

ECE521 Winter 2017: Assignment 3

FuYuan Tee, (999295837) * Chee Loong Soon, (999295793) †

March 24th, 2017

Contents

1 K-means	3
1.1 Learning K-means	3
1.1.1 Convexity of $\mathcal{L}(\mu)$	3
1.1.2 K-means without validation, ($K = 3$)	5
1.1.3 K-means without validation, ($K = 1, 2, 3, 4, 5$)	6
1.1.4 K-means with validation	10
2 Mixture of Gaussians (MoG)	11
2.1 The Gaussian cluster model	11
2.1.1 Deriving the latent variable posterior distribution for a data point, $P(z \mathbf{x})$	11
2.1.2 Computing the log probability density function, $\log \mathcal{N}(\mathbf{x} \mu_k, \sigma_k^2 \mathbf{I}_D)$.	12
2.1.3 Computing the conditional responsibilities, $\gamma(\mathbf{x}) = \log P(z \mathbf{x})$. .	14
2.2 Learning the MoG	16
2.2.1 Implicit cluster assignment variable	16
2.2.2 MoG without validation on <i>data2D.npy</i>	17
2.2.3 MoG with validation	19

*Equal Contribution (50%), fuyuan.tee@mail.utoronto.ca

†Equal Contribution (50%), cheeloong.soon@mail.utoronto.ca

2.2.4	K-means and MoG on <i>data100D.npy</i>	23
3	Discover Latent Dimensions	26
3.1	Factor Analysis	26
3.1.1	Deriving the marginal log likelihood for a single training example	26
3.1.2	Factor Analysis on <i>tinymnist.npz</i>	27
3.1.3	Comparison between PCA and FA	30
4	Appendices	32
4.1	FuYuan Tee's Implementation	32
4.1.1	Base Code for K-means	32
4.1.2	K-means - Q1.1.2	41
4.1.3	K-means - Q1.1.3	42
4.1.4	K-means - Q1.1.4	45
4.1.5	Base Code for MoG	47
4.1.6	MoG - Q2.2.2	60
4.1.7	MoG - Q2.2.3	61
4.1.8	MoG - Q2.2.4	70
4.1.9	Base code for FA	72
4.1.10	FA - Q3.1.2	84
4.1.11	FA - Q3.1.3	86
4.2	Soon Chee Loong's Implementation	87
4.2.1	K-Means	87
4.2.2	Mixture of Gaussians	96
4.2.3	Helper Functions	107
4.2.4	FactorAnalysis	110

1 K-means

1.1 Learning K-means

1.1.1 Convexity of $\mathcal{L}(\mu)$

$$L(\mu) = \sum_{n=1}^B \min_{k=1}^K \|x_n - \mu_k\|_2^2 \quad (1)$$

B is the number of training instances, K is the number of clusters. μ_k is the cluster mean for cluster k . x_n is the data points for the n^{th} input data. D is the dimension of the input data.

For Equation 1 to convex, it must satisfy the Jensen's Inequality as shown in Equation 2.

$$L(\alpha\mu^1 + (1 - \alpha)\mu^2) \leq \alpha L(\mu^1) + (1 - \alpha)L(\mu^2) \quad (2)$$

where $\alpha \in [0, 1]$

We will prove that the $L(\mu)$ function in Equation 1 is not Convex by a proof by Counter Example. We will show there exist an example that does not satisfy the Jensen Inequality, hence proving that the function $L(\mu)$ is not convex.

Proof By Counter-Example:

Let $B = 1, D = 1$ represent a one single dimension input data, x_1 . Let $K = 2$ represent two different number of clusters we are supposed to train and let μ_1 and μ_2 be their respective means.

The first configuration of cluster, μ^1 is shown in equation 3.

$$\begin{aligned}
 \mu^1 &= (\mu_1^1, \mu_2^1) = (1, 3) \\
 x_1 &= (1) \\
 L(\mu^1) &= \min[(x_1 - \mu_1^1)^2, (x_1 - \mu_2^1)^2] \\
 &= \min[(1 - 1)^2, (1 - 3)^2] \\
 &= \min(0, 4) \\
 &= 0
 \end{aligned} \tag{3}$$

The second configuration of cluster, μ^2 is shown in equation 4

$$\begin{aligned}
 \mu^2 &= (\mu_1^2, \mu_2^2) = (3, 1) \\
 x_1 &= (1) \\
 L(\mu^2) &= \min[(x_1 - \mu_1^2)^2, (x_1 - \mu_2^2)^2] \\
 &= \min[(1 - 3)^2, (1 - 1)^2] \\
 &= \min(4, 0) \\
 &= 0
 \end{aligned} \tag{4}$$

Therefore, from equation 2, set $\alpha = 0.5$.

The Right Hand side of 2 would be 0 as shown in equation 5.

$$\begin{aligned}
 \alpha L(\mu^1) + (1 - \alpha)L(\mu^2) &= 0.5 * 0 + (1 - 0.5) * 0 \\
 &= 0
 \end{aligned} \tag{5}$$

However, the Left hand side of 2 would be 1 as shown in equation 6.

$$\begin{aligned}
 L(\alpha\mu^1 + (1 - \alpha)\mu^2) &= L(0.5[1, 3] + (1 - 0.5)[3, 1]) \\
 &= L([2, 2]) \\
 &= \min[(1 - 2)^2, (1 - 2)^2] \\
 &= \min(1, 1) \\
 &= 1
 \end{aligned} \tag{6}$$

Since equation 6 is $>$ equation 5, this violates Jensen's Inequality as shown in 2. This proves that the Loss function, $L(\mu)$ is NOT convex.

1.1.2 K-means without validation, ($K = 3$)

The K-means algorithm was employed to cluster $data2D.npy$ using $K = 3$ clusters. An Adam optimizer was used in the training of the K-means cluster centers. The cluster centers, μ , learnt from this training are as shown in Table 1.

Table 1: Cluster centers using K-means ($K = 3$) on $data2D.npy$ without validation

Cluster	Center
1	(-1.1, -3.2)
2	(0.1, -1.5)
3	(1.2, 0.3)

A graph of the loss function, $\mathcal{L}(\mu) = \sum_{n=1}^B \min_{k=1}^K \|\mathbf{x}_n - \mu_k\|_2^2$, as training progressed is depicted in Figure 1.

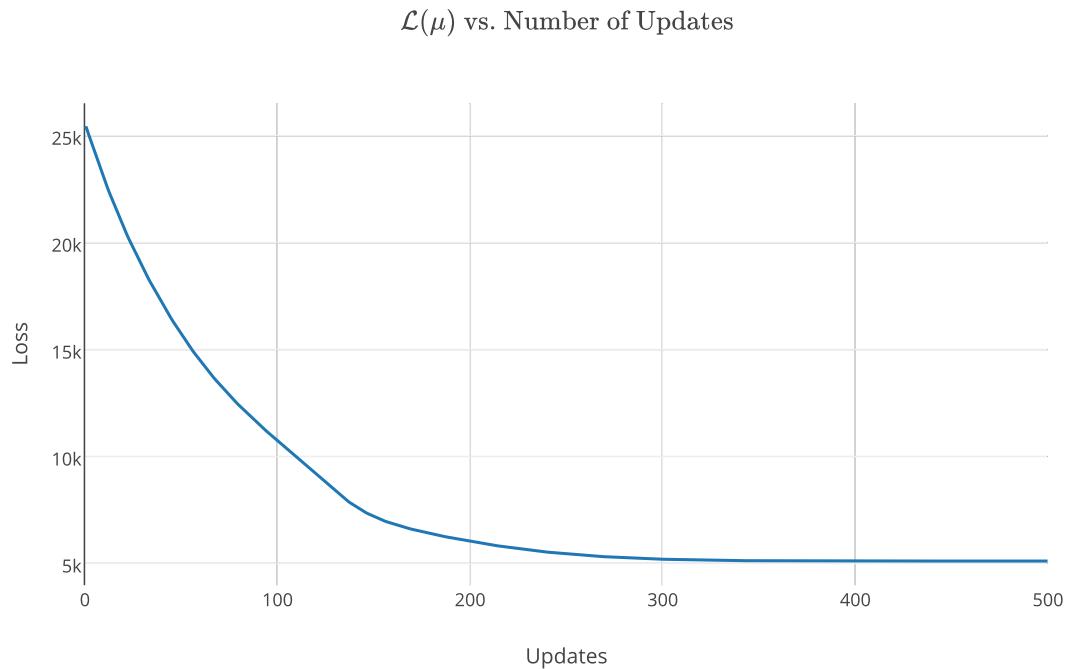


Figure 1: Loss function with respect to training updates.

1.1.3 K-means without validation, ($K = 1, 2, 3, 4, 5$)

The training procedure from the previous section was repeated for values of K ranging from 1 to 5. The training results are summarised in Table 2. The percentage of data points assigned to each cluster is also illustrated in Figure 2.

Table 2: K-means training results on *data2D.npy* without validation

K	Training Loss	Cluster	Center	% Data Points
1	38454	1	(0.1, -1.5)	100
2	9203	1	(1.1, -0.1)	49.5
		2	(-0.8, -2.9)	50.5
3	5111	1	(-1.1, -3.2)	38.2
		2	(0.1, -1.5)	23.8
		3	(1.2, 0.3)	38.0
4	3374	1	(0.8, -2.0)	13.5
		2	(-0.7, -1.1)	12.1
		3	(1.3, 0.3)	37.3
		4	(-1.1, -3.3)	37.1
5	2848	1	(0.3, -2.4)	8.7
		2	(-0.7, -1.0)	10.7
		3	(1.3, 0.3)	36.1
		4	(-1.1, -3.3)	35.8
		5	(1.1, -1.4)	8.6

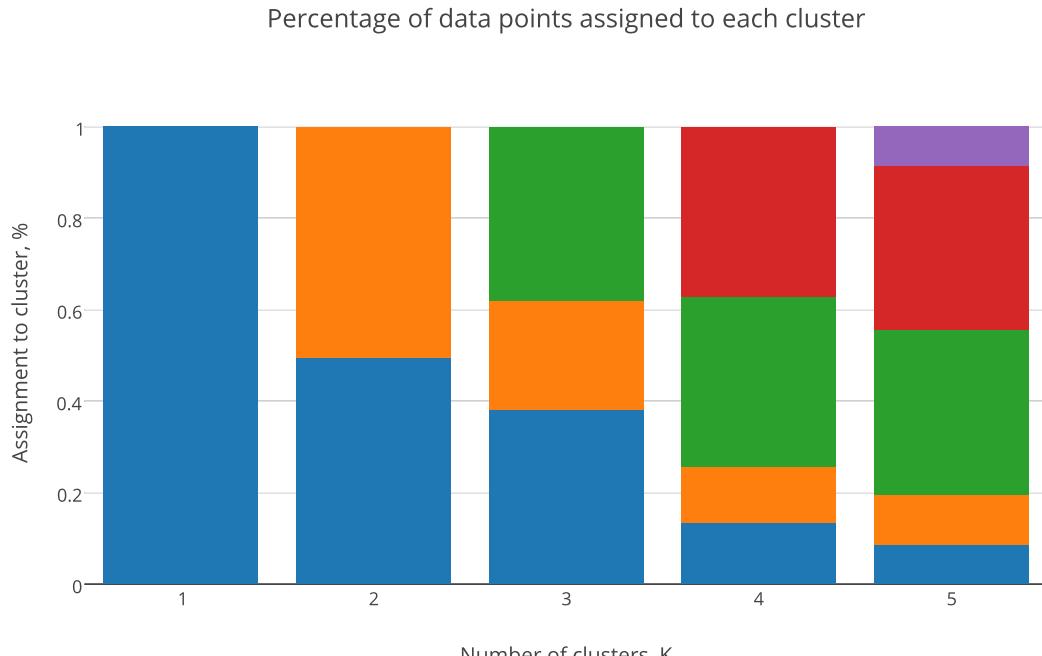


Figure 2: Percentage of data points assigned to the first, second, third, fourth and fifth clusters for $K = 1, 2, 3, 4, 5$.

To understand how many clusters to best cluster the dataset, we provided plots visualising the assignments of data points to each cluster. These illustrations can be found in Figures 3 and 4.

From inspecting these figures, the data points can be seen to comprise of 3 clusters. However, the hard K-means clustering of $K = 3$ leads to non-ideal clustering since portions of the data points that visually belong to the middle cluster were assigned to the top-most and bottom-most clusters (Figure 3, $K = 3$). Hence, a clustering assignment of $K = 4$ looks to be an appropriate and parsimonious choice.

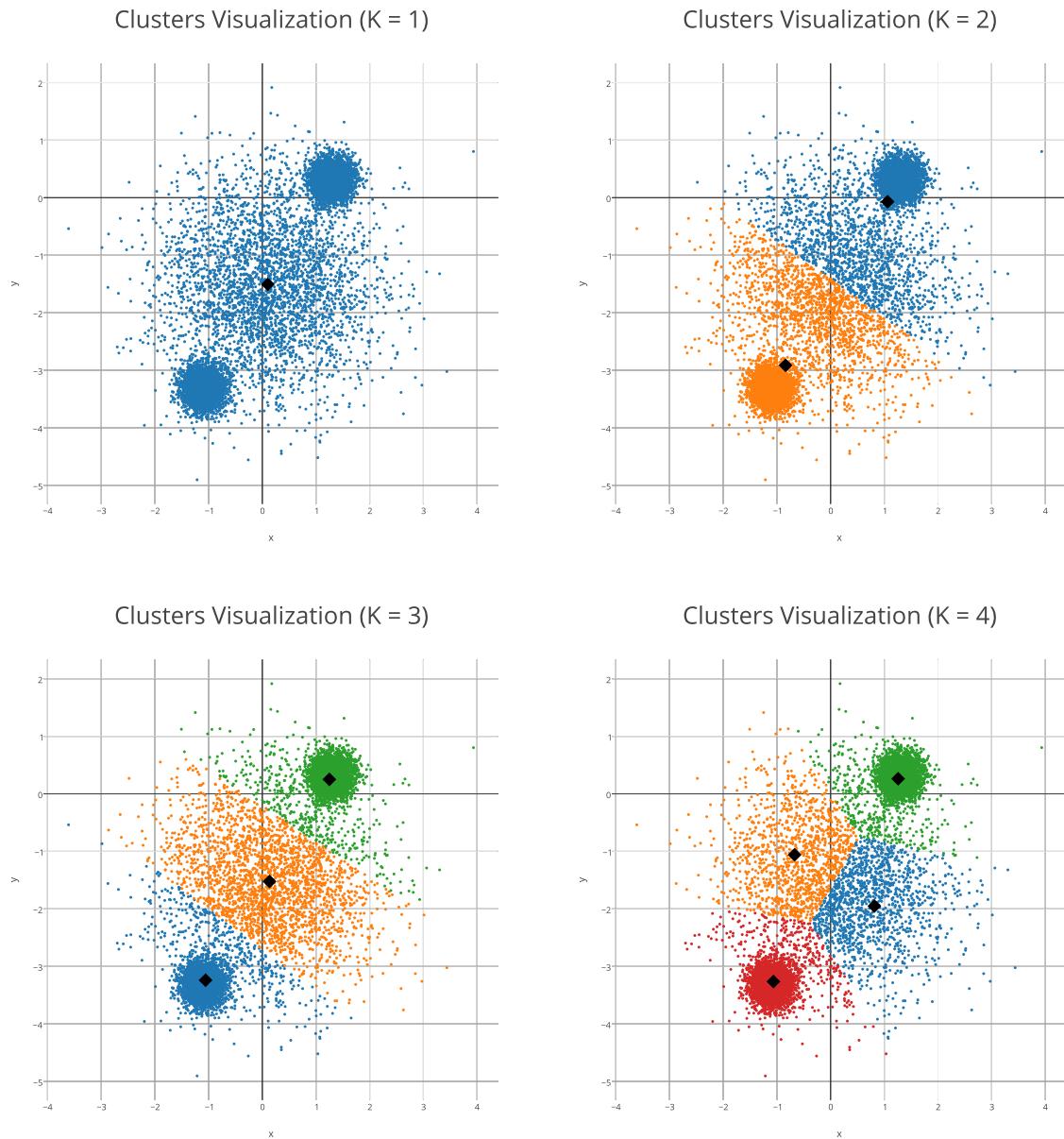


Figure 3: Assignment of data points to nearest K-means clusters, for $K = 1, 2, 3, 4$. For each run of K , data points are coloured to their corresponding cluster assignments: {
1, 2, 3, 4}. The black diamonds represent the cluster centers.

Clusters Visualization ($K = 5$)

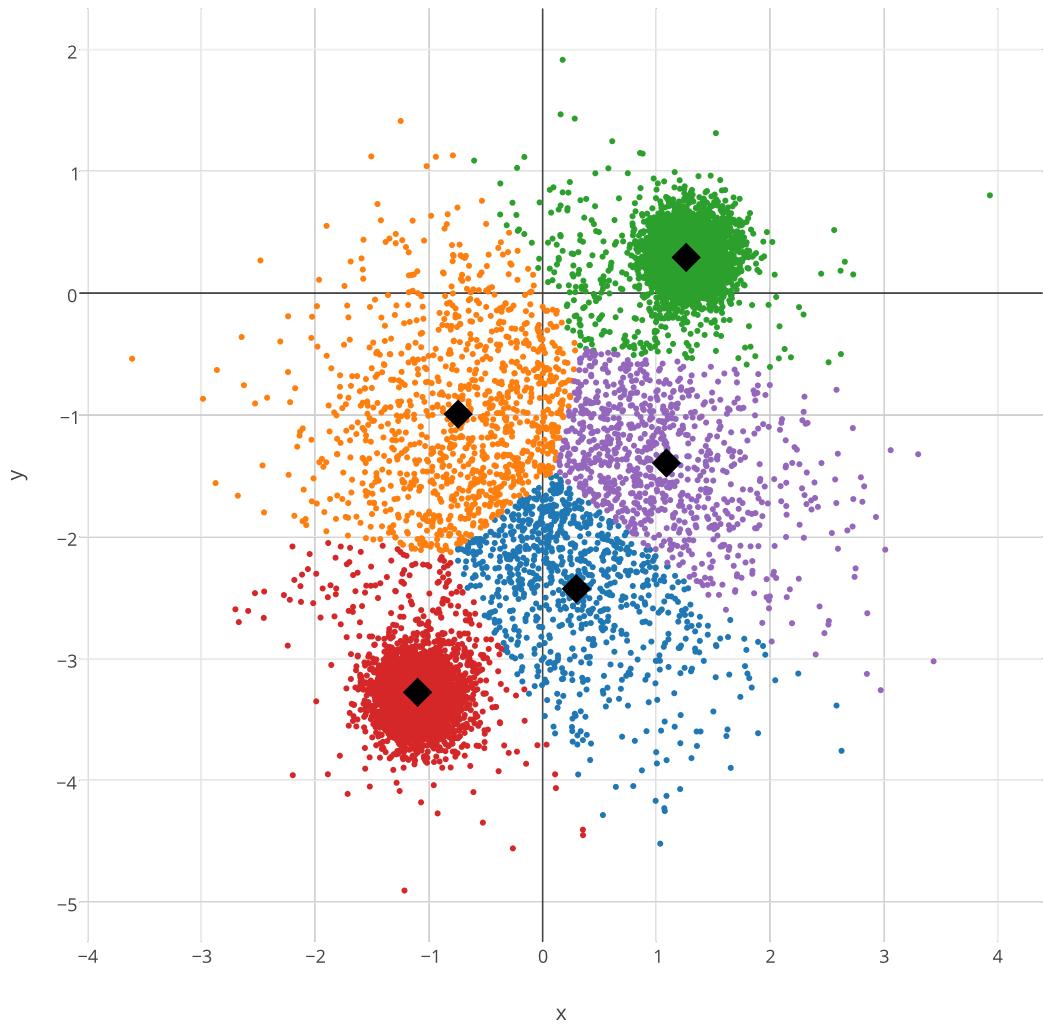


Figure 4: Assignment of data points to nearest K-means clusters, for $K = 5$. Data points are coloured to their corresponding cluster assignments: { 1, 2, 3, 4, 5 }. The black diamonds represent the cluster centers.

1.1.4 K-means with validation

Similarly, the training procedure from the previous section was repeated, with 1/3 of the data held out for validation.

The training results are summarised in Table 3. The percentage of data points assigned to each cluster has not changed significantly compared to the previous section. As such, one can refer to Figure 2 for the proportion of data points to each cluster.

From the table, $K = 5$ clusters seems to be the best, based on the lowest validation loss.

Table 3: K-means training results on *data2D.npy* with validation

K	Validation Loss	Cluster	Center	% Data Points
1	12877	1	(0.1, -1.5)	100
2	3140	1	(1.1, -0.1)	49.6
		2	(-0.8, -2.9)	50.4
3	1745	1	(-1.1, -3.2)	38.6
		2	(0.2, -1.5)	23.4
		3	(1.2, 0.2)	38.1
4	1133	1	(0.8, -2.0)	13.0
		2	(-0.6, -1.1)	12.2
		3	(1.3, 0.3)	37.3
		4	(-1.1, -3.3)	37.4
5	948	1	(0.3, -2.4)	8.9
		2	(-0.7, -1.0)	10.6
		3	(1.3, 0.3)	36.0
		4	(-1.1, -3.3)	36.1
		5	(1.1, -1.4)	8.5

2 Mixture of Gaussians (MoG)

2.1 The Gaussian cluster model

2.1.1 Deriving the latent variable posterior distribution for a data point, $P(z | \mathbf{x})$

Given a total of K clusters, let k be the k -th cluster the data points may assigned to: $k \in \{1, \dots, K\}$.

