ECE590-03: Homework #2
Construct, Train, and Optimize CNN Models

Hai Li

ECE Department, Duke University — September 2, 2020

Objectives

Homework #2 covers the contents of Lectures 05∼08. This assignment includes basic knowledge about CNNs, detailed instructions on how to setup a training pipeline for training image classifiers on the CIFAR-10 dataset, how to improve the training pipeline, and how to use advanced CNN architectures to improve the performance of image classifiers. In this assignment, you will gain hands-on experience training a neural network model on a real computer-vision dataset (i.e., CIFAR-10), while also learning techniques for improving the performance of your CNN model.

We encourage you to complete the Homework #2 on the JupyterLab server or Google CoLab since the model training will require the computing power of GPUs. Please refer to the **NumPy/PyTorch tutorial** slides on Sakai for the environment setup and instructions for NumPy/PyTorch utilities.

Warning: You are asked to complete the assignment independently.

This lab has **100** points plus **10** bonus points, yet your final score cannot exceed 100 points. The submission deadline will be **11:59pm, Thursday, September 17**. We provide a template named `lenet5-cifar10.ipynb` to start with, and you are asked to develop your own code based on this template. You will need to submit three independent files including: (1) *a self-contained report in PDF format* that provides answers to all the conceptual questions and clearly demonstrates all your lab results and observations, (2) `code.zip`, a zipped code file which contains 3 jupyter notebooks `lenet5-cifar10.ipynb`, `lenet5-cifar10-dev.ipynb`, and `resnet-cifar10.ipynb`, respectively from the three Labs; and (3) `predictions.csv`, your predicted label for each of the image in the CIFAR-10 testing dataset. **Note that 20 percent of the grade will be deducted if the submissions doesn't follow the above guidance.**

1 True/False Questions (30 pts)

For each question, please provide a short explanation to support your judgment.

Problem 1.1 (3 pts) Batch normalization pushes the mean of activations to zero. As each layer of the deep neural networks prefer normalized inputs, batch normalization helps a lot in improving the learning process of deep neural networks.

Problem 1.2 (3 pts) PyTorch uses dynamic graphs to represent deep neural networks. Tensors are eagerly computed during the forward/backward process. Thus, you can fetch the contents of tensors at any time of the computation.

Problem 1.3 (3 pts) Data augmentation is beneficial for any kinds of deep neural networks.

Problem 1.4 (3 pts) When we train on complicated datasets using an optimizer with dynamic learning rate (e.g., Adam optimizer), it is a common practice to increase the hyperparameter '`epsilon`' (e.g., increase to 1.0) to improve numeric stability and avoid oscillation.

Problem 1.5 (3 pts) Using both L-norm regularization and dropout ensures to achieve higher validation accuracy than using only one of these methods during training.

Problem 1.6 (3 pts) Compared to ReLU, leaky ReLU solves the problem of dead neurons (dying ReLU), but is more unstable due to non-zero gradient on the negative side of the activations.

Problem 1.7 (3 pts) During training, Lasso (L1) regularizer regularizes the model to have a higher sparsity compared to Ridge (L2) regularizer.

Problem 1.8 (3 pts) MobileNets use depthwise separable convolution to improve the model efficiency. If we replace all of the 3x3 convolution layers to 3x3 depthwise separable convolution layers in ResNet architectures, we are likely to observe approximately 9x speedup for these layers.

Problem 1.9 (3 pts) SqueezeNet puts most of the computations in later stages of the CNN design. Therefore, SqueezeNet uses fewer parameters than early CNN designs (e.g., AlexNet) while achieving comparable performance.

Problem 1.10 (3 pts) Deep residual networks introduce shortcut connections to smooth the loss surface.

2 Lab: Setting up the training pipeline (15 pts)

LeNet-5 is a classic convolutional neural network (CNN) architecture, originally designed for the task of hand-written digits recognition. It is composed of 2 CONV layers, 2 POOL layers and 3 FC layers. Although LeNet-5 is quite small and dated, it still contains many of the basic components of modern CNNs. Going through this model can provide us ideas about how to build a classifier and perform the training from scratch. This lab uses PyTorch to build LeNet-5 with 'ReLU' activation to conduct the CIFAR-10 task, as shown in Table 1.

