1 K-means

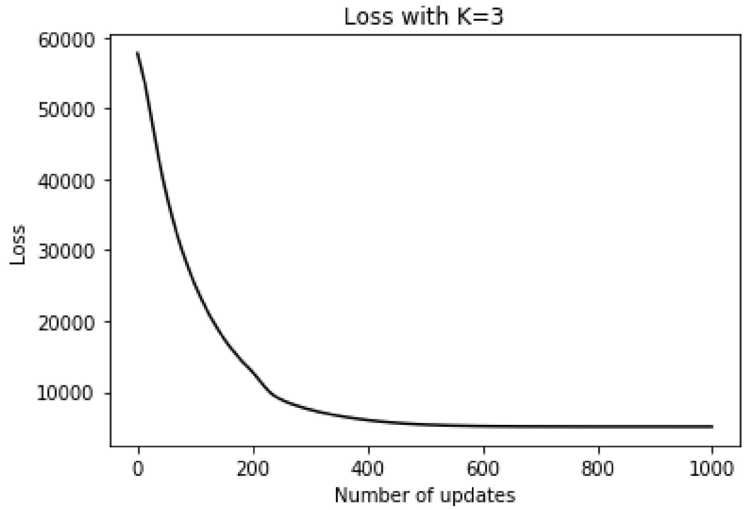
1.1 Learning K-means

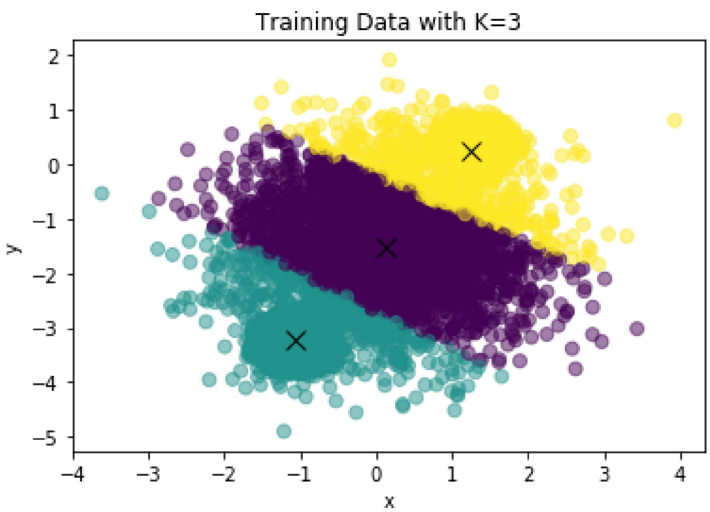
1.1.1 Helper functions and graphs

```
# Distance function for K-means
def distanceFunc(X, MU):
    X = tf.expand_dims(X, 1)
    MU = tf.expand_dims(MU, 0)
    pair_distance = tf.reduce_sum(tf.square(tf.subtract(X, MU)), 2)
    return pair_distance

class KMeans(object):
    def __init__(self, K, D):
        self.X = tf.placeholder(shape=[None, D], dtype=tf.float64)
        self.mu =
tf.Variable(tf.random_normal(shape=[K,D],dtype=tf.float64),
trainable=True, dtype=tf.float64)
        pair_distance = distanceFunc(self.X, self.mu)
        self.classes = tf.argmin(pair_distance, 1)
        self.loss = tf.reduce_sum(tf.reduce_min(pair_distance,1))
        self.optimizer = tf.train.AdamOptimizer(learning_rate=0.01)
        self.train_op = self.optimizer.minimize(self.loss)
        self.sess = tf.Session()
        self.sess.run(tf.global_variables_initializer())
    def train(self, data):
        feed_dict = {self.X : data}
        loss, _ = self.sess.run([self.loss, self.train_op], feed_dict)
        return loss
    def evaluate(self, data):
        feed_dict = {self.X : data}
        feed_dict = {self.X : data}
        loss, classes = self.sess.run([self.loss, self.classes],
feed_dict)
        return loss, classes
    def get_final_params(self, K):
        print("Number of Clusters: {}".format(K))
        mean = self.sess.run([self.mu])
        print("mu's: {}".format(mean[0]))
```

```python
        return mean

K = 3
model = KMeans(K, dim)
loss = []

for i in range(1000):
    loss.append(model.train(data))
mean = model.get_final_params(K)
plt.plot(np.arange(len(loss)), loss, 'k')
plt.xlabel("Number of updates")
plt.ylabel("Loss")
plt.title("Loss with K={}".format(K))
plt.show()
plt.clf()
train_loss, train_classes = model.evaluate(data)
plt.scatter(data[:, 0], data[:, 1], c=train_classes, s=50, alpha=0.5)
plt.plot(mean[0][:, 0], mean[0][:, 1], 'kx', markersize=10)
plt.title("Training Data with K={}".format(K))
plt.xlabel("x")
plt.ylabel("y")
plt.show()
plt.clf()
```

```
Number of Clusters: 3
mu's: [[ 0.13560835 -1.5225483 ]
 [-1.05657993 -3.23656808]
 [ 1.24929798  0.25105767]]
```






```
<matplotlib.figure.Figure at 0x189371e5588>
```

