COMPSCI 527 — Homework 5
Due on November 3, 2016

Please refer to homework 1 and the class mechanics web page for homework policies and formatting/submission instructions. The submission checklist for this assignment is as follows:

`hw5.pdf`, `classify.m`, `nn_batch.m`, `nn_conv.m`, `nn_dense.m`, `nn_flatten.m`, `nn_maxpool.m`, `nn_run.m`, `nn_softmax.m`, `ReLU.m`

Keep in mind that your code must also show up in `hw5.pdf` .

Convolutional Neural Nets

Download and unzip the file that comes with this assignment and change your MATLAB working directory to where the files reside. If you then type

```
load data
```

the following variables will show up in your environment:

```
img1, img2, img3, img4, net, test.
```

We will see the `img1`,...,`img4` variables later on. The cell array `net` specifies the structure and parameters of a ten-layer Convolutional Neural Net (CNN) that recognizes a digit from the set $\mathcal{L} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$ in each image presented as the input to the CNN.[1] Training a net this size is nontrivial, so Yilun, who designed most of this assignment, pre-trained the net for you using 60,000 images derived from the MNIST database we used in a previous assignment. The variable `net` contains all the weights that result from pre-training.

To make the results more interesting, the standard 28×28 MNIST images have been embedded at random positions into larger, 50×50 images with a black background, both for training and for testing. So the input to the CNN has dimension $50^2 = 2500$.

The variable `test` contains 10,000 images and labels for a test set that is separate from the training set. You may want to display some of the images in `test.image` to see what they look like. Note that `test.image` has size `[50, 50, 1, 10000]`. More generally, a set of n images of width w and height h with c channels is represented by an $h \times w \times c \times n$ array in this assignment. Since the MNIST images are gray level, they have only one channel. Because of this, we could have stored them in an array of size `[50, 50, 10000]`. However, outputs from the convolutional layers of the net have multiple channels, and we want to keep our representation consistent for easier programming. A single image is still an $h \times w \times c$ array if it has multiple channels, and an $h \times w$ array otherwise. This is because MATLAB automatically strips any trailing singleton dimensions from any array. For instance,

```
>> size(rand(50, 50, 1))
ans =
    50    50
```

and the trailing `1` has disappeared.

Implementing a CNN

In this section, you will write the MATLAB functions needed to implement a CNN. Unless otherwise specified, use a stride of 1 everywhere.

In all of your answers, you are allowed to iterate over anything you like (unless otherwise stated) but *not* over the pixels of an image. Include each function below in your PDF, and hand in each as a separate `.m` file as well. No sample outputs are required in this section.

1. Write a MATLAB function with header

```
function J = ReLU(I)
```

that applies the Rectified Linear Unit function to each entry of its input array `I`. The output array `J` has the same size as `I`.

[1] The unusual ordering of the elements in $\mathcal{L}$, with the zero at the end, was chosen to simplify MATLAB programming.

2. Write a MATLAB function with header

```
function J = nn_conv(I, layer)
```

where `I` is an input array of size $h \times w \times c_i$ and `layer` is a layer of a CNN (for instance, `net{1}` is a layer). The function takes an array `layer.V` of size $h_k \times w_k \times c_i \times c_o$ that represents c_o kernels, and a column vector `layer.b` of c_o biases, and computes an output array `J`. Slice `J(:, :, c)` of `J` is the result of convolving `I` with the three-dimensional kernel `layer.V(:, :, :, c)`, adding bias `b(c)`, and passing the result through `ReLU`. The convolution should be performed with the MATLAB function `convn` with the `valid` option, so that `J` has size $(h - h_k + 1) \times (w - w_k + 1) \times c_o$. As you write your code, you may assume that the given `layer` contains the appropriate fields.

3. Write a MATLAB function with header

```
function J = nn_maxpool(I, layer)
```

that takes an input array `I` of size $h \times w \times c$ and a CNN layer `layer`, and returns an output array `J` of size $(h/d) \times (w/d) \times c$ where the positive integer d is found in `layer.d`. Each entry of `J` is the maximum in a $d \times d$ window of `I`. All the windows together cover the image fully, are adjacent to each other, and do not overlap each other (so that the stride is d). You may assume that h and w are divisible by d, although a check for this in the function would be safer.

