

---

## CS589 Machine Learning - Fall 2020

### Homework 5: CNN and Dimensionality Reduction

Due: November 6, 11:59 pm

---

**Getting Started:** You should complete the assignment using your own installation of Python 3.6. Download the assignment archive from Moodle and unzip the file. This will create the directory structure as shown below. You will write your code under the Submission/Code directory. Make sure to put the deliverables (explained below) into the respective directories.

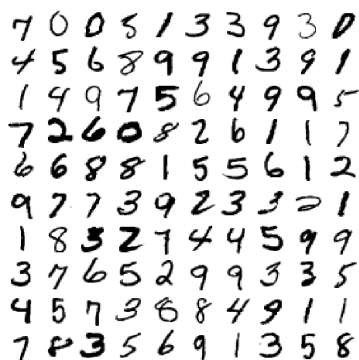
HW05

```
--- Data
    |-- Reduced MNIST Dataset
    |-- baboon.tiff
    Code
    |-- CNN_Template.py
    |-- svd_template.py
    |-- PCA_template.py
--- Submission
    |--Code
    |--Figures
    |--Predictions
```

If you are stuck on a question consider attending the office hours of the TA listed for that question.

**Data Sets:** For this homework, we'll use a subset of the MNIST dataset of handwritten digits containing 10 classes (digits 0-9). The first 100 digits (each 28x28 pixels) in the dataset are shown in the figure below.

Dataset	Training Cases	Test Cases	Dimensionality	Target
Reduced MNIST Dataset	10000	10000	784	10 Classes (0-9)



7 0 0 5 1 3 3 9 3 0  
4 5 6 8 9 9 1 3 9 1  
1 9 9 7 5 6 4 9 9 5  
7 2 6 0 8 2 6 1 1 7  
6 6 8 8 1 5 5 6 1 2  
9 7 7 3 9 2 3 3 2 1  
1 8 3 2 7 4 4 5 9 9  
3 7 6 5 2 9 9 3 3 5  
4 5 7 3 8 8 4 9 1 1  
7 4 3 5 6 9 1 3 5 8

**Deliverables:** This assignment has three types of deliverables: a report, code files, and Kaggle submissions.

- **Report:** The solution report will give your answers to the homework questions (listed below). The maximum length of the report is 5 pages in 11 point font, including all figures and tables. You can use any software to create your report, but your report must be submitted in PDF format.
- **Code:** The second deliverable is the code that you wrote to answer the questions, which will involve training classifiers and making predictions on held-out test data. Your code must be Python 3.6 (no iPython notebooks, other formats or code from other versions). You may create any additional source files to perform data analysis. However, you should aim to write your code so that it is possible to re-produce all of your experimental results exactly by running *python run.me.py* file from the Submissions/Code directory. Remember to comment your code. Points will be deducted from your assignment grade if your code is difficult to reproduce!

**Submitting Solutions:** When you complete the assignment, place your final code in Submission/Code and the Kaggle prediction files for your single best-performing submission in Submission/Predictions/best.csv. If you used Python to generate plots then place them in Submission/Figures. Finally, create a zip file called 'Submission.zip' from your 'Submission' directory only (do not include 'Data' directory). Only .zip files will be accepted for grading code (not .tar, .rar, .gz, etc.). You will upload your .zip file of code and your pdf report to Gradescope for grading. Further instructions for using Gradescope will be provided on Piazza and discussed in class.

**Academic Honesty Statement:** Copying solutions from external sources (books, web pages, etc.) or other students is considered cheating. Sharing your solutions with other students is considered cheating. Posting your code to public repositories like GitHub is also considered cheating. Any detected cheating will result in a grade of 0 on the assignment for all students involved, and potentially a grade of F in the course.

**Note:** You may use PyTorch for the model implementations in all questions unless otherwise specified.

## 1 Convolutional Neural Networks [40 points]

1. [10 points] In order to see the advantage of convolutional neural networks over regular ones, lets compute the number of parameters for the following layer. Suppose, the input to the layer is a  $N \times N$  dimensional feature which gets mapped to a  $N \times N \times C$  dimensional feature.
  - (a) [5 points] How many parameters will a fully-connected layer have? Assume that we have a parameter for each connection from an input unit to an output unit.
  - (b) [5 points] Now assume we have  $C$  filters of size  $K \times K$ , each of which is convolved with the input feature to produce a channel of the output feature. After suitable padding, the output of the convolution can be made equal to  $N \times N$  dimensions. How many parameters does this layer have?
2. [30 points] Now, implement a convolutional neural network for classifying MNIST digits using PyTorch and the provided template `CNN_template.py`. In particular, you need to define your layers in the `__init__()` function, and define the connectivity between the layers in the `forward()` function of class `Convnet(nn.Module)`. Use the configuration provided in the table below. For training, use a batch size of 32, and the Adam optimizer (`torch.optim.Adam`) with learning rate of 0.01 and weight decay of 0.0001 for 10 epochs. You may use the training code that is included in the template.

Layer	Activation	#Filters	Filter Size	Stride	Padding	o/p size	#Params
Input (28 x 28)							
Conv2d	ReLU	32	3x3	1	0		
Conv2d	ReLU	64	3x3	1	0		
MaxPool2d	-	-	2x2	2	0		
Dropout2d (p=0.25)							
Flatten()							
Linear	ReLU	128	-	-	-		
Dropout (p=0.5)							
Linear	Softmax	10	-	-	-		

Now, answer the following questions:

- [10 points] Fill the columns for output size (dimensions of the output feature map) and number of parameters for each layer in the above table.
- [10 points] Plot the training and test accuracy with respect to number of epochs of training (10 epochs, LR=0.01, weight decay=0.0001).
- [5 points] What fraction of the total number of parameters is contained in the convolutional layers and fully connected layers? Explain if these results are surprising to you.

## 2 Singular Value Decomposition [40 points + 10 Extra Credit]

Singular Value Decomposition<sup>1</sup> is a powerful matrix decomposition with many applications. In a real world scenario one is interested in a low rank approximation of the matrix. Specifically, consider a matrix  $\mathbf{X} \in \mathbb{R}^{n \times m}$ . According to the SVD theorem,  $\mathbf{X}$  can be expressed as:

$$\mathbf{X} = \sum_{i=1}^r \sigma_i u_i v_i^T \quad (1)$$

where  $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r$  are the singular values,  $u_i \in \mathbb{R}^n$  for  $i = 1, \dots, r$  are the left singular vectors and  $v_i \in \mathbb{R}^m$  for  $i = 1, \dots, r$  are the right singular vectors and  $r$  is the rank of the matrix. Let  $\mathbf{X}_k$  be the  $k$ -rank approximation of  $\mathbf{X}$ :

$$\mathbf{X}_k = \mathbf{U} \mathbf{S} \mathbf{V}^T = \sum_{i=1}^k \sigma_i u_i v_i^T \quad (2)$$

where  $\mathbf{S}$  is a  $K \times K$  diagonal matrix with positive elements,  $\mathbf{U}$  is an  $N \times K$  matrix such that  $U^T U = I$ , and  $\mathbf{V}$  is a  $D \times K$  matrix such that  $V^T V = I$ . Typically,  $k \ll r$ . Matrix  $\mathbf{X}_k$ , the  $k$ -rank approximation of  $\mathbf{X}$  is optimal with respect to the 2 norm and the Frobenius norm. For example:  $\|\mathbf{X} - \mathbf{X}_k\|_F \leq \|\mathbf{X} - \mathbf{C}\|_F$  for any matrix  $\mathbf{C}$  of rank at most  $k$  (including  $k$ ). In this exercise you will implement a practical algorithm which comes with guarantees on its quality.

**Implementation:** In many cases we are willing to compute a sub-optimal  $k$  rank approximation,  $\hat{\mathbf{X}}_k$  if we have important computational savings. You will implement an algorithm that finds  $\hat{\mathbf{X}}_k$  instead of  $\mathbf{X}_k$  and comes with guarantees on the quality of the approximation as already mentioned. This is done by calculating the approximation of the right singular matrix  $\mathbf{V}$ . The template for the implementation is in `Code/svd.template.py`. The function you will write should have the following input and output arguments:

- Input
  1. Matrix  $X \in \mathbb{R}^{n \times m}$
  2. Integer  $s \leq n$ .
  3. Integer  $k \leq s$ .

<sup>1</sup>[http://en.wikipedia.org/wiki/Singular\\_value\\_decomposition](http://en.wikipedia.org/wiki/Singular_value_decomposition)

- Output

1. Matrix  $\mathbf{H} \in \mathbb{R}^{m \times k}$ , which is the approximation to right singular matrix  $\mathbf{V}$
2.  $\lambda_1, \dots, \lambda_k \in \mathbb{R}^+$ .

Matrix  $H$  will contain the approximation to the top- $k$  right singular vectors and  $\lambda_1, \lambda_k$  is the approximation to the singular values.

1. [25 points] Implement the following algorithm:

- (a) For  $i = 1$  to  $n$  compute  $p_i = \|\mathbf{X}(i, :)\|^2 / \|\mathbf{X}\|_F^2$ .
- (b) For  $i = 1$  to  $s$ 
  - Pick an integer  $j$  from 1 to  $n$  with probability  $Pr(\text{pick } j) = p_j$ .
  - Include  $\mathbf{X}(j, :)$  as a row of  $S$ .
- (c) Compute  $SS^T$  and its singular value decomposition, i.e.,  $SS^T = \sum_{t=1}^s \lambda_t^2 w_t w_t^T$ .
- (d) Compute  $h_t = \frac{S^T w_t}{\|S^T w_t\|}$  for  $t = 1 \dots k$ .
- (e) Return matrix  $\mathbf{H}$  whose columns are the vectors  $h_1, \dots, h_k$  and  $\lambda_1 \geq \dots \geq \lambda_k$ .

You are allowed to use following functions to implement below algorithm.

- (a) `np.linalg.norm`: To calculate the frobenius norm of the matrix  $\|\mathbf{X}\|_F$ .
- (b) `np.linalg.svd`: To calculate SVD of the matrix  $SS^T$
- (c) `np.random.choice`: To randomly choose index  $j$  with probability  $p_j$ .

**Instruction:** As solution to this part, you need to provide your code *only* for SVD function in the **report** since it should not be more than one page. Refer latex code provided with Homework to see how to use *listings* package to include your python code in latex..

```

1 def SVD(A, s, k):
2     # TODO: Calculate probabilities p_i
3     n,m = A.shape
4
5     # TODO: Construct S matrix of size s by m
6     S = np.zeros((s,m))
7
8     # TODO: Calculate SS^T
9
10
11     # TODO: Compute SVD for SS^T
12
13
14     # TODO: Construct H matrix of size m by k
15     H = np.zeros((m,k))
16
17     # Return matrix H and top-k singular values sigma

```

2. [10 points Extra Credit] The matrix  $\hat{\mathbf{X}}_k$  can be calculated from  $H$  as follows:

$$\hat{\mathbf{X}}_k = \mathbf{X}\mathbf{H}\mathbf{H}^T$$

Explain why this is the case.

*Hint:*  $\mathbf{H}$  is the approximation of right singular matrix  $\mathbf{V}$ .

3. [15 points] Apply your implementation to the image in `Data/baboon.tiff` image for  $k = 60$  and  $s = 80$  and compare to the optimal 60-rank approximation.

- (a) Plot the original image  $\mathbf{X}$  and two reconstructed images optimal  $k$ -rank approximation  $\mathbf{X}_k$  and sub-optimal  $k$ -rank approximation  $\hat{\mathbf{X}}_k$  using  $\mathbf{H}$  as indicated in the previous question.
- (b) Report the error in terms of the Frobenius norm for both the optimal 60 rank produced from the SVD and for the 60 rank approximation produced by your implementation.

### 3 Principle Component Analysis [20 points]

In this part, you will be working on the MNIST dataset again (given in `Data/X_train.npy` and `Data/Y_train.npy`) and you will implement the PCA algorithm from scratch. The template for the implementation is in `Code/PCA_template.py`

1. **[5 points]** Use PCA to get a new set of bases. You are required to implement the PCA algorithm. You are allowed to use library functions for eigenvalue decomposition and covariate matrix, in Python `np.linalg.eig` and `np.cov` will do the trick. Plot the eigenvalues in descending order (eigenvalues at y-axis) and report how many components you would pick from the plot and explain why.
2. **[5 points]** Display the sample mean of the dataset as an image. Also display the sample mean images for each digit (use `subfigure` in `matplotlib` to plot all of these). Add all images to your report.
3. **[5 points]** Display the top 5 eigenvectors as images and add them to your report. Interpret these images, which number(s) do they look like, what is captured by these eigenvectors?
4. **[5 points]** Project your dataset on the first and second principal components and plot this projection (you may want to use a scatter plot), using a different color for each digit. Describe the plot, from the perspective of digit similarity. Which groups of digits are close to each other?