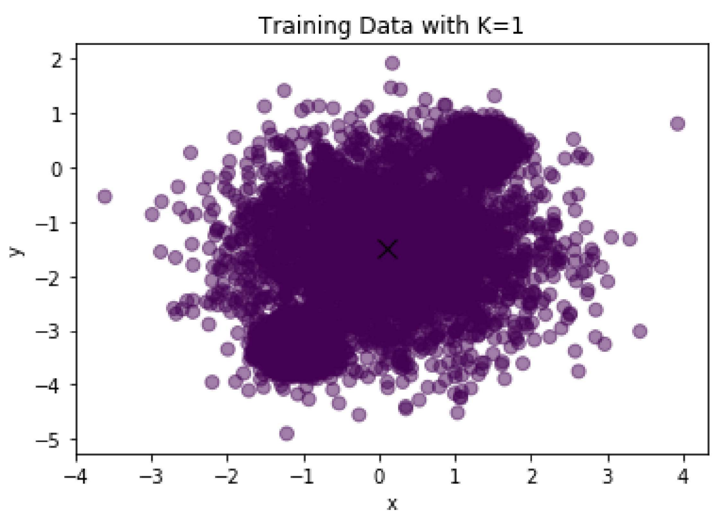
1.1.2 Graphs of different K values

```
for K in [1,2,3,4,5]:
    model = KMeans(K, dim)
    loss = []
    for i in range(4000):
        loss.append(model.train(data))
    mean = model.get_final_params(K)
    train_loss, train_classes = model.evaluate(data)
    percents = [0 for k in range(K)]
    for i in range(K):
        for c in train_classes:
```

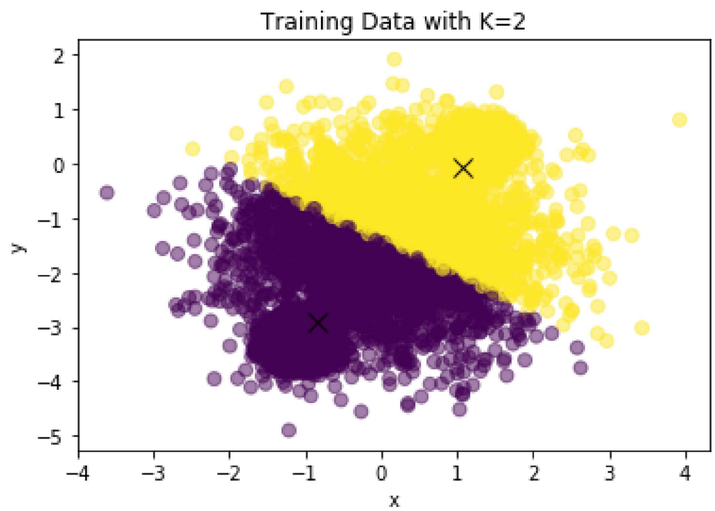
```
            if c == i:
                percents[i] += 1
    for idx, ele in enumerate(percents):
        print("Percent of data points belonging to cluster {}:
{}".format(idx+1,
ele/len(train_classes)))
    plt.scatter(data[:, 0], data[:, 1], c=train_classes, s=50,
alpha=0.5)
    plt.plot(mean[0][:, 0], mean[0][:, 1], 'kx', markersize=10)
    plt.title("Training Data with K={}".format(K))
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```

```
0 -3.27241730e+00]
 [ 1.25785989e+00  2.662
```

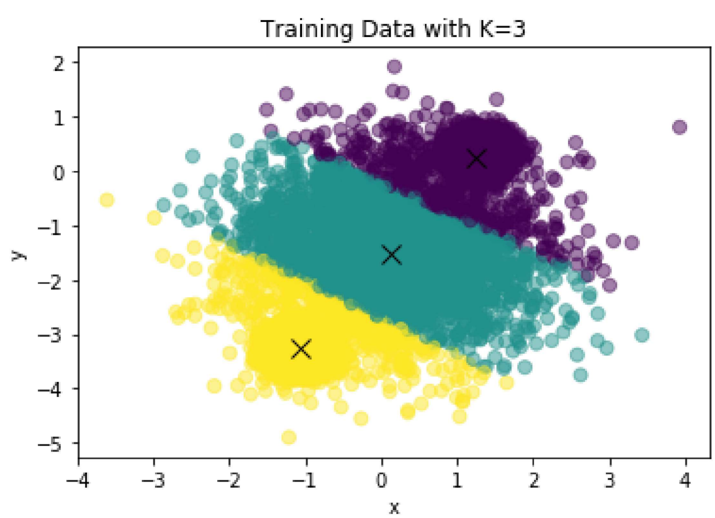
```
Number of Clusters: 1
mu's: [[ 0.10131727 -1.50542365]]
Percent of data points belonging to cluster 1: 1.0
```