From Bayes' Rule,

$$P(z = k | \mathbf{x}) = \frac{P(\mathbf{x} | z = k)P(z = k)}{P(\mathbf{x})} \quad (7)$$

Given the prior $P(z = k) = \pi_k$, and the probability density function being a multivariate Gaussian distribution,

$$\begin{aligned} P(z = k | \mathbf{x}) &= \frac{\pi_k P(\mathbf{x} | z = k)}{\sum_{j=1}^K P(\mathbf{x} | z = j)P(z = j)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^K P(\mathbf{x} | z = j)P(z = j)} \end{aligned} \quad (8)$$

Assuming that all D dimensions are independent and of equal variance, $\boldsymbol{\Sigma}_i = \sigma_i^2 \mathbf{I}_D \in \mathbb{R}^{D \times D}, \forall i = 1, \dots, K$ where $\sigma_i^2 \in \mathbb{R}$.

$$P(z = k | \mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \sigma_k^2 \mathbf{I}_D)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_j, \sigma_j^2 \mathbf{I}_D)} \quad (9)$$

2.1.2 Computing the log probability density function, $\log \mathcal{N}(\mathbf{x} | \mu_k, \sigma_k^2 \mathbf{I}_D)$

Taking the log of the probability density function,

$$\begin{aligned}
\log \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \sigma_k^2 \mathbf{I}_D) &= \log \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ki}, \sigma_k^2) \\
&= \sum_{i=1}^D \log \mathcal{N}(x_i | \mu_{ki}, \sigma_k^2) \\
&= \sum_{i=1}^D \log \left[\frac{1}{\sqrt{2\pi\sigma_k^2}} \exp \left\{ -\frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} \right\} \right] \\
&= \sum_{i=1}^D \left[-\frac{1}{2} \log(2\pi\sigma_k^2) - \frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} \right] \\
&= -\sum_{i=1}^D \frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} - \frac{D}{2} \log(2\pi\sigma_k^2)
\end{aligned} \tag{10}$$

The Python code snippet for implementing the Equation 10 is attached.

```

1      """
2      Calculate log probability density function for all pairs of B data points
3          ← and K clusters
4
5      Assumptions:
6      Dimensions are independent and have the same standard deviation, sigma
7      Output:
8      log PDF function (N x K)
9      """
10
11     def calc_log_gaussian_cluster_k(X, Mu, sigma_sq):
12         with tf.name_scope('log_gaussian_cluster'):
13             # Infer dimension of data
14             D = tf.shape(X)[1]
15
16             # Calculate Mahalanobis distance
17             ### Expand dim(X) to (N x 1 x D)
18             ### Expand dim(Mu) to (1 x K x D)
19             ### Reduce sum along the D-axis
20             with tf.name_scope('mahalanobis_dist'):
21                 dist = - tf.divide(tf.reduce_sum(tf.square(tf.expand_dims(X,
22                             axis=1) - tf.expand_dims(Mu, axis=0))), axis=2), 2 *
23                             tf.transpose(sigma_sq), name='mahalanobis_dist')
24
25             # Calculate log of gaussian constant term
26             ### Transpose sigma_sq to (1 x K)
27             with tf.name_scope('log_gauss_const'):
28                 log_gauss_const = - tf.multiply(tf.cast(D, tf.float32) / 2,
29                     tf.log(2 * np.pi * tf.transpose(sigma_sq))),
30                     name='log_gauss_const')
31
32             # Sum results
33             log_gaussian_cluster = tf.add(dist, log_gauss_const,
34                 name='log_gaussian_cluster')
35
36         return log_gaussian_cluster

```

2.1.3 Computing the conditional responsibilities, $\gamma(x) = \log P(z | x)$

$$\begin{aligned}
& \log P(z | \mathbf{x}) \\
&= \log \frac{\pi_k \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ki}, \sigma_k^2)}{\sum_{i=1}^K \pi_j \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ji}, \sigma_j^2)} \\
&= \log \left[\pi_k \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ki}, \sigma_k^2) \right] - \log \left[\sum_{j=1}^K \pi_j \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ji}, \sigma_j^2) \right] \\
&= \log \pi_k + \log \prod_{i=1}^D \mathcal{N}(x_i | \mu_{ki}, \sigma_k^2) - \log \left\{ \sum_{j=1}^K \pi_j \frac{1}{(2\pi\sigma_j^2)^{\frac{D}{2}}} \prod_{i=1}^D \exp \left[-\frac{(x_i - \mu_{ji})^2}{2\sigma_j^2} \right] \right\} \\
&= \log \pi_k - \sum_{i=1}^D \frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} - \frac{D}{2} \log(2\pi\sigma_k^2) - \log \left\{ \sum_{j=1}^K \frac{\pi_j}{(2\pi\sigma_j^2)^{\frac{D}{2}}} \exp \left[-\sum_{i=1}^D \frac{(x_i - \mu_{ji})^2}{2\sigma_j^2} \right] \right\} \\
&= -\log \left\{ \sum_{i=1}^K \exp \left[\log \left(\frac{\pi_j}{(2\pi\sigma_j^2)^{\frac{D}{2}}} \right) - \sum_{i=1}^D \frac{(x_i - \mu_{ji})^2}{2\sigma_j^2} \right] \right\} - \sum_{i=1}^D \frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} - \frac{D}{2} \log(2\pi\sigma_k^2) + \log \pi_k \\
&= -\log \left\{ \sum_{i=1}^K \exp \left[-\sum_{i=1}^D \frac{(x_i - \mu_{ji})^2}{2\sigma_j^2} - \frac{D}{2} \log(2\pi\sigma_j^2) + \log \pi_j \right] \right\} \\
&\quad - \sum_{i=1}^D \frac{(x_i - \mu_{ki})^2}{2\sigma_k^2} - \frac{D}{2} \log(2\pi\sigma_k^2) + \log \pi_k
\end{aligned} \tag{11}$$

The Python code snippet for implementing the Equation 11 is attached.

```

13     # Return log normalised posterior / conditional responsibilities
14
15     with tf.name_scope('log_gamma_z'):
16         cond_resp = tf.add(
17             tf.reduce_logsumexp(unnormalised_log_posterior, axis=1,
18             keep_dims=True), unnormalised_log_posterior,
19             name='log_gamma_z')
20
21     return cond_resp

```

The function `tf.reduce_logsumexp` allows for more numerically stable computation. As described in TensorFlow r1.0 documentation, `tf.reduce_logsumexp` prevents both numerical overflow and underflow, from taking the exponent of large inputs and from taking the log of small inputs respectively. This is mitigated by first subtracting the maximum input to the function, evaluating the log-sum-exp expression before adding the maximum value back to obtain the final result. Also, we take the log as adding a bunch of log probabilities is more stable than multiplying a bunch of probabilities with values below 1.

2.2 Learning the MoG

2.2.1 Implicit cluster assignment variable

Proving $\nabla_{\boldsymbol{\mu}} \log P(\mathbf{x}) = \sum_{k=1}^K P(z = k | \mathbf{x}) \nabla_{\boldsymbol{\mu}} \log P(\mathbf{x}, z = k)$.

Starting from the Left-Hand Side (LHS):

$$\begin{aligned}
LHS &\equiv \nabla_{\boldsymbol{\mu}} \log P(\mathbf{x}) \\
&= \frac{1}{P(\mathbf{x})} \nabla_{\boldsymbol{\mu}} P(\mathbf{x}) \\
&= \frac{1}{P(\mathbf{x})} \nabla_{\boldsymbol{\mu}} \left[\sum_{k=1}^K P(\mathbf{x}, z = k) \right] \\
&= \frac{1}{P(\mathbf{x})} \sum_{k=1}^K \nabla_{\boldsymbol{\mu}} P(\mathbf{x}, z = k) \\
&= \sum_{k=1}^K \frac{P(z = k | \mathbf{x})}{P(z = k | \mathbf{x}) P(\mathbf{x})} \nabla_{\boldsymbol{\mu}} P(\mathbf{x}, z = k) \\
&= \sum_{k=1}^K P(z = k | \mathbf{x}) \left[\frac{1}{P(\mathbf{x}, z = k)} \nabla_{\boldsymbol{\mu}} P(\mathbf{x}, z = k) \right] \\
&= \sum_{k=1}^K P(z = k | \mathbf{x}) \nabla_{\boldsymbol{\mu}} \log P(\mathbf{x}, z = k) \\
&\equiv RHS
\end{aligned} \tag{12}$$

Hence, it is shown that the cluster assignment variable, $P(z = k | \mathbf{x})$, is implicit in the maximisation of the marginal log-likelihood, $\log P(\mathbf{X})$.

2.2.2 MoG without validation on *data2D.npy*

In this section, the data points from *data2D.npy* were clustered using Mixture of Gaussian with number of clusters, $K = 3$. The loss function to be minimized in this approach is the negative marginal log-likelihood:

$$\begin{aligned}
-\log P(\mathbf{X}) &= -\log \prod_{n=1}^B \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \sigma_k^2) \\
&= -\sum_{n=1}^B \log \sum_{k=1}^K \frac{\pi_k}{(2\pi\sigma_k^2)^{\frac{D}{2}}} \exp \left[-\frac{(\mathbf{x} - \boldsymbol{\mu}_k)^T(\mathbf{x} - \boldsymbol{\mu}_k)}{2\sigma_k^2} \right] \\
&= -\sum_{n=1}^B \log \sum_{k=1}^K \exp \left[-\frac{1}{2\sigma_k^2} (\mathbf{x} - \boldsymbol{\mu}_k)^T(\mathbf{x} - \boldsymbol{\mu}_k) - \frac{D}{2} \log(2\pi\sigma_k^2) + \log \pi_k \right]
\end{aligned} \tag{13}$$

where B is the number of data points.

Given the constraints $\sigma_k^2 \geq 0 \quad \forall k \in 1, \dots, K$ and $\sum_{k=1}^K \pi_k = 1$, σ_k^2 and π_k are generated from unconstrained parameters ϕ_k and ψ_k respectively as follow:

$$\sigma_k^2 = \exp(\phi_k) \tag{14}$$

$$\pi_k = \frac{\exp(\psi_k)}{\sum_{k'=1}^K \exp(\psi_{k'})} \tag{15}$$

In implementing the MoG training procedure, an Adam Optimizer was used. No validation data points were used — all data points were used to train the model. $-P(\mathbf{X})$ and $\log \pi_k$ were calculated using TensorFlow functions *tf.log_softmax* and *tf.reduce_logsumexp* respectively.

The parameters learnt by the MoG model are summarised in Table 4. A graph of the loss function over number of updates is also provided in Figure 5.

Table 4: Trained MoG parameters for $K = 3$ on *data2D.npy* without validation

Cluster k	Center, μ_k	Variance, σ_k^2	Latent prior, $P(z = k) = \pi_k$
1	(−1.1, −3.3)	0.039	0.33
2	(0.11, −1.5)	0.99	0.33
3	(1.3, 0.31)	0.039	0.33

The loss is converges around 17132 as shown in Figure 5.

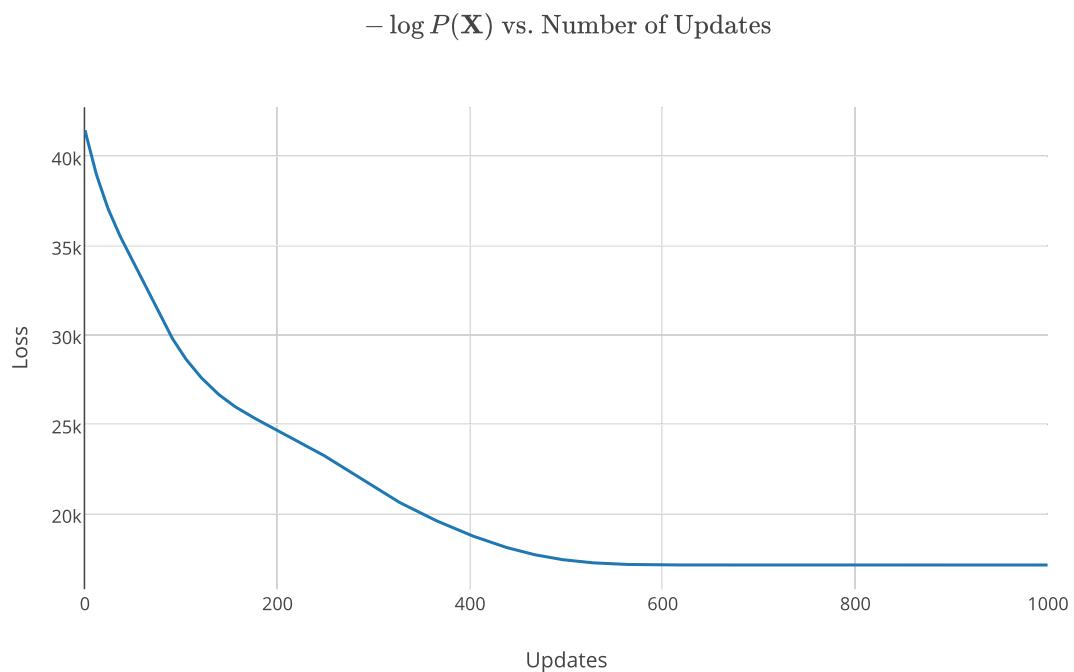


Figure 5: Loss function with respect to training updates.

2.2.3 MoG with validation

The training procedure from the previous section is repeated, this time with 1/3 of the *data2D.npy* data points held out for validation. MoG models for $K = 1, 2, 3, 4, 5$ were trained.

The training results are summarised in Table 5. A comparison of validation loss between MoG models and the percentage of data points assigned to the clusters in each MoG models are provided in Figures 6 and 7 respectively. Coloured scatters plots are provided in Figures 8 and 9, showing how data are assigned to each Gaussian cluster.

Table 5: MoG training results on *data2D.npy* with validation

K	Validation Loss	Cluster k	Center, μ_k	Variance, σ_k^2	Prior, π_k
1	11655	1	(0.10, -1.5)	1.92	1.00
2	8001	1	(-0.51, -2.45)	1.05	0.66
		2	(1.29, 0.30)	0.04	0.34
3	5743	1	(-1.10, -3.30)	0.04	0.34
		2	(0.13, -1.52)	0.98	0.33
		3	(1.30, 0.31)	0.04	0.33
4	5744	1	(0.20, -1.64)	0.96	0.16
		2	(-0.07, -1.41)	0.98	0.17
		3	(-1.10, -3.30)	0.04	0.34
		4	(1.30, 0.31)	0.04	0.33
5	5744	1	(-0.34, -1.96)	0.77	0.10
		2	(0.72, -1.82)	0.78	0.09
		3	(-1.10, -3.30)	0.04	0.34
		4	(1.30, 0.31)	0.04	0.33
		5	(0.07, -1.06)	0.87	0.15

MoG Model Performances on data2D.npy

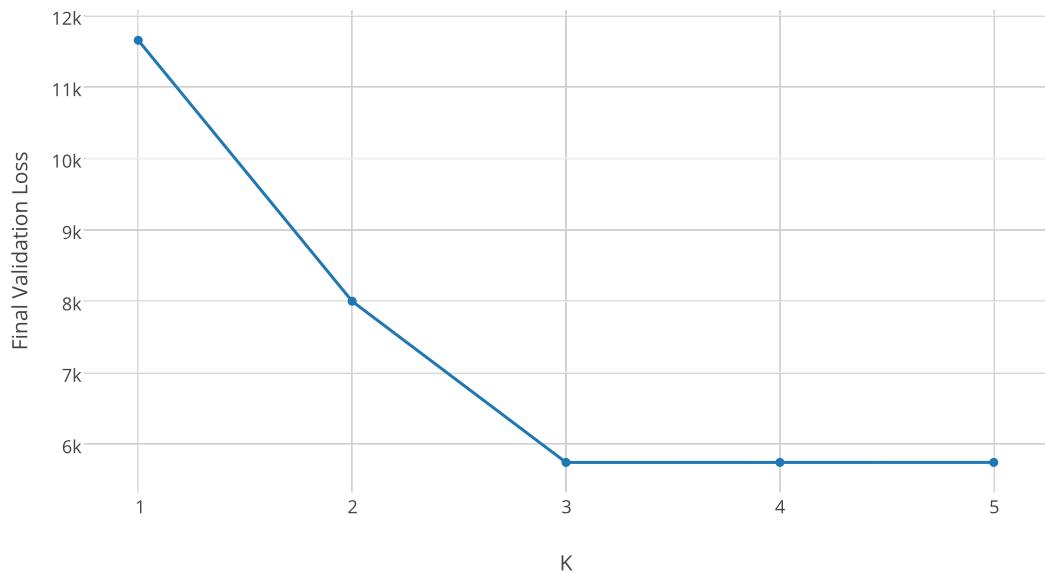


Figure 6: Comparison of model performance for various clusters ($K = 1, 2, 3, 4, 5$).

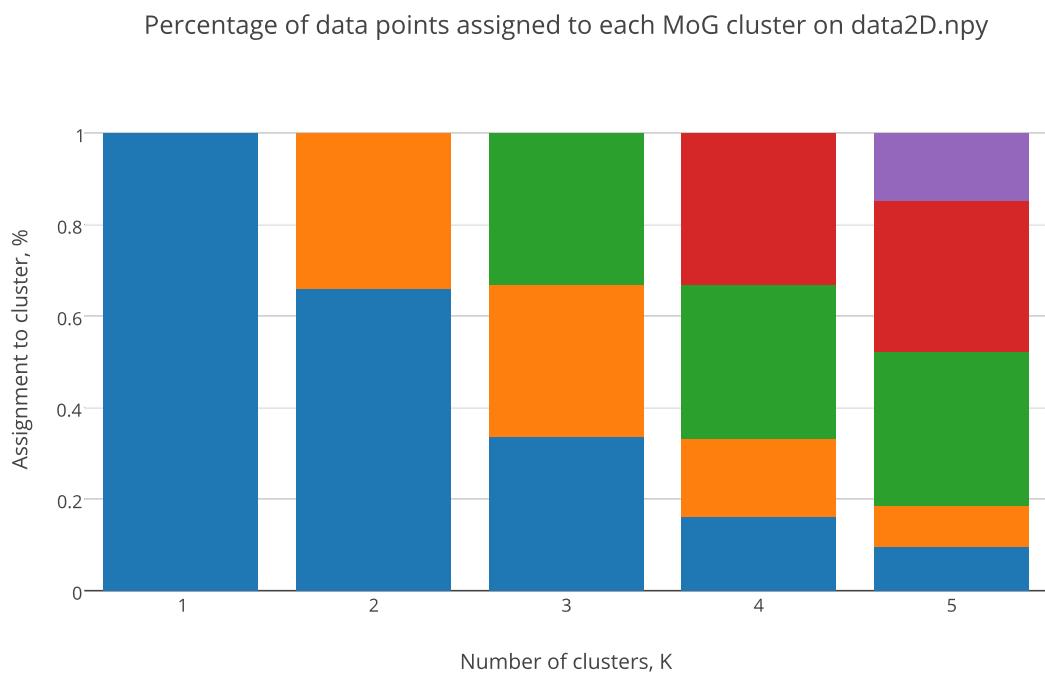
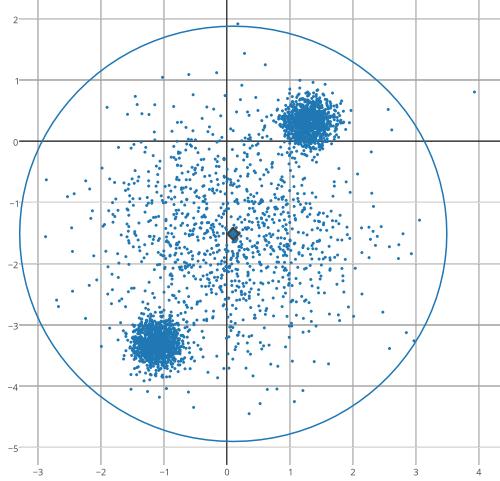
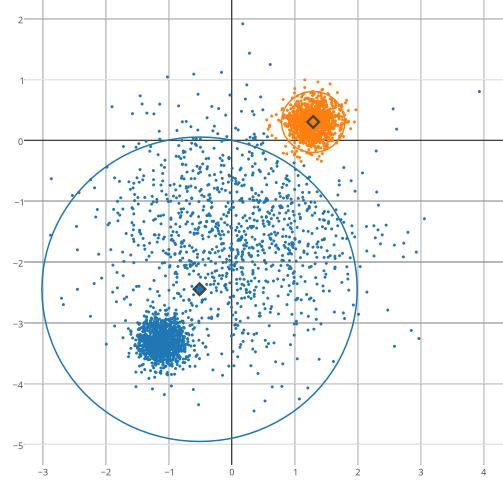


Figure 7: Loss function with respect to training updates.

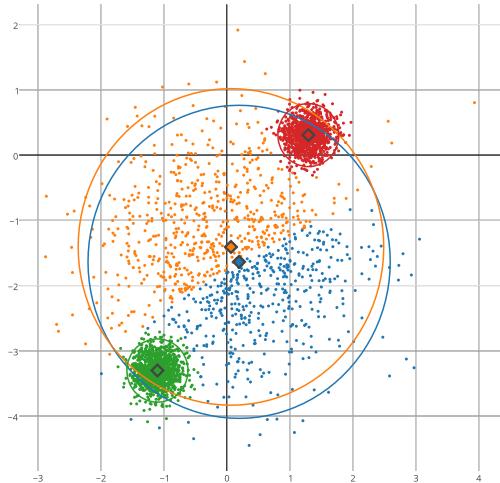
MoG Clustering Visualisation ($K = 1$)



MoG Clustering Visualisation ($K = 2$)



MoG Clustering Visualisation ($K = 4$)



MoG Clustering Visualisation ($K = 5$)

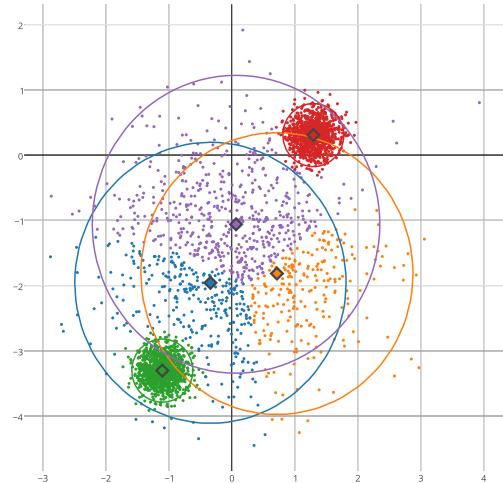


Figure 8: Assignment of validation data points to MoG clusters, for $K = 1, 2, 4, 5$. For each run of K , data points are coloured to their corresponding cluster assignments: { 1, 2, 3, 4, 5 }. The coloured diamonds represent the cluster centers, while the coloured circles represent 95% of the volume under the surfaces of the corresponding multivariate Gaussian distributions.

