CSE474/574: Introduction to Machine Learning(Fall 2015)

Instructor: Sargur N. Srihari
Teaching Assistants: Zhen Xu, Jun Chu, Yifang Liu

Project 3: Classification

Due Date: Wednesday, December 2, 2015 (11:59pm)

1 Overview

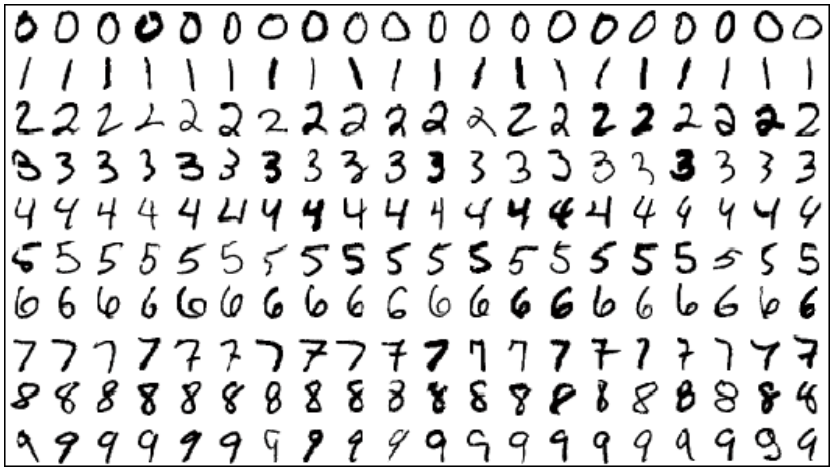
This project is to implement and evaluate classification algorithms. The classification task will be to recognize a 28×28 grayscale handwritten digit image and identify it as a digit among 0, 1, 2, ... , 9. You are required to do the following three tasks.

1. Implement logistic regression, train it on the MNIST digit images and tune hyperparameters (Appendix 1).
2. Implement single hidden layer neural network, train it on the MNIST digit images and tune hyperparameters such as the number of units in the hidden layer (Appendix 2).
3. Use a publicly available convolutional neural network package, train it on the MNIST digit images and tune hyperparameters (Appendix 3).

You can use mini-batch stochastic gradient descent which is described in Appendix 4 for the tasks. Before the submission due date, the autograder can give you feedback about the correctness of your implementation. After the due date, the TAs will test the performance of your models using another USPS dataset which is not available to the students. Different from project 2, students are given more flexibility in their implementation and the grader only tests the final performance.

1.1 Training Data

For the training of our classifiers, we will use the MNIST dataset. The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems.The database is also widely used for training and testing in the field of machine learning.



The database contains 60,000 training images and 10,000 testing images. The dataset could be downloaded from here:

http://yann.lecun.com/exdb/mnist/

The original black and white (bilevel) images from MNIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

1.2 Testing Data

We use USPS handwritten digit as the testing data for this project to test whether your models could be generalize to unseen data. Examples of each of the digits are given below.

Figure 1: Examples of each of the digits

Each digit has 2000 samples available for testing. These are segmented images scanned at a resolution of 100ppi and cropped. This dataset is for grading use and is not available to students.

1.3 Evaluation

Evaluate your solution on a seperate validation dataset using classification error rate

$$E = \frac{N_{\text{wrong}}}{N_{\text{V}}},$$

where N_{wrong} is the number of misclassification and N_{V} is the size of the validation dataset. Under the 1-of-K coding scheme, the input will be classified as

$$C = \arg\max_i y_i.$$

2 Plan of Work

1. **Extract feature values and labels from the data**: Download the MNIST dataset from the Internet and process the original data file into a MATLAB matrix that contains the feature vectors and a MATLAB vector that contains the labels.

2. **Data Partition**: The MNIST dataset is originally partitioned into a training set and a testing set. You will use this partition and train your model on the training set.

3. **Train model parameter**: For a given group of hyper-parameters such as the number of layers and the number of nodes in each layer, train the model parameters on the training set.

4. **Tune hyper-parameters**: Validate the classfication performance of your model on the validation set. Change your hyper-parameters and repeat step 3. Try to find what values those hyper-parameters should take so as to give better performance on the testing set.

3 Deliverables

1. You will submit three files in your submission using CSE submit script:

   (a) MAT-file for logistic regression and neural network implementation `proj3.mat`. It should include the following variables.

   `Wlr`: A $D \times K$ array for the weights for logistic regression.

   `blr`: A $1 \times K$ vector of biases for logistic regression.

   `Wnn1`: A $D \times J$ array for the first layer weights of the neural network. J is the number of nodes in the hidden layer.

   `Wnn2`: A $J \times K$ array for the second layer weights of the neural network

   `bnn1`: A $1 \times J$ array for biases for the first layer the neural network

   `bnn2`: A $1 \times K$ array for biases for the second layer the neural network

   `h`: A string representing the activation function for the neural network. It can be `'sigmoid'`, `'tanh'` or `'ReLu'`.

   This file could be submitted unlimited times before the due date and we will grade the correctness of your model online like in project 1 before the due date. After the due date, we will evaluate your model on the USPS data. This evaluation is conducted only **ONCE**.

   (b) PDF file for project report `proj3.pdf`

   Write a complete project report. The following are some example parts:

   i. Explain your model

   ii. Partition of the training, validation and test sets

   iii. Explain how you choose the parameters and hyper-parameters and their relationships

iv. Discussion of model complexity and performance

v. Compare the performances of different solutions

Use graphs, tables and so on to elaborate your report. For the third task, you are required to visualize the weights you learn in the convolutional neural network and include it in your project report.