```
Number of Clusters: 2
mu's: [[-0.83906251 -2.91458165]
 [ 1.06177171 -0.07067045]]
Percent of data points belonging to cluster 1: 0.5045
Percent of data points belonging to cluster 2: 0.4955
```

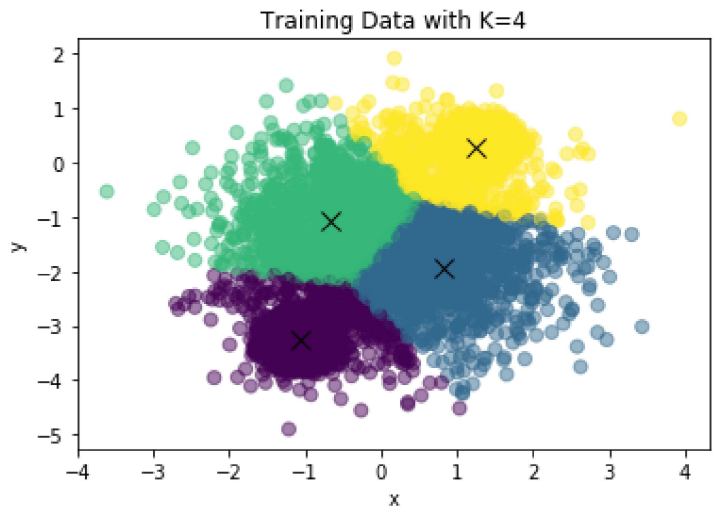



```
Number of Clusters: 3
mu's: [[ 1.25175298  0.24656856]
 [ 0.12183346 -1.52304193]
 [-1.05592679 -3.24319791]]
Percent of data points belonging to cluster 1: 0.3806
Percent of data points belonging to cluster 2: 0.2381
Percent of data points belonging to cluster 3: 0.3813
```

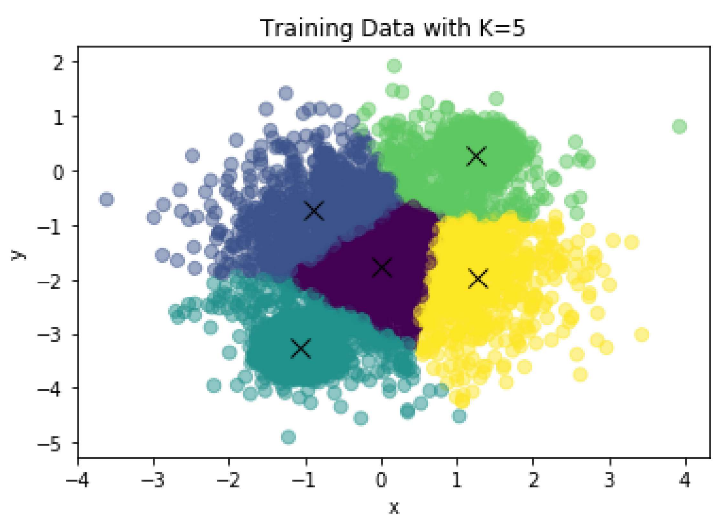



```
Number of Clusters: 4
mu's: [[-1.06420956 -3.26378059]
 [ 0.81574076 -1.95066826]
 [-0.67120585 -1.06043018]
 [ 1.25783646  0.26218621]]
Percent of data points belonging to cluster 1: 0.3713
Percent of data points belonging to cluster 2: 0.1349
```

```
Percent of data points belonging to cluster 3: 0.1209
Percent of data points belonging to cluster 4: 0.3729
```




