COMP6321: Machine Learning

Assignment 4

Due: 11:59 PM (EST), November 14, 2024, submit on Moodle.

Include your name and student ID!

- Submit your write-up in PDF (we do not accept hand-written submissions) and all source code as an ipynb file (with proper documentation). Your writeup should include written answers to all questions, including the reported results and plots of coding questions in a single PDF file. Please save the output of each cell or your coding questions may NOT be graded. Write a script for each programming exercise so the TAs can easily run and verify your results. Make sure your code runs!
- Text in square brackets are hints that can be ignored.

NOTE For all the exercises, you are only allowed to use the basic Python, Pytorch, Numpy, and matplotlib (or other libraries for plotting), unless specified otherwise.

Exercise 1: CNN Implementation (12 pts)

Note: Please mention your Python version (and maybe the version of all other packages).

In this exercise, you are going to run some experiments involving CNNs. You need to know Python and install the following libraries: Pytorch, matplotlib and all their dependencies. You can find detailed instructions and tutorials for each of these libraries on the respective websites.

For all experiments, running on CPU is sufficient. You do not need to run the code on GPUs. Before start, we suggest you review what we learned about each layer in CNN, and read at least this tutorial.

Note: for each question you should only apply the modification asked in that question. Do not incrementally add modifications as you move toward further questions.

1. (4 pts) Implement and train a VGG11 net on the MNIST dataset. VGG11 was an earlier version of VGG16 and can be found as model A in Table 1 of this paper, whose Section 2.1 also gives you all the details about each layer. The goal is to get the loss as close to 0 as possible. Note that our input dimension is different from the VGG paper. You need to resize each image in MNIST from its original size 28×28 to 32×32 [why?].

   For your convenience, we list the details of the VGG11 architecture here. The convolutional layers are denoted as `Conv(number of input channels, number of output channels, kernel size, stride, padding)`; the batch normalization layers are denoted as `BatchNorm(number of channels)`; the max-pooling layers are denoted as `MaxPool(kernel size, stride)`; the fully-connected layers are denoted as `Linear(number of input features, number of output features)`; the drop out layers are denoted as `Dropout(dropout ratio)`:

   ```
   - Conv(001, 064, 3, 1, 1) - BatchNorm(064) - ReLU - MaxPool(2, 2)
   - Conv(064, 128, 3, 1, 1) - BatchNorm(128) - ReLU - MaxPool(2, 2)
   - Conv(128, 256, 3, 1, 1) - BatchNorm(256) - ReLU
   - Conv(256, 256, 3, 1, 1) - BatchNorm(256) - ReLU - MaxPool(2, 2)
   - Conv(256, 512, 3, 1, 1) - BatchNorm(512) - ReLU
   - Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
   - Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU
   - Conv(512, 512, 3, 1, 1) - BatchNorm(512) - ReLU - MaxPool(2, 2)
   - Linear(0512, 4096) - ReLU - Dropout(0.5)
   - Linear(4096, 4096) - ReLU - Dropout(0.5)
   - Linear(4096, 10)
   ```

   You should use the following in your training process, unless specified otherwise: cross-entropy loss `torch.nn.CrossEntropyLoss`, and optimize using SGD optimizer with momentum=0.9.

   [This experiment will take up to 1 hour on a CPU, so please be cautious of your time. If this running time is not bearable, you may cut the training set to 1/10, so only have ∼600 images per class instead of the regular ∼6000.]

2. (0.5 pts) Once you've done the above, the next goal is to inspect the training process. Create the following plots:

   (a) test accuracy vs the number of epochs

   (b) training accuracy vs the number of epochs

   (c) test loss vs the number of epochs

   (d) training loss vs the number of epochs

   [Train the neural networks for 20 epochs. Although 20 epochs is not sufficient to reach convergence, it is sufficient to see the trend. You can reduce the number of epochs further if it is infeasible to train for 20 epochs.]

3. (2.5 pts) It is time to inspect the generalization properties of your final model. Flip and blur the test set images using any python library of your choice, and complete the following:

   (a) test accuracy vs type of flip. Try the following two types of flipping: flip each image from left to right, and from top to bottom. Report the test accuracy after each flip. What is the effect? You can read this doc to learn how to build a complex transformation pipeline. We suggest the following command for performing flipping:

   ```
   torchvision.transforms.RandomHorizontalFlip(p=1)
   torchvision.transforms.RandomVerticalFlip(p=1)
   ```

   (b) test accuracy vs Gaussian noise. Try adding standard Gaussian noise to each test image with variance 0.01, 0.1, 1 and report the test accuracies. What is the effect?

   For instance, you may apply a user-defined lambda as a new transform t which adds Gaussian noise with variance say 0.01:

   ```
   t = torchvision.transforms.Lambda(lambda x : x + 0.1*torch.randn_like(x))
   ```

4. (1.25 pts) Lastly, let us verify the effect of regularization. Retrain your model with data augmentation and test again. Report the test accuracy and explain what kind of data augmentation you use in retraining.

5. (1.25 pts) Train your CNN with AdaDelta and Adam. Plot the graphs for training and test accuracies vs the number of epochs for both optimizers. Compare the accuracies of the two optimizers with the SGD accuracies in Ex 1.2. Which optimizer performed better and why?

6. (1.25 pts) Replace ReLU activation with Sigmoid in your CNN and train it again. Plot the graphs for training and test accuracies vs the number of epochs. Compare it with your results in Ex 1.2. Do you see any change because of Sigmoid? If so, explain why.

7. (1.25 pts) Remove Dropout from your CNN and train it again. Plot the graphs for training and test accuracies vs the number of epochs. Compare it with your results in Ex 1.2. Did Dropout have any effect on your network performance? Explain.

Exercise 2: MLP Implementation (4 pts)

In this exercise, you are going to run some experiments involving multi-layer perceptrons (MLPs). You can implement them in Pytorch as well (see the instructions at the beginning of Exercise 1).

1. (2.5 pts) Implement a three-layer MLP (details below) and train it on the MNIST dataset. The details of the MLP architecture are:

   ```
   - Linear(784, 512) - ReLU
   - Linear(512, 512) - BatchNorm(512) - ReLU
   - Linear(512, 10)
   ```

You should use the cross-entropy loss `torch.nn.CrossEntropyLoss` at the end. Also, use the SGD optimizer with momentum=0.9.

[If you used 600 images per class in Exercise 1, then use the same images here.]

Note: You will need to flatten your images from 28×28 to 784×1 vectors, because the MLP expects a vector as an input.

2. (0.5 pts) After the training process, create the following plots:
   (a) test accuracy vs the number of epochs
   (b) training accuracy vs the number of epochs
   (c) test loss vs the number of epochs
   (d) training loss vs the number of epochs

   [Train the neural networks for 20 epochs. Although 20 epochs is not sufficient to reach convergence, it is sufficient to see the trend. You can reduce the number of epochs further if it is infeasible to train for 20 epochs.]

3. (1 pt) Compare the accuracies of the MLP and VGG11. Which network is better for MNIST classification and why?

4. Extra: 0.75 pts Add another layer in your MLP (see below) and train on the MNIST dataset again. The new MLP should look like this:

```
- Linear(784, 512) - ReLU
- Linear(512, 512) - BatchNorm(512) - ReLU
- Linear(512, 512) - BatchNorm(512) - ReLU
- Linear(512, 10)
```

   Plot the test accuracy and training accuracy vs the number of epochs, and compare it with the previous MLP. Which network performs better and why?