Homework 4

ECE 461/661: Introduction to Machine Learning for Engineers
Prof. Gauri Joshi and Prof. Carlee Joe-Wong
Due: Wednesday April 16th, 2025 at 8:59PM PT / 11:59PM ET
Tuesday, April 17th, 2025 at 5:59 am CAT

Please remember to show your work for all problems and to write down the names of any students that you collaborate with. Although you are encouraged to work together, **you must write your own solutions, which should reflect your understanding of the problem.** When answering coding questions, you may use packages such as `numpy` for incidental operations like matrix multiplication. However, **you may not use built-in packages that directly implement the specified functions for you**, unless explicitly specified in the question. The full collaboration and grading policies are available on the course website: https://www.andrew.cmu.edu/course/18-661/. You are strongly encouraged (but not required) to use Latex to typeset your solutions.

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.

1 Distributed SGD *[20 points]*

Consider that we have a system of m worker nodes and a parameter server performing distributed SGD (stochastic gradient descent). In each iteration, every worker node receives the model from the parameter server, computes one gradient step of the objective function locally using its local data, and sends the gradient to the parameter server. The parameter server does the aggregation of gradients using either synchronous SGD or asynchronous SGD.

The gradient calculation time X_i taken by each node i follows the exponential distribution with rate $\lambda = 2$, which has the following probability density function (PDF):

$$f_X(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{else} \end{cases} \tag{1}$$

Answer the following questions and make sure to explain your answers:

a. *[2 points]* What is the cumulative distribution function (CDF) of $f_X(x)$, i.e., $F_X(x)$?

b. *[2 points]* Define $X_{m:m}$ as the maximum of m i.i.d. (independently and identically distributed) instances $X_1, ..., X_m$ following the distribution X. What is the CDF of $X_{m:m}$, and what is the expected value $\mathbb{E}[X_{m:m}]$?

c. *[2 points]* Define $X_{1:m}$ as the minimum of m i.i.d instances $X_1, ..., X_m$ following the distribution X. What is the CDF of $X_{1:m}$, and what is the expected value $\mathbb{E}[X_{1:m}]$?

d. *[8 points]* In this sub-problem, we will simulate and compare the expected runtime per iteration of synchronous SGD and asynchronous SGD for different values of m. The time for each worker node to finish one gradient computation is exponentially distributed as given in part (a) with $\lambda = 2$, and it is i.i.d. across workers and iterations. Assume there is no communication delay.

Simulate 5000 iterations of training using Python for different values of m ranging from 1 to 20, and obtain the average runtime per iteration. Make a comparative plot of the average runtimes per iteration of synchronous and asynchronous SGD versus m. Explain the trends observed in the plot in 1-2 sentences. You may use packages inside `numpy.random` to draw random samples from the exponential distribution. Attach your plot and code in PDF format to the end of your homework.

e. *[6 points]* Write down the theoretical expressions for the expected runtimes per iteration of synchronous and asynchronous SGD in terms of m and λ (Hint: You can use the expressions derived in parts (b) and (c)). On the figure generated in part (d), also plot the theoretical expected runtimes versus m. Check whether the theoretical and simulated values align.

(a)

$$F_X(x) = \int_0^x 2e^{-2y}\mathrm{d}y = \begin{cases} 1 - e^{-2x} & \text{for } x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

(b) The CDF of $X_{m:m}$ is:

$$\begin{aligned} F_{X_{m:m}}(x) &= \Pr(\max(X_1, X_2, \ldots, X_m) \leq x) \\ &= \Pr(X \leq x)^m \\ &= F_X(x)^m \\ &= (1 - \exp(-\lambda x))^m \end{aligned}$$

The expected value $\mathbb{E}[X_{m:m}]$ was computed in the lecture slides and it is equal to

$$\begin{aligned} \mathbb{E}[X_{m:m}] &= \frac{1}{m\lambda} + \frac{1}{\lambda(m-1)} + \cdots + \frac{1}{\lambda} \\ &= \frac{H_m}{\lambda} \approx \frac{\log m}{\lambda} \end{aligned}$$

