HOMEWORK #3

ECBM E6040, Professor Aurel A. Lazar

Deadline: 11:59PM, April 14th, 2016

INSTRUCTIONS: This homework contains two programming assignments. Submission for this homework will be via bitbucket repositories created for each student and should contain the following

- All figures and discussions; document all parameters you used in the IPython notebook file, `hw3.ipynb`, which is already included in the homework 3 repository.
- Commit and push all the changes you made to the skeleton code in the Python files, `hw3a.py` and `hw3b.py`.

Programming

As the semester progresses, we are shifting our focus more and more towards programming.

In this homework, you will empirically study various regularization methods for neural networks, and experiment with different convolutional neural network (CNN) configurations. You should start by going through the Deep Learning Tutorials Project, especially, LeNet. The source code provided in the Homework 3 repository is excerpted from `logistic_sgd.py`, `mlp.py`, and `convolutional_mlp.py`.

As in the previous homework, you will be using the same street view house numbers (SVHN) dataset [1]. A recent ivestigation has achieved superior classification results on the SVHN dataset with above 95% accuracy (by using CNN with some modifications) [2].

Instead of reproducing the superior testing accuracy, your task is to explore the CNN framework from various points of view.

As in the previous homework, a python routine called `load_data` is provided to you for downloading and preprocessing the dataset. You should use it, unless you have absolute reason not to. The first time you call `load_data`, it will take you some time to download the dataset (about 180 MB). If you already have the dataset on the EC2 volume, you should simply reuse it. Please be careful **NOT TO** commit the dataset files into the repository. In addition to `load_data`, you are provided with various skeleton functions.

Note that all the results, figures, and parameters should be placed inside the IPython notebook file `hw3.ipynb`.

PROBLEM a (50 points)

In this problem, you are asked to empirically test several regularization methods for neural networks, which are discussed in Chapter 7 of the textbook. To better see the effect of regularization, you will be using a smaller training dataset down-sampled from the original SVHN dataset (generated by `load_data` with an additional input argument `ds_rate`). The testing dataset remains the same.

You will start by training a neural network model without any regularization, except optionally with L^1 or L^2 regularization. The testing result of this model serves as a baseline for comparison against different regularization methods. If you do use L^1 or L^2 regularization for the baseline model, you should also include them with the same parameters for other models with different regularization methods.

For neural network, you could use either MLP or CNN (from **Problem b**). A `myMLP` class has been provided to you.

i Implement an MLP or a CNN, and train it with the smaller dataset. Then, train the same model again with the complete dataset. Document your choice of parameters, and report the testing accuracy in both cases. For MLP, you could reuse any sets of parameters that you implemented in the previous homework.

ii Noise injection is a common method for regularization when the dataset is limited. For each example in the smaller dataset, generate several copies and and add a randomly sampled noise vector to each of them. A skeleton function `test_noise_inject_at_input` is provided to you. Train the same model from **(i)** with the new noisy dataset. Repeat the same procedure with another level of noise. Document your choice of noise, discuss the testing accuracy, and compare the result with those in **(i)**.

iii Another way of noise injection is to inject it into the weights of affine transformation between layers. A skeleton function `test_noise_inject_at_weight` is provided to you. Train the same model from **(i)** with the smaller dataset, but inject noise into the weights after each of the updates (More specifically, you need to modify the `updates` routine in the skeleton code). Document your choice of noise, discuss the testing accuracy, and compare the result with those in **(i)**.

iv Data augmentation is another way to overcome the limitation of small datasets.

It has been a particularly effective method for object recognition. You are asked to synthesize new data to augment the smaller dataset, and then train the model with the synthesized dataset. To do so, you create 4 new examples for each of the examples in the dataset by translating the example by 1 pixel along four different directions, and padding zeros to the missing part. If you have other ideas about data augmentation, you could implement them instead of using the one described here. A skeleton function `test_data_augmentation` is provided to you. Train the same model from **(i)** with the new dataset. Document your choice of noise, discuss the testing accuracy, and compare the result with those in **(i)**.

v Recent work has shown that one can fool a neural network with adversarial examples [4]. Such phenomenon is also discussed in section 7.13 in the textbook. You are asked to test and reproduce this phenomenon. To do so, take any models you trained in previous questions, and compute the gradient of the cost function with respect to the *input* (Please review section 7.13 of the textbook). Then, create an adversarial example by first picking an example with correct classification and adding an imperceptibly small vector to the input whose elements are equal to the sign of the elements of the gradient. Use the trained model to classify this adversarial example. Can you fool the model? Discuss your results, and plot the original input along with the adversarial example and a bar plot of the class-specific probabilities (output of the neural network) for the original input and the adversarial example.

