Due: 06/05/2024 23:59 on NTU COOL

Foundations of Artificial Intelligence: Homework 4

Instructor: Shang-Tse Chen & Yun-Nung (Vivian) Chen

Any form of cheating, lying, or plagiarism will not be tolerated. Students can get zero scores and/or fail the class and/or be kicked out of school and/or receive other punishments for misconduct. Discussions on course materials and homework solutions are encouraged, but you should write the final solutions alone and understand them fully. Books, notes, Internet, ChatGPT resources can be consulted but not copied from. Please list your references to avoid any plagiarism concerns.

1 Hand-written Part

For the hand-written part, please write down your calculations and reasoning steps. Only providing the final answer may result in score deductions.

Problem 1 (15 points)

`Swish` is an activation function that can be viewed as a "modification" of the logistic function. Swish has been shown to be better than ReLU in some deep learning models. Recall that the logistic function is defined as

$$\theta(s) = \frac{1}{1 + \exp(-s)}$$

Now Swish is defined as

$$\varphi(s) = s \cdot \theta(s)$$

What is $\varphi'(s)$?

Problem 2 (20 points)

In the class, we briefly talked about the famous **PageRank** algorithm, which assigns a rank to each web page. Given a set of pages: $P = \{P_1, P_2, \cdots, P_n\}$ and their incoming links: $in(P_i) = \{P_j \mid P_j \text{ has a link to } P_i, i \neq j\}$. The basic idea is that a web page should be ranked higher if it has 1) more incoming hyperlinks and 2) incoming links from high-ranked pages. The PageRank can be computed through the following iterative algorithm:

1. Initialize $\text{PageRank}_0(P_i) = \frac{1}{n}$ for each page.
2. Compute the updated PageRank for each page:

$$\text{PageRank}_1(P_i) = \sum_{P_j \in in(P_i)} \frac{\text{PageRank}_0(P_j)}{|out(P_j)|},$$

   where $|out(P_j)|$ denotes the out-degree of P_j.

3. Repeat step 2 until the ranks converge.

Now, consider the following example with 3 pages:

$$P = \{P_1, P_2, P_3\}$$
$$in(P_1) = \{P_2, P_3\}$$
$$in(P_2) = \{P_3\}$$
$$in(P_3) = \{P_1\}$$

We define the following variables:

$$\mathbf{v}_t = \left[\text{PageRank}_t(P_1), \text{PageRank}_t(P_2), \text{PageRank}_t(P_3)\right]^T$$
$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0.5 \\ 0 & 0 & 0.5 \\ 1 & 0 & 0 \end{bmatrix},$$

where $\mathbf{v}_t$ is the ranks after the t-th iteration, and $\mathbf{P}$ is the transition matrix between pages. We can rewrite the update rule above as $\mathbf{v}_t = \mathbf{P}\mathbf{v}_{t-1}$. Answer the following questions:

(A) Based on the iterative algorithm above, please compute the values of $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, $\mathbf{v}_4$, and $\mathbf{v}_5$.

(B) Recall that in the class, we mentioned that the algorithm converges when $\mathbf{v}^* = \mathbf{P}\mathbf{v}^*$. Solve this equation to derive $\mathbf{v}^*$. Note that you should provide a normalized $\mathbf{v}^*$, i.e., $\sum \mathbf{v}^* = 1$, as the answer.

Problem 3 (20 points)

We talked about the K-means clustering algorithm in class, which iteratively does partition optimization and prototype optimization until convergence. Given a set of 2-dimensional data points $\mathbf{X} = \{(1,2),(3,4),(7,0),(10,2)\}$, we would like to perform K-means clustering with $K = 2$. Answer the following questions. Note that you should write down the resulting cluster centroids and partitions of each iteration.

(A) Perform the K-means algorithm with $K = 2$ and initial centroids $\{\mu_1, \mu_2\} = \{(1,2),(3,4)\}$ until convergence.

(B) Perform the K-means algorithm with $K = 2$ and initial centroids $\{\mu_1, \mu_2\} = \{(1,2),(7,0)\}$ until convergence. Are the results different from (A)?

(C) Now consider adding a data point $(5,6)$ to $\mathbf{X}$. Show that the K-means algorithm converges to a local minimum with certain intial centroids. You can demonstrate this by showing that there exists a set of initial centroids that does not converges to the global minimum.

2 Programming Part

In the programming part, our goal is to create a simplified facial recognition system that can identify individuals from human face images. The main aim is to provide you with a comprehensive understanding of traditional and deep learning models used to extract valuable (dimension-reduced) features for this purpose. We will focus on two unsupervised learning models covered in the course: principal component analysis and autoencoder. Additionally, we will expand our exploration by introducing the denoising autoencoder as an advanced variation of the autoencoder.

