=1:numImages
        for j=1:numFilters
            x=1;
            y=1;
            for k=1:poolDim:convolvedDim
                for l=1:poolDim:convolvedDim
                    pooledFeatures(x,y,j,i) = mean2(convolvedFeatures...
                        (k:k+poolDim-1,l:l+poolDim-1,j,i));
                    if(y==(convolvedDim / poolDim))
                        x=x+1;
                        y=1;
```

```
                    else
                        y = y+1;
                    end
                end
            end
        end
    end
end
```

Listing 2: cnnPool.m

6 Forward Pass

```
function [cost, grad, preds, activations] = 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));

%%======================================================================
%% 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

%%% REPLACE THE FOLLOWNG LINE %%%
activations = cnnConvolve(filterDim, numFilters, images, Wc, bc);

% outputDim x outputDim x numFilters x numImages tensor for storing
% subsampled activations

%%% REPLACE THE FOLLOWNG LINE %%%
activationsPooled = cnnPool(poolDim,activations);

% Reshape activations into 2-d matrix, hiddenSize x numImages,
% for Softmax layer
%%% REPLACE THE FOLLOWING LINE %%%
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.

%%% COMPUTE THE SOFTMAX OUTPUT %%%
probs = zeros(numClasses,numImages);
Ywx = Wd*activationsPooled;
```

```
Ywxb = bsxfun(@plus,Ywx,bd);
Ynum = exp(Ywxb);
probs = bsxfun(@rdivide,Ynum,sum(Ynum,1));

%%======================================================================
%% 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.
indexes = sub2ind(size(probs), labels', 1:numImages);
cost = -mean(log(probs(indexes)));

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

deriv = probs;
deriv(indexes) = deriv(indexes) - 1;
deriv = deriv ./ numImages;

Wd_grad = deriv * activationsPooled';
bd_grad = sum(deriv, 2);

deriv2_pooled = Wd' * deriv;
deriv2_pooled = reshape(deriv2_pooled, outputDim, outputDim, numFilters, numImages);
delta_upsampled = zeros(convDim, convDim, numFilters, numImages);

for im_idx=1:numImages
    im = squeeze(images(:,:,im_idx));
    for f_idx=1:numFilters
        delta_pool = (1/poolDim^2) * kron(squeeze(deriv2_pooled(:,:,f_idx,im_idx)), ones
        (poolDim));
        delta_upsampled(:,:,f_idx, im_idx) = delta_pool .* ...
            activations(:,:,f_idx,im_idx).*(1-activations(:,:,f_idx,im_idx));
        delta_pool_sqz = squeeze(delta_upsampled(:,:,f_idx,im_idx));
        cur_grad = conv2(im, rot90(delta_pool_sqz, 2), 'valid');

        Wc_grad(:,:,f_idx) = Wc_grad(:,:,f_idx) + cur_grad;
        bc_grad(f_idx) = bc_grad(f_idx) + sum(delta_pool_sqz(:));
```

```
    end
end

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

end
```

Listing 3: cnnCost.m

7 Experiments

1. Question 1

2. In the beginning of the training we see that our filters look like random noise patches but as our training progresses we see that the filters achieve more structure and finally after 3 epochs we can see clearly on them shapes e.g.(lines, angles, curves) that they can recognize on images that we feed to our network.

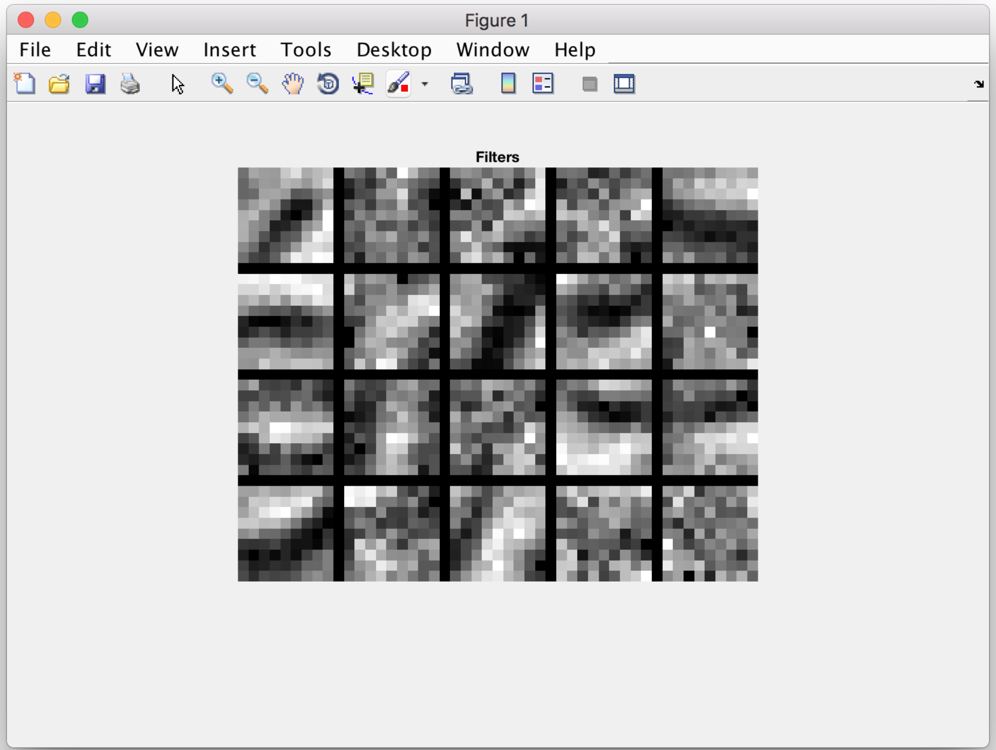

Figure 1: Filters after 3 epochs

3. Our network accuracy after 3 epochs is 97.24%