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 final loss layer is changed to cross-entropy loss layer when output probability is generated by softmax function.

The main challenge is to implement the forward and backpropagation functions of convolutional layer and pooling layer from scratch! But if you succeed, you will obtain a profound understanding of deep learning mechanism. So roll up your sleeves and get started!

Requirements

Currently we use python version 2.7, numpy version >= 1.12.0 and scipy version >= 0.17.0

Dataset Description

To load data, 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 matrix (or *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 confusing concept for hidden layer output. In this context, we interpret the i-th channel of the n-th sample in one mini-batch (or `output[n, i, :, :]` written in python syntax) as the convolutional output of layer's input with **one** filter W_i.

Python Files Description

`layers.py`, `network.py`, `solve_net.py` are identical 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.

The new loss layer in `loss.py` is `SoftmaxCrossEntropyLoss`. `Softmax` function can map 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 input. Given the groudtruth 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

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

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

In `SoftmaxCrossEntropyLoss` you need to implement its forward and backward methods.

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

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 $\mathbf{W}$ and bias $\mathbf{b}$. $\mathbf{W}$ is stored in 4 dimensional matrix with dimensions $n_{out} \times n_{in} \times k \times k$, where k specifies the height and width of each filter (or called `kernel_size`), n_{in} denotes the channel numbers of input which each filter will convolve with, and n_{out} denotes the number of 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 zeros 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 and 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 employ `im2col` function to lay out each pooling area and rearrange them in a matrix. Then perform pooling operation on each column. Theoretically, 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 would be very tricky and involve much endeavor to make it work! 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 two `Conv2D` + `Relu` + `AvgPool2D` modules and a final `Linear` layer to predict labels using `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 rectified linear unit for 4 different 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 (due to the limit of XueTang)**

Deadline: Oct. 25th

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