

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

Homework 3, Due: Tues. 04/02 at 11:59pm

Submission: Please upload your completed assignment by 11:59pm Tuesday, Apr. 2nd to Gradescope.

1 Training Neural Nets [5 pts]

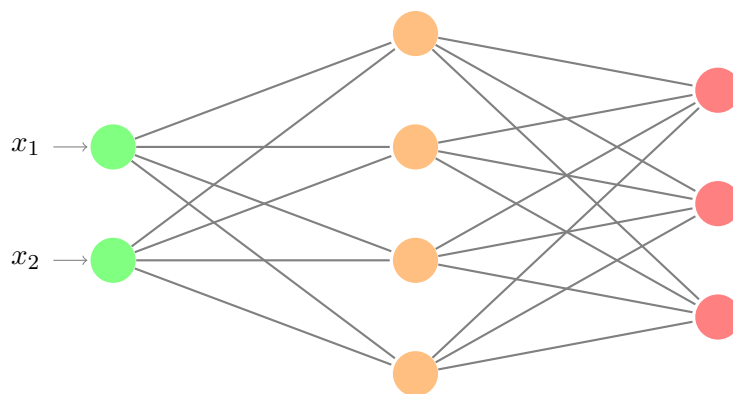


Figure 1: A two-layer neural network

Training a neural network is computationally expensive. For that reason, we often look for techniques to reduce the computational complexity of training. Imagine you are training the neural network pictured in Figure 1. Let the loss function be categorical cross-entropy and the activation function on the nodes in the output layer (the red nodes) be the softmax function. Combining backpropagation through the softmax activation function and the cross-entropy loss—as is done in `torch.nn.CrossEntropyLoss`—is one technique used to speed up training. Let t be the true label of an example and c be the predicted label of an example, with $t, c \in \{1, 2, 3\}$. Let \bar{z} be the network output logits and \hat{y}_c the probability that the network assigns to the input example as belonging to class c . Recall the softmax activation function, which transforms our network logits into a vector specifying class probabilities.

$$\hat{y}_c = S_c(\bar{z}) = \frac{\exp(z_c)}{\sum_{j=1}^3 \exp(z_j)}$$

In this problem, you will show that the computational complexity of the joint backpropagation through the softmax and cross-entropy functions is less than the alternative of independently backpropagating through each function. Express all of your answers only in terms of $t, c, k, \hat{y}_c, \hat{y}_k, y_c$, and y_k .

- (a) [1 pt] Let $y \in \mathbb{R}^3$ be a one-hot encoded vector with a one in the position of the true label and zero otherwise (i.e., if the true label is t , then $y_c = 1$ if $t = c$ and $y_c = 0$ if $t \neq c$). We define the

cross-entropy function as

$$\mathcal{L}(y, \hat{y}) = - \sum_{c=1}^3 y_c \log \hat{y}_c$$

Find $\frac{\partial \mathcal{L}}{\partial \hat{y}_c}$, the gradient of the cross-entropy loss with respect to the network prediction for class c .

- (b) [2 pts] Find $\frac{\partial \hat{y}_c}{\partial z_k}$, the gradient of the softmax prediction for class c with respect to network logit z_k . Hint: divide this problem into cases for $c = k$ and $c \neq k$.
- (c) [2 pts] Find $\frac{\partial \mathcal{L}}{\partial z_k}$, the gradient of the cross entropy loss with respect to network logit z_k . Hint: write the result from part (b) as a single expression using an indicator function and apply chain rule.

2 Clustering [10 pts]

Here you will simulate how the k -means clustering algorithm functions. Figure 2 shows an initialization of three means for the k -means algorithms (i.e. $k = 3$). Note that all of the points are in \mathbb{R}^2 and the gridlines are unit-sized.

