Assignment 7 Deep Learning

Laura Pijnacker s4332199 and Jacob Gavin s4658981

```
%  Wc      -  filterDim x filterDim x numFilters parameter matrix
%  Wd      -  numClasses x hiddenSize parameter matrix, hiddenSize is
%             calculated as numFilters*((imageDim-filterDim+1)/poolDim)^2
%  bc      -  bias for convolution layer of size numFilters x 1
%  bd      -  bias for dense layer of size hiddenSize x 1
```

1.c Report the size of each of these four variables and what they correspond to.

W_1 = 9 x 9 x 20 parameter matrix. 9 is the filter size for the conv layer and 20 is the number of filters for the conv layer.

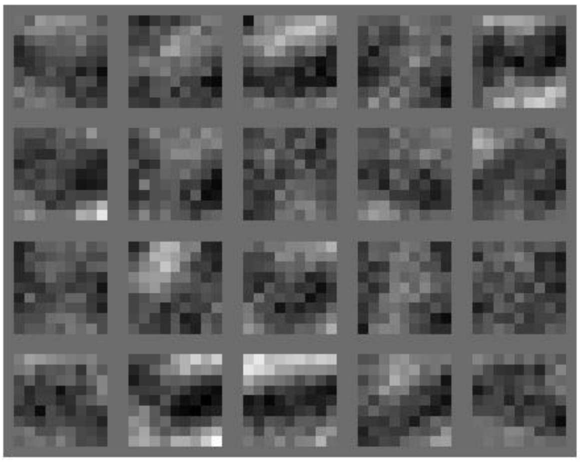
W_2 = 10 x (20*((28-9+1)/2)^2) = 10 x 2000 parameter matrix. 10 is the number of classes, 2000 is the hiddenSize.

b_1 = 20 x 1 bias for convolution layer. 20 is number of filters for the conv layer.

b_2 = 2000 x 1 bias for dense layer. 2000 is the hiddenSize

2-4 See appendix.

5.a We ran the algorithm several times:

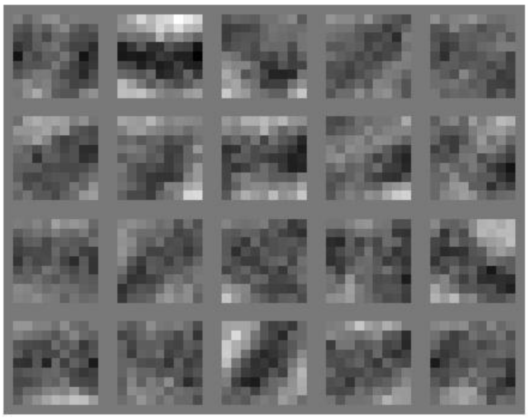


1.

Training (mini-batch) accuracy = 90.625%

Epoch 1: Cost on iteration 125 is 0.199898

Test accuracy = 89.6%

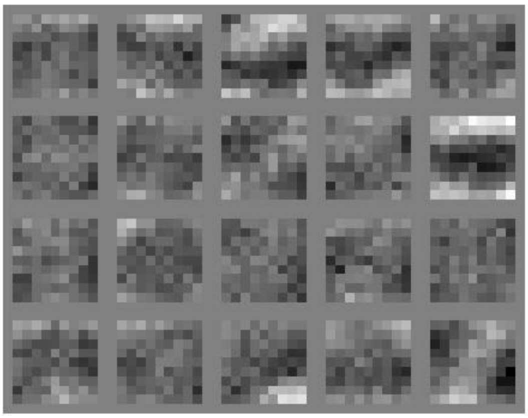


2.

Training (mini-batch) accuracy = 78.125%

Epoch 1: Cost on iteration 125 is 0.467494

Test accuracy = 89.9%

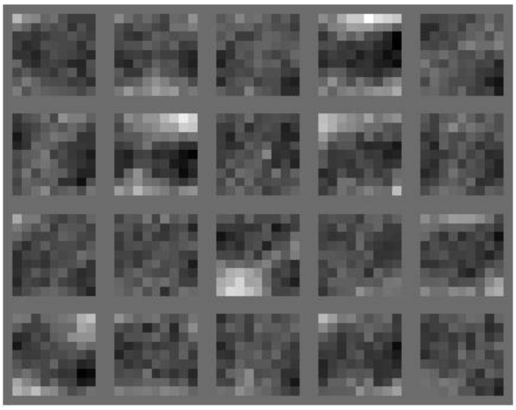


3.

Training (mini-batch) accuracy = 93.75%

Epoch 1: Cost on iteration 125 is 0.504021

Test accuracy = 88.8%



4.

