Name: Mariem Ahmed

ECE421 - Winter 2021
University of Toronto

Assignment 3:
Unsupervised Learning and Probabilistic Models

Due date: Monday, April 5, 2021

Submission: Submit both your report (a single PDF file) and all codes on Quercus.

Objectives:

In this assignment, you will implement learning and inference procedures for some of the probabilistic models described in class, apply your solutions to some simulated datasets, and analyze the results.

General Note:

- Full points are given for complete solutions, including justifying the choices or assumptions you made to solve each question. A written report should be included in the final submission.
- Programming assignments are to be solved and submitted **individually**. You are encouraged to discuss the assignment with other students, but you must solve it on your own.
- Please ask all questions related to this assignment on Piazza, using the tag `assignment3`.
- There are 3 starter files attached, `helper.py`, `starter_kmeans.py` and `starter_gmm.py` which will help you with your implementation.

NOTE: Some Code are used from these website as a guide for this Assignment

https://tslearn.readthedocs.io/en/stable/auto_examples/clustering/plot_kmeans.html (https://tslearn.readthedocs.io/en/stable/auto_examples/clustering/plot_kmeans.html)

https://austindavidbrown.github.io/post/2019/01/k-means-in-python/ (https://austindavidbrown.github.io/post/2019/01/k-means-in-python/)

https://www.kite.com/python/answers/how-to-remove-nan-values-from-a-numpy-array-in-python (https://www.kite.com/python/answers/how-to-remove-nan-values-from-a-numpy-array-in-python)

https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.04-Saving-Plots/ (https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.04-Saving-Plots/)

https://github.com/corvasto/Simple-k-Means-Clustering-Python/blob/master/kMeansClustering.py (https://github.com/corvasto/Simple-k-Means-Clustering-Python/blob/master/kMeansClustering.py)

https://github.com/ppai22/k-means-clustering-python (https://github.com/ppai22/k-means-clustering-python)

https://www.dummies.com/programming/big-data/data-science/how-to-visualize-the-clusters-in-a-k-means-unsupervised-learning-model/ (https://www.dummies.com/programming/big-data/data-science/how-to-visualize-the-clusters-in-a-k-means-unsupervised-learning-model/)

https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/ (https://www.machinelearningplus.com/predictive-modeling/k-means-clustering/)

https://realpython.com/k-means-clustering-python/#understanding-the-k-means-algorithm (https://realpython.com/k-means-clustering-python/#understanding-the-k-means-algorithm)

https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/ (https://www.datanovia.com/en/lessons/k-means-clustering-in-r-algorith-and-practical-examples/)

https://www.kernel-operations.io/keops/_auto_tutorials/kmeans/plot_kmeans_torch.html (https://www.kernel-operations.io/keops/_auto_tutorials/kmeans/plot_kmeans_torch.html)

https://jakevdp.github.io/PythonDataScienceHandbook/04.02-simple-scatter-plots.html (https://jakevdp.github.io/PythonDataScienceHandbook/04.02-simple-scatter-plots.html)

https://matplotlib.org/3.1.1/gallery/color/named_colors.html (https://matplotlib.org/3.1.1/gallery/color/named_colors.html)

1 K-means [9 pt.]

K-means clustering is one of the most widely used data analysis algorithms. It is used to summarize data by discovering a set of data prototypes that represent clusters of data. The data prototypes are usually referred to as cluster centers. Usually, K-means clustering proceeds by alternating between assigning data points to clusters and then updating the cluster centers. In this assignment, we will investigate a different learning algorithm that directly minimizes the K-means clustering loss function.

1.1 Learning K-means

The K cluster centers can be thought of as K, D-dimensional parameter vectors and we can place them in a $K \times D$ parameter matrix $\boldsymbol{\mu}$, where the k^{th} row of the matrix denotes the k^{th} cluster center $\boldsymbol{\mu}_k$. The goal of K-means clustering is to learn $\boldsymbol{\mu}$ such that it minimizes the loss function, $\mathcal{L}(\boldsymbol{\mu}) = \sum_{n=1}^{N} \min_{k=1}^{K} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2$, where N is the number of training observations.

Even though the loss function is not smooth due to the "min" operation, one may still be able to find its solutions through iterative gradient-based optimization. The "min" operation leads to discontinuous derivatives, in a way that is similar to the effect of the ReLU activation function, but nonetheless, a good gradient-based optimizer can work effectively.

