ECE421 Introduction of Machine Learning

Assignment 3: Unsupervised Learning and Probabilistic Models

Zetong Zhao

1 K-means

1.1 Learning K-means

1. Loss function is squared pairwise distance for all pair of N data points and K clusters. For a K x D parameter matrix $\boldsymbol{\mu}$, where the k^{th} row of the matrix denotes the k^{th} cluster center $\boldsymbol{\mu}_k$, the loss function is the following:

$$\mathcal{L}(\boldsymbol{\mu}) = \sum_{n=1}^{N} \min_{k=1}^{K} ||\mathbf{x}_n - \boldsymbol{\mu}_k||_2^2$$

```
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)

        X = tf.expand_dims(X, 1) #shape becomes (N, 1, D)
        square = tf.square((X-MU))
        pair_dist = tf.reduce_sum(square, 2) #change shape to (N, K)
        return pair_dist
```

Figure 1: squared pairwise distance TensorFlow implementation

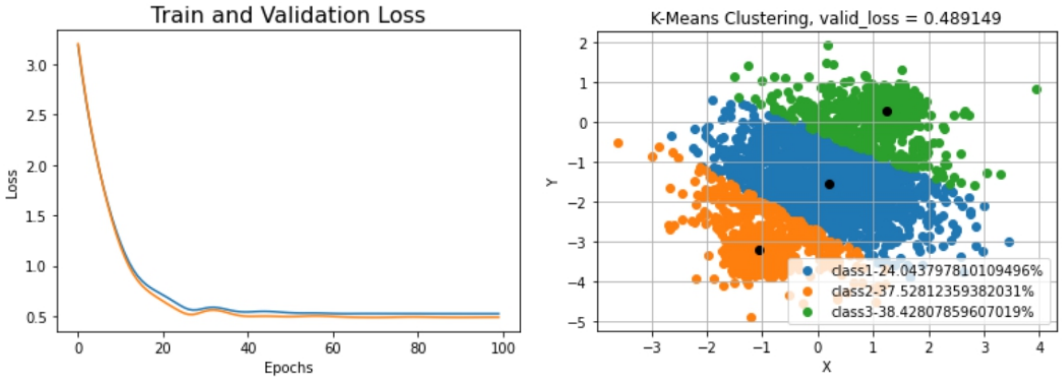

Figure 2: loss(left) with final validation loss=0.489 and clustering(right) when K=3

2.

	Final training loss	Final validation loss
K=1	3.838	3.862
K=2	0.9367	0.888
K=3	0.5234	0.4884
K=4	0.3481	0.3165
K=5	0.294	0.2666

Table 1: Training and validation loss

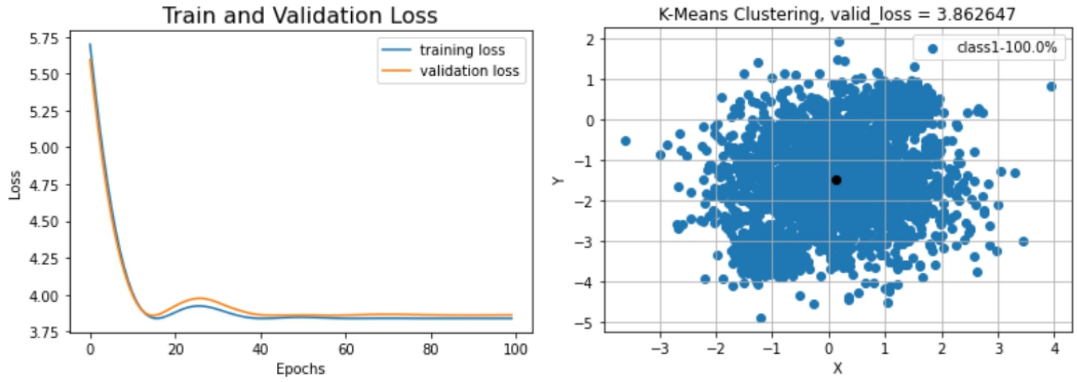

Figure 3: K-means loss(left) with final validation loss=3.86 and clustering(right) when K=1

