CSCI 567, Spring 2020
Yan Liu

Homework #4 Programming Assignments

General instructions

Starting point Create a folder and place all the scripts in the same folder.

- `hmm_model.json`
- `hmm.py`
- `hmm.sh`
- `pca.py`

High Level Descriptions

0.1 Tasks You will be asked to implement (1) hidden Markov models' (HMM) inference procedures and (2) Principal Component Analysis (PCA). Specifically, you will

- **For Problem 1**, finish implementing the functions `forward`, `backward`, `seqprob_forward`, `seqprob_backward`, and `viterbi`. Refer to `hmm.py` for more information.
- Run the scripts `hmm.sh`, after you finish your implementation. `hmm.sh` will output `hmm.txt`.
- **For Problem 2**, finish implementing the functions `pca`, `decompress` and `reconstruction_error`. Refer to `pca.py` for more information.
- Run `pca.py` (i.e., do `python3 pca.py`), after you finish your implementation, it will generate `pca_output.txt`.

0.2 Dataset In Problem 1, we will play with a small hidden Markov model. The parameters of the model are given in `hmm_model.json`.

In Problem 2, we will use a subset of MNIST dataset that you are already familiar with. As a reminder, MNIST is a dataset of handwritten digits and each entry is a 28x28 grayscale image of a digit. We will unroll those digits into one-dimensional arrays of size 784.

0.3 Cautions Please **DO NOT** import packages that are not listed in the provided code. Follow the instructions in each section strictly to code up your solutions. **Do not change the output format**. **Do not modify the code unless we instruct you to do so**.

Problem 1 Hidden Markov Models

In this problem, you are given parameters of a small hidden Markov model and you will implement three inference procedures. In `hmm_model.json`, you will find the following model parameters:

- π: the initial probabilities, $\pi_i = P(Z_1 = s_i)$;
- A: the transition probabilities, with $a_{ij} = P(Z_t = s_j | Z_{t-1} = s_i)$;
- B: the observation probabilities, with $b_{ik} = P(X_t = o_k | Z_t = s_i)$.

Now we observe a sequence O of length N. Your task is to write code to compute probabilities and infer about the possible hidden states given this observation. In particular, in **1.1** you need to implement the forward and backward procedures; in **1.2**, you need to compute $P(X_{1:N} = O)$, the probability of observing the sequence, using the forward or backward messages; finally in **1.3**, you need to infer the most likely state path using the Viterbi algorithm.

1.1 Please finish the implementation of the functions `forward()` and `backward()` in `hmm.py`:

- `forward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array α, where $\alpha[j, t-1] = P(Z_t = s_j, X_{1:t} = x_{1:t})$. *Note that this is* $\alpha[j, t-1]$ *instead of* $\alpha[j, t]$ *just because the index of an array starts from* 0.
- `backward(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as input and output a numpy array β, where $\beta[j, t-1] = P(X_{t+1:N} = x_{t+1:N} \mid Z_t = s_j)$.

1.2 Now we can calculate $P(X_{1:N} = O)$ from the output of your forward and backward algorithms. Please finish the implementation of the function `seqprob_forward()` and `seqprob_backward()` in `hmm.py`. Both of them should return $P(X_{1:N} = O)$. Note that in `seqprob_forward()` the input is only the forward messages, while in `seqprob_backward()` you have access to the model parameters too.

1.3 Next you need to find the most likely state path based on the observation O, namely,

$$\arg\max_{z^*_{1:N}} P(Z_{1:N} = z^*_{1:N} \mid X_{1:N} = O).$$

Please implement the Viterbi algorithm `viterbi()` in `hmm.py`. The function `viterbi(`π, A, B, O`)` takes the model parameters π, A, B and the observation sequence O as inputs and outputs a list `path` which contains the most likely state path $z^*_{1:N}$ (in terms of the state index) you have found.

After you finish each task in **1.1**, **1.2** and **1.3** in `hmm.py`, run the script `hmm.sh`. It will generate `hmm.txt`.

Hint: You can match the results you are getting. It will be close to the one below.

```
Total log probability of observing the sequence AGCGTA is -8.15031, -8.15031.
Viterbi best path is
1 1 1 1 1 1
```

Problem 2 Principal Component Analysis

In this question we will implement Principal Component Analysis (PCA) and use it for image compression. We will use black and white images in this example for illustratory purposes. (However, PCA can also be used for compression of color images too. For that you need to perform compression on each channel separately and then combine them.)

