Lab 2

Learning and convolutional neural networks

2.1 Learning a linear classifier

In this part, we will try to learn a linear classifier for blood cell detection. Note that the classifier could also be viewed as a minimal neural network consisting of three parts: a scalar product node (or fully-connected node), a constant (or bias) term and a logistic sigmoid function. To find good parameters we will try to minimize the negative log-likelihood over a small training set.

The output from our classifier is a probability p for the input patch being centered at a cell centre. The sigmoid function will make sure that $0 \leq p \leq 1$. To be more precise the output is

$$p = \frac{e^y}{1+e^y} \quad \text{where} \quad y = I \cdot w + w_0. \tag{2.1}$$

Instead of testing a bunch of manually chosen w's and w_0's, we will try to learn good values for all the parameters. This requires training examples, that you find in `cell_data.mat`.

Ex 2.1 Load the data using

```
load cell_data.mat
```

It loads a structure, `cell_data`, with two fields, `fg_patches` and `bg_patches`, corresponding to positive (centered blood cells) negative examples respectively.

Ex 2.2 Create two new variables, `examples` and `labels`. `examples` should be a cell structure containing the all the patches (both positives and negatives) and `labels` should be an array with the same number of elements such that `labels(i)` $= 1$ if `examples{i}` is a positive example, and `labels(i)` $= 0$ otherwise.

Ex 2.3 Split the data into training, (`examples_train`, `labels_train`), and validation, (`examples_val`, `labels_val`). The two should have a similar structure to `examples` and `labels`. Write on the report which percentage of the data did you use for validation.

2.2 Training the classifier

We will try to find parameters that minimize the negative log-likelihood on the training data. More precisely,

$$L(\theta) = \sum_{i \in S_+} -\ln p_i + \sum_{i \in S_-} -\ln(1-p_i) = \sum_i L_i(\theta), \tag{2.2}$$

where p_i refers to the classifier output for the ith training example. As in the lectures we will refer to the terms here as the partial loss L_i.

Before doing the next exercise, you need to work out how to compute the gradient of the partial loss L_i.

Ex 2.4 Make a function

```
[wgrad, w0grad] = partial_gradient(w, w0, example_train, label_train)
```

that computes the derivatives of the partial loss L_i with respect to each of the classifier parameters. Let the output `wgrad` be an array of the same size as the weight image, `w` (and let `w0grad` be a number).

At each iteration of stochastic gradient descent, a training example, i, is chosen at random. For this example the gradient of the partial loss, L_i, is computed and the parameters are updated according to this gradient. The most common way to introduce the randomness is to make a random reordering of the data and then going through it in the new order. One pass through the data is called an epoch.

Ex 2.5 Make a function

```
[w, w0] = process_epoch(w, w0, lrate, examples_train, labels_train)
```

that performs one epoch of stochastic gradient descent.

Ex 2.6 Initialize `w = s * randn(35,35);`, with `s = 0.01` and `w0 = 0;` and run 5 epochs on your training examples. Plot `w` after each epoch (or after each iteration if you are curious), to get a sense of what is happening. Also, try using different `s={10,1,0.1}` and plot `w` after 5 epochs for each value of `s`. Include on the report visualizations of `w` for the different values of `s`, along with an written explanation of what is happening.

Ex 2.7 As said before, at each iteration of stochastic gradient descent, a training example is chosen at random. Check what happens to `w` after 5 epochs when that training example is not chosen randomly but in sequence, i.e. first $i = 1$, then $i = 2$, and so on. Include on the report a visualization of `w` for this case. (Don't forget to change back your function to a random choice of i after this exercise)

Ex 2.8 Make a function

```
predicted_labels = classify(examples_val,w,w0);
```

that applies the classifier to the example data. After that, use it on `examples_train` and `examples_val` and check how much accuracy it gets for each by comparing the predicted labels with `labels_train` and `labels_val` respectively. Write on your report the highest accuracy you were able to achieve in the training and validation data. *Hint:* train the classifier for longer than 5 epochs to make sure that it converges.

Ex 2.9 The data for training this classifier consists on only 400 examples (less if you consider that you have split it into training and validation). To achieve higher accuracy it might be useful to perform some data augmentation before the training. In this exercise you will increase the number of elements in the training examples by M times. Make a function

```
[examples_train_aug,labels_train_aug] = augment_data(examples_train,labels_train,M)
```

that takes each sample of the original training data and applies M random rotations (you can use *Matlab* function `imrotate`), from which result M new examples. Store these new examples in `examples_train_aug` and their corresponding labels in `labels_train_aug`. Train the classifier with this augmented data and write on your report the new values for accuracy on the training and validation examples.

2.3 Convolutional neural networks

In the last part, your task is to train convolutional neural networks using Matlab.