Name	Type	Kernel size	depth/units	Activation	Strides
Conv 1	Convolution	5	6	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
Conv 2	Convolution	5	16	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
FC1	Fully-connected	N/A	120	ReLU	N/A
FC2	Fully-connected	N/A	84	ReLU	N/A
FC3	Fully-connected	N/A	10	None	N/A

Table 1: Adapted LeNet-5 Model. No padding is applied on both convolution layers. A flatten layer is required before FC1 to reshape the feature.

In HW1, we covered the PyTorch implementation of this adapted LeNet-5 and understood its structure by observing the shape of weights, input tensors and output tensors. In this lab, we will set up the training pipeline and actually train the LeNet-5 model on CIFAR-10 task.

Lab 1 (15 points)

In this assignment, you can refer to Jupyter Notebook `lenet5-cifar10.ipynb` for detailed instructions on how to construct a training pipeline for LeNet-5 model.

(a) (2 pts) As a sanity check, we should verify the implementation of the LeNet-5 model. How could you check whether the LeNet-5 model has the expected behavior?

(b) (2 pts) Dataset preprocessing is often used in both the training process and the testing process. Our target dataset, the CIFAR-10 dataset consists of 60,000 images with pixel values ranging from 0 to 255. How shall we normalize these images to have zero mean and unit variance? Please refer to **Step 1** on our Jupyter Notebook and fill out the required transformation for the CIFAR-10 dataset. (*Note that additional data augmentation like RandomCrop, RandomFlip transformations are allowed for this question*).

(c) (2 pts) To efficiently load a dataset for use in DNN training, we need to setup I/O handling procedures. Please refer to **Step 2** on our Jupyter Notebook and complete the data I/O pipeline. Note that instead of using the dataloader from `torchvision`, here you are asked use our own CIFAR10 dataloader, which is imported from `tools.dataset`. Otherwise, you will get incorrect evaluation results.

(d) (2 pts) We may instantiate our model and deploy it to GPUs for efficient training. Please refer to **Step 3** in the code to instantiate and deploy your LeNet-5 model on GPUs. How can you verify that your model is deployed on GPU? (Hint: check `nvidia-smi`).

(e) (2 pts) Loss functions are used to encode the learning objective. Now, we need to define this problem's loss function as well as the optimizer which will update our model's parameters to minimize the loss. In **Step 4**, please fill out the loss function and optimizer part. Note that we use SGD optimizer with momentum 0.9 as the default optimizer option. Remember to add L2 regularization into the optimizer to combat overfitting.

(f) (2 pts) Now we can kick off the training process. Please refer to **Step 5** for implementing the training process of our LeNet-5 model on the CIFAR-10 dataset. Follow the detailed instructions in **Step 5** for guidance.

(g) (3 pts) You can start training with your completed training pipeline **with the provided hyperparameter setting**. What is the initial loss value before you conduct any training step? How is it related to the number of classes in CIFAR-10? What can you observe from **training accuracy** and **validation accuracy**? Do you notice any problems with the current training pipeline?

(h) (Bonus, 4 pts) Currently, we take the default weight initialization method from PyTorch, which is sub-optimal. Try to adopt more advanced weight initialization techniques (e.g., Kaiming-Normal [1]). What can you observe when switching to a more advanced weight initialization?

At the end of Lab 1, we expect at least 63% validation accuracy if all the steps are completed properly. You are required to submit the completed version of `lenet5-cifar10.ipynb` for Lab 1.

3 Lab: Improving the training pipeline (35 pts)

In Lab 1, we develop a simplified training pipeline, which can give a decent LeNet-5 model. To obtain better training result, we will improve the training pipeline by employing data augmentation, improving the model design, and conducting hyperparameter tuning.

At the beginning of this lab, please duplicate the notebook in Lab 1 as `lenet5-cifar10-dev.ipynb`, and work on the new notebook. You are asked to train your model for a maximum of 100 epochs to reach at least 70% validation accuracy on the CIFAR-10 dataset.

