Deep Clustering

Khai Nguyen

khainguyen@temple.edu

CIS 5525 – Neural Computation

Project description

The goal of this project is to get you familiar with one deep learning platform and use it to solve one of the most fundamental problems, data clustering, with the help of Autoencoders to map the data into a latent space so that clustering purity can be improved.

Data set.

a) Toy data set: mixture of **three Gaussians data set**. You can use codes like the following

X = torch.cat((torch.randn(2000, 2),torch.randn(1500, 2)+13,torch.mm(torch.randn(1000, 2)+6,3*(torch.rand(2,2)-0.2))),0)

b) MNIST data sets. We will not use the image structure although this is image data set. Instead we will simply treat it as vectors. The data sets can be found http://yann.lecun.com/exdb/mnist/

Tasks

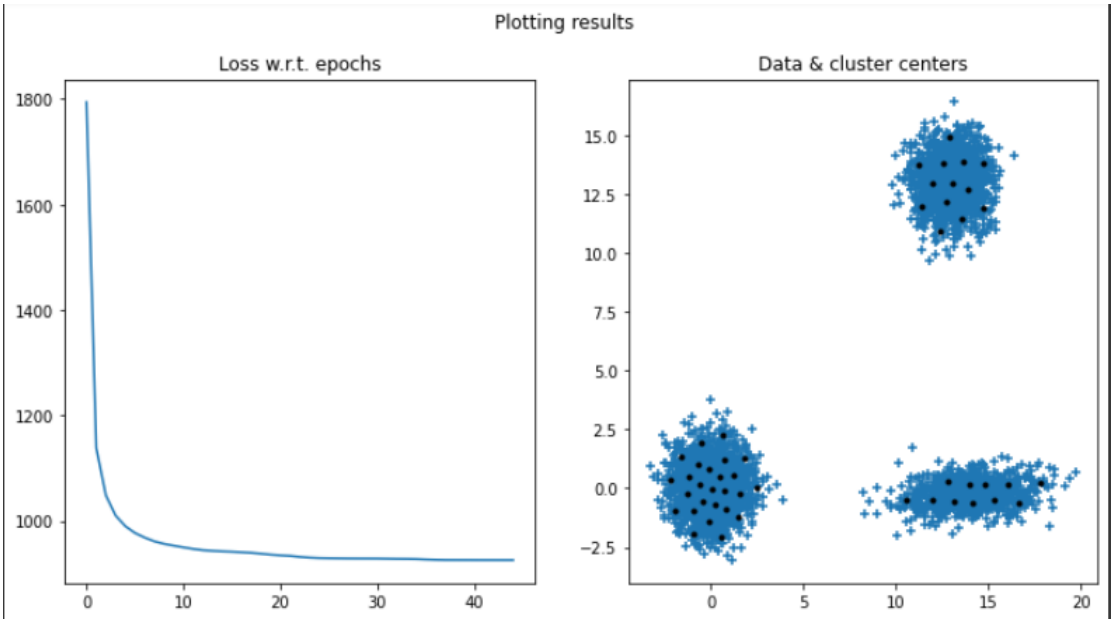
1. **Implement the discrete k-means algorithm on the toy data set.**
   Plot the loss function curve over epochs, and also the cluster centers.
   The goal is just to find good centroids with corresponding assignments for each sample.

$$\mu^*, c^* = \arg\min_{\mu,c} \sum_{i=1}^{m} \sum_{k=1}^{K} 1\{c_i = k\} ||x_i - \mu_k||^2$$

$$\mathcal{L} = \sum_{i=1}^{m} \sum_{k=1}^{K} 1\{c_i = k\} ||x_i - \mu_k||^2$$
$$= \sum_{i=1}^{m} ||x^i - \mu_{c^i}||^2$$

Procedure:

- Choose the number of clusters(K) and obtain the data points
- Place the centroids c_1, c_2, ... c_k randomly
- Repeat steps 4 and 5 until convergence or until the end of a fixed number of iterations
- for each data point x_i:
  - find the nearest centroid(c_1, c_2 .. c_k)
  - assign the point to that cluster
- for each cluster j = *1..k*
  - new centroid = mean of all points assigned to that cluster



By looking at the loss function for KMeans, we can see that this is a non-convex function. That means we cannot find the global optimal μ and c. We can only find a local optimum of them.