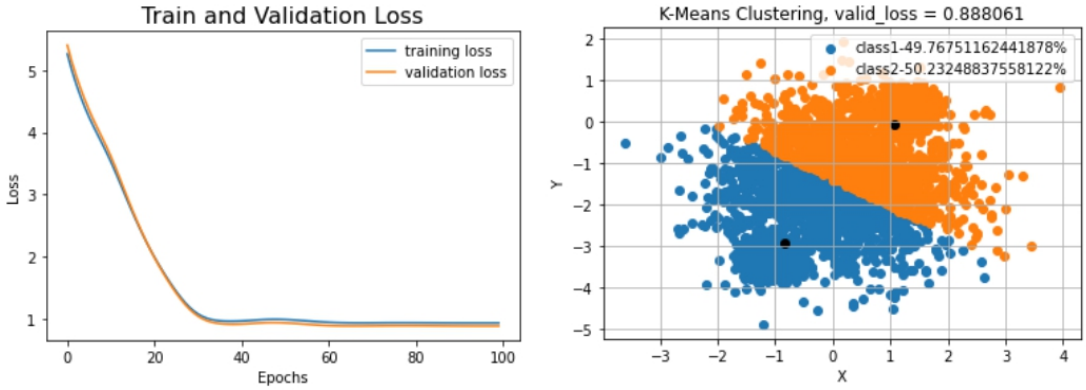

Figure 4: K-means loss(left) with final validation loss=0.888 and clustering(right) when K=2

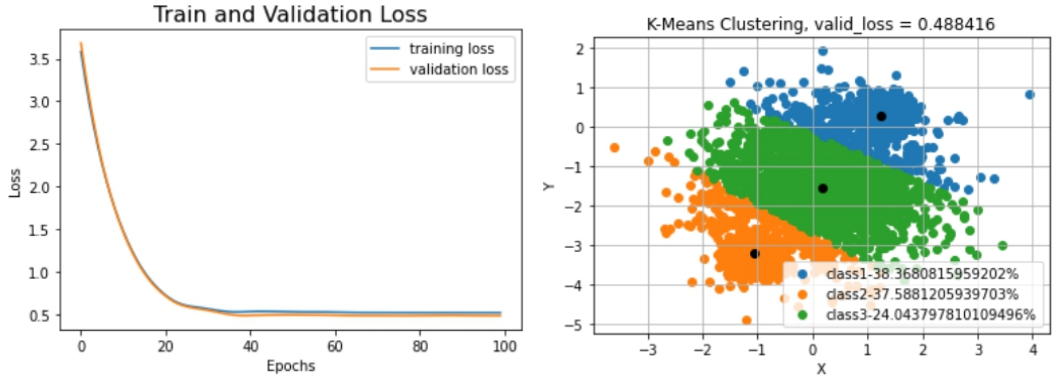

Figure 5: K-means loss(left) with final validation loss=0.488 and clustering(right) when K=3

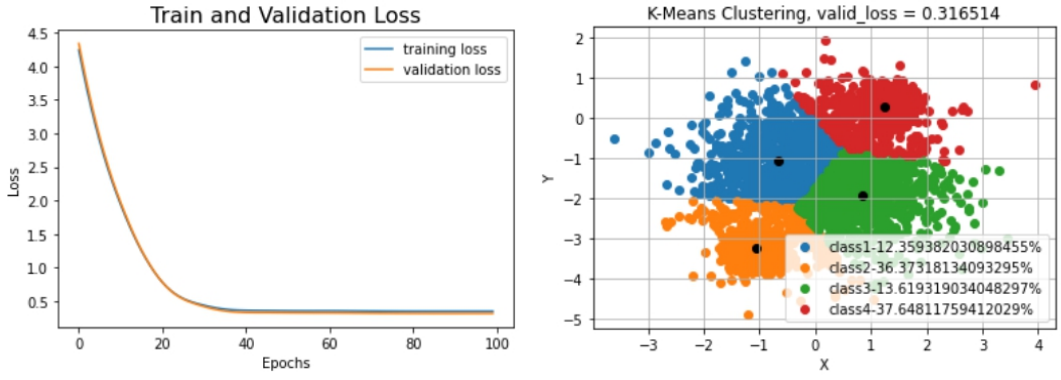

Figure 6: K-means loss(left) with final validation loss=0.317 and clustering(right) when K=4