(c) The CDF of $X_{1:m}$ is:

$$\begin{aligned} F_{X_{1:m}}(x) &= 1 - \Pr(\min(X_1, X_2, \ldots, X_m) > x) \\ &= 1 - \Pr(X > x)^m \\ &= 1 - \exp(-\lambda m x) \end{aligned}$$

Observe that $X_{1:m}$ is an exponential random variable with rate λm. Thus, its expected value is:

$$\mathbb{E}[X_{1:m}] = \frac{1}{m\lambda}$$

(d) The simulation results are shown in the plot. We observe that expected runtime of synchronous SGD increases with the number of workers, whereas that of asynchronous SGD decreases with the number of m and it is significantly less than the synchronous SGD runtime.

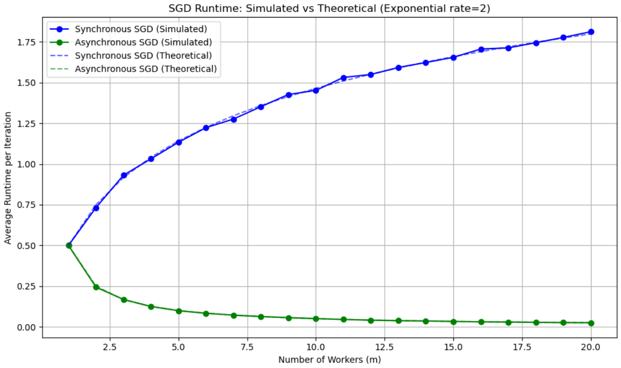

Figure 1: Comparison of synchronous and asynchronous SGD runtimes, both simulation and theoretical results from parts (c) and (d) respectively.

(e) The theoretical expected runtimes per iteration of synchronous and asynchronous SGD are

$$\mathbb{E}[T_{\text{sync}}] = \mathbb{E}[X_{m:m}] = \sum_{k=1}^{m} \frac{1}{\lambda k} \approx \frac{\log m}{\lambda}$$

$$\mathbb{E}[T_{\text{async}}] = \mathbb{E}[X_{1:m}] = \frac{1}{\lambda m}$$

The theoretical results are shown in the plot and they match with the simulation results.

2 K-means *[20 points]*

Given a set of data points $\{\mathbf{x}_n\}_{n=1}^N$, k-means clustering minimizes the following distortion measure (also called the "objective" or "clustering cost"):

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2$$

where $\boldsymbol{\mu}_k$ is the prototype of the k-th cluster and r_{nk} is a binary indicator variable. If $\mathbf{x}_n$ is assigned to the cluster k, r_{nk} is 1, and otherwise r_{nk} is 0. For each cluster, $\boldsymbol{\mu}_k$ is the prototype representative for all the data points assigned to that cluster.

a. *[10 points]* In lecture, we stated but did not prove that $\boldsymbol{\mu}_k$ is the mean of all points associated with the kth cluster, thus motivating the name of the algorithm. You will now prove this statement. Assuming all r_{nk} are known (i.e., assuming you know the cluster assignments of all N data points), show that the objective D is minimized when each $\boldsymbol{\mu}_k$ is chosen as the mean of all data points assigned to cluster k, for any k. This justifies the iterative procedure of k-means[1].

First, note that r_{nk} is simply an indicator function for the k^{th} cluster (if k is fixed). Thus, for any cluster with index k, the loss function is

$$D_k = \sum_{n=1}^{N} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|_2^2$$

[1] More rigorously, one would also need to show that if all $\boldsymbol{\mu}_k$ are known, then r_{nk} can be computed by assigning $\mathbf{x}_n$ to the nearest $\boldsymbol{\mu}_k$. You are not required to do so.

where $D = \sum_k^K D_k$. Assuming the labels are fixed, D_k for all $k \in [K]$ and D are convex in $\boldsymbol{\mu}_k$ for all $k \in [K]$. Thus, we can find the minimizer by taking the derivative with respect to μ_k and setting it to zero.

