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]

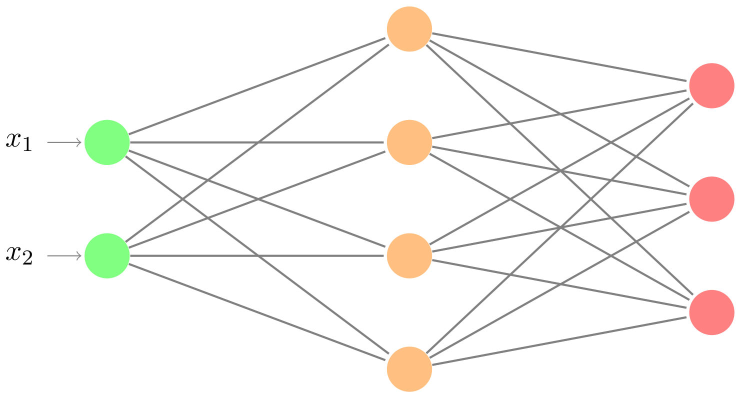

Figure 1: A two-layer neural network

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.

Solution:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \hat{y}_c} &= -\frac{y_c}{\hat{y}_c} \\ &= -\frac{[[c == t]]}{\hat{y}_c}\end{aligned}$$

- 1 pt - Correct solution

(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 $c \neq k$.

Solution:

$$\begin{aligned}\frac{\partial \hat{y}_c}{\partial z_k} &= \frac{\left([[c == k]]e^{z_k}\sum_j e^{z_j}\right) - e^{z_c}e^{z_k}}{(\sum_j e^{z_j})^2} && \text{Quotient Rule} \\ &= \frac{[[c == k]]e^{z_k}\sum_j e^{z_j}}{(\sum_j e^{z_j})(\sum_j e^{z_j})} - \frac{e^{z_c}e^{z_k}}{(\sum_j e^{z_j})^2} \\ &= [[c == k]]\frac{e^{z_k}}{\sum_j e^{z_j}} - \left(\frac{e^{z_c}}{\sum_j e^{z_j}}\right)\left(\frac{e^{z_k}}{\sum_j e^{z_j}}\right) && \text{re-group fraction as product} \\ &= [[c == k]]\hat{y}_k - \hat{y}_c\hat{y}_k \\ &= \boxed{\hat{y}_k([[c == k]] - \hat{y}_c)} \\ &= \begin{cases} -\hat{y}_c\hat{y}_k & c \neq k \\ \hat{y}_k(1 - \hat{y}_c) & c = k \end{cases}\end{aligned}$$

- 1 pt - Attempt shown
- 1 pt - Correct solution with derivation

(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.

Solution:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial z_k} &= \sum_c \frac{\partial \mathcal{L}}{\partial \hat{y}_c}\frac{\partial \hat{y}_c}{\partial z_k} \\
&= -\sum_c \frac{[[c == t]]}{\hat{y}_c}\hat{y}_k([[c == k]] - \hat{y}_c) \\
&= -([[k == t]] - \hat{y}_k) \\
&= \hat{y}_k - y_k
\end{aligned}$$

- 1 pt - Attempt shown
- 1 pt - Correct solution with derivation

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.

