Homework #7

18-461/661: Intro to ML for Engineers
Prof. Gauri Joshi and Prof. Carlee Joe-Wong
Due: Friday, April 24, 2020 at 8:59pm PT/11:59pm ET

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. The full collaboration and grading policies are available on the course website: https://www.andrew.cmu.edu/course/18-661/.

Your solutions should be uploaded to Gradescope (https://www.gradescope.com/) in PDF format by the deadline. We will not accept hardcopies. If you choose to hand-write your solutions, please make sure the uploaded copies are legible. Gradescope will ask you to identify which page(s) contain your solutions to which problems, so make sure you leave enough time to finish this before the deadline. We will give you a 30-minute grace period to upload your solutions in case of technical problems.

All students must complete Q1 on Fashion-MNIST. 18-661 students must also complete either Q2 or Q3. If you complete both, we will count the highest of these grades. 18-461 students only need to complete Q1.

We strongly advise starting early on this assignment. Unlike other homeworks, you can use built-in PyTorch and `sklearn` functions as needed (so you need not write your training algorithms from scratch).

You will be graded based on the correctness of your code, as well as your explanations for the sub-questions, but we are not imposing a minimum accuracy threshold for any of the models. We have provided Code and Write-up tags to help identify what you need to submit.

1 Learning to classify the "classy" digits *[60 points]*

Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

The goal of this task to leverage the power of Machine Learning, coupled with the comfort of PyTorch, to implement an end-to-end Multi-Layered Perceptron based system which can achieve a high classification accuracy on the Fashion-MNIST dataset.

1.1 Building a Vanilla Classifier *[40 points]*

1.1.1 Dependencies

Recommended Configuration: (i) `Python` 3.6 (ii) `PyTorch` 1.3.1 (iii) `Tensorboard` 1.14.0 (iv) `Numpy` 1.16.6 (v) `termcolor` 1.1.0

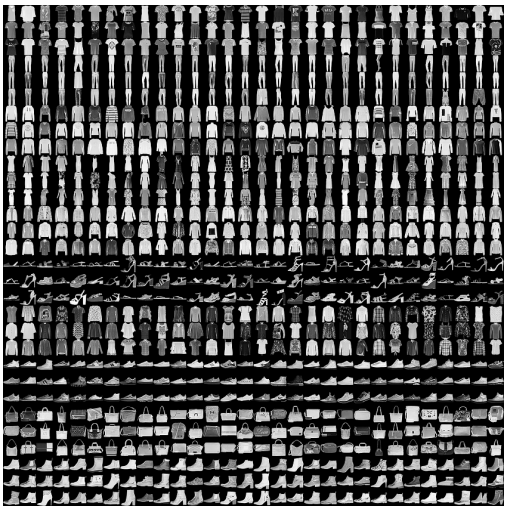

Figure 1: Fashion-MNIST

1.1.2 Starting with the data *[10 points]*

PyTorch provides an effective, yet simple to deal with method to handle large datasets in parallel, using the `torch.utils.data.dataloader` class. The dataloader class, can automatically divide the dataset into batches, while parallelising the tasks amongst multiple processes, To create a dataloader, first you have to create a "Dataset" class (inheriting the `torch.utils.data.Dataset` class), and overriding the `__len__` and `__getitem__` methods. **Please note that we have provided our own specialized dataset , and thus you can't use the Fashion MNIST loader directly from PyTorch.**

(a) Explain the logic that goes within both the `__len__` and `__getitem__` methods and why they are required for the dataloader. Write-up

(b) In the file `data.py` complete the methods provided, ensuring that the images are scaled to $[0, 1]$. Please explain the steps undertaken for processing the data in the write-up. Please ensure that you use the hyperparameters from the `config` file. Code Write-up

1.1.3 Designing the architecture *[5 points]*

(a) Define the architecture in the class `Network` in `network.py`. Use an architecture containing two hidden layers ($784 \rightarrow 100 \rightarrow 10$) with a `ReLU` activation after the first layer and a `Softmax` after the final logits to get probability scores. Note that the input images are sized 28×28 and thus the input is obtained by flattening the images, making it sized 784. The architecture has been visually delineated in Figure 2