```
Number of Clusters: 5
mu's: [[ 4.05478673e-04 -1.76772621e+00]
 [-8.81650533e-01 -7.38337943e-01]
 [-1.07243744e+00 -3.27241730e+00]
 [ 1.25785989e+00  2.66202358e-01]
 [ 1.27555853e+00 -1.97204033e+00]]
Percent of data points belonging to cluster 1: 0.111
Percent of data points belonging to cluster 2: 0.0751
Percent of data points belonging to cluster 3: 0.3674
Percent of data points belonging to cluster 4: 0.3704
Percent of data points belonging to cluster 5: 0.0761
```



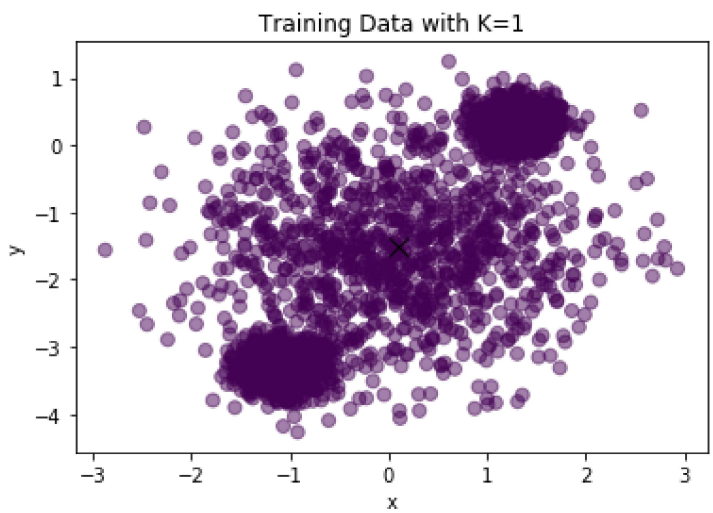
Five clusters is the best number of clusters as it leads to the best training.

1.1.3 Graphs of different K values with ⅓ of the data out

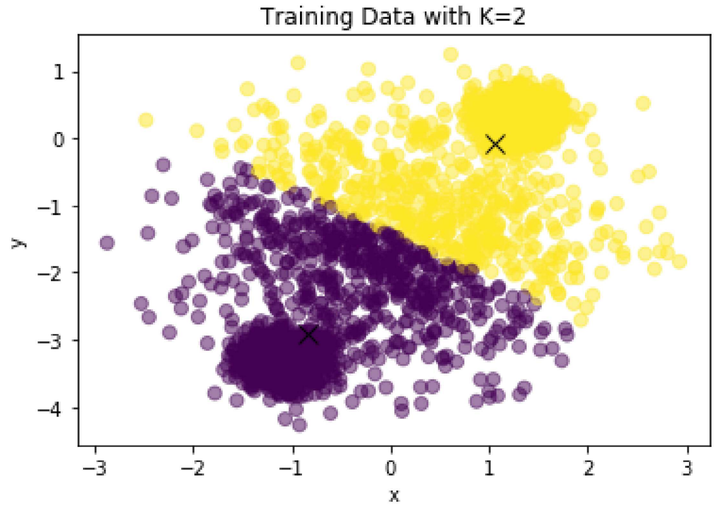
```
# Loading data
data = np.load('data2D.npy')
#data = np.load('data100D.npy')
[num_pts, dim] = np.shape(data)

is_valid = True

# For Validation set
if is_valid:
    valid_batch = int(num_pts / 3.0)
    np.random.seed(45689)
    rnd_idx = np.arange(num_pts)
    np.random.shuffle(rnd_idx)
    val_data = data[rnd_idx[:valid_batch]]

for K in [1,2,3,4,5]:
    model = KMeans(K, dim)
    loss = []
    for i in range(4000):
        loss.append(model.train(data))
    mean = model.get_final_params(K)
    val_loss, val_classes = model.evaluate(val_data)
    print("Validation Loss for cluster {}: {}".format(K, val_loss))
    percents = [0 for k in range(K)]
    for i in range(K):
        for c in val_classes:
            if c == i:
                percents[i] += 1
    for idx, ele in enumerate(percents):
        print("Percent of data points belonging to cluster {}:
{}".format(idx+1,
ele/len(val_classes)))
    plt.scatter(val_data[:, 0], val_data[:, 1], c=val_classes, s=50,
alpha=0.5)
    plt.plot(mean[0][:, 0], mean[0][:, 1], 'kx', markersize=10)
```

```
    plt.title("Training Data with K={}".format(K))
    plt.xlabel("x")
    plt.ylabel("y")
    plt.show()
```

```
Number of Clusters: 1
mu's: [[ 0.10280085 -1.50544773]]
Validation Loss for cluster 1: 12860.741347595147
Percent of data points belonging to cluster 1: 1.0
```




