Solution Homework Set #4

Problem 1 (PCA [10 pts])

You have the following data:

data #	1	2	3	4	5	6	7	8
x_1	5.51	20.82	-0.77	19.30	14.24	9.74	11.59	-6.08
x_2	5.35	24.03	-0.57	19.38	12.77	9.68	12.06	-5.22

You want to reduce the data into a single-dimension representation. You are given the first principal component $\mathbf{v}_1 = (-0.694, -0.720)$.

(a) **[4 points]** What is the representation (projected coordinate) along the first principal direction for data $\mathbf{x}^{(1)}$?

Answer: We project $\mathbf{x}^{(1)}$ to the first principal component $\mathbf{v}_1$

$$y_1^{(1)} = \mathbf{v}_1^T \mathbf{x}^{(1)} = -0.694 * 5.51 - 0.720 * 5.35 = -7.675939999$$

(b) **[3 points]** What are the feature values in the original space reconstructed using this first principal representation $y_1^{(1)}$?

Answer: The reconstruction can be done by

$$(\mathbf{v}_1^T)^T y_1^{(1)} = (-0.694 * -7.675939999, -0.720 * -7.675939999) = (5.33, 5.52)$$

(c) **[3 points]** What is the principal component score for $\mathbf{x}^{(1)}$ along the second principal direction? You can answer up to a change of sign.

Answer: There are 2 ways to solve this question. One is to explicitly compute the second principal component using eigendecomposition (requires non-trivial numerical computation). The other is based on the observation that for 2 dimensional data, the first 2 components should explain the entire variance in the dataset. Hence, we can recover the second principal component score as:

$$y_2^{(1)} = \pm\sqrt{((\mathbf{x}_1^{(1)})^2 + (\mathbf{x}_2^{(1)})^2 - (y_1^{(1)})^2)} = \pm\sqrt{((5.51)^2 + (5.35)^2 - (7.67594)^2)} = \pm 0.25$$

Problem 2 (Neural Networks [10 pts])

Suppose you have the following neural network described by the function $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \max(w_1 x_1 + w_2 x_2 + b, 0)$, where $\boldsymbol{x} = (x_1, x_2)^T \in \mathbb{R}^2$ is the input to the network and $\boldsymbol{\theta} = (w_1, w_2, b)^T \in \mathbb{R}^3$ are the network's parameters.

(a) **[3 points]** Draw the computation graph for $f_{\boldsymbol{\theta}}(\boldsymbol{x})$.

Answer:

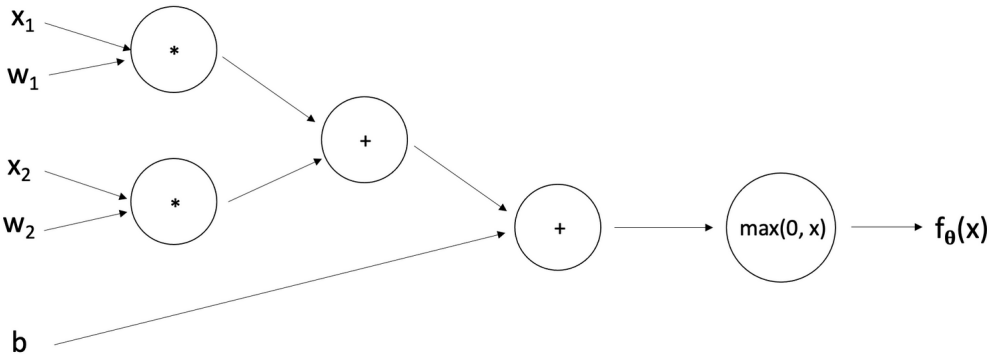

Figure 1: Computation Graph for Neural Network

(b) **[3 points]** Let $w_1 = 2, w_2 = 1, x_1 = 3, x_2 = -2, b = -1$. Compute the forward pass for $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ (i.e. compute $f_{\boldsymbol{\theta}}(\boldsymbol{x})$).