Please download `hw4.zip` which includes sample codes (hw4.py, src/pca.py, and src/autoencoder.py) and the dataset. The data we used here is processed from so-called Yale face data, but you are not allowed to download the original one. There are 11 images per subject. The 11 images, each of dimension 80×61, are taken under conditions of center-light, w/glasses, happy, left-light, w/o glasses, normal, right-light, sad, sleepy, surprised, and wink. We will take the last two images per subject (surprised, wink) for validation and the other 9 images for training.

You will be graded under the following environment:

- python 3.9
- numpy 1.24.3
- pandas 2.0.1
- Pillow 9.5.0
- scikit-learn 1.2.2
- torch 2.0.1
- tqdm 4.65.0

- matplotlib 3.7.1

You **MUST** use the provided sample code `hw4.py, pca.py and autoencoder.py` from NTU Cool or here as a starting point for your implementation.

2.1 Homework Description

Complete the following tasks:

1. **Principal Component Analysis**
   (a) Implement `fit` method for the class `PCA` in `pca.py`. The fit method should calculate the mean of the training data and the top n components eigenvectors of the mean-shifted training data. Run fit on the training dataset (of 135 images) with any n components, where n is larger than 4.
   (b) Implement the `reconstruct` methods for `PCA`. Transform the given file `subject_05_17.png` with PCA and reconstruct the image with PCA. Plot the original image and the reconstructed image side by side and put it in your report.
   (c) Implement the `transform` methods for `PCA`. Then use the transformed features to train a classifier of `LogisticRegression`, as done in `hw4.py`.

2. **Autoencoder**
   (a) Implement `fit` method for the class `Autoencoder` in `autoencoder.py`. The fit method should optimize the reconstruction error, i.e., the averaged squared error between x_n and $g(x_n)$, where $g(x_n)$ is the reconstructed example of x_n after going through the encoder and decoder of the associated `Autoencoder`. Please take the default architecture in the constructor of the `Autoencoder`, and train with any proper optimization routine in PyTorch. Plot the averaged squared error when running fit on the training dataset as a function of number of iterations (or epochs) and put it in your report.
   (b) Implement the `reconstruct` methods for `Autoencoder`. Transform the given file `subject_05_17.png` into lower dimension and reconstruct the image with `Autoencoder`. Plot the original image and the reconstructed image side by side and put it in your report.
   (c) Implement the `transform` methods for `Autoencoder`. Then use the transformed features to train a `LogisticRegression` classifier, as done in `hw4.py`.

3. **Denoising Autoencoder**
   (a) Implement `fit` method for the class `DenoisingAutoencoder` in `autoencoder.py`. The fit method should optimize the averaged squared error between x_n and $g(x_n + \epsilon)$, where $g(x_n + \epsilon)$ is the reconstructed example of x_n plus a Gaussian noise ϵ. Each component of ϵ is assumed to come from an independent Gaussian distribution of standard deviation noise factor, which is by default set to 0.2. Please take the default architecture in the constructor of the Autoencoder, and train with any proper optimization routine in PyTorch. Plot the averaged squared error when running fit on the training dataset as a function of number of iterations (or epochs) and put it in your report.

2.2 Deliverables

1. Source code (Python), including `hw4.py, src/pca.py, and src/autoencoder.py`
2. A detailed report answering the following questions:
   (a) Plot the mean vector as an image as well as the top 4 eigenvectors, each as an image, by calling the given plot component routine. Each of those top eigenvectors is usually called an *eigenface* that physically means an informative "face ingredient."
   (b) Plot the training curve of `Autoencoder` and `DenoisingAutoencoder`
   (c) Plot the original image and the images reconstructed with `PCA, Autoencoder`, and `DenoisingAutoencoder` side by side, ideally as large as possible. Then, list the mean squared error between the original image and each reconstructed image

(d) Modify the architecture in `Autoencoder` in its constructor. Try at least two different network architectures for the denoising autoencoder. You can consider trying a deeper or shallower or fatter or thinner network. You can also consider adding convolutional layers and/or other activation functions. Draw the architecture that you have tried and discuss your findings, particularly in terms of the reconstruction error that the architecture can achieve after decent optimization.

(e) Test at least 2 different optimizers, compare the training curve of `DenoisingAutoencoder` and discuss what you have found in terms of the convergence speed and overall performance of the model.

2.3 Grading Criteria

Criterion 1 PCA implementation: 10%

Criterion 2 Autoencoder implementation: 10%

Criterion 3 DenoisingAutoencoder implementation: 7%

Criterion 4 Transformation implementation: 4%

Criterion 5 Reconstruction implementation: 4%

Criterion 6 Report quality and clarity: 10%

Submission

Write a PDF report for P1 to P4 and Programming Part. Complete hw4.py, src/pca.py and src/autoencoder.py for Programming Part.

Then, submit a zip file to NTU COOL. The zip file should be named `b0x902xxx.zip` that contains a directory called `b0x902xxx/`, which includes two files, the completed `hw4.py` and `report.pdf`, and a folder `src` including `pca.py` and `autoencoder.py`.