IIT BOMBAY

Lab Assignment #10
Points: 5

Main TAs: *Parshant Arora, Vedang Dhirendra Asgaonkar, Mayank Jain, Barah Fazili*

Course: *CS 335* – Instructor: *Preethi Jyothi*
Due date: *October 23, 2023*

General Instructions

1. Download lab10.tgz(`https://drive.google.com/file/d/1ptPWnSx1BEVNNi7DoLGC-48oPCVVSxKn/view?usp=sharing`) from Moodle, and extract the file to get `lab10`.
2. For your final submission, you need to submit three `.py` files with all the TODOs implemented: `lab10_q1.py`, `lab10_q2a.py` and `kaggle.py`. And for Q2B, you will submit a Jupyter notebook titled `lab10_q2b.ipynb`. Submit all these files together as a single .tgz file named `[rollnumber].tgz`.
3. Your final submission is due on Moodle on or before **11.59 pm on Oct 24, 2023**.
4. You will get 1.5 points if we recover the top 2 eigenvalues using PCA and kernel PCA for the half-moons dataset. 1.5 points for the Kaggle task. 1 point for the feedforward autoencoder (with sparsity) and 1 point for the denoising autoencoder.

PCA and Kernel PCA

For this problem, you will implement both Principal Components Analysis (PCA) and kernel PCA for dimensionality reduction. These techniques are applied to the well-known half moons dataset.[a] We visualize the first two principal components using both PCA and kernel PCA to highlight how the points are well-separated using the latter (but not the former).

`lab10_q1.py` contains skeleton code with functions `pca` and `kernel_pca` that need to be filled in. For kernel PCA, we will use a Gaussian or RBF kernel defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\{-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2\}$$

where γ is a tunable hyperparameter (set to 15 by default in the code).

For PCA, recall that the $\mathbf{X}$ data matrix has to be centered before computing the covariance matrix. Similarly, in kernel PCA, the datapoints projected to the implicit feature space defined by the kernel need to be centered. For a dataset with n instances, the following transformation to the (uncentered) $n \times n$ kernel matrix K results in a centered $n \times n$ kernel matrix K':

$$K' = K - \mathbf{1}_n K - K\mathbf{1}_n + \mathbf{1}_n K \mathbf{1}_n$$

where $\mathbf{1}_n$ is an $n \times n$ matrix with every element taking the value $\frac{1}{n}$.

Rest of the details are specified in TODO comments within `lab10_q1.py`.

❄ Extra Credit: Climb the Leaderboard on Kaggle

In this part, you will apply PCA or kernel PCA to a real problem and learn a binary classifier on the reduced feature set. `kaggle.py` should be filled in from scratch. `train.csv` contains the training data for this problem with binary labels **g** (for good) and **b** (for bad). You should apply PCA or kernel PCA to the features in `train.csv` to any dimensionality you deem fit, train a binary classifier on the reduced feature set and make predictions for all the test instances in `test.csv`. Save the predictions in `submission.csv` and upload this file to the Kaggle competition at `https://www.kaggle.com/t/5a575c04100e4d5db3ab1e8cf5ff94ec` to be evaluated. The file format of `submission.csv` is id, label on each line where label = {g,b}. You can use any binary classifier (SVM, logistic Regression, etc.) you like. Your roll number should appear on both the "Public Leaderboard" (and eventually the "Private Leaderboard" after the assignment concludes). The current TA submission on the leaderboard uses a logistic regression classifier with kernel PCA, $\gamma = 0.1$, $k = 10$. Top-scoring performers on the "Private Leaderboard" (with a suitable threshold determined after the deadline passes) will be awarded up to 3 extra credit points. The exact breakdown of the three points for this question will be announced later.

[a] `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html`

Autoencoders: Sparsity, Convolutional, Denoising

For this problem, you will implement autoencoders and explore the following two facets:

A. **Sparsity.** In an autoencoder, consider an encoder network that projects an input $\mathbf{x} \in \mathbb{R}^d$ into a projection $\mathbf{h} \in \mathbb{R}^k$ (where $k < d$) and a decoder network that takes $\mathbf{h}$ as input and maps it to a $\mathbf{x}' \in \mathbb{R}^d$ where $\mathbf{x}'$ is a reconstruction of $\mathbf{x}$. In `lab10_q2a.py`, we aim to train an autoencoder to reconstruct MNIST digits. Implement a feedforward autoencoder where the encoder and decoder networks are implemented using feedforward layers with ReLU activations. More details about the encoder/decoder architectures are specified in the TODO comments in `lab10_q2a.py`. The reconstruction loss used to train the autoencoder is typically a mean-squared loss. We can additionally impose a sparsity prior on the latent representation $\mathbf{h}$; the resulting reconstruction loss then becomes:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}', \mathbf{h}, \lambda) = \frac{1}{n}\left(\sum_{i=1}^{n} \|\mathbf{x} - \mathbf{x}'\|_2^2 + \lambda \sum_{i=1}^{n} \|\mathbf{h}\|_1\right)$$

where λ is a tunable hyperparameter controlling the amount of sparsity in $\mathbf{h}$, $\|\|_1$ and $\|\|_2$ refer to L1 and L2 norms. Complete runs of the experiment with `num_epochs` set to 20 with the following four different values of $\lambda = \{1, 0.1, 0.01, 0\}$ and observe the test loss (printed on the last line of `lab10_q2a.py`).

B. **Convolutional Encoder/Decoders and Denoising.** In order to reconstruct images as in MNIST, rather than feedforward networks employed in the previous part, it is more natural to use convolutional layers for the encoder and decoder networks in an autoencoder. Implement convolutional encoder/decoder networks within `__init__` in the class definition of `ConvAutoEncoder` in `lab10_q2b.py`. Details are specified in the TODO comments. The rest of the code has been filled in. Observe the quality of the reconstructed images plotted using `ax.imshow`. Run this part as a Colab notebook since you will benefit from using a GPU. Download and submit the resulting notebook file as `lab10_q2b.ipynb`.

   Autoencoders are popular for the task of *denoising*. That is, autoencoders take a noisy image as input and aim to reconstruct the original image by removing the noise. In `train_model_with_noise`, we add random Gaussian noise to the input with a scaling factor (set to 0.1) and train the autoencoder to reconstruct the original image. The only TODO is to compute and print the average test reconstruction loss (MSEloss) for all the images in `test_set`. We print the average test loss for three different settings to check for robustness:

   i. Gaussian noise and the noise factor is the same during train and test.

   ii. Gaussian noise and the noise factor differs between train and test.

   iii. Gaussian noise during training and Bernoulli noise during test.

   Try different convolutional architectures and other hyperparameters and observe its effect on the test losses and visualize the resulting denoised images. (Code for the latter is already filled in.)