Neural Networks

Assignment 4 - Convolutional Neural Networks

George Petridis (s3001393)
Jussi Boersma (s2779153)

March 30, 2017

2 Theory Questions

2.1 Name two purposes of pooling and explain them using your own words.

Pooling reduces the number of required neurons, causing this way the amount of parameters and computations to decrease (thus the network becomes quicker). In addition, pooling controls overfitting. By using the mean of the neurons, it does not overfit on the training set.

2.2 Explain the importance of weight sharing in your own words.

In weight sharing all the neurons in a layer have the same weight. This allows the network to use convolution, which reduces the computation time and as such to have fewer parameters for optimization. Although this makes the network less flexible, it allows the network to avoid overfitting.

2.3 Explain why using a rectified linear unit (ReLU) activation function might yield better results in less time than a sigmoid or tanh activation function.

The ReLU activation function might yield better results because it is more biologically plausible than the sigmoid and the tanh activation function

2.4 Suppose we are given an image of 12 x 12 pixels with gray-scale values.

2.4.a Let's assume we have a convolutional layer connected to the input image with 3 filters with dimensions 3 x 3. No zero-padding is used and the stride is 1. How many neurons does this layer contain?

The image dimensions are 12x12 and the filter dimensions are 3x3. With a stride of 1 this means the filter 'fits' in the image 16 times, 4 times horizontally and 4 times vertically. So 1 filter results in 16 neurons. Since we have 3 of those filters, this results in $3 * 16 = 48$ neurons in the layer.

2.4.b Multi-layer perceptrons contain hidden layers that are fully connected to their respective preceding layer. In other words, every neuron in the hidden layer is connected to all neurons of the preceding layer. Suppose we have a hidden layer with the same amount of neurons as found in (a). How many connections are there between the input layer and the hidden layer? Compare this with the number of parameters for the layer in (a). What can you conclude?

This would result in $12 * 12 * 48 = 6912$ connections. From this we can conclude that this number is a lot bigger than the number of parameters in a convolution, therefore the computation time should be a lot smaller.

2.5 Suppose we are given a dataset about the acceptance of a car. It can be labeled unacceptable, acceptable, good or very good. The features are price, maintenance costs, the number of doors, the capacity in number of persons, the lug boot size and a safety measure graded between 0 and 5. Since we have 6 features, we might want to arrange the features in 2 x 3 images. We could now implement a CNN to classify the acceptance. Explain why this approach is questionable.

This approach is questionable because the image would consist of values of different meanings. A value for the price would differ greatly from a graded value for the safety measure. Unless we normalize all the values and make sure that the numbers are scaled (such that they are meaningful in relation to each other), it is of no use to put them in such an image, since doing computations on more than 1 value would result in uninformative numbers. For example, mean pooling in such an image could result in computations with the mean of a price and a graded safety measure, which would be a 'strange' number (if a number at all). In that case it would be more suitable to go over those features 1 by 1 (which would make the use of the image meaningless anyway).

4 Implementing the convolutional layer

4.3 The algorithm

4.3.1 Create a for loop from 1 through numImages.

```
for i = 1:numImages
```

4.3.2 Store the current image from images in a matrix im.

```
im = images(:, :, i); % select images at location i
```

4.3.3 Create a for loop from 1 through numFilters.

```
for j = 1:numFilters
```

4.3.4 Now compute the convolved feature as given in equation (2) by taking the current filter from W and computing the convolution of this filter with im. Use Matlab's built-in convolution function conv2.Do not forget to flip the weights first! You can use Matlab's rot90 to flip the matrix horizontally and vertically. The lecture slides explain why this is necessary. Note that we want to compute a 'valid' convolution. Check out the documentation of conv2 to see how to do this.

```
filter = W(:,:,j); % select filter at location j
filter = rot90(rot90(filter)); % flip the matrix horizontally and vertically
output = conv2(im,filter,'valid'); % use convolution on filter and image
bias = b(j,1); % add bias
activation = sigmoid(bias + output); % sigmoid activation computed
convolvedFeatures(:, :, j, i) = activation; % stored in convolvedFeatures at location (j
    ,i)
```

4.3.5 Store the result in the correct location of convolvedFeatures.

```
convolvedFeatures(:, :, j, i)=activation; %stored in convolvedFeatures at location (j,i)
```

5 Implementing the mean pooling layer

5.1 Create a loop for going through all images in the mini-batch.

```
for i = 1:numImages
```

5.2 Create a loop for going through all filters.

```
for j = 1:numFilters
```

5.3 Pool the features of each filter and store the result in pooledFeatures. Note that you might need more loops to do so. You can use Matlab's mean2 function to compute the mean of the elements of a matrix.

```
for l = 1:(convolvedDim / poolDim) %convolvedDim/poolDim = dimension of pooled output
    %compute pooledFeatures by dividing the convolvedFeature at
    %location (j,i) in (convolvedDim/poolDim)^2 blocks. The
    %pooledversion contains the mean of the values in these
    %blocks.
    pooledFeatures(k,l,j,i) = mean2(convolvedFeatures((poolDim*(k-1)+1):poolDim*k, ...
    (poolDim*(l-1)+1):poolDim*l,j,i));
end
```