Ex 2.10 Run

```
[imgs, labels] = digitTrain4DArrayData;
```

to load a dataset of images of digits into Matlab. You will find the 10000 digit images in `imgs`. Plot a few of them to see what the data looks like.

```
imagesc(imgs(:,:,:,342)), axis image, colormap gray
```

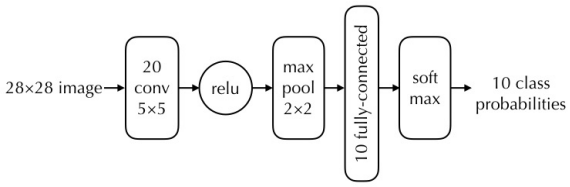
The next step is to define a network for classification. In Matlab, you do this by simply giving an array of the layers. For example, this would be a linear classifier similar to the one you trained for cells:

```
layers = [
   imageInputLayer([35 35 1]);
   fullyConnectedLayer(1);
   softmaxLayer();
   classificationLayer()];
```

Ex 2.11 Make a function

```
layers = basic_cnn_classifier()
```

that implements the following network in Matlab:



Apart from the layers above the functions `convolution2dLayer`, `reluLayer` and `maxPooling2dLayer` will be useful. Note that you have to set the *stride* for max pooling to 2 to get the expected downsampling.

Ex 2.12 Create a set of training options telling Matlab to use stochastic gradient descent with momentum (SGDM), for the optimization:

```
options = trainingOptions('sgdm');
```

Now train the network (using default parameters) by running

```
net = trainNetwork(imgs, labels, layers, options)
```

Ex 2.13 Try the network on a few of the training images. You can use `net.predict(img)` to get the ten output probabilities or `net.classify(img)` to get the most probable class.

Ex 2.14 Work out how many trainable parameters your network contains. (This is a common exam question.) Include the answer in your submission. If you explore what the data structure `net` actually contains, you can find the answer there as well. Note that the convolution layer does not use padding so the output from the convolution layer is smaller than the input.

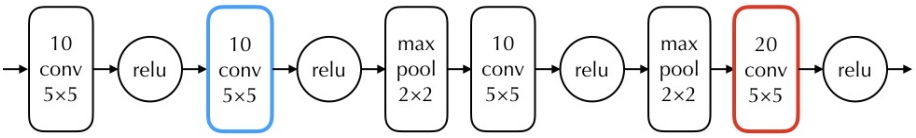
Ex 2.15 Matlab prints a lot of output, for example the accuracy on the training set. Recall from the lectures that this number is not very good for judging the quality of a classifier. Instead we should save a subset of the data as a validation set, that we can use to evaluate the trained network. Make a function

```
net = train_classifier(layers, imgs_train, labels_train, imgs_val, labels_val)
```

that runs a few epochs of training and then evaluates the accuracy on the validation set. In this case, Matlab has given us a separate test set, so we don't have to save images for that purpose. *Matlab hints:* You can run multiple images at once by stacking them along the fourth dimension. If you want to continue training the same network you can run

```
net = train_classifier(net.Layers, imgs_train, labels_train, imgs_val, labels_val)
```

Ex 2.16 To run a convolutional neural network we have to perform a massive amount of computations. Hence it is very important to consider the computational load when designing a network. For the network below, compare (roughly) the time consumption of the blue and the red. You can ignore effects of padding. Include your answer and your motivation in your submission.



Ex 2.17 Replace the blue box of the network in the figure above by a sequence of two layers of 10 3×3 convolutional filters. What changes in terms of network parameters, time consumption and accuracy? Answer this question on your report.

Ex 2.18 Make a copy of `basic_cnn_classifier.m` and name it `better_cnn_classifier.m`. Try modifying the network by adding more layers. Also experiment with the training options. How much can you improve the results? Include precision and recall for the alternative network in your submission.

Ex 2.19 Load the builtin test images using

```
[imgs_test, labels_test] = digitTest4DArrayData;
```

Run the network on all the test images. You apply the network to an image using

```
pred = net.classify(image)
```

Compute precision and recall for each of the 10 classes and include these in your submission.

Ex 2.20 Save three of the failure cases with names indicating what digit they were mistaken for. Include these in your submission. You can use `imwrite(img, 'mistaken_as_5.png')` to save an image if it is correctly scaled. Have a look at the file before submitting it, so it looks right.

Report

Hand in your code together with the images and results indicated in the exercises. Avoid uploading all the data to ping-pong as it will use a huge amount of storage.

Checklist

- **Function:** partial_gradient, process_epoch, classify, augment_data, basic_cnn_classifier, better_cnn_classifier, train_classifier. These will be tested automatically so make sure all submitted functions have the same name, input and output as defined in the lab PM.
- **Answers:** Ex 2.3, Ex 2.6, Ex 2.14, Ex 2.16, Ex 2.17. Include the answers in a separate text or pdf file or in the submission comment in ping-pong. All answers need to be motivated clearly.
- **Results:** Images from Ex 2.6, Ex 2.7 and 2.20 as well as accuracy values for Ex 2.8 and Ex 2.9, and precision and recall values for Ex 2.18 and Ex 2.19. Include the images in the same folder as your code. Write the precision and recall values in a separate text file or in the ping-pong submission comment.