Answer: See Figure 2 for computation graph with forward pass and backward pass values.

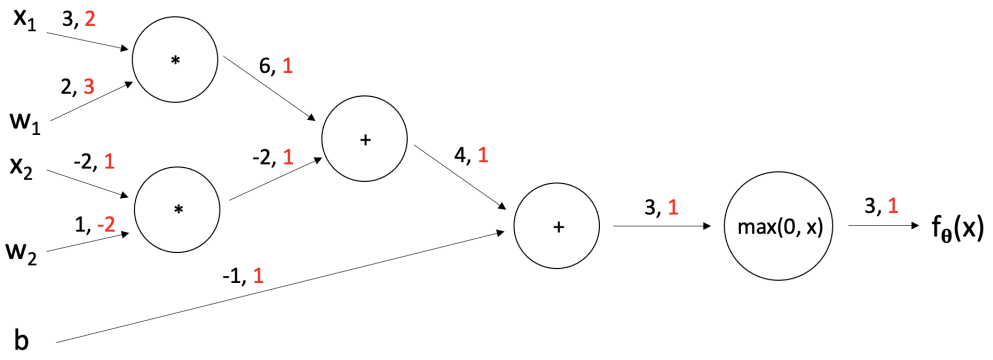

Figure 2: Computation Graph for Neural Network. Forward pass values are in black, and backward pass values are in red.

Alternatively, we can just compute $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \max(w_1x_1 + w_2x_2 + b, 0) = \max(2*3+1*(-2)-1, 0) = \max(3, 0) = 3$.

(c) **[6 points]** Let $w_1 = 2, w_2 = 1, x_1 = 3, x_2 = -2, b = -1$. Compute the backward pass for $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ with respect to w_1, w_2, b (i.e. compute $\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_1}, \frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_2}, \frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial b}$).

Answer: See Figure 2 for computation graph with forward pass and backward pass values. By the chain rule,

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_1} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_1}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)}, \tag{1}$$

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_2} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_2}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)}, \tag{2}$$

and

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial b} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial b}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)}. \tag{3}$$

We note that the derivative of $\text{ReLU}(x) = \max(x, 0)$ is

$$\frac{\partial(\max(x, 0))}{\partial x} = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0, \end{cases}$$

so as $w_1x_1 + w_2x_2 + b = 3$ (from part b), we have that

$$\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)} = 1.$$

We also note that

$$\frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_1} = x_1 = 3,$$

$$\frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_2} = x_2 = -2,$$

and

$$\frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial b} = 1.$$

Substituting these values into equations 1, 2, 3, we get

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_1} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_1}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)} = 3(1) = 3,$$

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial w_2} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial w_2}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)} = -2(1) = -2,$$

and

$$\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{x})}{\partial b} = \frac{\partial(w_1x_1 + w_2x_2 + b)}{\partial b}\frac{\partial(\max(w_1x_1 + w_2x_2 + b, 0))}{\partial(w_1x_1 + w_2x_2 + b)} = 1(1) = 1.$$

Problem 3 (A Two-Layer Neural Network for Binary Classification [12 pts])

In this exercise, we will walk through the forward and backward propagation process for a two-layer fully connected neural network. We will use the same data as in Homework 1, the MNIST dataset. The data you will be using is under the `MNIST` folder.

(a) **(0 pts)** Load and visualize the data by running the cells for Part (a) in the notebook.

(b) **(2 pts)** Implement the forward pass of the two-layer feed forward neural network with a ReLU non-linear layer. Specifically, implement code of "part (b)" to compute `scores` in the `loss()` function, take a screenshot of your code and paste it here. Further information and guidance are provided in the comments of the code.
(*Hint*: Our network has four set of parameters to be updated: First layer weights, first layer biases, second layer weights, and second layer biases. The weights and bias parameters are initialized in the `__init__()` function of the `TwoLayerNet` class.)