$$\begin{aligned}
\nabla_{\mu_k} D_k &= \sum_{n=1}^{N} r_{nk} 2(\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0 \\
\sum_{n=1}^{N} r_{nk}\boldsymbol{\mu}_k &= \sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n \\
\boldsymbol{\mu}_k &= \frac{1}{\sum_{n=1}^{N} r_{nk}} \sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n
\end{aligned}$$

This is exactly the mean of the data points that are in cluster k.

b. *[10 points]* As discussed in lecture, sometimes we wish to scale each feature in order to ensure that "larger" features do not dominate the clustering. Suppose that each data point $\mathbf{x}_n$ is a d-dimensional feature vector and that we scale the jth feature by a factor $w_j > 0$. Letting W denote a $d \times d$ diagonal matrix with the j'th diagonal entry being w_j, $j = 1, 2, \ldots, d$, we can write our transformed features as $\mathbf{x}' = W\mathbf{x}$.

Suppose we fix the r_{nk}, i.e., we take the assignment of data points $\mathbf{x}_n$ to clusters k as given. Our goal is then to find the cluster centers $\boldsymbol{\mu}_k$ that minimize the distortion measure

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| W\mathbf{x}_n - \boldsymbol{\mu}_k \right\|_2^2 .$$

Show that the cluster centers $\{\boldsymbol{\mu}_k\}$ that do so are given by $\boldsymbol{\mu}_k = \frac{1}{\sum_{n=1}^{N} r_{nk}} W \sum_{n=1}^{N} r_{nk}\mathbf{x}_n$.

$$D = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| W\mathbf{x}_n - \boldsymbol{\mu}_k \right\|_2^2$$

Again, define $D_k = \sum_{n=1}^{N} r_{nk} \| W\boldsymbol{x}_n - \boldsymbol{\mu}_k \|_2^2$. Assuming the labels are fixed, D_k for all $k \in [K]$ and D are convex in $\boldsymbol{\mu}_k$ for all $k \in [K]$. Thus, we can find the minimizer by taking the derivative with respect to μ_k and setting it to zero.

$$\nabla_{\mu_k} D_k = \sum_{n=1}^{N} -2 r_{nk}(\boldsymbol{W}\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0$$

$$\sum_{n=1}^{N} r_{nk}\boldsymbol{\mu}_k = \sum_{n=1}^{N} r_{nk}\boldsymbol{W}\boldsymbol{x}_n$$

$$\boldsymbol{\mu}_k = \frac{1}{\sum_{n=1}^{N} r_{nk}} W \sum_{n=1}^{N} r_{nk}\boldsymbol{x}_n$$

3 3-Dimensional Principal Component Analysis *[20 points]*

In this problem, we will perform PCA on 3-dimensional data step by step. We are given three data points:

$$\boldsymbol{x}_1 = [0, -1, -2], \boldsymbol{x}_2 = [1, 1, 1], \boldsymbol{x}_3 = [2, 0, 1],$$

and we want to find 2 principal components of the given data.

a. *[8 points]* First, find the covariance matrix $\boldsymbol{C}_X = X^T X$ where $X = \begin{bmatrix} \boldsymbol{x}_1 - \overline{\boldsymbol{x}} \\ \boldsymbol{x}_2 - \overline{\boldsymbol{x}} \\ \boldsymbol{x}_3 - \overline{\boldsymbol{x}} \end{bmatrix}$, where $\overline{\boldsymbol{x}} = \frac{1}{3}(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \boldsymbol{x}_3)$ is the mean of the data samples. Then, find the eigenvalues and the corresponding eigenvectors of $\boldsymbol{C}_X$. Feel free to use any numerical analysis program such as numpy, e.g., `numpy.linalg.eig` can be useful. However, you should explain what you inputted into this program.