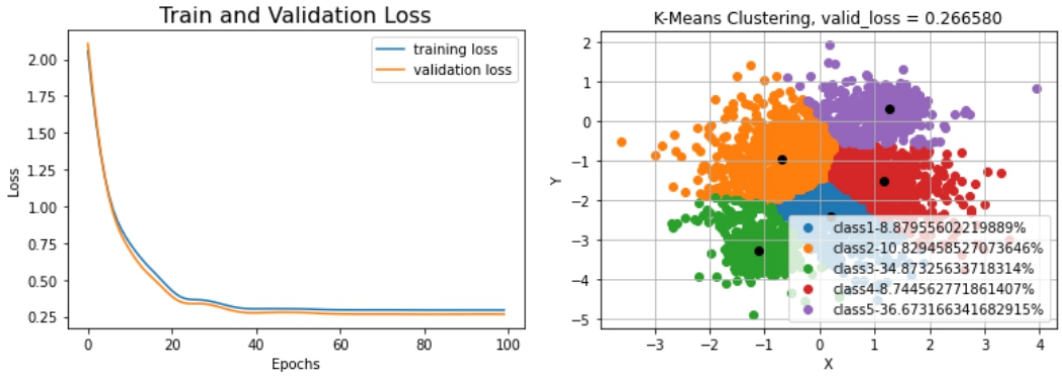

Figure 7: K-means loss(left) with final validation loss=0.267 and clustering(right) when K=5

From the figure K=1 above, we can see that one cluster is dense around the center and two small dense clusters on the left downside and right upside. Also, according to the figures, the two small dense clusters are always divided separately. When K is greater than 3, the clustering only separates the center cluster into several parts, which confirms that the small clusters should be two independent clusters. Since the larger cluster is dense around the center, it should not further divide. Thus, the three clusters(K=3) is more reasonable.

```
def K_means(K, D, epochs, trainset, validset, plot):
        X = tf.placeholder(tf.float64, shape=(None, D))
        MU = tf.Variable(tf.truncated_normal((K, D), mean=0,
                         stddev=1, dtype=tf.float64), trainable=True)
        loss = tf.reduce_sum(tf.reduce_min(distanceFunc(X, MU), axis=1))
        optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9,
                                           beta2=0.99, epsilon=1e-5).minimize(loss)

        train_loss = []
        valid_loss = []
        with tf.Session() as session:
                session.run(tf.global_variables_initializer())
                for i in range(epochs):
                        session.run([MU, optimizer], feed_dict={X: trainset})
                        loss_train = session.run(loss, feed_dict={X: trainset})
                        loss_valid = session.run(loss, feed_dict={X: validset})
                        train_loss.append(loss_train/trainset.shape[0])
                        valid_loss.append(loss_valid/validset.shape[0])
                        if (i==epochs-1):
                                print("final train_loss is:", loss_train/trainset.shape[0])
                                print("final valid_loss is:", loss_valid/validset.shape[0])
```

Figure 8(a): training code for k-means

```
        if (plot):
            plt.plot(range(epochs),train_loss,label="training loss")
            plt.legend(loc='best')
            plt.plot(range(epochs),valid_loss,label="validation loss")
            plt.legend(loc='best')
            plt.ylabel('Loss')
            plt.xlabel('Epochs')
            plt.title('Train and Validation Loss', fontsize=16)
            plt.show()
            best_MU = session.run(MU, feed_dict={X: trainset})
            clustering = session.run(tf.argmin(distanceFunc(X, best_MU), 1),
                                     feed_dict={X: trainset})
            legend = []
            color = ['c', 'b', 'g', 'r', 'm', 'y', 'k', 'w']
            for i in range(K):
                class_i=[]
                for j in range(len(clustering)):
                    if (clustering[j]==i):
                        class_i.append(trainset[j,:])
                class_i = np.array(class_i)
                plt.scatter(class_i[:, 0], class_i[:, 1], cmap='Pastel')
                legend.append("class"+str(i+1)+"-"
                              +str(100*np.sum(i==clustering)/len(clustering))+"%")
            plt.legend(legend)
            plt.scatter(best_MU[:, 0], best_MU[:, 1], c='black')
            plt.title("K-Means Clustering, valid_loss = %f" %(valid_loss[epochs-1]))
            plt.xlabel('X')
            plt.ylabel('Y')
            plt.grid()
            plt.show()
    return
```