MoG Clustering Visualisation ($K = 3$)

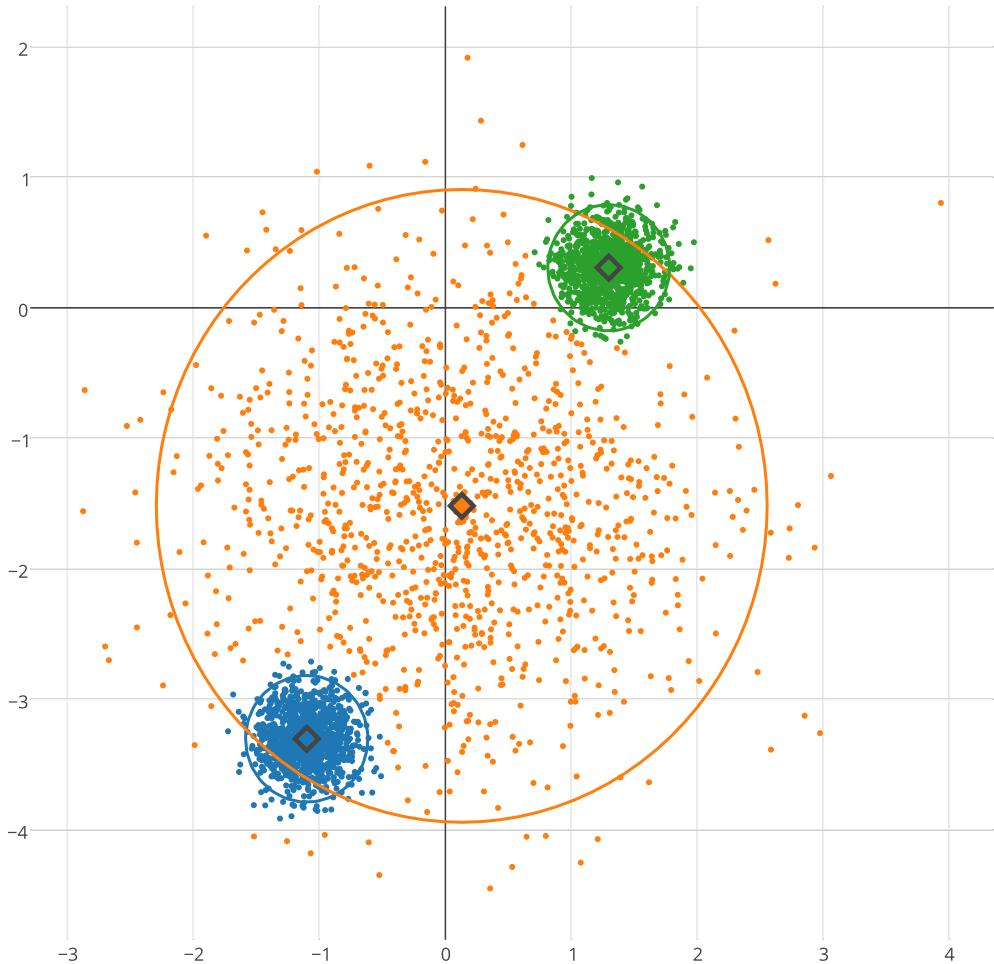


Figure 9: Assignment of validation data points to MoG clusters, for $K = 3$. Data points are coloured to their corresponding cluster assignments: { 1, 2, 3 }. The coloured diamonds represent the cluster centers, while the coloured circles represent 95% of the volume under the surfaces of the corresponding multivariate Gaussian distributions.

Based on the figures and the validation losses, it is clear that the best number of clusters, K , is 3. The scatter plots visually show the data points to be comprised of three clusters. Hence, it would make the most sense to assign 3 *soft* clusters for this clustering assignment. Similarly, from Table 5 and Figure 6, the loss plateaus from $K = 3$ onwards. As such, the most parsimonious model should be chosen to be $K = 3$.

2.2.4 K-means and MoG on *data100D.npy*

K-means and MoG were used to cluster a 100-dimensional data set, *data100D.npy*. Both these algorithms were run with $K = 1, \dots, 15$. The validation losses for both these clustering techniques were calculated and summarised in Table 6 and Figure 10.

Table 6: Validation loss (in thousands) of K-means and MoG on *data100.npy*

K	K-means Loss ('000s)	MoG Loss ('000s)	zazxx
	$\mathcal{L}(\mu)$	$-\log P(\mathbf{X})$	
1	337	475	
2	269	475	
3	220	475	
4	220	407	
5	152	367	
6	73	367	
7	73	367	
8	73	367	
9	73	367	
10	72	367	
11	72	367	
12	72	367	
13	72	290	
14	71	290	
15	72	290	

From Figure 10, the validation loss begins plateauing at $K = 6$ for K-means and $K = 5$ and 13 for MoG. In the interest of model simplicity and accuracy, any one of these three values are potentially suitable as to cluster the data.

To gain more intuition of the data set as in the coloured 2D scatter plots from Sections 1.1.2 and 2.2.3, the spatial distribution of *data100D.npy* data points was inspected by projecting the data points to 3 dimensions using the top 3 principal components, using Principal Component Analysis (PCA). The top 3 principal components explained 68.4% of the variance. These projections were done using TensorBoard's Embedding tab. Snapshots of the spatial distributions are shown in Figure 11.

The projected dataset indicate that there are 5 distinct clusters within the dataset. Using this intuition, and from the choices of $K = 5, 6$, or 13 from Figure 10, the best number of clusters is chosen to be $K = 5$.

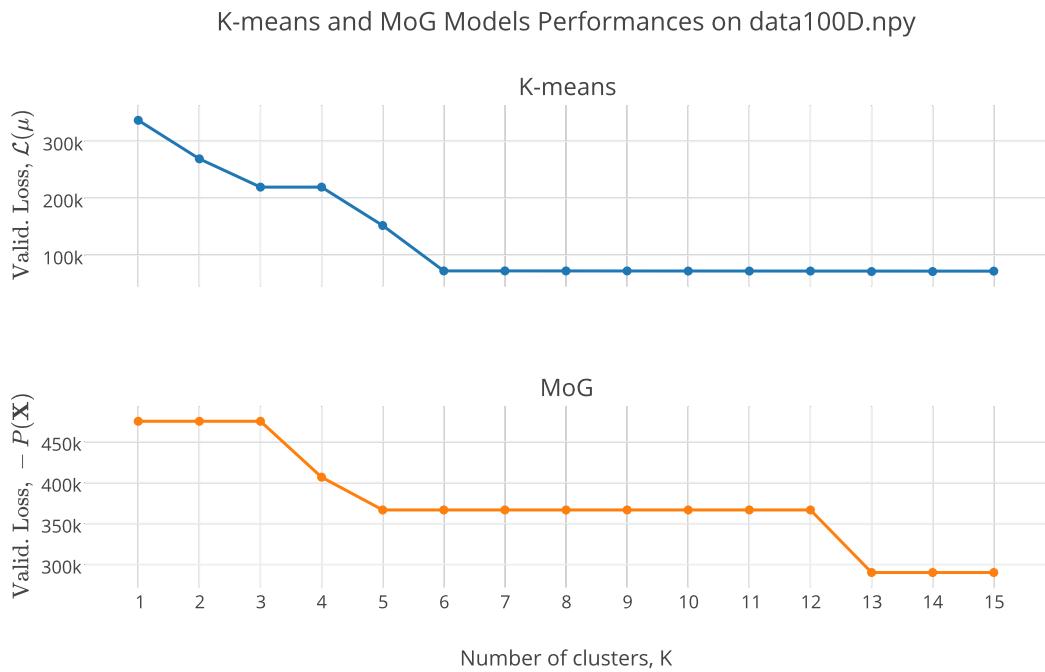


Figure 10: Comparison between K-means and MoG performance on *data100D.npy* for $K = 1, \dots, 15$.

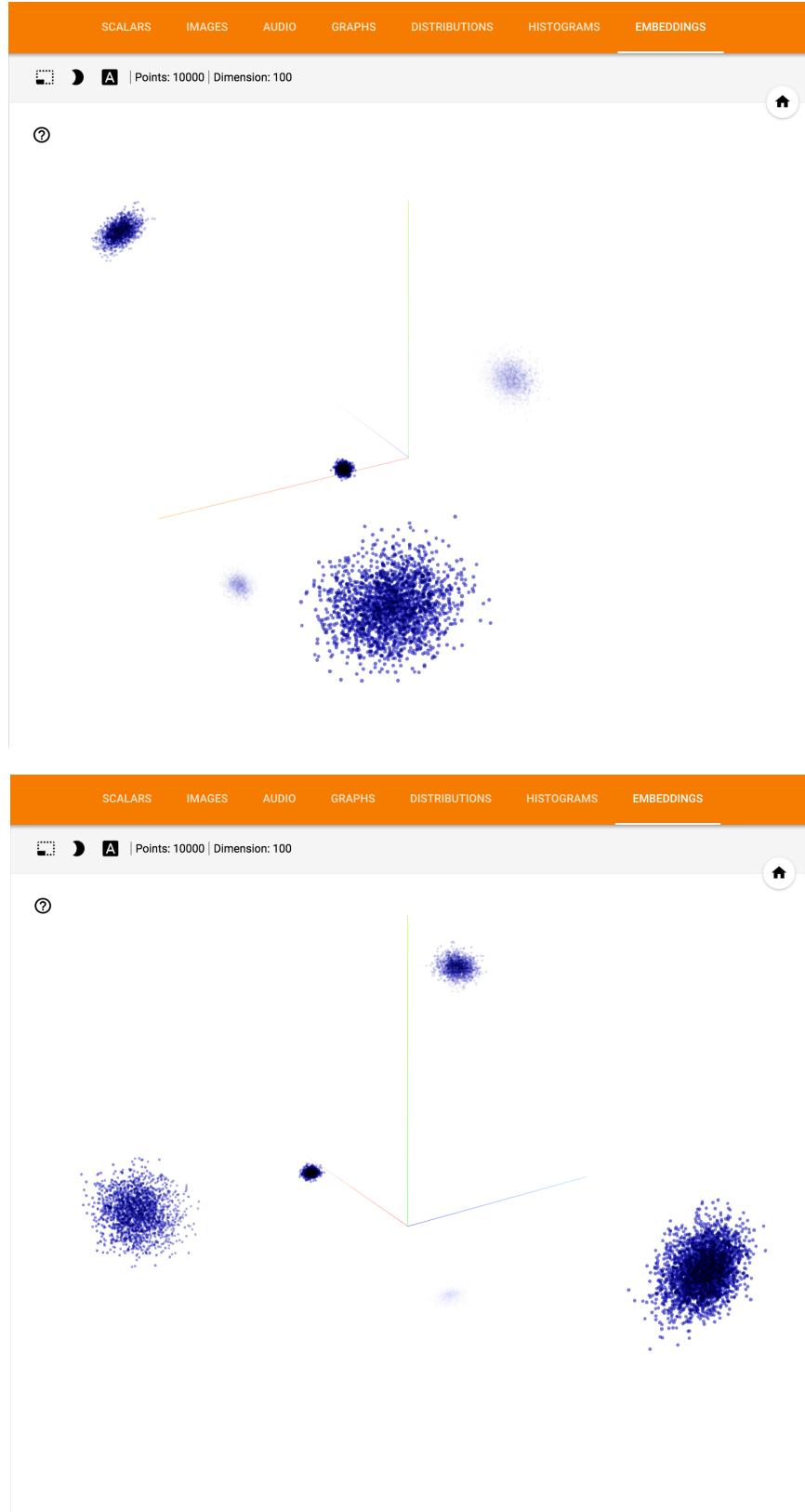


Figure 11: Projected 3D spatial distributions of *data100D.npy* data points using PCA from TensorBoard Embeddings.

3 Discover Latent Dimensions

3.1 Factor Analysis

3.1.1 Deriving the marginal log likelihood for a single training example

Given

$$P(\mathbf{s}_n) = \mathcal{N}(\mathbf{s}_n; \mathbf{0}, \mathbf{I}_K) \quad (16)$$

$$P(\mathbf{x}_n | \mathbf{s}_n) = \mathcal{N}(\mathbf{x}_n; \mathbf{W}\mathbf{s}_n + \boldsymbol{\mu}, \boldsymbol{\Psi}) \quad (17)$$

where

$$\mathbf{s}_n \in \mathbb{R}^K \quad (18)$$

$$\mathbf{x}_n, \boldsymbol{\mu} \in \mathbb{R}^D \quad (19)$$

$$\mathbf{W} \in \mathbb{R}^{D \times K} \quad (20)$$

$$\boldsymbol{\Psi} = \begin{bmatrix} \psi_1 & 0 & \cdots & 0 \\ 0 & \psi_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_k \end{bmatrix} \in \mathbb{R}^{D \times D} \quad (21)$$

Quoting Equation 3 from the handout of Assignment 3,

$$P(\mathbf{y}) = \mathcal{N}(\mathbf{y}; \mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\Lambda^{-1}\mathbf{A}^T) \quad (22)$$

for a single training example,

$$\begin{aligned} \log P(\mathbf{X}) &= \log P(\mathbf{x}) \\ &= \log \mathcal{N}(\mathbf{x}; \mathbf{W} \cdot \mathbf{0} + \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\mathbf{I}_K + \mathbf{W}^T) \\ &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Psi} + \mathbf{W}\mathbf{W}^T) \end{aligned} \quad (23)$$

Hence, it is proven that the marginal log likelihood of the factor analysis model also follows a Gaussian distribution.

3.1.2 Factor Analysis on *tinymnist.npz*

Factor Analysis is carried out on *tinymnist.npz* which contains 8x8 grayscale images of the digits 3 and 5, using $K = 4$. The parameters trained were Ψ and \mathbf{W} . μ was not trained as it is by definition the average of each feature in the design matrix, \mathbf{X} .

The parameters were trained by maximizing the marginal log-likelihood function $P(\mathbf{X}) = \sum_{n=1}^B P(\mathbf{x}_n)$. $\psi_i \quad \forall i = 1, \dots, D$ are constrained to be more than zero. Hence, ψ_i were initialised using unconstrained variable ϕ_i as such:

$$\psi_i = \exp(\phi_i) \quad (24)$$

Since the covariance matrix of the probability density function $(\Psi + \mathbf{W}\mathbf{W}^T)$ has a determinant not equal to 1, the matrix determinant is implemented in TensorFlow using Cholesky decomposition.

The marginal log likelihood post-training can be found in Table 7.

Table 7: Marginal log likelihoods of factor analysis model post-training on *tinymnist.npz*

Data	$\log P(\mathbf{X})$
Training	8453
Validation	1267
Test	4729

The each trained weights vector in the latent matrix, \mathbf{W} is visualised as 8x8 grayscale heatmap, as depicted in Figure 12. From the figure, Factor Analysis has learnt the following latent dimensions:

Table 8: Latent dimensions found from Factor Analysis of *tinyMNIST.npz*

Latent Dimensions	Comments
s_1	Captures the variability in the top-right and bottom-half corners of the image. Modelling the variability in the top-right corner is key in distinguishing between 3s and 5s.
s_2	Captures the variability in the top-left and lower-right of the image. This image has an S shape and can be used to identify 5s.
s_3	Captures the variability in the top-half of the image. The contrast between the black and white pixels help to identify the upper half of the digit 3.
s_4	Captures the variability in the bottom-half of the image.

Weights Visualisation of Latent Matrix, W

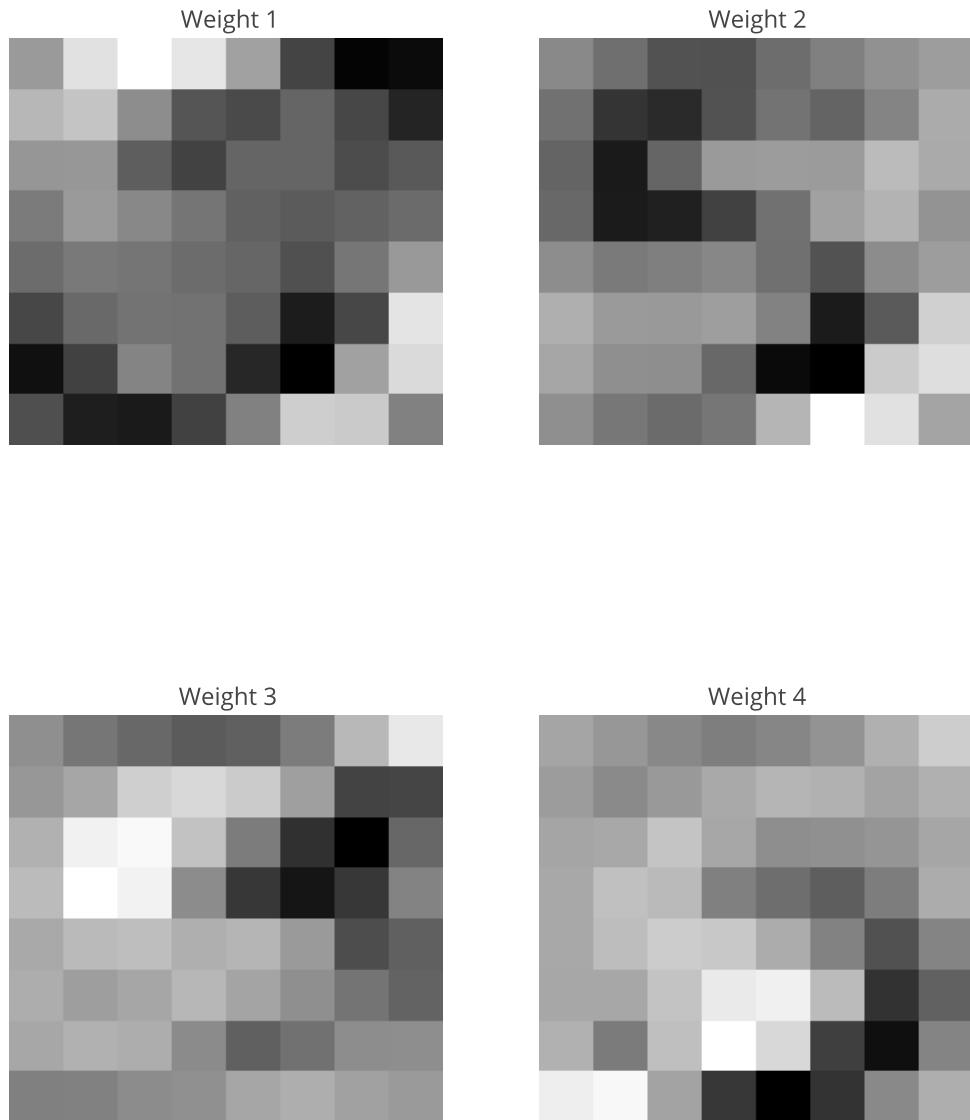


Figure 12: Visualisations of latent matrix, W , for the $K = 4$ latent variables.

3.1.3 Comparison between PCA and FA

A 3D latent variable, \mathbf{s} , was sampled from a multivariate Gaussian, $\mathcal{N}(\mathbf{0}, \mathbf{I}_3)$. The variable is then used to generate a data point, \mathbf{x} , using the following relations:

$$\begin{aligned} x_1 &= s_1 \\ x_2 &= s_1 + 0.001s_2 \\ x_3 &= 10s_3 \end{aligned} \tag{25}$$

This was repeated 200 times to generate a toy dataset consisting of 200 3D points.

For PCA, the largest principal component of this toy dataset was obtained by finding the eigenvector, \mathbf{e}_n , corresponding to the largest eigenvalue of the dataset covariance matrix, Σ .

$$\mathbf{e}_n \text{ where } n = \arg \max_i \lambda_i \quad \forall \lambda_i \in \sigma(\Sigma) \tag{26}$$

where $\sigma(\Sigma)$ is the spectrum of data covariance matrix.

The largest principal component was found to be $\mathbf{e}_n = \begin{bmatrix} 1.21 \times 10^{-6} \\ 1.37 \times 10^{-3} \\ 1.00 \times 10^0 \end{bmatrix}$.

From this result, it can be seen that the main principal component found using PCA is approximately in the x_3 direction:

$$\begin{aligned} \mathbf{e}_n^T \mathbf{x} &= x_3 + 1.212 \times 10^{-6}x_1 + 1.373 \times 10^{-3}x_2 \\ &\approx x_3 \end{aligned} \tag{27}$$

Meanwhile, assuming $K = 1$, Factor Analysis learns the direction of maximum corre-

lation ($x_1 + x_2$) as shown from the trained $\mathbf{W} = \begin{bmatrix} 0.4798 & 0.4616 & 3.145 \times 10^{-9} \end{bmatrix}$.

$$\begin{aligned}
\mathbf{W}\mathbf{x} &= 0.4798x_1 + 0.4616x_2 - 3.145 \times 10^{-9}x_3 \\
&\approx 0.4798x_1 + 0.4616x_2 \\
&= \begin{bmatrix} 0.4798 & 0.4616 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \\
&= \mathbf{k} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\end{aligned} \tag{28}$$

4 Appendices

Both of us implemented the assignment separately to maximise our learnings. The separate implementations can be found below. The graphs come from both separate implementations.

