IDS594: Machine Learning w/ Python Application *Updated: 10/02/2018*

Homework #2

Instructor: Moontae Lee *Total Points: 100+30*

Policy

1. HW2 is due by 10/15/2018 11:59PM in Central Time. One submission per each group.
2. You are allowed to work individually or as a group of up to three students.
3. Having wider discussions is not prohibited. Put all the names of students beyond your group members. However individual students/groups must write their own solutions.
4. Put your write-up and results from the coding questions in a single pdf file. Compress R source codes into one zip file. Each student/group must submit only two files. (You will lose the points if the answers for coding questions are not included the pdf report)
5. If you would include some graphs, be sure to include the source codes together that were used to generate those figures. Every result must be reproducible!
6. Maximally leverage Piazza to benefit other students by your questions and answers. Try to be updated by checking notifications in both Piazza and the class webpage.
7. Late submissions will be penalized 20% per each late day. No assignment will be accepted more than 3 days after its due date. For HW2, it will be 10/18/2018 11:59pm

Problem 1: Neural Network Framework by PyTorch [25 points]

For working with neural networks, you are supposed to design a network with various layers relying on your intuition and understanding about the data. Then, after the forward pass, the backward pass initiates backpropagation, updating all the learnable parameters via autograd functionality. In this assignment, you are required to write the forward and the backward passes for various types of neural layers such as convolutional and pooling layers as well as vanilla fully-connected layers. Playing with CIFAR-10 object detection dataset,[1] the eventual goal is to design your own model that can mix all what you have learned so far toward maximizing the detection accuracy.

Concretely speaking, you are going to implement four different neural networks. **Softmax** is the simplest one without any hidden layer, which is essentially equivalent to simple multiclass classification. **TwoLayerNN** consists of one hidden layer. Feed-forward will go in and out the hidden layer via fully-connected interaction. **ConvNet** consists of convolutional hidden

[1] `https://www.cs.toronto.edu/~kriz/cifar.html`

layers and pooling step based on the convolution and pooling operations. **MyModel** will be your customized model which can be as complex as possible to achieve the best result on this dataset.

Prior to designing each parts of these four models, Problem 1 will guide you to understand the high-level structures and the program flow of this assignment. It first reads command-line arguments about model hyperparameters, and then read CIFAR-10 dataset. After it loads one specified model among the above four, it trains the model and measure both training and validation errors. Then it reports overall performance on test data.

Your first job is to fill up each of the TO-DO parts within *train.py*. The code includes a vast amount of comments for helping your detailed understanding. Try to read every single line in this file from the top to the bottom. Later once you implement the first softmax model, it will be great to revisit the *train.py*, adding many print functions various places and verifying what is really going on at each part.

(a) Is the training done per mini-batch (i.e., update the model parameters for each batch) or done per every single example (i.e., update stochastically)? What is the relation between epoch and batches?

(b) The first TO-DO is located before defining our training function. Similar to what we have learned in PyTorch examples in class, you are supposed to instanciate at least one optimizer among many predefined options in the *torch.optim* package. Be careful in receiving the model parameters. (Note: Learning Rate (lr) and Momentum (momentum)[2] must be plugged directly from what you have received in the command-line options rather than fixing as single values)

(c) Inside the train function, for each batch of training images, you should get the outputs by feeding the current batch of images to the model. Then you should explicitly compute the loss based on the criterion that is already selected as Cross Entropy.[3] Note also that given the loss, you should perform one backward pass, later updating the model parameters. (Note: Be careful to remove existing gradient information before running one backward pass)

Problem 2: Softmax (Multiclass) Classification [25 points]

Problem 1 explains the learning framework of this assignment. In this problem, you are going to implement a simple softmax classifier, understanding the overall processes with your simple neural network model without having any hidden layer.

Recall that PyTorch offers various building blocks in the **torch.nn** module. Typically, a PyTorch model is a subclass of this module, thereby gaining a multitude of features. Now the goal is to implement a softmax classifier by filling up the TO-DO sections of *models/softmax.py*.

[2]This is not covered in the class, but useful for state-of-the-art optimizers.

[3]This is the most popular loss function for multiclass classification as we have the least square loss function for linear regression.