```
Number of Clusters: 2
mu's: [[-0.83906251 -2.91458165]
 [ 1.06177171 -0.07067045]]
Validation Loss for cluster 2: 2959.574286571295
Percent of data points belonging to cluster 1: 0.5178517851785178
Percent of data points belonging to cluster 2: 0.4821482148214821
```



Number of Clusters: 3

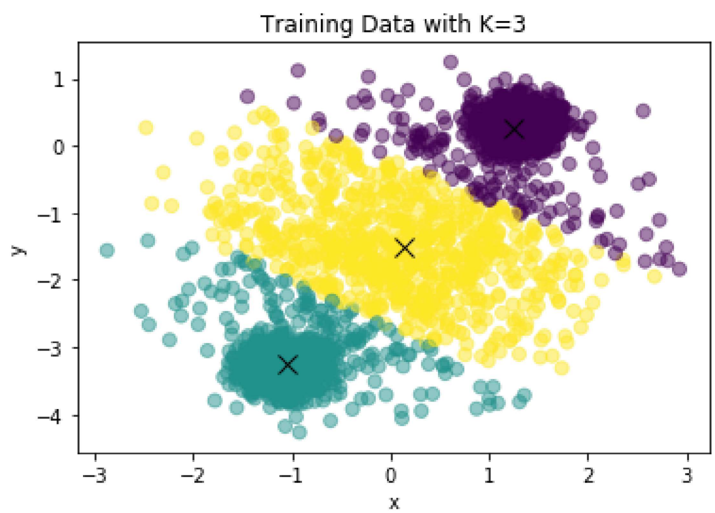
mu's: [[1.2492981   0.25105821]
 [-1.05655711 -3.24010093]
 [0.13498249 -1.52270314]]

Validation Loss for cluster 3: 1618.019318568063

Percent of data points belonging to cluster 1: 0.3726372637263726

Percent of data points belonging to cluster 2: 0.39783978397839787

Percent of data points belonging to cluster 3: 0.2295229522952295

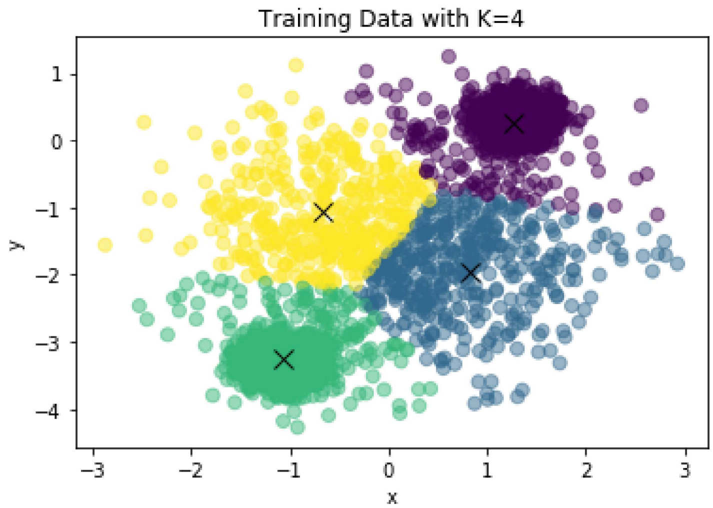


Number of Clusters: 4

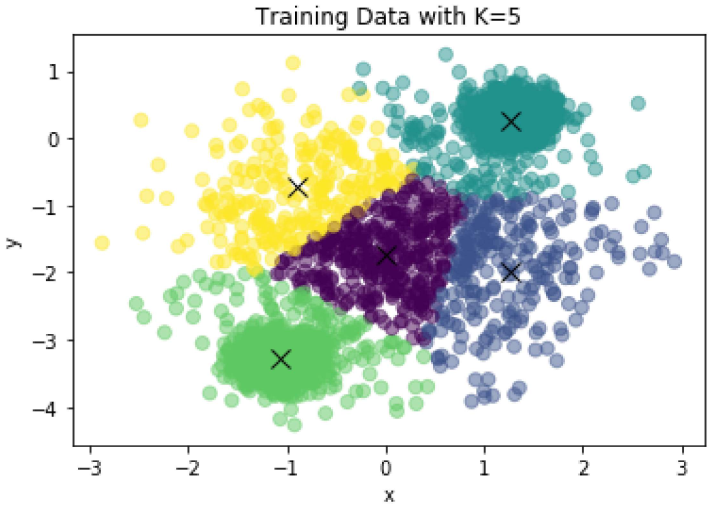
mu's: [[1.25819626 0.26221602]
 [0.81574076 -1.95066826]
 [-1.06420956 -3.26378059]
 [-0.67072014 -1.05942896]]

Validation Loss for cluster 4: 1053.55327451021