4.1 FuYuan Tee's Implementation

4.1.1 Base Code for K-means

```
1 # Import relevant packages
2 import tensorflow as tf
3 import numpy as np
4
5 import time
6 import datetime
7
8 # Non-interactive plotting
9 import matplotlib.pyplot as plt
10
11 # Interactive plotting
12 from plotly import tools
13 import plotly.plotly as py
14 import plotly.graph_objs as go
15 import plotly.grid_objs as gro
16 import plotly.offline as pyo
17 from plotly.offline import download_plotlyjs
18
19 # Configure environment
20 %config InlineBackend.figure_format = 'retina'
21 np.set_printoptions(precision=3)
22
23 # Global Variables
24 CURRENT_DIR =
    ↵ '/Users/christophertee/Dropbox/University/MASc/Courses/Winter 2017' +
    ↵ '\
```

```

25          '/ECE521 (Inference Algorithms & Machine
26              ↪ Learning) /Assignment 3'
27
28 # Activate Plotly Offline for Jupyter
29 pyo.init_notebook_mode(connected=True)
30
31 # Define global variable SEED
32 SEED = 521
33
34 # Load data into memory
35 """
36 data2D.npy contains 10,000 data points of dimension 2
37 data100D.npy contains 10,000 data points of dimension 100
38 """
39 # Load data
40 data2D = np.load("./Data/data2D.npy")
41 data100D = np.load("./Data/data100D.npy")
42
43 # Set random seed
44 np.random.seed(521)
45
46 # Generate random index
47 randIdx2D = np.arange(len(data2D))
48 randIdx100D = np.arange(len(data100D))
49
50 # Randomise data2D
51 np.random.shuffle(randIdx2D)
52 data2D = data2D[randIdx2D]
53
54 # Randomise data100D
55 np.random.shuffle(randIdx100D)
56 data100D = data100D[randIdx100D]
57
58 """
59 Creates a graph for K-means based on the loss function above:

```



```

93     dist = tf.reduce_sum(tf.square(tf.expand_dims(X, axis=2) -
94         ↪ tf.expand_dims(tf.transpose(mu), axis=0)), \
95             axis=1, name='distances')
96
97     # Create responsibility indices to track which datapoint belongs
98     ↪ to which cluster
99
100    with tf.name_scope('responsibility'):
101        _, resp = tf.nn.top_k(-dist, name='responsibility_indices')
102        resp = tf.cast(resp + 1, tf.int64)
103
104    # Calculate loss
105
106    with tf.name_scope('loss'):
107        loss = tf.reduce_sum(tf.reduce_min(dist, axis=1), name='loss')
108        tf.summary.scalar('loss', loss)
109
110    # Create Adam optimizer
111
112    with tf.name_scope('optimizer'):
113        optimizer = tf.train.AdamOptimizer(learning_rate=0.01,
114            ↪ beta1=0.9, beta2=0.99, epsilon=1e-5).minimize(loss)
115
116    """
117    Run k-means clustering algorithm to cluster datapoints
118    """
119    def run_k_means(K_list, data_dim, has_valid=False, device='cpu'):
120        """
121        If has_valid is true, subsets:
122            first 2/3 of data as training data
123            remaining 1/3 of data as validation data
124        """
125        def subset_data(D):

```

```

126     if D == 2:
127         data = data2D
128     elif D == 100:
129         data = data100D
130     divider = data.shape[0] * 2 / 3
131     return data[:divider], data[divider:]
132
133     """
134     Calculate percentage of points belonging to each of K clusters
135     """
136     def calculate_composition(K, resp_idx):
137         composition = np.array([])
138         for k in range(K):
139             composition = np.append(composition,
140                                     np.true_divide(np.sum(resp_idx == k + 1),
141                                     resp_idx.shape[0]))
140         return composition
141
142 #####
143 ## Function begins ##
144 #####
145 """
146 cluster_centres: 11 x K x D
147 resp_idx:          N x 11
148 """
149
150 # Assert correct value for data_dim
151 assert data_dim == 2 or data_dim == 100
152
153 # Define locally global function
154 MAX_ITER = 500
155 CURR_TIME = '{:b%d %H_%M_%S}'.format(datetime.datetime.now())
156 SUMMARY_DIR = CURRENT_DIR + LOG_DIR + '/K-means/' + CURR_TIME
157
158 # Create list to store run results
159 results = []

```

```

160
161     for K in K_list:
162         # Clear any pre-defined graph
163         tf.reset_default_graph()
164
165         # Build TensorFlow graph
166         X, mu, resp, loss, optimizer, merged = build_k_means(K, data_dim,
167             ↳ device)
168
169         # Select appropriate input_data
170         if has_valid:
171             input_data, valid_data = subset_data(data_dim)
172         else:
173             input_data = data2D if D == 2 else data100D
174
175         # Create arrays to log session losses, cluster_centres and
176         → responsibility indices
177         train_loss = np.array([])[:, np.newaxis]
178         if has_valid:
179             valid_loss = np.array([])[:, np.newaxis]
180             cluster_centres = np.array([])[:, np.newaxis,
181                 ↳ np.newaxis].reshape(0, K, data_dim)
182             resp_idx = np.array([])[:, ,
183                 ↳ np.newaxis].reshape(input_data.shape[0], 0)
184
185         # Begin session
186         with tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
187             ↳ log_device_placement=False)) as sess:
188             # Log start time
189             start_time = time.time()
190
191             # Create sub-directory title
192             sub_dir = '/K={},dim={},valid={}'.format(K, data_dim,
193                 ↳ has_valid)
194
195             # Create summary writers

```

```

190     train_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir +
191                                         ↵  '/train', graph=sess.graph)
192
193     if has_valid:
194
195         valid_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir
196                                         ↵  + '/valid')
197
198
199     # Initialise all TensorFlow variables
200     tf.global_variables_initializer().run()
201
202
203     # Define iterator
204     currIter = 0
205
206
207     # Calculate training (and validation) loss,
208     # cluster centres and responsibility indices before any
209     # training
210
211     err, summaries, clusters, indices = sess.run([loss, merged,
212                                                 ↵  mu, resp], feed_dict={X:input_data})
213
214     train_loss = np.append(train_loss, err)
215
216     train_writer.add_summary(summaries, currIter)
217
218
219     if has_valid:
220
221         err, summaries = sess.run([loss, merged],
222                               ↵  feed_dict={X:valid_data})
223
224         valid_loss = np.append(valid_loss, err)
225
226         valid_writer.add_summary(summaries, currIter)
227
228
229     if data_dim == 2:
230
231         cluster_centres = np.append(cluster_centres,
232                                     ↵  clusters[np.newaxis, :, :], axis=0)
233
234         resp_idx = np.append(resp_idx, indices, axis=1)
235
236
237     # Begin training
238
239     while currIter < MAX_ITER:
240
241         # Train graph
242
243         _, err, summaries = sess.run([optimizer, loss, merged],
244                                     ↵  feed_dict={X:input_data})

```

```

219
220     # Add training loss
221     train_writer.add_summary(summaries, currIter + 1)
222     train_loss = np.append(train_loss, err)
223
224     # Log validation loss
225     if has_valid:
226         err, summaries = sess.run([loss, merged],
227             ↪ feed_dict={X:valid_data})
228         valid_loss = np.append(valid_loss, err)
229         valid_writer.add_summary(summaries, currIter)
230
231     # Log responsibility indices and cluster centres every 10%
232     # of maximum iteration
233     if ((float(currIter) + 1) * 100 / MAX_ITER) % 10 == 0:
234         clusters, indices = sess.run([mu, resp],
235             ↪ feed_dict={X:input_data})
236
237         cluster_centres = np.append(cluster_centres,
238             ↪ clusters[np.newaxis, :, :], axis=0)
239         resp_idx = np.append(resp_idx, indices, axis=1)
240
241
242     # Post training progress to user, every 100 iterations
243     if currIter % 100 == 99:
244         if not has_valid:
245             print 'iter: {:3d}, train_loss: '
246                 ↪ {:.3lf}'.format(currIter,
247                     ↪ train_loss[currIter])
248         else:
249             print 'iter: {:3d}, train_loss: {:.3lf}, '
250                 ↪ valid_loss: {:.3lf}' \
251                     .format(currIter + 1,
252                         ↪ train_loss[currIter],
253                         ↪ valid_loss[currIter])
254
255         currIter += 1

```

```

246
247     # End of while loop
248     print 'Max iteration reached'
249     train_writer.close()
250
251     if has_valid:
252         valid_writer.close()
253
254     # Calculate composition of points belonging to each cluster
255     composition = calculate_composition(K, resp_idx[:, -1])
256
257     if not has_valid:
258         results.append(
259             {
260                 'K': K,
261                 'train_loss': train_loss,
262                 'cluster_centres': cluster_centres,
263                 'responsibility_indices': resp_idx.astype(int),
264                 'composition': composition,
265                 'time_of_run': '{:%b%d
266                               ↵ %H_%M_%S}'.format(datetime.datetime.now())
267             }
268         )
269     else:
270         results.append(
271             {
272                 'K': K,
273                 'train_loss': train_loss,
274                 'valid_loss': valid_loss,
275                 'cluster_centres': cluster_centres,
276                 'responsibility_indices': resp_idx.astype(int),
277                 'composition': composition,
278                 'time_of_run': '{:%b%d
279                               ↵ %H_%M_%S}'.format(datetime.datetime.now())
280             }
281         )

```

```

280     # TODO calculate convergence
281
282     if not has_valid:
283         print 'K: {:3d}, train loss: {:.1f}, duration:
284             {:.1f}s\n' \
285             .format(K, train_loss[-1], time.time() -
286                     start_time)
287
288     else:
289         print 'K: {:3d}, train loss: {:.1f}, valid loss: {:.1f},
290             duration: {:.1f}s\n' \
291             .format(K, train_loss[-1], valid_loss[-1],
292                     time.time() - start_time)
293
294
295     print 'RUN COMPLETED'
296
297     return results

```

4.1.2 K-means - Q1.1.2

```

1 # Run K-means
2 results_1_1_2 = run_k_means(K_list=[3], data_dim=2)

```

4.1.3 K-means - Q1.1.3

```
1 # Run K-means
2 results_1_1_3 = run_k_means(K_list=[1, 2, 3, 4, 5], data_dim=2)
3
4 # Save results
5 np.save('./Results/K-means/1_1_3.npy', results_1_1_3)
6
7 # Plot losses
8 def loss_IGraph(loss):
9     # Define data to plot
10    trace = go.Scatter(
11        x = range(loss.shape[0]),
12        y = loss
13    )
14    data = go.Data([trace])
15
16    # Define layout
17    layout = go.Layout(
18        title = '$\\mathcal{L} (\\mathbf{\\mu})$ vs. Number of
19                  Updates',
20        xaxis = {'title': 'Updates'},
21        yaxis = {'title': 'Loss'}
22    )
23
24    # Define figure
25    figure = go.Figure(data=data, layout=layout)
26
27    # Generate plot
28    py.iplot(figure, filename='/ECE521: A3/Q1: K-means/Q1.2_k_means_loss',
29              sharing='private')
30    return pyo.iplot(figure)
31
32 # Generate loss function graph
33 figure = loss_IGraph(results_1_1_3[2]['train_loss'])
```

```

33 # Cluster Visualisation
34 """
35 Colour data points by clusters generated by K-means algorithm
36 Input:
37     cluster_centres: coordinates of cluster centres (K x D)
38     resp_idxes:      final responsibility indices for each run of K (N x
39     ↪ num_subplots)
40 """
41
42 def visualise_k_means_clusters(result):
43     # Store cluster centres and responsibility indices
44
45     cluster_centres = result['cluster_centres'][-1,:,:]
46     resp_idx = result['responsibility_indices'][:, -1]
47
48     # Define colour list as per Plotly's default colour list
49     colour_list = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
50     ↪ '#8c564b']
51
52     # Create traces for each cluster
53     traces = []
54     for k in range(np.amax(resp_idx)):
55         # Create trace for data points in cluster k
56         traces.append(go.Scatter(
57             x = data2D[resp_idx == k + 1][:,0],
58             y = data2D[resp_idx == k + 1][:,1],
59             hoverinfo = 'none',
60             mode = 'markers',
61             marker = {
62                 'size': 4,
63                 'color': colour_list[k],
64             }
65         ))
66
67         # Create trace for cluster centre k
68         traces.append(go.Scatter(
69             x = [cluster_centres[k][0]],

```

```

67     y = [cluster_centres[k][1]],
68     name = 'Cluster {}'.format(k + 1),
69     mode = 'markers',
70     marker = {
71         'size': 15,
72         'symbol': 'diamond',
73         'color': '#000000'
74     }
75 )
76
77 # Add traces
78 traces = go.Data(traces)
79
80 # Generate figure layout
81 layout = go.Layout(
82     height = 800,
83     showlegend = False,
84     title = 'Clusters Visualization (K = {})'.format(result['K']),
85     xaxis = {'title': 'x'},
86     yaxis = {'title': 'y'}
87 )
88
89 # Generate figure
90 figure = go.Figure(data=traces, layout=layout)
91
92 py.iplot(figure, filename='/ECE521: A3/Q1:
93     ↳ K-means/Q1.3_cluster_viz_K={}'.format(result['K']),
94     ↳ sharing='private')
95
96 return pyo.iplot(figure)

```

4.1.4 K-means - Q1.1.4

```
1 # Run K-means
2 results_1_1_4 = run_k_means(K_list=[1, 2, 3, 4, 5], data_dim=2,
3     ↪ has_valid=True)
4
5 # Save results
6 np.save('./Results/K-means/1_1_4.npy', results_1_1_4)
7
8 # Generate bar chart
9 """
10 Generate a bar chart for each model showing percentage of data points
11     ↪ belong to each cluster
12 """
13
14 def cluster_assignment_IGraph(results):
15     # Define colour list as per Plotly's default colour list
16     colour_list = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
17         ↪ '#8c564b']
18
19     # Define empty figure
20     figure = {
21         'data': [],
22         'layout': {}
23     }
24
25     # Define data to plot
26     for i, result in enumerate(results):
27         for k in range(result['K']):
28             trace = go.Bar(
29                 x = [i + 1],
30                 y = [result['composition'][k]],
31                 marker = {'color': colour_list[k]},
32                 name = 'Cluster {}'.format(k + 1)
33             )
34             figure['data'].append(trace)
```

```

32 # Define layout
33 figure['layout'] = {
34     'title': 'Percentage of data points assigned to each cluster',
35     'xaxis': {'title': 'Number of clusters, K'},
36     'yaxis': {'title': 'Assignment to cluster, %'},
37     'barmode': 'stack',
38     'showlegend': False
39 }
40
41 # Generate plot
42 py.iplot(figure, filename='/ECE521: A3/Q1:
43     ↳ K-means/Q1.3_assignment_bar_chart', sharing='private')
44
45 # Generate loss function graph
46 figure = cluster_assignment_IGraph(results_1_1_3)

```

4.1.5 Base Code for MoG

```
1 # Import relevant packages
2 import tensorflow as tf
3 import numpy as np
4
5 import time
6 import datetime
7
8 # Non-interactive plotting
9 import matplotlib.pyplot as plt
10
11 # Interactive plotting
12 from plotly import tools
13 import plotly.plotly as py
14 import plotly.graph_objs as go
15 import plotly.grid_objs as gro
16 import plotly.offline as pyo
17 from plotly.offline import download_plotlyjs
18
19 # Configure environment
20 %config InlineBackend.figure_format = 'retina'
21 np.set_printoptions(precision=3)
22
23 # Global Variables
24 CURRENT_DIR =
25     ↪ '/Users/christophertee/Dropbox/University/MASc/Courses/Winter 2017' +
26     ↪ \
27         '/ECE521 (Inference Algorithms & Machine
28             ↪ Learning) /Assignment 3'
29 LOG_DIR = '/Logs'
30
31 # Activate Plotly Offline for Jupyter
32 pyo.init_notebook_mode(connected=True)
33
34 # Define global variable SEED
```

```

32 SEED = 521
33
34 # Load data into memory
35 """
36 data2D.npy contains 10,000 data points of dimension 2
37 data100D.npy contains 10,000 data points of dimension 100
38 """
39 # Load data
40 data2D = np.load("./Data/data2D.npy")
41 data100D = np.load("./Data/data100D.npy")
42
43 # Set random seed
44 np.random.seed(521)
45
46 # Generate random index
47 randIdx2D = np.arange(len(data2D))
48 randIdx100D = np.arange(len(data100D))
49
50 # Randomise data2D
51 np.random.shuffle(randIdx2D)
52 data2D = data2D[randIdx2D]
53
54 # Randomise data100D
55 np.random.shuffle(randIdx100D)
56 data100D = data100D[randIdx100D]
57
58 """
59 Builds TensorFlow graph for MoG
60
61 Input:
62     K: number of clusters
63     D: dimension of data (only 2 or 100 allowed)
64 Internal variables:
65     X: input data matrix (N x D)
66     Mu: cluster centres (K x D)
67     sigma_sq: cluster variance (K x 1)

```

```

68     log_pi: log of latent cluster variables (K x 1)
69
70     """
71
72     def build_MoG(K, D, device='cpu'):
73         """
74             Calculate log probability density function for all pairs of B data
75             points and K clusters
76
77             Assumptions:
78                 Dimensions are independent and have the same standard deviation,
79                 → sigma
80
81             Output:
82                 log PDF function (N x K)
83
84             """
85
86     def calc_log_gaussian_cluster_k(X, Mu, sigma_sq):
87
88         with tf.name_scope('log_gaussian_cluster'):
89
90             # Infer dimension of data
91             D = tf.shape(X)[1]
92
93
94             # Calculate Mahalanobis distance
95             ### Expand dim(X) to (N x 1 x D)
96             ### Expand dim(Mu) to (1 x K x D)
97             ### Reduce sum along the D-axis
98
99             with tf.name_scope('mahalanobis_dist'):
100
101                 dist = -
102
103                     → tf.divide(tf.reduce_sum(tf.square(tf.expand_dims(X,
104                         → axis=1) - tf.expand_dims(Mu, axis=0))), axis=2), 2 *
105                         → tf.transpose(sigma_sq), name='mahalanobis_dist')
106
107
108                 # Calculate log of gaussian constant term
109                 ### Transpose sigma_sq to (1 x K)
110
111                 with tf.name_scope('log_gauss_const'):
112
113                     log_gauss_const = - tf.multiply(tf.cast(D, tf.float32) /
114                         → 2, tf.log(2 * np.pi * tf.transpose(sigma_sq))),
115                         → name='log_gauss_const')
116
117
118                 # Sum results

```

```

97     log_gaussian_cluster = tf.add(dist, log_gauss_const,
98                                     ← name='log_gauss_cluster')
99
100
101    '''
102
103    Calculate log probability cluster variable z given x, a.k.a.
104    ← conditional responsibilities, gamma
105
106    Output:
107    conditional responsibilities (N x K)
108
109    '''
110
111    def calc_log_conditional_responsibilities(X, Mu, sigma_sq, log_pi):
112        with tf.name_scope('log_conditional_responsibilities'):
113            # Calculate unnormalised_log_posterior P(z/x)
114            with tf.name_scope('unnormalised_log_posterior'):
115                unnormalised_log_posterior =
116                    calc_log_gaussian_cluster_k(X, Mu, sigma_sq) +
117                    tf.transpose(log_pi)
118
119                # Return log normalised posterior / conditional
120                ← responsibilities
121                with tf.name_scope('log_gamma_z'):
122                    cond_resp = tf.add(-
123                        tf.reduce_logsumexp(unnormalised_log_posterior,
124                            ← axis=1, keep_dims=True), \
125                            unnormalised_log_posterior, \
126                            name='log_gamma_z')
127
128        return cond_resp
129
130    '''
131
132    Calculates the negative log marginal probability, -log P(X), aka the
133    ← loss function for MoG
134
135    Output:
136        - log P(X) (scalar)

```

```

125
126     '''  

127     def calc_neg_log_marg_prob(X, Mu, sigma_sq, log_pi):  

128         with tf.name_scope('loss'):  

129             loss =  

130                 → tf.negative(tf.reduce_sum(tf.reduce_logsumexp(calc_log_gaussian_cluste  

131
132             '''  

133             Helper function to add histogram tag to variables  

134             Input:  

135                 var: variable to be tagged with histogram summary  

136
137             def _add_histogram(vars_):  

138                 for var in vars_:  

139                     tf.summary.histogram(var.op.name, var)  

140
141             #####  

142             ## Function begins ##  

143             #####  

144             # Fix TF graph seed  

145             tf.set_random_seed(SEED)  

146
147             # Define computation device  

148             try:  

149                 assert device == 'cpu' or device == 'gpu'  

150             except AssertionError:  

151                 print 'Invalid device chosen. Please use \'cpu\' or \'gpu\''
152                 quit()  

153             device = '/' + device + ':0'  

154
155             with tf.device('/cpu:0'):  

156                 # Define placeholder  

157                 with tf.name_scope('placeholder'):  


```

```

158     X = tf.placeholder(tf.float32, shape=[None, D], name='inputs')
159
160     # Define parameters
161
162     with tf.variable_scope('parameters'):
163         Mu = tf.get_variable('cluster_centres', shape=[K, D],
164                             initializer=tf.truncated_normal_initializer(seed=SEED))
165         phi = tf.get_variable('latent_for_sigma_sq', shape=[K, 1],
166                             initializer=tf.truncated_normal_initializer(seed=SEED))
167         psi = tf.get_variable('latent_for_pi', shape=[K, 1],
168                             initializer=tf.truncated_normal_initializer(seed=SEED +
169                                         1))
170
171
172     with tf.name_scope('log_pi'):
173         log_pi =
174             tf.transpose(tf.nn.log_softmax(tf.transpose(psi)),
175                         name='log_pi')
176
177     with tf.device(device):
178
179         # Calculate conditional responsibilities
180
181         log_resp = calc_log_conditional_responsibilities(X, Mu, sigma_sq,
182                                         log_pi)
183
184         # Calculate loss
185
186         loss = calc_neg_log_marg_prob(X, Mu, sigma_sq, log_pi)
187         tf.summary.scalar('loss', loss)
188
189         # Define optimizer
190
191         optimizer = tf.train.AdamOptimizer(learning_rate=0.01, \
192                                         beta1=0.9, beta2=0.99,
193                                         epsilon=1e-5).minimize(loss)
194
195     with tf.device('/cpu:0'):
196
197         # Add histogram summaries for variables of interest

```

```

186     _add_histogram([Mu, phi, psi, sigma_sq, log_pi, log_resp])
187
188     # Merge all summaries
189     merged = tf.summary.merge_all()
190
191     return X, Mu, sigma_sq, log_pi, log_resp, loss, optimizer, merged
192
193     """
194     Runs MoG training algorithm more efficiently by not saving loss values.
195     Tensorboard embedding enabled
196     """
197
198     def run_MoG_v2(K_list, D, QUES_DIR, has_valid=False, device='cpu'):
199
200         """
201         If has_valid is true, subsets:
202             first 2/3 of data as training data
203             remaining 1/3 of data as validation data
204         """
205
206         def subset_data(D):
207
208             if D == 2:
209                 data = data2D
210
211             elif D == 100:
212                 data = data100D
213
214             divider = data.shape[0] * 2 / 3
215
216             return data[:divider], data[divider:]
217
218
219         """
220         Embed data for visualization purposes
221         """
222
223         def embed_data(D, train_writer):
224
225             # Define input data
226
227             input_data = data2D if D == 2 else data100D
228
229             input_data_name = 'data{}D.npy'.format(D)
230
231
232             # Create variable to embed
233
234             data_to_embed = tf.Variable(input_data, name=input_data_name,
235                                         trainable=False, collections=[])

```

```

221
222     # Define projector configurations
223     config = projector.ProjectorConfig()
224
225     # Add embedding
226     embedding = config.embeddings.add()
227
228     # Connect tf.Variable to embedding
229     embedding.tensor_name = data_to_embed.name
230
231     # Evaluate tf.Variable
232     sess.run(data_to_embed.initializer)
233
234     # Create save checkpoint
235     saver = tf.train.Saver([data_to_embed])
236     saver.save(sess, SUMMARY_DIR + sub_dir + '/train/model.ckpt',
237                → MAX_ITER)
238
239     # Write projector_config.pbtxt in LOG_DIR
240     projector.visualize_embeddings(train_writer, config)
241
242     ######
243     ## Function begins ##
244     #####
245
246     cluster_centres: 11 x K x D
247     train_resp:           11 x N x K
248
249     #
250
251     # Assert correct value for D
252     assert D == 2 or D == 100
253
254     # Define locally global function
255     MAX_ITER = 1500
256     CURR_TIME = '{:%b%d %H_%M_%S}'.format(datetime.datetime.now())
257     SUMMARY_DIR = CURRENT_DIR + LOG_DIR + '/MoG' + QUES_DIR + '/' +
258                 → CURR_TIME

```

```

256
257     # Create list to store run results
258     results = []
259
260     for K in K_list:
261         # Clear any pre-defined graph
262         tf.reset_default_graph()
263
264         # Build TensorFlow graph
265         X, Mu, sigma_sq, log_pi, log_resp, loss, optimizer, merged =
266             ↳ build_MoG(K, D, device)
267
268         # Select appropriate input_data
269         if has_valid:
270             input_data, valid_data = subset_data(D)
271         else:
272             input_data = data2D if D == 2 else data100D
273
274         # Create arrays to log cluster_centres, cluster_variances, pi's,
275         ↳ and responsibility indices
276         train_loss = np.array([])[:, np.newaxis]
277         if has_valid:
278             valid_loss = np.array([])[:, np.newaxis]
279             valid_resp = np.array([])[:, np.newaxis,
280                 ↳ np.newaxis].reshape(0, valid_data.shape[0], K)
281             cluster_centres = np.array([])[:, np.newaxis,
282                 ↳ np.newaxis].reshape(0, K, D)
283             cluster_variances = np.array([])[:, np.newaxis].reshape(0, K)
284             cluster_pi = np.array([])[:, np.newaxis].reshape(0, K)
285             train_resp = np.array([])[:, np.newaxis, np.newaxis].reshape(0,
286                 ↳ input_data.shape[0], K)
287
288         # Begin session
289         with tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
290             ↳ log_device_placement=False)) as sess:
291             # Log start time

```

```

286     start_time = time.time()
287
288     # Create sub-directory title
289     sub_dir = '/K={},D={},valid={}'.format(K, D, has_valid)
290
291     # Create summary writers
292     train_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir +
293                                         ' /train', graph=sess.graph)
294
295     if has_valid:
296         valid_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir +
297                                             ' /valid')
298
299     # Initialise all TensorFlow variables
300     tf.global_variables_initializer().run()
301
302     # Define iterator
303     curr_iter = 0
304
305     # Calculate training (and validation) loss,
306     # cluster centres and responsibility indices before any
307     # training
308     err, summaries, clusters, variances, log_prior_pi,
309     # log_train_indices = sess.run([loss, merged, Mu, sigma_sq,
310     # log_pi, log_resp], feed_dict={X:input_data})
311     train_loss = np.append(train_loss, err)
312     train_writer.add_summary(summaries, curr_iter)
313
314     # Log clusters and responsibility indices
315     cluster_centres = np.append(cluster_centres,
316                                 clusters[np.newaxis,:,:,:], axis=0)
317     cluster_variances = np.append(cluster_variances,
318                                 np.transpose(variances), axis=0)
319     cluster_pi = np.append(cluster_pi,
320                           np.exp(np.transpose(log_prior_pi)), axis=0)
321
322     train_resp = np.append(train_resp,
323                           np.exp(log_train_indices)[np.newaxis,:,:,:], axis=0)

```

```

314
315     # Log validation data
316
317     if has_valid:
318         err, log_valid_indices, summaries = sess.run([loss,
319             ↳ log_resp, merged], feed_dict={X:valid_data})
320
321         valid_loss = np.append(valid_loss, err)
322         valid_resp = np.append(valid_resp,
323             ↳ np.exp(log_valid_indices) [np.newaxis, :, :], axis=0)
324         valid_writer.add_summary(summaries, curr_iter)
325
326
327     # Begin training
328
329     while curr_iter < MAX_ITER:
330         # Train graph
331         _, summaries, err = sess.run([optimizer, merged, loss],
332             ↳ feed_dict={X:input_data})
333
334         # Add training loss
335         train_loss = np.append(train_loss, err)
336         train_writer.add_summary(summaries, curr_iter + 1)
337
338
339         # Log validation loss
340         if has_valid:
341             summaries, err = sess.run([merged, loss],
342                 ↳ feed_dict={X:valid_data})
343
344             valid_loss = np.append(valid_loss, err)
345             valid_writer.add_summary(summaries, curr_iter)
346
347
348         # Log responsibility indices and cluster centres every 10%
349             ↳ of maximum iteration
350
351         if ((float(curr_iter) + 1) * 100 / MAX_ITER) % 10 == 0:
352             clusters, variances, log_prior_pi, log_train_indices =
353                 sess.run([Mu, sigma_sq, log_pi, log_resp],
354                     ↳ feed_dict={X:input_data})
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779

```

```

343         cluster_centres = np.append(cluster_centres,
344             ← clusters[np.newaxis, :, :], axis=0)
345         cluster_variances = np.append(cluster_variances,
346             ← np.transpose(variances), axis=0)
347         cluster_pi = np.append(cluster_pi,
348             ← np.exp(np.transpose(log_prior_pi)), axis=0)
349
350         train_resp = np.append(train_resp,
351             ← np.exp(log_train_indices) [np.newaxis,:,:],
352             ← axis=0)
353
354     if has_valid:
355         log_valid_indices = sess.run(log_resp,
356             ← feed_dict={X:valid_data})
357         valid_resp = np.append(valid_resp,
358             ← np.exp(log_valid_indices) [np.newaxis, :, :],
359             ← axis=0)
360
361         # Post training progress to user, every 100 iterations
362         if curr_iter % 100 == 99:
363             print 'iter: {:3d}'.format(curr_iter + 1)
364
365         curr_iter += 1
366
367     # End of while loop
368     print 'Max iteration reached'
369
370     # Embed data
371     embed_data(D, train_writer)
372
373     # Close writers
374     train_writer.close()
375
376     if has_valid:
377         valid_writer.close()
378
379     if not has_valid:

```

```

371         results.append(
372             {
373                 'K': K,
374                 'train_loss': train_loss,
375                 'cluster_centres': cluster_centres,
376                 'cluster_variances': cluster_variances,
377                 'cluster_pi': cluster_pi,
378                 'train_resp': train_resp,
379                 'time_of_run': '{:%b%d
380                               ↪ %H_%M_%S}'.format(datetime.datetime.now())
380             }
381         )
382     else:
383         results.append(
384             {
385                 'K': K,
386                 'train_loss': train_loss,
387                 'valid_loss': valid_loss,
388                 'cluster_centres': cluster_centres,
389                 'cluster_variances': cluster_variances,
390                 'cluster_pi': cluster_pi,
391                 'train_resp': train_resp,
392                 'valid_resp': valid_resp,
393                 'time_of_run': '{:%b%d
394                               ↪ %H_%M_%S}'.format(datetime.datetime.now())
394             }
395         )
396
397     # TODO calculate convergence
398     print 'K: {:3d}, duration: {:.3lf}s\n'.format(K, time.time() -
399                                         ↪ start_time)
400
401     print 'RUN COMPLETED'
402     return results

```

4.1.6 MoG - Q2.2.2

```
1 # Run MoG
2 results_2_2_2 = run_MoG_v2(K_list=[3], D=2, QUES_DIR='/Q2.2.2')
3
4 # Save results
5 np.save('./Results/MoG/2_2_2.npy', results_2_2_2)
6
7 # Plot loss graph
8 def loss_IGraph(loss):
9     # Define data to plot
10    trace = go.Scatter(
11        x = range(loss.shape[0]),
12        y = loss
13    )
14    data = go.Data([trace])
15
16    # Define layout
17    layout = go.Layout(
18        title = '$-\log P(\mathbf{X})$ vs. Number of Updates',
19        xaxis = {'title': 'Updates'},
20        yaxis = {'title': 'Loss'}
21    )
22
23    # Define figure
24    figure = go.Figure(data=data, layout=layout)
25
26    # Generate plot
27    py.iplot(figure, filename='/ECE521: A3/Q2: Mixture of
28        ↳ Gaussians/Q2.2_MoG_loss', sharing='private')
29    return pyo.iplot(figure)
30
31 # Generate loss function graph
32 figure = loss_IGraph(results_2_2_2[0]['train_loss'])
```

4.1.7 MoG - Q2.2.3

```
1 # Run MoG
2 results_2_2_3 = run_MoG_v2(K_list=[1, 2, 3, 4, 5], D=2,
3     ↪ QUES_DIR='/Q2.2.3', has_valid=True)
4
5 # Save results
6 np.save('./Results/MoG/2_2_3.npy', results_2_2_3)
7
8 # Plot graph of performance comparison
9 def IGraph_2_2_3(results):
10     valid_loss = [result['valid_loss'][-1] for result in results]
11
12     figure = {
13         'data': [],
14         'layout': {}
15     }
16
17     figure['data'].append({
18         'x': [k + 1 for k in range(5)],
19         'y': valid_loss,
20     })
21
22     figure['layout'] = {
23         'title': 'MoG Model Performances on data2D.npy',
24         'showlegend': False,
25         'xaxis': {'title': 'K', 'dtick': 1},
26         'yaxis': {'title': 'Final Validation Loss'}
27     }
28
29     py.iplot(figure, \
30             filename='/ECE521: A3/Q2: Mixture of
31                 ↪ Gaussians/Q2.3_compare_MoG_clusters', \
32             sharing='private')
33
34     return pyo.iplot(figure)
```

```

33 IGraph_2_2_3(results_2_2_3)

34

35 # Generate cluster assignment bar chart
36 """
37 Generate a bar chart for each model showing percentage of data points
38     ↳ belong to each cluster
39 """
40
41
42 # Define colour list as per Plotly's default colour list
43 colour_list = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',
44     ↳ '#8c564b']
45
46 # Define empty figure
47 figure = {
48     'data': [],
49     'layout': {}
50 }
51
52 # Define data to plot
53 for i, result in enumerate(results):
54     for k in range(result['K']):
55         trace = go.Bar(
56             x = [i + 1],
57             y = [result['cluster_pi'][-1][k]] if is_MoG == True else
58                 ↳ [result['composition'][k]],
59             marker = {'color': colour_list[k]},
60             name = 'Cluster {}'.format(k + 1)
61         )
62         figure['data'].append(trace)
63
64 # Define layout
65 figure['layout'] = {
66     'title': 'Percentage of data points assigned to each {} cluster on
67             ↳ data{}D.npy'\

```

```

65     .format('MoG' if is_MoG == True else 'K-means', D),
66
67     'xaxis': {'title': 'Number of clusters, K'},
68
69     'yaxis': {'title': 'Assignment to cluster, %'},
70
71     'barmode': 'stack',
72
73     'showlegend': False
74
75 }
76
77
78 # Generate plot
79
80 py.iplot.figure, \
81     filename='/ECE521: A3/Q2: Mixture of
82     ↳ Gaussians/Q{}_assignment_bar_chart_{}D_{}\'
83
84     .format(question_name, D, 'MoG' if is_MoG is True else
85             ↳ 'K-means'), \
86
87     sharing='private')
88
89
90
91
92
93 def visualise_MoG_clusters(result):
94     '''

```

```

95      Convert hex values of type string to RGB of type int
96      Input:
97          colour_list: numpy array of type string (numColour x 1)
98      Output:
99          RGB: RGB component of type int (numColour x 3)
100         '''
101
102     def _hex_to_rgb(colour_list):
103
104         RGB = np.array([])[np.newaxis,:].reshape(0,3)
105
106         # Split hex values into R, G, B components
107
108         # Convert components to int and store in RGB array
109
110         for colour in colour_list:
111
112             RGB = np.append(RGB, np.array([int(colour[1:3], 16), \
113                                         int(colour[3:5], 16), \
114                                         int(colour[5:7], \
115                                             ↳ 16)]).reshape(1, 3), \
116                                         ↳ axis=0)
117
118         return RGB
119
120
121         '''
122
123     Convert RGB of type int to hex string of format '#xxxxxx'
124     Input:
125         RGB: RGB component of type int (N x 3)
126     Output:
127         hex_colours: (N x 1)
128
129         '''
130
131     def _rgb_to_hex(RGB):
132
133         hex_colours = np.array([])
134
135         # Convert RGB ints to a single hex string
136
137         RGB = RGB.astype(int)
138
139         for colour in RGB:
140
141             hex_colours = np.append(hex_colours,
142                                     ↳ '#{:02X}{:02X}{:02X}'.format(colour[0], colour[1],
143                                         ↳ colour[2]))
144
145         return hex_colours
146
147
148         '''

```

```

127     Return the 'average' colour based on Plotly's default colour list and
128     ↪ responsibility index
129
130     Input:
131         idx: responsibility index (N x K)
132
133     Output:
134         average_colour (N x 1)
135
136     '''
137
138     def get_colour_gradient(resp):
139
140         # Assert error if there are more colours than available colours
141
142         N = resp.shape[0]
143
144         K = resp.shape[1]
145
146         try:
147
148             assert K <= colour_list.shape
149
150         except AssertionError:
151
152             print 'Not enough colours to colour all K clusters. Consider
153             ↪ increasing number of colours in colour_list.'
154
155
156         # Matrix multiply resp (N x K) and RGB-ed colour_list (K x 3) to
157         ↪ obtain 'average' colour
158
159         # Multiply max resp to whiten less certain data points
160
161         # assigned_colour = np.matmul(resp, _hex_to_rgb(colour_list[:K]))
162
163         assigned_colour = np.matmul(np.eye(K, dtype='int')[np.argmax(resp,
164             ↪ axis=1)], _hex_to_rgb(colour_list[:K]))
165
166         white_layer = np.repeat(255, N * 3).reshape(N, 3)
167
168
169         # Append white_layer to assigned_colour on axis=2
170
171         # pre_whitened (N x K x 2)
172
173         pre_whitened = np.append(assigned_colour[:, :, np.newaxis],
174             ↪ white_layer[:, :, np.newaxis], axis=2)
175
176
177         # Create weights (N x 2)
178
179         # Second layer takes the converse of the maximum responsibility (N
180         ↪ x 1)
181
182         weights = np.append(np.ones(N)[:, np.newaxis], 1 - np.amax(resp,
183             ↪ axis=1)[:, np.newaxis], axis=1)
184
185

```

```

156     # Conform shape of weights to shape of pre_whitened
157     weights = np.transpose(np.tile(weights, (3, 1, 1)), (1, 0, 2))
158
159     # Perform weighted-average to colours
160     whitened_colour = np.average(pre_whitened, weights=weights,
161                                   axis=2)
162
163     # Return matrix of colour in hex form
164     return _rgb_to_hex(whitened_colour)
165
166     """
167     Create x- and y-coordinates for ellipses for each cluster
168     Assumptions:
169         Joint independence and equal marginal variances
170         Dimension of data point is 2
171     Returns:
172         ellipse: x- and y-coordinates for K ellipses (N x K x D)
173
174     def calc_ellipse_coordinates(centres, variances):
175         # Create trace for region to encompass 95% of the points (using
176         # Chi-squared critical value)
177         # Assuming joint independence and equal marginal variances
178
179         # Chi-squared with df 2 and alpha=5%
180         crit_val = 5.991
181
182
183         # Calculate axes length
184         axis_lengths = np.sqrt(variances * crit_val)
185
186         # Calculate coordinates to trace ellipse
187         t = np.arange(-np.pi, np.pi + np.pi / 50, np.pi / 50) # Parameter
188         x = np.transpose(centres[:, 0][:, np.newaxis]) + axis_lengths *
189             np.cos(t)[:, np.newaxis]
190         y = np.transpose(centres[:, 1][:, np.newaxis]) + axis_lengths *
191             np.sin(t)[:, np.newaxis]

```

```

188     # Stack x- and y-coordinates along axis=2
189     ellipse = np.stack([x, y], axis=2)
190
191     return ellipse
192
193 ######
194 ## Function begins ##
195 #####
196
197 # Define K and divider between training and validation data
198 K = result['K']
199 divider = data2D.shape[0] * 2 / 3 # Anything before K is part of the
→   training data. Anything after is part of validation data
200
201 # Store cluster parameters and responsibility indices
202 centres = result['cluster_centres'][-1]
203 variances = result['cluster_variances'][-1]
204 train_resp = result['train_resp'][-1]
205 valid_resp = result['valid_resp'][-1]
206
207 # Create ellipse coordinates
208 ellipse = calc_ellipse_coordinates(centres, variances)
209
210 # Define colour list as per Plotly's default colour list
211 colour_list = np.array(['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728',
→   '#9467bd', '#8c564b'])
212
213 # Define blank figure
214 figure = {
215     'data': [],
216     'layout': {}
217 }
218
219 # Create trace for training data points
220 # Create trace for validation data points
221 valid_data_trace = {

```

```

222     'x': data2D[divider][:,0],
223     'y': data2D[divider][:,1],
224     'mode': 'markers',
225     'hoverinfo': 'none',
226     'marker': {
227         'size': 4,
228         'color': colour_list[np.argmax(valid_resp, axis=1)]
229             #get_colour_gradient(valid_resp)
230     }
231
232     # Append data traces
233     figure['data'].append(valid_data_trace)
234
235     for k in range(K):
236         # Create trace for cluster centres
237         centre_trace = {
238             'x': np.round([centres[k][0]], 3),
239             'y': np.round([centres[k][1]], 3),
240             'name': 'Cluster {}'.format(k + 1),
241             'mode': 'markers',
242             'marker': {
243                 'size': 12,
244                 'symbol': 'diamond',
245                 'color': colour_list[k],
246                 'line': {'width': 3}
247             }
248         }
249
250         # Create trace for region encompassing 95% of data points
251         variance_trace = {
252             'x': ellipse[:,k,:][:,0],
253             'y': ellipse[:,k,:][:,1],
254             'hoverinfo': 'none',
255             'mode': 'lines',
256             'name': 'Cluster {}'.format(k + 1),

```

```

257         'marker': {
258             'color': colour_list[k]
259         }
260     }
261
262     # Add cluster trace
263     for trace in [centre_trace, variance_trace]:
264         figure['data'].append(trace)
265
266     # Generate figure layout
267     figure['layout'] = go.Layout(
268         width = 900,
269         height = 900,
270         showlegend = False,
271         title = 'MoG Clustering Visualisation (K = {})'.format(K),
272         xaxis = {'range': [-4, 4], 'autorange': False},
273         yaxis = {'range': [-5, 2], 'autorange': False}
274     )
275
276     # Upload graph to cloud
277     py.iplot(figure, \
278             filename='/ECE521: A3/Q2: Mixture of
279             ↳ Gaussians/Q2.3_MoG_clusters_K={}'.format(K), \
280             sharing='private')
281
282     return pyo.iplot(figure)
283
284 fig2_2_3 = []
285 for result in results_2_2_3:
286     fig2_2_3.append(visualise_MoG_clusters(result))