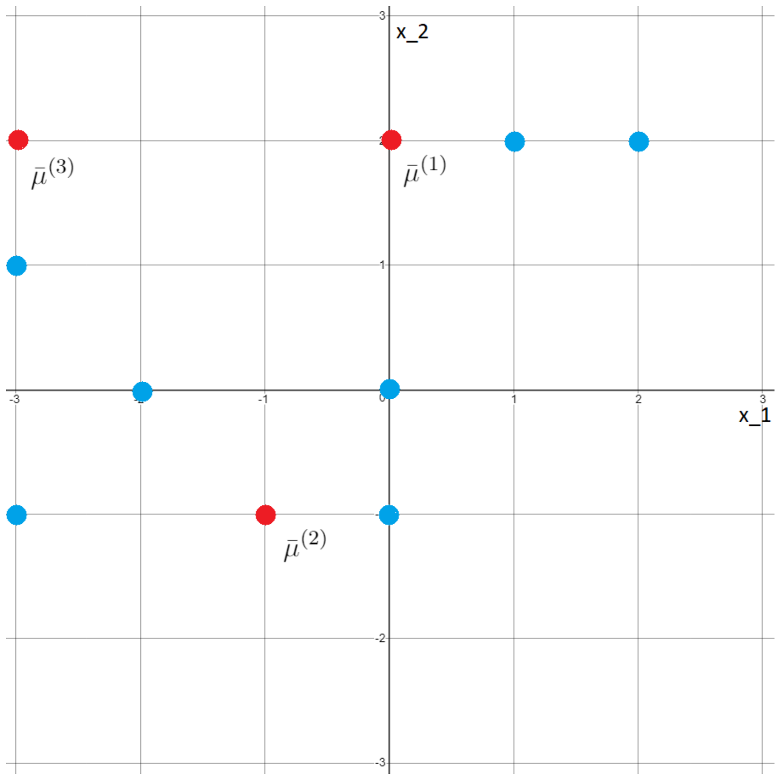

Figure 2: Blue — points, Red — means

(a) [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.

Solution: $\bar{\mu}^{(1)} = [1.5, 2]^T$, $\bar{\mu}^{(2)} = [-1.25, -0.5]^T$, and $\bar{\mu}^{(3)} = [-3, 1]^T$

- 1 pt - Attempt shown
- 2 pts - Correct solution

(b) [1 pt] Given the initialization in Figure 2, how many k-means iterations (updates of the means) will be required until the algorithm converges?

Solution: 1 or 2 iterations (2 if you counted the 2nd loop through the data where the means did not change as an iteration, 1 if you did not count the final loop).

- 1 pt - Correct solution

(c) [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)?

Solution:

$$\begin{aligned}
\nabla_{\bar{\mu}} \left(\|\bar{\mu}\|^2 + \sum_{i \in C} \|\bar{x}^{(i)} - \bar{\mu}\|^2 \right) &= 0 \\
2\bar{\mu} + \sum_{i \in C} 2(\bar{\mu} - \bar{x}^{(i)}) &= 0 \\
(|C| + 1)\bar{\mu} - \sum_{i \in C} \bar{x}^{(i)} &= 0 \\
\bar{\mu}^* &= \frac{\sum_{i \in C} \bar{x}^{(i)}}{|C| + 1}
\end{aligned}$$

- 1 pt - Attempt shown
- 1 pt - Gradient with respect to $\bar{\mu}$
- 2 pts - Correct solution with derivation

(d) [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 $\bar{\mu}$?

Solution: The standard k-means update is:

$$\bar{\mu} = \frac{\sum_{i \in C} \bar{x}^{(i)}}{|C|}$$

If we add points $\bar{x}^{(0)} = [0, 0]^T$ to the cluster and perform the standard k-means update (average of points), we get:

$$\bar{\mu} = \frac{\bar{x}^{(0)} + \sum_{i \in C} \bar{x}^{(i)}}{1 + |C|} = \frac{\sum_{i \in C} \bar{x}^{(i)}}{1 + |C|}$$

- 2 pts - Reasonable solution (doesn't have to be the same as the solution here)

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?

Solution: Poor centroid initialization could cause the model to converge to an arbitrarily bad local minimum. Specifically for k-means, each iteration makes a greedy update with no regard for global optimality.

- 1 pt - Reasonable explanation

(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.

Solution: See solution code `Point.distance()`.

- 1 pt - Correct code

(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`.

Solution: See solution code `Cluster.get_centroid()`.

- 2 pts - Correct code

(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.

Solution: See solution code `ClusterSet.get_centroids().`

- 1 pt - Correct code

(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.

