MNIST Digits Classification with MLP

Background

MNIST digits dataset is a widely used database for image classification in machine learning field. It contains 60,000 training samples and 10,000 testing samples. Each sample is a 784×1 column vector, which is transformed from an original 28×28 pixels grayscale image. Some typical digits images are shown below.



In this homework, you need to use multilayer perceptron (MLP) to perform digits classification.

Requirements

Currently we use python version 3.5 and numpy version >= 1.15.0

Dataset Description

To load data, first extract `.gz` files under `./data`, and just use

```python
from load_data import load_mnist_2d
train_data, test_data, train_label, test_label = load_mnist_2d('data')
```

Then `train_data, train_label, test_data` and `test_label` will be loaded in `numpy.array` form. Digits range from 0 to 9, and corresponding labels are from 0 to 9.

Attention: during your training process, information about testing samples in any form should never be introduced.

Python Files Description

In neural network, almost every data processing can be viewed as a functional layer. And a neural network can be constructed by stacking multiple layers to define a certain data processing pipeline. So our neural network implementation is guided by *modularity* idea. Each layer class has four main methods: constructor, forward, backward and update. For some trainable layers with weights and biases, constructor functions as parameter initialization and update function will update parameters by stochastic gradient descent. Forward represents the data processing performed by the layer and backward performs backpropagation operations. In `layers.py` each layer's definition is listed below.

- `Layer`: base class for all layers

- `Linear`: treat each input as a row vector and produce an output vector by doing matrix multiplication with weight and then adding bias $u = xW + b$
- `Relu`: linear rectifier activation unit, compute the output as $f(u) = \max(0, u)$
- `Sigmoid` sigmoid activation unit, compute the output as $f(u) = \frac{1}{1+\exp(-u)}$

Also, there are two kinds of loss layers defined in `loss.py`

- `EuclideanLoss` compute the mean of sum of squares of differences between inputs and labels $\frac{1}{2N}\sum_n^N \|T(n) - y(n)\|_2^2$, where N denotes the batch size (the number of samples in one mini-batch)
- `SoftmaxCrossEntropyLoss`. Given the groudtruth labels $\mathbf{t}^{(1)}, \cdots, \mathbf{t}^{(N)}$ (one-hot encoding form) and the corresponding prediction vectors $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(N)}$, `SoftmaxCrossEntropyLoss` can be computed in the form $E = \frac{1}{N}\sum_{n=1}^N E^{(n)}$ where

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

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

When running backpropagation algorithm, `grad_output` is an important variable to compute gradient in each layer. We define `grad_output` to be **the derivative of loss with respect to layer's output.**

Attention: Since layer definitions here are a little different from lecture slides because we explicitly split out activation layers, you should implement backward method of activation layer separately. Hope you realize this.

Attention: The definition of `SoftmaxCrossEntropyLoss` layer is a little different from slides, since we don't include trainable parameters θ in the layer, because this parameter can be explicitly separated out and functions exactly as an extra `Linear` layer.

There are also other four files included in the codes.

- `utils.py` includes some utility functions
- `network.py` describe network class, which can be utilized when defining network architecture and performing training
- `solve_net.py` which includes `train_net` and `test_net` functions to help training and testing (doing stuff like forward, backward, weights update, logging information).
- `run_mlp.py` the main script for running the whole program. It demonstrates how to simply define a neural network by sequentially adding layers.

If you implement layers correctly, just by running `run_mlp.py`, you can obtain lines of logging information and reach a relatively good test accuracy. All the above files are encouraged to be modified to meet personal needs.

Attention: any modifications of these files or adding extra python files should be explained and documented in README.

Report

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

1. plot the loss value against to every iteration during training
2. construct a neural network with one hidden layer (`Input -> Linear -> Activation -> Linear -> Loss`), and compare the difference of results when using `Sigmoid` and `Relu` as activation function (you can discuss the difference from the aspects of training time, convergence and accuracy)

3. conducting same experiments above, but with two hidden layers (`Input -> Linear -> Activation -> Linear -> Activation -> Linear -> Loss`). Also, compare the difference of results between one layer structure and two layers structure
4. compare the difference between `EuclideanLoss` and `SoftmaxCrossEntropyLoss` for all the above experiments

Attention: The current hyperparameter settings may not be optimal for good classification performance. Try to adjust them to make test accuracy as high as possible (at least 98% test accurarcy).

Attention: Any deep learning framework or any other open source codes are **NOT** permitted in this homework. Once discovered, it shall be regarded as plagiarism.

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 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 in the submission files**

Deadline: Mar. 28th

TA contact info: Yulong Wang (王宇龙）, wang-yl15@mails.tsinghua.edu.cn