```

4.1.8 MoG - Q2.2.4

```
1 # Run K-means
2 results_2_2_2_4_K_means = run_k_means(K_list=[1, 2, 3, 4, 5, 6, 7, 8, 9,
3   ↳ 10, 11, 12, 13, 14, 15], data_dim=100, device='cpu', has_valid=True)
4
5 # Run MoG
6 results_2_2_4_MoG = run_MoG_v2(K_list=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
7   ↳ 12, 13, 14, 15], D=100, has_valid=True, device='gpu',
8   ↳ QUES_DIR='/Q2.2.4')
9
10
11 # Save results
12 np.save('./Results/MoG/2_2_4_MoG_x15.npy', results_2_2_4_MoG)
13 np.save('./Results/MoG/2_2_4_K-means.npy', results_2_2_2_4_K_means)
14
15
16 # Plot validation losses
17 def IGraph_2_2_4(K_means, MoG):
18     valid_loss_K_means = [result['valid_loss'][-1] for result in K_means]
19     valid_loss_MoG = [result['valid_loss'][-1] for result in MoG]
20
21
22     figure = tools.make_subplots(rows=2, cols=1, shared_xaxes=True,
23       ↳ subplot_titles=('K-means', 'MoG'))
24
25     K_means_trace = {
26         'x': [k + 1 for k in range(15)],
27         'y': valid_loss_K_means,
28         'name': 'K-means'
29     }
30
31
32     MoG_trace = {
33         'x': [k + 1 for k in range(15)],
34         'y': valid_loss_MoG,
35         'name': 'MoG'
36     }
37
38     figure.append_trace(K_means_trace, 1, 1)
```

```

31     figure.append_trace(MoG_trace, 2, 1)

32

33     figure['layout'].update({
34         'height': 500,
35         'width': 800,
36         'title': 'K-means and MoG Models Performances on data100D.npy',
37         'xaxis1': {'title': 'Number of clusters, K', 'dtick': 1},
38         'yaxis1': {'title': '$\\text{Valid. Loss, }$',
39                     '→ $\\mathcal{L}(\\mathbf{\\mu})$'},
40         'yaxis2': {'title': '$\\text{Valid. Loss, }$ - $P(\\mathbf{X})$'},
41         'showlegend': False
42     })
43
44
45     py.iplot(figure, filename='/ECE521: A3/Q2: Mixture of
46             → Gaussians/Q2.4_K_means_vs_MoG', sharing='private')
47
48     return pyo.iplot(figure)
49
50     IGraph_2_2_4(results_2_2_4_K_means, results_2_2_4_MoG)

```

4.1.9 Base code for FA

```
1 # Import relevant packages
2 import tensorflow as tf
3 import numpy as np
4
5 import time
6 import datetime
7
8 # Non-interactive plotting
9 import matplotlib.pyplot as plt
10
11 # Interactive plotting
12 from plotly import tools
13 import plotly.plotly as py
14 import plotly.graph_objs as go
15 import plotly.grid_objs as gro
16 import plotly.offline as pyo
17 from plotly.offline import download_plotlyjs
18
19 # Configure environment
20 %config InlineBackend.figure_format = 'retina'
21 np.set_printoptions(precision=3)
22
23 # Global Variables
24 CURRENT_DIR =
25     ↵ '/Users/christophertee/Dropbox/University/MASc/Courses/Winter 2017' +
26     ↵ \
27         ↵ '/ECE521 (Inference Algorithms & Machine'
28         ↵ Learning) /Assignment 3'
28 LOG_DIR = '/Logs'
29
30 # Activate Plotly Offline for Jupyter
31 pyo.init_notebook_mode(connected=True)
32
33 # Define global variable SEED
```

```

32 SEED = 521
33
34 # Load data into memory
35
36 """
37 tinymnist.npz consists of images of '3's and '5's with dimensions (8 x 8)
38
39 train_data: 700 data points
40 valid_data: 100 data points
41 test_data: 400 data points
42 """
43 with np.load("./Data/tinyminst.npz") as data :
44     # Generate _data
45     train_data, train_target = data ["x"], data["y"]
46     valid_data, valid_target = data ["x_valid"], data ["y_valid"]
47     test_data, test_target = data ["x_test"], data ["y_test"]
48
49 """
50 Builds TensorFlow graph for FA
51
52 Input:
53     K: number of latent variables
54     D: dataset dimension
55     device: CPU or GPU to perform computation-heavy ops
56 Internal variables:
57     X: input data matrix (N x D)
58     mu: mean of each input (D x 1)
59     psi_vector: variance of x_n given s_n for each dimension (D x 1)
60     W: latent_matrix that projects s_n from K-dimensions to D-dimensions
61     ↵ (K x D)
62     Sigma: Marginal covariance matrix (D x D)
63 """
64
65 def build_FA(K, D, device='cpu'):
66     """
67     Helper function to add histogram tag to variables
68     Input:

```

```

67     var: variable to be tagged with histogram summary
68     """
69
70     def _add_histogram(vars_):
71         for var in vars_:
72             tf.summary.histogram(var.op.name, var)
73
74         ##### Function begins #####
75         #####
76
77     # Define computation device
78     try:
79         assert device == 'cpu' or device == 'gpu'
80     except AssertionError:
81         print 'Invalid device chosen. Please use \'cpu\' or \'gpu\''
82         quit()
83     device = '/' + device + ':0'
84
85     # Fix TF graph seed
86     tf.set_random_seed(SEED)
87
88     with tf.device('/cpu:0'):
89         # Define placeholder
90         with tf.name_scope('placeholder'):
91             X = tf.placeholder(tf.float32, shape=[None, D], name='inputs')
92
93         # Define parameters
94         with tf.variable_scope('generated_parameters'):
95             phi = tf.get_variable('latent_for_psi', shape=[D, 1], \
96
97                                     initializer=tf.truncated_normal_initializer(seed=SEED))
98             W = tf.get_variable('latent_matrix_W', shape=[D, K], \
99
100                                     initializer=tf.truncated_normal_initializer(seed=SEED))
101             mu = tf.get_variable('mean', shape=[D, 1],
102                                     initializer=tf.truncated_normal_initializer(seed=SEED))

```

```

100
101    with tf.device(device):
102        # Calculate gaussian parameters
103        with tf.name_scope('gaussian_parameters'):
104            psi_vector = tf.exp(phi, name='variance_vector_psi')
105            Psi = tf.multiply(tf.eye(tf.shape(psi_vector)[0]), psi_vector,
106                             name='Psi_matrix')
107            Sigma = tf.add(Psi, tf.matmul(W, tf.transpose(W)),
108                           name='Sigma')
109
110        # Calculate projection matrix to infer latent variable s
111        with tf.name_scope('projection_matrix'):
112            Sigma_s_posterior = tf.matrix_inverse(tf.eye(K) \
113                                                 + tf.matmul(tf.transpose(W), \
114                                                 tf.matmul(tf.matrix_inverse(Psi),
115                                                 W)))
116            W_proj = tf.matmul(Sigma_s_posterior,
117                               tf.matmul(tf.transpose(W), tf.matrix_inverse(Psi)))
118
119        # Calculate loss function
120        with tf.name_scope('marginal_log_likelihood'):
121            with tf.name_scope('Mahalanobis_dist'):
122                # Expand variables for tensor multiplication
123                X_expanded = tf.expand_dims(X, axis=2)
124                mu_expanded = tf.expand_dims(mu, axis=0)
125
126                # Calculate mahalanobis distance
127                Sigma_inv_tiled =
128                    tf.tile(tf.expand_dims(tf.matrix_inverse(Sigma),
129                                 axis=0), \
130                                 multiples=[tf.shape(X)[0], 1, 1],
131                                 name='Sigma_inv_tiled')
132
133                dist = tf.reduce_sum(- 1. / 2 * \

```

```

128             tf.matmul(tf.transpose(X_expanded -
129                         ↪ mu_expanded, perm=[0,2,1]), \
130                         tf.matmul(Sigma_inv_tiled, \
131                         X_expanded - \
132                         ↪ mu_expanded)), \
133                         ↪ \
134                         name='Mahalanobis_dist')
135
136
137     # Calculate log gaussian constant
138
139     with tf.name_scope('log_gauss_const'):
140
141         log_gauss_const = tf.negative(tf.cast(tf.shape(X)[0],
142                         ↪ tf.float32) * tf.cast(D, tf.float32) \
143                         * tf.log(2. * np.pi) / 2, \
144                         name='log_gauss_const')
145
146
147     # Calculate log_determinant
148
149     with tf.name_scope('log_det'):
150
151         test = tf.cholesky(Sigma)
152
153         log_det = tf.negative(tf.cast(tf.shape(X)[0], tf.float32)
154                         ↪ \
155                         * \
156                         ↪ tf.reduce_sum(tf.log(tf.diag_part(tf.cholesky(
157                         Sigma))), \
158                         name='log_det'))
159
160
161     # Calculate loss function
162
163     loss = tf.negative(tf.add_n([dist, log_gauss_const, log_det]),
164                         ↪ name='loss')
165
166     tf.summary.scalar('loss', loss)
167
168
169     # Define optimizer
170
171     with tf.name_scope('Adam_optimizer'):
172
173         optimizer = tf.train.AdamOptimizer(learning_rate=0.01, \
174                         beta1=0.9, beta2=0.99,
175                         ↪ epsilon=1e-5).minimize(loss)
176
177
178     with tf.device('/cpu:0'):

```

```

156     # Add histogram summaries for variables of interest
157     _add_histogram([psi_vector, W, Sigma])
158
159     # Merge all summaries
160     merged = tf.summary.merge_all()
161
162     return X, mu, psi_vector, W, Sigma, W_proj, loss, optimizer, merged,
163     → test
164
165     """
166     Runs FA training algorithm. Tensorboard embedding enabled
167     """
168
169     def run_FA(K_list, QUES_DIR, D=64, device='cpu'):
170
171         """
172             Embed data for visualization purposes
173         """
174
175         def embed_data(D, train_writer):
176
177             # Define input data
178
179             if D == 64:
180
181                 input_data = train_data
182
183                 input_data_name = 'tinymnist_training'
184
185             elif D == 3:
186
187                 input_data = toy_data
188
189                 input_data_name = 'toy_data'
190
191             # Create variable to embed
192
193             data_to_embed = tf.Variable(input_data, name=input_data_name,
194             → trainable=False, collections[])
195
196             # Define projector configurations
197
198             config = projector.ProjectorConfig()
199
200
201             # Add embedding
202
203             embedding = config.embeddings.add()
204
205
206             # Connect tf.Variable to embedding

```

```

190     embedding.tensor_name = data_to_embed.name
191
192     # Evaluate tf.Variable
193     sess.run(data_to_embed.initializer)
194
195     # Create save checkpoint
196     saver = tf.train.Saver([data_to_embed])
197     saver.save(sess, SUMMARY_DIR + sub_dir + '/train/model.ckpt',
198                ↳ MAX_ITER)
199
200     # Write projector_config.pbtxt in LOG_DIR
201     projector.visualize_embeddings(train_writer, config)
202
203     ######
204     ## Function begins ##
205     #####
206
207     # Check compatibility of dataset
208     try:
209         assert D == 64 or D == 3
210     except AssertionError:
211         print 'Incompatible dataset dimension, D. Please use 64 or 3\
212             for tinymnist or toy dataset respectively'
213         quit()
214
215     # Define locally global function
216     MAX_ITER = 1200 if D == 64 else 5000
217     CURR_TIME = '{:0b%H_%M_%S}'.format(datetime.datetime.now())
218     SUMMARY_DIR = CURRENT_DIR + LOG_DIR + '/FA/' + QUES_DIR + '/' +
219             ↳ CURR_TIME
220
221     # Create list to store run results
222     results = []
223
224     for K in K_list:
225         # Clear any pre-defined graph

```

```

224     tf.reset_default_graph()
225
226     # Build TensorFlow graph
227     X, mu, psi_vector, W, Sigma, W_proj, loss, optimizer, merged, test
228     ↵ = build_FA(K, D, device)
229
230     # Select appropriate input_data
231     if D == 64:
232         input_data = train_data
233     elif D == 3:
234         input_data = toy_data
235
236     # Create arrays to log losses, psi and W
237     train_loss = np.array([])[:, np.newaxis]
238     if D == 64:
239         valid_loss = np.array([])[:, np.newaxis]
240         test_loss = np.array([])[:, np.newaxis]
241
242     mean = np.array([])[:, np.newaxis].reshape(0, D)
243     psi = np.array([])[:, np.newaxis].reshape(0, D)
244     Ws = np.array([])[:, np.newaxis, np.newaxis].reshape(0, D, K)
245     Wproj = np.array([])[:, np.newaxis, np.newaxis].reshape(0, K, D)
246
247     # Begin session
248     with tf.Session(config=tf.ConfigProto(allow_soft_placement=True,
249                     ↵ log_device_placement=False)) as sess:
250         # Log start time
251         start_time = time.time()
252
253         # Create sub-directory title
254         sub_dir = '/K={},D={}{}'.format(K, D)
255
256         # Create summary writers
257         train_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir +
258             '/train', graph=sess.graph)
259         if D == 64:

```

```

257         valid_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir
258             ↵ + '/valid')
259
260         test_writer = tf.summary.FileWriter(SUMMARY_DIR + sub_dir
261             ↵ + '/test')
262
263     # Initialise all TensorFlow variables
264     tf.global_variables_initializer().run()
265
266     # Define iterator
267     curr_iter = 0
268
269     # Calculate training (and validation) loss,
270     # cluster centres and responsibility indices before any
271     # → training
272     err, summaries, curr_mu, curr_psi, curr_W, curr_W_proj = \
273         sess.run([loss, merged, mu, psi_vector, W, W_proj],
274             ↵ feed_dict={X: input_data})
275
276     train_loss = np.append(train_loss, err)
277
278     train_writer.add_summary(summaries, curr_iter)
279
280     # Log psi and W
281     mean = np.append(mean, np.transpose(curr_mu), axis=0)
282     psi = np.append(psi, np.transpose(curr_psi), axis=0)
283     Ws = np.append(Ws, curr_W[np.newaxis, :, :], axis=0)
284     Wproj = np.append(Wproj, curr_W_proj[np.newaxis, :, :], axis=0)
285
286     if D == 64:
287
288         # Log validation loss
289         err, summaries = sess.run([loss, merged],
290             ↵ feed_dict={X:valid_data})
291
292         valid_loss = np.append(valid_loss, err)
293
294         valid_writer.add_summary(summaries, curr_iter)
295
296         # Log test loss
297         err, summaries = sess.run([loss, merged],
298             ↵ feed_dict={X:test_data})

```

```

287     test_loss = np.append(test_loss, err)
288     test_writer.add_summary(summaries, curr_iter)
289
290     # Begin training
291     while curr_iter < MAX_ITER:
292         # Train graph
293         _, summaries, err = sess.run([optimizer, merged, loss],
294                                     feed_dict={X:input_data})
295
296         # Add training loss
297         train_loss = np.append(train_loss, err)
298         train_writer.add_summary(summaries, curr_iter + 1)
299
300         if D == 64:
301             # Log validation loss
302             summaries, err = sess.run([merged, loss],
303                                     feed_dict={X:valid_data})
304             valid_loss = np.append(valid_loss, err)
305             valid_writer.add_summary(summaries, curr_iter)
306
307             # Log test loss
308             err, summaries = sess.run([loss, merged],
309                                     feed_dict={X:test_data})
310             test_loss = np.append(test_loss, err)
311             test_writer.add_summary(summaries, curr_iter)
312
313             # Log responsibility indices and cluster centres every 10%
314             # of maximum iteration
315             if ((float(curr_iter) + 1) * 100 / MAX_ITER) % 10 == 0:
316                 curr_mu, curr_psi, curr_W, curr_W_proj = sess.run([mu,
317                                         psi_vector, W, W_proj], feed_dict={X:input_data})
318
319                 mean = np.append(mean, np.transpose(curr_mu), axis=0)
320                 psi = np.append(psi, np.transpose(curr_psi), axis=0)
321                 Ws = np.append(Ws, curr_W[np.newaxis, :, :], axis=0)
322                 Wproj = np.append(Wproj, curr_W_proj[np.newaxis, :, :],
323                                 axis=0)

```

```

318
319         # Post training progress to user, every 100 iterations
320         if curr_iter % 100 == 99:
321             print 'iter: {:3d}, train_loss:
322                 ↪ {:.5f}'.format(curr_iter + 1, train_loss[-1])
323
324             curr_iter += 1
325
326         # End of while loop
327         print 'Max iteration reached'
328
329         # Embed data
330         embed_data(D, train_writer)
331
332         # Close writers
333         train_writer.close()
334         if D == 64:
335             valid_writer.close()
336             test_writer.close()
337
338         if D == 64:
339             results.append(
340                 {
341                     'K': K,
342                     'train_loss': train_loss,
343                     'valid_loss': valid_loss,
344                     'test_loss': test_loss,
345                     'psi': psi,
346                     'mean': mean,
347                     'W': Ws,
348                     'W_proj': Wproj,
349                     'time_of_run': '{}:{}{}'.format(datetime.datetime.now().hour,
350                                         datetime.datetime.now().minute,
351                                         datetime.datetime.now().second)
352                 }
353             )
354
355         elif D == 3:

```

```

352         results.append(
353             {
354                 'K': K,
355                 'train_loss': train_loss,
356                 'psi': psi,
357                 'mean': mean,
358                 'W': Ws,
359                 'W_proj': Wproj,
360                 'time_of_run': '{:%b%d'
361                             ↳ '%H_%M_%S}'.format(datetime.datetime.now())
361             }
362         )
363
364     # TODO calculate convergence
365     print 'K: {:3d}, duration: {:.1f}s\n'\
366         .format(K, time.time() - start_time)
367
368     print 'RUN COMPLETED'
369
370     return results

```

4.1.10 FA - Q3.1.2

```
1 # Run FA
2 results_3_1_2 = run_FA(K_list=[4], QUES_DIR='/Q3.1.2', device='gpu')
3
4 # Save results
5 np.save('./Results/FA/3_1_2.npy', results_3_1_2)
6
7 # Visualise weights
8 '''
9 Creates 2x2 subplot of weights
10 Each subplot consists of an 8x8 heatmap for the weight of each latent
    ↪ variable
11 Input:
12     weights: final trained weights (D x K)
13 '''
14 def visualise_weights(weights):
15     # Define colour list as per Plotly's default colour list
16     colour_list = np.array(['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728',
    ↪ '#9467bd', '#8c564b'])
17
18     # Define empty figure
19     figure = tools.make_subplots(rows=2, cols=2, subplot_titles=('Weight
    ↪ 1', 'Weight 2', 'Weight 3', 'Weight 4'))
20
21     # Define subplot traces
22     for i, weight in enumerate(np.transpose(weights)):
23         trace = go.Heatmap(
24             z = np.reshape(weight, (8,8)), # TODO Reverse order
25             colorscale = [[0, '#000000'], [1, '#FFFFFF']],
26             showscale = False
27         )
28         figure.append_trace(trace, i / 2 + 1, i % 2 + 1)
29
30         figure['layout'][f'xaxis{i}'].update(showticklabels =
    ↪ False, ticks = '')
```

```
31     figure['layout']['yaxis{}'.format(i + 1)].update(showticklabels =
32         False, ticks = '', autorange='reversed')
33
34     figure['layout'].update(
35         height = 900,
36         width = 800,
37         showlegend = False,
38         title = 'Weights Visualisation of Latent Matrix, W'
39     )
40
41     py.iplot(figure, filename='/ECE521: A3/Q3: Factor
42         Analysis/Q1.2_weights_viz', sharing='private')
43
44     return pyo.iplot(figure)
45
46
47     visualise_weights(results_3_1_2[0]['W'][-1])
```

4.1.11 FA - Q3.1.3

```
1 # Generate toy dataset
2 # Reset random seed
3 np.random.seed(SEED)
4
5 # Define variable to store toy data
6 toy_data = np.array([])[np.newaxis, :].reshape(0, 3)
7
8 for i in range(5000):
9     # Sample s from normal distribution
10    s = np.random.randn(3,1)
11
12    # Define conversion matrix from s to x
13    A = np.array([[1, 0, 0],
14                  [1, 0.001, 0],
15                  [0, 0, 10]])
16
17    toy_data = np.append(toy_data, np.transpose(np.matmul(A, s)), axis=0)
18
19 # Train using PCA
20 '''
21 Finds the largest principal component by finding the normalised
22     ↳ eigenvector corresponding
23 to the largest eigenvalue of the data covariance matrix
24 '''
25
26 def largest_component_PCA(data):
27     # Obtain covariance matrix
28     sigma = np.cov(np.transpose(data))
29
30     # Calculate eigenvalues and eigenvectors
31     e_value, e_vector = np.linalg.eig(sigma)
32
33     # Return the largest principle component
34     PC = e_vector[np.argmax(e_value)][:, np.newaxis]
```

```

34     # Save data
35     np.save('./Results/FA/3_1_3_PCA_x{}.npy'.format(data.shape[0]), PC)
36
37     return PC
38
39 print 'Principle component: \n{}'.format(largest_component_PCA(toy_data))
40
41 # Train using FA
42 results_3_1_3 = run_FA(K_list=[1], D=3, QUES_DIR='Q3.1.3', device='cpu')

```

4.2 Soon Chee Loong's Implementation

4.2.1 K-Means

```

1 import tensorflow as tf
2 import numpy as np
3 from dataInitializer import DataInitializer
4 import datetime
5 import sys
6
7 class KMeans(object):
8     def __init__(self, questionTitle, K, trainData, validData, hasValid,
9                  dataType, numEpoch = 50, learningRate = 0.1):
10         """
11             Constructor
12         """
13         self.K = K
14         self.dataType = dataType
15         self.trainData = trainData
16         self.validData = validData
17         self.D = self.trainData[0].size # Dimension of each data
18         self.hasValid = hasValid
19         self.learningRate = learningRate
20         self.numEpoch = numEpoch

