Assignment 1 – CMPUT 328

(Updated September 25th)

A. Assignment 1:

1. Linear Regression with Hinge Loss (Support Vector Machine) – 60%:

Implement **Multiclass** Linear Regression with Hinge Loss on the MNIST dataset in Tensorflow using **Stochastic Gradient Descent**.

Binary Classification case: The loss function Linear Regression with Hinge Loss for **binary classification** is defined as:

$$L(\omega; D) = \frac{1}{2}\|\omega\|_2^2 + C\sum_{i=1}^{N}\max(0, 1 - y_i(\omega^T x_i + b))$$

where:
$D = \{(x_i, y_i)\}_{i=1}^{N},\ x_i \in R^d$ and $y_i \in \{-1, +1\}$
D is your data with x_i being one image vectorized and y_i being its binary label.
ω and b are the parameters that your algorithm has to learn.
N Is the number of data points in a batch.
The first term $\frac{1}{2}\|\omega\|_2^2$ in the above loss function is the **Regularization Loss** and the second term $C\sum_{i=1}^{N}\max(0, 1 - y_i(\omega^T x_i + b))$ is called the **Hinge Loss.** C is a hyperparameter that is used to control the strength of the Hinge loss. This C hyperparameter is equivalent to inverse of the term $\frac{\gamma}{2}$in your Linear Regression lecture:

$$L = \frac{1}{2}\sum_{i=1}^{n}(y_i^p - y_i)^2 + \frac{\gamma}{2}\sum_{j=1}^{m}\theta_j^2$$

Minimizing this Hinge Loss with Regularization is equal to finding maximum margin hyperplanes that divide your data and is used in the classical linear Support Vector Machine algorithm.
The term $\omega^T x_i + b$ is called the **score** of the positive class. If this score for a test example is positive, we classify it as positive, otherwise as negative.

Multiclass case: To do classification on the MNIST dataset which has 10 classes, you would have to extend the above binary classifier to multiclass case. In this assignment, we will use a **one-vs-all** scheme for multiclass case. That is, for each class, we build a binary classifier that classify if a data point belong to this class or belong to the rest. In test time, we run the binary classifiers of **all** classes and choose the class that has the **highest score** output $\omega^T x_i + b$ from its binary classifier as the predicted class. Binary classifier for each class will have their own respective parameters ω and b. In summary, you will have 10 Loss functions, 10 ω, 10 b and 10 scores for 10 classes.

Stochastic Gradient Descent: You need to use stochastic gradient descent to find the parameters for this assignment. You should use a Tensorflow Optimizer (tf.train.GradientDescentOptimizer, tf.train.AdamOptimizer...) to do the gradient derivation and back-propagation automatically for you like in the tutorials on e-class. To obtain good performance on this algorithm, you will probably need to do grid search on the hyperparameters like C, the learning rate and the batch size.

NOTE: A correct implementation of this algorithm using the tf.train.GradientDescentOptimizer

optimizer will have an accuracy of around 88% to a little bit more than 91% on the **validation** set when trained for 50 epochs.
BONUS POINTS: If you can derive the gradients of the loss function L with respect to parameters ω and b in this algorithm by yourself, you will get additional 5% bonus for this assignment. If you do, write the derivation in a file called "**report.pdf**".

ADDITIONAL MATERIALS: For this assignment, you just need to identify what's the parameters in the loss function and how to express that loss as Tensorflow expression the use a Tensorflow Optimizer to train it. However, if you need help with understanding SVM, here are some resource:
https://www.cs.utah.edu/~piyush/teaching/13-9-print.pdf (slide 3 and 14)
https://en.wikipedia.org/wiki/Support_vector_machine#Sub-gradient_descent

2. Linear Regression with Exponential Loss – 20%:
Now, do the same as above but this time for the **Exponential Loss**. The Exponential Loss is used to replace the Hinge Loss and is defined as: $\exp(-y_i(\omega^T x_i + b))$. The total loss including the Regularization Loss will be:

$$L(\omega; D) = \frac{1}{2}\|\omega\|_2^2 + C\sum_{i=1}^{N}\exp(-y_i(\omega^T x_i + b))$$

Note that the Exponential Loss is quite similar to the Hinge Loss and give comparable performance. However, the Exponential Loss is quite vulnerable to number exploding (ie. its value easily go to Infinity). Therefore, you need to choose the initialization of your parameters very carefully and use a reasonably small learning rate. If you see NaNs in your loss value, it's probably mean the learning rate or the initial values of your parameters are too large.

NOTE: A correct implementation of this algorithm using the tf.train.GradientDescentOptimizer optimizer will have an accuracy of around 88% to a little bit more than 90% on the **validation** set when trained for 50 epochs.
BONUS POINTS: If you can derive the gradients of the loss function L with respect to ω and b in this algorithm by yourself, you will get additional 5% bonus for this assignment. If you do, write the derivation in a file called "**report.pdf**".

