# Unsupervised Clustering Report

## <u>Deep Embedding Clustering (DEC)</u>

### Introduction

DEC is an unsupervised neural network clustering algorithm using data driven approach which learns a mapping from data space to a lower dimensional feature space to iteratively optimize the clustering objective. Using deep neural network, it simultaneously learns both feature representations and cluster assignments. In order to do this task in an unsupervised enviroment, the authors introduced a novel idea of iteratively refine the clusters using an auxiliary target distribution that is derived from the soft clustering assignment (probability to assign to each cluster). In doing so, it gradually improves both the clustering and the feature representation.

### Method

The authors were inspired by parametric t-SNE. Instead of using the KL divergence loss to minimize distance from the original data space, they define a centroid-based probability distribution and minimize its KL divergence to an auxiliary target distribution to simultaneously improve both clustering assignments and feature representation. In other words, the KL divergence is used to calculate the information that is loss when they approximate one distribution with another distribution. This would be able to help hardening the soft cluster assignment; in this case the probability of assigning to a specific cluster. Centroid-based method is chosen by the authors in the purpose of reducing the complexity and hence reducing the required time to run the algorithm.

Supposed we want to cluster n points $\{x_i \in X\}_{i=1}^{n}$ into k clusters, each represented by a centroid $\mu_j, j = 1, \dots, k$. The authors suggested to first do dimensional reduction using a non-linear mapping $f_\theta : X \to Z$, where Z is the feature space and $\theta$ is the parameter that they want to train. Then, $f_\theta$ can be parameterized using deep neural network (DNN). Using this set up, it allows the authors to simultaneously learn the centroid variable $\{\mu_j \in Z\}_{j=1}^{k}$ in the feature space Z and the parameters $\theta$ in the DNN that maps the data space to the feature space.

There are two phases in this algorithm, the first one is parameter initialization using stacked autoencoder (SAE) and use the encoder layer as the initial mapping between the data space and feature space. The second phase then is the parameter optimization, where they iterate between computing an auxiliary target distribution and minimizing the KL divergence loss. The structure of the network that they use can be seen below.

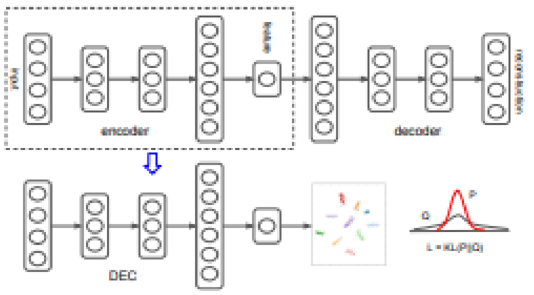

*Figure 1.* Network structure

*Parameter Initialization Phase*

The authors chose to initialize the parameter using SAE because the result of the past studies indicate that the method consistently produce meaningful and well-separated representation on real-world datasets. They use initialize the SAE layer by layer where each layer is a two layers neural network of denoising autoencoder that serves the purpose of reconstructing previous layer's output after a random corruption in the initial data took place. The network is defined as the following:

$$\tilde{x} \sim Dropout(x)$$
$$h = g_1(W_1\tilde{x} + b_1)$$
$$\tilde{h} \sim Dropout(h)$$
$$y = g_2\left(W_2\tilde{h} + b_2\right)$$

Here $Dropout(.)$ is a stochastic mapping that randomly drops (setting to 0) a portion of its input dimension, $g_1$ is the activation function for the encoding layer and $g_2$ is the activation function for the decoding layer. The parameter $\theta$ represents a set of trainable parameters $\{W_1, b_1, W_2, b_2\}$.

The training is performed by minimizing the L2 loss between $x$ and $y$. The authors use ReLUs in all encoder/decoder pairs expect for the activation function of the decoder in the first pair because it needs to reconstruct input data that may have positive and negative values, and the activation function of the encoder in the last pair, so that the final data embedding retains full information. After the greedy-wise layer training is done, the authors concatenate all encoder layers followed by all decoder layers, in reverse layer-wise training order (like the image above) to form a deep autoencoder and then finetune it to minimize the reconstruction loss. The final result of this phase is a multilayer deep autoencoder with a bottleneck coding layer in the middle. Then, the decoder part is discarded, and the result is used as an initial mapping from the data space to the feature space. The initial cluster centroid is also initialized in this phase by performing k-means in the feature space so that they got k initial centroids $\{\mu_j\}_{j=1}^{k}$.