Let $\boldsymbol{X}$ be an $N \times D$ data matrix, where N is the number of images and D is the number of pixels of each image. For a given parameter $K < D$, you need to use PCA to compress $\boldsymbol{X}$ into an $N \times K$ matrix. This is done via first centering the data, then finding the top K eigenvectors of the covariance matrix $\boldsymbol{X}^T\boldsymbol{X}$, and finally projecting the original data onto these eigenvectors.

Since each of these eigenvectors has dimensionality D, it can also be represented as an image of the same size as the dataset. For example, in Figure 1 we demonstrate top eigenvectors corresponding to all images with digit 6 in MNIST together with their eigenvalues. Figure 2 demonstrates the mean value of MNIST images that correspond to digit 6.

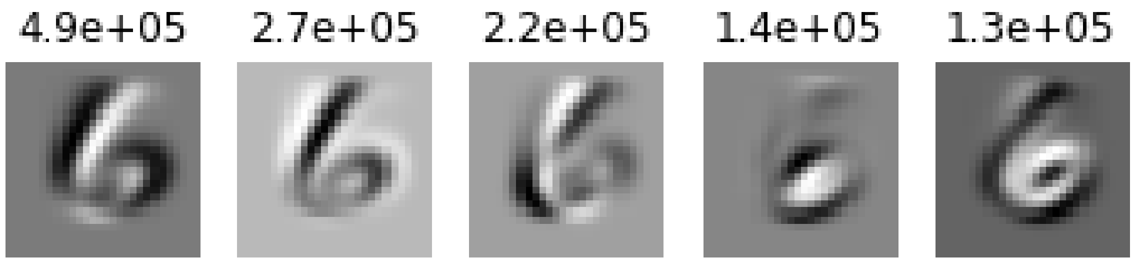

Figure 1: Eigenvectors together with their eigenvalues

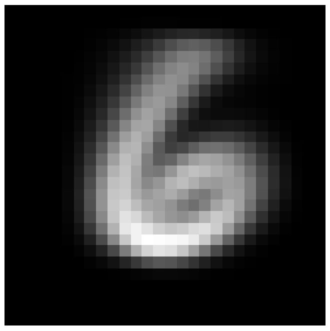

Figure 2: Mean image

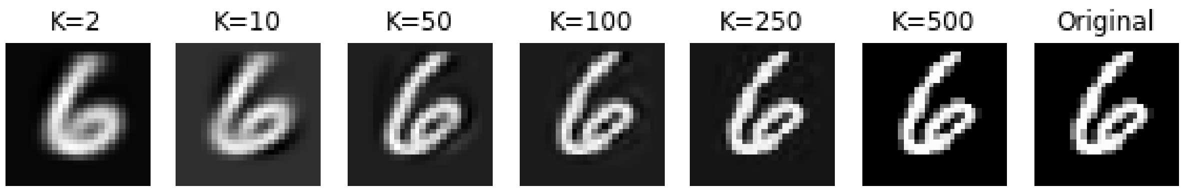

Figure 3: Original image (on the right) and decompressed versions with different compression rates.

Compression

Please implement PCA in function `pca` of `pca.py` file, which returns the compressed representation $\boldsymbol{Y} \in \mathbb{R}^{N\times K}$ and a matrix $\boldsymbol{M} \in \mathbb{R}^{D\times K}$ that consists of the top K eigenvectors, represented as column vectors. Note that we have subtracted the mean values from the data before passing it to `pca`, so you do not need to do this step.

Decompression

With the eigenvector matrix $\boldsymbol{M}$, we can approximately reconstruct the original dataset as $\hat{\boldsymbol{X}} = \boldsymbol{Y}\boldsymbol{M}^T$. Implement this step in function `decompress` of `pca.py`.

In Figure 3 we show several decompressed images with different compression rates (K) and the original image. It is easy to see that larger K leads to better reconstruction. To quantify the reconstruction error between the original image $\boldsymbol{x}$ and the reconstructed one $\hat{\boldsymbol{x}}$, we calculate the pixel-wise mean-square-error as $\frac{1}{D}\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2$. Implement this step in function `reconstruction_error` of `pca.py`.

After implementation, run `python3 pca.py` to generate `pca_output.txt`.

Hint: You can match the results you are getting. It will be close to the one below.

```
Total reconstruction error after compression with 2 principal components is 391.253935
Total reconstruction error after compression with 10 principal components is 241.480441
Total reconstruction error after compression with 50 principal components is 81.578760
Total reconstruction error after compression with 100 principal components is 39.198913
Total reconstruction error after compression with 250 principal components is 9.681371
Total reconstruction error after compression with 500 principal components is 0.281609
```