ECE421 - Winter 2021
University of Toronto

Assignment 3:
Unsupervised Learning and Probabilistic Models

Due date: Monday, April 5, 2021

Submission: Submit both your report (a single PDF file) and all codes on Quercus.

Objectives:

In this assignment, you will implement learning and inference procedures for some of the probabilistic models described in class, apply your solutions to some simulated datasets, and analyze the results.

General Note:

- Full points are given for complete solutions, including justifying the choices or assumptions you made to solve each question. A written report should be included in the final submission.
- Programming assignments are to be solved and submitted **individually**. You are encouraged to discuss the assignment with other students, but you must solve it on your own.
- Please ask all questions related to this assignment on Piazza, using the tag `assignment3`.
- There are 3 starter files attached, `helper.py`, `starter_kmeans.py` and `starter_gmm.py` which will help you with your implementation.

1 K-means [9 pt.]

K-means clustering is one of the most widely used data analysis algorithms. It is used to summarize data by discovering a set of data prototypes that represent clusters of data. The data prototypes are usually referred to as cluster centers. Usually, K-means clustering proceeds by alternating between assigning data points to clusters and then updating the cluster centers. In this assignment, we will investigate a different learning algorithm that directly minimizes the K-means clustering loss function.

1.1 Learning K-means

The K cluster centers can be thought of as K, D-dimensional parameter vectors and we can place them in a $K \times D$ parameter matrix $\boldsymbol{\mu}$, where the k^{th} row of the matrix denotes the k^{th} cluster center $\boldsymbol{\mu}_k$. The goal of K-means clustering is to learn $\boldsymbol{\mu}$ such that it minimizes the loss function, $\mathcal{L}(\boldsymbol{\mu}) = \sum_{n=1}^{N} \min_{k=1}^{K} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|_2^2$, where N is the number of training observations.

Even though the loss function is not smooth due to the "min" operation, one may still be able to find its solutions through iterative gradient-based optimization. The "min" operation leads to discontinuous derivatives, in a way that is similar to the effect of the ReLU activation function, but nonetheless, a good gradient-based optimizer can work effectively.

1. Implement the `distanceFunc()` function in `starter_kmeans.py` file to calculate the squared pairwise distance for all pair of N data points and K clusters.

```
def distanceFunc(X, MU):
# Inputs
    # X: is an NxD matrix (N observations and D dimensions)
    # MU: is an KxD matrix (K means and D dimensions)
# Outputs
    # pair_dist: is the squared pairwise distance matrix (NxK)
```

   Hint: To properly use broadcasting, you can first add a dummy dimension to `X`, by using `tf.expand_dims` or `torch.unsqueeze`, so that its new shape becomes `(N, 1, D)`.

   For the dataset `data2D.npy`, set $K = 3$ and find the K-means clusters $\boldsymbol{\mu}$ by minimizing the $\mathcal{L}(\boldsymbol{\mu})$ using the gradient descent optimizer. The parameters $\boldsymbol{\mu}$ should be initialized by sampling from the standard normal distribution. Include a plot of the loss vs the number of updates.

   Use the Adam optimizer for this assignment with the following hyper-parameters: `learning_rate=0.1`, `beta1=0.9`, `beta2=0.99`, `epsilon=1e-5`. The learning should converge within a few hundred updates.

2. Hold out 1/3 of the data for validation, and for each value of $K = 1, 2, 3, 4, 5$:

   (a) Train a K-means model.

(b) Include a 2D scatter plot of training data points colored by their cluster assignments.

(c) Compute and report the percentage of the training data points belonging to each of the K clusters. (include this information in the legend of the scatter plot.)

(d) Compute and report the loss function over validation data. (include this in the caption of the scatter plot.)

Based on the scatter plots, comment on the best number of clusters to use.

