Deep Neural Networks (Głębokie Sieci Neuronowe)

Module 4: Deep clustering

Krzysztof Krawiec

Wydział Informatyki i Telekomunikacji
Politechnika Poznańska
2019/2020
http://www.cs.put.poznan.pl/kkrawiec/

Clustering

- Unsupervised assignment of data points to clusters.
- Formally: partitioning.
- Notable representatives: k-means, k-medoids, agglomerative clustering, ...
- Implicitly, we often expect a clustering algorithm to discover the decision classes.
    - However, this is not guaranteed to happen.

On the clustering task

Bad news: Not a well-defined task.

- In general: partition the set of observations into subgroups/clusters of examples that exhibit high degree of similarity within clusters and low degree of similarity between clusters; usually also:
  - Produce a model applicable to new (test) data,
  - Estimate the number of clusters automatically.
  - Meet other expectations, e.g. make the cluster sizes roughly balanced.
- Typical models: centroid-based, density-based, graph-based, connectivity-based (hierarchical clustering), etc.

Good news: For some types of models, user's degree of satisfaction with the model can be expressed in a formula.

- That formula can be turned into a loss function that can be optimized in DL.

Motivating example: clustering of image data

- Labelled data is scarce in many (most) applications (e.g. medicine)
- Images carry a lot of internal structure:
  - Dependencies between pixels
  - Spatial relationships between objects
  - Random images are highly improbable
- Possible approach: autoencoder
- The essence of the idea:
  - Train autoencoder on unlabelled data
  - Use the latent layer as the representation in which the clustering is conducted
- Form of representation learning.

Review of deep clustering methods

Clustering with Deep Learning: Taxonomy and New Methods

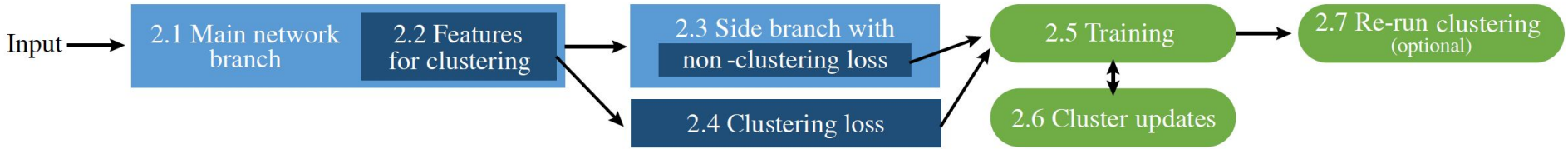
Elie Aljalbout, Vladimir Golkov, Yawar Siddiqui, Maximilian Strobel, Daniel Cremers

https://arxiv.org/abs/1801.07648

Note: a review paper.

Taxonomy

- Neural network training, including
  - Neural network
    - Architecture,
    - Set of features used for clustering,
  - Losses:
    - Non-clustering loss,
    - Clustering loss,
    - Method to combine the two losses,
  - Cluster updates,
- (Optional) Re-running clustering after network training
- Paper organization/ reflects, to an extent, the steps of training procedure:



Neural architectures for clustering

- Multilayer Perceptron
- Convolutional Neural Network
- Deep Belief Network (DBN):
  - Generative graphical model, consisting of several layers of latent variables.
- Generative Adversarial Network (GAN):
  - A system of two competing neural network models G and D that engage in a zero-sum game.
- Variational Autoencoder (VAE):
  - A Bayesian network with an autoencoder architecture that learns the data distribution (generative model).

Set of deep features used for clustering

As in many other DL architectures, there is no need to strictly delineate the feature extraction stage and the ‘proper’ model building stage (a clear advantage!)

- These co-occur seamlessly along the processing (unless mandated by the architecture, or other aspect)
- Gradient resulting from the loss function is transformed by the clustering network and informs the feature extracting phase
    - The is more directed than in traditional feature selection/construction methods, which typically rely on filter or wrapper approach.

The authors review architectures that ‘harvest’ features:

- From single (last) layer
- Several layers
    - Note: `global’ features can be collected from entire layers via pooling or aggregation
    - Applicable to arbitrary data, not only images.

