ECE421 - Winter 2020

Assignment 3:

Unsupervised Learning and Probabilistic Models

Haotian Luo 1004102865
Yuheng Wang 1004222995

1. K-Means

1.1 Learning K-Means

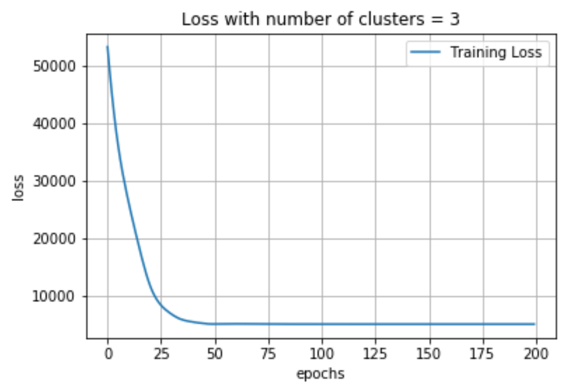
In [0]:

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import helper as hlp

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

N = num_pts
D = dim

# Distance function for K-means
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)
    # TODO

    return tf.reduce_sum(tf.squared_difference(tf.expand_dims(X,1),tf.expand_dims(MU,0)), 2)

def K_Means(k, is_valid=False):
    # For Validation set
    if is_valid:
        valid_batch = int(num_pts / 3.0)
        train_batch = N - valid_batch
        np.random.seed(45689)
        rnd_idx = np.arange(num_pts)
        np.random.shuffle(rnd_idx)
        val_data = data_in[rnd_idx[:valid_batch]]
        data = data_in[rnd_idx[valid_batch:]]
    else:
        data = data_in

    MU = tf.Variable(tf.random_normal([k, D]))
    X = tf.placeholder(tf.float32, shape = [None, D])

    dist_matrix = distanceFunc(X, MU)
    dist = tf.reduce_min(dist_matrix, axis = 1)
    cluster = tf.argmin(dist_matrix, axis = 1)
    _, _, count = tf.unique_with_counts(cluster)
    percentage = tf.divide(count,N)

    loss_mu = tf.reduce_sum(dist)
    optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9, beta2=0.99, epsilon=1e-5)
    optimizer = optimizer.minimize(loss_mu)

    train_record = []

    init = tf.global_variables_initializer()
    session = tf.InteractiveSession()
    session.run(init)

    for i in range(200):
        _, loss = session.run([optimizer, loss_mu], feed_dict = {X: data})
        train_record.append(loss)

    clusters, percentages = session.run([cluster, percentage], feed_dict = {X: data})

    if is_valid is True:
        valid_loss = session.run([loss_mu], feed_dict = {X: val_data})

        session.close()
        return valid_loss

    else:
        plt.figure()
        plt.title('Loss with number of clusters = {}'.format(k))
        plt.plot(train_record,'')
        plt.xlabel('epochs')
        plt.ylabel('loss')
        plt.grid()
        plt.legend(['Training Loss'])
        plt.savefig('Training_Loss_k={}.png'.format(k))

        print(percentages)

        plt.figure()
        plt.title('Data Points with number of clusters = {}'.format(k))
        plt.scatter(data_in[:,0], data_in[:,1], c=clusters, s=0.5, alpha=0.8)
        plt.xlabel('Dimension 1')
        plt.ylabel('Dimension 2')
        plt.grid()
        plt.savefig('Data_Points_k={}.png'.format(k))
```

```
        session.close()
        return

K_Means(1, False)
K_Means(2, False)
K_Means(3, False)
K_Means(4, False)
K_Means(5, False)