Figure 8(b): training code for k-means

2 Mixtures of Gaussian

2.1 The Gaussian cluster mode

1. Computing the log probability density function:

$$\mathcal{N}(\mathbf{x}\,;\,\boldsymbol{\mu}^k,\sigma^{k2}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{\mathbf{x}-\boldsymbol{\mu}^k}{\sigma^k})^2}$$

$$\log\mathcal{N}(\mathbf{x}\,;\,\boldsymbol{\mu}^k,\sigma^{k2}) = \log(\frac{1}{\sigma^k\sqrt{2\pi}}) - \frac{1}{2}(\frac{\mathbf{x}-\boldsymbol{\mu}^k}{\sigma^k})^2$$

```
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
    pair_dist = distanceFunc(X,mu)
    sigma = tf.squeeze(sigma)
    log_1 = -0.5*tf.log((2 * m.pi*sigma))
    log_2 = -0.5*distanceFunc(X,mu)/sigma
    return log_1+log_2
```

Figure 9: Gaussian PDF coding

2. The function computes the log probability of the cluster variable z given the data vector **x**: log P(z|**x**)

```
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))
        return log_post-reduce_logsumexp(log_post, keep_dims=True)
```

Figure 10: log possibility coding

Reason for using reduce_logsumexp():

Here we are calculating logP(z|**x**)P(z=k) = logP(z|**x**)+logP(z=k). We need to calculate the the exponential function and take the summation of logarithm. The function is able to compute the exponential terms first and perform the log summation, while the reduce_sum() function cannot perform the the summation of logarithm.

2.2 **Learning the MoG**

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

$\pi^k = \exp(\varphi^k) / \sum_{k'} \exp(k')$

$\mathcal{L}(\boldsymbol{\mu}; \sigma, \pi) = -\log P(\mathbf{X}) = -\log \prod_n \sum_k \pi^k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^k, \sigma^{k^2}) = -\sum_n \log \sum_k \pi^k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^k, \sigma^{k^2})$

$$= -\sum_n \log \sum_k \pi^k \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}^k, \sigma^{k^2}) = -\sum_n \log \sum_k \frac{\exp(\varphi^k * -\frac{1}{2}(\frac{\mathbf{x}-\boldsymbol{\mu}^k}{\sigma^k})^2)}{\sum_{k'} \exp(k')}$$

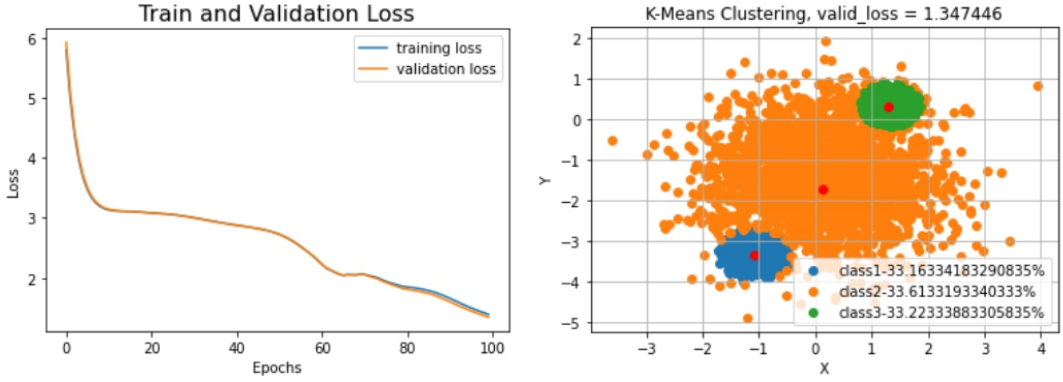

Figure 11: MoG loss(left) with final validation loss=1.35 and clustering(right) when K=3

2. Results from MoG training

	Final training loss	Final validation loss
K=1	2.883	2.879
K=2	2.8768	2.8253
K=3	1.855	1.8183
K=4	1.3117	1.284
K=5	1.3034	1.2634

