MNIST Classification with MLP (Score 10)

Homework 3 for Deep Learning, Autumn 2021

Deadline: 2021.10.19 23:55:00

1 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.

In this homework, you are required to use **Multilayer Perceptron** to perform MNIST classification.

2 MLP for MNIST Classification

2.1 Files Description

There are several files included in the **./homework3/** folder:

- **homework3.ipynb** describes the main contents of this homework. **Please read this file carefully**.
- **network.py** describes network class, which can be utilized when defining network architecture and performing model training.
- ***optimizer.py** describes SGD optimizer class, which can be used to perform forward and backward propagation.
- **solver.py** describes training and testing pipeline.
- **plot.py** describes plot_loss_and_acc function which can be used to plot curves of loss and accuracy.

In addition, there are several layers defined in **./criterion/** and **./layers/**. Our implementation is guided by *modularity* idea. Each layer class has three methods: *__init__*, *forward* and *backward*.

__init__ method is used to define and initialize some parameters. *forward* and *backward* are used to perform forward and backward propagation respectively.

- ***FCLayer** treats each input as a simple column vector (need to reshape if necessary) and produces an output vector by doing matrix multiplication with weights and then adding biases: $\mathbf{u} = \mathbf{W}\mathbf{x} + \mathbf{b}$.
- ***SigmoidLayer** is a sigmoid activation unit, computing the output as $f(\mathbf{u}) = \frac{1}{1+\exp(-\mathbf{u})}$.
- ***ReLULayer** is a linear rectified unit, computing the output as $f(\mathbf{u}) = max(\mathbf{0}, \mathbf{u})$.
- ***EuclideanLossLayer** computes the sum of squares of differences between inputs and labels $\frac{1}{2}\sum_n \|logits(n) - gt(n)\|_2^2$.
- ***SoftmaxCrossEntropyLossLayer** can be viewed as a mapping from input to a probability distribution in the following form:

$$P(t_k = 1|\mathbf{x}) = \frac{\exp(x_k)}{\sum_{j=1}^{K} \exp(x_j)} \tag{1}$$

where x_k is the k-th component in the input vector $\mathbf{x}$ and $P(t_k = 1|\mathbf{x})$ indicates the probability of being classified to class k given the input. Since the output of softmax layer can be interpreted as a probability distribution, we can compute the delta likelihood and its logarithm form is also called cross entropy error function:

$$E = -\ln p(t^{(1)}, ..., t^{(N)}) = \sum_{n=1}^{N} E^{(n)} \tag{2}$$

where

$$E^{(n)} = -\sum_{k=1}^{K} t_k^{(n)} \ln h_k^{(n)} \tag{3}$$

$$h_k^{(n)} = P(t_k^{(n)} = 1|\mathbf{x}^{(n)}) = \frac{\exp(x_k^{(n)})}{\sum_{j=1}^{K} \exp x_j^{(n)}}. \tag{4}$$

The definition of the softmax loss layer is a little different from *homework2*, since we don't include trainable parameters θ in this layer. However these parameters can be explicitly extracted out to form an individual **FCLayer**.

2.2 Requirements

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

- Record the training and testing 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 (you can discuss the differences from the aspects of training time, convergence and accuracy).
- Compare the differences of results when using **EuclideanLoss** and **SoftmaxCrossEntropyLoss** as loss function.
- Construct a MLP with **two hidden layers** (choose the number of hidden units by your own), using any activation function and loss function. Also, compare the differences of results between one layer structure and two layers structure.
- The given hyerparameters maybe performed not very well. You can modify the hyerparameters by your own, and observe how does these hyerparameters affect the classification performance. Write down your observation and record these new results in the report.

3 Attention

- You need to submit all codes and a report (at least two pages **in PDF format**). Delete the MNIST dataset before submit.
- Pay attention to the efficiency of your implementation. Try to finish this homework 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). Any extra modifications of above homework files or adding extra Python files should be explained and documented.
- Any open source neural network toolkits, such as TensorFlow, Caffe, PyTorch, are **NOT** permitted in finishing homework-3.
- **Plagiarism is not permitted**.