print('k = 1, valid loss = {}'.format(K_Means(1, True)))
print('k = 2, valid loss = {}'.format(K_Means(2, True)))
print('k = 3, valid loss = {}'.format(K_Means(3, True)))
print('k = 4, valid loss = {}'.format(K_Means(4, True)))
print('k = 5, valid loss = {}'.format(K_Means(5, True)))
```

1.1.1

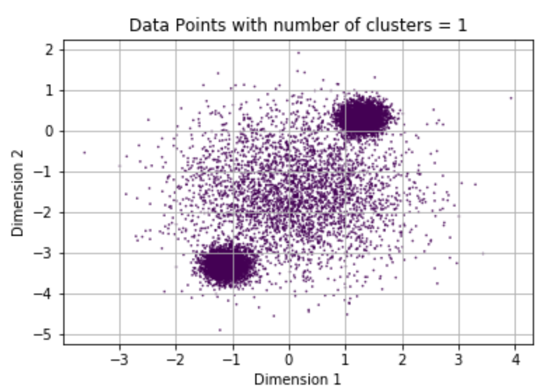
The plot below shows the loss in 200 echos when K = 3 with K-Means Model.

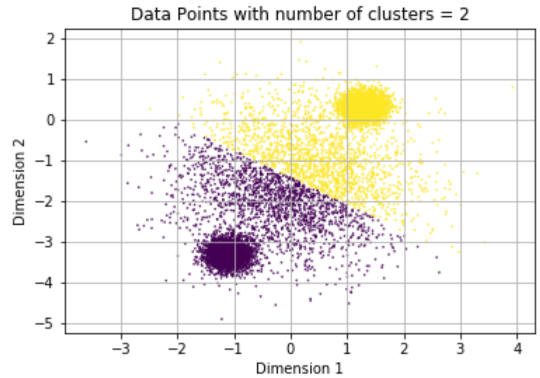


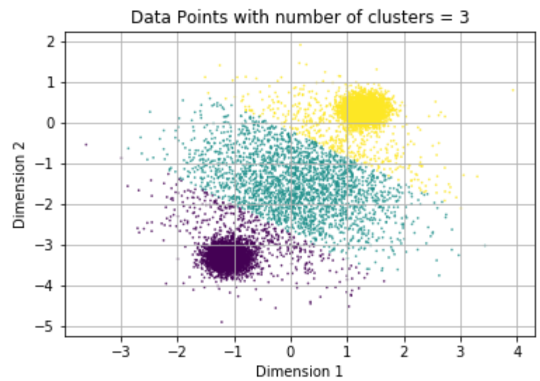
1.1.2

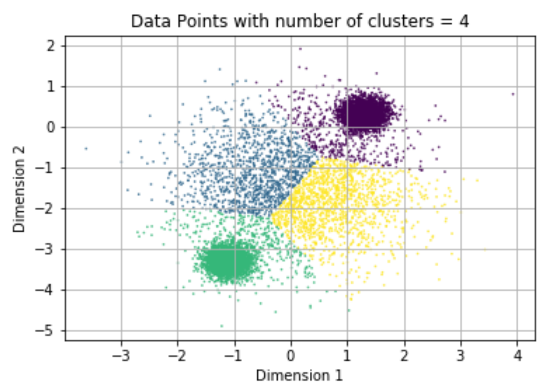
The chart below reports the percentage of data points in each clusters, when K = 1,2,3,4,5

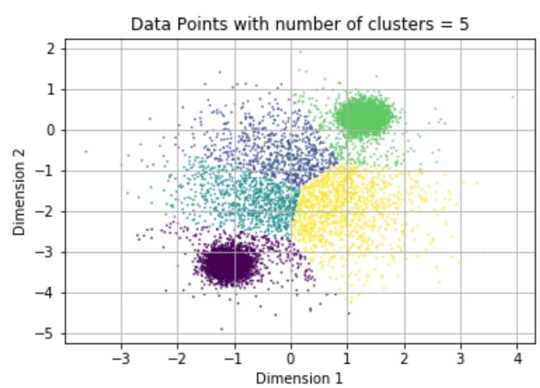
value of K	in Cluster 1	in Cluster 2	in Cluster 3	in Cluster 4	in Cluster 5
1	100%				
2	49.54%	50.46%			
3	23.81%	38.13%	38.06%		
4	13.49%	37.13%	12.1%	37.28%	
5	11.38%	36.3%	8.83%	35.93%	7.56%











1.1.3

In the below chart, we held 1/3 of the traning samples for validation set and used the rest for training set. Final validation losses are calculated for each value of K.

value of K	Validation Loss
1	12870.1
2	2960.66
3	1629.33
4	1054.59
5	902.64

As shown in the chart, the higher value of K, the less validation loss. Therefore, the best number of clusters is 5.

2. Mixture of Gaussians

2.1 The Gaussian cluster mode

In [0]:

```
# Distance function for GMM
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 pairwise distance matrix (NxK)
    # TODO
    return tf.reduce_sum(tf.squared_difference(tf.expand_dims(X,1),tf.expand_dims(MU,0)), 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

    distance = distanceFunc(X,mu)
    exp = - tf.divide(distance, 2 * tf.transpose(sigma))
    coef = - (D / 2) * tf.log(2 * np.pi) - (1/2) * tf.log(tf.transpose(sigma))
    return tf.add(coef, exp)



