Homework 0

Part 1: Getting Started with ECE6504

In this course, we will be using python considerably (most assignments will need a good amount of python).

Anaconda

Although many distributions of python are available, we recommend that you use the Anaconda Python (https://store.continuum.io/cshop/anaconda/). Here are the advantages of using Anaconda:

- Easy seamless install of python packages (http://docs.continuum.io/anaconda/pkg-docs) (most come standard)
- It does not need root access to install new packages
- Supported by Linux, OS X and Windows
- Free!

We suggest that you use either Linux (preferably Ubuntu) or OS X. Follow the instructions here (http://docs.continuum.io/anaconda/install) to install Anaconda python. Remember to make Anaconda python the default python on your computer. Common issues are addressed here in the FAQ (http://docs.continuum.io/anaconda/faq).

TODO

Install Anaconda python.

Python

If you are comfortable with python, you can skip this section!

If you are new to python and have sufficient programming experience in using languages like C/C++, MATLAB, etc., you should be able to grasp the basic workings of python necessary for this course easily.

We will be using the Numpy (http://www.numpy.org/) package extensively as it is the fundamental package for scientific computing providing support for array operations, linear algebra, etc. A good tutorial to get you started is here (http://cs231n.github.io/python-numpy-tutorial/). For those comfortable with the operations of MATLAB, this (http://sebastianraschka.com/Articles/2014_matlab_vs_numpy.html) might prove useful.

For some assignments, we will be using the IPython notebook (http://ipython.org/notebook.html). IPython is a command shell for interactive computing developed primarily for python. The notebook is a useful environment where text can be embedded with code enabling us to set a flow while you do the

assignments. If you have installed Anaconda and made it your default python, you should be able to start the IPython notebook environment with:

TODO

```
ipython notebook
```

The IPython notebook files have `.ipynb` extension which you should be able to open now by navigating to the right directory.

Part 2: Starting homework 0

This homework is a warm up for the rest of the course. As part of this homework you will:

- Implement a Multi-Class Support Vector Machine (SVM)
    - vectorized loss function **4 points**
    - vectorized gradient computation **4 points**
- Implement Softmax Regression (SR)
    - vectorized loss function **4 points**
    - vectorized gradient computation **4 points**
- Implement Stochastic Gradient Descent **2 points**
- Tune the hyper parameters using Spearmint **2 points**

You will train the classifiers on images in the CIFAR-10 dataset (http://www.cs.toronto.edu/%7Ekriz/cifar.html). The CIFAR-10 is a toy dataset with 60000 images of size 32 X 32, belonging to 10 classes. You need to start with `svm.ipynb` first to implement the SVM and then go ahead with `softmax.ipynb` to implement logistic regression.

This homework is based on assignment 1 (http://cs231n.github.io/assignment1/) of the CS231n course at Stanford.

TODO

Download the starter code here (https://github.com/batra-mlp-lab/VT-F15-ECE6504-HW0/archive/1.0.zip).

Getting the dataset

Make sure you are connected to the internet. Navigate to the `f15ece6504/data` folder and run the following:

TODO

```
./get_datasets.sh
```

This script will download the python version of the database for you and put it in `f15ece6504/data/cifar-10-batches.py` folder.

Getting Spearmint

As part of this homework, you will be using Spearmint to tune the hyper-parameters like learning rate, regularization strength, etc. Spearmint is a software package to perform Bayesian optimization. It is designed to automatically run experiments in a manner that iteratively adjusts a number of parameters so as to minimize some objective in as few runs as possible.

If you have Anaconda installed, setting up Spearmint should be pretty straightforward. You can find installation and usage instructions here (https://github.com/HIPS/Spearmint). You need to use the command line interface to work with Spearmint. To get an idea of how Spearmint works, look up the examples (https://github.com/HIPS/Spearmint/tree/master/examples/simple) provided in the `Spearmint/examples` folder.

TODO

Install Spearmint. Follow instructions in the IPython notebook.

Deliverables

Run the `collectSubmission.sh` script and upload the resulting zip file.

Part 3: SVM and Logistic Regression

As you might already know, both SVM and SR classifiers are linear models but they use different loss functions (hinge loss in SVMs vs softmax loss in SR). Here is a brief summary of the classifiers and if you need a detailed tutorial to brush up your knowledge, this (http://cs231n.github.io/linear-classify/) is a nice place.

Before we go into the details of a classifier, let us assume that our training dataset consists of instances $x_i \in \mathbb{R}^D$ of dimensionality D. Corresponding to each of the training instances, we have labels $y_i \in 1, 2, \ldots, K$, where K is the number of classes. In this homework, we are using the CIFAR-10 database where $N = 50,000$, $K = 10$, $D = 32 \times 32 \times 3$ (image of size 32×32 with 3 channels - Red, Green and Blue).

Classification is the task of assigning a label to the input from a fixed set of categories or classes. A classifier consists of two important components:

Score function: This maps every instance x_i to a vector p_i of dimensionality K. Each of these entries represent the class scores for that image. Both SVM and Logistic Regression have a linear score function given by:

$$p_i = f(x_i; W, b)$$

where,

$$f(x; W, b) = Wx + b$$

Here, W is a matrix of weights of dimensionality $K \times D$ and b is a vector of bias terms of dimensionality $K \times 1$. The process of training is to find the appropriate values for W and b such that the score corresponding to the correct class is high. In order to do this, we need a function that evaluates the performance. Using this evaluation as feedback, the weights can be updated in the right 'direction' to improve the performance of the classifier.

We make a minor modification to the notation before proceeding further. The bias term can be incorporated within the weight matrix W making it of dimensionality $K \times (D + 1)$. The i^{th} row of the weight matrix W is represented as a column vector w_i so that $p_i^j = w_j^T x_i$. The superscript j denotes the j^{th} element of p_i, the score vector corresponding to x_i.

Loss function: This function quantifies the correspondence between the predicted scores and ground truth labels. The loss of the SVM is given by:

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \left[\max\left(0, p_i^j - p_i^{y_i} + \Delta\right) \right]$$

Here, Δ is the margin. The loss function penalizes when the correct class is not greater than all the other scores by at least Δ. The loss of the Logistic Regression is given by:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log\left(\frac{e^{p_i^{y_i}}}{\sum_j e^{p_i^j}}\right)$$

If the weights are allowed to take values as high as possible, the model can overfit to the training data. To prevent this from happening a regularization term $R(W)$ is added to the loss function. The regularization term is the squared some of the weight matrix W. Mathematically,

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

The regularization term $R(W)$ is usually multiplied by the regularization strength λ before adding it to the loss function. λ is a hyper parameter which needs to be tuned so that the classifier generalizes well over the training set.

The next step is to update the weight parts such that the loss is minimized. This is done by Stochastic Gradient Descent (SGD). The weight update is done as:

$$W := W - \eta \nabla L$$

Here, ∇L is the gradient of the loss function and the factor η is the learning rate. SGD is usually performed by computing the gradient w.r.t. a randomly selected batch from the training set. This method is more efficient than computing the gradient w.r.t the whole training set before each update is performed.

References:

1. CS231n Convolutional Neural Networks for Visual Recognition (http://cs231n.stanford.edu)