Homework #4

RELEASE DATE: 05/20/2022

DUE DATE: 06/10/2022 23:59 on NTU COOL

RED BUG FIX: 5/23/2022 06:00

QUESTIONS ABOUT HOMEWORK MATERIALS
ARE WELCOMED ON THE NTU COOL FORUM.

Unless granted by the instructor in advance, you must upload your solution to NTU COOL as instructed by the TA.

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 those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

You should write your solutions in English or Chinese with the common math notations introduced in class or in the problems. We do not accept solutions written in any other languages.

Hand-written Part

1. (10%) 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 us

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

Now Swish is defined as

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

What is $\varphi'(s)$?

2. (10%) We discussed about cost-sensitive classification in class, which takes a cost matrix C such that C_{ij} (the element in the i-th row and the j-th column) represents the cost charged when classifying a class-i example as a class-j one. Consider the following cost matrix:

$$\begin{bmatrix} 0 & 1 & 10 & 100 \\ 200 & 0 & 2 & 20 \\ 30 & 300 & 0 & 3 \\ 4 & 40 & 400 & 0 \end{bmatrix}$$

For a size-N data set with $\frac{N}{4}$ examples for each class, consider a classifier g such that $E_{\text{in}}(g)$ is 5%. That is,

$$\frac{1}{N}\sum_{n=1}^{N}[y_n \neq g(\mathbf{x}_n)] = 5\%.$$

What is the maximum total cost that the classifier g has to pay on those examples? What is the minimum total cost that the classifier has to pay?

3. (10%) For a $d^{(0)} - d^{(1)} - d^{(2)} - \ldots - d^{(L)}$ fully-connected neural network, the total number of neurons is

$$D = \sum_{\ell=1}^{L} d^{(\ell)}$$

and the total number of weights is

$$\sum_{\ell=1}^{L} d^{(\ell-1)} \cdot d^{(\ell)},$$

which ignores the so-called 'bias' terms in the network. Let the number of inputs $d^{(0)} = 10$ and the total number of neurons $D = 100$. Over all possible choices of $L, d^{(1)}, d^{(2)}, \ldots, d^{(L)}$, what is the maximum total number of weights (and what neural network architecture does it correspond to)? What is the minimum total number of weights (and what neural network architecture does it correspond to)?

4. (10%) We discussed about active learning in class. One popular active learning strategy is called uncertainty sampling, which queries an example $\mathbf{x}$ with the maximum uncertainty. In logistic regression, the uncertainly can be defined as

$$u(\mathbf{x}) = 1 - 2\left|\theta_{\mathbf{w}}(\mathbf{x}) - \frac{1}{2}\right|,$$

where $\theta_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$ for the logistic regression weights $\mathbf{w}$. Prove that

$$\underset{n=1,2,\ldots,N}{\arg\max}\ u(\mathbf{x}_n) = \underset{n=1,2,\ldots N}{\arg\min}\ \left|\mathbf{w}^T\mathbf{x}_n\right|.$$

That is, the maximum uncertainty example is also the minimum margin example, which is of score closest to 0.

Programming Part

In the programming part, we will develop a simplified facial recognition system, which aims to return the subject identity behind the human face images. Our goal is to help you better understand both traditional and deep learning models for extracting useful (dimension-reduced) representations for this task. We will let you study two unsupervised learning models that have been taught in class: principal component analysis and autoencoder. We will then enrich our study with an extension of the autoencoder, called the denoising autoencoder.

Please download `data.zip` from NTU COOL. The dataset contains 165 grayscale images in PNG format that come from 15 subjects. The dataset is processed from the so-called Yale face data, but you are prohibited to (and really do not need to) download the original dataset. 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.

Your code will be graded under the following environment:

- python 3.8
- numpy 1.19.5
- pandas 1.1.2
- matplotlib 3.3.4
- torch 1.10.2
- tqdm 4.62.3
- scikit-learn 0.24.2, while we **prohibit** using sklearn.decomposition.PCA

5. (10%) Implement the `fit` method for the class `PCA` that represents the principal component analysis model. 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` ≥ 4. Then, 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."

6. (10%) Implement the `fit` method for the class `Autoencoder` that represents the autoencoder model. The `fit` method should optimize the reconstruction error, i.e., the averaged *squared error* between $\mathbf{x}_n$ and $g(\mathbf{x}_n)$, where $g(\mathbf{x}_n)$ is the reconstructed example of $\mathbf{x}_n$ after going through the `encoder` and `decoder` of the associated `AutoencoderModel`. Please take the default architecture in the constructor of the `AutoencoderModel`, 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).

7. (10%) Implement the `fit` method for the class `DenoisingAutoencoder` that represents the denoising autoencoder model. The `fit` method should optimize the averaged *squared error* between $\mathbf{x}_n$ and $g(\mathbf{x}_n + \boldsymbol{\epsilon})$, where $g(\mathbf{x}_n + \boldsymbol{\epsilon})$ is the reconstructed example of $\mathbf{x}_n$ plus a Gaussian noise $\boldsymbol{\epsilon}$. Each component of $\boldsymbol{\epsilon}$ 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 `AutoencoderModel`, 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).

8. (10%) Modify the architecture in `AutoencoderModel` with the callback function 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.

9. (10%) Implement the `reconstruct` methods for `PCA` and `Autoencoder`. Transform the given file `subject05_06.png` onto a latent space of dimension 16 and then reconstruct the image with `PCA`, `Autoencoder` and `DenoisingAutoencoder`. Plot the original image and the reconstructed images side by side, ideally as large as possible. Then, list the mean squared error between the original image and each reconstructed image.

10. (10%) Implement the `transform` methods for `PCA` and `Autoencoder`. Then, the `fit_transform` method provided in the abstract class `DataTransformer` allows you to `fit` the training data and `transform` the very same training data to a dimension-reduced representation. Use the transformed data to train a `LogisticRegression` classifier, as done in `main.py`. List the validation accuracy for `PCA`, `Autoencoder` and `DenoisingAutoencoder`.

Submission

Write a PDF report for P1-P10. Complete `pca.py`, `autoencoder.py` for the 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 three files, the completed `pca.py`, `autoencoder.py` and `report.pdf`.