Note that $\overline{x}_1 = 1, \overline{x}_2 = 0, \overline{x}_3 = 0$. To center the data, we subtract $\overline{\boldsymbol{x}} = [1, 0, 0]^T$ from each data point. From this we get:

$$C_X = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 2 & 3 \\ 3 & 3 & 6 \end{bmatrix}$$

Characteristic polynomial: $\lambda^3 - 10\lambda^2 + 9\lambda = \lambda(\lambda - 9)(\lambda - 1) = 0$. From this we obtain:

Eigenvalues $= 9, 1, 0$.

Eigenvectors $= \begin{bmatrix} 1/\sqrt{6} \\ 1/\sqrt{6} \\ 2/\sqrt{6} \end{bmatrix}, \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}, \begin{bmatrix} -1/\sqrt{3} \\ -1/\sqrt{3} \\ 1/\sqrt{3} \end{bmatrix}.$

b. *[4 points]* Using the result above, find the first two principal components of the given data.

Principal components $= \begin{bmatrix} 1/\sqrt{6} \\ 1/\sqrt{6} \\ 2/\sqrt{6} \end{bmatrix}, \begin{bmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \\ 0 \end{bmatrix}.$

c. *[8 points]* Now we want to represent the data $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_3$ using a 2-dimensional subspace instead of a 3-dimensional one. PCA gives us the 2-D plane which minimizes the difference between the original data and the data projected to the 2-dimensional plane. In other words, $\boldsymbol{x}_i$ can be approximated as:

$$\widetilde{\boldsymbol{x}}_i = a_{i1}\boldsymbol{u}_1 + a_{i2}\boldsymbol{u}_2 + \overline{\boldsymbol{x}},$$

where $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are the principal components we found in 3.b.. Figure 2 gives an example of what this might look like.

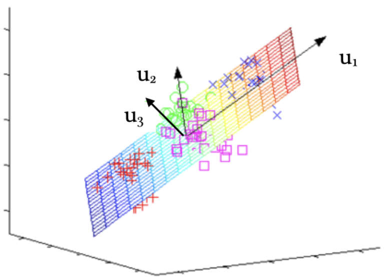

Figure 2: Example of 2-D plane spanned by the first two principal components.

Find a_{i1}, a_{i2} for $i = 1, 2, 3$. Then, find the $\widetilde{\boldsymbol{x}}_i$'s and the difference between $\widetilde{\boldsymbol{x}}_i$ and $\boldsymbol{x}_i$, i.e., $||\widetilde{\boldsymbol{x}}_i - \boldsymbol{x}_i||^2$ for $i = 1, 2, 3$. (Again, feel free to use any numerical analysis program to get the final answer. But, show your calculation process.)

We can get a_{ij}'s from $a_{ij} = \boldsymbol{x}_i \cdot \boldsymbol{u}_j$, and $b_3 = \overline{\boldsymbol{x}} \cdot \boldsymbol{u}_3$.

$$a_{11} = -1.86, a_{12} = 1.22$$
$$a_{21} = 1.67, a_{22} = 0$$
$$a_{31} = 1.86, a_{32} = 1.22$$
$$b_3 = -0.157$$

From this, we get:

$$\widetilde{\boldsymbol{x}}_1 = \begin{bmatrix} -0.013 & -1.043 & -1.971 \end{bmatrix}, \widetilde{\boldsymbol{x}}_2 = \begin{bmatrix} 0.953 & 0.739 & 1.166 \end{bmatrix}, \widetilde{\boldsymbol{x}}_3 = \begin{bmatrix} 2.054 & 0.310 & 0.81 \end{bmatrix},$$
$$||\widetilde{\boldsymbol{x}}_1 - \boldsymbol{x}_1||^2 = 0.00291, ||\widetilde{\boldsymbol{x}}_2 - \boldsymbol{x}_2||^2 = 0.0979, ||\widetilde{\boldsymbol{x}}_3 - \boldsymbol{x}_3||^2 = 0.135.$$

Therefore, we obtain the perfect data reconstruction, i.e. the construction error is zero. One interesting thing to notice that when the data dimension is d and the number of data points is also d, it is always sufficient to reconstruct the original data from $d - 1$ principal components, when we know that the mean is $\mathbf{0}$. It is because, after knowing $d - 1$ points, we can always deduct one remaining point from the known points, which means that there are only $d - 1$ linearly independent components.