1. Implement the `distanceFunc()` function in `starter_kmeans.py` file to calculate the squared pairwise distance for all pair of N data points and K clusters.

```
def distanceFunc(X, MU):
# Inputs
    # X: is an NxD matrix (N observations and D dimensions)
    # MU: is an KxD matrix (K means and D dimensions)
# Outputs
    # pair_dist: is the squared pairwise distance matrix (NxK)
```

Hint: To properly use broadcasting, you can first add a dummy dimension to `X`, by using `tf.expand_dims` or `torch.unsqueeze`, so that its new shape becomes `(N, 1, D)`.

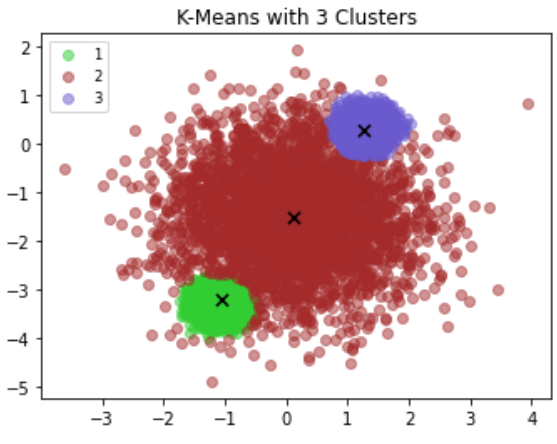
For the dataset `data2D.npy`, set $K = 3$ and find the K-means clusters $\boldsymbol{\mu}$ by minimizing the $\mathcal{L}(\boldsymbol{\mu})$ using the gradient descent optimizer. The parameters $\boldsymbol{\mu}$ should be initialized by sampling from the standard normal distribution. Include a plot of the loss vs the number of updates.

Use the Adam optimizer for this assignment with the following hyper-parameters: `learning_rate=0.1`, `beta1=0.9`, `beta2=0.99`, `epsilon=1e-5`. The learning should converge within a few hundred updates.

In [ ]:
```
# Distance function for K-means
def distanceFunc_Kmeans(X, MU):
    # Inputs
      # X: is an NxD matrix (N observations and D dimensions)
      # MU: is an KxD matrix (K means and D dimensions)
    # Outputs
      # pair_dist: is the squared pairwise distance matrix (NxK)
      # TODO
    X = tf.expand_dims(X, 1)
    MU = tf.expand_dims(MU, 0)
    pair_dist = tf.reduce_sum(tf.square(tf.subtract(X, MU)), 2)
    return pair_dist
```

In [ ]:
```

```

```
MU:
[[-1.05419931 -3.2284328 ]
 [ 0.12848035 -1.51165358]
 [ 1.25435577  0.25094891]]
```



2. Hold out 1/3 of the data for validation, and for each value of $K = 1, 2, 3, 4, 5$:

   (a) Train a K-means model.

   (b) Include a 2D scatter plot of training data points colored by their cluster assignments.

   (c) Compute and report the percentage of the training data points belonging to each of the K clusters. (include this information in the legend of the scatter plot.)

   (d) Compute and report the loss function over validation data. (include this in the caption of the scatter plot.)

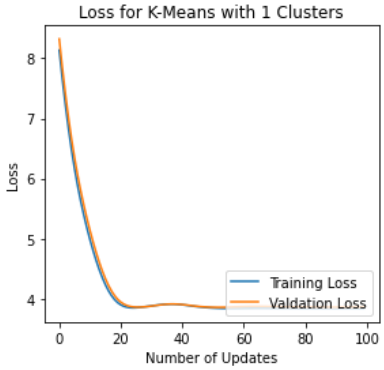
Based on the scatter plots, comment on the best number of clusters to use.

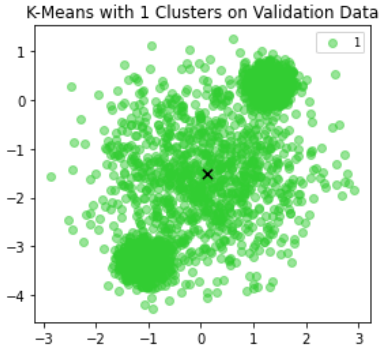
Answer:

Based on the Graphs below, it is noticeable that loss will decrease as mean (mu) increase. Therefore, 3-means gives the best result. This model is considered to be overfitting. k-means = 3, I believe it is the best k value for k-means on this data set.