(c) **(1 pts)** What is the formula for calculating ℓ_2 regularization? Write it down here only the regularization term with factor $\frac{\lambda}{2}$. Then, implement the ℓ_2 regularization term in "part (c)", take a screenshot of your code and paste it here.
Solution: regLoss $= \frac{\lambda}{2}\left(\|w_1\|_2^2 + \|w_2\|_2^2\right)$. It is also correct if in the answer $\lambda = 1$.

(d) **(2 pts)** Recall that we can express the final hypothesis for a binary classifier using sigmoid (only 1 output unit) or softmax (2 output units). Implement the softmax cross entropy loss and the gradient w.r.t the inputs in the function `softmax_loss(x,y)`, take a screenshot of your code and paste it here.
Suppose your softmax cross entropy loss function takes $\boldsymbol{X}$ and $\boldsymbol{Y}$ as inputs. Specifically, input $\boldsymbol{X}$ is the output score after the second layer, where $x_j^{(i)}$ is the output score of sample i being in class j; input $\boldsymbol{Y}$ is the target label represented as one-hot vectors, where $y^{(i)} \in \mathbb{R}^C$ is the one-hot encoded label of sample i, e.g, for binary classification, $y^{(i)} = [0, 1]$ or $y^{(i)} = [1, 0]$. What is the formula for calculating the softmax cross-entropy loss and gradient? Write it down here. You can write your formula using $x_j^{(i)}$ and $y_j^{(i)}$.
Solution:

$$\text{CE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{C} y_j^{(i)} \log(p_j^{(i)}), \tag{4}$$

where $y_i \in \mathbb{R}^C$ is the one-hot encoded label of sample i, and $p_j^{(i)} = \text{softmax}(x^{(i)})_j = \frac{\exp(x_j^{(i)})}{\sum_{k=1}^{C}\exp(x_k^{(i)})}$ is the predicted softmax probability of sample i being in class j. The gradient of the softmax cross-entropy with respect to the scores s_i is:

$$\text{gradient} = \frac{1}{n}\sum_{i=1}^{n}(p^{(i)} - y^{(i)}). \tag{5}$$

(e) **(3 pts)** Implement the back-propagation process by completing the gradient computation for W2 and b2 in "part (e)", take a screenshot of your code and paste it here. What is the formula for calculating the gradient of W2 and b2? Write it down here.
Solution:
grads['W2'] = a1.T.dot(dscore) + reg * W2
grads['b2'] = np.ones(N).dot(dscore)

(f) **(4 pts)** Implement the prediction function in "part (f)" and check the correctness of your implementation above by running the predictions on validation and test sets. Adjust the learning rate in $10^{-5}, 10^{-4}, 10^{-3}, 5 \times 10^{-3}, 10^{-1}$. Report the best accuracy you get and the corresponding learning rate. Briefly discuss your observations of using different learning rates.

Solution: Learning rate with 1e-5 doesn't converge, while with 0.1 diverges. Learning rate with 1e-3 is the best setting, test accuracy is 0.9683 (a value between 0.95 and 0.97 is acceptable).

Problem 4 (k-means Clustering [15 pts])

In this problem, we shall apply k-means Algorithm to a popular visual classification dataset CIFAR-10 [1]. CIFAR-10 dataset consists of 60K labeled 32 x 32 colored images i.e., each image would have 3 channels corresponding to RGB colors. Out of the total 60K images, 50K images belong to the training set and 10K belong to the testing set. Within the scope of this problem, we shall work with the testing set only for computational efficiency purposes. Additionally, as k-means is an unsupervised learning algorithm, we shall discard the labels associated with the images.

Please find the code skeleton for this problem in `Fall23-CS146-HW4.ipynb` notebook shared with you as a part of this HW.

A Implementing *k*-means

Rather than training k-means from scratch, we are going to make use of a pre-existing implementation available in scikit-learn.[2]

(a) **(5 pts)** Make use of the `dataloader()` function to load the data in our notebook. Within this function, the data is loaded using `datasets` package in Tensorflow. [3] **Tip**: Run this notebook in Google Colab as it saves you a lot of time spent in installing Tensorflow.