```

```

20     self.miniBatchSize = self.trainData.shape[0] # miniBatchSize is
21         ↪ entire data size
22     self.questionTitle = questionTitle
23     self.optimizer = tf.train.AdamOptimizer(learning_rate =
24         ↪ self.learningRate, beta1=0.9, beta2=0.99, epsilon=1e-5)
25     # Execute KMeans
26     self.KMeansMethod()
27
28
29
30     def printPlotResults(self, xAxis, yTrainErr, yValidErr, numUpdate,
31         ↪ minAssignTrain, currTrainData, centers, minAssignValid):
32         figureCount = 0 # TODO: Make global
33         import matplotlib.pyplot as plt
34
35         print "K: ", self.K
36         print "Iter: ", numUpdate
37         print str(self.K) + " Lowest TrainLoss", np.min(yTrainErr)
38         print str(self.K) + " Lowest ValidLoss", np.min(yValidErr)
39         # Count how many assigned to each class
40         numTrainAssignEachClass = np.bincount(minAssignTrain)
41         numValidAssignEachClass = np.bincount(minAssignValid)
42         print "Train Percentage Assignment To Classes:",
43             ↪ percentageTrainAssignEachClass
44         print "Train Assignments To Classes:", numTrainAssignEachClass
45         percentageTrainAssignEachClass =
46             ↪ numTrainAssignEachClass/float(sum(numTrainAssignEachClass))
47
48         percentageValidAssignEachClass = percentageTrainAssignEachClass # Initialize
49
50
51     if self.hasValid:
52         print "Valid Assignments To Classes:", numValidAssignEachClass
53         percentageValidAssignEachClass =
54             ↪ numValidAssignEachClass/float(sum(numValidAssignEachClass))
55         print "Valid Percentage Assignment To Classes:",
56             ↪ percentageValidAssignEachClass

```

```

48     trainStr = "Train"
49     validStr = "Valid"
50     typeLossStr = "Loss"
51     typeScatterStr = "Assignments"
52     trainLossStr = trainStr + typeLossStr
53     validLossStr = validStr + typeLossStr
54     iterationStr = "Iteration"
55     dimensionOneStr = "D1"
56     dimensionTwoStr = "D2"
57     paramString = "K" + str(self.K) + "Learn" + str(self.learningRate) +
58         "NumEpoch" + str(self.numEpoch)
59
59     # Train Loss
60     figureCount = figureCount + 1
61     plt.figure(figureCount)
62     title = trainStr + typeLossStr + paramString
63     plt.title(title)
64     plt.xlabel(iterationStr)
65     plt.ylabel(typeLossStr)
66     plt.plot(np.array(xAxis), np.array(yTrainErr), label =
67             trainLossStr)
68     plt.legend()
69     plt.savefig(self.questionTitle + title + ".png")
70     plt.close()
71     plt.clf()
72
72     # Valid Loss
73     if self.hasValid:
74         figureCount = figureCount + 1
75         plt.figure(figureCount)
76         title = validStr + typeLossStr + paramString
77         plt.title(title)
78         plt.xlabel(iterationStr)
79         plt.ylabel(typeLossStr)
80         plt.plot(np.array(xAxis), np.array(yValidErr), label =
81             validLossStr)

```

```

81         plt.legend()
82
83         plt.savefig(self.questionTitle + title + ".png")
84
85         plt.close()
86
87         plt.clf()
88
89
90     if self.dataType != "2D":
91
92         return
93
94
95     # Plot percentage in each different classes as well
96
97     # Scatter plot based on assignment colors
98
99     # Including percentage as the label
100
101    figureCount = figureCount + 1
102
103    plt.figure(figureCount)
104
105    title = trainStr + typeScatterStr + paramStr
106
107    plt.title(title)
108
109    plt.xlabel(dimensionOneStr)
110
111    plt.ylabel(dimensionTwoStr)
112
113    colors = ['blue', 'red', 'green', 'black', 'yellow', 'magenta',
114              'cyan', 'brown', 'orange',
115              'aqua']
116
117    colors = colors[:self.K]
118
119    plt.scatter(currTrainData[:, 0], currTrainData[:, 1],
120                c=minAssignTrain, s=10, alpha=0.5)
121
122    for i, j, k in zip(centers, percentageTrainAssignEachClass,
123                        colors):
124
125        plt.plot(i[0], i[1], 'kx', markersize=15, label=j, c=k)
126
127    plt.legend()
128
129    plt.savefig(self.questionTitle + title + ".png")
130
131    plt.close()
132
133    plt.clf()
134
135
136    if self.hasValid:
137
138        # Valid Assignments
139
140        figureCount = figureCount + 1
141
142        plt.figure(figureCount)
143
144        title = validStr + typeScatterStr + paramStr

```

```

114         plt.title(title)
115         plt.xlabel(dimensionOneStr)
116         plt.ylabel(dimensionTwoStr)
117         colors = ['blue', 'red', 'green', 'black', 'yellow',
118                   'magenta', 'cyan', 'brown', 'orange',
119                   'aqua']
120         colors = colors[:self.K]
121         plt.scatter(self.validData[:, 0], self.validData[:, 1],
122                     c=minAssignValid, s=10, alpha=0.5)
123         for i, j, k in zip(centers, percentageValidAssignEachClass,
124                             colors):
125             plt.plot(i[0], i[1], 'kx', markersize=15, label=j, c=k)
126         plt.legend()
127         plt.savefig(self.questionTitle + title + ".png")
128         plt.close()
129         plt.clf()
130
131
132     def PairwiseDistances(self, X, U):
133         """
134         input:
135             X is a matrix of size (B x D)
136             U is a matrix of size (K x D)
137         output:
138             Distances = matrix of size (B x D) containing the pairwise
139             Euclidean distances
140
141         """
142         batchSize = tf.shape(X)[0]
143         dimensionSize = tf.shape(X)[1]
144         numClusters = tf.shape(U)[0]
145         X_broadcast = tf.reshape(X, (batchSize, 1, dimensionSize))
146         sumOfSquareDistances =
147             tf.reduce_sum(tf.square(tf.subtract(X_broadcast, U)), 2)
148         return sumOfSquareDistances
149
150
151     def KMeansMethod(self):
152         """
153

```

```

145      Build Graph and execute in here
146      so don't have to pass variables one by one
147      Bad Coding Style but higher programmer productivity
148      '''
149
150      # Build Graph
151      U = tf.Variable(tf.truncated_normal([self.K, self.D]))
152      train_data = tf.placeholder(tf.float32, shape=[None, self.D],
153                                  ↳ name="trainingData")
154      sumOfSquare = self.PairwiseDistances(train_data, U)
155      minSquare = tf.reduce_min(sumOfSquare, 1)
156      minAssignments = tf.argmin(sumOfSquare, 1)
157      loss = tf.reduce_sum(minSquare)
158      validLoss = loss
159      minValidAssignments = minAssignments
160
161      if self.hasValid:
162          valid_data = tf.placeholder(tf.float32, shape=[None, self.D],
163                                      ↳ name="validationData")
164          validSumOfSquare = self.PairwiseDistances(valid_data, U)
165          validLoss = tf.reduce_sum(tf.reduce_min(validSumOfSquare, 1))
166          minValidAssignments = tf.argmin(validSumOfSquare, 1)
167
168          train = self.optimizer.minimize(loss)
169
170          # Session
171          init = tf.global_variables_initializer()
172          sess = tf.InteractiveSession()
173          sess.run(init)
174          currEpoch = 0
175          minAssign = 0
176          centers = 0
177          xAxis = []
178          yTrainErr = []

```

```

179     currTrainDataShuffle = self.trainData
180
181     while currEpoch < self.numEpoch:
182         np.random.shuffle(self.trainData) # Shuffle Batches
183
184         currTrainDataShuffle = self.trainData
185         step = 0
186
187         while step * self.miniBatchSize < self.trainData.shape[0]:
188             feedDicts = {train_data:
189                         ↳ self.trainData[step * self.miniBatchSize:(step + 1) * self.miniBatchSize]
190
191             if self.hasValid:
192                 feedDicts = {train_data:
193                             ↳ self.trainData[step * self.miniBatchSize:(step + 1) * self.miniBatchSize]
194                             ↳ valid_data: self.validData}
195
196             __, minAssignTrain, minAssignValid, centers, errTrain,
197             ↳ errValid = sess.run([train, minAssignments,
198             ↳ minValidAssignments, U, loss, validLoss], feed_dict =
199             ↳ feedDicts)
200
201             xAxis.append(numUpdate)
202
203             yTrainErr.append(errTrain)
204
205             yValidErr.append(errValid)
206
207             step += 1
208
209             numUpdate += 1
210
211             currEpoch += 1
212
213             if currEpoch % 50 == 0:
214                 logStdOut("e: " + str(currEpoch))
215
216             if self.dataType == "2D":
217
218                 print "Center Values", centers
219
220                 self.printPlotResults(xAxis, yTrainErr, yValidErr, numUpdate,
221                         ↳ minAssignTrain, currTrainDataShuffle, centers,
222                         ↳ minAssignValid)
223
224
225     def executeKMeans(questionTitle, K, dataType, hasValid):
226
227         """
228
229             Re-loads the data and re-randomize it with same seed anytime to ensure
230             ↳ replicable results
231
232         """
233
234         logStdOut(questionTitle)

```

```

206     print questionTitle
207
208     trainData = 0
209
210     validData = 0
211
212     # Load data with seeded randomization
213     dataInitializer = DataInitializer()
214
215     if hasValid:
216
217         trainData, validData = dataInitializer.getData(dataType, hasValid)
218
219     else:
220
221         trainData = dataInitializer.getData(dataType, hasValid)
222
223     # Execute algorithm
224
225     kObject = KMeans(questionTitle, K, trainData, validData, hasValid,
226                       → dataType)
227
228     logElapsedTime(questionTitle + "K" + str(K))
229
230
231
232     # Global for logging
233
234     questionTitle = "" # Need to be global for logging to work
235
236     startTime = datetime.datetime.now()
237
238     figureCount = 1 # To not overwrite existing pictures
239
240
241     def logStdOut(message):
242
243         # Temporary print to std out
244
245         sys.stdout = sys.__stdout__
246
247         print message
248
249         # Continue editing same file
250
251         sys.stdout = open("result" + questionTitle + ".txt", "a")
252
253
254     def logElapsedTime(message):
255
256         ''' Logs the elapsedTime with a given message '''
257
258         global startTime
259
260         endTime = datetime.datetime.now()
261
262         elapsedTime = endTime - startTime
263
264         hours, remainder = divmod(elapsedTime.seconds, 3600)
265
266         minutes, seconds = divmod(remainder, 60)
267
268         totalDays = elapsedTime.days
269
270         timeStr = str(message) + ': Days: ' + str(totalDays) + " hours: "
271
272         → str(hours) + ' minutes: ' + str(minutes) + ' seconds: '
273
274         → str(seconds)

```

```

241 logStdOut(timeStr)
242 startTime = datetime.datetime.now()
243
244 if __name__ == "__main__":
245     print "ECE521 Assignment 3: Unsupervised Learning: K Means"
246
247     # Unsupervised => Data has no label or target
248     """
249     questionTitle = "1.1.2"
250     dataType = "2D"
251     hasValid = False # No validation data
252     K = 3
253     executeKMeans(questionTitle, K, dataType, hasValid)
254     """
255
256     """
257     questionTitle = "1.1.3"
258     diffK = [1, 2, 3, 4, 5]
259     dataType = "2D"
260     hasValid = False
261     for K in diffK:
262         executeKMeans(questionTitle, K, dataType, hasValid)
263     """
264
265     """
266     questionTitle = "1.1.4"
267     diffK = [1, 2, 3, 4, 5]
268     dataType = "2D"
269     hasValid = True
270     for K in diffK:
271         executeKMeans(questionTitle, K, dataType, hasValid)
272     """
273
274     # Run using 100D data
275     questionTitle = "2.2.4.1"
276     diffK = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```

```

277     dataType = "100D"
278     hasValid = True
279     for K in diffK:
280         executeKMeans(questionTitle, K, dataType, hasValid)
281     #

```

4.2.2 Mixture of Gaussians

```

1 import tensorflow as tf
2 import numpy as np
3 import sys
4 from dataInitializer import DataInitializer
5 from utils import *
6 import datetime
7 import sys
8
9 class MixtureOfGaussians(object):
10     def __init__(self, questionTitle, K, trainData, validData, hasValid,
11                  dataType, numEpoch = 500, learningRate = 0.001):
12         """
13             Constructor
14         """
15         self.K = K # number of clusters
16         self.dataType = dataType
17         self.trainData = trainData
18         self.validData = validData
19         self.D = self.trainData[0].size # Dimension of each data
20         self.hasValid = hasValid
21         self.learningRate = learningRate
22         self.numEpoch = numEpoch
23         self.miniBatchSize = self.trainData.shape[0] # miniBatchSize is
24             # entire data size
25         self.questionTitle = questionTitle
26         self.optimizer = tf.train.AdamOptimizer(learning_rate =
27             self.learningRate, beta1=0.9, beta2=0.99, epsilon=1e-5)

```

```

25      # Execute Mixture of Gaussians
26      self.MixtureOfGaussiansMethod()
27
28  def printPlotResults(self, xAxis, yTrainErr, yValidErr, numUpdate,
29      ↳ minAssignTrain, currTrainData, clusterMean, clusterStdDeviation,
30      ↳ clusterPrior, minAssignValid):
31      figureCount = 0 # TODO: Make global
32      import matplotlib.pyplot as plt
33      if self.dataType == "2D":
34          print "mean", clusterMean
35          print "K: ", self.K
36          print "Iter: ", numUpdate
37          numTrainAssignEachClass = np.bincount(minAssignTrain)
38          numValidAssignEachClass = np.bincount(minAssignValid)
39          print "Train Assignments To Classes:", numTrainAssignEachClass
40          percentageTrainAssignEachClass =
41              ↳ numTrainAssignEachClass/float(sum(numTrainAssignEachClass))
42          print "Train Percentage Assignment To Classes:",
43              ↳ percentageTrainAssignEachClass
44          percentageValidAssignEachClass = percentageTrainAssignEachClass #
45              ↳ Initialize
46          if self.hasValid:
47              print "Valid Assignments To Classes:", numValidAssignEachClass
48              percentageValidAssignEachClass =
49                  ↳ numValidAssignEachClass/float(sum(numValidAssignEachClass))
50              print "Valid Percentage Assignment To Classes:",
51                  ↳ percentageValidAssignEachClass
52              print "prior", clusterPrior
53              print "prior.shape", clusterPrior.shape
54              print "prior Sum", np.sum(clusterPrior)
55              print "stdDeviation", clusterStdDeviation
56              print "stdDeviationShape", clusterStdDeviation.shape
57              print str(self.K) + "Lowest TrainLoss", np.min(yTrainErr)
58              print str(self.K) + "Lowest ValidLoss", np.min(yValidErr)
59
60      trainStr = "Train"

```

```

54     validStr = "Valid"
55     typeLossStr = "Loss"
56     typeScatterStr = "Assignments"
57     trainLossStr = trainStr + typeLossStr
58     validLossStr = validStr + typeLossStr
59     iterationStr = "Iteration"
60     dimensionOneStr = "D1"
61     dimensionTwoStr = "D2"
62     paramStr = "K" + str(self.K) + "Learn" + str(self.learningRate) +
63         "NumEpoch" + str(self.numEpoch)
64
65     # Train Loss
66     figureCount = figureCount + 1
67     plt.figure(figureCount)
68     title = trainStr + typeLossStr + paramStr
69     plt.title(title)
70     plt.xlabel(iterationStr)
71     plt.ylabel(typeLossStr)
72     plt.plot(np.array(xAxis), np.array(yTrainErr), label =
73             trainLossStr)
74     plt.legend()
75     plt.savefig(self.questionTitle + title + ".png")
76     plt.close()
77     plt.clf()
78
79     # Valid Loss
80     if self.hasValid:
81         figureCount = figureCount + 1
82         plt.figure(figureCount)
83         title = validStr + typeLossStr + paramStr
84         plt.title(title)
85         plt.xlabel(iterationStr)
86         plt.ylabel(typeLossStr)

```

```

87     plt.savefig(self.questionTitle + title + ".png")
88     plt.close()
89     plt.clf()
90
91     if self.dataType != "2D":
92         return
93
94     # Plot percentage in each different classes as well
95     # Scatter plot based on assignment colors
96     # Including percentage as the label
97     # Train Scatter Plot
98
99     figureCount = figureCount + 1
100    plt.figure(figureCount)
101    plt.axes()
102    title = trainStr + typeScatterStr + paramStr
103    plt.title(title)
104    plt.xlabel(dimensionOneStr)
105    plt.ylabel(dimensionTwoStr)
106    colors = ['blue', 'red', 'green', 'black', 'yellow']
107    plt.scatter(currTrainData[:, 0], currTrainData[:, 1],
108                c=minAssignTrain, s=10, alpha=0.5)
109    colors = colors[:self.K]
110    for i, j, k, l in zip(clusterMean, percentageTrainAssignEachClass,
111                           colors, clusterStdDeviation[0]):
112        plt.plot(i[0], i[1], 'kx', markersize=15, label=j, c=k)
113        circle = plt.Circle((i[0], i[1]), radius=2*l, color=k,
114                             fill=False)
115        plt.gca().add_patch(circle)
116    plt.legend()
117    plt.savefig(self.questionTitle + title + ".png")
118    plt.close()
119    plt.clf()
120
121
122    if self.hasValid:
123        # Valid Scatter Plot
124        figureCount = figureCount + 1
125        plt.figure(figureCount)

```

```

120         plt.axes()
121         title = validStr + typeScatterStr + paramString
122         plt.title(title)
123         plt.xlabel(dimensionOneStr)
124         plt.ylabel(dimensionTwoStr)
125         colors = ['blue', 'red', 'green', 'black', 'yellow']
126         plt.scatter(self.validData[:, 0], self.validData[:, 1],
127                     c=minAssignValid, s=10, alpha=0.5)
128         colors = colors[:self.K]
129         for i, j, k, l in zip(clusterMean,
130                               percentageValidAssignEachClass, colors,
131                               clusterStdDeviation[0]):
132             plt.plot(i[0], i[1], 'kx', markersize=15, label=j, c=k)
133             circle = plt.Circle((i[0], i[1]), radius=2*l, color=k,
134                                  fill=False)
135             plt.gca().add_patch(circle)
136         plt.legend()
137         plt.savefig(self.questionTitle + title + ".png")
138         plt.close()
139         plt.clf()

140     def PairwiseDistances(self, X, U):
141         """
142             input:
143                 X is a matrix of size (B x D)
144                 U is a matrix of size (K x D)
145             output:
146                 Distances = matrix of size (B x K) containing the pairwise
147                 Euclidean distances
148
149         """
150         batchSize = tf.shape(X)[0]
151         dimensionSize = tf.shape(X)[1]
152         numClusters = tf.shape(U)[0]
153         X_broadcast = tf.reshape(X, (batchSize, 1, dimensionSize))
154         sumOfSquareDistances =
155             tf.reduce_sum(tf.square(tf.subtract(X_broadcast, U)), 2)

```

```

150     return sumOfSquareDistances
151
152 def LnProbabilityXGivenZ(self, data, mean, variance):
153     sumOfSquare = self.PairwiseDistances(data, mean)
154     logLikelihoodDataGivenCluster =
155         ↳ tf.add(-tf.multiply(tf.cast(self.D,
156             ↳ tf.float32)/2.0, tf.log(tf.constant(2.0*np.pi)*variance)),
157             ↳ -tf.divide(sumOfSquare, 2.0*variance)))
158     return logLikelihoodDataGivenCluster
159
160
161 def LnProbabilityZGivenX(self, data, mean, variance, lnPriorBroad):
162     lnProbabilityXGivenZ = self.LnProbabilityXGivenZ(data, mean,
163         ↳ variance)
164     # lnPriorBroad = tf.log(tf.reshape(prior, (1, self.K)))
165     numerator = lnPriorBroad + lnProbabilityXGivenZ
166     lnProbabilityX = tf.reshape(reduce_logsumexp(numerator, 1),
167         ↳ (tf.shape(data) [0], 1))
168     lnProbabilityZGivenX = numerator - lnProbabilityX
169     return lnProbabilityZGivenX
170     # Monotonically increasing, others doesnt matter ??
171     # return numerator
172
173
174 def LnProbabilityX(self, data, mean, variance, lnPriorBroad):
175     lnProbabilityXGivenZ = self.LnProbabilityXGivenZ(data, mean,
176         ↳ variance)
177     # lnPriorBroad = tf.log(tf.reshape(prior, (1, self.K)))
178     numerator = lnPriorBroad + lnProbabilityXGivenZ
179     lnProbabilityX = tf.reshape(reduce_logsumexp(numerator, 1),
180         ↳ (tf.shape(data) [0], 1))
181     return lnProbabilityX
182
183
184 def MixtureOfGaussiansMethod(self):
185     """
186
187     Build Graph and execute in here
188     so don't have to pass variables one by one
189     Bad Coding Style but higher programmer productivity

```