4 Clustering Human Activity using Inertial Sensors Data *[40 points]*

In this assignment, you will explore and apply clustering techniques to the Human Activity Recognition Using Smartphones dataset. This dataset (available at UCI archive) was created from recording data from 30 individuals performing six different activities (walking, walking upstairs, walking downstairs, sitting, standing, laying) while wearing a smartphone equipped with inertial sensors at their waist. We will be working with two primary forms of this data:

a. **Feature Extracted Data:** This dataset contains pre-processed features derived from the raw sensor signals. Check the file "**features_info.txt**" to learn more.

b. **Raw Sensor Data:** This dataset contains the original, unprocessed time-series data from the smartphone's accelerometer and gyroscope sensors.

Note:

for this question use the starter notebook (`hw4_4_starter_notebook.ipynb`) to guide your answers.

4.1 Import Data and Plotting *[5 points]*

a. Import the training and testing data from the feature extracted data files described in the `README.txt`. Compine the features and the labels into one dataframe. Display the first 5 rows of the training feature dataframe.

b. Scale the data so that for each feature the mean is zero and the standard deviation is 1. Then perform PCA on the scaled data to find the first two principal components.

c. Visualize the training data by creating a scatter plot of its first two principal components. Color the points in the scatter plot according to their respective activity labels. Include this scatter plot in your solution PDF.

4.2 Choosing The Optimal Number of Clusters *[10 points]*

Now we will use K-means clustering (as taught in class) and attempt to choose the optimal number of clusters (k) using two different methods and analyse the results.

a. **Elbow Method**:

The Elbow Method is a populer heuristic approach used in K-Means clustering to determine the optimal number of clusters. It provides a systematic way of identifying the best k by analyzing how the distortion (see Question 2) changes as k increases. The distortion is also referred to as inertia, which measures the compactness of clusters.

To determine the optimal k, we plot distortion against k. The curve usually shows a steep drop initially, followed by a slower decline. The "elbow" is the point where this change in slope is most pronounced. This 'elbow' point is usually determent from the plot directly but it can also be define mathmatically as the value of k where the second derivative of the distortion is maximized.

Plot the distortion versus k for $k = 2$ to $k = 15$. Choose the value of k where the curve sharply flattens ("elbow"), and then create a scatter plot that visualizes the clusters for the chosen k. Each point in the scatter plot should correspond to a data point, with different colors or shapes the different clusters. Use the two principal components you found in part 4.1 as the axes of your plot. In your comments, describe how the distortion changes with increasing k.

b. **Adjusted Rand Index (ARI)**

The Adjusted Rand Index (ARI) is a clustering evaluation metric that measures the alignment between a predicted clustering and a ground truth labels. In our case, the clustering obtained by K-means and the true activity labels.

The ARI score is normalized so that $ARI = 1$ indicates perfect clustering (identical to the ground truth), $ARI = 0$ corresponds to random clustering (no better than chance), and $ARI < 0$ suggests clustering worse than random assignment[2].

Compute ARI for $k = 2$ to $k = 15$ using the true activity labels. (you can use sklearn `adjusted_rand_score` for this task) Pick the value of k with maximal ARI and describe how the ARI changes with increasing k. Create a scatter plot of the resulting clusters, as you did for the elbow method clusters above.

4.3 Prototype Selection using K-means Clustering *[10 points]*

Prototype selection refers to the process of choosing a subset of representative data points from a larger dataset to reduce storage, computational cost, and redundancy while preserving essential information for classification or clustering tasks. It can also help reduce labeling time, particularly in scenarios where manual annotation is costly and time-consuming, such as active learning and semi-supervised learning. k-means clustering can be used to select prototypes by selecting prototypes from each of the k identified clusters, thus ensuring a diversity of data in the selected prototypes.