Lab 2 (35 points)

(a) (6 pts) Data augmentation techniques help combat overfitting. Please add two data augmentations — *random crop* and *random flip* — into the training pipeline. You may explore some options (hyperparameters) for these data augmentations. What kind of options (hyperparameters) for these data augmentations can give the best result?

(b) (15 pts) Model design is another important factor in determining performance on a given task. Now, modify the design of LeNet-5 as instructed below:

- (5 pts) Add a batch normalization layer after each convolution/fully-connected layer except the last layer. What can you observe from the training process?
- (5 pts) Use empirical results to show that batch normalization allows a larger learning rate.
- (5 pts) Implement Swish [2] activation in PyTorch, and replace all of the 'ReLU' activations in LeNet-5 to 'Swish.' Does it improve the performance of LeNet-5?

(c) (14 pts) Hyperparameter settings are very important and can have a large impact on the final model performance. Based on the improvements that you have made to the model design thus far, tune some of the hyperparameters as instructed below:

- (7 pts) Apply different learning rate values: 1.0, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001, to see how the learning rate affects the model performance, and report results for each. Is a large learning rate usually beneficial for model training? If not, what can you conclude from the choice of learning rate?
- (7 pts) Use different L2 regularization strengths: 0.01, 0.001, 0.0001, 0.00001, 0.0, to see how the L2 regularization strength affects the model performance. Report the results for each regularization strength value along with commentary on the importance of this hyperparameter.
- (Bonus, 6 pts) Switch the regularization penalty from L2 penalty to L1 penalty. This means you may not use the `weight_decay` parameter in PyTorch builtin optimizers, as it does not support L1 regularization. Instead, you can try to add L1 penalty as a part of the loss function. Compare the distribution of weight parameters after L1/L2 regularization. Describe your observations.

Up to now, you shall have an improved training pipeline for CIFAR-10. We will continue to use it to produce better results by replacing LeNet-5 more advanced CNN models. After completing Lab 2, you are required to submit `lenet5-cifar10-dev.ipynb`.

4 Lab: Advanced CNN architectures (20 pts)

The fine-tuned training pipeline for LeNet-5 developed in Lab 2 still has limited performance. This is mainly because the structure of LeNet-5 is not good enough for CIFAR-10 task. Thus, in this lab we replace the LeNet-5 model with a more advanced ResNet [3] architecture, to enhance the representation ability of our neural network model. We expect to see much higher accuracy on CIFAR-10 after switching to ResNets. Here, please duplicate your work in the previous lab as `resnet-cifar10.ipynb`.

Lab 3 (20 points)

(a) (8 pts) Implement the ResNet-20 architecture by following Section 4.2 of the ResNet paper [3]. This lab is designed to have you learn how to implement a DNN model yourself. **DO NOT borrow any code from online resource.**

(b) (12 pts) Tune your ResNet-20 model to reach an accuracy of higher than 90%. You may use all of the previous techniques that you have learned so far. **For resource consideration, we recommend training the ResNet-20 model for a maximum of 200 epochs.**

Grading of this problem will take place on the holdout testing dataset, which you generally do not have any labels (refer to `tools/dataloader_test.py` and use the data loader inside to get your test data). For grading purposes, you are required to submit your predicted labels for each image in the test dataset. Upon submission, we will compare your predicted labels with the ground-truth labels to compute your final score.

Note, we provided a `sample.csv` file which shows the expected submission format for the predicted labels.

After completing Lab 3, you are required to submit `resnet-cifar10.ipynb`.

Info: Additional requirements:

- **DO NOT** train on the test set or use pretrained models to get unfair advantage. We have conducted a special preprocessing on the original CIFAR-10 dataset. As we have tested, "cheating" on the full dataset will give only 6% accuracy on our final test set, which means being unsuccessful in this assignment.
- **DO NOT** copy code directly online or from other classmates. We will check it! The result can be severe if your codes fail to pass our check.

Info: As this assignment requires much computing power of GPUs, we suggest:

- Plan your work in advance and start early. We will **NOT** extend the deadline because of the unavailability of computing resources.
- Be considerate and kill Jupyter Notebook instances when you do not need them.
- **DO NOT** run your program forever. Please follow the recommended/maximum training budget in each lab.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

[2] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv preprint arXiv:1710.05941*, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.