6 Implementing the forward pass

6.1 Read the file until the initialization of activations. Rewrite this line such that activations obtains the convolved features.

```
% activations are computed using the cnnConvolve function.
activations = cnnConvolve(filterDim, numFilters, images, Wc, bc);
```

6.2 Now pool the features and store the result in activationsPooled.

```
% the caluculated activations are pooled using the cnnPool function
activationsPooled = cnnPool(poolDim, activations);
```

6.3 The next layer requires the input for one image to be a vector. This means we have to reshape our 4D activationsPooled of size 10 x 10 x 20 x 256 to a 2D matrix of size 2000 x 256. Use Matlab's reshape to do this.

```
% The first dimension is a multiplication of the first 3 dimensions that
% it had before, the second dimension is the fourth dimension. So
% 10*10*20*256 would become 2000*256
size_activations = size(activationsPooled);
dim_1 = size_activations(1)*size_activations(2)*size_activations(3);
dim_2 = size_activations(4);
activationsPooled = reshape(activationsPooled, dim_1, dim_2);
```

6.4 Compute the output of the network.

6.4.a First, let us compute part of the numerator of equation (4).

```
Y_wx = Wd * activationsPooled;
```

6.4.b Now let us compute the matrix $Y_{wx,b}$ in which the bias is also added. We will now add b_i to each i-th row.

```
Y_wxb = bsxfun(@plus, Y_wx, bd);
```

6.4.c Now apply exp to the full matrix. After this we have the matrix that contains the numerator of equation (4). We will denote this matrix Y_{num}.

```
Y_num = exp(Y_wxb);
```

6.4.d Recall that each column of Y_{num} corresponds to a single image. Hence, we want to normalize the activations of the neurons of a single column, such that the sum equals 1. To accomplish this we divide by the sum of the elements in a column.

```
norm_Y_num = bsxfun(@rdivide, Y_num, sum(Y_num,1));
```

6.4.e After dividing by the sums per column, we have obtained the softmax output for the full mini-batch. Store the result of Y in probs.

```
probs = norm_Y_num;
```

7 Experiments

7.1 Mention all parameters that are relevant to the training procedure and what their role is. This includes the parameters outside your own implemented pieces of code. Do not explain formulas/loops etc. Make sure your experiment is reproducible by anybody that at least understands CNNs.

In the training stage, the network loops over all images and applies a convolution on a filter and the image. It does this for every filter (in our case, there are 20 filters). For the activation a bias is added to the output of the convolution. This is stored in the 'convolvedFeatures' variable at location (filterNumber, imageNumber). The parameters used for the training procedure are theta (unrolled parameter vector with initialized weights), images (60000 28*28 images used as training set), labels (the labels that belong to the images, so the network can test itself) and options (struct containing the learn parameters, they are subdivided into number of epochs, number of samples in the minibatch, alpha (learning rate) and a momentum constant).

All these parameters are used to compute the opttheta, which is a vector containing the optimized parameters.

7.2 Include the image of the filters after 3 epochs of training. What can you say about the filters now when compared to the beginning of the training phase? Explain.

Figure 1 shows the filters as they were at the beginning of the training phase, while in Figure 2 there is the representation of the trained filters. Initially we see that there are many random noisy patches, while after the training we can see that there is a lot more structure to each filter.

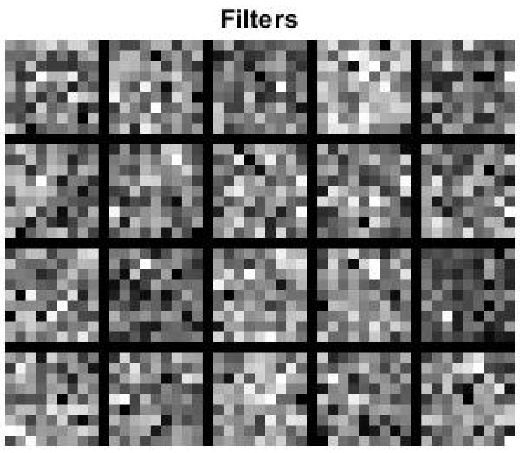

Figure 1: Illustration of the filters at the start of the training.

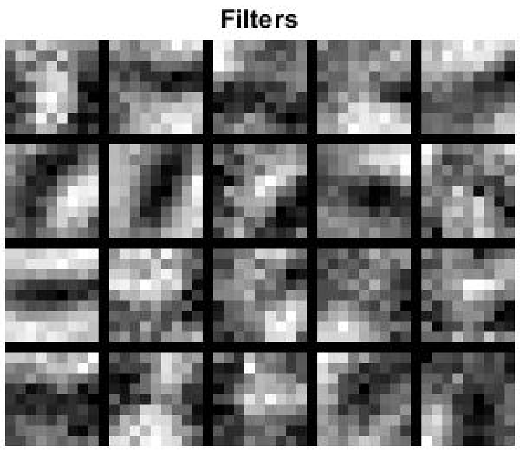

Figure 2: Illustration of the filters after training.

7.3 Report the accuracy of the network on the test set after 3 epochs. It should be higher than 97%.

The accuracy of the trained network after 3 epochs was 97.23%, which was indeed higher than 97%.