Table 2: training and validation loss of MoG

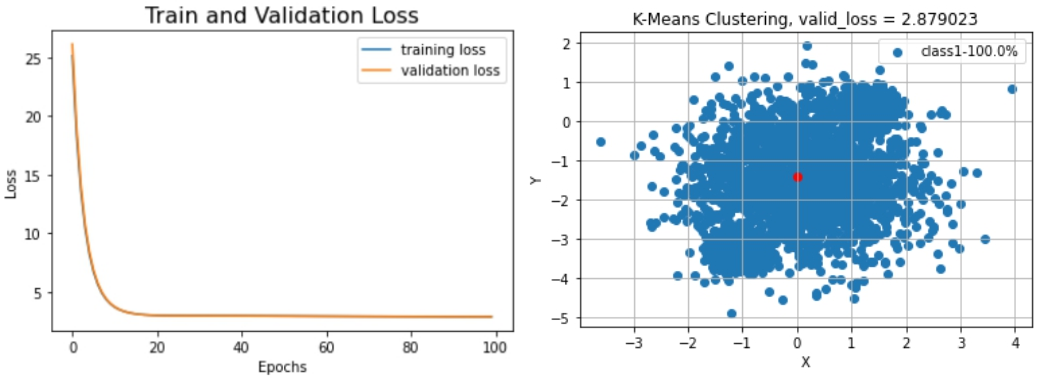

Figure 12: MoG loss(left) with final validation loss=2.88 and clustering(right) when K=1

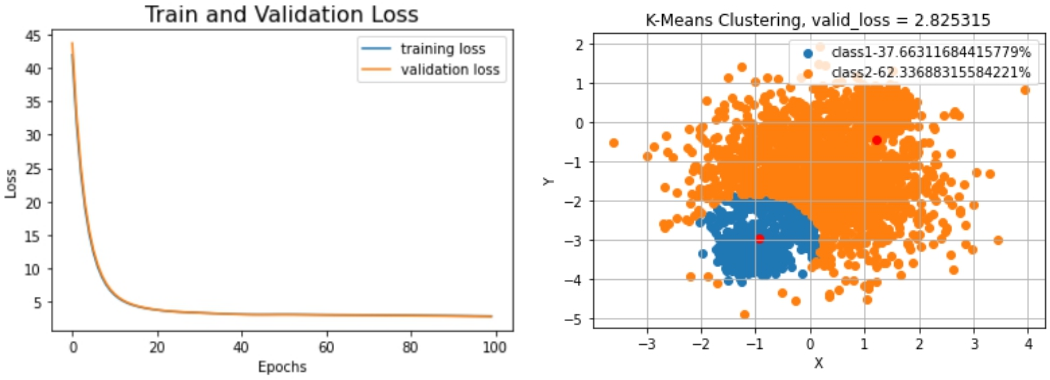

Figure 13: MoG loss(left) with final validation loss=2.83 and clustering(right) when K=2

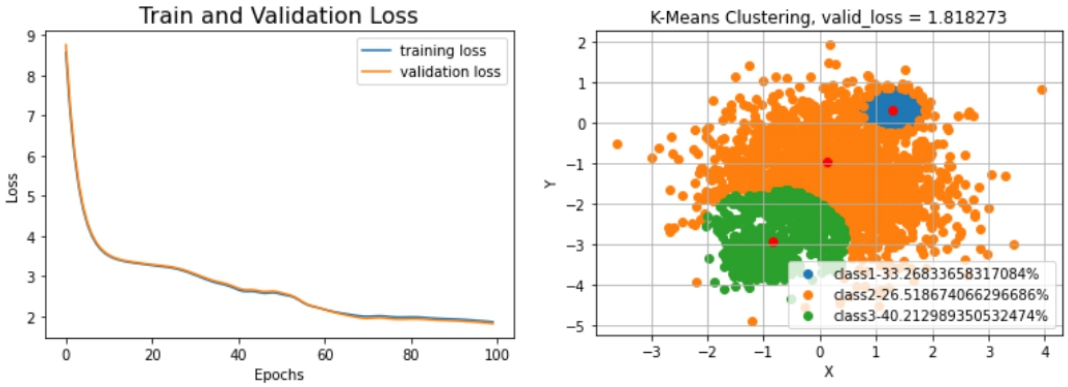

Figure 14: MoG loss(left) with final validation loss=1.82 and clustering(right) when K=3

