ECE 661: Homework #2
Construct, Train, and Optimize CNN Models

Hai Li

ECE Department, Duke University — September 9, 2021

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. When conducting the lab projects, actively referring to the **NumPy/PyTorch tutorial** slides on Sakai for instructions on the environment setup and NumPy/PyTorch utilities can be very helpful.

❖

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 **11:55pm, Monday, September 23**. We provide a template named `simplenn-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 PDF report, which provides answers to all the conceptual questions and clearly demonstrates all your lab results and observations. Remember, **do NOT generate PDF from your jupyter notebook to serve as the report**, which can increase the TA's burden of grading.

2. `code.zip`, a zipped code file which contains 3 jupyter notebooks `simplenn-cifar10.ipynb`, `simplenn-cifar10-dev.ipynb`, and `resnet-cifar10.ipynb`, respectively for the three labs.

3. `predictions.csv`, your predicted label for each image in the provided CIFAR-10 testing split. See detailed instructions in Lab (3).

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 normalizes the batch inputs by subtracting the mean, so the outputs of BN module will have exactly the zero mean.

Problem 1.2 (3 pts) PyTorch provides an efficient way of tensor computation and many modularized implementation of layers. However, you still need to write your own code for back-propagation.

Problem 1.3 (3 pts)   Data augmentation techniques are always beneficial for any kinds of CNNs and any kinds of images.

Problem 1.4 (3 pts)   Without Dropout, the CNNs can hardly or at least converge very slowly during the training.

Problem 1.5 (3 pts)   L-norm regularization can effectively combat overfitting. If Dropout is further incorporated at the same time, the performance is even better.

Problem 1.6 (3 pts)   During training, Ridge (L2) regularizer regularizes the model to have a higher sparsity compared to Lasso (L1) regularizer.

Problem 1.7 (3 pts)   Compared to ReLU, leaky ReLU solves the problem of dead neurons, but might make the training more unstable due to non-zero gradient on the negative side of the activations.

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)   To achieve fewer parameters than early CNN designs (e.g., AlexNet) while achieving comparable performance, SqueezeNet puts most of the computations in later stage of the CNN design.

Problem 1.10 (3 pts)   The shortcut connections in ResNets result in less smooth loss surface.

2   Lab (1): Training SimpleNN for CIFAR-10 classification (15 pts)

Just like in HW1, here we start with a simple CNN architecture which we term as SimpleNN. It is composed of 2 CONV layers, 2 POOL layers and 3 FC layers. The detailed structure of this model is shown in Table 1.

Name	Type	Kernel size	depth/units	Activation	Strides
Conv 1	Convolution	5	8	ReLU	1
MaxPool	MaxPool	2	N/A	N/A	2
Conv 2	Convolution	3	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: SimpleNN structure. No padding is applied on both convolution layers. A flatten layer is required before FC1 to reshape the feature.

In this lab, beyond model implementation, you will learn to set up the whole training pipeline and actually train a classifier to perform image classification on the CIFAR-10 dataset. A little bit of background on CIFAR-10 [1] that might be helpful. CIFAR-10 is one of the most famous/popular benchmarks for image recognition/classification. It consists of 10 classes (e.g., bird, dog, car, airplane), and the data samples are all 32x32 color images. You may go to the official website for more information `https://www.cs.toronto.edu/~kriz/cifar.html`.

In this assignment, please refer to Jupyter Notebook `simplenn-cifar10.ipynb` for detailed instructions on how to construct a training pipeline for SimpleNN model. **Note**, remember to unzip the provided `tools.zip` to your workspace before getting started.

(a) (2 pts) As a sanity check, we should verify the implementation of the SimpleNN model at **Step 0**. How can you check whether the model is implemented correctly?

Hint: 1) Consider creating dummy inputs that are of the same size as CIFAR-10 images, passing them through the model, and see if the model's outputs are of the correct shape. 2) Count the total number of parameters of all conv/FC layers and see if it meets your expectation.

(b) (2 pts) Data preprocessing is crucial to enable successful training and inference of DNN models. Specify the preprocessing functions at **Step 1** and briefly discuss what operations you use and why.

(c) (2 pts) During the training, we need to keep feeding data to the model, which requires an efficient data loading process. This is typically achieved by setting up a `dataset` and a `dataloader`. Please go to **Step 2** and build the actual training/validation datasets and dataloaders. Note, instead of using the `CIFAR10` dataset class from `torchvision.datasets`, here you are asked to use our own CIFAR-10 dataset class, which is imported from `tools.dataset`. As for the dataloader, we encourage you to use `torch.utils.data.DataLoader`.

(d) (2 pts) Go to **Step 3** to instantiate and deploy the SimpleNN model on GPUs for efficient training. How can you verify that your model is indeed 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.

(f) (2 pts) Please go to **Step 5** to set up the training process of SimpleNN on the CIFAR-10 dataset. Follow the detailed instructions in **Step 5** for guidance.

(g) (3 pts) You can start training now **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 do not decay the learning rate during the training. Try to decay the learning rate (you may play with the `DECAY_EPOCHS` and `DECAY` hyperparameters in Step 5). What can you observe compared with no learning rate decay?

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 `simplenn-cifar10.ipynb` for Lab (1).

3 Lab (2): Improving the training pipeline (35 pts)

In Lab (1), we develop a simplified training pipeline. To obtain better training result, we will improve the training pipeline by employing data augmentation, improving the model design, and tuning the hyperparameters.

Before start, please duplicate the notebook in Lab (1) as `simplenn-cifar10-dev.ipynb`, **and work on the new notebook**. You goal is to reach at least 70% validation accuracy on the CIFAR-10 dataset within a maximum training budget of **100 epochs**.

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 SimpleNN as instructed below:

- (5 pts) Add a batch normalization layer after each convolution 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 on you own, and replace all of the ReLU activations in SimpleNN to Swish. Does it improve the performance of SimpleNN?

(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. Remember, you are required to submit `simplenn-cifar10-dev.ipynb` for Lab (2).

4 Lab (3): Advanced CNN architectures (20 pts)

The improved training pipeline for SimpleNN developed in Lab (2) still has limited performance. This is mainly because the structure of SimpleNN is not good enough for CIFAR-10 task. Thus, in this lab we replace the SimpleNN 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, you may duplicate your jupyter notebook for Lab (2) as** `resnet-cifar10.ipynb` **to serve as a starting point.**

> **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, **so do NOT borrow any code from online resource.**
>
> (b) (12 pts) Tune your ResNet-20 model to reach an accuracy of higher than 90% on the validation dataset. 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.**
>
> We will grade this task by evaluating your trained model on the holdout testing dataset (which you do not have any labels). What you need to do is to make predictions on test data using your tuned ResNet-20 model, and save the predictions into the `predictions.csv` file. We provide `save_test_predictions.ipynb` to showcase how to save your predictions in required format. Upon submission, we will directly compare your predicted labels with the ground-truth labels to compute your score.

After completing Lab (3), you are required to submit `resnet-cifar10.ipynb` and the your prediction results `predictions.csv`.

> ❶ **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.

> ❶ **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] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[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.