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

    # TODO
    num = tf.add(log_PDF, tf.transpose(log_pi))
    den = reduce_logsumexp(num, keep_dims=True)
    return tf.subtract(num,den)
```

Here, we need to do likelihood calculations in log-domain instead of linear-domain, so we use the ***reduce_logsumexp*** function instead of using ***tf.reduce_sum*** directly, thus we can prevent arithmetic underflow in this way.

2.2 Learning the MoG

In [0]:

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
#import helper as hlp

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

N = num_pts
D = dim

def MoG(k, is_valid=False):
    # For Validation set
    train_batch = N
    train_data = data
    if is_valid:
        valid_batch = int(num_pts / 3.0)
        train_batch = N - valid_batch
        np.random.seed(45689)
        rnd_idx = np.arange(num_pts)
        np.random.shuffle(rnd_idx)
        val_data = data[rnd_idx[:valid_batch]]
        train_data = data[rnd_idx[valid_batch:]]

    Mu = tf.Variable(tf.random.truncated_normal([k, D]))
    X = tf.placeholder(tf.float32, shape = [None, D])
    phi = tf.Variable(tf.random.truncated_normal([k, 1]))
    psi = tf.Variable(tf.random.truncated_normal([k, 1]))
    sigma = tf.exp(phi)
    log_pi = logsoftmax(psi)

    log_Gauss = log_GaussPDF(X,Mu,sigma)
    log_pstr = log_posterior(log_Gauss, log_pi)

    mle_predict = tf.argmax(log_pstr,axis=1)
    logloss = - tf.reduce_sum(reduce_logsumexp(log_Gauss + tf.transpose(log_pi)),axis=0)
    _, _, count = tf.unique_with_counts(mle_predict)
    percentage = tf.divide(count, train_batch)


    optimizer = tf.train.AdamOptimizer(learning_rate=0.1, beta1=0.9, beta2=0.99, epsilon=1e-5)
    optimizer = optimizer.minimize(logloss)

    train_record = []

    init = tf.global_variables_initializer()
    session = tf.InteractiveSession()
    session.run(init)

    for i in range(200):
        _, loss = session.run([optimizer, logloss], feed_dict = {X: train_data})
        train_record.append(loss)

    predict, percentages = session.run([mle_predict, percentage], feed_dict = {X: train_data})

    if is_valid is True:
        valid_loss = session.run(logloss, feed_dict = {X: val_data})
        session.close()
        return valid_loss

    else:

        plt.figure()
        plt.title('Loss with number of gaussian clusters = {}'.format(k))
        plt.plot(train_record,'')
        plt.xlabel('epochs')
        plt.ylabel('loss')
        plt.grid()
        plt.legend(['Training Loss'])
        plt.savefig('Gaussian_Training_Loss_k={}.png'.format(k))

        plt.figure()
        plt.title('Data Points with number of gaussian clusters = {}'.format(k))
        plt.scatter(data[:,0], data[:,1], c=predict, s=0.5, alpha=0.8)
        plt.xlabel('Dimension 1')
        plt.ylabel('Dimension 2')
        plt.grid()
        plt.savefig('Gaussian_Data_Points_k={}.png'.format(k))


        print("Final value of phi and psi:")
        print(phi.eval())
        print(psi.eval())

        print("Final value of sigma and pi:")
```

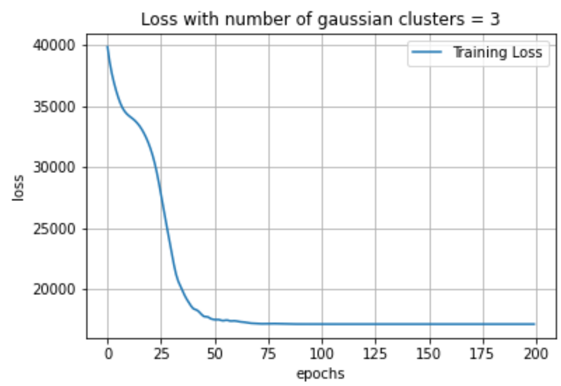
```
        print(sigma.eval())
        print(tf.exp(log_pi).eval())


        session.close()
        return
```

2.2.1

These charts summarizes the final model parameters after learning the samples, when the number of clusters K = 3

K =	ϕ_k =	ψ_k =	σ_k =	π_k =
1	-0.0127	0.0147	0.987	0.335
2	-3.249	0.0143	0.0388	0.333
3	-3.242	0.1387	0.0391	0.332

The plot below shows the loss in 200 echos when K = 3 with MoG Model.