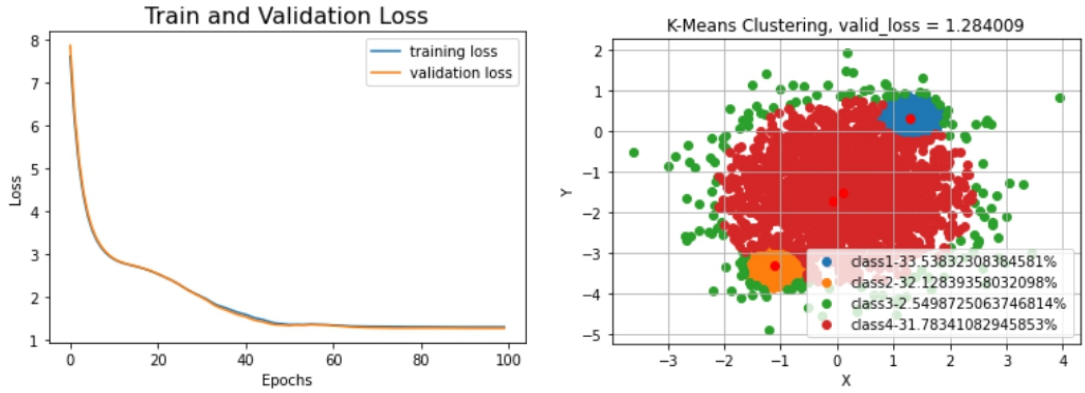

Figure 15: MoG loss(left) with final validation loss=1.284 and clustering(right) when K=4

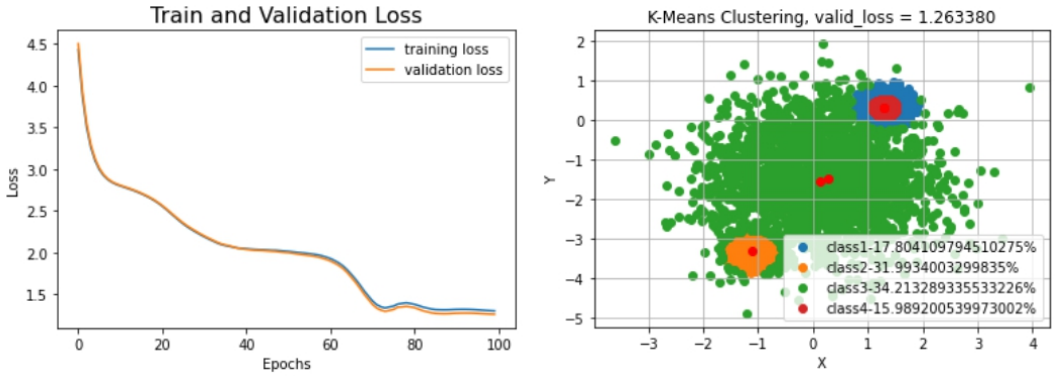

Figure 16: MoG loss(left) with final validation loss=1.263 and clustering(right) when K=5

