ECE521: Inference Algorithms and Machine Learning
University of Toronto
Assignment 2: Logistic Regression and Neural Network
Due date: February 15, at Noon

1 Introduction

The purpose of this assignment is to investigate the performance of logistic regression and neural networks in classifying different letters in a new dataset. In this assignment, you will gain some experience in training a neural network, and also you are going to use an effective way to avoid overfitting in neural networks. All of the implementations should be in Python, NumPy and TensorFlow.

2 notMNIST Dataset

The dataset that you will use in this assignment is a permuted version of notMNIST[1], which contains 28-by-28 images of 10 letters (A to J) taken from different fonts. This dataset has 18720 instances, which you should divide into different sets for training and testing. In some questions you need to divide them into training, validation and testing sets. The provided file is in **.npz** format which is a Python format. You can load this file as follows.

```
with np.load("notMNIST.npz") as data:
    images, labels = data["images"], data["labels"]
```

3 Logistic Regression (Softmax)

In the first part of the assignment, you will use logistic regression for classifying the letters. Training targets are integer numbers from 0 to 9, so you should implement softmax with 10 outputs.

[1]http://yaroslavvb.blogspot.ca/2011/09/notmnist-dataset.html

Task 1 : Classification Error using LR

Implement the logistic regression learning algorithm. Use softmax for the output layer and log-likelihood as your objective function that should be maximized. For this section, you should divide your dataset into 3 different sets: the first 15000 images for training, the next 1000 instances for validation and the remaining part for testing. Train your model without any regularization using stochastic gradient descent with momentum. Adding momentum is a useful technique for speeding up training and preventing the cost function from oscillation. The momentum level is a hyperparameter for your system and you should come up with a good value for it. Plot the log-likelihood of the training data and the log-likelihood of the validation data vs the number of epochs. Also, plot the number of training errors and validation errors vs the number of epochs. When do you think is the best time to stop training? Using that model, compute the number of test errors.

4 Neural Network

In this part, we use a neural network to classify the letters in the dataset. Again, divide your dataset into 3 different sets: the first 15000 images for training, the next 1000 instances for validation and the remaining part for testing. In all of the tasks, use RELU as your activation function for the hidden units, and softmax for the output layer. Use log-likelihood as your training objective. The neural network described below should take about an hour to train on an Intel core i7 CPU @1.73 GHz with 4GB RAM.

Task 2: Neural Network Training

Implement a simple neural network with one hidden layer and 1000 hidden units. Train your neural network with the aforementioned characteristics. For training your network, you are supposed to find a reasonable value for your learning rate. To pick a good value for a hyperparameter, such as the learning rate, you should train your neural network using different values of the hyperparameters (e.g., learning rate) and choose the configuration that gives you the best validation log-likelihood. You'll need to play around with the best range, by watching the training curve, and then run full experiments for 5 different learning rates. You may also find it useful to decay the weights as training procedure goes on. Plot the training and validation log-likelihood vs the number of epochs and also plot the number of training and validation errors. Select the best model using the validation log-likelihood and report the number of test errors for that model. In these experiments, make sure to use early stopping to help avoid overfitting.

Task 3: Number of Hidden Units

Instead of using 1000 hidden units, train different neural networks with 100, 500 and 1000 hidden units. Use the validation data to select the best model for each network size and then the best model overall. Use it for classifying test cases and report the number of test errors. In one sentence, summarize your observation about the effect of number of hidden units in the final results.

Task 4: Number of Layers

For this task, train a neural network with two layers of 500 hidden units. Plot the training and validation log-likelihood vs the number of epochs and also plot the number of training and validation errors. Select the best model and report the number of test errors. How does this compare to the one-layer architecture?

Task 5: Dropout

Dropout is a powerful technique for avoiding overfitting and obtaining better overall performance of a neural network. Using the same architecture as in Task 2, use dropout in the hidden layer of neural network (dropout rate of 0.5) and train you neural network. For forward propagation during testing and validation, don't use dropout, but divide the activity of each hidden unit by two, before forward propagation to the next layer, as described in class. Plot the training and validation log-likelihood vs the number of epochs and repeat for the number of training and validation errors. Select the best model and report the number of test errors. Compare the results with the case of no dropout. In one sentence, summarize your observation about the effect of dropout.

Task 6: Crowd-Sourcing for Hyperparameter Search

Finding a good set of hyperparameters is a critical step for getting a good result. Searching over a large number of hyperparameter settings is computationally intensive and can be done efficiently in parallel using GPUs[2]. If you have a GPU repeat the above questions while trying out a large number of hyperparmaeter settings and report the results.

Many students won't have access to a GPU, so we'll use the entire class for parallel computing, or crowd-sourcing the hyperparameter search. Each group should try as many hyperparameter settings as they can (at least 5 though), and the settings should be picked randomly to as to avoid different groups using the same settings. To do this, seed your random number generator to the time of your experiment and then randomly sample the **log of learning rate** uniformly between -4 and -2, the **number of layers** from 1 to 3, and the **number of hidden units per layer** between 100 and 500. Also, randomly choose your model to use **dropout** or not. Using these hyperparameters, train your neural network and use the validation data to pick a model. Then compute the number of validation errors and the number of test errors. Repeat this procedure as many times as you can, but at least 5 times. For each of these experiments, you should report the hyperparameter settings, the validation errors and the test errors. We'll combine these to obtain a crowd-sourced prediction.

[2]http://www.nvidia.ca/object/what-is-gpu-computing.html