```
Percent of data points belonging to cluster 1: 0.36603660366036606
Percent of data points belonging to cluster 2: 0.1266126612661266
Percent of data points belonging to cluster 3: 0.38823882388238823
Percent of data points belonging to cluster 4: 0.11911191119111911
```




```
Number of Clusters: 5
mu's: [[ 3.52771383e-04 -1.74411351e+00]
 [ 1.26572106e+00 -1.99750336e+00]
 [ 1.25803255e+00  2.64756182e-01]
 [-1.07174598e+00 -3.27167070e+00]
 [-8.91467780e-01 -7.35423503e-01]]
Validation Loss for cluster 5: 900.0445702068167
Percent of data points belonging to cluster 1: 0.10681068106810682
Percent of data points belonging to cluster 2: 0.07170717071707171
Percent of data points belonging to cluster 3: 0.3639363936393639
Percent of data points belonging to cluster 4: 0.38433843384338434
Percent of data points belonging to cluster 5: 0.07320732073207321
```



Here, three clusters is the best number of clusters as you can distinctly see three clusters in the data (one big cluster in the middle and two smaller ones on the side). Also, five clusters aren't better than three as five is overfitting.

2 Mixtures of Gaussians

2.1 The Gaussian Cluster Mode

$$p(x; \mu_k, \Sigma_k) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp\left\{ -\frac{1}{2} (x-\mu_k) \Sigma_k^{-1} (x-\mu_k)^T \right\}$$

$$\log\left[P(x; \mu_k, \Sigma_k)\right] = -\log\left(2\pi^{D/2} |\Sigma_k|^{1/2}\right) - \frac{1}{2}(x-\mu_k)\Sigma_k^{-1}(x-\mu_k)^T$$

$$P(z=k \mid \underline{X}) = \frac{P(z=k) \prod_{n=1}^{N} P(x_n; \mu_k, \Sigma_k)}{\sum_{k=1}^{K} P(z=k) \prod_{n=1}^{N} P(x_n; \mu_k, \Sigma_k)}$$

$$\log\left[P(z=k \mid \underline{X}) \right] = \frac{\log(\pi_k) + \sum_{n=1}^{N} \log\left[P(x_n; \mu_k, \Sigma_k) \right]}{\sum_{k=1}^{K} \left\{ \log(\pi_k) + \sum_{n=1}^{N} \log\left[P(x_n; \mu_k, \Sigma_k) \right] \right\}}$$

```
# Distance function for GMM
def distanceFunc(X, MU):
 X = tf.expand_dims(X, 1)
 MU = tf.expand_dims(MU, 0)
 pair_distance = tf.reduce_sum(tf.square(tf.subtract(X, MU)), 2)
 return pair_distance

def log_GaussPDF(X, mu, sigma):
  dist = distanceFunc(X,mu)
  sigma = tf.squeeze(sigma)
  temp = tf.cast(tf.rank(X), dtype=tf.float32)
  func = -0.5 * temp * (tf.math.log(2*np.pi * sigma))
  pdf = func - (dist/(2*sigma))
  return pdf

