HW1 18739

Due Date: Feb 15 Before Class

1 Introduction

In this homework, you will be implementing a simple logistic regression in Python to classify images of hand-written digits in the *MNIST* dataset. In this dataset, each input image is of size 28×28 and reshaped into a size 784×1 vector. The output is a number from 0 to 9 representing the image class. **You will be solving the same problem in three levels of implementation**: First, you will be using *Keras*, which is a high-level neural-network API. Secondly, you will be using *Theano*, which is a popular deep learning library in Python, and finally, you will be implementing the the algorithm from ground up by deriving the gradient formulas yourself using only simple packages such as `numpy`.

You can load the data by running the corresponding section in the starter code. You can find the data file is split into training set, validation set and testing set. The training set is used to train a model and the validation set is used to tune the hyper-parameters used in the model. The test set is only used to report a final score/accuracy. In this problem, we have only two hyper-parameters: learning rate and batch size. Learning rate determines the size of step to take for each gradient update. Batch size determines the amount of data to use in one gradient update step. Number of epochs(1 epoch means going through all of the training data set once) is usually determined using early stopping mechanism, but in this homework you can treat it as a parameter and specify it yourself.

You will be using *Jupyter Notebook* for your homework. It is a web application where you can edit/run code, write text/equations as well as visualize plots, all in one place. You can install and consult the documentation at http://jupyter.org/.

2 Problem Description

Logistic regression is a probabilistic, linear classifier. It is parametrized by a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$. Classification is done by projecting data points onto a set of hyperplanes, the distance to which is used to determine a class membership probability. $\mathbf{X}$ is the input matrix consisting of all the images in one training batch, $\mathbf{X}_i$ is the i-th image represented by the i-th column. $\mathbf{Y}'$ is the ground truth,$\mathbf{Y}$ is the estimation, $\mathbf{L}()$ is the loss function. N is the batch size, (for one batch of 100 images of size 28*28 in 10 classes, the dimensions of **X,Y** are (784,100) and (10,100),respectively). Loss function for a batch is defined as ($\odot$ means elementwise multiplication):

$$\begin{aligned}\mathbf{L}(\mathbf{X},\mathbf{Y};\mathbf{W},\mathbf{b}) &= -\frac{1}{N}\mathbf{Y}' \odot log\mathbf{Y} \\ &= -\frac{1}{N}\sum_j \sum_i \mathbf{Y}'_{j,i} log\mathbf{Y}_{j,i}\end{aligned}$$

for each x, y in **X,Y**:

$$L(x,y;\mathbf{W},\mathbf{b}) = \sum_j y'_j log y_j$$

The conditional probability for each image ($\mathbf{X}_i$) x is calculated using the *Softmax* function:

$$\begin{aligned}y_j = P(y=j|x,\mathbf{W},\mathbf{b}) &= softmax(\mathbf{W}_j^T x + \mathbf{b}_j) \\ &= \frac{e^{\mathbf{W}_j^T x + \mathbf{b}_j}}{\sum_k e^{\mathbf{W}_k^T x + \mathbf{b}_k}}\end{aligned}$$

Now we have the relationship between **L** and **W**,**b**, by using the chain rule, we can derive the gradient $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{L}}{\partial \mathbf{b}}$. Most deep learning software calculates the gradients automatically for you. In Level 3 of this homework, you will need to derive it by yourself. After deriving the gradient values, we can perform stochastic gradient descent to find the optimal **W** and **b**.
After you find the optimal **W** and **b**, you can predict on unseen data :

$$y_{pred} = \text{argmax}_i P(y=i|x,\mathbf{W},\mathbf{b})$$

3 Before You Start

3.1 Software requirement

You should install Python(2 or 3), `numpy`, `gzip`, `keras`, `Theano` and `matplotlib` by either using `pip`, `Anaconda` or `Docker`;

3.2 Dataset and Loading

The dataset is included in `hw1.zip`. The starter code for loading the dataset is included in `hw1.ipynb`. It loads the dataset into three separate sets: training set, validation set and test set.

4 Level 1 - Keras Implementation(10 pts)

Keras is a high level neural network API which runs on top of Tensorflow, CNTK or Theano backend. Before you import Keras, make sure you also have one of these backends installed. You can choose among these backends if you have more than one installed.