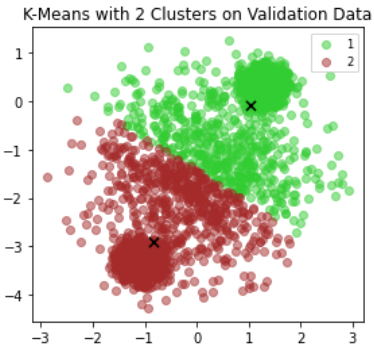
In [ ]:

```
K = 1
                                          Training                  Validat
ion
                Cluster 1:                100.00%                   100.0
0%

Final Value of Loss for both:             3.8382                    3.860
8
```



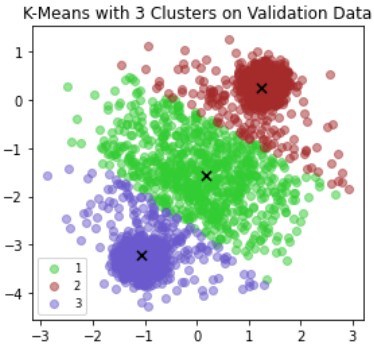



```
K = 2
                                          Training                  Validat
ion
                Cluster 1:                50.32%                    48.2
4%
                Cluster 2:                49.68%                    51.7
6%

Final Value of Loss for both:             0.9366                    0.888
2
```



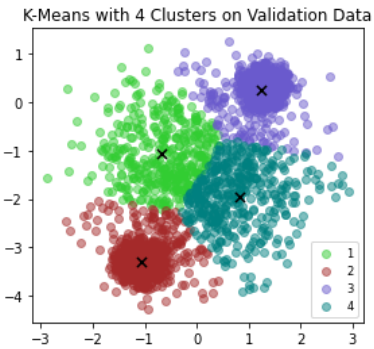
K = 3

	Training	Validation
Cluster 1:	24.03%	22.83%
Cluster 2:	38.41%	37.35%
Cluster 3:	37.56%	39.81%
Final Value of Loss for both:	0.5234	0.4890

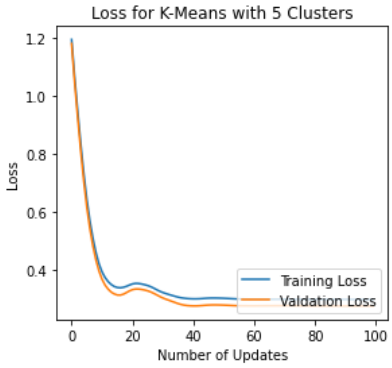


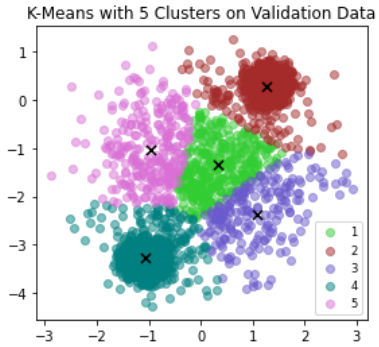
K = 4

	Training	Validation
Cluster 1:	12.21%	11.91%
Cluster 2:	36.19%	38.79%
Cluster 3:	37.66%	36.63%
Cluster 4:	13.93%	12.66%
Final Value of Loss for both:	0.3481	0.3164



```
K = 5
                                        Training                  Validat
ion
                Cluster 1:               11.22%                    10.5
0%
                Cluster 2:               36.79%                    36.1
2%
                Cluster 3:               7.44%                     6.36%
                Cluster 4:               36.03%                    38.7
0%
                Cluster 5:               8.52%                     8.31%

Final Value of Loss for both:            0.2980                    0.275
0
```





Increasing K beyond 3 only partitions the large middle cluster in all plots where $K \geq 3$. The cluster centers do not appear to be near any centered bunch of data in plots with $K > 3$. Rather, they appear to be splitting up a bigger cluster.

2 Mixtures of Gaussians [16 pt.]

Mixtures of Gaussians (MoG) can be interpreted as a probabilistic version of K-means clustering. For each data vector, MoG uses a latent variable z to represent the cluster assignment and uses a joint probability model of the cluster assignment variable and the data vector: $P(\mathbf{x}, z) = P(z)P(\mathbf{x} \mid z)$. For N iid training cases, we have $P(\mathbf{X}, \mathbf{z}) = \prod_{n=1}^{N} P(\mathbf{x}_n, z_n)$. The Expectation-Maximization (EM) algorithm is the most commonly used technique to learn a MoG. Like the standard K-means clustering algorithm, the EM algorithm alternates between updating the cluster assignment variables and the cluster parameters. What makes it different is that instead of making hard assignments of data vectors to cluster centers (the "min" operation above), the EM algorithm computes probabilities for different cluster centers, $P(z|\mathbf{x})$. These are computed from $P(z = k|\mathbf{x}) = P(\mathbf{x}, z = k)/\sum_{j=1}^{K} P(\mathbf{x}, z = j)$.