2. Implement differentiable k-means clustering with k = 10 directly on the 784-dimensional images (training set of MNIST 60K images). Plot the **evolution of the clustering accuracy** over epochs, and report the **average clustering accuracy** over 10 random initialized runs.
3. **Deep clustering: Perform k-means clustering on the bottleneck layer of an auto-encoder.**

   Due to difficulty in designing a combined loss function for simultaneous train both AE and differentiable KMeans, the approach chosen for this task was do some initial training for AE to obtain decent weights for the encoder of AE, then the encoder weights will be further trained *together with* the probabilistic KMeans clustering layer.

   i. Architecture:

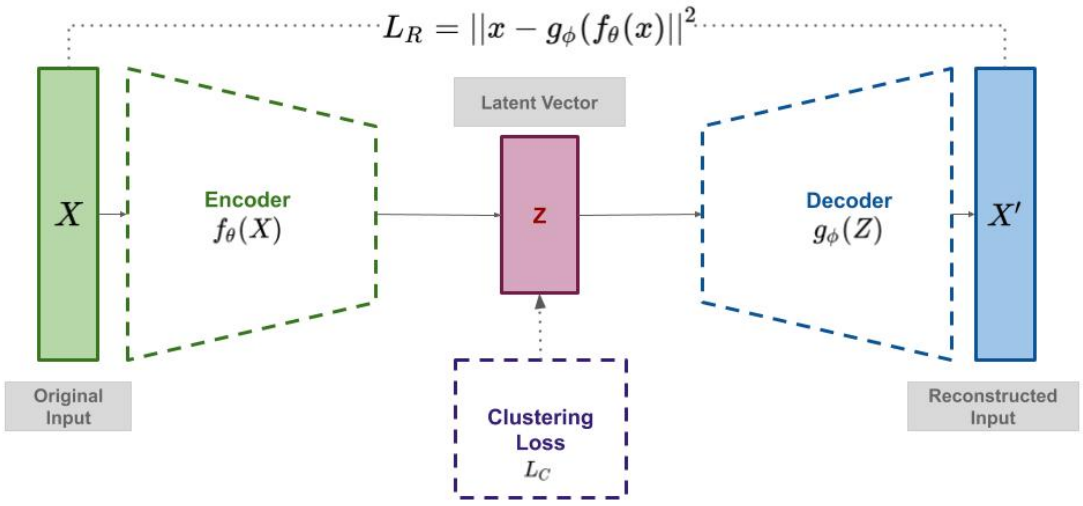


Figure 1. Architecture of AE + KMeans

As shown, the architecture of the combine approach involves a Autoencoder that learns the representation of the data in the latent layer Z, and with the help of clustering on the latent layer, we achieve a clustering method suitable for high-dimensional input datasets.

ii. Objective function:

On the latent layer, we want to iteratively refine the clusters by matching the probabilistic assignment (from the cluster indicator matrix W) to an auxiliary **target distribution *p_ij***. The choice of probabilistic assignment is Stochastic Neighbor Embedding (SNE) due to its effectiveness in converting the high-dimensional Euclidean distances between datapoints into conditional probabilities that represent similarities. SNE also have the property where nearby datapoints possess higher the conditional probability.

With ***q_ij*** being the probability of assigning sample *i* to cluster *j,* and where *z_i* is the embedding of *x_i* in the latent layer.

$$q_{ij} = \frac{(1 + \|z_i - \mu_j\|^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + \|z_i - \mu_{j'}\|^2/\alpha)^{-\frac{\alpha+1}{2}}},$$

α is the degree of freedom of the Student's t-distribution. Here our choice of α = 1.