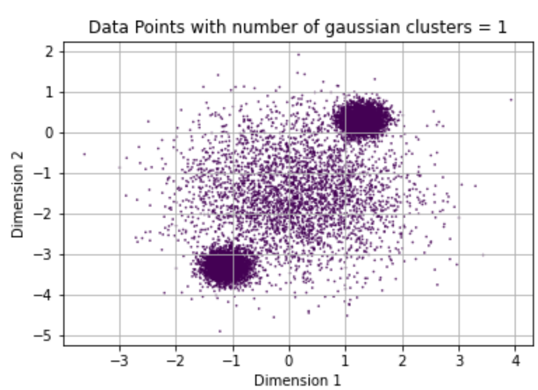
2.2.2

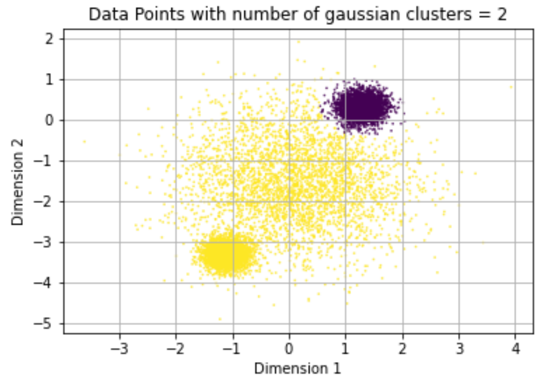
In the below chart, we held 1/3 of the traning samples for validation set and used the rest for training set. Final validation losses are calculated for each value of K.

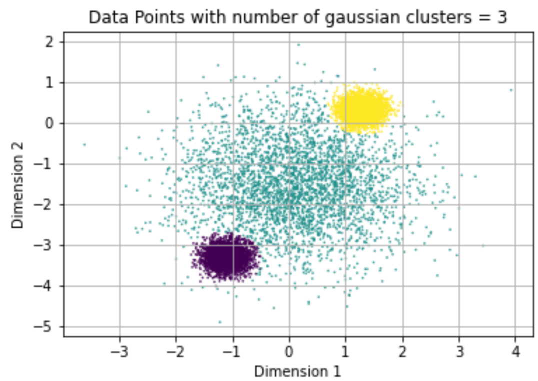
value of K	Validation Loss
1	11651.44
2	8013.86
3	5632.74
4	5630.33
5	5630.22

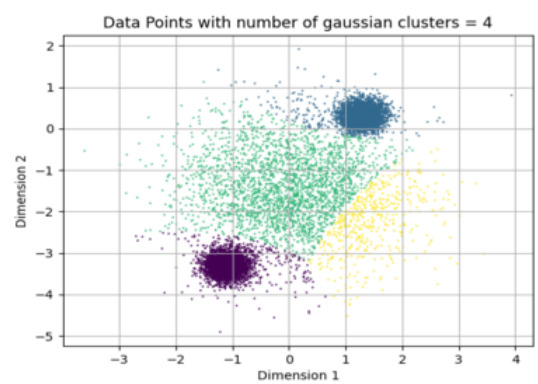
- As shown in the above chart, the higher value of K, the less validation loss.
- However, one can observe that for $K \geq 3$, the valiadation loss stabalizes. Having more than 3 clusters does not improve the model futhermore.
- Therefore, the best number of clusters is 3.

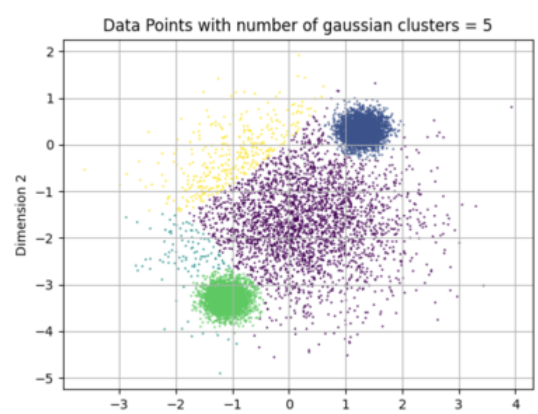
The following plots show clusters of data points when training with the MoG model, K = 1,2,3,4,5











2.2.3

In the below chart, we held 1/3 of the data for validation, and we ran both the K-Means model and the MoG model. The dataset is the **"data100D.npy"**. Final validation losses are calculated.

value of K	Validation Loss of K-Means	Distortion to next choice of K	Validation Loss of MoG	Distortion to next choice of K
5	123213.305	53045.255	386271.97	224344.8
10	70078.05	696.13	161927.17	-890.27
15	69381.92	115.14	162817.44	890.27
20	69266.78	751.65	161927.17	-60.33
30	68515.13		162530.47	

We compare across different values of K in each model and find the lowest distortion. Distortion is the difference in validation loss between current choice of K and the next choice of K, and the best value of K is chosen with the lowest distortion:

- With the MoG model, the number of clusters appears to be 10
- With the K-Means model, the number of clusters appears to be 15

Comparing the learnt results between K-Means and MoG:

- K-Means model has strict improvements on loss as K is increasing. The more number of clusters, the better the learnt result.
- MoG model stabalizes after a certain threshold number of clusters is reached. When every cluster is distinct enough, break down the data points furthermore does not improve the learnt result significantly.