While the Expectation-Maximization (EM) algorithm is typically the go-to learning algorithm to train MoG and is guaranteed to converge to a local optimum, it suffers from slow convergence. In this assignment, we will explore a different learning algorithm that makes use of gradient descent.

2.1 The Gaussian cluster mode [7 pt.]

Each of the K mixture components in the MoG model occurs with probability $\pi^k = P(z = k)$. The data model is a multivariate Gaussian distribution centered at the cluster mean (data center) $\boldsymbol{\mu}^k \in \mathbb{R}^D$. We will consider a MoG model where it is assumed that for the multivariate Gaussian for cluster k, different data dimensions are independent and have the same standard deviation, σ^k.

1. Use the K-means distance function `distanceFunc` implemented in 1.1 to implement `log_GaussPDF` function by computing the log probability density function for cluster k: $\log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^k, {\sigma^k}^2)$ for all pair of N data points and K clusters. Include the snippets of the Python code.

The probability density function for cluster K is a multivariate Gaussian distribution represented by this formula ->

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \sigma^2) = \frac{1}{(2\pi)^{D/2}\,\sigma} \exp\left[-\frac{1}{2\sigma^2}(\mathbf{x}-\boldsymbol{\mu})^\top(\mathbf{x}-\boldsymbol{\mu})\right]$$

In [ ]:
```
def log_GaussPDF(X, MU, sigma):
    # Inputs
    # X: N X D
    # mu: K X D
    # sigma: K X 1

    # Outputs:
    # log Gaussian PDF N X K
    input = -(1/2) * tf.log(2 * np.pi * tf.transpose(sigma)**2)
    pair_dist = distanceFunc_GMM(X, MU)
    calc_pairdi = input - tf.square(pair_dist) / (2 * tf.transpose(sigma
)**2)
    return calc_pairdi
```

2. Write a vectorized Tensorflow Python function that computes the log probability of the cluster variable z given the data vector $\mathbf{x}$: $\log P(z|\mathbf{x})$. The log Gaussian pdf function implemented above should come in handy. The implementation should use the function `reduce_logsumexp()` provided in the helper functions file. Include the snippets of the Python code and comment on why it is important to use the log-sum-exp function instead of using `tf.reduce_sum`.

ANSWER

For numerical stability computational efficiency, the log of the probabilities instead of the actual probabilities themselves. In order to compute the posterior, take a sum of the actual probabilites.

`logsumexp` adds the two numbers in log-domain while ensuring the result is also in log-domain.

In [ ]:
```
def log_posterior(log_PDF, log_pi):
    # Input
    # log_PDF: log Gaussian PDF N X K
    # log_pi: K X 1

    # Outputs
    # log_post: N X K

    log_post = tf.add(log_PDF, tf.transpose(log_pi))
    log_sum = reduce_logsumexp(log_post, keep_dims=True)
    posterior = tf.subtract(log_post, log_sum)
    return posterior
```

2.2 Learning the MoG [9 pt.]

The marginal data likelihood for the MoG model is as follows (here "marginal" refers to summing over the cluster assignment variables):

$$P(\mathbf{X}) = \prod_{n=1}^{N} P(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} P(z_n = k) P(\mathbf{x}_n \mid z_n = k)$$
$$= \prod_{n} \sum_{k} \pi^k \mathcal{N}(\mathbf{x}_n ; \boldsymbol{\mu}^k, {\sigma^k}^2)$$

The loss function we will minimize is the negative log likelihood $\mathcal{L}(\boldsymbol{\mu}, \sigma, \pi) = -\log P(\mathbf{X})$. The maximum likelihood estimate (MLE) is a set of the model parameters $\boldsymbol{\mu}, \sigma, \pi$ that maximize the log likelihood or, equivalently, minimize the negative log likelihood.