MoG is predicting varying from K-means. The algorithm is predicting based on the guassians of the clusters, so the points on the edge may be parts of the center cluster and high density points may be another guassian. From the table above, we can see that the losses are similar among 3-5 clusters. When K=1-3, the GMM gives the reasonable clusters, when K is higher than 3, there are dense clusters separated or surrounded by less dense clusters with similar center. For example, when K=4, the green dots are sporadic around the red parts, which could be in the same cluster since they have the very close centers. Therefore, K=3 would be a reasonable number of clusters.

```
def GMM(K,D,epochs,trainset,validset,plot):
        #define variables
        X = tf.placeholder(tf.float32, shape=(None,D))
        mu = tf.Variable(tf.random_normal([K, D], stddev = 1))
        sigma = tf.Variable(tf.random_normal([K, 1], stddev = 1))
        pi = tf.Variable(tf.random_normal([K, 1], stddev = 1))
        sigma = tf.exp(sigma)
        log_pdf = log_GaussPDF(X,mu,sigma)
        log_pi = logsoftmax(pi)


        #defien loss & optimizer
        #loss = tf.reduce_sum(log_posterior(log_pdf,log_pi))
        loss= - tf.reduce_sum(reduce_logsumexp(log_pdf + tf.squeeze(log_pi), keep_dims=True))
        optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9,
                                           beta2=0.99, epsilon=1e-5).minimize(loss)
        clustering = tf.argmax(log_posterior(log_pdf,log_pi),1)
        train_loss = []
        valid_loss = []
        with tf.Session() as session:
                session.run(tf.global_variables_initializer())
                for i in range(epochs):
                        session.run([optimizer, mu, clustering, sigma, log_pi], feed_dict={X: trainset})
                        loss_train = session.run(loss, feed_dict={X: trainset})
                        loss_valid = session.run(loss, feed_dict={X: validset})
                        train_loss.append(loss_train/trainset.shape[0])
                        valid_loss.append(loss_valid/validset.shape[0])
                        if (i==epochs-1):
                                print("final train_loss is:", loss_train/trainset.shape[0])
                                print("final valid_loss is:", loss_valid/validset.shape[0])
                if (plot):
                        plt.plot(range(epochs),train_loss,label="training loss")
                        plt.plot(range(epochs),valid_loss,label="validation loss")
                        plt.legend(loc='best')
                        plt.ylabel('Loss')
                        plt.xlabel('Epochs')
                        plt.title('Train and Validation Loss', fontsize=16)
                        plt.show()
```

Figure 17(a): training code for MoG

```
                clustering,mu = session.run([tf.argmax(log_posterior(log_pdf,log_pi),1),mu]
                                                        ,feed_dict={X:trainset})
                legend = []
                for i in range(K):
                        #print("class",i,"percentage is", np.sum(i==clustering)/len(clustering))
                        class_i=[]
                        for j in range(len(clustering)):
                                if (clustering[j]==i):
                                        class_i.append(trainset[j,:])
                        class_i = np.array(class_i)
                        if (len(class_i)!=0):
                                plt.scatter(class_i[:, 0], class_i[:, 1])
                                legend.append("class"+str(i+1)+"-"
                                +str(100*np.sum(i==clustering)/len(clustering))+"%")
                plt.legend(legend)

                plt.scatter(mu[:, 0], mu[:, 1], c='red')
                plt.title("K-Means Clustering, valid_loss = %f" %(valid_loss[epochs-1]))
                plt.xlabel('X')
                plt.ylabel('Y')
                plt.grid()
                plt.show()
        return
```

Figure 17(b): training code for MoG

3. Compare results of the K-means and MoG

	Final training loss K-means	Final validation loss K-means	Final training loss MoG	Final validation loss MoG
K=5	36.885	37.161	47.471	47.065
K=10	37.066	36.737	14.007	14.032
K=15	35.404	35.065	6.266	6.258
K=20	37.016	36.969	7.088	7.072
K=25	21.528	21.542	9.602	6.564
K=30	21.528	21.544	7.261	7.221

Table 3: training and validation loss of MoG and K-means

K-means: The algorithm divides the points into separate groups. With larger number of clustering, K-means will separate the original groups into more groups. The loss is expected to decrease, since the distances between points and centers are smaller. The data from the table above is the same as expected but is a very small decreasing trend for K smaller than 25. The sudden decrease at K=25 may indicate that the algorithm finds a much better way to distribute the data.

MoG: As the results from 2D points, the MoG loss will not have a clear change after a certain number of clusters. If the number of clusters is too small, the cluster cannot find the suitable cluster, e.g. when K=2 in 2D plot, the loss will be higher than others. After the best number of clusters, the center of the more clusters will be close to each other. From the results in the table, we can see that the loss stays stable since K=15. Thus K=15 would be a suitable value for clustering.

Compare the results: K-means and MoG provide different result for number of clusters for the same groups of data. Also, the loss for MoG is much smaller than K-means for the most possible number of clusters (K=25 for K-means, and K=15 for MoG). This means when the data is not distributed evenly and there are several dense clusters, the normal distribution is a better estimation of the data. So the points can fit better with MoG. Above all, K=15 is a more possible answer for 100D data points.