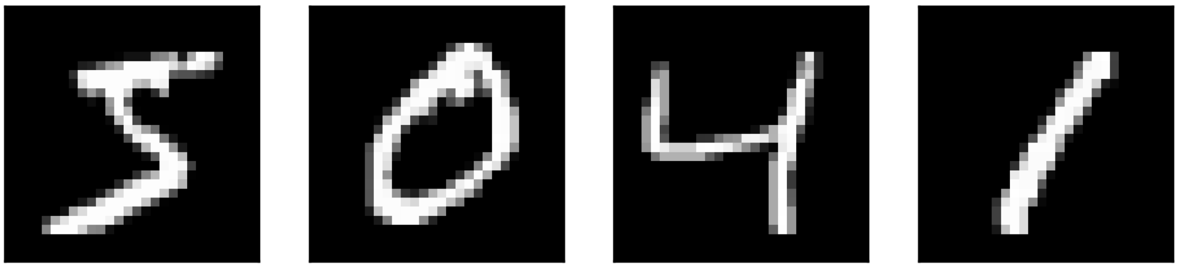
MNIST Digit Classification with MLP

Background

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



In this homework, you need to implement multilayer perceptron (MLP) to perform digit classification on MNIST.

Requirements

- python >= 3.6
- numpy >= 1.12.0

Hints

We may use different python environments through this course (because different packages have different dependencies). We highly recommend you to use anaconda (or miniconda) to manage your packages. You can download anaconda here (https://mirrors.tuna.tsinghua.edu.cn/help/anaconda/) without the charge of tsinghua net.

Dataset Description

To load data, you may 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.

NOTE: during the training process, any information about testing samples should never be introduced.

Python Files Description

In neural networks, each basic operation 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 the neural network implementation is a modular design. 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 method will update parameters via stochastic gradient descent. Forward method represents the data processing performed by the layer and backward performs backpropagation operations. In `layers.py` and `loss.py`, each layer's definition is listed as below.

- `Layer`: base class for all layers
- `Linear`: treat each input as a row vector x and produce an output vector by doing matrix multiplication with weight W and then adding bias $b : u = xW + b$
- `Relu`: linear rectifier activation unit, compute the output as $f(u) = max(0, u)$
- `Sigmoid`: sigmoid activation unit, compute the output as $f(u) = \frac{1}{1+exp(-u)}$
- `EuclideanLoss`: compute the squares of the Euclidean norm of differences between inputs $y(n)$ and labels $t(n)$: $\frac{1}{2N}\sum_{n=1}^{N}||t(n) - y(n)||_2^2$, where N denotes the batch size (the number of samples in one mini-batch). The labels in this task are represented in one-hot form.
- `SoftmaxCrossEntropyLoss` : `Softmax` function can map the input to a probability distribution in the following form:

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

  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 ground-truth labels $\mathbf{t}^{(1)}, \cdots, \mathbf{t}^{(N)}$ (one-hot encoding form) and the corresponding predicted 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 essential variable to compute gradient in each layer. We define `grad_output` as the derivative of loss with respect to layer's output.

NOTE: Since layer definition here is a little different from lecture slides because we explicitly split out activation layers, you should implement backward method in activation layer separately. Hope you realize this.

All files included in the codes:

- `layers.py`: you should implement `Layer`, `Linear`, `Relu`, `Sigmoid`
- `loss.py`: you should implement `EuclideanLoss`, `SoftmaxCrossEntropyLoss`
- `load_data.py`: load mnist dataset
- `utils.py`: some utility functions
- `network.py`: network class which can be utilized when defining network architecture and performing training
- `solve_net.py`: train_net and test_net functions to help training and testing (running forward, backward, weights update and 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.

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

Report

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

1. plot the **loss** and **accuracy** against to every iteration during training for **all experiments**.
2. construct a neural network with one hidden layer, and compare the difference of results when using `Sigmoid` and `Relu` as activation function, as well as using `EuclideanLoss` and `SoftmaxCrossEntropyLoss` as loss function (you can discuss the difference from the perspectives of training time, convergence and accuracy). (At least 3 experiments needed and remember controlling variables.)
3. conducting the same experiments above, but with two hidden layers. Also, compare the difference of results between one-layer structure and two-layer structure. (At least 2 more experiments needed.)

NOTE: The current hyperparameter settings may not be optimal for good classification performance. Try to adjust them to make test accuracy as high as possible.

NOTE: 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 a report document and the codes, as required as follows:

- **Report:** well formatted and readable summary to describe the network structure and details, experimental settings, results and your analysis. Source codes should not be included in the report. Only some essential lines of codes are permitted. The format of a good report can be referred to a top-conference paper. (Both Chinese and English are permitted.)
- **Codes:** organized source code files with README for extra modifications or specific usage. Ensure that TAs can easily reproduce your results following your instructions. DO NOT include model weights/raw data/compiled objects/unrelated stuff over 50MB (due to the limit of XueTang).

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

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

Note: The accuracy of your best model is required to exceed 95%. If not, TAs believe there must be something wrong with your code. Even if you reach the requirement, TAs will go through your code for any possible bugs.

Deadline

October 7

TA contact: 关健，j-guan19@mails.tsinghua.edu.cn