Cifar-10 Classification with MLP and CNN

Background

After implementing the details about MLP, you might know how an ANN is implemented and trained. In this homework, we will focus on cifar-10 classification problem. In fact, neural networks with standard structures are often encapsulated as modules in deep learning frameworks, e.g., Tensorflow, PyTorch. It's convenient and also important for us to master the skills to use these modules and construct our models with deep learning frameworks.

You will be permitted to use frameworks such as PyTorch. We hope that you can implement MLP and CNN to finish the task of cifar-10 classification.

In addition, you should implement 2 another techniques.

Dropout, which aims to prevent overfitting. During training process, individual nodes are either "dropped out" of the net with probability p or kept with probability $1-p$, so that a reduced network is left, and incoming and outgoing edges to a dropped-out node are also removed. When testing, we would ideally like to find a sample average of all possible 2^n dropped-out networks; unfortunately this is unfeasible for large values of n. However, we can find an approximation by using the full network with each node's output weighted by a factor $1-p$, so the expected value of the output of any node is the same as in the training stage.

In this code, we implement dropout in an alternative way. During the training process, we scale the remaining network nodes' output by $1/(1-p)$. At testing time, we do nothing in the dropout layer. It's easy to find that this method has similar results to original dropout.

Batch normalization, which aims to deal with the internal covariate shift problem. Specifically, during training process, the distribution of each layer's inputs will change when the parameters of the previous layers change. Researchers proposed to do batch normalization of the input to activation function of each neuron, so that the input of each mini-batch has a mean of 0 and a variance of 1. To normalize a value x_i across a mini-batch,

$$BN_{initial}(x_i) = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where μ_B and σ_B denote the mean and standard deviation of the mini-batch, ϵ is a small constant to avoid dividing by zero. The transform above might limit the representation ability of the layer, thus we extend it to the following form:

$$BN(x_i) = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

where γ and β are learnable parameters. For instance, the output of the hidden layer in MLP is

$$y = \sigma(Wx + b)$$

After we normalize the input to activation function σ, the output can be represented as

$$y = \sigma(BN(Wx + b))$$

Hint:

1. Batch normalization of CNN should obey the convolutional property, thus different units in the same feature map should be normalized in the same way. Refer to **Reference[2]**.
2. When we test our samples **one by one**, the normalization may not work because we don't have minibatch this time. We should maintain the population mean and variance during training process and use them to estimate the "μ_B" and "σ_B" when testing. Simply, you can try to calculate the moving average of μ_B and σ_B when training, and take them as the population mean and variance. Refer to **Reference[2]**. The moving average **should not** be optimized in `optimizer.step()`.

Some classes/functions/attributes that you may need:

- `register_buffer`: This is typically used to register a buffer that should not to be considered a model parameter.
- `training`: Internal attributes (`bool`) of any class which are subclass of `nn.Module`. When setting `model.eval()`, `model.training` is set `False` automatically.
- `torch.bernoulli`: sampled from Bernoulli distribution by a success probability.

You are **NOT allowed** to use any classes or functions related to BN or dropout (e.g., nn.BatchNorm2d, nn.BatchNorm1d, F.batch_norm, nn.Dropout, F.dropout, etc.) in PyTorch.

Requirements

- Python 3
- PyTorch >= 1.4.0

Dataset Description

Utilize load_data.py to read the training set and test set. During your training process, information about testing samples in any form should never be introduced. Note that the shapes of data are different in MLP and CNN.

- MLP: To load data, use `load_cifar_2d()` in `load_data.py`.
- CNN: To load data, use `load_cifar_4d()` in `load_data.py`.

Python Files Description

In this homework, we provide unfinished implementation of MLP and CNN in **PyTorch** framework. Both programs share the same code structure:

- `main.py`: the main script for running the whole program.
- `model.py`: the main script for model implementation, including some utility functions.
- `load_data.py`: functions for data loading

MLP:

You are supposed to:

1. Implement "input -- Linear – BN – ReLU – Dropout – Linear – loss" network in forward() in model.py .
2. Implement **BatchNorm** and **Dropout** classes in model.py , and use them in 1.

CNN:

You are supposed to:

1. Implement “input – Conv – BN – ReLU – Dropout – MaxPool – Conv – BN – ReLU – Dropout – MaxPool – Linear – loss” network in **forward()** in `model.py`
2. Implement **BatchNorm2d** and **Dropout** classes in model.py , and use them in 1.