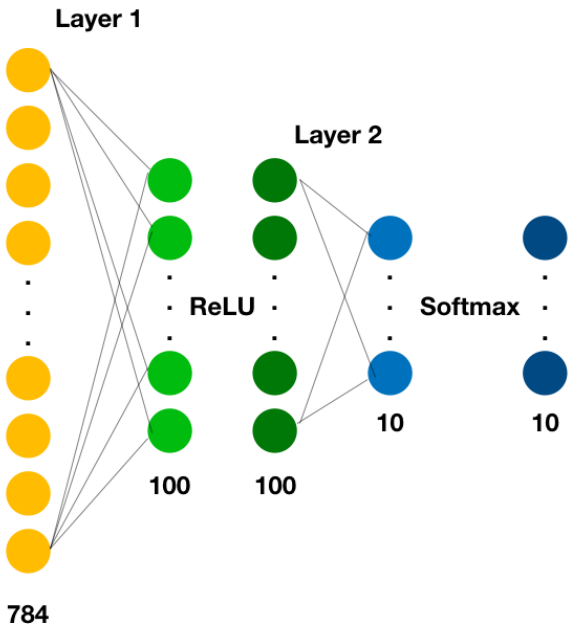

Figure 2: Architecture for Task 1

(b) Write code for saving and loading the model in the `load` and `save` methods of `Network`, which can be used to stop the training in an intermediate epoch and load it later to resume training. Code

1.1.4 A holistic view of the training *[10 points]*

TensorBoard provides the visualization and tooling needed for machine learning experimentation, by exposing an extremely easy to use interface for plotting scalars, images, histograms while training. Plot the following things after every epoch (see below for settings to train the model) using tensorboard and include in your writeup: Code Write-up

(a) Training Loss

(b) Test Accuracy

(c) Visualize the 784 weights incoming into any node in the first hidden layer by reshaping in the shape of images, and do this for 2 any hidden nodes.
[Hint: Think about the weights connecting all the input nodes to one node in the first layer]

(d) Current learning rate

1.1.5 Training the model *[15 points]*

It's finally time to train your model on the specialized Fashion-MNIST dataset. Using the file `train.py`, complete the following functionalities in the code:

(a) Create a `criterion` object, which defines the objective function of our model. Code

(b) Create an `optimizer` **(Stochastic Gradient Descent)**, using the hyperparameters provided via the `config` file. Code

(c) Complete the backpropogation by resetting the gradients, doing a backward pass, and then updating the parameters of your model. Code

(d) Learning Rate Scheduling - Use the `ReduceLROnPlateau` scheduler to anneal the learning rate based on the test set accuracy. Read about the `patience` and `factor` parameters of this scheduler, and explain their role. Use the hyper-parameters from the `config` file Code Write-up

(e) Complete the `run` method, which takes in the network, and runs it over all the data points. Code

(f) Finally, train the model using the hyper-parameters in the `config` file. As a milestone, you should be able to achieve around 75% accuracy on the test set, within the 20 epochs. Include the tensorboard screenshots in the writeup and report the final train loss and test accuracy Write-up

1.2 Improving our Vanilla Classifier *[20 points]*

Complete the following steps **(in order)**. Also, please note that the steps consist of ablation experiments for our training. We will fix the prior choices while performing the next experiment, for our comparisons. Please refer to the TL;DR section for the final set of results to be reported:

(a) Initialize the weights of your Multi-Layered Perceptron in the `init_weights` function using the following strategies:

  (a) All weights and biases zeros

  (b) Using Xaiver Normal initialization for the weights and zeros for the biases

  At the end, report performance on the test set using the two initialization strategies and explain the difference in performance, if any. Code Write-up

(b) Data Augmentation has been a very useful technique for effective training of Deep Learning models. Give two examples of how data augmentation can be useful (In **any** task of your choice). Write-up

(c) Now, think of how we can do data augmentation for our task. Please note that we have modified our dataset, and it is **strongly recommended** to visually inspect both the training and test images. Now train your classifier with data augmentation and report the test accuracy after 20 epochs. Also, explain the augmentation that you have added, and the reason for the change in performance, if any. *[Note: Use the Xavier Normal initialization from the previous experiment]* Code Write-up