This function will return 10K testing images of CIFAR-10 dataset along with their ground truth label. We provide `visualize(X, ind)` function to visualize index `ind` in the data `X`. Before using k-means algorithm, we need to make sure that the data has the shape (10000, N), where N = 32 x 32 x 3, instead of (10000, 32, 32, 3). Your first task would be to implement `reshape` function that will convert the tensor form of the input to a 2D matrix of the form 10000 x N. Take a screenshot of your code and paste it here.

(b) **(5 pts)** Once you have reshaped your input, you are ready to apply k-means algorithm to your data. Note that the number of cluster centers k is a hyperparameter. As we are in the unsupervised regime, we do not have access to any labeled validation set that could be used to decide the best for k. Thus, we use some pre-defined metrics to decide what choice of k should be the best. In this HW, we are going to use the *Sum of squared distances of samples to their closest cluster center* as our score metric. However, we are also aware that

[1] https://www.cs.toronto.edu/~kriz/cifar.html

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[3] https://www.tensorflow.org/datasets/catalog/cifar10#:~:text=The%20CIFAR%2D10%20dataset%20consists,images%20and%2010000%20test%20images.&text=Versions%3A,3.0.

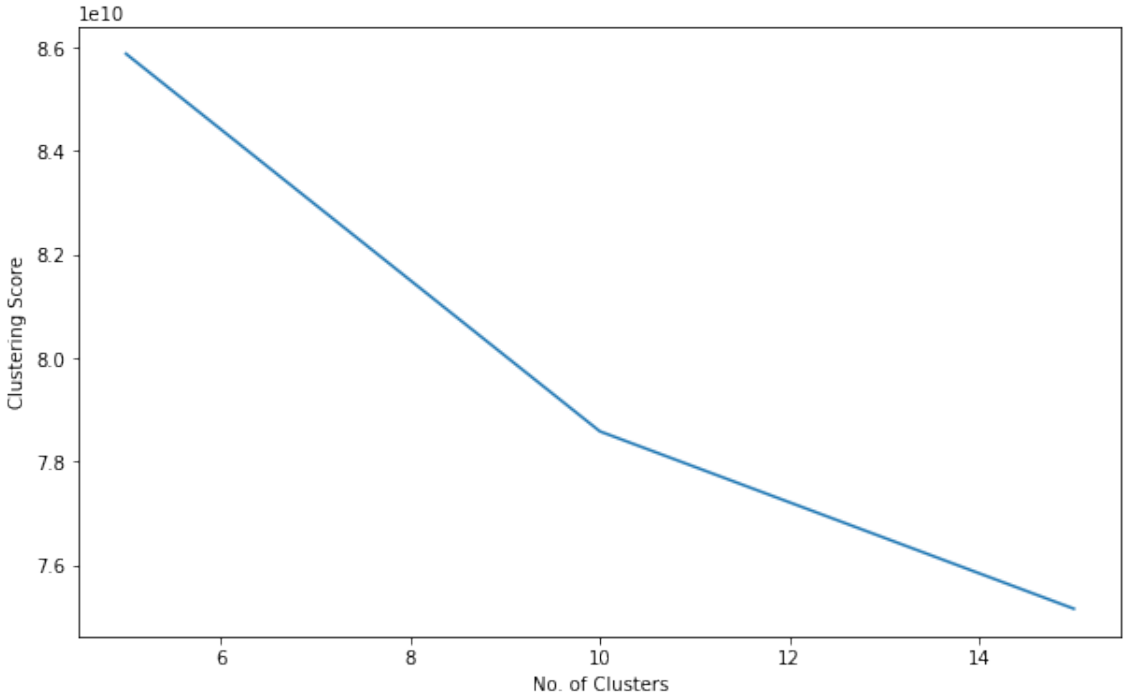

Figure 3: Plot

k-means algorithm can lead to different convergence based on the initialization of the clusters. To account for the differences between the score due to various initializations, we apply the algorithm across 3 random seeds (`random_state` in the code). Here, your task is to write a few lines of code to fit the k-means algorithm and calculate the score metric for all three runs, take a screenshot of your code and paste it here. Finally, submit the results graph you get after running the plotting block in the notebook that takes in your calculated scores as the input.