PROBLEM b (50 points)

In this problem, you will experiment with convolutional neural networks. The CNN model is similar to LeNet from the Deep Learning tutorial, with the exception that it handles images with 3 color channels, whereas LeNet targets grey-scaled image.

i Implement an CNN with 2 convolution hidden layers for multi-channel inputs. First, go through the skeleton function `test_lenet()` in `hw3b.py`, and finish the missing part. After finishing the function, experiment with parameters, in particular, the number of filters in hidden layers. Document at least three different sets of parameters explicitly, and discuss the accuracy of your test results.

ii Implement a multi-stage CNN, as shown in Figure.1. First, go through the skeleton function `test_convnet()` in `hw3b.py`, and finish the missing part. After finishing the function, experiment with all the parameters, in particular, the number of filters in hidden layers as well as the shape of filters. Document at least three different sets of parameters explicitly, and discuss the accuracy of your test results.

iii The multi-stage CNN model you implemented in the previous question has a

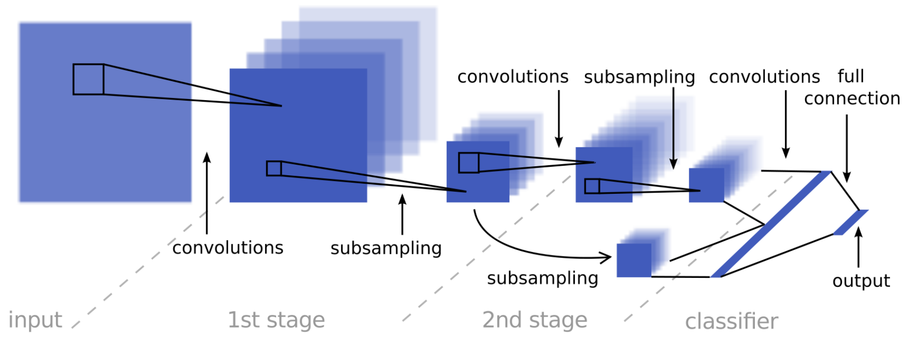

Figure 1: Excerpted from [2]. A 2-stage CNN architecture where multi-stage features (MS) are fed into a 2-layer classifier. The first stage features are branched out, down-sampled again and then concatenated to second stage features.

nonstandard feed-forward structure, but the THEANO package is still able to compute the gradient of the cost function with respect to different parameters via the back-propagation algorithm. Discuss why the back-propagation algorithm can be applied to this model. You might want to review the section about the back-propagation algorithm in the textbook.

iv The state-of-the-art neural networks for object recognition usually implement a CNN in cascade with a MLP[3]. Implement a network with two convolution layers in cascade with a MLP with 2 hidden layers. Train the model, and document the testing accuracy. How does this model perform compared to your implementation of the MLP with 4 hidden layers in Homework 2?

BONUS PROBLEM (25 points)

i A nice advantage of CNNs that separates it from other machine learning models is that it is capable of learning features all the way from pixels to the classifier, whereas other methods usually require multiple hand-crafted features. You are asked to compare the performance of a CNN with hand-picked features versus one with learned features. Specifically, use a CNN from the first question with 3 filter sets at the input layer (each set has 3 filters for each color channels), and train the whole network. Then, use the same CNN model, but replace the 3 filter sets of the input layer with your own design (ex., Gaussian filters). For each filter set, you can use the same filter for each color channel. Train the model without updating the designed filters. Document and compare the testing accuracy of both models, and plot the filters learned via training and the filters you designed.

ii Another advantage of a CNN is that it greatly reduces the number of parameters in the network. The fewer parameters imply that a CNN can be usually trained in shorter time than a MLP with same amount of neurons and layers. You are asked to compare a CNN and an MLP. In particular, implement a CNN with two convolution hidden layers, and a fully-connected MLP with three hidden layers (Note that in a CNN, convolution hidden layers are followed by a fully connected perceptron and an output layer). For each layer of the MLP, use the same number of neurons (activation functions) as the corresponding layer in the CNN. You can reuse the CNN from **(i)**. Document the number of parameters of both models (total number of entries in all filters for CNN, and total number of entries in all weight matrices for MLP). Discuss the run-time for both models, and the testing accuracy.

NEED HELP:

If you have any questions you are advised to use Piazza forum which is accessible through Canvas system.

GOOD LUCK!

References

[1] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.*

[2] Pierre Sermanet, Soumith Chintala, Yann LeCun, "Convolutional Neural Networks Applied to House Numbers Digit Classification," *ICPR 2012*

[3] Tara N. Sainath, Abdel-rahman Mohamed, Brian Kingsbury, Bhuvana Ramabhadran, "Deep convolutional neural networks for LVCSR," *ICASPP 2013*

[4] Anh Nguyen, Jason Yosinski, Jeff Clune, "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images," *IEEE CVPR 2015.*