*Clustering with KL divergence (Parameter Optimization)*

Now that the initialization of the non-linear mapping $f_\theta$ and cluster centroids $\{\mu_j\}_{j=1}^{k}$ are done, the next step is to improve the clustering/optimization. First, the authors compute the soft assignment between the obtained embedded points and the cluster centroids using the Student's t-distribution kernel that measures the similarity between the embedded point $z_i$ and the cluster centroid $\mu_j$. The equation is the following:

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

The obtain result $q_{ij}$/soft assignment can be interpreted as the probability of assigning the sample $i$ to cluster $j$. In general case, $\alpha$ is the degree of freedom of the Student's t-distribution, however, since the cross-validation on $\alpha$ is not possible for unsupervised learning, the authors let $\alpha = 1$ in this study.

After obtaining the soft assignment, the authors iteratively refine the cluster by minimizing the KL divergence loss between the soft assignments $q_i$ and the auxiliary distribution $p_i$. Hence, the objective function of this study is the following:

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

The key variable here is the auxiliary distribution $p_i$ since it would need to be able to refine the soft assignments $q_i$. Choosing the right *P* is crucial in this study and the authors want *P* to have the following properties:

1. Strengthen the prediction/improving the cluster purity
2. Put more emphasis on data points assigned with high confidence
3. Normalize the loss contribution of each centroid to prevent large clusters from distorting the hidden feature space.

This led to them choosing $p_i$ by raising the soft cluster assignment $q_i$ to the second power and then normalizing it by the frequency per cluster using the following equation:

$$p_{ij} = \frac{\frac{q_{ij}^2}{f_j}}{\sum_{j'}\left(\frac{q_{ij'}^2}{f_j'}\right)}, where\ f_{j=}\sum_i q_{ij}\ \ is\ soft\ cluster\ frequency$$

The purpose of setting $P$ in this manner is so that the auxiliary target distribution would have a stricter assignment probability than the soft cluster assignment (closer to 0 and 1). Hence, it would allow the soft cluster assignments to be refined by the auxiliary target distribution so that the cluster impurity would improve. Also, this would also put more emphasis on the data points that are assigned with high confidence (high probability) and prevent large clusters from distorting the hidden feature space. As the iteration goes by, the high confidence assignment would help to improve the low confidence assignments. The optimization step is then conducted using Stochastics Gradient Descent (SGD) with momentum by jointly optimize the cluster center $\{\mu_j\}$ and the DNN parameters $\theta$. The gradient of the objective function L with respect to feature-space embedding of each data point $z_i$ and each cluster centroid $\mu_j$ are computed as the following:

$$\frac{\partial L}{\partial z_i} = \frac{\alpha+1}{\alpha}\sum_j\left(1+\frac{\left|\left|z_i-\mu_j\right|\right|^2}{\alpha}\right)^{-1} \times (p_{ij}-q_{ij})(z_i-\mu_j)$$

$$\frac{\partial L}{\partial \mu_j} = -\frac{\alpha+1}{\alpha}\sum_i\left(1+\frac{\left|\left|z_i-\mu_j\right|\right|^2}{\alpha}\right)^{-1} \times (p_{ij}-q_{ij})(z_i-\mu_j)$$

The gradients with respect to the embedded data point $z_i$ are then passed down to the DNN and used in standard backpropagation to compute the DNN's parameter gradient. The stopping parameter here is then the tolerance level of % of points change cluster assignment between two consecutives iterations.

**Possible Issues:**

1. Initializing parameters relies on deep autoencoder to get a good initialization, might carry over the problems that autoencoder is facing.
2. Choosing the auxiliary target distribution is crucial in this study, aside from the suggested equation, is there any other distribution that work?
3. Have problem with large dataset such as STL.

## Invariant Information Clustering (IIC)

### Introduction

IIC is an unsupervised neural network clustering algorithm that follows the method of semantic clustering, in which the learned function directly outputs discrete assignments. It uses the knowledge of information theory to maximize mutual information between the class assignments of each pair. The trained network outputs semantic labels immediately, so no further processing is required, which makes this algorithm easy to understand. Furthermore, by using mutual information, degenerate solution is avoided compared to other clustering algorithms. IIC directly trains a randomly initialized neural network into a classification function, end-to-end without any labels. The objective function is simply just the mutual information between the function’s classifications for paired data samples.