3. k-nearest neighbors – 20%:
Implement k-nearest neighbors classification algorithm on the MNIST dataset in Tensorflow. The k-nearest neighbor algorithm work as following:

- First, choose a distance function (for example: euclidean distance, Manhattan distance, cosine similarity...) on the input space.
- For every test data point, find k points in the train set that have closest distances to this point. These are called k-nearest neighbors of it. Classification result of the test data point are determined by a majority vote of its neighbor.

You will need to vary value of **k** and choose the distance function to reach optimal performance. Usually, the euclidean distance is good enough for most task.
As this algorithm take a long time to run, the test will be conducted on only 1000 samples from the test set.

NOTE: A correct implementation of this algorithm will have an accuracy of around 94->96% on the **validation** set.
ADDITIONAL MATERIALS: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

B. Submission:

1. Structure of the code directory:

Extract the zip file, you will find a directory called **"student code"**. There are 5 files contained in this directory:

- **main.py**: main file used to call 3 algorithms in this assignment and save the result. You don't need to and should not modify this file. Also, this file will be run to mark your result. After this file run, you will see a file called "**result.txt**" in the project directory which you can open to see the accuracy and run time of your algorithms.
- **svm.py**, **exponential.py**, **knn.py**: 3 files contains the 3 algorithms required in this assignment: multiclass Linear Regression with Hinge Loss, Exponential Loss and k-nearest neighbors. You will need to put your code into the function **run()** in these 3 files.
- **time_test_file.py**: Run this file, it will output a number in seconds. Let call this a unit time. For each algorithm, you code should not take longer than some amount of unit time to finish. Note that, to get the correct unit time, your computer should not run anything else at the time you run this file.
  Limit for this assignment: Each algorithm must finish before **650 unit time**.

2. What to do:

Coding: Implement the 3 algorithms described in section A into each file svm.py (for linear regression with hinge loss), exponential.py (for linear regression with exponential loss) and knn.py (for k-near neighbors). To be specific, you need to put your algorithms into the **run()** function in each file:

```
def run(x_test):
    # TODO: Write your algorithm here
    raise NotImplementedError
    predicted_y_test = []
    return predicted_y_test
```

Updated September 25th: To be specific, you must train your algorithm in the body of this function using the MNIST training data. After that, use the trained algorithm to predict the label of x_test (which is a 2d array of size [None, 784]) and return them as predicted_y_test. predicted_y_test should be a list or an 1D numpy array that contain the labels (from 0 to 9) of the digits in x_test in respective order.
You should **NOT** remove the **x_test** parameter of the **run()** function.
You can put more parameters into the run() function by adding arguments with default values into that function. You can do whatever you want with the **run()** function as long as it can be called from the **main.py** file.
One useful trick is to use arguments with default values and implementing a hyperparameters search. After that, put the best hyperparameters set you find into the default values of the **run()** function like this:

```
def run(x_test, param1=best_value_for_param1, param2=best_value_for_param2, etc...):
    # TODO: put your algorithm here

    predicted_y_test = []
    return predicted_y_test


def hyperparameters_search():
    best_hyperparameters_set = <Initial Value>
    best_validation_accuracy = Inf
    for param1 in param1_values:
        for param2 in param2_values:
            ...
                predicted_y_test = run(x_validation, param1=param1, param2=param2,...)
                # compute accuracy on validation set here
                if new_validation_accuracy > best_validation_accuracy:
                    update best validation accuracy
                    update best hyperparameters set


if __name__ == '__main__':
    hyperparameters_search()
```

(You DO NOT need to and SHOULD NOT do put hyperparameters search into the run() function. Doing so will slow down your code considerably!)
Another good practice is to train and validate your algorithms on a small subset of the data first before running them on the whole data set. Also, to make your hyperparameters search not depend on randomness, remember to put **np.random.seed(0)** before each run.

Report: If you are going for the bonus points, you have to write a report in which you describe how did you derive the gradient for relevant parameters in the ***Linear Regression with Hinge Loss*** and ***Linear Regression with Exponential Loss*** section. The report should be saved as "**report.pdf**"

NOTE: Your algorithms are **NOT** allowed to touch the test set in any manner, including using the mean and variance of the test set.

3. What to turn in:

You need to turn in 4 files: **svm.py**, **exponential.py**, **knn.py** and **report.pdf** (if applicable)
Zip all of them into a single zip file and upload to eclass.
Deadline: Oct 13th, 23:55

C. Scoring Rubric:

A.1. Linear Regression with Hinge Loss (Support Vector Machine):
Correct implementation 60% (correct means test set accuracy from **88%** and up), bonus 5%
A.2. Linear Regression with Exponential Loss:
Correct implementation 20% (correct means test set accuracy from **88%** and up), bonus 5%
A.3. k-nearest neighbors:
Correct implementation 20% (correct means test set accuracy from **94%** and up)
If in any part, your accuracy is lower than the threshold, you lose 10% mark of that part for each 1% lower accuracy.
NOTE: If any of your algorithms doesn't finish by the time limit, you get 0% for that part of the assignment.
Late policy: you lose 10% of the total mark that you would get for every late day after due date.