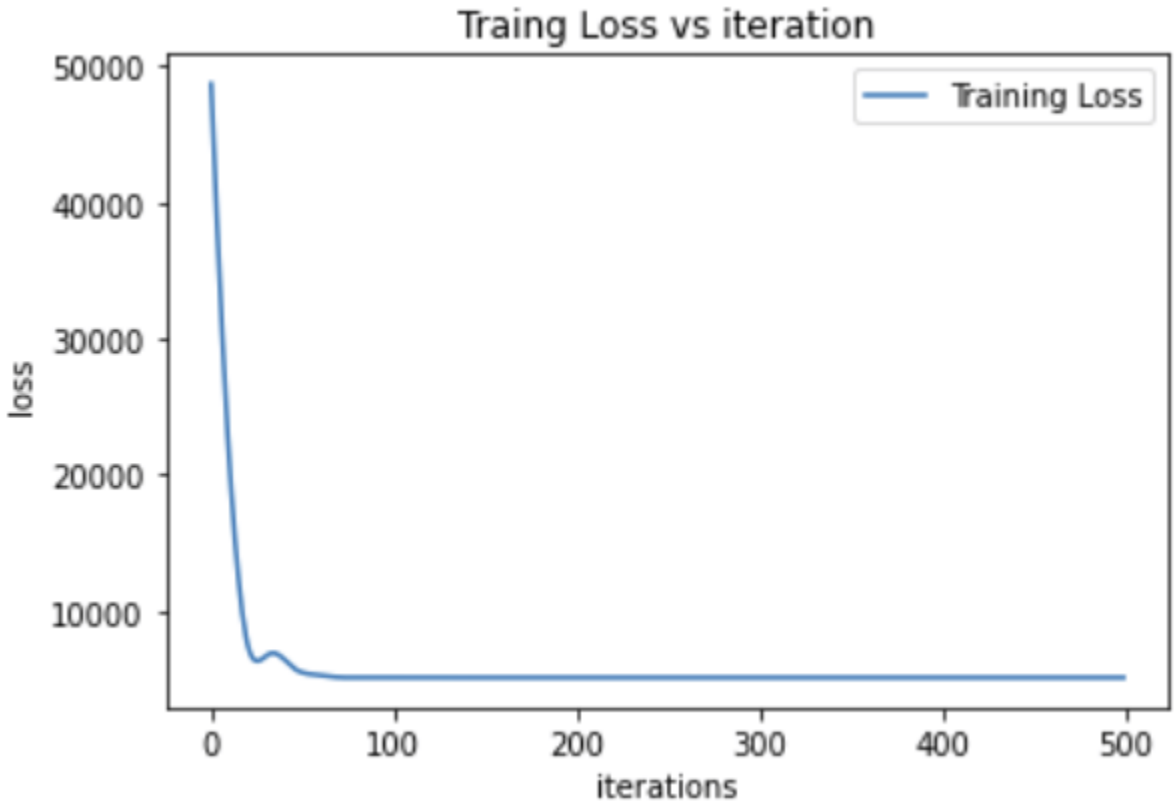
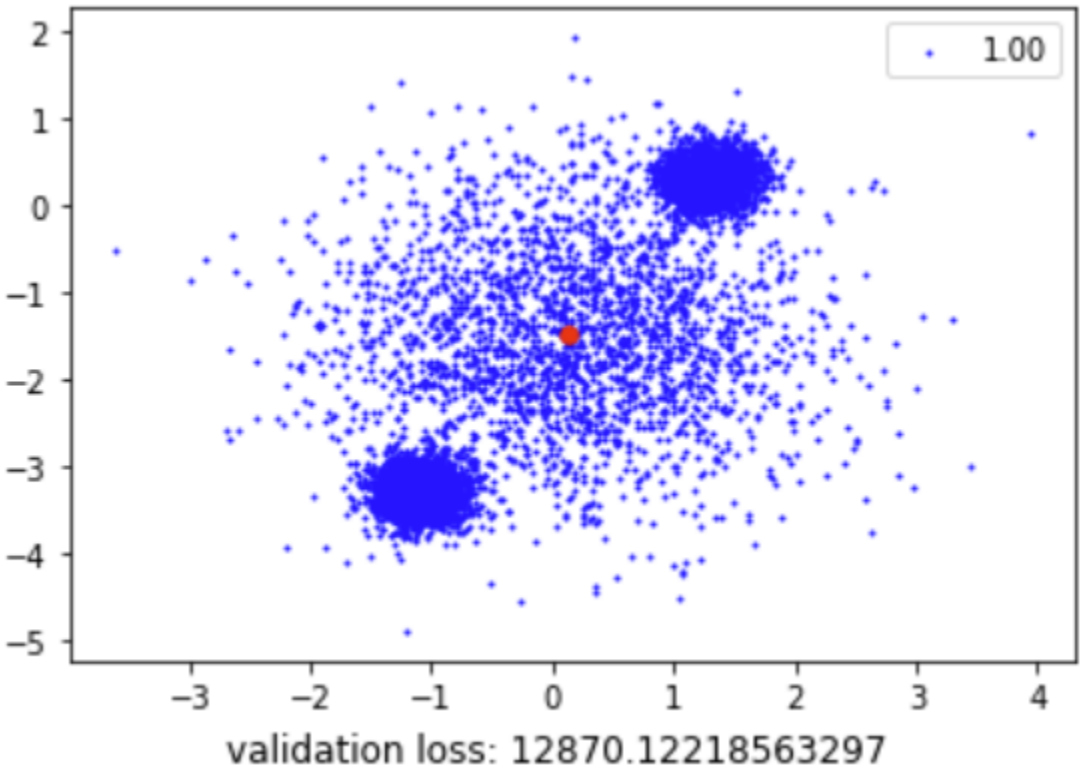
Section 1

```python
# Distance function for K-means, independent test passed
def distance_func(X, mu):
    """ Inputs:
          X: is an NxD matrix (N observations and D dimensions)
          mu: is an KxD matrix (K means and D dimensions)

        Output:
          pair_dist: is the squared pairwise distance matrix (NxK)
    """
    pair_dist = tf.expand_dims(X, axis=1)
    pair_dist = (pair_dist - mu) ** 2
    pair_dist = tf.reduce_sum(pair_dist, -1)

    return pair_dist
```