According to the authors, IIC also robust to two common issues faced by other clustering algorithms. The first one is clustering degeneracy, which is the tendency for a single cluster to dominate the predictions or for clusters to disappear. The main reason is because of the entropy maximization component within the mutual information which will not be maximized if degeneracy solution is achieved. Furthermore, the conditional entropy minimization within the mutual information also allows it to be optimal for the model to predict for each image a single class with certainty. The second issue is noisy data with unknown or distractor classes which exist in dataset like STL-10. IIC addresses this issue by using auxiliary overclustering, which creates an auxiliary output layer that is parallel to the main output layer, trained to produce overclustering that is ignored during the test time. The structure of the network is the following:

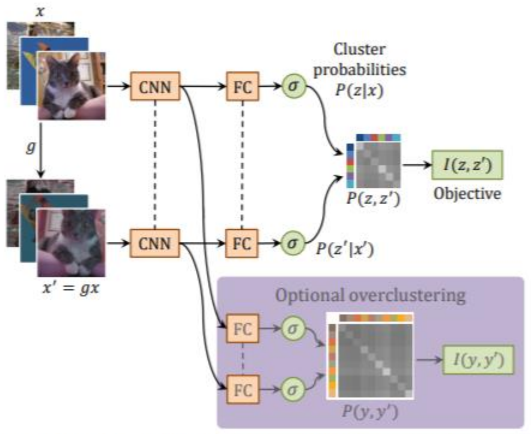

Figure 2: IIC for image clustering. Dashed line denotes shared parameters, $g$ is a random transformation, and $I$ denotes mutual information (eq. (3)).

## Method

*General Approach*

Generally, IIC can be used to cluster any unlabeled paired data by training a network to predict cluster identities. Suppose we have $x$, $x' \in X$ a paired data sample from a joint probability distribution $P(x, x')$. Here, x and x' can be different images containing the same objects, such as a dog with a sky background and a dog with a grass background. The main objective of IIC is to learn a representation $\Phi: X \to Y$ which maps the original data space $X$ into the feature space $Y$ by preserving what is common between $x$ and $x'$ while discarding instance specific details, in the example here is the grass and sky background. Then, the objective function is the following:

$$\max_{\Phi} I(\Phi(x), \Phi(x'))$$

The objective function can be interpreted as maximizing the predictability of $\Phi(x)$ from $\Phi(x')$ and vice versa. The immediate effect of the equation above is that to make $\Phi(x')$ the same with $\Phi(x)$. This does not mean simply minimizing representation distance like other clustering algorithm which might lead to degeneracy solution. The entropy within $I(\,.\,)$ allows the method to avoid degeneracy solution. If $\Phi$ is a bottleneck neural network, the objective function would also serve as a tool to discard instance-specific details from the input data, because the feature space is a finite set of class indices, so clustering imposes natural bottleneck. If bottleneck is not use, setting $\Phi$ to an identity function would solve the equation, due to the inequality $I(x, x') \geq I(\Phi(x), \Phi(x'))$.

The author also uses soft clustering instead of hard clustering so that the output of $\Phi(x) \in [0,1]^C$ represent the distribution of a discrete random variable $z$ over $C$ classes, formally given by $P(z = c|x) = \Phi_c(x)$. This represents the probability of assigning to a specific class. For the joint case, the pair of input $x$ and $x'$ have cluster assignment variables $z$ and $z'$ respectively. The conditional joint distribution is then given by $P(z = c, z' = c'|x, x') = \Phi_c(x). \Phi_{c'}(x')$. The implication of the equation is that the clustering assignment variables $z$ and $z'$ are independent conditioned on the inputs $x, x'$, however, they are generally not independent after marginalization over a dataset of input pairs $(x_i, x_i'), for\ i = 1, \dots, n$. An example that the authors gave is that, for a trained classification network $\Phi$ and a dataset of image pairs where each image contains the same object but randomly in different position, the class assignment variable, $z$ will have a strong statistical relationship with the second class assignment variable $z'$, because one is predictive of the other, so that highly dependent on one another.

After marginalization over a dataset, the joint probability distribution $P$ is given by a $C \times C$ matrix where each row $c$ and column $c'$ constitutes $P_{cc'} = P(z = c, z' = c')$, it can also be represented as:

$$P = \frac{1}{n} \sum_{i=1}^{n} \Phi(x_i) \, . \, \Phi(x_i')^T$$

$P$ is symmetrized using $\frac{P+P^T}{2}$. The matrix $P$ can be interpreted as a correlation matrix between $z$ and $z'$. Then, the initial objective function can be computed by plugging in $P$ into the equation, so that:

$$I(z, z') = I(P) = \sum_{c=1}^{C} \sum_{c'=1}^{C} P_{cc'} \, . \ln \left( \frac{P_{cc'}}{P_c . P_{c'}} \right)$$

The mutual information should be at maximum when $x$ is equal to $x'$, because $x$ is fully predictable from $x'$ and vice versa. The information should be at minimum when $x$ is independent of $x'$, where they are not predictable at all from one another because they don't share any mutual information.