Training (mini-batch) accuracy = 93.75%

Epoch 1: Cost on iteration 125 is 0.221329

Test accuracy = 88.7%

In all four of the cases above the accuracy was right below 90%. While a convolutional layer could work very well for this problem, it's not perfect in our case because the training is done in mini-batches of 32. Because of the small amount of digits, it's very likely for overfitting to occur.

b. Values in the images represent higher weights for that part of the image, on a certain layer. Lighter pixels mean smaller weights. The network is less responsive for these points. For example, weights on the edges/corners are often light-coloured because digits rarely touch corners, therefore making the corner of the images not very useful in classificating digits.

Appendix:

softmax.m

```
% This function implements the softmax function. It computes and returns
% the output of the softmax function. Unlike the sigmoid function, the
% gradient of the softmax function is not required.
function f_a = softmax(a)

% TIP: The softmax function requires exponentiating a. Exponentiating large
% numbers might result in an overflow. You can prevent an overflow by
% subtracting the maximum of a(:, i) from a(:, i) for all i before
% exponentiating a. You do *not* have to modify the following.
a = bsxfun(@minus, a, max(a));

% Compute the output of the softmax function (i.e. f_a).
%
% Write your code below:
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
for i=1:size(a, 2)
    f_a(:,i) = exp(a(:,i))./sum(exp(a(:,i)));
end
% ---------------------------------------------------------------------------
```

```
% ---------------------------------------------------------------------------

end
```

forwardprop.m

```
% This function implements forward propagation. It computes and returns the
% outputs of the hidden and output units as well as the gradients of the
% hidden units  (i.e. f_a_2, f_a_3 and grad_f_a_2).
function [f_a_2, f_a_3, grad_f_a_2] = forwardprop(args, b_1, b_2, W_1, W_2,
X)

% Compute the inputs of the hidden units (i.e. a_2).
%
% For each image and hidden unit pair (i.e. i and j, respectively), you
% should convolve X(:, :, i) with W_1(:, :, j) and add b_1(j) to obtain the
% corresponding a_2 (i.e. a_2(:, :, j, i)).
%
% TIP: You can use the conv2 function for convolution. If you use the conv2
% function, you should use the 'valid' shape parameter. Note that the conv2
% function flips its second input horizontally and vertically, which is
% something that you do not want and should prevent by flipping the second
% input of the conv2 function horizontally and vertically before using
% it. For example, to flip a matrix A horizontally and vertically, you can
% use rot90(A, 2).
%
% A nested for loop is provided for your convenience.
%
% Write your code in the for loop:
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
a_2 = zeros(args.imageDim - args.filterDim + 1, args.imageDim -
args.filterDim + 1, args.numFilters, size(X, 3));

for i = 1 : size(X, 3) % loop over images

    for j = 1 : args.numFilters % loop over hidden units

        % Write your code here
        % -----------------------------------------------------------------

        %flip = rot90(W_1, 2);
        a_2(:, :, j, i) = conv2(X(:,:,i), rot90(W_1(:,:,j), 2), 'valid') +
b_1(j);

        % -----------------------------------------------------------------

    end

end

% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------

% Compute the outputs of the hidden units (i.e. f_a_2) and their gradients
% (i.e. grad_f_a_2).
%
% You do *not* have to modify the following.
[f_a_2, grad_f_a_2] = sigmoid(a_2);
```

```
% Compute the mean pooled outputs of the hidden units (i.e. f_a_2).
%
% For each image and hidden unit pair (i.e. i and j, respectively), you
% should mean pool the corresponding f_a_2 (i.e. f_a_2(:, :, j, i)). That
% is, you should average all nonoverlapping (by default) 2 by 2 blocks of
% f_a_2(:, :, j, i).
%
% TIP: You can use the im2col function to rearrange blocks of
% f_a_2(:, :, j, i) into columns, average the columns and use the col2im
% function to rearrange the columns into blocks.
%
% A nested for loop is provided for your convenience.
%
% Note that the mean pooled outputs of the hidden units should be stored in
% f_a_2. You might want the store them in a temporary variable during the
% loop and assign the temporary variable to f_a_2 after the loop.
%
% Write your code in the for loop:
% -----------------------------------------------------------------------
% -----------------------------------------------------------------------
temp = zeros(size(f_a_2, 1) / args.poolDim, size(f_a_2, 2) / args.poolDim,
size(f_a_2, 3), size(f_a_2, 4));

for i = 1 : size(X, 3) % loop over images

    for j = 1 : args.numFilters % loop over hidden units

        % Write your code here
        % ---------------------------------------------------------------

        %f_a_2(:, :, j, i)
        col = im2col(f_a_2(:, :, j, i),[2 2], 'distinct');
        m = mean(col);
        temp(:, :, j, i) = col2im(m, [1 1], [10 10]);

        % ---------------------------------------------------------------

    end

end

f_a_2 = temp;
% -----------------------------------------------------------------------
% -----------------------------------------------------------------------

% Compute the inputs of the third layer units (i.e. a_3).
%
% You do *not* have to modify the following.
a_3 = bsxfun(@plus, W_2 * reshape(f_a_2, [], size(f_a_2, 4)), b_2);

% Compute the outputs of the output units (i.e. f_a_3).
%
% You do *not* have to modify the following.
f_a_3 = softmax(a_3);

end
```

