Cifar-10 Classification with MLP and CNN

Background

In this homework, we will try cifar-10 classification problem. After implementing the details about MLP, you might know more about it. In fact, similar neural networks with standard structures are often encapsulated as modules in deep learning frameworks. 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 Pytorch. We hope that you can understand the characteristics of them and implement MLP and CNN to finish the task of cifar-10 classification.

We introduce 2 tricks to improve the performance.

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 . 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 μ_B, and σ_B when training, and take them as the population mean and variance. Refer to **Reference[2]**.

You are **allowed** to use all necessary API in PyTorch, including `torch.nn.BatchNorm2D` and `torch.nn.Dropout`.

Requirements

- Python >= 3.6
- torch >= 1.1

Dataset Description

We use cifar-10 dataset, see the link for details.

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:

Implement "input -- Linear -- BN -- ReLU -- Dropout -- Linear -- loss" network in `model.py`.

CNN:

You are supposed to:

Implement "input -- Conv -- BN -- ReLU -- Dropout -- MaxPool -- Conv -- BN -- ReLU -- Dropout -- MaxPool -- Linear -- loss" network in `model.py`

Report

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

1. Write down how you fill the arguments of `model`.
2. Plot the loss value (both training loss and validation loss) 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. Construct MLP and CNN without batch normalization, and discuss the effects of batch normalization.
5. Tune the drop rate in Dropout Layer, and discuss the effects of dropout.

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**

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

- `Report.pdf`
- `codes/`
    - `cnn/`
        - `*.py`
        - `README.md`
    - `mlp/`
        - `*.py`
        - `README.md`

Deadline: July 27th

Submission to HUANG Fei （黄斐）, huangfei382@163.com

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.