```

179      """
180
181      # Build Graph
182
183      # Mean location matters a lot in convergence
184      clusterMean = tf.Variable(tf.truncated_normal([self.K, self.D],
185                                mean=-1, stddev=2.0)) # cluster centers
186
187      clusterStdDeviationConstraint =
188          tf.Variable(tf.truncated_normal([1, self.K], mean=0,
189                                stddev=0.1))
190
191      clusterVariance = tf.exp(clusterStdDeviationConstraint)
192
193      clusterStdDeviation = tf.sqrt(clusterVariance)
194
195      # Uniform intialization
196
197      clusterPriorConstraint = tf.Variable(tf.ones([1, self.K]))
198
199      logClusterConstraint = logsoftmax(clusterPriorConstraint)
200
201      clusterPrior = tf.exp(logClusterConstraint)
202
203
204      trainData = tf.placeholder(tf.float32, shape=[None, self.D],
205                                name="trainingData")
206
207
208      sumOfSquare = self.PairwiseDistances(trainData, clusterMean)
209
210      lnProbabilityXGivenZ = self.LnProbabilityXGivenZ(trainData,
211                                clusterMean, clusterVariance)
212
213      lnProbabilityX = self.LnProbabilityX(trainData, clusterMean,
214                                clusterVariance, logClusterConstraint)
215
216      loss = (tf.reduce_sum(-1.0 * lnProbabilityX))
217
218      # This is needed to decide which assignment it is
219
220      lnProbabilityZGivenX = self.LnProbabilityZGivenX(trainData,
221                                clusterMean, clusterVariance, logClusterConstraint)
222
223      probabilityZGivenX = tf.exp(lnProbabilityZGivenX)
224
225      check = tf.reduce_sum(probabilityZGivenX, 1) # Check probabilities
226
227          sum to 1
228
229      # Assign classes based on maximum posterior probability for each
230
231          data point
232
233      minAssignments = tf.argmax(lnProbabilityXGivenZ, 1) # No prior
234
235          contribution during assignment
236
237      minAssignments = tf.argmax(lnProbabilityZGivenX, 1) # Prior
238
239          contributes during assignment

```

```

204
205     #
206     ↵  -----
207     #logLikelihoodDataGivenCluster =
208     ↵  self.LnProbabilityZGivenX(trainData, clusterMean,
209     ↵  clusterStdDeviation, clusterPrior)
210     validLoss = loss # initialization
211     minValidAssignments = minAssignments #Initialization
212     if self.hasValid:
213         valid_data = tf.placeholder(tf.float32, shape=[None, self.D],
214             ↵  name="validationData")
215         validLoss = tf.reduce_sum(-1.0 *
216             ↵  self.LnProbabilityX(valid_data,
217             ↵  clusterMean, clusterVariance, logClusterConstraint))
218         validLnProbabilityZGivenX =
219             ↵  self.LnProbabilityZGivenX(valid_data, clusterMean,
220             ↵  clusterVariance, logClusterConstraint)
221         minValidAssignments = tf.argmax(validLnProbabilityZGivenX, 1)
222             ↵  # Prior contributes during assignment
223
224         train = self.optimizer.minimize(loss)
225
226         # Session
227         init = tf.global_variables_initializer()
228         sess = tf.InteractiveSession()
229         sess.run(init)
230         currEpoch = 0
231         minAssignTrain = 0
232         minAssignValid = 0
233         centers = 0
234         xAxis = []
235         yTrainErr = []
236         yValidErr = []
237         numUpdate = 0
238         step = 0
239         currTrainDataShuffle = self.trainData

```

```

231     while currEpoch < self.numEpoch:
232         np.random.shuffle(self.trainData) # Shuffle Batches
233         step = 0
234         while step * self.miniBatchSize < self.trainData.shape[0]:
235             feedDicts = {trainData:
236                         ↳ self.trainData[step * self.miniBatchSize:(step+1) * self.miniBatchSize]
237             if self.hasValid:
238                 feedDicts = {trainData:
239                             ↳ self.trainData[step * self.miniBatchSize:(step+1) * self.miniBatchSize]
240                             ↳ valid_data: self.validData}
241             _, minAssignTrain, paramClusterMean, paramClusterPrior,
242             ↳ paramClusterStdDeviation, zGivenX, checkZGivenX,
243             ↳ errTrain, errValid, minAssignValid = sess.run([train,
244             ↳ minAssignments, clusterMean, clusterPrior,
245             ↳ clusterStdDeviation, lnProbabilityZGivenX, check,
246             ↳ loss, validLoss, minValidAssignments], feed_dict =
247             ↳ feedDicts)
248             xAxis.append(numUpdate)
249             yTrainErr.append(errTrain)
250             yValidErr.append(errValid)
251             step += 1
252             numUpdate += 1
253             currEpoch += 1
254
255             if currEpoch%100 == 0:
256                 logStdOut("e: " + str(currEpoch))
257
258             # Calculate everything again without training
259             feedDicts = {trainData: self.trainData}
260
261             # No training, just gather data for valid assignments
262             if self.hasValid:
263                 feedDicts = {trainData: self.trainData, valid_data:
264                             ↳ self.validData}
265
266             minAssignTrain, paramClusterMean, paramClusterPrior,
267             ↳ paramClusterStdDeviation, zGivenX, checkZGivenX, errTrain,
268             ↳ errValid, minAssignValid = sess.run([minAssignments,
269             ↳ clusterMean, clusterPrior, clusterStdDeviation,
270             ↳ lnProbabilityZGivenX, check, loss, validLoss,
271             ↳ minValidAssignments], feed_dict = feedDicts)

```

```

254     # Count how many assigned to each class
255     currTrainDataShuffle = self.trainData
256     self.printPlotResults(xAxis, yTrainErr, yValidErr, numUpdate,
257                           ↳ minAssignTrain, currTrainDataShuffle, paramClusterMean,
258                           ↳ paramClusterStdDeviation, paramClusterPrior, minAssignValid)
259
260
261     def executeMixtureOfGaussians(questionTitle, K, dataType, hasValid,
262                                     ↳ numEpoch, learningRate):
263
264         """
265             Re-loads the data and re-randomize it with same seed anytime to ensure
266             replicable results
267
268         """
269
270         logStdOut(questionTitle)
271         print questionTitle
272
273         trainData = 0
274
275         validData = 0
276
277         # Load data with seeded randomization
278         dataInitializer = DataInitializer()
279
280         if hasValid:
281
282             trainData, validData = dataInitializer.getData(dataType, hasValid)
283
284         else:
285
286             trainData = dataInitializer.getData(dataType, hasValid)
287
288
289         # Execute algorithm
290
291         kObject = MixtureOfGaussians(questionTitle, K, trainData, validData,
292                                     ↳ hasValid, dataType, numEpoch, learningRate)
293
294         logElapsedTime(questionTitle + "K" + str(K) + "NumEpoch" +
295                      ↳ str(numEpoch))
296
297
298         # Global for logging
299
300         questionTitle = "" # Need to be global for logging to work
301         startTime = datetime.datetime.now()
302
303         figureCount = 1 # To not overwrite existing pictures
304
305
306         def logStdOut(message):
307
308             # Temporary print to std out

```

```

284     sys.stdout = sys.__stdout__
285     print message
286     # Continue editing same file
287     sys.stdout = open("result" + questionTitle + ".txt", "a")
288
289 def logElapsedTime(message):
290     ''' Logs the elapsedTime with a given message '''
291     global startTime
292     endTime = datetime.datetime.now()
293     elapsedTime = endTime - startTime
294     hours, remainder = divmod(elapsedTime.seconds, 3600)
295     minutes, seconds = divmod(remainder, 60)
296     totalDays = elapsedTime.days
297     timeStr = str(message) + ': Days: ' + str(totalDays) + " hours: " +
298             str(hours) + ' minutes: ' + str(minutes) + ' seconds: ' +
299             str(seconds)
300
301     logStdOut(timeStr)
302     startTime = datetime.datetime.now()
303
304 if __name__ == "__main__":
305     print "ECE521 Assignment 3: Unsupervised Learning: GaussianCluster"
306     """
307     # Gaussian Cluster Model
308     questionTitle = "2.1.2" # Implemented function
309     questionTitle = "2.1.3" # Implemented FUnction
310     print "ECE521 Assignment 3: Unsupervised Learning: Mixture of
311         Gaussian"
312     questionTitle = "2.2.2"
313     dataType = "2D"
314     hasValid = False # No validation data
315     K = 3
316     numEpoch = 200
317     learningRate = 0.1
318     # Note: Loss will be higher since no validation data
319     executeMixtureOfGaussians(questionTitle, K, dataType, hasValid,
320         numEpoch, learningRate)

```

```

316     # '''
317
318     '''
319     questionTitle = "2.2.3"
320     dataType = "2D"
321     hasValid = True
322     diffK = [1, 2, 3, 4, 5]
323     numEpoch = 200
324     learningRate = 0.1
325     for K in diffK:
326         executeMixtureOfGaussians(questionTitle, K, dataType, hasValid,
327             ↳ numEpoch, learningRate)
327     # '''
328
329     questionTitle = "2.2.4.2"
330     dataType = "100D"
331     hasValid = True
332     diffK = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
333     numEpoch = 150
334     learningRate = 0.1
335     for K in diffK:
336         executeMixtureOfGaussians(questionTitle, K, dataType, hasValid,
337             ↳ numEpoch, learningRate)
337     # '''

```

4.2.3 Helper Functions

```

1 import numpy as np
2 import tensorflow as tf
3
4 class DataInitializer(object):
5     def __init__(self):
6         """
7             Data Initializer code

```

```

8      """
9
10     self.data2D = np.load("data2D.npy")
11     self.data100D = np.load("data100D.npy")
12     self.tinyData = np.load("tinymnist.npz")
13
14
15     def getData(self, dataType, hasValid):
16
17         """
18
19         Returns the train, validation, and test data for 2D dataset
20         that has already been randomized
21
22         """
23
24         data = 0
25
26         if dataType == "2D":
27
28             data = self.data2D
29
30         elif dataType == "100D":
31
32             data = self.data100D
33
34         if hasValid:
35
36             trainData, validData = self.splitDataRandom(data, hasValid)
37
38             """
39
40             print 'data' + str	dataType + ' All: (number of data, data
41             ↵ dimension):', data.shape
42
43             print 'data' + str	dataType + ' Train: (number of data, data
44             ↵ dimension):', trainData.shape
45
46             print 'data' + str	dataType + ' Valid: (number of data, data
47             ↵ dimension):', validData.shape
48
49             """
50
51             return trainData, validData
52
53         trainData = self.splitDataRandom(data, hasValid)
54
55             """
56
57             print 'data' + str	dataType + ' All: (number of data, data
58             ↵ dimension):', data.shape
59
60             print 'data' + str	dataType + ' Train: (number of data, data
61             ↵ dimension):', trainData.shape
62
63             """
64
65             return trainData
66
67
68     def getTinyData(self):

```

```

39      """
40
41     Returns the train, validation, and test data for tinyMnist dataset
42     that has already been randomized.
43     """
44
45     trainData, trainTarget = self.tinyData["x"], self.tinyData["y"]
46     validData, validTarget = self.tinyData["x_valid"], self.tinyData
47         ["y_valid"]
48     testData, testTarget = self.tinyData["x_test"],
49         self.tinyData["y_test"]
50
51     print trainData.shape
52     print trainTarget.shape
53     print validData.shape
54     print validTarget.shape
55     print testData.shape
56     print testTarget.shape
57
58     # TODO: Randomize data?
59
60     return trainData, trainTarget, validData, validTarget, testData,
61         testTarget
62
63
64
65
66
67
68
69
70

```

```

71         return trainData, validData
72     else:
73         trainData = data[randIdx[::]]
74         return trainData

```

4.2.4 FactorAnalysis

```

1 import tensorflow as tf
2 import numpy as np
3 import sys
4 from dataInitializer import DataInitializer
5 from utils import *
6 import datetime
7 import sys
8 import matplotlib.pyplot as plt
9
10 class FactorAnalysis(object):
11     def __init__(self, questionTitle, K, trainData, trainTarget,
12                  validData, validTarget, testData, testTarget, numEpoch = 500,
13                  learningRate = 0.001):
14
15         """
16             Constructor
17         """
18
19         self.K = K # number of factors
20         self.trainData = trainData
21         self.trainTarget = trainTarget
22         self.validData = validData
23         self.validTarget = validTarget
24         self.testData = testData
25         self.testTarget = testTarget
26         self.D = self.trainData[0].size # Dimension of each data
27         self.learningRate = learningRate
28         self.numEpoch = numEpoch
29         self.miniBatchSize = self.trainData.shape[0] # miniBatchSize is
30             → entire data size

```

```

26     self.questionTitle = questionTitle
27
28     self.optimizer = tf.train.AdamOptimizer(learning_rate =
29         ↳ self.learningRate, beta1=0.9, beta2=0.99, epsilon=1e-5)
30
31     # Execute Factor Analysis
32
33     self.FactorAnalysisMethod()
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58

```

`self.questionTitle = questionTitle

self.optimizer = tf.train.AdamOptimizer(learning_rate =
 ↳ self.learningRate, beta1=0.9, beta2=0.99, epsilon=1e-5)

Execute Factor Analysis

self.FactorAnalysisMethod()

def saveGrayscaleImage(self, image, width=8, height=8, imageName=""):

 """ This plots an image given its width and height

 image is the image to plot

 imageName is the name of the image to save as.

 """

 figureCount = 0 # TODO: Make global

 plt.figure(figureCount)

 # Draw each figures (8, 8)

 currImage = image[:]

 currImage = np.reshape(currImage, (width,height))

 plt.imshow(currImage, interpolation="nearest", cmap="gray")

 plt.savefig(str(imageName) + ".png")

def printTensor(self, tensorToPrint, trainData, message=""):

 init = tf.global_variables_initializer()

 sess = tf.InteractiveSession()

 sess.run(init)

 printDict = {trainData: self.trainData}

 valueToPrint = sess.run([tensorToPrint], feed_dict = printDict)

 print message, valueToPrint

 print "shape", np.array(valueToPrint).shape

 plt.close()

 plt.clf()

def printPlotResults(self, xAxis, yTrainErr, yValidErr, numUpdate,
 ↳ currTrainDataShuffle, factorMean, factorCovariance,
 ↳ factorWeights):

 figureCount = 0 # TODO: Make global

 import matplotlib.pyplot as plt

 print "mean", factorMean`

```

59     print "K: ", self.K
60
61     print "Iter: ", numUpdate
62
63     print "mean", factorMean
64
65     print "meanShape", factorMean.shape
66
67     print "CoVariance", factorCovariance
68
69     print "CoVarianceShape", factorCovariance.shape
70
71     print "Lowest TrainLoss", np.min(yTrainErr)
72
73     print "Lowest ValidLoss", np.min(yValidErr)

74
75
76
77     # Train Loss
78
79     figureCount = figureCount + 1
80
81     plt.figure(figureCount)
82
83     title = trainStr + typeLossStr + paramString
84
85     plt.title(title)
86
87     plt.xlabel(iterationStr)
88
89     plt.ylabel(typeLossStr)
90
91     plt.plot(np.array(xAxis), np.array(yTrainErr), label =
92             trainLossStr)
93
94     plt.legend()
95
96     plt.savefig(self.questionTitle + title + ".png")
97
98     plt.close()
99
100    plt.clf()

101
102    # Valid Loss
103
104    figureCount = figureCount + 1
105
106    plt.figure(figureCount)

```

```

93     title = validStr + typeLossStr + paramString
94
95     plt.title(title)
96
97     plt.xlabel(iterationStr)
98
99     plt.ylabel(typeLossStr)
100
101    plt.plot(np.array(xAxis), np.array(yValidErr), label =
102              ↳ validLossStr)
103
104    plt.legend()
105
106    plt.savefig(self.questionTitle + title + ".png")
107
108    plt.close()
109
110    plt.clf()
111
112
113
114
115
116
117
118
119
120
121
122
123

```

Weight Images

```

103
104    for i in xrange(self.K):
105        imageTitle = self.questionTitle + "WeightDim" + str(i) + "K" +
106                      ↳ str(self.K) + "NumEpoch" + str(self.numEpoch)
107
108        self.saveGrayscaleImage(factorWeights[:, i], 8, 8, imageTitle)
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123

```

def FactorAnalysisMethod(self):

```

117
118
119
120
121
122
123

```

Build Graph and execute in here

so don't have to pass variables one by one

Bad Coding Style but higher programmer productivity

```

117
118
119
120
121
122
123

```

Build Graph

```

117
118
119
120
121
122
123

```

print "trainShape", self.trainData.shape

print "validShape", self.validData.shape

print "testShape", self.testData.shape

```

117
118
119
120
121
122
123

```

factorMean = tf.Variable(tf.truncated_normal([self.D]))

factorWeights = tf.Variable(tf.truncated_normal([self.D, self.K]))

factorStdDeviationConstraint =

```

117
118
119
120
121
122
123

```

↳ tf.Variable(tf.truncated_normal([self.D]))

factorTraceCoVariance = tf.exp(factorStdDeviationConstraint)

```

124     factorCovariance = tf.diag(factorTraceCoVariance) +
125         ↳ tf.matmul(factorWeights, tf.transpose(factorWeights))
126     factorCovarianceInv = tf.matrix_inverse(factorCovariance)
127     logDeterminantCovariance = 2.0 *
128         ↳ tf.reduce_sum(tf.log(tf.diag_part(tf.cholesky(factorCovariance)) ))
129
130     # Train Loss
131
132     xDeductU = tf.subtract(trainData, factorMean)
133     xDeductUTranspose = tf.transpose(xDeductU)
134     total = tf.trace(tf.matmul(tf.matmul(xDeductU,
135         ↳ factorCovarianceInv), xDeductUTranspose))
136     logProbability = -0.5 * (total + self.D * tf.log(2.0 * np.pi) +
137         ↳ logDeterminantCovariance)
138     loss = tf.negative(logProbability)
139
140
141     validDeductU = tf.subtract(validData, factorMean)
142     validDeductUTranspose = tf.transpose(validDeductU)
143     validTotal = tf.trace(tf.matmul(tf.matmul(validDeductU,
144         ↳ factorCovarianceInv), validDeductUTranspose))
145     validLogProbability = -0.5 * (validTotal + self.D * tf.log(2.0 *
146         ↳ np.pi) + logDeterminantCovariance)
147     validLoss = tf.negative(validLogProbability)
148
149     train = self.optimizer.minimize(loss)
150
151     # Session
152
153     init = tf.global_variables_initializer()
154     sess = tf.InteractiveSession()
155     sess.run(init)
156     currEpoch = 0
157     minAssignTrain = 0
158     minAssignValid = 0
159     centers = 0
160     xAxis = []
161     yTrainErr = []
162     yValidErr = []
163     numUpdate = 0

```

```

154     step = 0
155     currTrainDataShuffle = self.trainData
156     while currEpoch < self.numEpoch:
157         np.random.shuffle(self.trainData) # Shuffle Batches
158         step = 0
159         while step * self.miniBatchSize < self.trainData.shape[0]:
160             feedDicts = {trainData:
161                         ↳ self.trainData[step * self.miniBatchSize:(step+1) * self.miniBatchSize]
162                         ↳ validData: self.validData}
163             _, errTrain, errValid, paramFactorMean,
164             ↳ paramFactorCovariance, paramFactorWeights =
165             ↳ sess.run([train, loss, validLoss, factorMean,
166             ↳ factorCovariance, factorWeights], feed_dict =
167             ↳ feedDicts)
168             xAxis.append(numUpdate)
169             yTrainErr.append(errTrain)
170             yValidErr.append(errValid)
171             step += 1
172             numUpdate += 1
173             currEpoch += 1
174
175             if currEpoch%100 == 0:
176                 logStdOut("e: " + str(currEpoch))
177
178             # Calculate everything again without training
179             feedDicts = {trainData: self.trainData, validData: self.validData}
180             errTrain, errValid, paramFactorMean, paramFactorCovariance,
181             ↳ paramFactorWeights = sess.run([loss, validLoss, factorMean,
182             ↳ factorCovariance, factorWeights], feed_dict = feedDicts)
183
184             # Count how many assigned to each class
185             currTrainDataShuffle = self.trainData
186             self.printPlotResults(xAxis, yTrainErr, yValidErr, numUpdate,
187             ↳ currTrainDataShuffle, paramFactorMean, paramFactorCovariance,
188             ↳ paramFactorWeights)
189
190
191     def executeFactorAnalysis(questionTitle, K, numEpoch, learningRate):
192         """
193

```

```

180     Re-loads the data and re-randomize it with same seed anytime to ensure
181     ↵  replicable results
182
183     """
184
185     logStdOut(questionTitle)
186     print questionTitle
187     trainData = 0
188     validData = 0
189     # Load data with seeded randomization
190     dataInitializer = DataInitializer()
191     trainData, trainTarget, validData, validTarget, testData, testTarget =
192         ↵  dataInitializer.getTinyData()
193
194     # Execute algorithm
195     kObject = FactorAnalysis(questionTitle, K, trainData, trainTarget,
196         ↵  validData, validTarget, testData, testTarget, numEpoch,
197         ↵  learningRate)
198     logElapsedTime(questionTitle + "K" + str(K) + "NumEpoch" +
199         ↵  str(numEpoch))
200
201     # Global for logging
202     questionTitle = "" # Need to be global for logging to work
203     startTime = datetime.datetime.now()
204     figureCount = 1 # To not overwrite existing pictures
205
206     def logStdOut(message):
207         # Temporary print to std out
208         # sys.stdout = sys.__stdout__ # TODO: Uncomment this
209         print message
210         # Continue editing same file
211         # sys.stdout = open("result" + questionTitle + ".txt", "a") #TODO:
212             ↵  Uncomment this
213
214     def logElapsedTime(message):
215         """ Logs the elapsedTime with a given message """
216         global startTime
217         endTime = datetime.datetime.now()

```

```

210     elapsedTime = endTime - startTime
211     hours, remainder = divmod(elapsedTime.seconds, 3600)
212     minutes, seconds = divmod(remainder, 60)
213     totalDays = elapsedTime.days
214     timeStr = str(message) + ': Days: ' + str(totalDays) + " hours: " +
215             str(hours) + ' minutes: ' + str(minutes) + ' seconds: ' +
216             str(seconds)
217     logStdOut(timeStr)
218     startTime = datetime.datetime.now()
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
```