MNIST Digits Classification with CNN

Yulong Wang wang-yl15@mails.tsinghua.edu.cn

1 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! But if you succeed, you will obtain a profound understanding of deep learning mechanism. So roll up your sleeves and get started!

2 Dataset Description

All the data are stored in `mnist.mat`. We use the same data format as described in Homework 1. Please refer to the dataset description section in Homework 1 for details. Since CNN operates on two-dimensional input, one needs to convert the raw data of 784×1 vector into 28×28 pixels image. The data preprocessing and conversion have been implemented in `test_CNN.m`, so you don't need to worry about this part :-).

Attention: However, after the conversion, you may notice that a typical batch of image data passing through the network would be a 4-dimensional matrix (or *tensor*) with dimensions $H \times W \times C \times N$, 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 data of the n-th sample in a single batch (or `data(:, :, i, n)` written in MATLAB syntax) as the convolutional output of bottom layer's input with **one** filter W_i.

3 MATLAB Files Description

In the source package, these files including `fcLayer.m`, `Relu.m`, `Network.m` and `softmaxLossLayer.m` are identical to those included in Homework 1 and 2. So you can reuse your implementations in this task. `Fit_cnn.m` is very similar to `Fit_mlp.m` with only minor modifications adapted for 4 dimensional input data. `test_CNN.m` 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: If you add any major modifications to these files (like code refactoring or exception handling), please explain them in README file.

There are two new layer files `convLayer.m` and `poolLayer.m`. But the realizations of forward and backpropagation are wrapped in four function files, `nnconv.m`, `nnpool.m`, `nnconv_bp.m` and `nnpool_bp.m`. Here are some important descriptions about these files:

- `convLayer.m` describes the convolutional layer which performs convolution with input data. It consists of two trainable parameters weight **W** and bias **b**. **W** is stored in 4 dimensional matrix with dimensions $k \times k \times n_{in} \times n_{out}$, 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.
- `nnconv.m` 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 to add to each side of the input. Therefore the expected first dimension of output should be equal to $H + 2\times$`pad` $-$ `kernel_size` $+ 1$ and width likewise.
- `poolLayer.m` describes the pooling layer. For simplicity, we **only** need to implement average pooling operation in non-overlapping style. Therefore the expected first dimension of output should be equal to $(H + 2\times$`pad`$)$ / `kernel_size` and width likewise.

Hint: To accelerate convolution and pooling, you should avoid complex nested for loops and use matrix multiplication with MATLAB built-in 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 conduct the summation 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 or stochastic pooling). However, the backpropagation would be very tricky and involve much endeavor to make it work!

4 Report

We perform the following experiments in this homework:

1) plot the loss function value against to every iteration during training
2) construct a CNN with two convolutional layers, and 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 filters of first layer in CNN, refer to caffe filter 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: MATLAB neural network toolbox, theano/caffe framework and any other open source codes are **NOT** permitted in this homework. Once discovered, it shall be regarded as plagiarism.

Deadline: Nov. 2th

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