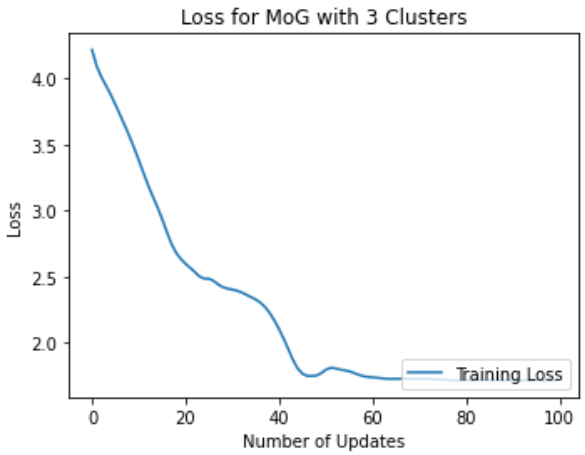
1. Implement the loss function using log-sum-exp function and perform MLE by directly optimizing the log likelihood function using gradient descent.

   Note that the standard deviation has the constraint of $\sigma \in [0, \infty)$. One way to deal with this constraint is to replace σ^2 with $\exp(\phi)$ in the math and the software, where ϕ is an unconstrained parameter. In addition, π has a simplex constraint, that is $\sum_k \pi^k = 1$. We can again replace this constrain with unconstrained parameter ψ through a softmax function $\pi^k = \exp(\psi^k) / \sum_{k'} \exp(\psi^{k'})$. A log-softmax function, `logsoftmax`, is provided for convenience in the helper functions file.

   For the dataset `data2D.npy`, set $K = 3$ and report the best model parameters it has learned. Include a plot of the loss vs the number of updates.

In [ ]:

```
MU:
[[ 0.10350643 -1.52896335]
 [-1.1010236  -3.30296461]
 [ 1.29834846  0.30900253]]


sigma: [0.99328105 0.19659176 0.20020551]


pi: [0.31866376 0.34208264 0.33925359]
```



2. Hold out 1/3 of the data for validation, and for each value of $K = 1, 2, 3, 4, 5$:

   (a) Train a MoG model.

   (b) Include a 2D scatter plot of training data points colored by their cluster assignments.

   (c) Compute and report the percentage of the training data points belonging to each of the K clusters. (include this information in the legend of the scatter plot.)

   (d) Compute and report the loss function over validation data. (include this in the caption of the scatter plot.)

   Explain which value of K is best, based on the validation loss.

Answer

There is similar result in the GMM (Gaussian Mixture Model) plots as in the K-means plots, but it is much more pronounced because the GMM clusters take into account the adjustable prior and are probabilistic rather than forcing each data point to be assigned to the nearest cluster. When compared to the K-means case, this makes for more well-defined clusters.

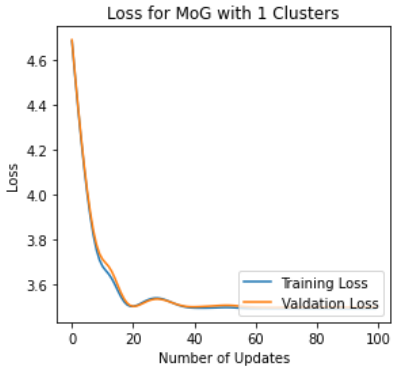
When $K = 2$ is used, we get a more interesting result. The GMM can locate one of the two clusters called with high accuracy. Despite the heavy cluster center's location, it is noticable that some points near the large cluster center are actually assigned to the other cluster! This is not the same as the K-means case. The GMM model appears to better reflect the situation here, as these far points are more likely to be center cluster outliers rather than large cluster points.

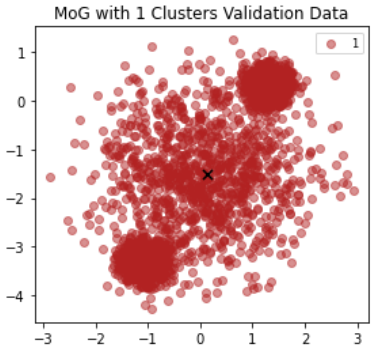
With $K = 3$, The GMM clearly finds to include the most clusters and also identifies the other points to the central cluster, as predicted from the labeling of the data. As increase K beyond 3, new cluster centers being assigned to the central cluster. This appears to be redundant, as both centers appear to refer to the same central cluster. This problem only worsens as K increases. As a result **$K = 3$ appears to be the best for this data.**

In [ ]:

K = 1

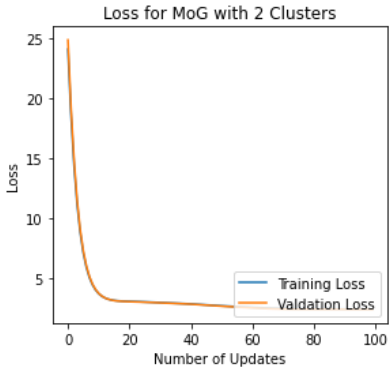
```
                                    Training Loss            Validat
ion Loss
Final Value of Loss for both:        3.4898                   3.495
6
```