In this section, we will simulate this scenario by limiting the number of labeled samples used to train a logistic regression classifier.

a. **Random Selection:** Begin by implementing a uniformly random selection strategy to choose 120 data points from the training data set. After selecting the prototypes, use the resulting dataset to train a logistic regression model, with the activity type as the label, and calculate the accuracy. Repeat this experiment 10 times and report the average accuracy.

b. **Using K-means Clustering by Class**: Now perform k-means clustering on the training dataset using $k = 20$ for each label independently and identify the cluster centers. Based on these centers, you will then for each cluster centroids choose the closest points as representative training data (you will get 120 points). Use these points to train a logistic regression model. Compare the accuracy of the new model to the one trained in part (a) to assess how using clustering for prototype selection affects the performance.

4.4 Autoencoder for Feature Learning *[15 points]*

In this section, we will implement an autoencoder, a type of neural network used for unsupervised learning of efficient codings. Autoencoders work by encoding input data into a lower-dimensional representation and then decoding it back to the original input (see Fig. 3). This process forces the network to learn a compressed representation of the data.

Autoencoders are particularly useful for dimensionality reduction, feature extraction, and denoising. In this part of the problem, we will leverage an autoencoder to create a more compact and more informative representation of the raw sensor data from the Human Activity Recognition dataset by training an autoencoder on the raw sensor readings.

This question will provide you with hands-on experience using PyTorch to build and train and implementing a bidirectional GRU autoencoder neural network.

[2] This suggests that the clustering systematically splits cohesive groups apart, causing ARI to be negative.

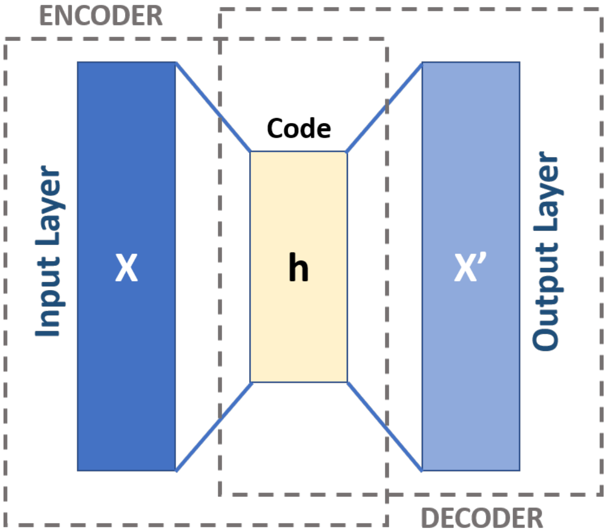

Figure 3: Image showing autoencoder structure [Source Wikipedia].

a. **Create Dataset Class:** Begin by loading the raw sensor data as described in the README.txt file. Ensure the data is correctly formatted for input into the autoencoder. Create a PyTorch Dataset class to efficiently load and batch the raw sensor data. Utilize a DataLoader to create batches for training the autoencoder.

b. **Autoencoder Implementation:** Implement the TimeSeriesAE autoencoder model using PyTorch. This model should use GRU layers for both the encoder and decoder and an embedding of size 64. Instantiate the model, define an appropriate loss function, and choose an optimizer. Train the autoencoder for 10 epochs. Plot the training loss vs. epochs.

c. **Embedding Extraction and Visualization:** Extract the encoded representations (embeddings) from the trained autoencoder for the training data. To do this, pass the training data through the encoder part of the autoencoder model. Use the first two embeddings to create a scatter plot of the embeddings, colored by the activity labels. Include this plot in your solution PDF.

d. **Clustering Evaluation and Comparison:** Perform K-means clustering on the extracted 64D embeddings. Evaluate the clustering performance using the Adjusted Rand Index (ARI) by comparing the cluster labels with the true activity labels. Compare the ARI obtained by clustering the autoencoder embeddings with the clustering performance you obtained from the hand-engineered features in Part 4.2. Discuss which of the two provides the highest ARI and the potential reasons for that.