Report

In the experiment report, you need to answer the following basic questions:

1. Explain how `self.training` work. Why should training and testing be different?
2. Construct the MLP and CNN with batch normalization and dropout. **Write down** the hyperparameters that you use to obtain the best performance. **Plot** the loss value and accuracy (for both training and validation) against to every iteration during training.
3. Explain why training loss and validation loss are different. How does the difference help you tuning hyper-parameters?
4. Report the final accuracy for testing. Compare the differences between the results of MLP and CNN.
5. Construct MLP and CNN without batch normalization, and discuss the effects of batch normalization.
6. Tune the drop rate for MLP and CNN, respectively, and discuss the effects of dropout.

Attention: On your final submission, you need to submit MLP and CNN codes with **BN and dropout**.

Note: Keep at least one digit after the decimal point when you report the loss and accuracy(e.g., 55.5%).

Code Checking

We introduce a code checking tool this year to avoid plagiarism. You **MUST** submit a file named `summary.txt` along with your code, which contains what you modified and referred to. You should follow the instructions below to generate the file:

1. Fill the codes. Notice you should only modify the codes between `# TODO START` and `# TODO END`, the other changes should be explained in `README.txt`. **DO NOT** change or remove the lines start with `# TODO`.

2. Add references if you use or refer to a online code, or discuss with your classmates. You should add a comment line just after `# TODO START` in the following formats:

   1. If you use a code online: `# Reference: https://github.com/xxxxx`
   2. If you discuss with your classmates: `# Reference: Name: Xiao Ming Student ID: 2018xxxxxx`

   You can add multiple references if needed.

   Warning: You should not copy codes from your classmates, or copy codes directly from the Internet, especially for some codes provided by students who did this homework last year. In all circumstances, you should at least write more than 70% codes. (You should not provide your codes to others or upload them to Github before the course ended.)

   警告：作业中不允许复制同学或者网上的代码，特别是往年学生上传的答案。我们每年会略微的修改作业要求，往年的答案极有可能存在错误。一经发现，按照学术不端处理（根据情况报告辅导员或学校）。在任何情况下，你至少应该自己编写70%的代码。在课程结束前，不要将你的代码发送给其他人或者上传到github上。

3. Here is an example of your submitted code:

```
    def forward(self, input):
        # TODO START
        # Reference: https://github.com/xxxxx
        # Reference: Name: Xiao Ming Student ID: 2018xxxxxx
        your codes...
        # TODO END
```

4. At last, run `python ./code_analyze/analyze.py`, the result will be generated at `./code_analyze/summary.txt`. Open it and check if it is reasonable. A possible code checking result can be:

```
########################
# Filled Code
########################
# ..\codes\layers.py:1
    # Reference: https://github.com/xxxxx
    # Reference: Name: Xiao Ming Student ID: 2018xxxxxx
    your codes...

########################
# References
########################
# https://github.com/xxxxx
# Name: Xiao Ming Student ID: 2018xxxxxx

########################
# Other Modifications
########################
# _codes\layers.py -> ..\codes\layers.py
# 8 -         self._saved_tensor = None
# 8 +         self._saved_tensor = None # add some thing
```

Submission Guideline

You need to submit both report and codes, which are:

- **Report**: well formatted and readable summary including your results, discussions and ideas. Source codes should not be included in report writing. Only some essential lines of codes are permitted for explaining complicated thoughts.
- **Codes**: organized source code files with README for **extra modifications** (other than `TODO`) or specific usage. Ensure that others can successfully reproduce your results following your instructions. **DO NOT include model weights/raw data/compiled objects/unrelated stuff over 50MB (due to the limit of XueTang**)
- **Code Checking Result**: You should only submit the generated `summary.txt`. **DO NOT** upload any codes under `code_analysis`. However, TAs will regenerate the code checking result to ensure the correctness of the file.

You should submit a `.zip` file name after your student number, organized as below:

- `Report.pdf/docx`
- `summary.txt`
- `codes/`
    - `cnn/`

        - `*.py`
        - `README.md/txt`
    - `mlp/`
        - `*.py`
        - `README.md/txt`

Deadline: Oct. 18th

TA contact info:

- Jian Guan (关健)， j-guan19@mails.tsinghua.edu.cn

Reference

[1] Hinton G E, Srivastava N, Krizhevsky A, et al. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012. [2] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal CovariateShift, In Proceedings of the International Conference on Machine Learning, 2015: 448-456.