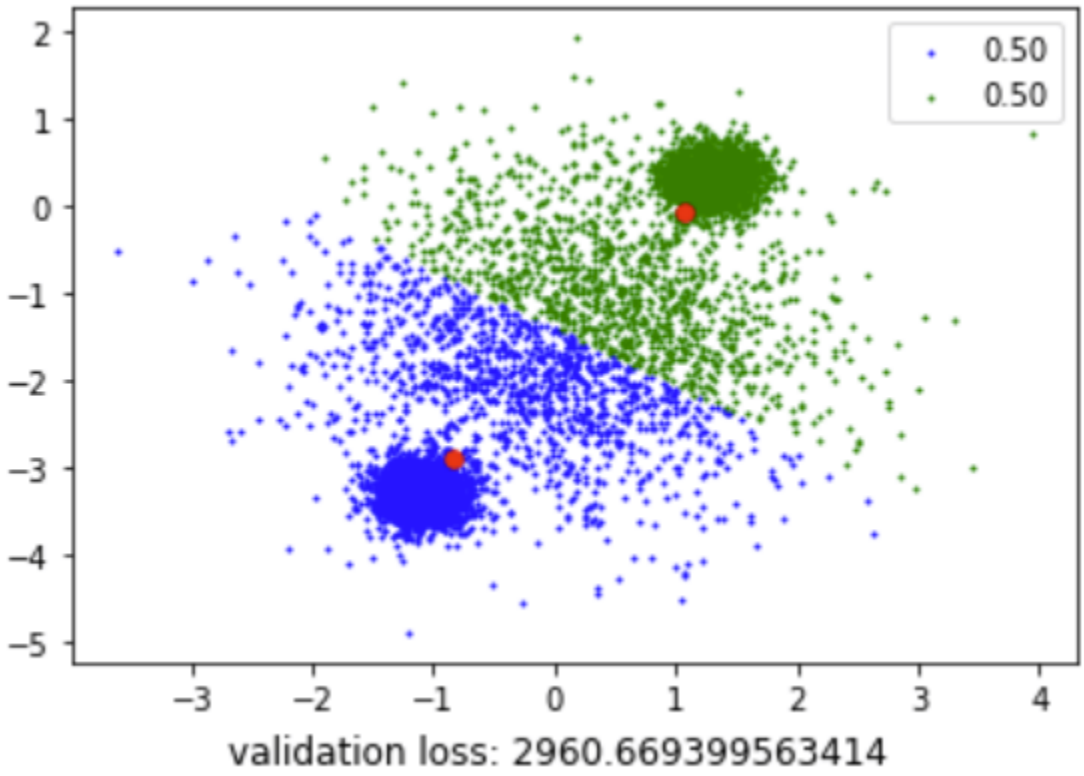


Plot 1: K-mean loss vs iteration

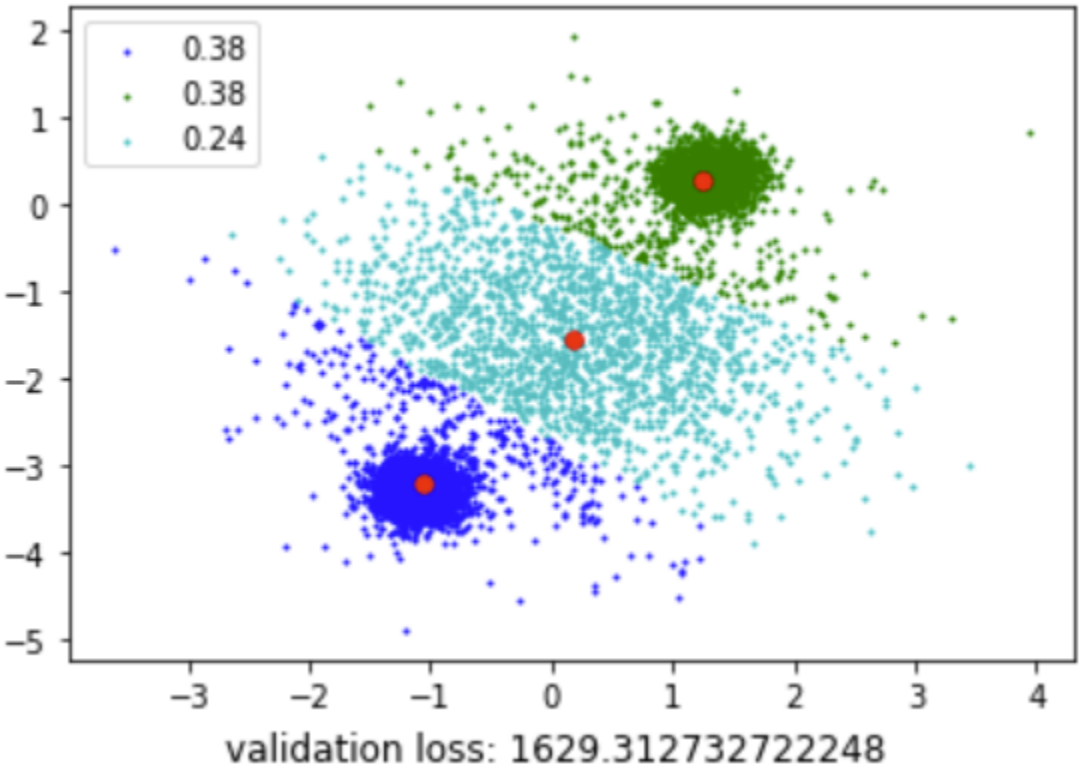
It can be observed from plot 1 that the training fully converge within 200 iterations, so 200 is chosen for 1.2 training