Then we also need a target distribution *p_ij* to reduce *q_ij* towards. An approach for this is doing self-training where an initial classifier and an unlabeled dataset is taken, then the dataset is labeled with the classifier in order to train on its own high confidence predictions. Now because ***q_i*** are soft assignments, it is more natural to choose *p_i* as soft probabilistic targets.

One choice is:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij'}^2 / f_{j'}},$$

where $f_j = \sum_i q_{ij}$ are soft cluster frequencies. Then the training is done by minimizing the Kullback-Leibler divergence loss between *p_i* & *q_i*

$$L = \mathrm{KL}(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

iii. Gradient decent:

After setting the objective function as above, gradient descent is taken w.r.t. the model parameters. We choose μ_i and z_i since the gradient on z_i can be passed down to the AE encoder to compute the gradient for AE w.r.t the weights $\boldsymbol{w}$ of AE layers. The optimizer choice here is Stochastic Gradient Descent (SGD) with momentum.

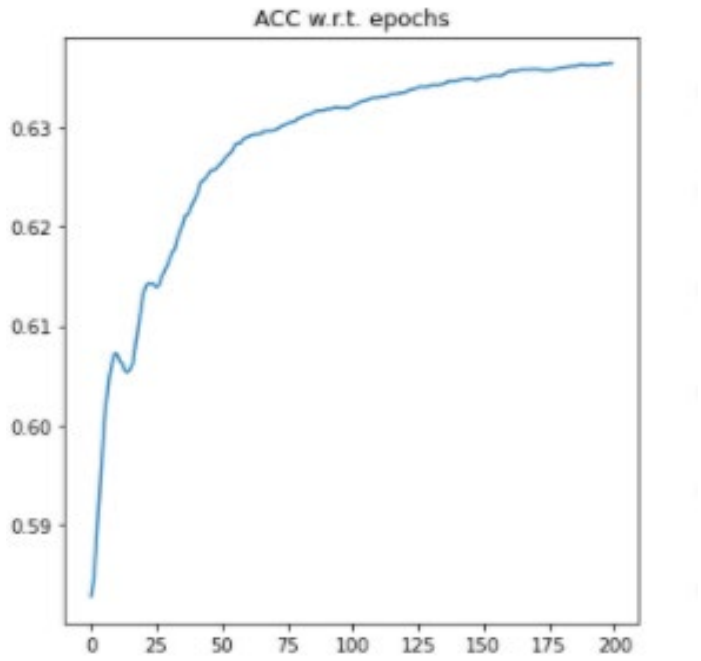
iv. Experiment

Different choices of number of layers in AE; nonlinear activation functions, and dimension of the latent layers are tested and reported in the table below.

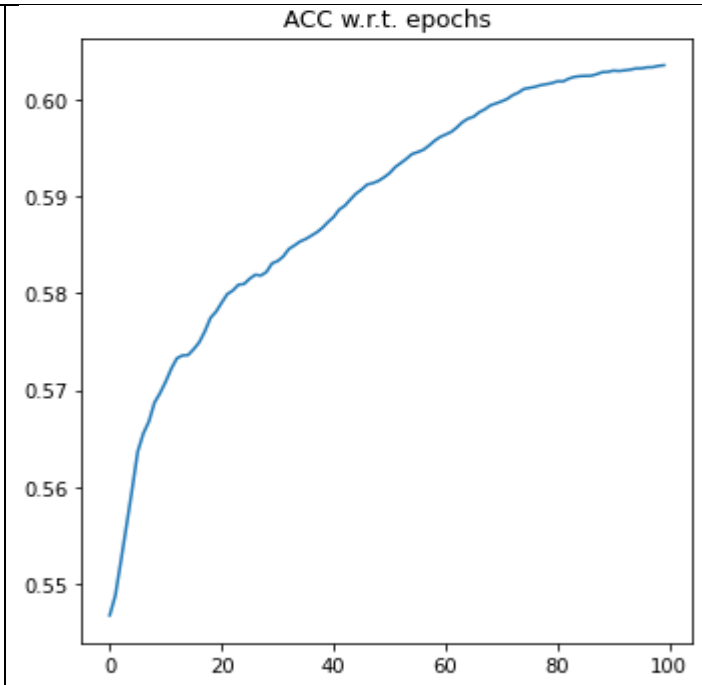
AE-encoder layer	**Activation**	**Latent dim**	**ACC**
d–128–10	ReLU (encoder & decoder)	5	0.63644
d–128–10	ReLU (encoder & decoder)	10	0.60617
d–128–10	Sigmoid (encoder) - ReLU (decoder)	10	0.57452
d–512–256–128–64–10	**ReLU (encoder & decoder)**	**10**	**0.81947**
d–512–256–128–64–10	Sigmoid (encoder) - ReLU (decoder)	10	0.11252