(d) Read about Dropout and how it affects training of deep networks. Now, add dropout to your network and include in the write-up the effect of dropout in the performance of your model. Explain the effect of dropout if it causes any, and explain why dropout doesn't help, if it doesn't. *[Note: Use Xavier Normal initialization, and data augmentation from the prior parts]* Code Write-up

TL;DR for reporting results

To make sure you have not missed any part of the question, results for the following experiments have to be reported:

(a) Vanilla Classifier (VC)

(b) VC + Zero weights initialization

(c) VC + Xavier normal weights initialization (XNW)

(d) VC + XNW + Data Augmentation (DA)

(e) VC + XNW + DA + Dropout

2 Decision Tree for Spotify Data *[40 points]*

In this problem, you will implement a Decision Tree classifier using scikit-learn and use it to classify the Spotify data set. In this data set, an individual has generated a list of songs, each with a set of features, and whether the individual liked or disliked the song. The goal of this problem is to create a decision tree classifier to predict whether this individual would like or dislike a song based on a list of features. We encourage you to use `sklearn`'s `DecisionTreeClassifier` class for this problem.

2.1 Import Data *[10 points]*

(a) Import the data into a Pandas dataframe. Pandas is a data analysis library that is very useful for machine learning projects. Examine the data. Which features, if any, appear to not be useful for classification and should be removed? Print the final list of the feature names that you believe to be useful. Code Write-up

(b) Of the remaining features which you believe may be useful for classification, which feature(s) do you estimate will be the most important? Which feature(s) will be the least important? Briefly explain your answers. Write-up

(c) Create a Pandas dataframe with just the useful features you have selected, and a separate data series for the targets (labels) of each sample. Code

(d) Divide the full dataset into a training set and testing set, with 80% of the data used for training. Consider using the `train_test_split` function for this step. Code

2.2 Training the Model *[10 points]*

(a) Determine the best hyper-parameters for your decision tree using cross-validation with at least 5 folds. Search across at least 3 hyper-parameters for Decision Trees. It is recommended to look at 'criterion', 'max_depth', and 'class_weight', but you are welcome to explore additional or alternative hyper-parameters. The `GridSearchCV` module may be helpful here. Report which hyper-parameters you searched over and the best hyper-parameter values. Code Write-up

(b) Train your model with the best hyper-parameters found in Q2.2a. Run it on the test data to generate predictions for the test data. Your final accuracy may vary, but expect it to be around 70%. Code

2.3 Evaluating the Model *[20 points]*

(a) Generate the precision, recall, accuracy, and F1-score for your predictions from Q2.2b. These metrics are all refinements of the classification accuracy. The `sklearn.metrics` modules may help with this. What are your results? Code Write-up

(b) Generate a confusion matrix to visualize your predictions (Figure 3 has an example; note that your matrix may have very different values). The `sklearn.metrics` module may also be useful here. Code Write-up

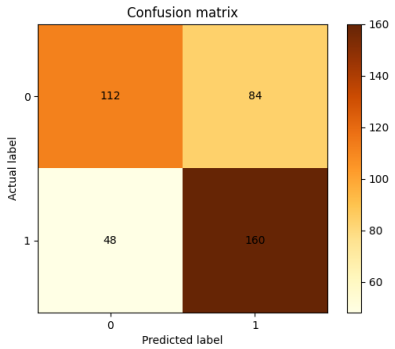

Figure 3: Confusion Matrix Example

(c) Generate a representation of your decision tree from Q2.2b using the `graphviz` and `export_graphviz` functions. Figure 4 shows an example decision tree output (note that your results may look very different from this example). Code Write-up

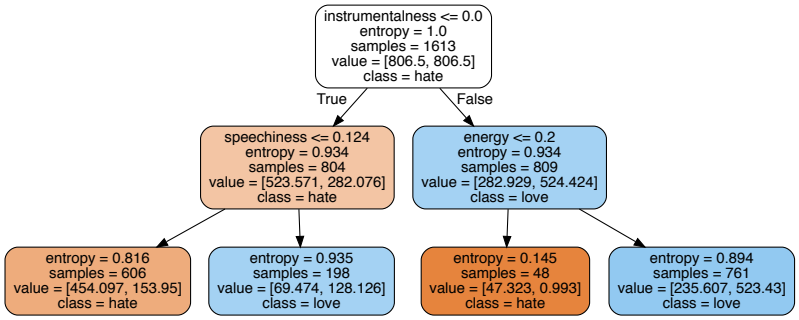

Figure 4: Decision Tree Visualization Example

(d) Determine the relative importance of each feature for the tree you trained in Q2.2b. The importance of a feature is computed as the (normalized) total reduction of the criterion brought by that feature. How well do these results match your initial (qualitative) estimates of each feature's importance in Q2.1a? Print the importance of each feature. Code Write-up

Hint: This quantity is computed as you train your model, and thus should not require additional computations on your part.