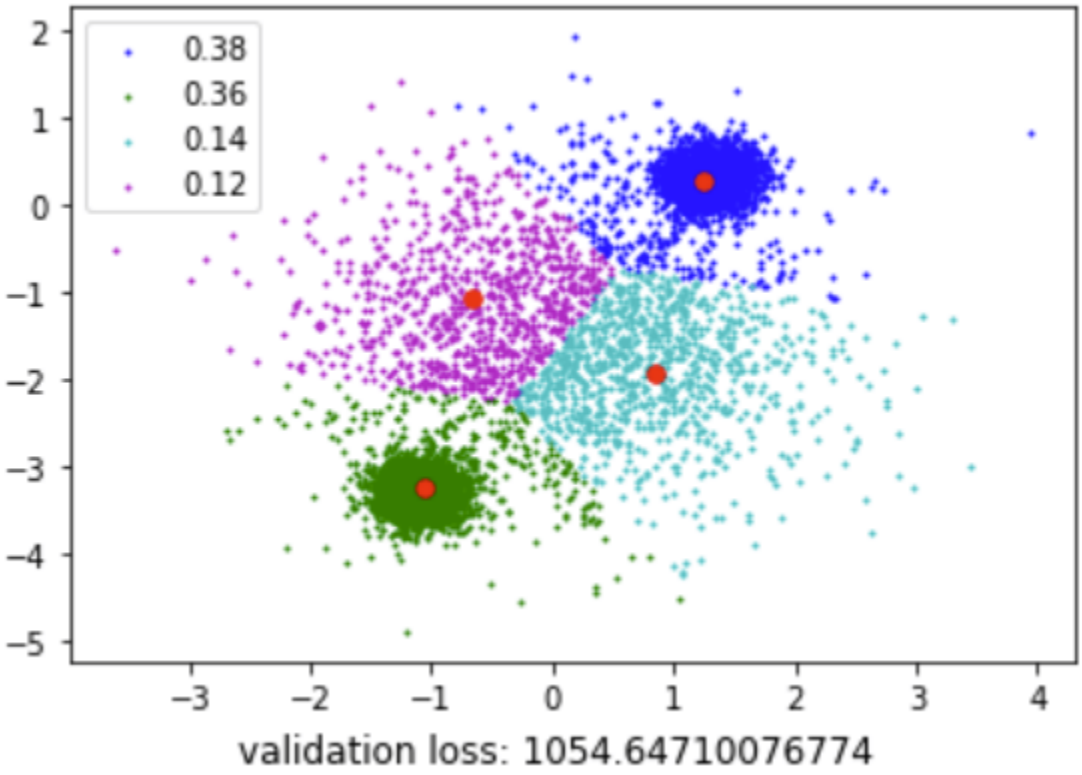
Plot 2: loss vs iteration @ K = 1



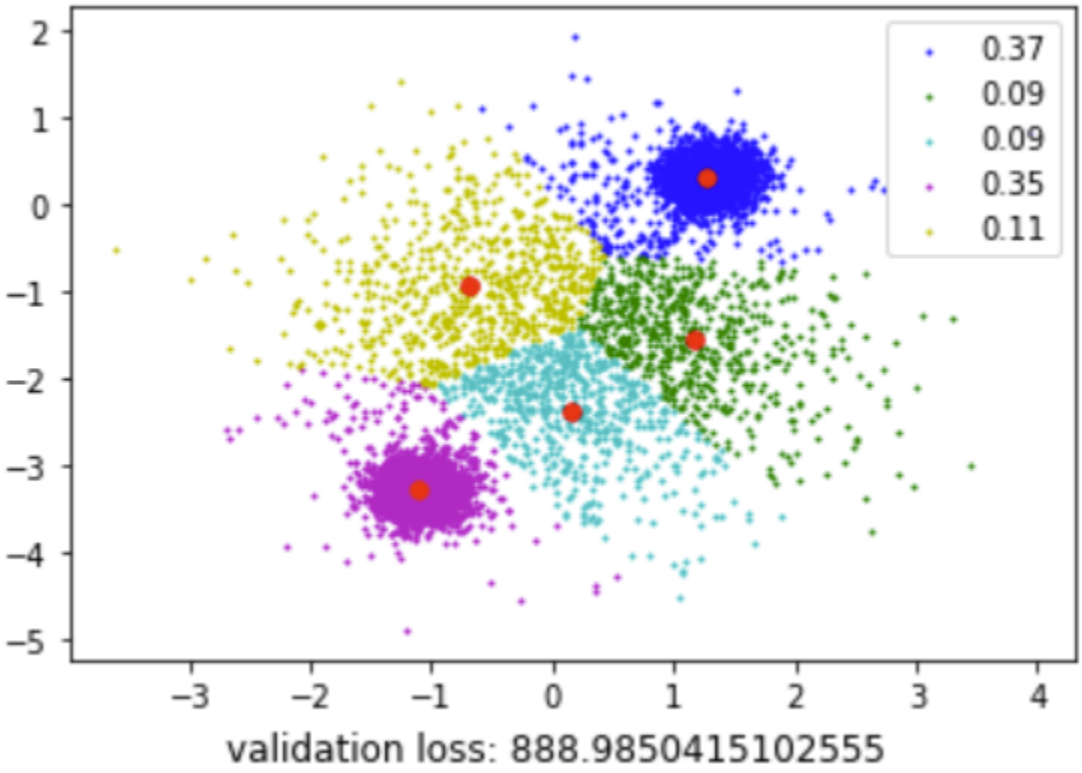
Plot 3: loss vs iteration @ K = 2



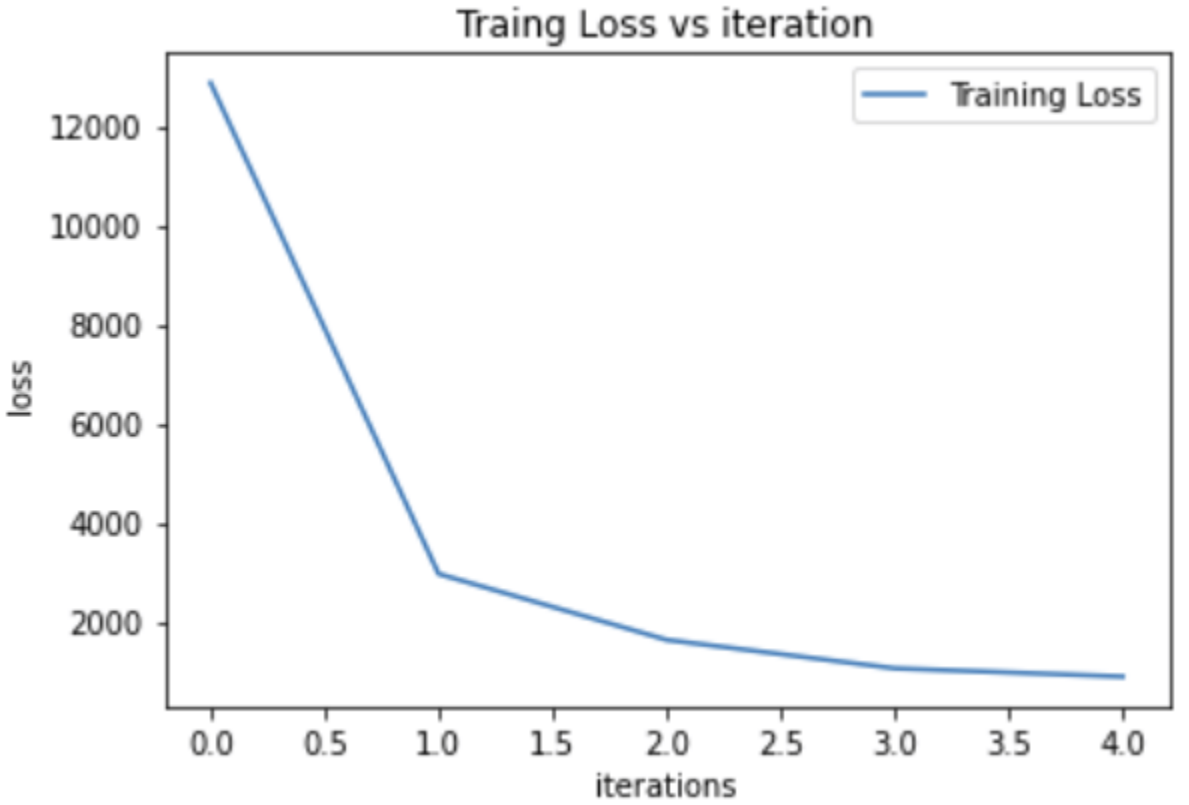
Plot 4: loss vs iteration @ K = 3



Plot 5: loss vs iteration @ K = 4



Plot 6: loss vs iteration @ K = 5



Plot 7: validation loss vs K = 1, …, 5

From plot 7, the validation loss drops slower after K = 2. Hence, K = 2 is the best number of clusters.

Section 2.1

```
# calculate log normal distributions for all data and clusters,
# Args: data [N*D], center [K*D], variance [K]
# Returns: log probability [N*K]
def log_gauss_pdf(X, mu, sigma, dim):
  # taking the log of the distribution, probability is in two terms: coef and exp
  # steps in report
  dist = distance_func(X, mu)
  dist = tf.cast(dist, tf.float32)
  exp = - dist / 2 / sigma

  coef = - (tf.math.log(2 * math.pi * sigma)) * dim / 2

  return exp + coef

# this is the log_posterior function, I just name it differently
# calculate the log probability of cluster given data
# Args: data [N*D], weight [K]
# Returns: log probability [N*K]

def log_posterior(log_pdf, log_pi):

  logWeightedProb = log_pdf + log_pi
  logSum = reduce_logsumexp(logWeightedProb, reduction_indices=1, keep_dims=True)
  return logWeightedProb - logSum
```

Log_sum_exp() is must used because it calculates the sum of the variables under the log domain instead of original data.

