MNIST Digits Classification with Theano (MLP + CNN)

Background

In this homework, we will continue working on MNIST digits classification problem by utilizing popular deep learning framework Theano. It is a python library that allows efficient mathematical expressions optimization and evaluation. It mainly features on transparent use of GPU and efficient symbolic differentiation, which are important to implement convolutional neural networks.

Theano has been well documented on its main website. So please follow its official instructions when installing the library. More thorough tutorials about how to construct neural networks with Theano can be found on its tutorial pages.

Attention: for linux/mac users, you can install theano from terminal by using `sudo pip install theano`. For windows users, the most comfortable way is to install Anaconda or Canopy, which includes almost all dependencies and use its package manager to install theano. After finishing installation, create a file `.theanorc` (or `.theanorc.txt` for windows users) and write following configurations into it.

```
[global]
floatX = float32
```

It sets the default data type retured by theano tensors. Then move the configuration file to the location `/home/<YOUR-USER-NAME>/` (or `C:\User\<YOUR-USER-NAME>\` for windows users)

Dataset Description

All the data are stored `data` folder. `mnist.py` file will handle the data loading. Please refer to the dataset description section in Homework 1 for details.

Python Files Description

- `layers.py`: including `Relu`, `Sigmoid`, `Softmax`, `Linear`, `Convolution`, `Pooling` layer implementations
- `loss.py`: `CrossEntropyLoss` implementation
- `network.py`: class for assembling layers to a network. Notice there are 3 functions of network after compiling: `train`, `test`, `predict`.
- `optimizer.py`: SGD optimizers. Given parameters and corresponding gradients, return updated parameters
- `solve_net.py`: function which controls the training and testing process.
- `mnist.py`: data loading
- `utils.py`: some auxillary functions.
- `run_mlp.py`, `run_cnn.py`: main programs for MLP and CNN

Attention: most of layers have already been implemented in Theano library, so study the documentation carefully to find your treasure!

Report

We perform the following experiments in this homework:

1) construct a MLP using Relu as activation function and compare the difference of results when using one hidden layer and two hidden layers. The network is trained with `Softmax` with `CrossEntropyLoss`.

2) construct a CNN with 2 `Convolution` + `Relu` + `Pooling` modules and a final `Linear` layer to predict labels. Compare the difference of results with 1). Also compare the difference with results you get in Homework 2.

3) construct Adagrad optimizer and compare the results with 2) (from training time/convergence/testing accuracy). Adagrad is an adaptive learning rate method originally proposed by Duchi et al.. It can be described as follows:

```
# Assume the gradient dx and parameter vector x
cache += dx**2
x = x - learning_rate * dx / (sqrt(cache) + eps)
```

Submission Guideline:

- **report**: well formatted 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 any extra installation or other procedures. Ensure one can successfully *reproduce* your results following your instructions. **DO NOT** include model weights/raw data/compiled objects/unrelated stuff over 100MB...

Deadline: Nov. 9

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