MNIST Digits Classification with CNN

Background

In this homework, we will continue working on MNIST digits classification problem by utilizing convolutional neural network (CNN).

The main challenge is to implement the forward and backpropagation functions of convolutional layer and pooling layer from scratch! If you succeed, you will obtain a profound understanding of deep learning mechanism.

Requirements

Currently we use python version 3.5, numpy version >= 1.15.0 and scipy version >= 0.17.0

Dataset Description

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

```
from load_data import load_mnist_4d
train_data, test_data, train_label, test_label = load_mnist_4d('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: Image data passing through the network should be a 4-dimensional tensor with dimensions $N \times C \times H \times W$, where H and W denote the height and width of image, C denotes the number of image channels (1 for gray scale and 3 for RGB scale), and N denotes the number of batch size. Among these dimensions, channel number C would be a little hard to interpret for hidden layer output. In this context, we interpret the i-th channel of the n-th sample in one mini-batch `output[n, i, :, :]` as the output of convoluational layer's `input[n, :, :, :]` with **one** filter `W[i, :, :, :]`.

Python Files Description

`layers.py`, `network.py`, `solve_net.py` are similar to those included in Homework 1.`run_cnn.py` is the main script for running whole program. It demonstrates how to define a neural network by sequentially adding layers and train the net.

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

There are two new layers `Conv2D` and `AvgPool2D` in `layers.py`. But the implementations of forward and backward are included in `functions.py`. Here are some important descriptions about these classes and functions:

- `Conv2D` describes the convolutional layer which performs convolution with input using weight. It consists of two trainable parameters weight **W** and bias **b**. **W** is stored in 4 dimensional matrix with dimensions $n_{out} \times n_{in} \times k \times k$, where k (or called `kernel_size`) specifies the height and width of each 2D kernel, n_{in} denotes the channel numbers of input, and n_{out} denotes the number of 3D filters.

- `conv2d_forward` implements the convolution operation given layer's weight and bias and input. For simplicity, we **only** need to implement the standard convolution with `stride` equal to 1. There is another important parameter `pad` which specifies the number of zeroes added to each side of **input** (not output!). Therefore the expected height of output should be equal to $H + 2 \times$`pad` $-$ `kernel_size` $+ 1$ and width likewise.

- `AvgPool2D` describes the pooling layer. For simplicity, we **only** need to implement average pooling operation in non-overlapping style (which means `kernel_size` = `stride`). Therefore the expected height of output should be equal to ($H + 2 \times$`pad`) / `kernel_size` and width likewise.

Hint: To accelerate convolution and pooling, you should avoid too many nested `for` loops. Instead, use matrix multiplication and numpy, scipy functions as much as possible. To implement convolution with multiple input channels, one possible way is to utilize `conv2` function to perform convolution channel-wise and then sum across channels. To implement pooling operation, one can use `im2col` function to lay out each pooling area and rearrange them in a matrix. Generally, by using `im2col` properly one can implement both convolution and pooling operation in the most general form (like convolution with `stride` bigger than 1 and max pooling). However, the backpropagation form would be tricky and it needs much work to succeed! There are also faster ways to implement the above required convolution and pooling operations, try to explore and discover them!

Report

We perform the following experiments in this homework:

1. plot the loss value against to every iteration during training

2. construct a CNN with structure `Conv-ReLU-Pool-Conv-ReLU-Pool-Reshape-Linear` to predict labels training with `SoftmaxCrossEntropyLoss`. Compare the difference of results you obtained when working with MLP (you can discuss the difference from the aspects of training time, convergence, numbers of parameters and accuracy)

3. try to visualize the first convolution layer's output after ReLU for 0-9 digit images. Refer to caffe visualization tutorial [1] for more details.

Attention: Source codes should not be included in report. Only some essential lines of codes are permitted to be included for explaining complicated thoughts.

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 over 50MB**

Deadline: Apr. 18th

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

[1]http://nbviewer.ipython.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb