UNIVERSITY OF MICHIGAN
Department of Electrical Engineering and Computer Science
EECS 445 Introduction to Machine Learning
Fall 2019

Homework 3, Due: Tues. 11/26 at 11:59pm

Submission: Please upload your completed assignment by 11:59pm Tuesday, Nov. 26th to Gradescope.

1 Convolutional Neural Network [9 pts]

1.1 CNN Architecture [3 pts]

While working on Project 2, Yiming tried to build a convolutional neural network (CNN) to help classify Junghwan's posters. At the beginning, Yiming designed a simple convolutional neural network defined by the layers in the left column of the table below. Note the following definitions:

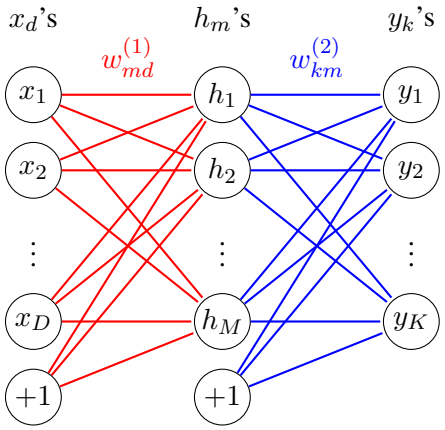
- `CONV5-N` denotes a convolutional layer with N filters, where each filter is $5 \times 5 \times D$ with D being the depth of the activation volume at the previous layer. Padding is 2, and stride is 1.
- `POOL2` denotes a 2×2 max-pooling layer with stride 2 (no padding)
- `FC-N` denotes a fully-connected layer with N neurons.

(a) [3 pts] Fill in the size of output at each layer, and the number of learned parameters at each layer. Write your answers as a multiplication (e.g. $128 \times 128 \times 3$ $(H \times W \times C)$) instead of a single number (for both the output sizes and the number of learned parameters).

Layer	**Output Size**	**Number of Learned Parameters**
Input	$32 \times 32 \times 3$	0
CONV5-5		
POOL2		
CONV5-10		
POOL2		
FC-10		

1.2 Training Neural Nets [6 pts]

After training the above convolutional neural network above, Yiming found the performance of previous architecture on the test dataset was bad. Thus, instead of using a single fully connected layer, Yiming decided to use the following structure at the end of the second pooling layer,



where Yiming flattens the output from the second pooling layer to a vector $\bar{x} = [x_1, x_2, \ldots, x_D]^T$, and adds 2 fully connected layers `FC-`M (with some activation function F_1) and `FC-`K (with some activation function F_2). Here, F_1, F_2 are some nonlinear activation functions where $F_1 : \mathbb{R}^M \to \mathbb{R}^M$, $F_2 : \mathbb{R}^K \to \mathbb{R}^K$.

Note: $w_{ji}^{(l)}$ is the weight for l^{th} fully connected layer, connecting unit j with input i.

We have computed forward and backward propagation of values element-wise in class and discussion. However, this is fairly inefficient when implemented in Python since it requires nested Python loops. Instead, we use matrix operations to avoid explicit loops and speed up our computation since Python functions can be internally optimized to iterate efficiently, possibly using multiple threads.

(a) [1 pt] We can write the forward propagation of these 2 layers in matrix form:

$$\bar{h} = F_1(W^{(1)}\bar{x} + \bar{b}^{(1)})$$

$$\bar{y} = F_2(W^{(2)}\bar{h} + \bar{b}^{(2)})$$

Define $W^{(1)}, W^{(2)}, \bar{b}^{(1)}, \bar{b^{(2)}}, \bar{x}, \bar{h}$, and $\bar{y}$ in terms of $x_1, \ldots, x_D$, $h_1, \ldots, h_M$, $y_1, \ldots, y_k$, and the $w_{ji}^{(l)}$.

(b) [2 pts] Given the derivative of the function $F_2' = G_2$ where $G_2 : \mathbb{R}^K \to \mathbb{R}^K$. Suppose we compute some loss L at the end of the neural network and we have $\frac{\partial L}{\partial \bar{y}} = D$. Find the expression for $\frac{\partial L}{\partial W^{(2)}}$, $\frac{\partial L}{\partial \bar{h}}$, $\frac{\partial L}{\partial \bar{b}^{(2)}}$.

Hint: Make use of the element-wise product function $\bar{x} \odot \bar{y} = [x_1 y_1, x_2 y_2, \ldots, x_n y_n]^T$ for $\bar{x}, \bar{y} \in \mathbb{R}^n$. Use dimension information to confirm your results.

(c) [2 pts] Given the derivative of the function $F_1' = G_1$ where $G_1 : \mathbb{R}^M \to \mathbb{R}^M$. Find the expression for $\frac{\partial L}{\partial W^{(1)}}$, $\frac{\partial L}{\partial \bar{x}}$, $\frac{\partial L}{\partial \bar{b}^{(1)}}$. (You may find your previous results useful.)

(d) [1 pts] After calculating the gradients, based on the previous results, we can update our parameters based on SGD. Suppose the learning rate is η. Write the update rule for $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$.

2 Poster Clusters [18 pts]

Thanks to all of your hard work in Project 2, Junghwan has a way to easily sort through all of the newest movies. Unfortunately, in his excitement, Junghwan misplaced the labels on his posters so you have promised to create him an *unsupervised* algorithm for categorizing the posters.

Like the autoencoder in Project 2, we will attempt to identify structure in the data without training on labels. You will be implementing the k-means clustering algorithm as well as a variant of the algorithm, k-means++, which has the additional property of encouraging good cluster centroid initializations. These are simple yet powerful unsupervised methods for grouping similar examples within a dataset. You will then compare the results of this clustering with the performance of the spectral clustering algorithm. You will write your code in `clustering_classes.py` and `clustering.py`.

(a) [1 pt] Why are poor centroid initializations often a problem?

(b) [1 pt] **Implement** the function `Point.distance()` in the file `clustering_classes.py`. This function takes as arguments two `Point` objects and returns the Euclidean distance between the points. The feature data for each point is stored in the `features` member variable.

(c) [2 pts] **Implement** the function `Cluster.get_centroid()` in the file `clustering_classes.py`. This function returns the centroid of the current `Cluster` object as a `Point` object. As in lecture, this is calculated by finding the mean point of the cluster. You may denote a point as having no label by excluding the label argument. The default value specified in the `Point` constructor will initialize this value to `None`.

(d) [1 pt] **Implement** the function `ClusterSet.get_centriods()` in the file `clustering_classes.py`. This function returns the centroids of a `ClusterSet` object as a list of `Point` objects.

(e) [1 pt] **Implement** the function `random_init()` in the file `clustering.py` according to the function specification. This function returns the cluster centroid initialization for the regular k-means algorithm.

(f) [2 pts] **Implement** the function `k_means_pp_init()` in the file `clustering.py` according to the function specification. This function returns the cluster centroid initialization for the k-means++ algorithm. Pseudocode for this initialization can be found below.

> 1) Randomly select one point as the first centroid.
> 2) For each point x, calculate the distance from x to the nearest centroid that has already been selected. Denote this distance D_x.
> 3) Select the next centroid from the list of points with the probability of selecting point x being proportional to D_x^2. Note that for the sake of implementing this in Python it may be necessary to normalize your calculated distances.
> 4) Continue the previous 2 steps until k points have been selected.

(g) [4 pts] **Implement** the function `k_means()` in the file `clustering.py` according to the function specification. Note that you must construct your own implementation; the use of library functions that trivialize this problem (e.g., `sklearn` implementations) is prohibited.

(h) [4 pts] You will now apply these algorithms to the posters dataset. For the sake of reducing computation time, we will consider only the first 400 images from the dataset (`00000.jpg` through `00399.jpg`). We have provided a preprocessed version of this dataset and the code for loading it in. Within the `main` function in `clustering.py`, use your implementation of the `k_means` function to **determine the best k-value in the range** $[1, 10]$ for each k-means initialization method as well as the provided spectral clustering algorithm. Note that there is not a single correct answer for these values. You should evaluate clustering performance using cluster purity, which is implemented for you in `ClusterSet.get_score()`. The purity metric is the ratio of points within a cluster with the most common label to all points in the cluster. All three of these algorithms require a random initialization, so performance may vary from run to run. To mitigate this effect, **run** your algorithm 10 times for each candidate k value and **average** over the purity scores. A correct implementation may take up to ten minutes to run.

Plot the average performance vs. k for each clustering algorithm on one graph by **implementing** the function `plot_performance()`. **Include** this graph in your write-up. Don't forget to label your axes and include a legend. Your plot should look similar to Figure 1:

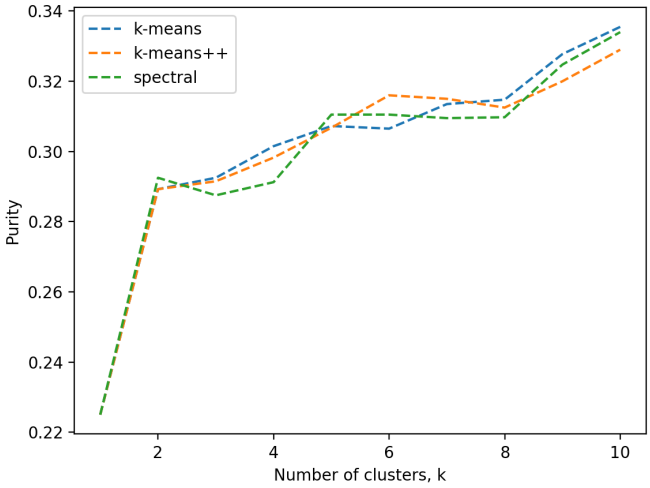


Figure 1: Purity of each clustering method as a function of k

(i) [2 pt] Using the best k-values you found for each algorithm, compare the performance of both initialization techniques. **Report** the average, minimum, and maximum cluster purity of the resulting clusters. Briefly discuss which method works best and why. **Compare** your purity score to the score we would expect from a random assignment of points to the k clusters.

3 Spectral Clustering [5 pts]

Here you will run through the spectral clustering algorithm by hand. Figure 2 shows the graph that we will be working with.

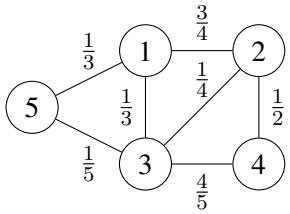


Figure 2: Spectral Clustering graph

(a) [1 pt] Compute our similarity matrix W, degree matrix D, and the corresponding graph Laplacian L based on the graph in Figure 2.

(b) [1 pt] Using the `scipy.linalg.eigh` method to solve for the eigenvectors corresponding to the $k = 2$ lowest eigenvalues. List out the eigenvalues and the corresponding eigenvectors (round to four decimal places). Refer to the documentation for more details: https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.eigh.html.

(c) [1 pt] Based on the eigenvalues and the eigenvectors computed in part b, what is our cluster assignment? (Hint: as in lecture 16, plot the rows of the eigenvectors and the clusters should be clear)

(d) [1 pt] Complete problem b again but with $k = 3$. List out the eigenvalues and the corresponding eigenvectors (round to four decimal places).

(e) [1 pt] What are our cluster assignments now?

3.1 Code Prerequisites

- If you are currently using `scipy` version 1.2.1 (or later), you should be sufficiently up-to-date. You can verify this by running `conda list scipy`
- If not, please run `conda update scipy` to update

4 Collaborative Filtering [12 points]

4.1 K-Nearest Neighbors

Recall from class that we can predict a user's rating of an item using the ratings from similar users. As it turns out, we can also predict item ratings using users' ratings on similar *items*, an approach called "item-item collaborative filtering" that was patented by Amazon in 1998 (U.S. Patent 6,266,649 B1) and published

in 2001 ("Item-Based Collaborative Filtering Recommendation Algorithms" by Badrul Sarwar, et al.). In other words, given user a and item i that a has not rated, we can predict a's rating on i using the ratings that a has made on items similar to i.

(a) [1 pt] Assume you have n users and m items in your database. Your ratings are stored in a matrix $X \in \mathbb{R}^{n \times m}$, where X_{ai} is user a's rating on item i. Let $\bar{r}^{(j)}$ be the jth row vector in X and $\bar{c}^{(j)}$ be the jth column vector in X. Which vector would you use to represent item k?