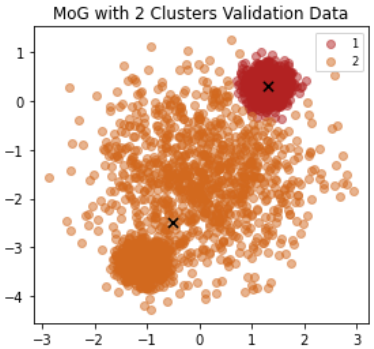




K = 2

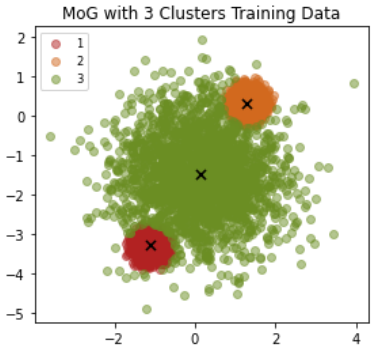
```
                                    Training Loss            Validat
ion Loss
Final Value of Loss for both:        2.4248                   2.396
3
```

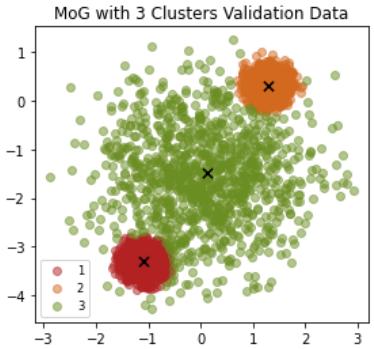




K = 3

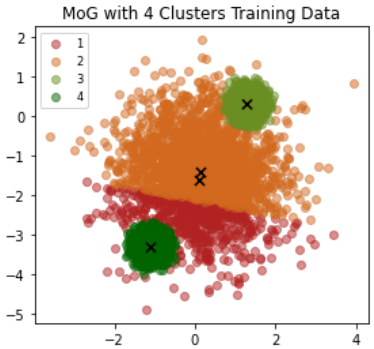
```
                                    Training Loss            Validat
ion Loss
Final Value of Loss for both:        1.7324                   1.699
9
```

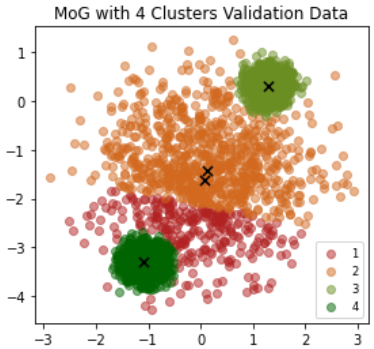




K = 4

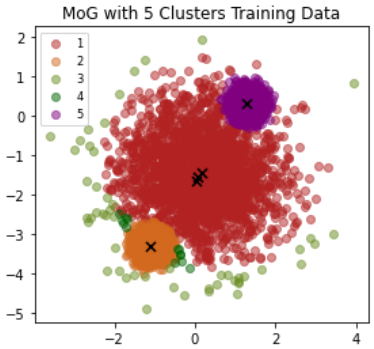
	Training Loss	Validation Loss
Final Value of Loss for both:	1.7262	1.6900

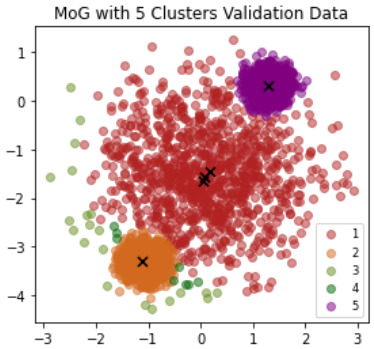




K = 5

	Training Loss	Validation Loss
Final Value of Loss for both:	1.7266	1.6917





3. Run both the K-means and the MoG learning algorithms on `data100D.npy` for $K = \{5, 10, 15, 20, 30\}$ (Hold out 1/3 of the data for validation). Comment on how many clusters you think are within the dataset by looking at the MoG validation loss and K-means validation loss. Compare the learnt results of K-means and MoG.