(c) Your script used to generate the final version MAT-file `proj3.m`. This script should be able to generate `proj3.mat` given MNIST data.

2. Submit a hard copy of your project report before the end of the first class after the due date.

4 Due Date and Time

The due date is **11:59PM, Dec 2**. Hard copy of your project report must be handed in before the end of the first class after the due date.

Appendix 1 Logistic Regression

Suppose we use 1-of-K coding scheme $\mathbf{t} = [t_1, ..., t_K]$. for our multiclass classfication task. Our multiclass logistic regression model could be represented in the form:

$$p(\mathcal{C}_k|\mathbf{x}) = y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{1}$$

where the activation a_k are given by $a_k = \mathbf{w}_k^\top \mathbf{x} + b_k$. The cross-entropy error function for multiclass classification problem seeing a training sample $\mathbf{x}$ would be

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k \tag{2}$$

where $y_k = y_k(\mathbf{x})$. The gradient of the error function would be

$$\nabla_{\mathbf{w}_j} E(\mathbf{x}) = (y_j - t_j)\mathbf{x}$$

You can then use stochastic gradient descent which uses first order derivatives to update

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t - \eta \nabla_{\mathbf{w}_j} E(\mathbf{x})$$

to find the optimum of the error function and find the solution for $\mathbf{w}_j$.

Appendix 2 Single Layer Neural Network

Suppose we are using a neural network with one hidden layer. Suppose the input layers is denoted by x_i and the output is y_k. The feed forward propagation is as follows:

$$z_j = h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)}\right)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + b_k^{(2)}$$

$$y_k = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

where z_j are the activation of the hidden layer and $h(\cdot)$ is the activation function for the hidden layer. You have three choices for the activation function: logistic sigmoid, hyperbolic tangent or rectified linear unit.

We use cross-entropy error function

$$E(\mathbf{x}) = -\sum_{k=1}^{K} t_k \ln y_k$$

where $y_k = y_k(\mathbf{x})$. The backpropagation is done as follows,

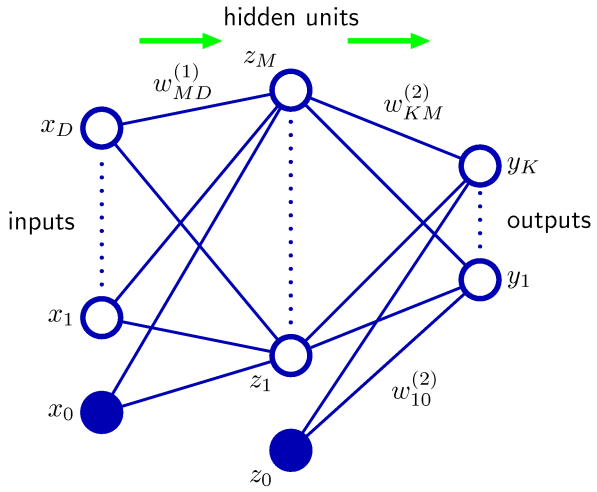


Figure 2: Network diagram for the two- layer neural network

$$\delta_k = y_k - t_k$$

$$\delta_j = h'(z_j) \sum_{k=1}^{K} w_{kj}\delta_k$$

The gradient of the error function would be

$$\frac{\partial E}{\partial w^{(1)}_{ji}} = \delta_j x_i, \qquad \frac{\partial E}{\partial w^{(2)}_{kj}} = \delta_k z_j$$

Having the gradients, we will be able to use stochastic gradient descent to train the neural network.

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} E(\mathbf{x})$$

where $\mathbf{w}$ is all parameters of the neural network.

Appendix 3 Convolutional Neural Network

For the convolutional neural network, you can use packages from online. Therefore, in this project description, we will not go into detail of the implementation. Training of convolutional neural network involves intensive computation, so depending on whether you have GPU available, there are two recommended ways to do it:

1. If you don't have GPU available. In this case, the simplest way would be to take advantage of the computation power of the student server metallica.cse.buffalo.edu. You can use this MATLAB package https://github.com/rasmusbergpalm/DeepLearnToolbox

2. If you do have access to GPU, then there will be various choices for you. There are lots of CNN packages designed for GPU computation, such as Theano, Caffe and so on.

The autograder will not deal with CNN grading. However, instead, in the project report, you need to describe how you choose the structure of CNN and how well the model performs in detail. You are also required to visualize the weights trained by the model.

Appendix 4 Mini-batch stochastic gradient descent

Mini-batch stochastic gradient descent is something between batch gradient descent and stochastic gradient descent. In each iteration of the mini-batch SGD, it samples a small chunk of samples $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m$ from the training data and uses this chunk to update the parameters $\mathbf{w}$:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \sum_{i=1}^{m} \nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$$

The strength of mini-batch SGD compared to SGD is that the computation of $\sum_{i=1}^{m} \nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$ can usually be performed using matrix operation and thus largely out-performs the speed of computing $\nabla_{\mathbf{w}} E\left(\mathbf{z}_i\right)$ individually and updating $\mathbf{w}$ sequentially. However, within same computing time, mini-batch SGD updates the weights much more often than batch gradient descent, which gives mini-batch SGD faster converging speed. The choice of mini-batch size m is the tradeoff of the two effects.

Instead of randomly sampling $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_m$ from the training data each time, the normal practice is we randomly shuffle the training set $\mathbf{x}_1, ..., \mathbf{x}_N$, partition it into mini-batches of size m and feed the chunks sequentially to the mini-batch SGD. We loop over all training mini-batches until the training converges.