3 Clustering for COVID-19 Data *[40 points]*

In this problem you will analyse COVID data and perform clustering using libraries like scikit-learn and Pandas. Two datasets will be used for this problem:

(a) us-states.csv: This dataset contains information about COVID cases and death in each City per date. The relevant columns here are: ['date', 'City', 'cases' and 'deaths'].

(b) statelatlong.csv: This dataset contains information about each City and its corresponding Latitude and Longitude coordinates. The relevant columns here are: ['Latitude', 'Longitude', 'City']

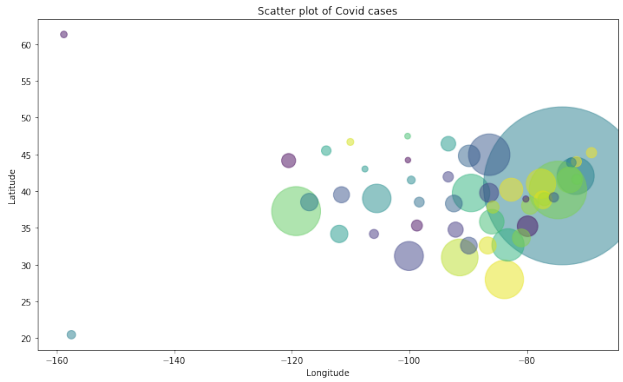

Figure 5: COVID data scatter plot

3.1 Import Data and Plotting *[5 points]*

(a) Import the us-states.csv and statelatlong.csv files into a Pandas dataframe. Merge the two dataframes based on 'City' on the us-states.csv so that the merged data contains latitude and longitude columns for each state, in addition to the COVID case information. Print the first 5 rows of the merged dataframe. Code Write-up

(b) Find all the data rows in your merged dataframe for the date 25th March 2020 and print the first 5 rows. For the subsequent questions below, this subset of data will be used and referred to as the *sampled data*. Code

(c) Make a weighted scatter plot of the sampled data (from part 3.1(b)) where x-axis is Latitude, y-axis is Longitude and the size of each point is scaled according to the number of cases. Attach the scatter plot to your solution PDF. See Figure 5 for an example. (Note: check scatter plots from matplotlib for plotting) Code Write-up

3.2 Geographical Distribution of Cases *[20 points]*

Now we will use K-means clustering (as taught in class) and its variant called weighted K-means to cluster the data. The K-means algorithm divides a set of N samples $\boldsymbol{x}_n$ for $n = 1, \ldots, N$ into K disjoint clusters, each described by the mean $\boldsymbol{\mu}_k$ of the samples in the cluster. The K-means algorithm aims to choose means (centroids) to minimise the inertia, also called the "within cluster sum-of-squares criteria," which is mathematically defined as:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{1}$$

In weighted K-means, each data point has an associated sample weight. This allows us to assign more weight to samples when computing cluster centers and values of inertia. For example, assigning a weight of 2 to a sample is equivalent to adding a duplicate of that sample to the dataset X. Our goal is then to choose the cluster means to minimize a modified version of the inertia (1):

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} w_n ||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2 \tag{2}$$

where w_n is the weight of the n^{th} data point.

Consider three columns of the sampled data (from part 3.1(b)): ['Longitude','Latitude', 'Cases'] for the subsequent questions (You can use the K-means clustering function from `sklearn` to implement both unweighted and weighted K-means clustering). Use kmeans++ initialisation for all the below sub-questions.