Solution: See solution code `random_init().`

- 1 pt - Correct code

(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.
```

Solution: See solution code `k_means_pp_init().`

- 2 pts - Correct code

(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.

Solution: See solution code `k_means()`.

- 1 pt - Attempt shown
- 3 pts - Correct code

(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:

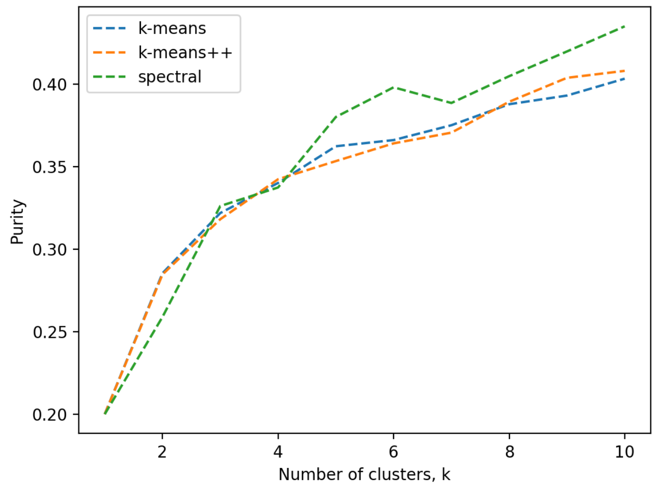

Figure 3: Purity of each clustering method as a function of k

Solution: See solution code `plot_performance()`. See Figure 3 for plot.
We look for an 'elbow' in the graph. A reasonable answer for the graph above is $k = 3$ — although one could claim that any $k = 3, 4, 5, 6$ is the best for k. Your specific answer must reflect the elbow in your plot for points to be awarded. Note that purity will approach 1 as k approaches the number of points since we can simply learn 1 cluster per point. In general the k-means++ should be above the k-means line. The performance of the two algorithms, however, is very close, and the lines may cross.

- 1 pt - Reasonable best k value for each method based on 'elbow'
- 2 pts - Reasonable plot
- 1 pt - Correct code (if graph looks similar, code is probably correct)

(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.

Solution: See table below for best k value choices and the corresponding purity values. Your best k could be different, but it should in general be around 3 to 6.

Algorithm	**best k**	**Average**	**Max**	**Min**
k-means	3	0.3220	0.3125	0.3325
k-means++	3	0.3183	0.3075	0.3300
Spectral	3	0.3263	0.3250	0.3275

Since we utilize 5 classes of landmark images, the purity score of random cluster assignments is 0.2. The above clustering results are clearly better than random.

- 1 pt - Reasonable purity values for each method
- 1 pt - Reasonable explanation and comparison

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.

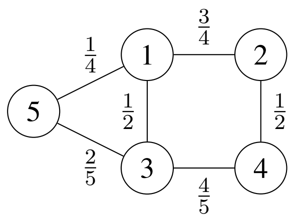

Figure 4: Spectral Clustering graph

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

Solution: We first construct our similarity matrix W and the corresponding degree matrix D.

$$W = \begin{bmatrix} 1 & \frac{3}{4} & \frac{1}{2} & 0 & \frac{1}{4} \\ \frac{3}{4} & 1 & 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 1 & \frac{4}{5} & \frac{2}{5} \\ 0 & \frac{1}{2} & \frac{4}{5} & 1 & 0 \\ \frac{1}{4} & 0 & \frac{2}{5} & 0 & 1 \end{bmatrix} \qquad D = \begin{bmatrix} \frac{5}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{9}{4} & 0 & 0 & 0 \\ 0 & 0 & \frac{27}{10} & 0 & 0 \\ 0 & 0 & 0 & \frac{23}{10} & 0 \\ 0 & 0 & 0 & 0 & \frac{33}{20} \end{bmatrix}$$