Back to why degenerate solutions are avoided in this case, the expression of mutual information can be expanded into the difference between cluster assignment entropy and conditional cluster assignment entropy:

$$I(z, z') = H(z) - H(z|z')$$

Hence, it can be interpreted as maximizing the cluster assignment entropy $H(z)$ and minimizing the conditional assignment entropy $H(z|z')$. The smallest value of the conditional entropy happens when cluster assignments are exactly predictable from one another ($H(z|z') = 0$) and the largest possbile value of the entropy happens when all clusters are equally likely to be picked ($H(z) = \ln(C)$). Hence, the mutual information is not maximized when degenerate solution is achieved, and instead achieved when data is assigned evenly between the cluster.

*Image Clustering*

For unsupervised image clustering, the paired samples $(x, x')$ are often not available, the authors overcame this issue by using generated image pairs. The pairs consist of the original image $x$ and its randomly perturbed version $x' = gx$.

Substituting into the general equation, the following maximization problem is obtained for image clustering:

$$\max_{\Phi} I(\Phi(x), \Phi(gx))$$

The perturbation $g$ can be scaling, skewing, rotation, or geometric transformation. Then, the IIC algorithm can recover the factor which is invariant to which of pair is picked.

*Auxiliary Overclustering*

This method is applicable for STL-10 dataset where the training data consists of both relevant and irrelevant classes. The authors added an auxiliary overclustering head to the network that is trained with the full dataset, while the main output head is trained using only the relevant classes. The idea is to create more clusters than the actual classes in the auxiliary overclustering head, and it can be used to increase the general expressiveness of the feature representation, so that the network could learn more compared to the one without auxiliary overclustering head. This application is also applicable for the cases where all classes are relevant. In that case, both the main output head and the overclustering head is trained with full dataset. The result obtained by the author indicates that for STL10 dataset, the auxiliary overclustering improves the clustering accuracy from 44% to 61%.

*Architecture*

All networks are randomly initialized and consist of either ResNet or VGG11-like, followed by one or more heads (linear predictors). For increased robustness, each head is duplicated h=5 times with a different random initialization.

**Possible Issues**

1. The idea of multiplying head might backslash as it would lead to an inconsistent result. Based on the replication work that I did, this issue arises, sometimes the one head can be as accurate as 99%, but the other heads are around 91%. There should be a way to deal with this issue.
2. The algorithm requires a lot of steps/epochs, it might be questionable on how many steps is enough, and what happen if the numbers of steps is too many. Based on my replication, if the number of epochs are not enough, some subheads might have not converged.
3. Seems to be sensitive to hyperparameters, changing the number of epochs and learning rate affected the result significantly.

***Selected Result from Replication and Analysis on MNIST Dataset***

**IIC**

**<u>The hyperparameters for the model and the resulted error</u>**

| Batch | Learning Rate | Epoch | Sub1 | Sub2 | Sub3 | Sub4 | Sub5 | Average |
|---|---|---|---|---|---|---|---|---|
| 700 | 1e-4 | 200 | 0.0221 | 0.1598 | 0.1023 | 0.0089 | 0.1022 | 0.0791 |
| 700 | 1e-4 | 400 | 0.0779 | 0.0914 | 0.0780 | 0.0913 | 0.0084 | 0.0694 |
| 700 | 1e-4 | 800 | 0.0216 | 0.0216 | 0.0216 | 0.0216 | 0.0216 | 0.0216 |
| 700 | 1e-4 | 3200 | 0.0078 | 0.0984 | 0.0211 | 0.1013 | 0.0078 | 0.0473 |
| 700 | 5e-4 | 800 | 0.1594 | 0.1568 | 0.0207 | 0.1635 | 0.0207 | 0.1042 |
| 700 | 1e-5 | 1000 | 0.0217 | 0.1135 | 0.1649 | 0.1129 | 0.0627 | 0.0951 |
| 700 | 5e-5 | 1000 | 0.1235 | 0.0219 | 0.1234 | 0.1207 | 0.1234 | 0.1026 |

Seems like the best learning rate is **<u>1e-4</u>**, and at least **<u>800</u>** epochs are recommended, otherwise some subhead might not converge to the final accuracy. A lot of inconsistency between the subhead. Best single subhead has accuracy of **<u>99.2%</u>** as reported in the paper, however, the average accuracy seems a little bit lower compared to the one in the paper.

**DEC**

**<u>Accuracy for DEC algorithm</u>**

First run: **0.82**

Second run: **0.85**

Third run: **0.89**

Seems like re-running the trained data increases the accuracy.