Clustering loss: hard assignment

k-Means loss: Assures that the new representation is k-means-friendly (Yang et al., 2016a)

$$L(\theta) = \sum_{i=1}^{N} \sum_{k=1}^{K} s_{ik} \| z_i - \mu_k \|^2,$$

Where:

- z_i: embedded data point,
- μ_k: cluster center,
- s_{ik}: boolean variable for assigning z_i to μ_k

Question 1: How to implement this using typical DL components?
Question 2: What is the main weakness of this loss definition?

Some questions

Question 1: How to implement this using typical DL components?

Answer:

- Tensor broadcasting
- Square error
- Minimum (we don't even need to explicitly define the s_{ik}

Question 2: What is the main weakness of this loss definition?

Answer:

- It's not only non-differentiable – it's discontinuous
- Can seriously destabilize the gradients and thus the entire training process
- Solution: soft assignment of objects to clusters.

Clustering loss: soft assignment

Student's t-distribution can be used as the kernel to measure the similarity (van der Maaten and Hinton, 2008) between points μ_j and centroids:

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

where ν is a constant (typically 1).

- Note the minus sign in the exponents!

The q_{ij}s:

- are normalized,
- implement soft assignment of objects to clusters,

thus can be used to replace the s_{ik} in the cluster assignment loss.
Question: Can they, really?

Clustering loss: soft assignment

Student's t-distribution can be used as the kernel to measure the similarity (van der Maaten and Hinton, 2008) between points μ_j and centroids:

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

Note: q_{ij}s define a probability distribution Q. If we had a reference distribution P, we could define our loss as KL divergence:

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

Question: Where can we get *P* from?

Where can we get *P* from?

Naive option: define P as delta distribution (one-hot vector).

However, because Q is soft, it's better to define P also as a soft distribution.

What do we want?

1. Strengthen predictions (i.e., improve cluster purity),
2. Put more emphasis on data points assigned with high confidence, and
3. Normalize loss contribution

This motivation is quoted from:
Unsupervised Deep Embedding for Clustering Analysis, Junyuan Xie, Ross Girshick, Ali Farhadi, https://arxiv.org/abs/1511.06335

Clustering loss: soft assignment

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

By squaring the original distribution and then normalizing it, the auxiliary distribution *P* forces assignments to have stricter probabilities (closer to 0 and 1).

$$p_{ij} = \frac{q_{ij}^2/\Sigma_i q_{ij}}{\Sigma_{j'}(q_{ij'}^2/\Sigma_i q_{ij'})}.$$

Loss: a divergence between these two distributions:

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

This is called cluster assignment hardening loss (Xie, Girshick, Farhadi 2016)

Comments

This may be considered a form of self-training (Nigam & Ghani, 2000):

1. We label the original (unlabelled) dataset by the initial classifier C to it,
2. The labels produced by C (here: after hardening) become targets for C