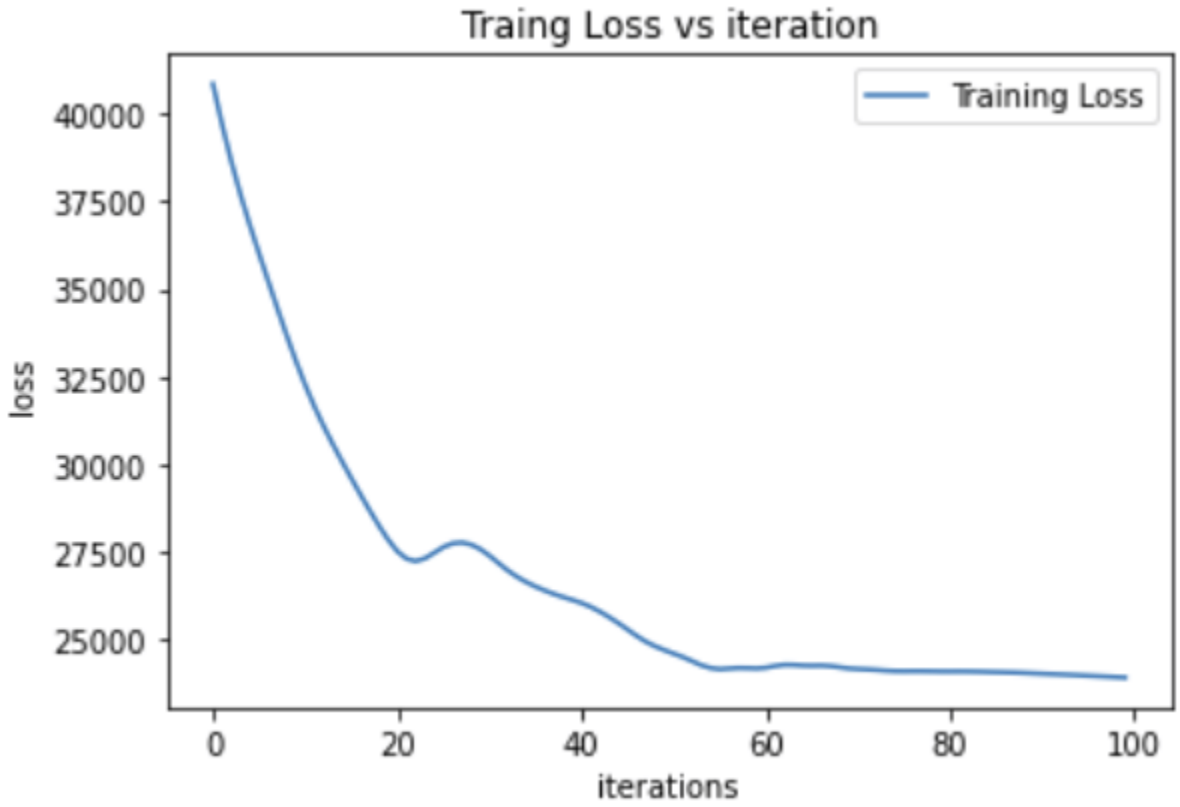
Section 2.2

The hyperparameters are:

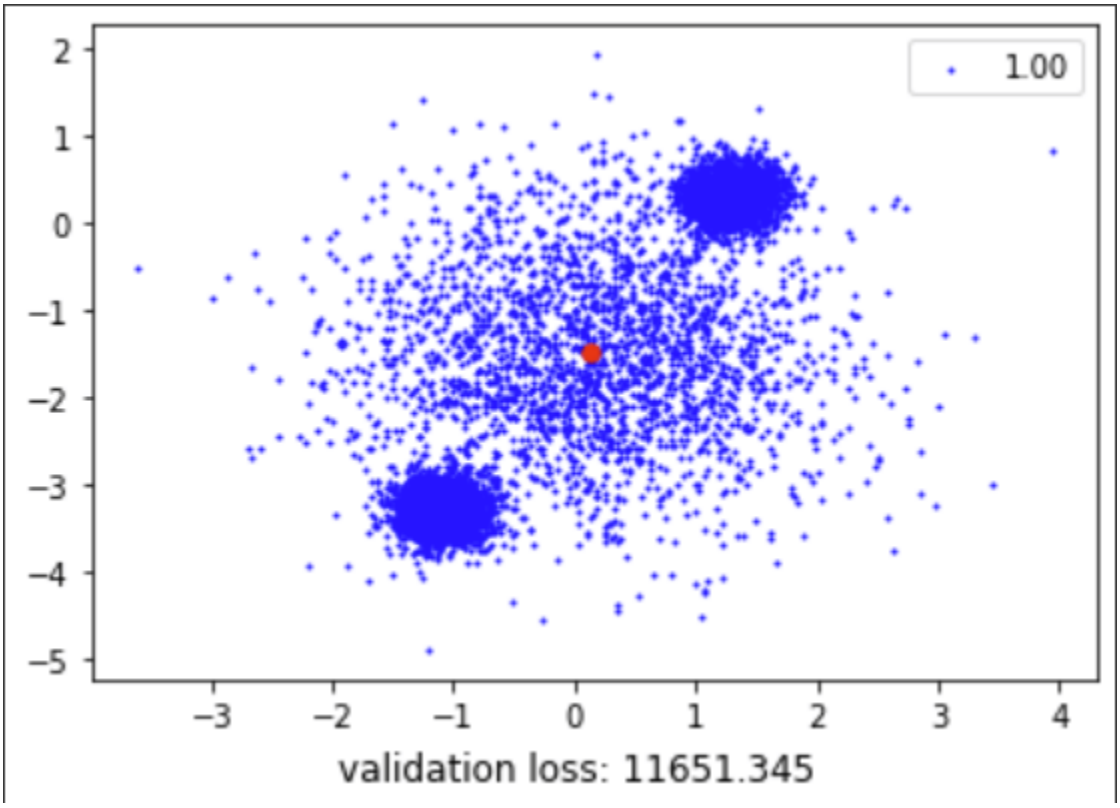
mu = [[-1.1006, -3.3062], [0.106, -1.5275], [1.2986, 0.3092]]

sigma = [0.0391, 0.9872, 0.0389]

