COMP3314 Machine Learning

Programming Assignment 2: Convolutional Neural Network

Release date: April 11, 2020
Due date: 11:59pm, May 8, 2020
(20% deduction for every 24 hours after the due date)

Task:

This assignment is about the implementation of a modified LeNet on the dataset MNIST, it is highly related to the implementation of the forward and backward methods of commonly seen neural network components such as fully connected layers, convolutional layers, activation layers and max-pooling layers. The implemented components will be tested on the classic MNIST dataset. A code template is provided by the tutors to facilitate the implementation.

Datasets:

This experiment will be conducted on the handwritten digit recognition dataset MNIST. It is a 10-category classification which classifies handwritten digit images into 0-9. The training set contains 60000 examples and the testing set contains 10000 examples. Each input image is gray scale image and has 32*32 pixels.

Guidelines:

[1] Students should implement a class of LeNet. It should have the method of forward propagation, and backward propagation. Students can divide the task into several modules (fc/conv/relu, etc.) and combine them together. **Do not develop your work based on existing neural network packages (pytorch/tensorflow/…)**.
[2] The provided code template is written in python: it already completes the data loading, data preprocessing and random sampling work. It also includes a simple training schedule and a simple testing metric. But **students can choose any other languages, and they are encouraged to modify the provided code template if needed and try different training schemes/regularizations/learning rates**. For instance, more metrics can also be added into the code template if related analysis is provided in the project report, and more epochs can be utilized to train the model.
[3] LeNet has 5 parameterized layers, which includes 2 convolutional layers and 3 fully-connected layers. In order to facilitate the implementation, we modify the second custom convolutional layer as a common convolutional layer. **The structure of LeNet network to implement is described as followed**:
Layer C1 is a convolution layer takes input images of size 3*32*32. It has 6 convolution kernels of 5*5. It has no padding and the size of its output feature is 6*28*28.

Layer S2 is the max-pooling layer with size and stride of 2. Its output feature size is 6*14*14. S2 is followed by a ReLU activation function.
Layer C3 has 16 convolution kernels of 5*5. It has no padding and output feature size is 16*10*10.
Layer S4 is similar to S2, with size of 2*2 and output feature map of 16*5*5. S4 is also followed by a ReLU activation function.
Layer C5 has 120 convolution kernels of 5*5. It has no padding and output feature size is 120*1*1. C5 is followed by a ReLU activation function.
Layer F6 is fully connected to the C5 output feature vector of size 120*1*1 and has 84 output channels. The output feature of F6 is 84. F6 is followed by a ReLU activation function.
Layer F7 is fully connected to the F6 output feature vector of size 84. It has 10 output features which corresponds to the 10 classes for digit 0-9.
Cross-entropy taught in the lecture slides is the cost function in this assignment. Weight regularization is not compulsory in this task. The optimization policy is stochastic gradient descend.
[4] **A brief introduction of provided code template** is as follows:
1) Hyper-prarameters are in config.yaml
2) Data pre-processing: Normalized into 0-1. Then it is scaled with z-score method ((a-mean)/std).
3) Data format: Input batch is a tuple where the first element is the input image batch with size (n_batch, size, size, 1) and the second element is the sample label with size (n_batch, 1).
4) Training scheme: 20 epochs in total, learning rates for epochs are set to [5e-2]*2, [2e-2]*3, [1e-2]*3, [5e-3]*4 and [1e-3]*8 respectively. The data has been shuffled for each epoch, and the learning policy is gradient descend taught in the lecture.
5) User interface: Students should implement a model with at least ***construction method, forward method and backward method***.
Construction method should has the form:

```
model = LeNet5()
```

It is called in the main function in **line 167**.
Forward method:
During training, it should has the form:

```
loss = model.Forward_Propagation(batch_image, batch_label, mode='train')
```

During evaluating, it should has the form:

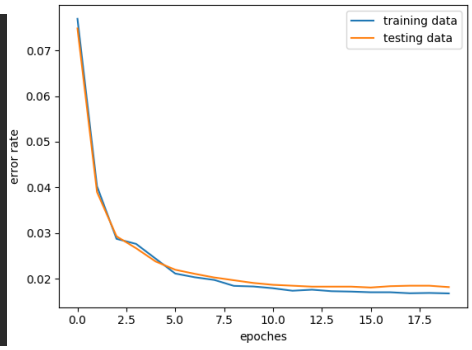
```
error_rate = model.Forward_Propagation(test_data[0], test_data[1], mode='eval')
```

An additional variable mode is added here because during evaluation, the forward method only returns the predicted result. During training, the function returns the cross entropy. Students can also use two separate methods to implement it. They are called in **line 131** and **137/138** respectively.
Backward method should has the form:

```
model.Back_Propagation(lr_global)
```

It is called during training in **line 134**.
6) Running: Configure the config.yaml if needed and run A2.py. If the program is correctly implemented, it should looks like:

```
---------- epoch 19 begin ----------
learning rate: 0.001
batch size: 256
Training:
100%|██████████| 235/235 [04:05<00:00,  1.04s/it]
Done, total cost of epoch 19: 67987.77143455627
0/1 error of testing set:  185 / 10000
Time used:  320.8031828403473 sec
---------- epoch 19 end ------------
---------- epoch 20 begin ----------
learning rate: 0.001
batch size: 256
```



*code is modified from https://github.com/mattwang44/LeNet-from-Scratch

[5] **Report writing**: There are many possible ways to implement the back-propagation of these layers. Thus students should **include the mathematical expression of their own Jacobian matrices in their reports** as illustrated in slide Lecture 4 pages 52-68. Tutors will check the implementation accordingly. Students are also encouraged to **assess the model of its over-fitting condition, its convergence of training, error rate, etc**. **If the code template is modified to replace the provided training scheme, a detailed comparison should be provided between different experiment settings.**

Submission Instructions:

[1] One report in **pdf** describing the implementation and discussion about the model.

[2] Source codes packed in **zip** format that can be unzipped and compiled. Attach trained model file linkage and remove trained models in case zipped file exceeds file size limit.