4. Write a MATLAB function with header

```
function J = nn_flatten(I, ~)
```

that takes an input array `I` of any size and a second, unused argument and returns a column vector `J` that results from flattening `I` with the MATLAB colon operator `(:)`. The second argument is included because the function will be called with two arguments, for uniformity with the other layer functions you are implementing. However, since the argument is not used, it is not given a name. The '`~`' symbol is MATLAB's convention for a name placeholder.

5. Write a MATLAB function with header

```
function J = nn_dense(I, layer)
```

that takes an input array `I` of size $n \times 1$ and a CNN layer `layer`, and returns an output array `J` of size $m \times 1$ that results from applying a dense (*a.k.a.* fully-connected) layer with ReLU nonlinearity to `I`. The $m \times n$ weight matrix of the layer is `layer.V` and the $m \times 1$ bias vector is `layer.b`.

6. Write a MATLAB function with header

```
function J = nn_softmax(I, layer)
```

that implements a soft-max layer. This layer has the same format as `nn_dense`, so that `layer` will have fields `layer.V` and `layer.b`. However, the ReLU function is now replaced by the soft-max function described in the notes. The length m of the output is 10, the number of digits in set $\mathcal{L}$.

It is possible for a CNN layer to return all zeros. If the (10-dimensional) input to the soft-max function is all zeros, just return the input itself.

7. Write a MATLAB function with header

```
function I = nn_run(I, net)
```

that takes a single image `I` of size $h \times w \times c$ and a cell array `net` that specifies a CNN, and outputs the result of running `net` on `I`. Each entry `net{k}` of the cell array is a data structure that contains field `net{k}.function`, plus additional fields that vary by layer. The field `net{k}.function` is a handle to the function that implements the layer, and refers to one of the `nn_*` functions you implemented above. You may assume that each layer contains the appropriate fields, and you need not explicitly check for bad input arguments.

8. Write a MATLAB function with header

```
function y = nn_batch(imgs, net)
```

that takes an array `imgs` of n images and a cell array `net` that specifies a CNN, and outputs a $m \times n$ array containing the results of running `net` on each image in `imgs`. The array `imgs` has size $h \times w \times c \times n$. You may assume that the last layer of `net` is a soft-max layer, so you can get the net's output size m as `length(net{end}.b)`. [Hint: Feel free to use a `for` loop.]

Note: In situations in which speed is critical, every function you implemented is typically implemented on a GPU, including the part in which predictions for multiple images in a batch are computed in parallel. Thus, `nn_batch` simulates this behavior on a single-threaded CPU machine.

Testing the CNN

As a sanity check, run your CNN on an image of all zeros as follows:

```
load data
y = nn_run(zeros(50), net);
```

The vector `y` should contain the following entries (as a column vector; here they are printed in a row to save space):

$$\begin{bmatrix} 0.1054 & 0.0987 & 0.0873 & 0.0959 & 0.1094 & 0.0843 & 0.1006 & 0.0997 & 0.1171 & 0.1016 \end{bmatrix}$$

As you see, the net is quite uncertain as to which digit the input image contains, as appropriate.

9. Use the function `nn_batch` to write a MATLAB function with header

```
function [y, softmax] = classify(imgs, net)
```

that returns an $n \times 1$ vector `y` with the labels predicted by the CNN `net` for the n images in the $h \times w \times c \times n$ array `imgs`. If all ten entries in the output of the CNN for a particular image are zero, the corresponding entry in `y` should be set to `NaN`. The function also returns a $10 \times n$ matrix `softmax` equal to the output from `nn_batch`.

Then compute the *confusion matrix* obtained on the 10,000 images in `test.image`, using the corresponding true labels in `test.label` and your function `classify`.

The confusion matrix is a 10×10 matrix C whose entry $C(i, j)$ is the number of test samples for which the true label is $i\%10$ and the label returned by the CNN is $j\%10$ (make sure you do not confuse rows and columns). The percent symbol denotes the 'modulo' operator, so $i\%10$ is just i for all digits, except that $10\%10 = 0$ (so zero is the last digit).