(b) [2 pts] Recall the Pearson correlation coefficient that we used to calculate similarity between two users a and b:

$$sim(a,b) = \frac{\sum_{j \in R(a,b)} \left(Y_{aj} - \tilde{Y}_{a:b}\right)\left(Y_{bj} - \tilde{Y}_{b:a}\right)}{\sqrt{\sum_{j \in R(a,b)} \left(Y_{aj} - \tilde{Y}_{a:b}\right)^2 \sum_{j \in R(a,b)} \left(Y_{bj} - \tilde{Y}_{b:a}\right)^2}}$$

Here, $R(a,b)$ is the set of items that both a and b have rated and $\tilde{Y}_{a:b}$ is the average rating by user a on items rated both by a and b. We want to edit this formulation to fit the item-item case so that we compute the similarity between items i and j. How would you redefine $R(i,j)$ and $\tilde{Y}_{i:j}$?

(c) [2 pts] Assume you have the following dataset, defined as in part a:

$$X = \begin{bmatrix} 1 & 0 & 5 \\ 3 & 4 & 3 \end{bmatrix}$$

The items are rated on a 5-point scale, with 0 indicating a rating that does not exist yet. Find the similarity between items 1 and 3 and provide a short interpretation of the result.

4.2 UV Decomposition

We can think of collaborative filtering as matrix factorization. Any $n \times m$ matrix X can be approximated by a lower rank matrix UV^T where U is $n \times d$ and V is $m \times d$ and $d \leq \min\{m, n\}$. Let $\bar{u}^{(i)} \in \mathbb{R}^d$ be the i^{th} row vector of U and $\bar{v}^{(j)}$ be the j^{th} row vector of V. Then $[UV^T]_{ij} = \bar{u}^{(i)} \cdot \bar{v}^{(j)}$. We want to use this low rank matrix to approximate observed binary labels in Y. Let $Y_{ij} \in \{-1, 1\}$ if the (i,j) entry is observed, and $Y_{ij} = 0$ otherwise. Then we would like to find U and V such that:

$$Y_{ij}[UV^T]_{ij} = Y_{ij}(\bar{u}^{(i)} \cdot \bar{v}^{(j)}) > 0 \text{ whenever } Y_{ij} \neq 0$$

Note that this is trivially possible if $X = UV^T$ is full rank (each element can be chosen independently from each other). The smaller d we choose, the fewer adjustable parameters we have in U and V. Therefore, d controls the complexity of the solution that satisfies the constraints.

It is often advantageous to potentially avoid satisfying all the constraints, e.g., if some of the labels may be mistakes. We might also suspect that we gain something by turning the estimation problem into a maximum

margin problem, for $\bar{u}^{(i)}$'s given $\{\bar{v}^{(j)}\}$ and for $\bar{v}^{(j)}$'s given $\{\bar{u}^{(i)}\}$. The formulation with slack is given by:

$$\begin{aligned} \min_{U,V,\xi} \quad & \sum_{i=1}^{n} \frac{1}{2}\|\bar{u}^{(i)}\|^2 + \sum_{j=1}^{m} \frac{1}{2}\|\bar{v}^{(j)}\|^2 + C \sum_{i,j:Y_{ij}\neq 0} \xi_{ij} \\ \text{subject to} \quad & Y_{ij}(\bar{u}^{(i)} \cdot \bar{v}^{(j)}) \geq 1 - \xi_{ij}, \\ & \xi_{ij} \geq 0 \text{ for all } (i,j) \text{ where } Y_{ij} \neq 0 \end{aligned}$$

(a) [2 pts] If we fix U, i.e., fix all $\bar{u}^{(1)}, ..., \bar{u}^{(n)}$ the optimization problem reduces to m independent optimization problems. Define these optimization problems for $\bar{v}^{(j)}$ where $j = 1, ..., m$. What do these optimization problems correspond to?

(b) [1 pt] If we fix V, what does the optimization problem reduce to?

(c) [4 pt] Consider a simple two-user and two-movie example and let the observed rating matrix and our initial guess for V be given by:

$$Y = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

i. Solve by hand $\hat{U} = [\bar{u}^{(1)}, \bar{u}^{(2)}]^T$, given the initial V. Given this V and your solution $\hat{U}$, can you predict Y_{22}?

ii. Solve by hand $\hat{V} = [\bar{v}^{(1)}, \bar{v}^{(2)}]^T$, given your solution $\hat{U}$. Can you predict Y_{22} now? If so, what is the resulting prediction?

REMEMBER Submit your completed assignment to Gradescope by 11:59pm on Tuesday, November 26th. Attach any code as an appendix.