(a) **Unweighted K-means clustering by location.** We will first use K-means with two features ['Longitude','Latitude']. An important hyper-parameter in K-means clustering is the number of clusters K. Use the Elbow Method to determine this optimal value of K. Increment the value of K from 1 to 50 with a step size of 1 and plot the K-means loss function versus K where the loss function is given in (1). Include the plot in your answer. Code Write-up

(b) **Unweighted K-means clustering by location.** Take the value of $K = 15$ and perform K-means clustering. In a single plot, make a scatter plot of the data points ('Longitude' (x-axis) vs 'Latitude' (y-axis)) and a scatter plot of the cluster centers. Make sure the data points and centroids have different colors and attach the plot below. Print the names of the states which are clustered together and attach it in the solution PDF. Code Write-up

(c) **Weighted K-means by location.** We will use weighted K-means here, where the two features are ['Longitude','Latitude'] and the weight of each sample corresponds to the ['cases'] feature. Use the Elbow Method to determine this optimal value of K by incrementing the value of K from 1 to 50 with a step size of 1 and plot the weighted K-means loss vs. the value of K, where the loss function is given in (2). Include the resulting plot in your answer. Code Write-up

(d) **Weighted K-means by location.** Take the value of $K = 15$ and perform weighted K-means clustering. In a single plot, make a scatter plot of the data ('Longitude' (x-axis) vs 'Latitude' (y-axis)) and a scatter plot of the cluster centers. Make sure the data points and centroids have different colors and attach the plot below. Print the name of the states which are clustered together and attach it in the solution PDF. Code Write-up

(e) Compare the state clustering obtained from unweighted K-means (b) and weighted K-means (d). Which method do you think provides better-defined clusters and why? Write-up

3.3 Growth Rate Modelling and Distribution Analysis *[15 points]*

In this section we will analyze the growth rate of COVID cases in different states using us-states.csv. For ease of numeric analysis and uniformity, replace the ['date'] column values with that of numbers ranging from 0 to the number of rows in the DataFrame.

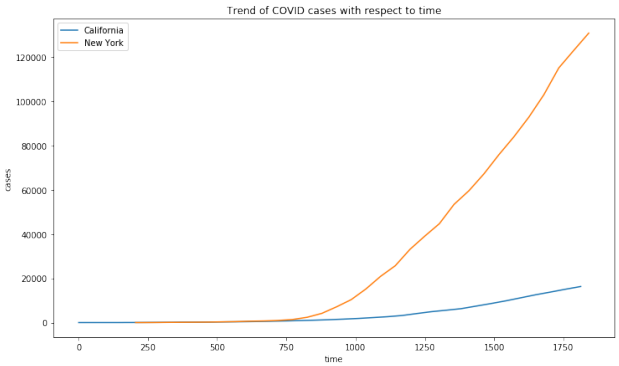

Figure 6: Growth of cases over time

(a) Plot the ['cases'] vs ['date'] for California and New York in a single plot, as shown in Figure 6. Observe that the number of cases grows exponentially with time in both locations. Code

(b) **Exponential modeling.** We will now fit an exponential model $y = Ae^{Bx}$ to model the exponential trend of ['cases'] with respect to time. In other words, we want to find the values of A and B that best fit the data. Realize this model can also be written as $\log y = \log A + Bx$, which reduces the problem to a linear regression task (you can use `np.polyfit` for this task). Find the values of A and B separately for each state and report them in your answer. Code Write-up

(c) **Exponential modeling.** Use the above model $y = Ae^{Bx}$ learnt for the states of California and New York to plot the predicted number of cases versus time. Superimpose your plots of this model on the plot of the actual cases versus time obtained in 3.3(a). Attach the combined plot to your answer. Code Write-up

(d) **K-means clustering by growth rate.** We will use un-weighted K-means here, where the two features are the parameter values [A, B] and each data point corresponds to a location. The clusters will be based here on features [A, B]. We will use the Elbow Method to determine the optimal value of K. Increment the value of K from 1 to 50 with a step size of 1 and plot the un-weighted K-means loss vs the value of K. Attach the obtained plot here. Code Write-up

(e) **K-means clustering by growth rate.** Take the value of $K = 8$ and perform un-weighted K-means clustering. In a single plot, make a scatter plot of the data points (A vs B) and a scatter plot of the cluster centers. Make sure the data points and centroids have different colors and attach the plot to your solution PDF. Print the names of the states which are clustered together and attach them in the solution PDF. Code Write-up

(f) Examine the clusters that you obtained in part (e). Can you explain why particular states are clustered together? Write-up