Table 2. Clustering Accuracy based on AE encoder Layers/Activation Functions/Latent Dimensions. Aside from ReLU and Sigmoid, SELU and CELU can be alternatives for ReLU as activation choice.

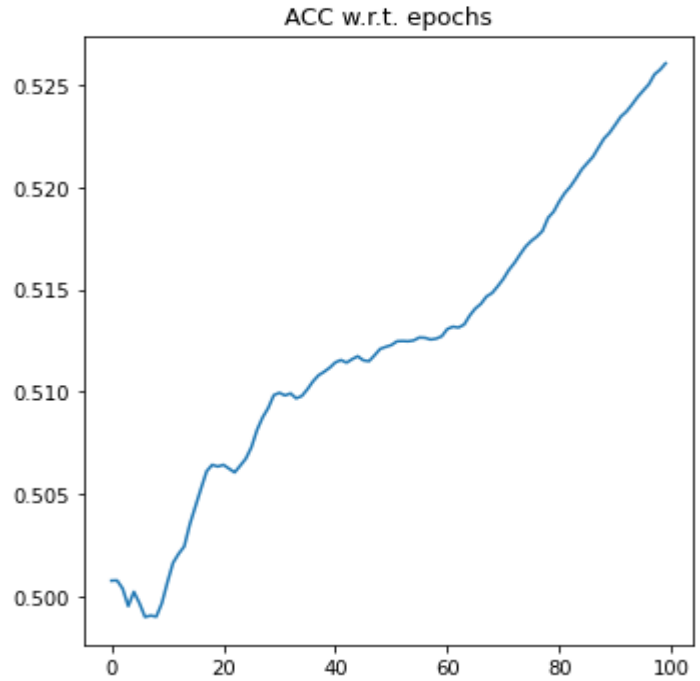
Table 3. ACC based on epochs for each choices



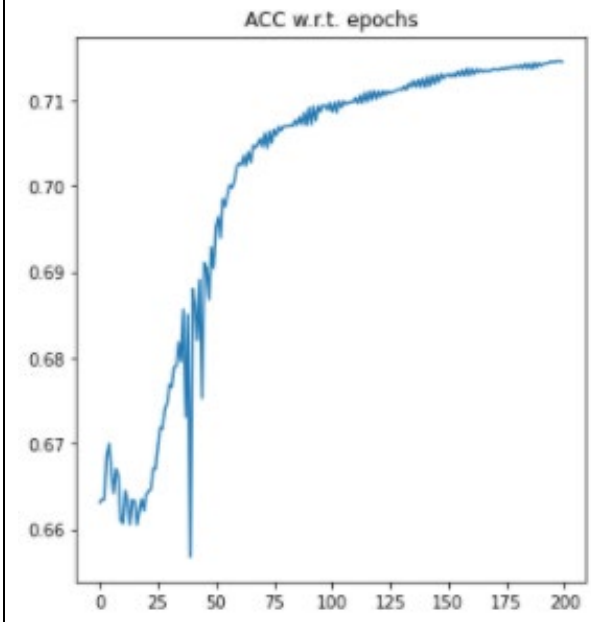
AE layer = (d–128–10), ReLU (encoder & decoder), dim=5