def log_posterior(log_PDF, log_pi):
  num = tf.add( tf.squeeze(log_pi), log_PDF)
  den = reduce_logsumexp(num, axis=1, keep_dims=True)
  return num - den
```

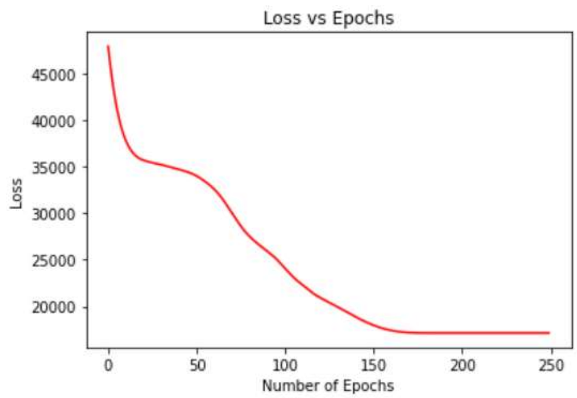
It is important to use the log-sum-exp function instead of using the tf.reduce_sum() as it prevents log overflow and makes it more stable.

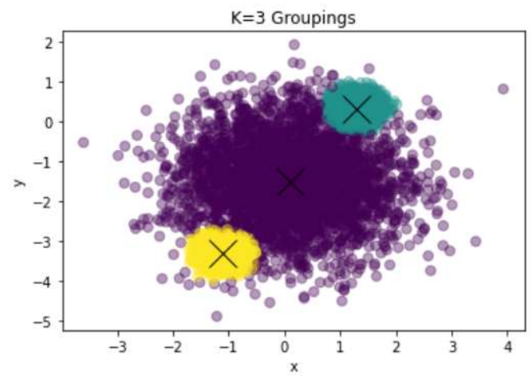
2.2 Learning the MoG

1.

Best Model parameters with epoch 250 with a final loss of 17132.377

Cluster	μ	σ	π
1	[0.10620279 -1.5268961]	0.9869102	-1.0949415
2	[1.2998326 0.3091553]	0.0388506 4	-1.0982761
3	[-1.1014876 -3.306175]	0.0392558 5	-1.1026341



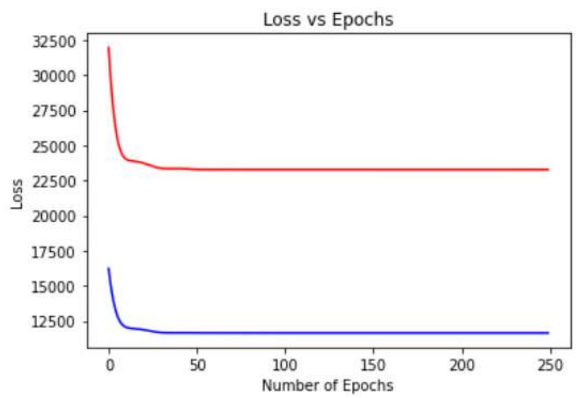


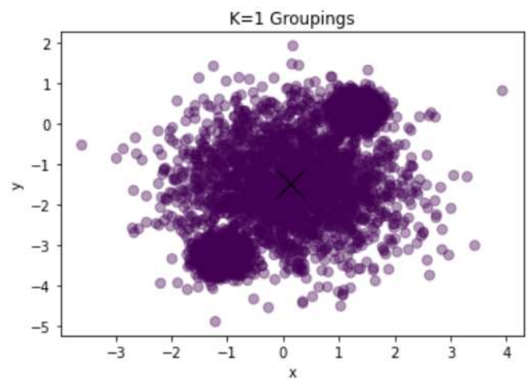
2.

	K= 1	K=2	K=3	K=4	K=5
Validation Loss	23265.98	16156.143	11505.929	11505.811	11505.761

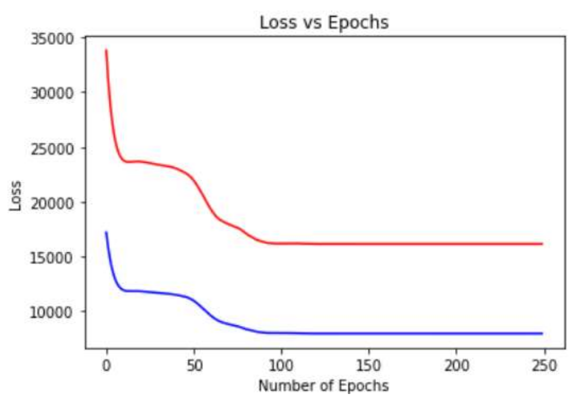
The validation loss converges to 11505 as K>=3. Thus, the best value of K is 3.

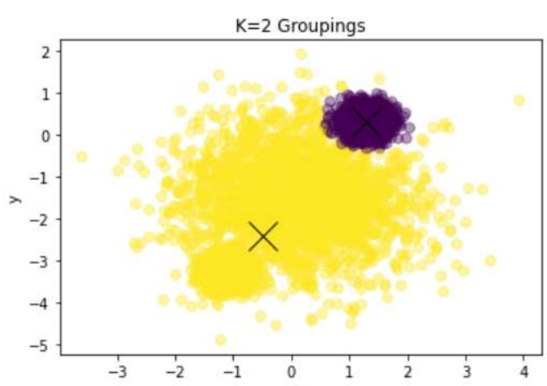
K=1



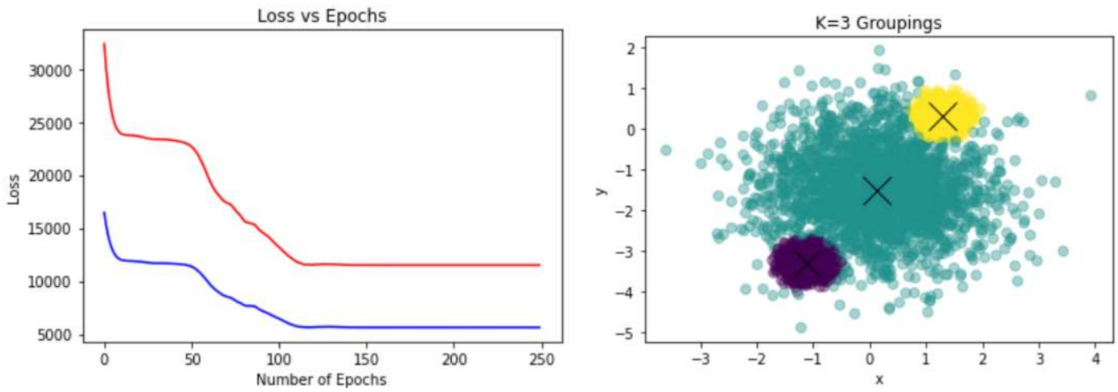


K=2