Note that softmax regression is just a multiclass extension of logistic regression, which is already implemented as one block in **nn.module**, allowing you to access all of its parameters and sub-moduels through **.parameters()** and **.modules()** methods.

Once you implement it, you can train a model by executing *run_softmax.sh*.[4] As a result, it will save the trained model into a *softmax.pt*, also producing *softmax.log* that contains all the training details. For convenience, two types of visualization codes are provided. The code *log-viz.ipynb* visualizes your *softmax.log*, whereas the code *filter-viz.ipynb* visualizes your model *softmax.pt*. While these codes are written as Jupyter Notebooks, but you can easily convert them into python scripts if necessary. Do not forget to adjust hyperparameters specified in *run_softmax.sh* multiple times to get reasonable performance.

(a) Recall we have seen **DynamicNet** example in the PyTorch lecture. To design your own building block, you were supposed to initialize your layers in the **__init__()**. Fill up the first TO-DO in this constructor. (Note: Your input layer must immediately connect to the output layer via fully-connected interaction.)

(b) Recall that sequential application of your modules to the given data realizes the actual forward pass. Fill up the second TO-DO in **forward()** method so that it will output the prediction values. (Hint: Whereas we dealt with only a single input in class, note that we are now dealing with a batch of input images at once).

(c) Use the code *log-visualize.ipynb* to generate the *loss-vs-# iterations* plot on the training data and the *accuracy-vs-# iterations* plot on the validation data. By varying the hyperparameters, pick the best model based on the validation results. Report your visualizations and the accuracy of your best model on the test data.

(d) Fill up the TO-DO section in *filter-visualize.ipynb* so that you can visualize the learned weight parameters between the input and the output layer. Save it into a *softmax_gridfilt.png*.

(e) Perform an error analysis. Give some examples of difficult images to predict the correct labels, and speculate why. Report if you have further exciting observations.

Problem 3: Two-layer Neural Network [25 points]

By now you should have a fairly good understanding about PyTorch and our framework from data loading to output visualization. Repeat the five questions in Problem 2 by implementing *models/twolayernn.py*. Note that there must be only one hidden layer, and both the input-to-hidden layer and the hidden-to-output layer must be fully-connected. Use the ReLU activation function for imposing non-linearity via the hidden layer.

You can run the experiment with the user-specified hyperparameters by executing *run_twolayernn.sh* this time. For the part (d), save a visualization of the learned weight parameters between the input and the hidden layer into a file *twolayernn_gridfilt.png*.

[4]If you are using Windows, try to create the similar shell script as *run_softmax.bat*

Problem 4: Convolutional Neural Network [25 points]

Repeat the five questions analogously to Problem 2 and 3 by implementing *models/convnet.py*. You should have just one convolutional layer where the kernel size means the width/height of a square convolution filter that you are to use. For doing the part (a), be careful in computing the right size of output dimensions after the convolution with a specified size of filter. The result of convolution must be fully-connected to the output layer. For the part (b), note that the results of convolution must be go through ReLU activation and then be max-pooled.

You can run the experiment with the user-specified hyperparameters by executing *run_convnet.sh* this time. For the part (d), save a visualization of the learned weight parameters between the input and the hidden layer into a file *convnet_gridfilt.png*.

Problem 5: Build Your Own Neural Network [30 points]

(Optional) Repeat the five questions analogously to Problem 2 and 3 by implementing your own model in *models/mymodels.py*. Ideally "conv-relu-conv-relu-pool-softmax" will be the best to try. Try to also vary the optimizer by suing AdaGrad, AdaDelta, and Adam for the Problem 1-(b). (Note: You can refer to the PyTorch manual for instanciating different optimzers)

You can run the experiment with the user-specified hyperparameters by executing *run_mymodel.sh* this time. For the part (d), save a visualization of the learned weight parameters between the input and the hidden layer into a file *mymodel_gridfilt.png*.

Your code submission must include:

- One code *train.py*
- Four codes *models/{softmax, twolayernn, convnext, mymodel}.py*
- Four scripts *run_{softmax, twolayernn, convnext, mymodel}.sh* with your finalized best hyperparamters.
- One notebook *filter-visualize.ipynb*.
- Four output *.log* files from each of your four models.
- Four output images *{softmax, twolayernn, convnext, mymodel}_gridfilt.png* with your best configuration.