Answer

K Means

We can see from the graphs below that the results for K-means follow a similar pattern regardless of the number of clusters. However, as the number of clusters increases, the final loss values continue to decrease (or staying approximately the same). This makes sense in the context of the K-means algorithm. Allowing the algorithm to find more clusters would allow for more precise partitioning of the input space. Because there are more cluster centers, we expect the average distance from each data point to its respective cluster center to decrease. Data points are simply assigned to the cluster whose center is nearest to them. This is exactly what we see in the K-means graphs.

Mixture of Gaussians (MoG)

As K increase in the MoG (Mixture of Gaussian) models, different results. it is We discovered from the 2D data set that as K increased significantly beyond the desired number of clusters, the MoG model began adding additional cluster centers that were roughly in the same location as one of the other cluster centers. The redundant clusters appear to be similar to the clusters that they mirror but have a considerably lower πk(prior probability). Increasing K should therefore not see significantly better loss values as only very few points are allocated to redundant additional clusters and the probabilties overall are low.

If K is too low, the loss is expected to be considerably higher than optimal. We found that the MG model was discovered and converged on a dense cluster, but could not converge on the second dense cluster, when we set K=2, because there weren't enough clusters. Accordingly, the remaining cluster was labeled with all points outside the identified cluster.

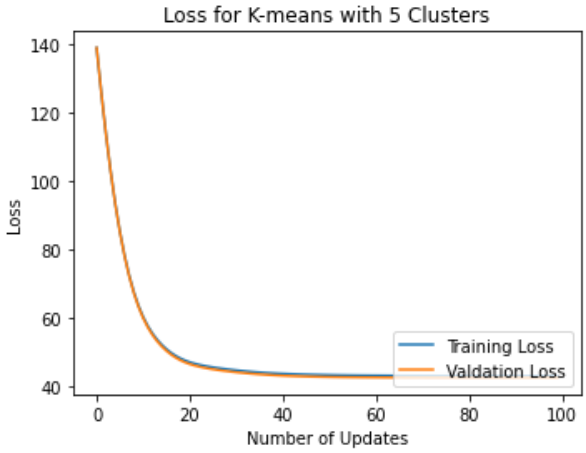
These expectations correspond to the MoG plots. The loss values are just over 100 with K=5. The K=5 case is not sufficient for the dataset clusters. For K=10, a similar occurs. The K=15 and K=20 and K=30 plots, however, all converge with the loss values approximately 49. We notice that K over 15 did not lead to a substantial loss reduction. This is consistent with those cases where K is too high and it can be concluded that K beyond 15 has some redundant clusters. From this, **it is probably about 15 clusters** for the 100D dataset.

Comparison

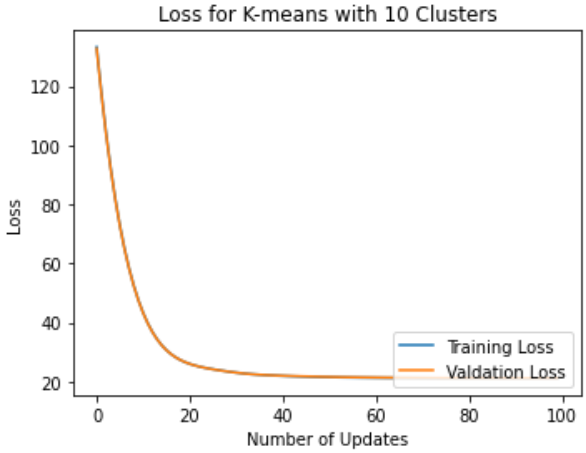
K-means seems to perform better for small K because the probabilities of dividing the data into clusters are not limited to gaussian. However, it is noted that for large K the MoG model works better.

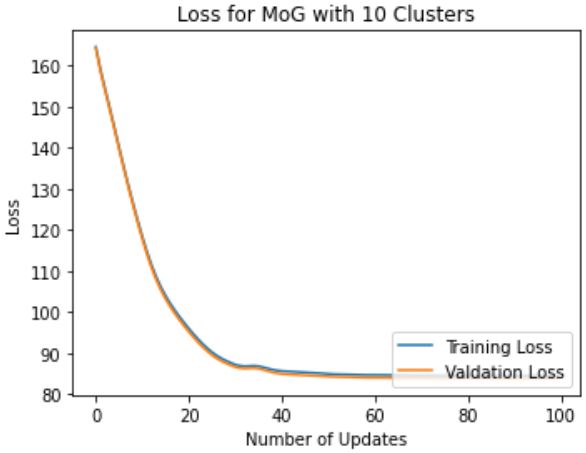
In [ ]:

```
K = 5
                                        Training              Validat
ion
                K-Means Final Loss:      43.2816                42.72
17
                MoG Final Loss:          93.7071                94.85
80
```