AE layer = (d–128–10), ReLU (encoder & decoder), dim=10

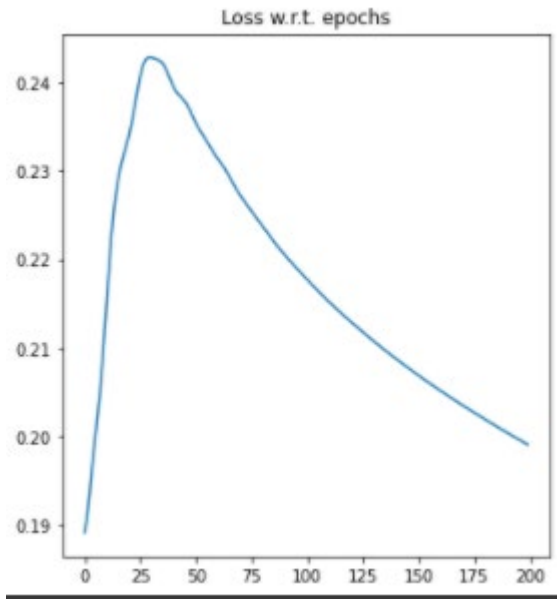


AE layer = (d–128–10), Sigmoid (encoder) - ReLU (decoder), dim=10

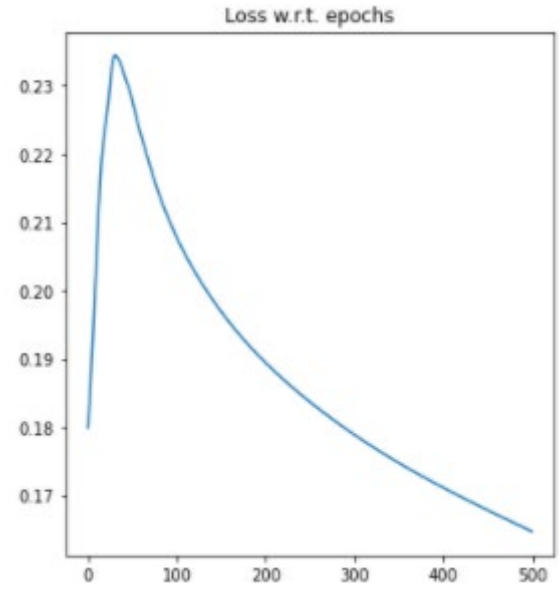


*AE layer = **(d–512–256–128–64–10)**, ReLU (encoder & decoder), dim=10*

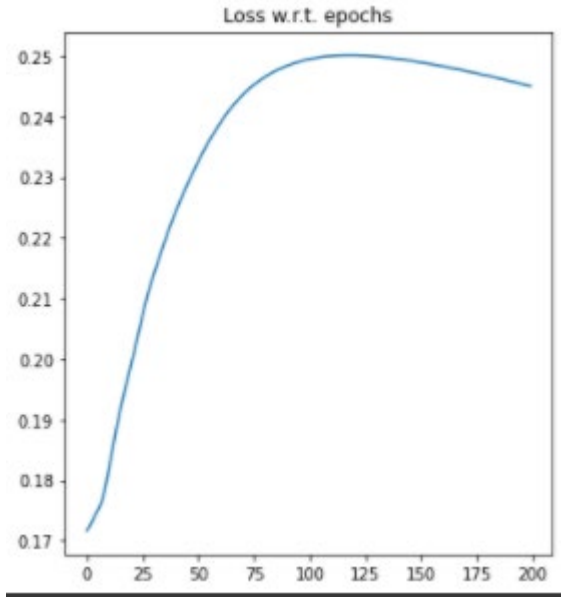
Table 4. Loss based on epochs for each choices



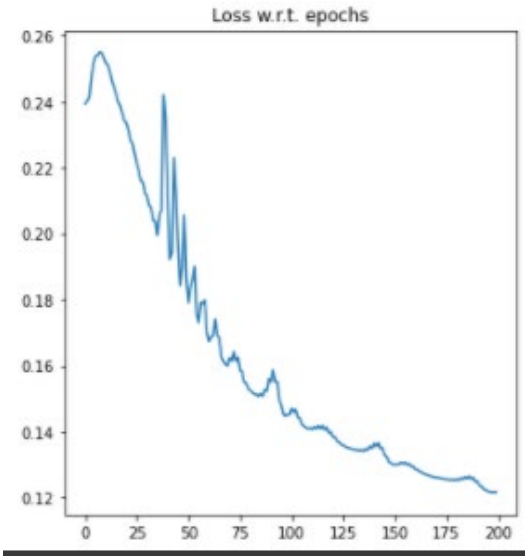
AE layer = (d–128–10), ReLU (encoder & decoder, dim=5



