Assignment 1: Machine Learning Basics
MA-INF 2313: Deep Learning for Visual Recognition

Due Date Theoretical: 27.10.2017
Due Date Programming: 03.11.2017
Assistant: Fadime Sener `sener@cs.uni-bonn.de`

1 Theoretical Exercises *(15 pts)*

1. *(5 pts)* The bias of an estimator is defined as

$$\text{Bias}(\hat{\theta}_m) = E(\hat{\theta}_m) - \theta,$$

where $E(\hat{\theta}_m)$ is the expectation over data and θ is the true value. When the difference between the expectation and true value is zero, the estimator is unbiased.

We define two types of sample variances $\hat{\sigma}_m^2$ and $\tilde{\sigma}_m^2$

$$\hat{\sigma}_m^2 = \frac{1}{m}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2 \qquad \text{and} \qquad \tilde{\sigma}_m^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2.$$

For a 1-D Gaussian distribution, show whether these two sample variances are biased or unbiased. *Hint:* $\mathbb{E}\left[\sum_{i=1}^{m}\left(x^{(i)} - \hat{\mu}_m\right)^2\right] = (m-1)\cdot\sigma^2$.

2. *(5 pts)* The *mean squared error* (MSE) of an estimator is defined as

$$\text{MSE}(\hat{\theta}_m) = \mathbb{E}\left[\left(\hat{\theta} - \theta\right)^2\right].$$

The MSE measures the expected squared "deviation" between the estimator and the true parameter value and is a good measure for the generalization error.

Derive the bias variance trade-off from the definition of MSE.

3. *(5 pts)* The *Maximum A Posteriori* (MAP) estimator is defined as

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\; p(\boldsymbol{\theta}|\boldsymbol{X}).$$

By treating $\boldsymbol{\theta}$ as a random variable and applying Bayes Rule, we get

$$\boldsymbol{\theta}_{\text{MAP}} = \underset{\boldsymbol{\theta}}{\text{argmax}}\; \left[\sum_{i=1}^{m}\log p(x^{(i)}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\right].$$

The MAP estimate can also be applied to the conditional likelihood $p(\boldsymbol{Y}|\boldsymbol{X},\boldsymbol{\theta})$. Derive MAP of conditional likelihood based on Bayes' rule.

2 Programming Exercises *(15 points)*

In this assignment, you will implement, train, test and report the performance of a k-Nearest Neighbor(kNN) classifier and a linear classifier with the logistic function. You will use the MNIST database[1], which has become a benchmark for testing a wide range of classification algorithms. Please download the following 4 files:
* train-images-idx3-ubyte.gz
* train-labels-idx1-ubyte.gz
* t10k-images-idx3-ubyte.gz
* t10k-labels-idx1-ubyte.gz

1)K-Nearest Neighbors (kNN) Classifier *(8 points)*

K-Nearest Neighbors (kNN) classifiers are very simple machine learning algorithms which is often used as a benchmark for more complex algorithms. kNN requires no work at training time, only stores all training data and their labels. At test time, for a given test data, it finds the k closest training data points according to some distance metrics and predicts a class label based on majority voting. The kNN algorithm has two hyper-parameters, k and the distance metric. k is the number of closest training data points and the distance metric is a function that is used to compute the similarity between pairs of data points. In this exercise, you will use the Euclidean distance as the distance metric for your kNN classifiers and cross validate the value of k.

1. *(3 points)* In the .zip file, you are provided with two files, `hw1_demo_knn.py` and `hw1_knn.py`. Implement the empty functions in `hw1_knn.py` by following the instructions given in `hw1_demo_knn.py`. Please make sure that your implementations are working correctly. (Remember that all the data points in the training set are compared with the data in the test set, it will be highly efficient implementing vectorized functions in `hw1_knn.py`)

2. The MNIST dataset consists of 28x28 pixel images of handwritten digits, there are 60000 training images and 10000 test images. In this part of the exercise, you are asked to create a new file with name `hw1_main_knn.py` and implement following tasks.

   (a) *(0.5 point)* In `hw1_demo_knn.py` you are given a subset of the dataset with two classes 0 and 1. Following the instructions given in `hw1_demo_knn.py` load data for "ALL CLASSES" (60000 training images and 10000 test images).

   (b) *(2 points)* Construct a training set with 1000 examples by randomly selecting from the training set, 100 samples per digit class. Report your accuracy for $k = 1$ and $k = 5$ neighbors. Visualize 1 and 5 nearest neighbor images for the first 10 test

[1] `http://yann.lecun.com/exdb/mnist/`

samples. Plot your confusion matrix to see which classes are mixed up with each other. (You can use sklearn.metrics library to compute the confusion matrix)

(c) *(1.5 points)* Implement a function which uses 5-fold cross-validation, on the training dataset to test the accuracy of a kNN classifier on the test set. Test your functions for k neighbors $k = 1, 2, ..., 15$ and plot these accuracies. Which k is performing best?

(d) *(1 point)* Instead of using 100 examples per digit class, use all images in the training set (60000 samples). Report your observations for $k = 1$ neighbor and the k neighbors which performed best in your cross validation evaluation. How much did your functions' computational cost of classifying the test data increased? How well did your classifier perform on the test set?

2)Linear Classifier with Logistic Function *(7 points)*

In this task you will implement a linear classifier. In this classification model we have following linear mapping :

$$h(X, W, b) = \sigma(WX + b)$$

where where X is a D-dimensional input vector, W is a D-dimensional weight vector called the weights, b is a scalar called the bias vector, and $\sigma(.)$ is the Sigmoid function

$$h(a) = \sigma(a) = \frac{1}{1 + exp^{-a}}$$

The classification rule for the classifier for a given input vector x is

$$y^{'} = \begin{cases} 1 & \text{if } h(X) > T \\ 0 & \text{otherwise.} \end{cases}$$

where 1 denotes the positive label and 0 the negative label. T is a threshold value.

To train the classifier you will use $L2$ loss function :

$$L(w, b) = \sum_{i=1}^{N} (y_i - h(x_i, W, b))^2$$

1. *(3 points)* Derive the partial derivatives of the loss function with respect to w and b.
   (a) $\frac{\partial}{\partial w} L(w, b) = ?$
   (b) $\frac{\partial}{\partial b} L(w, b) = ?$

2. *(3 points)* In the .zip file, you are provided with two files, `hw1_demo_linear.py` and `hw1_linear.py`. You should implement the empty functions in `hw1_linear.py` by following the instructions given in `hw1_demo_linear.py`.

   In the `predict()` function you will implement the $h(x, W, b)$ linear mapping. In `sigmoid()`, you are required to implement the Sigmoid function. In `l2loss()` you will implement the $L2$ loss $L(w, b)$ and it should return three items: the loss value given the current W, the gradient with respect to W and the gradient with respect to b. The function `train()` should perform gradient descent with a constant stepsize.

3. *(1 points)* Plot your loss function for t=1000 iterations.