Project2: MNIST Classification with MLP

1 Abstrct

In this experiment you need to design and implement an MLP using python to classify MNIST handwritten digits dataset, and write the BP algorithm by yourself. You need to finish the following requirements:

- Construct an MLP with one hidden layer of 256 units using **Sigmoid** activation function and the **softmax_cross_entropy _loss**;
- Implement the optimizer SGD, and add momentum to it with the hyperparameter **momentum=0.9**.
- Redo using **ReLU** activation function and compare the performance;

We have provided a code framework for you, you just need to complete the **#TODO**s.

2 Introduction

MNIST digits dataset is a widely used dataset for image classification in machine learning field. It contains 60,000 training examples and 10,000 testing examples. The digits have been size-normalized and centered in a fixed-size image. Each example is a 784×1 matrix, which is transformed from an original 28×28 grayscale image. Digits in MNIST range from 0 to 9. Some examples are shown below. **Note**: During training, information about testing examples should never be used in any form.

3 MLP for MNIST Classification

3.1 Files Description

There are several files or folders included in the **./work/MLP/Project2** folder:

- **data/*ubyte** where the the MNIST dataset are saved, including the training data, training label, test data and test label.
- **solver.py** where we have implemented the training and test pipeline, including the main function. We recommend that you start reading the code from this file.
- **network.py** where we have implemented the network module, which can be utilized when defining network architecture and performing model training.
- **dataloader.py** where we have implemented the dataloader, which can be used to prepare the data for training and test.
- **visualize.py** where we have implemented the **plot_loss_and_acc** function, which can be used to plot curves of loss and accuracy.
- **optimizer.py** where you need to implement the SGD optimizer class, which can be used to perform the forward and backward propagation.
- **layers/fc_layer.py** where you need to implement the fully connected layer, which maps the input vector $\mathbf{x}$ into an output vector $\mathbf{u}$, by $\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W}$ is the matrix saving layer weights, and $\mathbf{b}$ is the bias.
- **layers/sigmoid_layer.py** where you need to implement the **Sigmoid** activation function, which computes the final output of a layer as $f(\mathbf{u}) = \frac{1}{1+\exp(-\mathbf{u})}$.
- **layers/relu_layer.py** where you need to implement the **ReLU** activation function, which computes the final output of a layer as $f(\mathbf{u}) = max(\mathbf{0}, \mathbf{u})$.
- **loss.py** where you need to implement the **softmax_cross_entropy _loss**, you can learn more detail about this loss function by yourself.

3.2 Requirements

You are required to complete the **# TODO** parts in above files. **You need to submit all codes and a short report** with the following requirements:

- Record the training and test accuracy, plot the training loss curve and training accuracy curve in the report.
- Compare the differences of results when using **Sigmoid** and **ReLU** as activation function.
- Compare the differences of results when using momentum or not. (You can discuss the differences from the aspects of training time, convergence and accuracy).
- The given hyperparameters maybe perform not very well. You can modify the hyperparameters on your own, and observe how do these hyperparameters affect the classification performance. Write down your observation and record these new results in the report.

4 Attention

- You need to submit all codes and a report (at least two pages **in PDF format**). Remove the folder for saving MNIST dataset before you submit the final results.
- Pay attention to the efficiency of your implementation. Try to finish this project without the use of **for-loops**, using matrix multiplication instead.
- Do not paste a lot of codes in your report (only some essential lines could be included). Adding any extra Python files should be explained and documented.
- Any open source neural network toolkits, such as TensorFlow, PyTorch, are **NOT** permitted.
- **Plagiarism is not permitted**.