AE layer = (d–128–10), ReLU (encoder & decoder), dim=10



AE layer = (d–128–10), Sigmoid (encoder) - ReLU (decoder), dim=10



*AE layer = (**d–512–256–128–64–10**), ReLU (encoder & decoder), dim=10*

The best configuration experimented was d–512–256–128–64–10 for the AE layers, ReLU (encoder & decoder) activation function, dim=10 for latent layer, with the resulting clustering accuracy ACC of 0.81947.

v. Cluster visualization

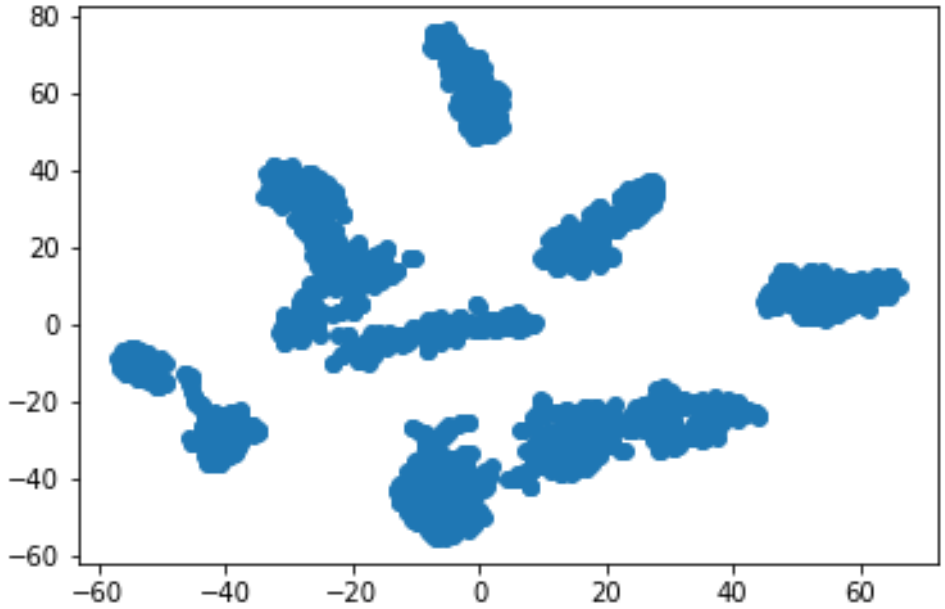


*Table 5. t-SNE visualization of clusters obtained with AE layer = **(d–512–256–128–64–10)**, **ReLU** (encoder & decoder), **dim=10***

vi. Discussion

- Clearly, using traditional KMeans to cluster on high dimensional data input is computationally expensive and ineffective. Therefore, our combine approach is useful in such situations with the help of AE layer.
- The combine AE + differentiable K-means model has linear complexity in the number of data points which allows it to scale to large datasets.

- **Unsupervised Clustering Accuracy (ACC)**
  ACC is the unsupervised equivalent of classification accuracy. ACC differs from the usual accuracy metric such that it uses a mapping function m to find the best mapping between the cluster assignment output c of the algorithm with the ground truth y. This mapping is required because an unsupervised algorithm may use a different label than the actual ground truth label to represent the same cluster.

$$ACC = max_m \frac{\sum_{i=1}^{n} 1\{y_i = m(c_i)\}}{n}$$

  Aside from ACC, we also have Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI) as the performance evaluation metrics.

- **References**
  https://arxiv.org/pdf/1808.07292.pdf
  https://arxiv.org/pdf/1511.06335.pdf