B Visualization

(5 pts) Another aspect of applying k-means algorithm involves visualizing the cluster assignments. As described above, the CIFAR-10 data is 3092 dimensional (32 x 32 x 3) which is quite impossible to visualize for us. However, we can project our high dimensional input data into lower dimensional space using PCA algorithm discussed in the class. Once you have the PCA output of the data, we can visualize the transformed low dimensional data easily.

We provide a simple implementation of PCA using its support in `sklearn.decomposition`. To get some sense of how PCA can be beneficial in plotting the high dimensional data, we provide a code block that provides you the PCA of the input data along with its ground truth labels. Running that block should output a plot which looks like Figure 4.

Your task is to first apply k-means algorithm with $k = 10$ and `random_state = 42` to the PCA-transformed data, and get the predicted label assignments to each data sample. Careful inspection of the k-means documentation should help you to get the predicted labels in one-line of code. Using these predicted labels, implement a code to get a scatter plot of the low-dimensional PCA data

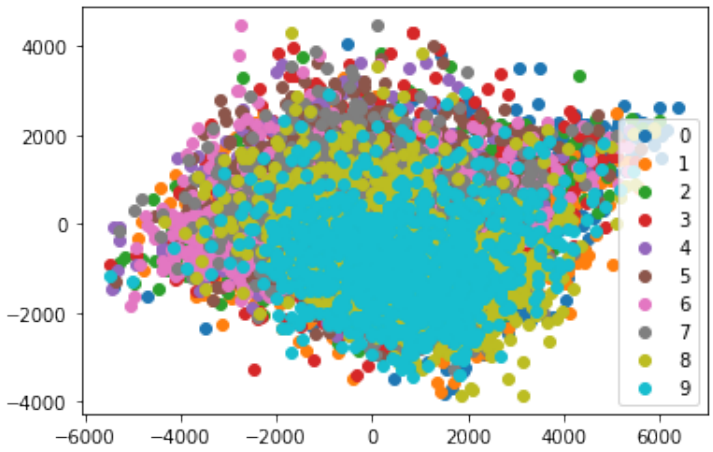

Figure 4: PCA of CIFAR-10 test data in 2-D with ground truth labels.

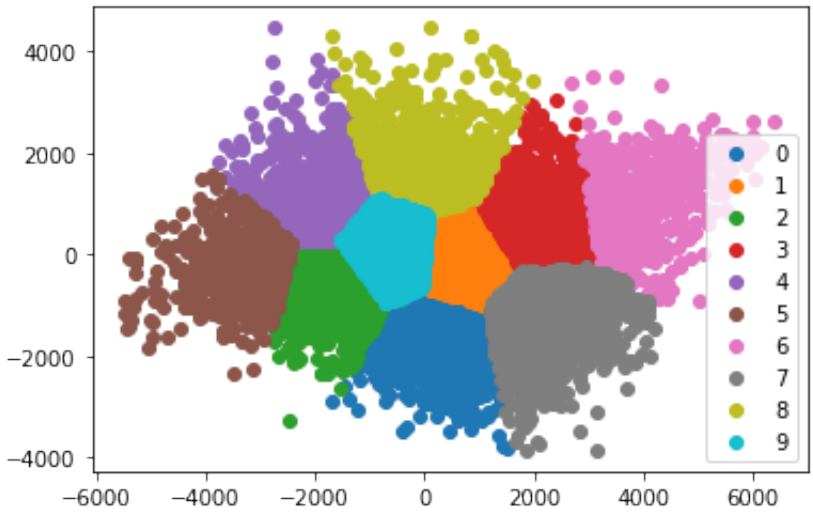

Figure 5: Visualization

with these new assignments. Take a screenshot of your code and paste it here. Submit the final graph you get for this part.