Deep Clustering via Cluster Hardening

Unsupervised Deep Embedding for Clustering Analysis

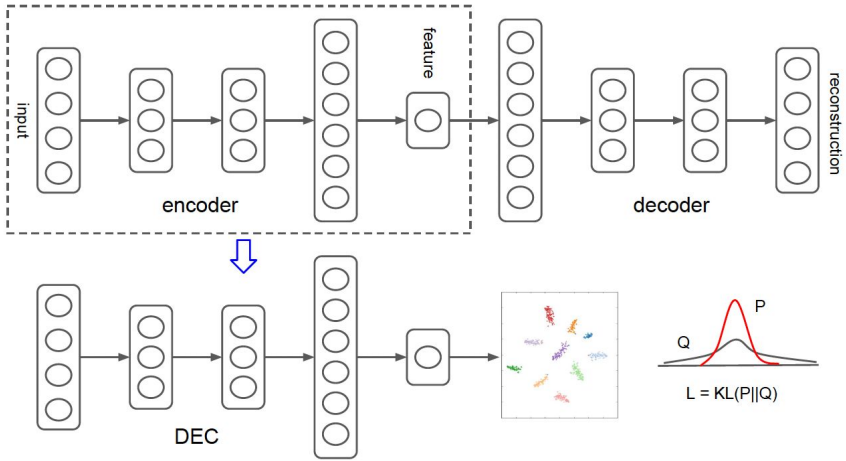
https://arxiv.org/abs/1511.06335

Junyuan Xie, Ross Girshick, Ali Farhadi

Method’s name: DEC: Deep Embedded Clustering

Training: Phase 1

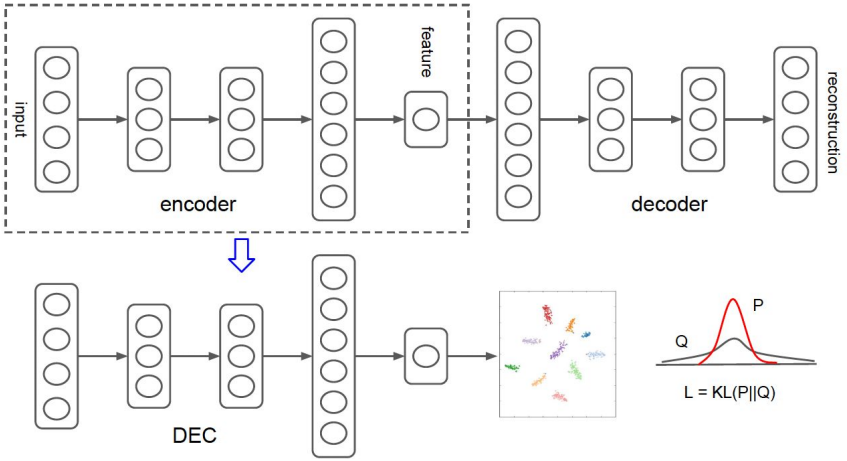
- Network architecture: “Denoising” stacked autoencoder
- Iterative training of successive layers using input-output MSE
  - ReLU, dropout
- Trained layers stacked into a deep autoencoder and fine-tuned (SGD)
- Resulting encoder (its latent representation) used as feature space



Training: Phase 2

1. Run traditional k-means to obtain the initial locations of centroids μ_k (20 restarts, picking the best solution)
2. Calculate P, Q and KL loss
3. Update the centroids
4. If more than tol% examples changed assignment, go to step 2

Notice: This phase does not introduce any extra parameters (except for ν)



Empirical evaluation

Quality metric: unsupervised clustering accuracy

$$ACC = \max_{m} \frac{\sum_{i=1}^{n} \mathbf{1}\{l_i = m(c_i)\}}{n}$$

- Assumes ‘optimistic assignment’: a cluster is labelled according to the majority class of examples that belong to it
- Can be efficiently calculated using the Hungarian method.

Unsupervised clustering accuracy

```
def acc(y_true, y_pred):
    """
    Calculate clustering accuracy. Require scikit-learn installed

    # Arguments
        y: true labels, numpy.array with shape `(n_samples,)`
        y_pred: predicted labels, numpy.array with shape `(n_samples,)`

    # Return
        accuracy, in [0,1]
    """
    y_true = y_true.astype(np.int64)
    assert y_pred.size == y_true.size
    D = max(y_pred.max(), y_true.max()) + 1
    w = np.zeros((D, D), dtype=np.int64)
    for i in range(y_pred.size):
        w[y_pred[i], y_true[i]] += 1
    from sklearn.utils.linear_assignment_ import linear_assignment
    ind = linear_assignment(w.max() - w)
    return sum([w[i, j] for i, j in ind]) * 1.0 / y_pred.size
```

https://github.com/XifengGuo/DEC-keras/blob/master/metrics.py

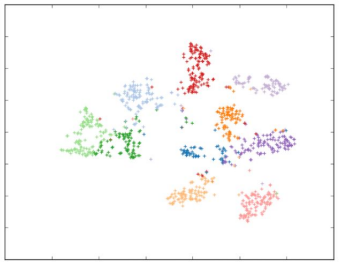
Empirical evaluation

Results:

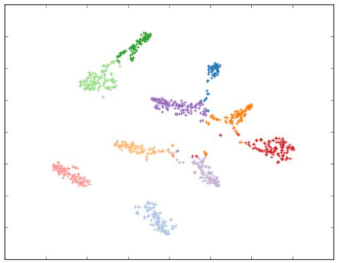
Method	MNIST	STL-HOG	REUTERS-10k	REUTERS
k-means	53.49%	28.39%	52.42%	53.29%
LDMGI	84.09%	33.08%	43.84%	N/A
SEC	80.37%	30.75%	60.08%	N/A
DEC w/o backprop	79.82%	34.06%	70.05%	69.62%
DEC (ours)	**84.30%**	**35.90%**	**72.17%**	**75.63%**