backprop.m

```
% This function implements back propagation. It computes and returns the
% gradients of the cross entropy function with respect to the biases and
% weights (i.e. grad_E_b_1, grad_E_b_2, grad_E_W_1 and grad_E_W_2).
function [grad_E_b_1, grad_E_b_2, grad_E_W_1, grad_E_W_2] = backprop(args,
f_a_2, f_a_3, grad_f_a_2, T, W_2, X)

% Compute the deltas of the output units (i.e. delta_3).
%
% You do *not* have to change the following.
delta_3 = (f_a_3 - T) / size(X, 3);

% Compute the gradients of the error function with respect to the biased
% and weights in the output layer (i.e. grad_E_b_2 and grad_E_w_2).
%
% You do *not* have to change the following.
grad_E_b_2 = sum(delta_3, 2);
grad_E_W_2 = delta_3 * reshape(f_a_2, [], size(X, 3))';

% Compute the deltas of the hidden units (i.e. delta_2).
%
% You do *not* have to change the following.
delta_2 = W_2' * delta_3;
delta_2 = reshape(delta_2, (args.imageDim - args.filterDim + 1) /
args.poolDim, (args.imageDim - args.filterDim + 1) / args.poolDim,
args.numFilters, []);

for i = size(X, 3) : -1 : 1

    for j = args.numFilters : -1 : 1

        temp(:, :, j, i) = (1 / args.poolDim ^ 2) * kron(delta_2(:, :, j,
i), ones(args.poolDim));

    end

end

delta_2 = temp;
delta_2 = delta_2 .* grad_f_a_2;

% Compute the gradients of the cross entropy function with respect to the
% biases and weights in the hidden layer (i.e. grad_E_b_1 and grad_E_W_1).
%
% For each hidden unit (i.e. j), you should sum delta_2(:, :, j, :) over
% everything but j to obtain the corresponding grad_E_b_1 (i.e.
% grad_E_b_1(j)).
%
% Recall that in the case of a fully-connected hidden layer, grad_E_W_1 is
% given by delta_2 * X'. However, in the case of a convolutional hidden
% layer, grad_E_W_1 is given by convolution of delta_2 and X. For each
% image and hidden unit pair (i.e. i and j, respectively), you should
% convolve X(:, :, i) with delta_2(:, :, j, i) to obtain the corresponding
% grad_E_W_1 (i.e. grad_E_W_1(:, :, j, i)). You should then sum
% grad_E_W_1(:, :, j, i) over i.
%
% TIP: You can use the conv2 function for convolution. If you use the conv2
% function, you should use the 'valid' shape parameter. Note that the conv2
% function flips its second input horizontally and vertically, which is
% something that you do not want and should prevent by flipping the second
```

```matlab
% input of the conv2 function horizontally and vertically before using
% it. For example, to flip a matrix A horizontally and vertically, you can
% use rot90(A, 2).
%
% A nested for loop is provided for your convenience.
%
% Write your code in the for loop:
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------
grad_E_b_1 = zeros(args.numFilters, 1);
grad_E_W_1 = zeros(args.imageDim - size(delta_2, 1) + 1, args.imageDim -
size(delta_2, 2) + 1, args.numFilters);

for i = 1 : size(X, 3) % loop over images

    for j = 1 : args.numFilters % loop over hidden units

        % Write your code here
        % -------------------------------------------------------------------
        grad_E_b_1(j) = grad_E_b_1(j) + sum(sum(delta_2(:, :, j, i)));

        grad_E_W_1(:, :, j) = grad_E_W_1(:, :, j) + conv2(X(:, :, i),
rot90(delta_2(:, :, j, i), 2), 'valid');
        %grad_E_W_1(i) = sum(grad_E_W_1(:, :, j, i));

        % -------------------------------------------------------------------

    end

end
% ---------------------------------------------------------------------------
% ---------------------------------------------------------------------------

end
```