Also report the test error rate for the classifier as a percentage. You may compute that either from the confusion matrix or directly from the classification results.

Hand in your code (both in the PDF file and as a separate MATLAB file) for `classify.m`. Also provide (in the PDF only) the snippet of code you used to compute error rate and confusion matrix. Report your test error rate and confusion matrix in the PDF file.

10. How many images in `test.image` have a true label of 4 and are misclassified as 9? Make a figure with all these images. Also show in your PDF file the code snippet you used to find the indices for these images. Finally state whether you think that the classifier was confident in its mistake for these images, and support your claim.

[Continued on the next page]

What is the CNN Thinking?

This part of the assignment is somewhat more open-ended than the rest, in that it asks you to experiment with the given CNN and report your observations and conclusions using some ideas from `http://cs231n.github.io/understanding-cnn/`. If you are interested, you can explore other visualization methods on that website.

To help you with this part, the function in `mask.m` is provided with this assignment. It takes a single image as input and yields an output array of as many images as the input has pixels. The images in the output array are copies of the input image, except that each output image has a different square masked out.

The point of playing with these masked images is to figure out what parts of an image the CNN is relying on in order to estimate what digit the image represents. We can also use the masked images to "confuse" the CNN by hiding parts of a digit, or parts of an image that contains multiple digits. This provides some insight into what the CNN is "thinking."

More specifically, `mask` takes an image `I` of size $h \times w \times c$ and an odd number N, and outputs an array of images `imgs` of size $h \times w \times c \times (h \cdot w)$. Slice `imgs(:,:,:,n)` is a copy of `I` but with a 0-valued mask of size $N \times N$ centered at the `n`-th pixel of the image, where pixels are numbered in the standard MATLAB order. This is the order you would obtain if you rearranged the pixels in `I` into the vector `I(:)`. The mask is allowed to exceed the image boundaries, so there is a mask at every pixel position, and masks close to the pixel boundaries are smaller. You may want to run

```
imgs = mask(img1, 29);
```

and look at some of the resulting images.

11. For each of `img1`, `img2`, `img3`, and `img4` (provided), create a 50×50 image P whose pixel (i, j) shows the value that the CNN specified by `net` outputs for the true label when a mask of size 29×29 and centered on (i, j) is applied to the image. For instance, the true label for `img1` is 7. Then, $P(20, 15)$ is the value of y_7, the seventh entry of the CNN output when the input is a copy of `img1` with the mask centered at pixel $(20, 15)$. You would expect y_7 to be close to 1 for images that contain a 7, and lower for other images.

Use 7, 2, 7, and 7 as true labels for `img1`, `img2`, `img3`, and `img4` respectively.

What pattern do you observe? Explain your observations.

To answer these questions, show four figures (one for each image). Each figure has the original image on the left and the corresponding P image on the right. To display P, use black for value 0 and white for value 1. You can achieve this by saying `imagesc(P); colormap gray`. Also enclose the display of P in a border to delineate the image region from white background. You can do this by saying the following after displaying P:

```
axis on;
set(gca, 'XTick', [], 'YTick', []);
```

This surrounds the image with a boundary with no tick marks or axis labels.

Hand in your four figures, the function or script you used to produce P (only in the PDF file, and for one image), and your answers to the questions above. [Hint: use your function `nn_batch` to create the values in P. You need no additional `for` loops to answer this question.]

12. Rather than looking only at the value for the true label, you will now look at the entire output from the CNN `net` for all the masked images you produced. For each of `img1`, `img2`, `img3`, and `img4`, create a 10×2500 matrix Y whose column $Y(:, n)$ is the soft-max output of the CNN for the n-th image in the array `imgs` (same as for the previous problem).

Show your four images Y as gray-level images (`colormap gray`), and state which is which. Include a color bar. Hand in your four figures on a single page in the PDF. Also include the function or script you used to produce Y (only in the PDF file, and for one image). No comments are needed for this problem.

13. For `img4`, display the masked image that score highest for each of the ten class labels in set $\mathcal{L}$. Give a title to each masked image showing the label for which it scores highest. Thus, your solution for this problem will have 10 images, each with a title containing a single digit.