Problem Set 7: Deep Learning
Due May 9, 2017

You may run into problems installing MatConvNet on your laptop, so please start this assignment early so the TA can help you if you have trouble.

There will be equal emphasis on the write-up and the code for this assignment. Please read through the assignment carefully and address all questions.

First, you need to download the data. We will use a subset of the CelebA dataset, which consists of face images labeled with 40 binary attributes (blond hair, female, etc.). The data can be downloaded from www.cs.umd.edu/~djacobs/CMSC426/Spring17/CelebA.zip. The dataset contains 20,000 64x64 images, 10,000 for training, and 5,000 each for validation and testing. CelebA/images contains the data, and CelebA/Files contains the train, validation, and test split files.

1. Download and install MatConvNet. Run MNIST. Classify your own hand-written digit.

a. Download and install MatConvNet from http://www.vlfeat.org/matconvnet. If you have a Windows machine, you may run into issues, so **PLEASE START EARLY** so the TA can help you.

b. Once you have installed MatConvNet, you should be able to run the MNIST example provided in <MatConvNetDir>/examples/mnist/cnn_mnist.m. Run this example. It trains a network for 20 epochs (complete runs through the training dataset). Notice the plots created after each epoch. The objective function, top 1 error, and top 5 error are plotted. Since there are 10 digits, cnn_mnist.m performs 10-way classification.
MNIST is a dataset of handwritten digits (0-9). If you're interested in the dataset, you can find more information here: http://yann.lecun.com/exdb/mnist.

c. Your trained net is now in <MatConvNetDir>/data/mnist-baseline-simplenn/net-epoch-20.mat. To test your trained network, write three different hand-written digits on a piece of paper. Take a picture of each digit, and save them. Convert the images to grayscale, resize them to 28x28, and convert them to `single` type. Also subtract the mean from them.

```
im = imread('nine.jpg');
im = rgb2gray(im);
im = single(im);
im = imresize(im, [28 28]);
im = im - mean(im(:));
```

Load your trained network, and remove the final softmax layer.
To load the network: `trainedNet = load('data/mnist-baseline-simplenn/net-epoch-20.mat');`

After loading the network, `trainedNet.net` contains the network weights. `trainedNet.net.layers(end)` contains the softmax layer. So, to remove the softmax layer simply delete the last layer: `trainedNet.net.layers(end) = [];`

```
%to get probabilities
trainedNet.net.layers{end+1} = struct('name', 'prob',
'type', 'softmax') ;
res = vlsimplenn(trainedNet.net, im);
```

`res(end).x` contains the score for each class with `res(end).x(1)` corresponding to 0 and `res(end).x(10)` corresponding to 9. The predicted class will be the one with the largest value. For my image (nine.jpg), `res(end).x(10)` has the largest value.

Don't worry if you do not get the correct classes for your digits. Think about why this may be happening. (Hint: look at the MNIST data (http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf), and compare it with your hand-written digits). Provide your images and results in your write-up. Discuss any unexpected results.

2. **Run an SVM classifier for gender recognition on CelebA dataset.**
Before using a deep convolutional network as in question 3, we will first try to tackle the task of gender recognition in a more conventional way. Specifically, we will extract some hand-designed features for each image and then train a binary classifier using these features.
Image features are some numerical representations of an image, which encode useful information for statistical analysis. Here we use a simple hand-designed feature, *Local Binary Pattern* (LBP) feature, to represent each image. LBP can capture the texture information in an image and is widely used as a face descriptor. To classify the LBP features into different classes, we use the well-known *Support Vector Machine* (SVM) classifier. Similar to the linear classifiers introduced in class, SVM tries to find a line (or hyperplane) that separates the data samples from different classes.
   a. **Data preparation.** Write a function with signature

   ```
   [X_train, Y_train, X_val, Y_val, X_test, Y_test] =
                        load_data()
   ```

   for loading train, validation, and test files from Data/CelebA/Files. `X_train, X_val, X_test` contain image data from training, validation and testing set, respectively. `Y_train, Y_val, Y_test` are corresponding gender labels. `X_train` is a 3-D matrix with size `(N, H, W)`, where `N` is the number of training images, `H=W=64` is the size of the images. `Y_train` is a 1-D vector with length `N`.
   To read the train, validation and test files, you can use the function `textread()` as follows.