From these matrices, we compute our graph Laplacian.

$$L = \begin{bmatrix} \frac{3}{2} & -\frac{3}{4} & -\frac{1}{2} & 0 & -\frac{1}{4} \\ -\frac{3}{4} & \frac{5}{4} & 0 & -\frac{1}{2} & 0 \\ -\frac{1}{2} & 0 & \frac{17}{10} & -\frac{4}{5} & -\frac{2}{5} \\ 0 & -\frac{1}{2} & -\frac{4}{5} & \frac{13}{10} & 0 \\ -\frac{1}{4} & 0 & -\frac{2}{5} & 0 & \frac{13}{20} \end{bmatrix}$$

- 1 pt - Correct matrices

(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.

Solution:
1. Eigenvalue: $2.9106 \times 10^{-16} \approx 0$, Eigenvector: $[-0.4472, -0.4472, -0.4472, -0.4472, -0.4472]^T$
2. Eigenvalue: $6.7456 \times 10^{-1} = 0.6746$, Eigenvector: $[-0.1214, -0.4294, 0.0244, -0.3120, 0.8384]^T$

- 1 pt - Correct solution

(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)

Solution: Following the hint, we can plot the rows of the eigenvectors and obtain the plot in Figure 5.

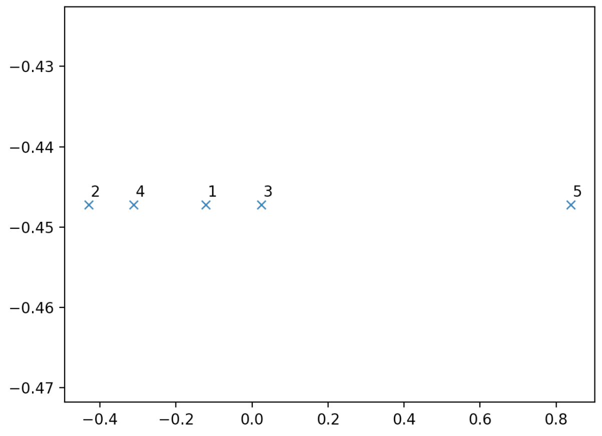

Figure 5: Plotting rows of the eigenvectors computed in part b

From this plot, it is obvious that the cluster assignments should be points 1, 2, 3, 4 in one cluster and point 5 in the other cluster.

- 1 pt - Correct cluster assignments

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

Solution:
1. Eigenvalue: $2.9106 \times 10^{-16} \approx 0$, Eigenvector: $[-0.4472, -0.4472, -0.4472, -0.4472, -0.4472]^T$
2. Eigenvalue: $6.7456 \times 10^{-1} = 0.6746$, Eigenvector: $[-0.1214, -0.4294, 0.0244, -0.3120, 0.8384]^T$
3. Eigenvalue: 1.1498, Eigenvector: $[0.4758, 0.4641, -0.4066, -0.6207, 0.0874]^T$

UPDATE: Since the first eigenvalue is 0, the corresponding eigenvector can be either $[-0.4472, -0.4472, -0.4472, -0.4472, -0.4472]^T$ or $[0.4472, 0.4472, 0.4472, 0.4472, 0.4472]^T$. Both are correct.

- 1 pt - Correct solution

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

Solution: Following the hint in part c, we can plot the rows of the eigenvectors and obtain the plot in Figure 6.

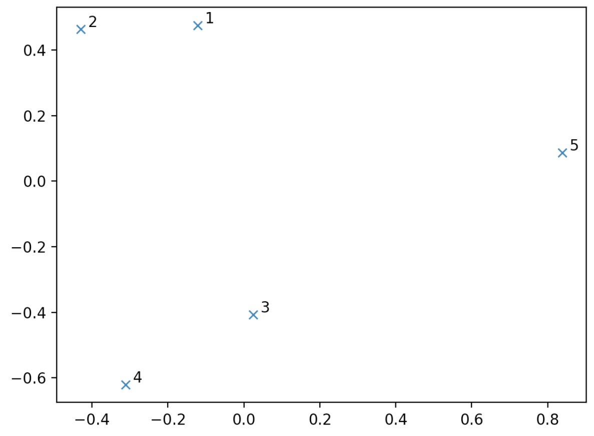

Figure 6: Plotting rows of the eigenvectors computed in part d

From this plot, it is obvious that the cluster assignments should be points 1 and 2 in one cluster, points 3 and 4 in one cluster, and point 5 in the other cluster.

- 1 pt - Correct cluster assignments

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