4.1 Implementation

Complete the code in corresponding section of `hw1.ipynb`, you may want to to consult the documentation for Keras on https://keras.io/. In principle, your program should contain the following parts with each no more than a line or two:

- Preprocess your data and initialize your model;
- Add one layer to your model with a specified score/activation function;
- Compile your model with the desired loss function, optimizer and metrics;
- Fit your training data (You can also specify your validation data using the validation set here);
- Predict on the test data and report the test accuracy(the percentage of images correctly predicted);

In this section, you can choose your parameters(batch size, learning rate, epoch) by just playing around with the parameters. Run the code and report the following in a separate markdown cell in the notebook:

- The parameters you chose;
- The test accuracy;
- The run time for training;

5 Level 2 - Theano Implementation(30 pts)

Theano is a deep learning framework which let you to build a computational graph easily. Complete the code in the corresponding section in `hw1.ipynb`, you may want to to consult the documentation for Theano at http://deeplearning.net/software/theano/. In principle, your program should contain the following parts:

- The `LogisticRegression` class with functions `score()` and `test_accuracy()`, you don't have to use the starter code, though.
- In your `main()` function, you will implement the following:
  - Initialize symbolic variables;
  - Initialize cost function, score function using the symbolic variables;
  - Define gradients and update rules;
  - Define your training function;
  - Train your model and report your accuracy on the test set.
- Note:
  - You should make use of Theano's "numpy-like" functions such as `theano.tensor.log()` or `theano.tensor.nnet.softmax()` and have as much of your operations using Theano operations and data structures as possible.
  - You program in this section should be self contained and independent from other sections;
  - You don't have to use the validation set for this section, you may use the same parameters you used in section I.
- Report the parameters of choosing, test accuracy and run time for training in a separate markdown cell.

6 Level 3 - Python Implementation(55 pts)

In this section, you are going to solve the same problem using only `numpy`. You will actually derive the gradient expressions and hard code it into the program. Also, you are going to use the validation set to select the best learning rate. You can choose your own epoch and batch size.

6.1 Derivation(20 pts)

Derive the Gradients for $\mathbf{W}$ and $\mathbf{b}$: $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ and $\frac{\partial \mathbf{L}}{\partial \mathbf{b}}$, include all the necessary computation steps. Make sure your the dimensions of your vectors are consistent with the ones in the problem description.

6.2 Implementation(35 pts)

You program in this section should be self contained and independent from other sections; Please don't make use of any functions other than `numpy` functions. (No Keras or Theano function is allowed in this section.) Since your gradient computation is based on $\mathbf{Y}$ being a categorical matrix, you will need to convert the labels in $\mathbf{Y}'$. For example, label 8 will be converted into $[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$.

6.2.1 SGD(20 pts)

Complete functions `softmax()`, `sgd()` and `evaluate()` based on your derivations. Again, you don't have to use the starter code at all and feel free to write helper functions.

6.2.2 Parameter Tuning(15 pts)

In this section, you will find the best learning rate by using the validation set. Train the model using each learning rate in vector $Lr = [10^{-10}, 10^{-9} \cdots, 10^{2}]$ and evaluate the model using the validation set. Plot the validation accuracy against learning rate. Report the following in a separate markdown cell:

- Best learning rate and the corresponding test accuracy using the test set.
- Your run time for training using that learning rate.

7 Comparison(5 pts)

Compare the accuracy, run time among the three implementations. Of the two deep learning softwares(Keras and Theano), what are the pros and cons of each?

Submission

You have to submit the files according to the following procedures:

1. Put your derivation of 6.1 into a single pdf file ⟨your_andrew_id⟩_hw1_derivation.pdf. It is strongly suggested that you use Latex.
2. Rename the program file (hw1.ipynb) as ⟨your_andrew_id⟩_hw1.ipynb.
3. Make sure that you have run all your program in all the cells before you submit so that the results/plots can be seen by simply opening the file. In other words, the grader won't have to run the program to see your results.

Put these two files into ⟨your_andrew_id⟩_HW1 folder, zip the folder into ⟨your_andrew_id⟩_HW1.zip and submit the zip file on Canvas.