   ```
   [name, label] = textread('celebA_gender_train.txt', '%s
                        %d');
   ```

   It will return a cell 'name' with file names in it and a vector 'label' with labels in it. Then you can create a matrix `X_train` with size `(N, H, W)` and loop for each element in the cell 'name' to read the image and put it into the matrix `X_train`. You can use the function `imread()` to load images, for example,

```
I = imread(['../files/', name{i}]);
```

Note that you need to use the braces {} to index a cell. Also, the image you load may be a color image with 3 channels (RGB). You need to convert it to grayscale image before putting it into your data matrix. You can use the function `rgb2gray()` here.

b. **Model training and testing.** We have provided you with Matlab files: svm_gender.m, LBP.m and getmapping.m. The files are used to train and test an SVM classifier and need to go in the same directory as your function *load_data()*. After loading the data, run the function svm_gender ([Acc, model] = svm_gender(X_train, Y_train, X_val, Y_val, X_test, Y_test);) and report the performance you got on the training and testing set, respectively.

c. **Challenge Problem: Regularization.** In the file svm_gender.m, we use the function *fitclinear* to train an SVM classifier. This function has an optional argument 'Lambda' which controls the strength of regularization (see the documentation of *fitclinear).* Set different lambda values and see how they will affect the training and testing performance. Report your results and briefly discuss your conclusion.

3. **Train a network to recognize gender.**

We have provided you with the following Matlab files: cnn_gender.m, cnn_gender_init.m, cnn_gender_deploy.m, and cnn_gender_setup_data.m. These files need to go in <MatConvNetDir>/examples/426. We also provide you with cnn_test.m which needs to go in <MatConvNetDir>/examples. The CelebA dataset you downloaded earlier needs to go in <MatConvNetDir>/data/. Copy getImageStats.m and getImageBatch.m from <MatConvNetDir>/examples/imagenet to <MatConvNetDir>/examples/426.

You need to specify the network architecture in cnn_gender_init.m. Your architecture should be written in `function net = orig(net, opts)`.

Here is some sample code from <MatConvNetDir>/examples/imagenet/cnn_imagenet_init.m. We have added some annotations (in italics) so you can better understand how to create your network.

```
net.layers = {} ;
```

This initializes the layers of network. At this point the network has no layers.

```
net = add_block(net, opts, '1', 11, 11, 3, 96, 4, 0) ;
```

This adds a few layers to the network. The `add_block` function adds a convolution layer, a batch normalization layer, and a ReLU layer. See the comments in cnn_gender_init.m for more details on `add_block`.

```
net = add_norm(net, opts, '1') ;
```

This adds a normalization layer at the end (top) of the network (after the ReLU from `add_block`). See the comments in cnn_gender_init.m for more information about `add_norm`.

```
net.layers{end+1} = struct('type', 'pool', 'name', 'pool1', ...
                           'method', 'max', ...
```

```
                                'pool', [3 3], ...
                                'stride', 2, ...
                                'pad', 0) ;
```

This specifies a new layer at the end (top) of the current network. This layer is a Pooling layer, this is specified after the `'type'` parameter. Each layer has a name, and this one is given the name `'pool1'` since it is the first pooling layer. The type of pooling must be specified, and in this layer max pooling is used. Max pooling chooses the maximum value in a window and discards the rest. The method (or type of pooling) is specified by `'max'`. Here the window is 3x3, which is specified by `'pool', [3 3]`. And this pooling operation is applied every 2 pixels, which is specified by `'stride', 2`.

In cnn_imagenet_init.m in the `alexnet` function, there are 5 convolution layers, and then a few fully connected layers. Fully connected layers are specified like convolution layers in MatConvNet, where the convolutions are the same size as the input.

```
net = add_block(net, opts, '6', 6, 6, 256, 4096, 1, 0) ;
```

This specifies the first Fully Connected (FC) layer in the `alexnet` function. The input to this layer is 6x6x256 (256 6x6 feature maps). By convolving this input with 4096 filters of size 6x6x256, we connect each input neuron to each output neuron. So when adding a FC layer, the convolution size must be the same as the input size, and the output size must be the number of neurons in the fully connected layer.

a. Specify the following architecture for gender recognition:

Conv1: Convolution Layer 11x11 Filters with 16 channels, stride=1, pad=0
    Batch Norm (This is a part of add_block)
    ReLU Layer (This is a part of add_block)
    Normalization Layer
    Max Pooling Layer 3x3, stride 2, pad 0
Conv2: Convolution Layer 5x5 Filters with 32 channels, stride=1, pad=2.
    Batch Norm
    ReLU Layer
    Normalization Layer
    Max Pooling Layer 3x3, stride 2, pad 0
Conv3: Convolution Layer 3x3 Filters with 64 channels, stride=1, pad=1.
    Batch Norm
    ReLU Layer
Conv4: Convolution Layer 3x3 Filters with 64 channels, stride=1, pad=1.
    Batch Norm
    ReLU Layer
Conv5: Convolution Layer 3x3 Filters with 64 channels, stride=1, pad=1.
    Batch Norm
    ReLU Layer
    Max Pooling Layer 3x3, stride 2, pad 0
FC6: Fully Connected Layer with 64 units, stride=1, pad=0.
    Batch Norm
    ReLU Layer

Dropout 50% (Note that the default dropout rate is 50% so this does not need to be specified in your code. See `add_dropout` in cnn_gender_init.m)

FC7: Fully Connected Layer with 64 units, stride=1, pad=0.