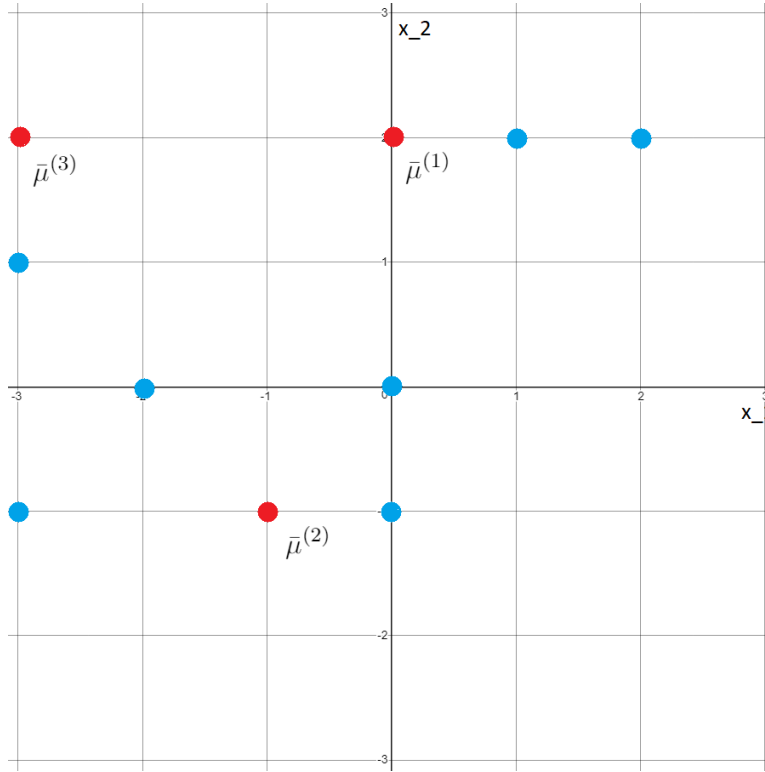


Figure 2: Blue — points, Red — means

- [3 pts] What are the numerical values of the means $\bar{\mu}^{(1)}$, $\bar{\mu}^{(2)}$, and $\bar{\mu}^{(3)}$ after one update of the means (use the initialization provided in Figure 2)? Mark the positions of the updated means in Figure 2 and label them.
- [1 pt] Given the initialization in Figure 2, how many k -means iterations (updates of the means) will be required until the algorithm converges?
- [4 pts] We can modify the k -means algorithm slightly by biasing the selection of the cluster means. We can regularize how the cluster means are set. In particular, consider a cluster C (a set of indices of training points). We will obtain the new mean for this cluster by minimizing:

$$\|\bar{\mu}\|^2 + \sum_{i \in C} \|\bar{x}^{(i)} - \bar{\mu}\|^2$$

with respect to vector $\bar{\mu}$. What is the resulting optimal $\bar{\mu}$ in terms of $\bar{x}^{(i)}$ and $|C|$ (the size of the cluster)?

- [2 pt] We can understand the new setting of the cluster mean in part (c) as a standard k -means update with additional point(s) in the cluster C . What is/are these point(s) that we should add to the cluster C so as to obtain the same value for the mean vector $\bar{\mu}$?

3 Landmark Clusters [18 pts]

Thanks to all of your hard work in Project 2, Steven has begun traveling the world and filling his Instagram with perfectly captioned photos. As he continues using his adventure, he now wants to save his photos into proper albums for later editing. Unfortunately, Steven lost the labels so you have promised to create Steven an *unsupervised* algorithm for placing images into their proper albums.

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_centroids()` 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 landmarks dataset. For the sake of reducing computation time, we will consider only the first 400 images from the dataset. 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 3:

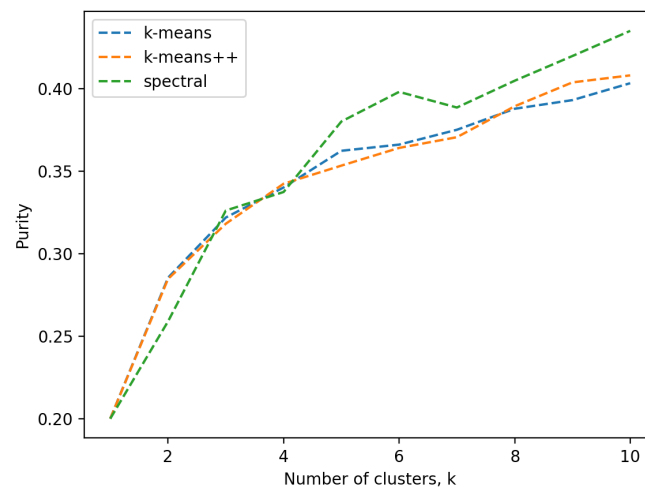


Figure 3: Purity of each clustering method as a function of k

- (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.

4 Spectral Clustering [5 pts]

Here you will run through the spectral clustering algorithm by hand. Figure 4 shows the graph that we will be working with.

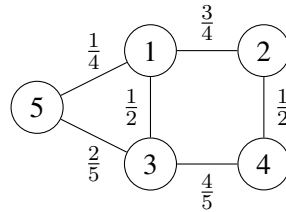


Figure 4: Spectral Clustering graph

- (a) [1 pt] Compute our similarity matrix W , degree matrix D , and the corresponding graph Laplacian L based on the graph in Figure 4.
- (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?

REMEMBER Submit your completed assignment to Gradescope by **11:59pm on Tuesday, April 2nd**. Attach any code as an appendix.