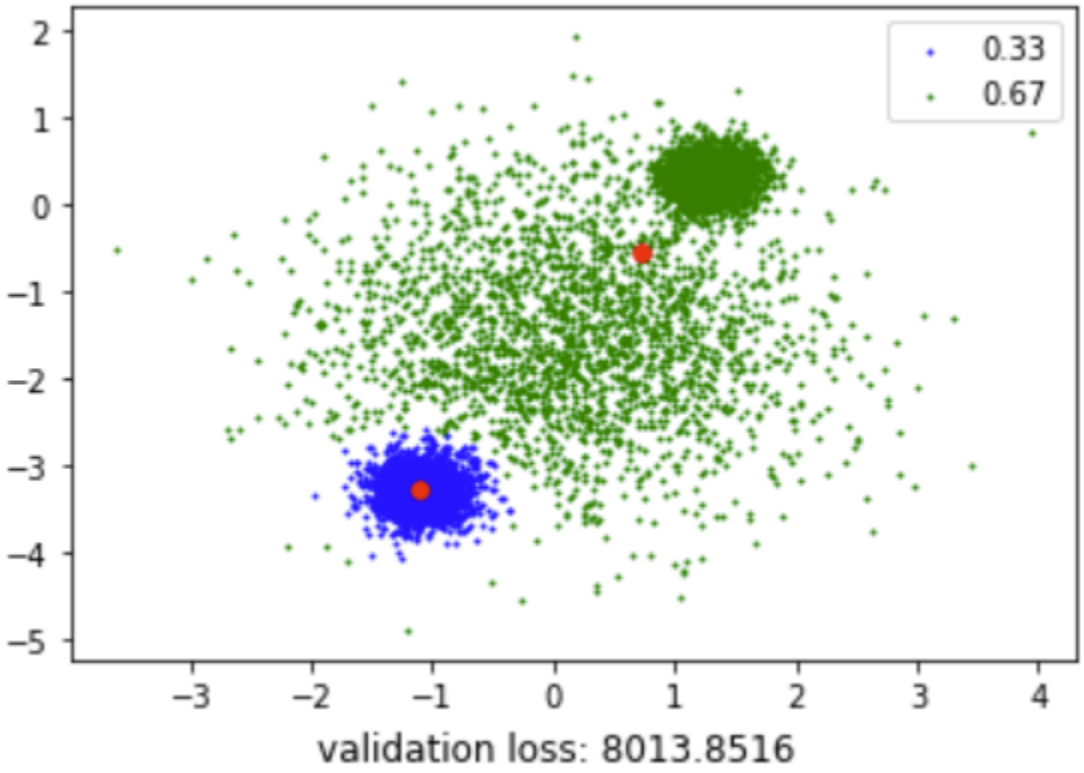
pi = [0.3319, 0.3347, 0.3334]



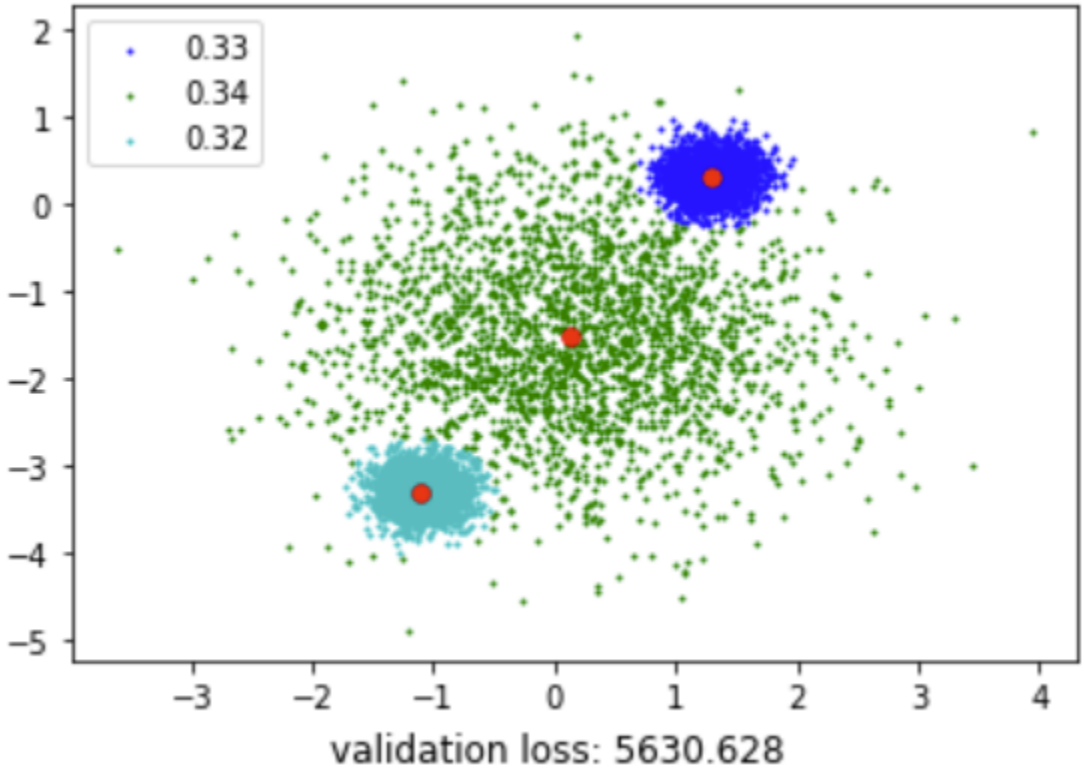
Plot 8: training loss vs updates @ K = 3



