ECE 685D HW1

Deliverability: Please upload your Jupyter Notebook file (.ipynb file) with all outputs and necessary derivations exported as a PDF or HTML to Sakai. All necessary coding documentation can be found in `https://pytorch.org/docs/stable/index.html`. **We highly encourage you to submit your derivation in LaTeX. If you choose to submit a hand-written derivation, please make sure your derivation is legible. If you have hand-written solutions, please scan/take a photo and insert them into their designated space.** You can follow this tutorial to insert images to .ipynb files: `https://www.geeksforgeeks.org/insert-image-in-a-jupyter-notebook/`

1 Problem 1: Linear regression on a simple dataset(30 pts)

In this problem, you will implement a multi-linear regression model from scratch. Please download the **Concrete Strength regression** dataset from `https://www.kaggle.com/datasets/maajdl/yeh-concret-data`

A multi-linear regression model takes the form

$$Y = X\hat{\beta} + \hat{\epsilon} \; s.t. \; \hat{\epsilon} \sim N(0, \hat{\sigma} I)$$

Here, $X \in \mathbb{R}^{N \times m}$ such that each column represents an independent variable, and $Y \in \mathbb{R}^{N \times 1}$ represents the column for the dependent variable to be predicted. We assume our model minimizes the MSE loss (i.e., $\hat{\beta}_{opt} = arg \min_{\hat{\beta}} ||Y - X\hat{\beta}||_2^2$)

Given the dataset, "concrete compressive strength" is the variable to be predicted.

(1) Given all independent variables plus a bias term, please find a $\hat{\beta}_0 \in \mathbb{R}^{9\times 1}$ $s.t.$ $||Y - X\hat{\beta}_0||_2$ is minimized. You may only use matrix operations (e.g., multiplication, decomposition, inverse) to find β_0. You are not allowed to use any statistical libraries.

(2) Randomly split the dataset into a training set (75%) and a validation set (25%). Use the training set to build one model with $\hat{\beta} \in \mathbb{R}^{i\times 1}$ for $i \in \{7, 8, 9\}$ each. Then, compute the MSE loss of your prediction on the validation set for each model. What do you observe from the results of your three models? Do models with more dependent variables always perform better? You may use any combination of independent variables of your choice.

2 Problem 2: Multinomial Logistic regression from pre-trained feature extractor (45pts)

In the problem, you will implement a logistic regression model from scratch to classify MNIST when given a pre-trained feature extractor. Specifically, you will walk through the following steps:

Step 1:
Load the pre-trained weight to your feature extractor. The code defining the architecture of your feature extractor is included in the attachment.

Step 2:
Extract latent representation (denoted as $h \in \mathbb{R}^k$) from each sample in MNIST (denoted as $x \in \mathbb{R}^{784}$) with the pre-trained feature extractor.

Step 3:
Derive the gradient of $\mathbf{W}$ *and* b w.r.t. the cross-entropy loss between label and prediction for the model

$$\hat{y} = arg \max \sigma(W^T h + b)$$

Step 4:
Use stochastic gradient descent (SGD) implemented from scratch (with matrix operations only) and the gradient found in step 3 to find a good weight for parameter $\mathbf{W}$ *and* b such that the cross-entropy loss is (mostly) minimized.

3 Problem 3: Transformation from a uniform distribution (25pts)

The Box–Muller transform is a popular sampling method for generating pairs of independent, standard, normally distributed random variables from a pair of independent, standard uniformly random numbers. Specifically, let $U_1 \sim Uniform(0,1), U_2 \sim Uniform(0,1)$ and define the random variable

$$\Theta = 2\pi U_1, R = \sqrt{-2ln(U_2)}$$

Show that the random variable vector

$$Z = (R\cos(\Theta), R\sin(\Theta))^T$$

follows a standard bivariate normal distribution. (i.e.,$Z \sim N(0, I_2)$)

Then, make a plot to show the transformed distribution is indeed Gaussian. You should start with np.random.rand() to sample from a uniform distribution.

4 Problem 4: Singular Value Decomposition (bonus 15pts)

In linear algebra, the singular value decomposition (SVD) is a factorization of a real or complex matrix. Such a decomposition is usually denoted as

$$\mathbf{A} = \mathbf{U\Sigma V}^T$$

Conceptually, it projects the original matrix to orthogonal planes. One application of SVD is dimensionality reduction: consider a full-rank square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, removing the k smallest singular value in $\mathbf{\Sigma}$ would result in a rank-(n-k) approximation $\mathbf{A'} = \mathbf{U}[:, 0:k]\mathbf{\Sigma}[0:k, 0:k]\mathbf{V}[:, 0:k]^T$ such that $||\mathbf{A'} - \mathbf{A}||_2$ is minimized. Propose and implement an algorithm that is capable of storing at least 6 differently looking images sized $(n \times n)$ at recognizable quality with less or equal to $3 \times n \times n$ float numbers. Your submission should contain:

1. The plots of original images
2. A compression algorithm
3. The output matrix from the compression
4. Images reconstructed from the compressed matrix