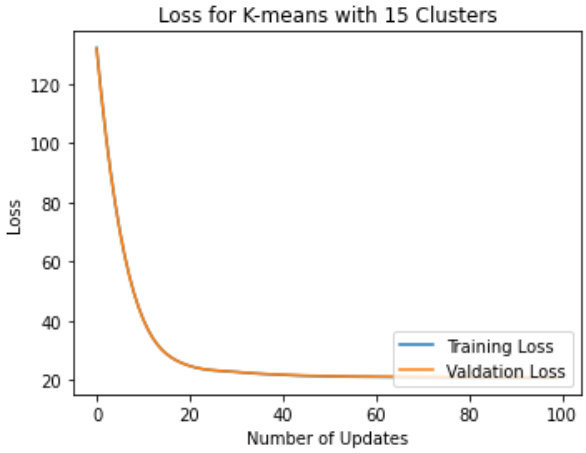


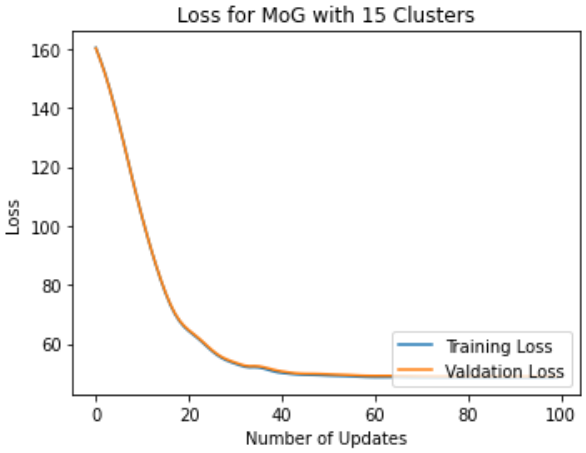

```
K = 10
                                        Training              Validat
ion
                K-Means Final Loss:      21.1043                21.19
31
                MoG Final Loss:          84.4998                83.84
77
```





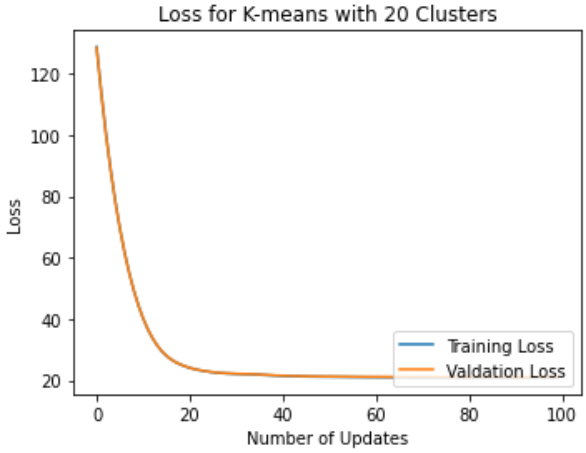

```
K = 15
                                        Training              Validat
ion
                K-Means Final Loss:      20.8225                20.99
63
                MoG Final Loss:          48.5198                48.98
39
```





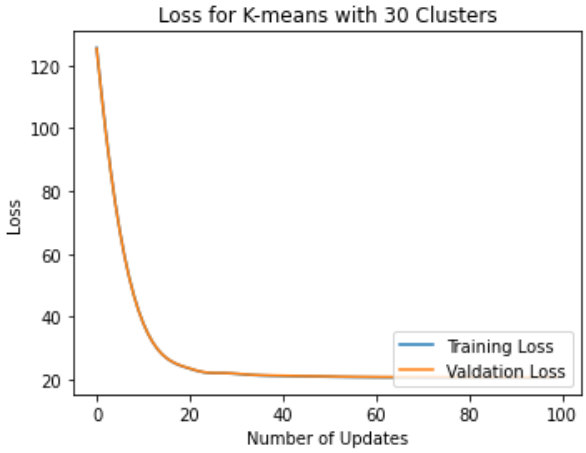
K = 20

	Training	Validation
K-Means Final Loss:	20.8964	21.0595
MoG Final Loss:	48.3934	48.8766



K = 30

	Training	Validation
K-Means Final Loss:	20.5768	20.7558
MoG Final Loss:	47.9328	48.5799


Loss for K-means with 30 Clusters
Loss
Number of Updates
Training Loss
Valdation Loss


Loss for MoG with 30 Clusters
Loss
Number of Updates
Training Loss
Valdation Loss