Batch Norm

ReLU Layer

Dropout 50%

FC8_gender: Fully Connected Layer with 2 units, stride=1, pad=0

PLEASE take advantage of `add_block`, and `add_norm` functions in cnn_gender_init.m. Note that `add_block` adds a convolution layer, a batch normalization layer, and a ReLU layer.

b. Train this network by running cnn_gender.m. This should take about 10 minutes to train on your laptops.
cnn_gender_init loads train, validation, and test files from Data/CelebA/Files (1 is male, 2 is female). This will run for 4 epochs. Looking at cnn_gender.m, you will see that first `cnn_train` is called and then `cnn_test`. `cnn_train` trains the network and produces the training and validation errors. `cnn_test` loads the test data and evaluates the trained network on this data. When running cnn_gender.m, you will see many things printed. First, the network architecture is printed, then the number of parameters are printed, as well as the amount of memory required to store the data during training. Then for each iteration, the training error is printed, and after each epoch, the current model is tested on the validation data, printing the error for that as well. The only error we care about is the `top1err`. The `top5err` will always be 0, since there are only two classes. This is repeated for 4 epochs. Your final model will be saved in <MatConvNet>/data/gender-orig-bnorm-simplenn/net-epoch-4.mat. `cnn_test` is then called. This simply loads the test data and evaluates the final model on this data. Again, the network architecture is printed along with the parameter and data memory information. Though the errors will say *`val: epoch 04:`* the trained network is being evaluated on the test set. The results of training are being saved after each epoch in <MatConvNetDir>/data/gender-orig-bnorm-simplenn. Every time you restart training, it will start from the most recently saved epoch. In order to train from scratch, you must delete the directory <MatConvNetDir>/data/gender-orig-bnorm-simplenn.
How is the number of epochs being specified? What are the different learning rates? When is the learning rate changed? What is your final validation error? Your test error?
After epoch 1, the training error should be around 0.25. At the end of training, the training error should be around 0.08, the validation error should be around 0.06 and the test error should be around 0.07.

c. Download three images from the Internet, of some public figures of your choosing. Resize the images to 64x64 and use your trained network to classify the images. Remember 1 is male and 2 is female. Submit your images as well as your results along with a discussion.

```
trainedNet = load('data/gender-orig-bnorm-simplenn/net-
epoch-4.mat');
```

```
%remove dropout, softmax, batch norm etc.
trainedNet.net = cnn_gender_deploy(trainedNet.net);
%load image and compute res as in problem 1
...
```

4. **Shrink the network from problem 3.**
   Starting with the network from problem 3, "shrink" it as much as possible while still achieving a maximum error of 0.1 on the test set. By "shrink" we mean reduce the number of parameters. This can be done by using fewer layers, fewer convolutional filters, more pooling to reduce the image size, etc. The network from problem 3 has 220,000 parameters (the number of parameters is printed when training starts). Feel free to do this however you like. Specify the network architecture in cnn_gender_init.m and please name the architecture 'mini'. Note that to run this 'mini' architecture you will need to change `opts.modelType = 'orig' ;` in cnn_gender.m and `opts.model = 'orig' ;` in cnn_gender_init.m. Do not change the number of epochs or the learning rate.
   For 50% credit, reduce the number of parameters to below 100,000.
   For 75% credit, reduce the number of parameters to below 50,000.
   For 100% credit, reduce the number of parameters to below 10,000.
   The student who submits the network with the fewest number of parameters will receive **10 BONUS POINTS** on this assignment.
   Provide your final architecture in your write-up and show how the number of parameters is calculated.

5. **<u>Challenge Problem</u>: Train a multi-task network to recognize gender and smiling at the same time. Please create a new set of files (cnn_gs*) for this problem.**
   Train a network that recognizes both gender and smiling. Start with your mini network from problem 4. Do this in two ways (specifying two different architectures):
   1. By replacing the final layer for gender with a final layer which has 4 units (1=male not smiling, 2=female not smiling, 3=male smiling, 4=female smiling).
   2. By connecting FC7 to two FC8s, one for gender and one for smiling. (This requires changing vl_simplenn.m).

Submission: Please submit all Matlab files along with a README so that the TA can run your code. For the CNN problems, the TA should be able to run your code by adding it to their <MatConvNet> directory. Submit your trained models for each problem along with your code.