MNIST Digits Classification with MLP – Softmax Loss

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

1 Background

In this homework, we will continue working on MNIST digits classification problem by utilizing multilayer perceptron (MLP) model. However, the final loss layer is substituted by softmax loss layer and you need to implement it from scratch.

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.

Attention: Considering the time-consuming process of uploading files to Xuetang, we will **NOT** include dataset in the homework distribution files anymore

3 MATLAB Files Description

In the source package, these files including `fcLayer.m`, `Relu.m`, `Sigmoid.m`, `Network.m`, `Fit_mlp.m` and `test_mlp.m` are identical to those included in Homework 1. So you can reuse your implementations in this task.

Attention: If you add any major modifications to these files (like code refactoring or exception handling), please explain them in README file.

A new layer file is `softmaxLossLayer.m`. A softmax layer can be viewed as a mapping from input to a probability distribution in the following form:

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

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. Since the output of softmax layer can be interpreted as a probability distribution, we can compute the data likelihood and its logarithm form is also called cross entropy error function,

$$E = -\ln p(t^{(1)}, \cdots, t^{(N)}) = \sum_{n=1}^{N} E^{(n)} \tag{2}$$

where

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

$$h_k^{(n)} = P(t_k^{(n)} = 1|\mathbf{x}^{(n)}) = \frac{\exp(x_k^{(n)})}{\sum_{j=1}^{K} \exp(x_j^{(n)})} \tag{3}$$

In `softmaxLossLayer.m` you need to implement its forward and backpropagation functions.

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

Attention: The usage of softmax layer can be very flexible by our definition and similar to a nonlinear transform unit (like `Relu` and `Sigmoid`). But by convention its input should be the output of a `fcLayer` and its loss calculation should adopt cross entropy error form, since latter can result in a neat backpropagation formula. It would be a bit challenging to implement a *pure* `softmaxLayer` without specific loss designation.

4 Report

We perform the same experiments in Homework 1:

1) plot the loss function value against to every iteration during training
2) construct a neural network with one hidden layer, and compare the difference of results when using `Sigmoid` and `Relu` as activation function (you can discuss the difference from the aspects of training time, convergence and accuracy)
3) conducting same experiments in 2), but with two hidden layers. Also, compare the difference of results between one layer structure and two layers structure

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: Oct. 19th