2 Mixtures of Gaussians [16 pt.]

Mixtures of Gaussians (MoG) can be interpreted as a probabilistic version of K-means clustering. For each data vector, MoG uses a latent variable z to represent the cluster assignment and uses a joint probability model of the cluster assignment variable and the data vector: $P(\mathbf{x}, z) = P(z)P(\mathbf{x} \mid z)$. For N iid training cases, we have $P(\mathbf{X}, \mathbf{z}) = \prod_{n=1}^{N} P(\mathbf{x}_n, z_n)$. The Expectation-Maximization (EM) algorithm is the most commonly used technique to learn a MoG. Like the standard K-means clustering algorithm, the EM algorithm alternates between updating the cluster assignment variables and the cluster parameters. What makes it different is that instead of making hard assignments of data vectors to cluster centers (the "min" operation above), the EM algorithm computes probabilities for different cluster centers, $P(z|\mathbf{x})$. These are computed from $P(z = k|\mathbf{x}) = P(\mathbf{x}, z = k)/\sum_{j=1}^{K} P(\mathbf{x}, z = j)$.

While the Expectation-Maximization (EM) algorithm is typically the go-to learning algorithm to train MoG and is guaranteed to converge to a local optimum, it suffers from slow convergence. In this assignment, we will explore a different learning algorithm that makes use of gradient descent.

2.1 The Gaussian cluster mode [7 pt.]

Each of the K mixture components in the MoG model occurs with probability $\pi^k = P(z = k)$. The data model is a multivariate Gaussian distribution centered at the cluster mean (data center) $\boldsymbol{\mu}^k \in \mathbb{R}^D$. We will consider a MoG model where it is assumed that for the multivariate Gaussian for cluster k, different data dimensions are independent and have the same standard deviation, σ^k.

1. Use the K-means distance function `distanceFunc` implemented in 1.1 to implement `log_GaussPDF` function by computing the log probability density function for cluster k: $\log \mathcal{N}(\mathbf{x}\,;\, \boldsymbol{\mu}^k, {\sigma^k}^2)$ for all pair of N data points and K clusters. Include the snippets of the Python code.

2. Write a vectorized Tensorflow Python function that computes the log probability of the cluster variable z given the data vector $\mathbf{x}$: $\log P(z|\mathbf{x})$. The log Gaussian pdf function implemented above should come in handy. The implementation should use the function `reduce_logsumexp()` provided in the helper functions file. Include the snippets of the Python code and comment on why it is important to use the log-sum-exp function instead of using `tf.reduce_sum`.

2.2 Learning the MoG [9 pt.]

The marginal data likelihood for the MoG model is as follows (here "marginal" refers to summing over the cluster assignment variables):

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

The loss function we will minimize is the negative log likelihood $\mathcal{L}(\boldsymbol{\mu}, \sigma, \pi) = -\log P(\mathbf{X})$. The maximum likelihood estimate (MLE) is a set of the model parameters $\boldsymbol{\mu}, \sigma, \pi$ that maximize the log likelihood or, equivalently, minimize the negative log likelihood.

1. Implement the loss function using log-sum-exp function and perform MLE by directly optimizing the log likelihood function using gradient descent.

   Note that the standard deviation has the constraint of $\sigma \in [0, \infty)$. One way to deal with this constraint is to replace σ^2 with $\exp(\phi)$ in the math and the software, where ϕ is an unconstrained parameter. In addition, π has a simplex constraint, that is $\sum_k \pi^k = 1$. We can again replace this constrain with unconstrained parameter ψ through a softmax function $\pi^k = \exp(\psi^k) / \sum_{k'} \exp(\psi^{k'})$. A log-softmax function, `logsoftmax`, is provided for convenience in the helper functions file.

   For the dataset `data2D.npy`, set $K = 3$ and report the best model parameters it has learned. Include a plot of the loss vs the number of updates.

2. Hold out 1/3 of the data for validation, and for each value of $K = 1, 2, 3, 4, 5$:

   (a) Train a MoG model.

   (b) Include a 2D scatter plot of training data points colored by their cluster assignments.

   (c) Compute and report the percentage of the training data points belonging to each of the K clusters. (include this information in the legend of the scatter plot.)

   (d) Compute and report the loss function over validation data. (include this in the caption of the scatter plot.)

   Explain which value of K is best, based on the validation loss.

3. Run both the K-means and the MoG learning algorithms on `data100D.npy` for $K = \{5, 10, 15, 20, 30\}$ (Hold out 1/3 of the data for validation). Comment on how many clusters you think are within the dataset by looking at the MoG validation loss and K-means validation loss. Compare the learnt results of K-means and MoG.