Neural Networks for AI

LAB 5

Fernanda Marana (s3902064)
Floris Cornel (s2724685)

June 6, 2019

1 Experiments

1.1 Convince the reader of your report that you have understood the network. 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/loopsetc. Make sure your experiment is reproducible by anybody that at least understands CNNs.

As explained on the laboratory document, the main purpose of this exercise was to implement a CNN to classify images of handwritten digits in the correct prediction. The creation of a CNN - like a typical neural network - also consists on inputs, hidden layers and outputs, but they already assume that the input are images and, as a consequence, the layers have neurons arranged in 3 dimensions: width, height, depth. Additionally, since images consist on great amount of volumes, CNNs are known for having three very important processes: the convolution, the pooling and the normalization (loss) function.

The convolution consists on going through each image and each filter so the activation map can be created. Also, on this step the network starts to learn filters that activate when they detect some type of visual feature. So, by doing the cnnConvolve function, the learning process begins with the specific parameters set on the network. An important note is that the weights are rotated two times to do the convolution (otherwise, we had be doing the cross- correlation, i.e., measuring the similarity between two signals, which is not the objective). The forward propagation in this exercise does the activation of neurons with the convolution, pooling and softmax function while the backpropagation receives the error obtained by the result and backpropagates it through the softmax and subsampling layers.

Pooling's purpose is to reduce the dimensions of the data by obtaining an average, in this case, between a defined number of elements on the matrix. Since images can have big dimensions and we can be dealing with a great number of them, reducing the spatial size of the representation can help the amount of computation in the network be also reduced.

The softmax function is one type of the normalization layers that is responsible to measure the compatibility between a prediction and the ground truth label. In other words, the prediction would be what the neural network detect as the number that is currently processing and the truth label would be the actually number that is represented by the handwriting.

Finally, some important point worthy noticing is that the forward propagation and back propagation are still used to receive - from the input given- a response and "reshape" (do some alterations) on the network based on it.

1.2 What can you say about the filters now when compared to the beginning of the training phase? Explain.

It is possible to see that the filters are starting to create a pattern in which they identify a feature of the image. In the beginning, they were very random but as the network start to learn each number, the patterns are being created. As explained on the section 1.1, the reason why this is happening is due to the forward and back propagation on the network that are helping the learning of what filters activate when they detect some type of visual feature.

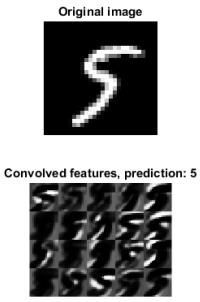

Figure 1: Weight matrix

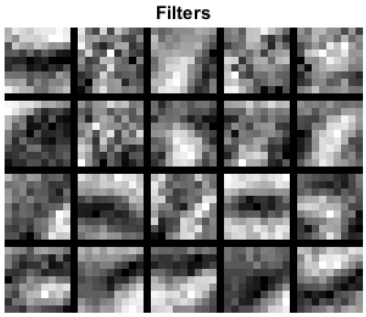

Figure 2: Correct recognition of 3 patterns.

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

The accuracy reached is 0.971600

2 Code

```
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 k=1:numImages
    im = images(:,:,k);
    for l=1:numFilters
        weight = rot90(rot90(W(:,:,l)));
        result = conv2(im,weight,'valid');

        convolvedFeatures(:,:,l,k) = logsig(b(l)+result);
    end
end



end
```

Listing 1: cnnConvolve.m

```
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);
```

```
20
21
22
23 % Instructions:
24 %   Now pool the convolved features in regions of poolDim x poolDim,
25 %   to obtain the
26 %   (convolvedDim/poolDim) x (convolvedDim/poolDim) x numFeatures x numImages
27 %   matrix pooledFeatures, such that
28 %   pooledFeatures(poolRow, poolCol, featureNum, imageNum) is the
29 %   value of the featureNum feature for the imageNum image pooled over the
30 %   corresponding (poolRow, poolCol) pooling region.
31 %
32 %   Use mean pooling here.
33
34 %%% Add code here
35 separation =  poolDim * ones(1, convolvedDim /poolDim);
36
37 for i=1:numImages
38     for j=1:numFilters
39
40         d=mat2cell(convolvedFeatures(:,:,j,i),separation, separation);
41         new = zeros(size(d,1));
42         for k=1:size(d,1)
43             for r=1:size(d,1);
44             new(k,r) = mean2(d{k,r});
45             end
46          end
47
48        % result = reshape(new, (convolvedDim/poolDim),(convolvedDim/poolDim));
49         pooledFeatures(:,:,j,i) = new;
50     end
51
52 end
53
54 end
```

Listing 2: cnnPool

```
1 function [cost, grad, preds, activations] = cnnCost(theta,images,labels,numClasses,...
2                                 filterDim,numFilters,poolDim,pred)
3 % Calcualte cost and gradient for a single layer convolutional
4 % neural network followed by a softmax layer with cross entropy
5 % objective.
6 %
7 % Parameters:
8 %  theta      -  unrolled parameter vector
9 %  images     -  stores images in imageDim x imageDim x numImges
10 %                array
11 %  numClasses -  number of classes to predict
12 %  filterDim  -  dimension of convolutional filter
13 %  numFilters -  number of convolutional filters
14 %  poolDim    -  dimension of pooling area
15 %  pred       -  boolean only forward propagate and return
16 %                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));
Wc_grad_sze = (size(Wc));
Wd_grad_sze = size(Wd);
bc_grad_sze = size(bc);
bd_grad_sze = 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 = zeros(convDim,convDim,numFilters,numImages);
activations = cnnConvolve(filterDim, numFilters, images, Wc, bc);
% outputDim x outputDim x numFilters x numImages tensor for storing
% subsampled activations
size (activations);
%%% REPLACE THE FOLLOWNG LINE %%%
activationsPooled = zeros(outputDim,outputDim,numFilters,numImages);
activationsPooled = cnnPool(poolDim,activations );
size (activationsPooled);

% Reshape activations into 2-d matrix, hiddenSize x numImages,
% for Softmax layer
%%% REPLACE THE FOLLOWING LINE %%%
activationsPooled = reshape(activationsPooled,size(Wd,2) , 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);
probs = Wd*activationsPooled;
for j=1:  size(probs,1)
    probs(j,:) = probs(j,:) + bd(j);
end

Y_num = exp(probs);


for k=1: size(Y_num,2)
    total_sum = sum(Y_num(:,k));
    Y_num(:,k)= Y_num(:,k)/total_sum;

end
probs = Y_num;
%%=========================================================================
%% 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: cnnPool