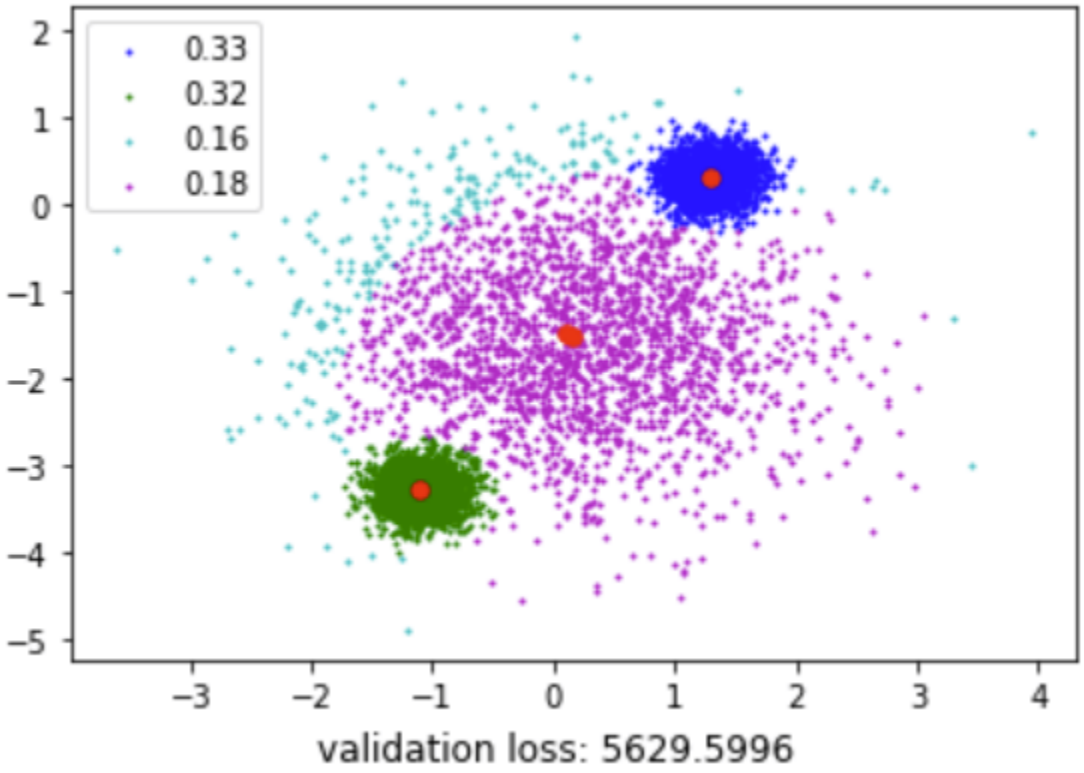
Plot 9: validation loss @ K = 1



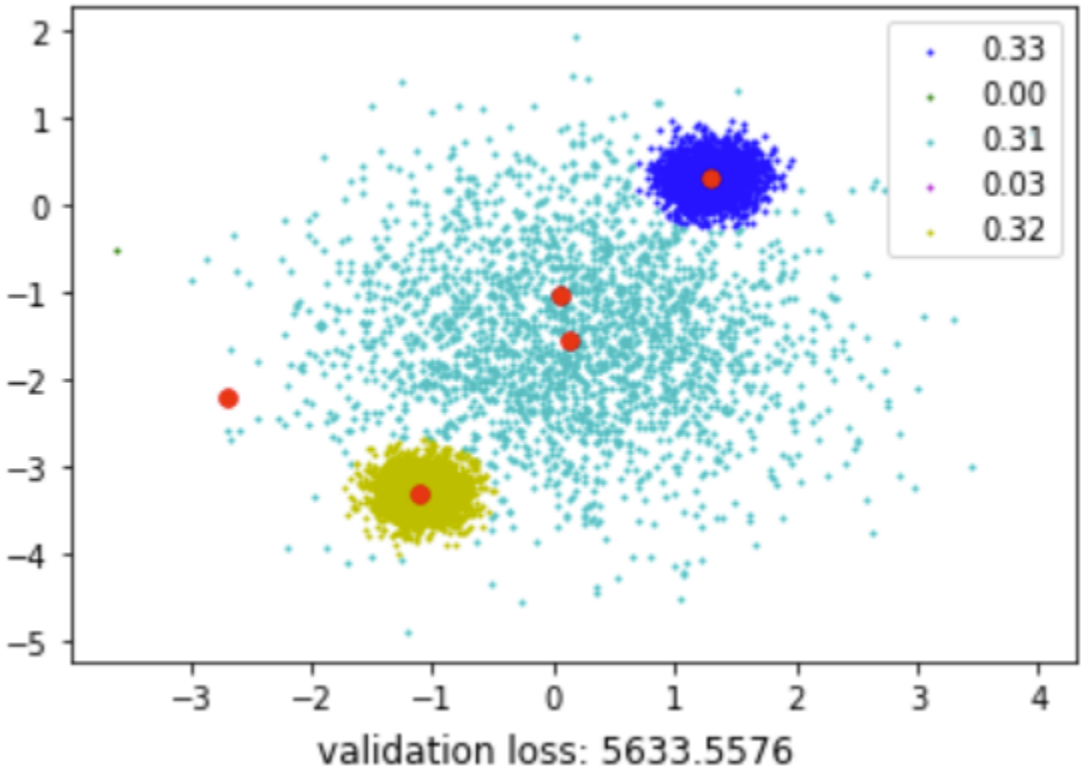
Plot 10: validation loss @ K = 2



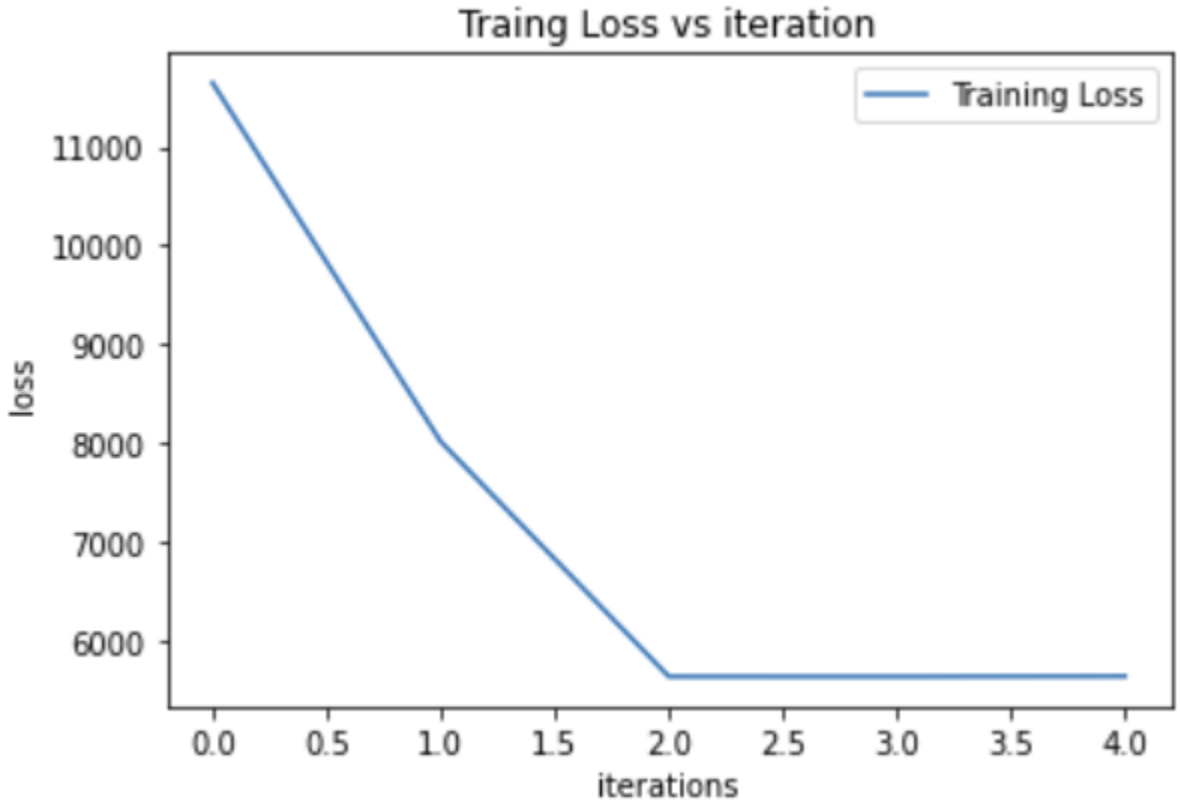
Plot 11: validation loss @ K = 3



Plot 12: validation loss @ K = 4

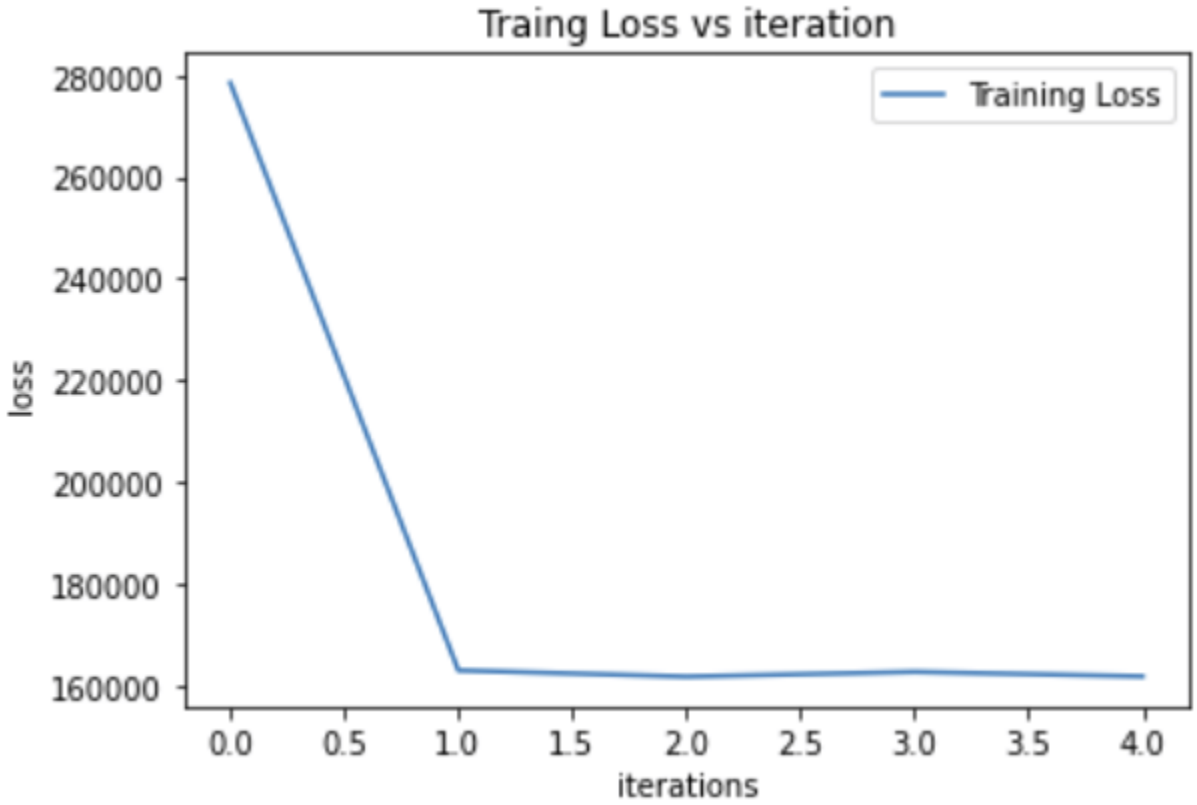


Plot 12: validation loss @ K = 5



Plot 12: validation loss vs K = 1 … 5

It can be observed that the validation loss no longer drops for K greater than 3, so the best K = 3.



Plot 13: validation loss vs K = 5 … 25 @ 100d with GoM

Plot 13: validation loss vs K = 5 … 25 @ 100d with K-means

From the above two plots, it can be estimated that the dataset has 10 clusters. For K-means, increasing K way higher than actual cluster number can bring negative results. However, because GoM has the probability of each cluster so this will not be a problem.