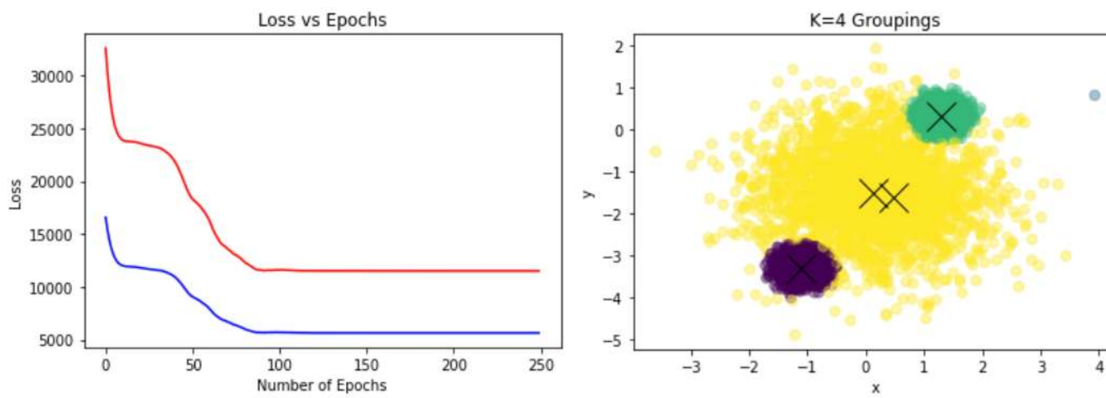




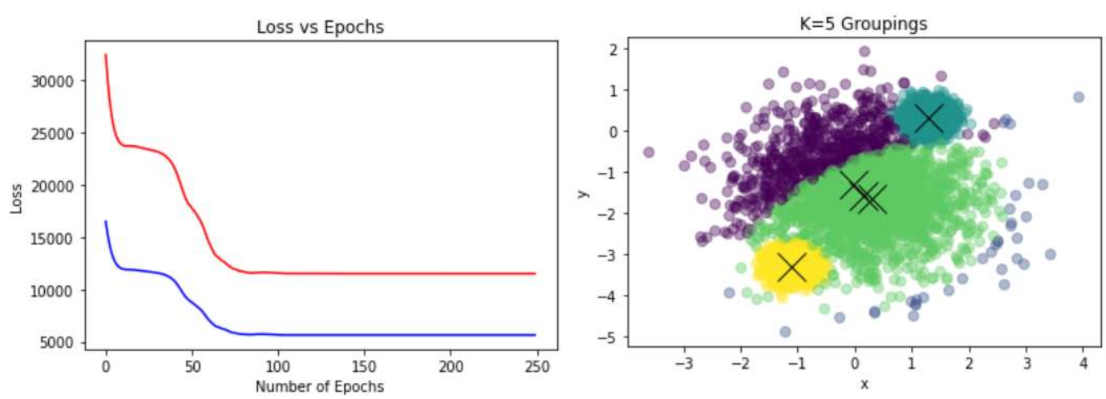
K=3


Loss vs Epochs
Loss
Number of Epochs
K=3 Groupings
y
x

K=4


Loss vs Epochs
Loss
Number of Epochs
K=4 Groupings
y
x

K=5


Loss vs Epochs
Loss
Number of Epochs
K=5 Groupings
y
x

3. **Validation Loss**

	K=5	K=10	K=15	K=20	K=30
MoG	44172.74	21365.492	22078.865	21382.27	21333.742
K-Means	122440.13	71076.23	71794.89	71171.29	69930.73

Looking at the MoG and K-Means Validation Loss, there are around K=10 clusters as the validation loss converges for K>=10 for both the MoG and K-Means.

Comparison Between MoG and K-Means:

- MoG performed better than K-means as the validation loss is much higher for the K-Means
- MoG: There was a significant decrease in the validation loss when going from K=5 to K=10 and the changes in the validation loss for K>=10 are minimal.
- K-Means: Similarly to MoG, there was a significant decrease in the validation loss between K=5 to K=10 and the changes in the validation loss for K>=10 are minimal.