- Two image datasets, two text datasets (represented as TFIDF of 2000 most frequent word stems)
  - All datasets L2-normalized $\frac{1}{d}\|x_i\|_2^2$
- DEC: the full method
  - encoder receives the gradient from the loss function and is being updated
- DEC w/o backprop:
  - encoder is frozen
- LDMGI and SEC: spectral clustering methods

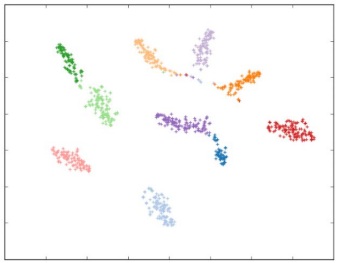
Latent space adaptation during learning (MNIST)



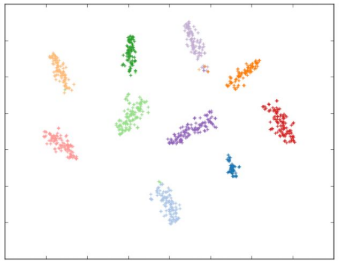
(a) Epoch 0



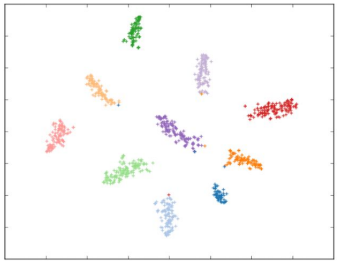
(b) Epoch 3



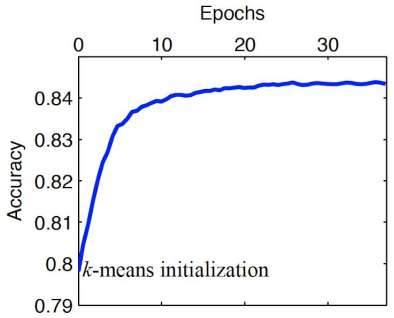
(c) Epoch 6



(d) Epoch 9

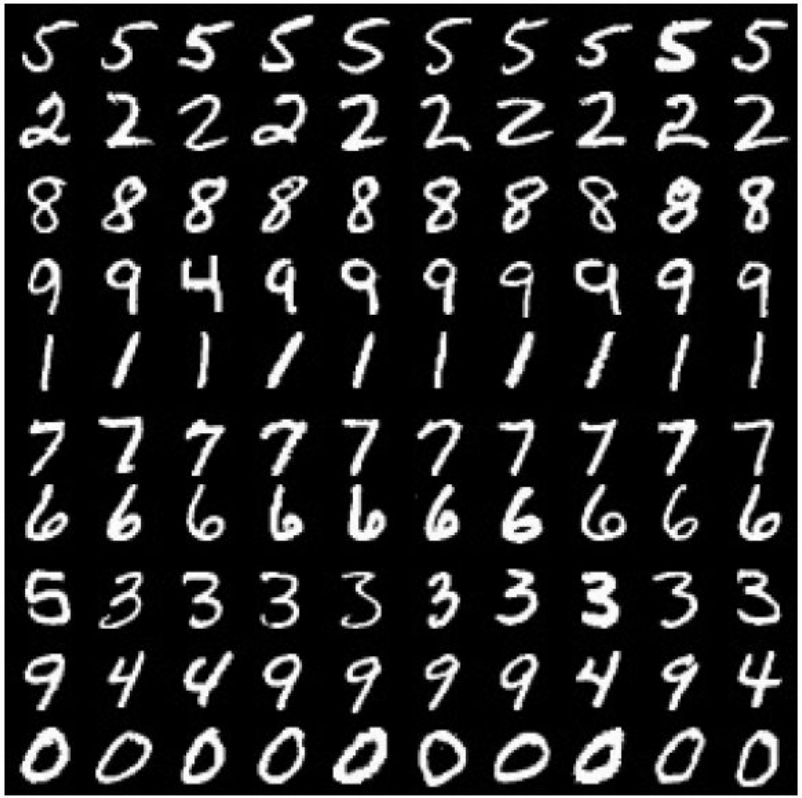


(e) Epoch 12

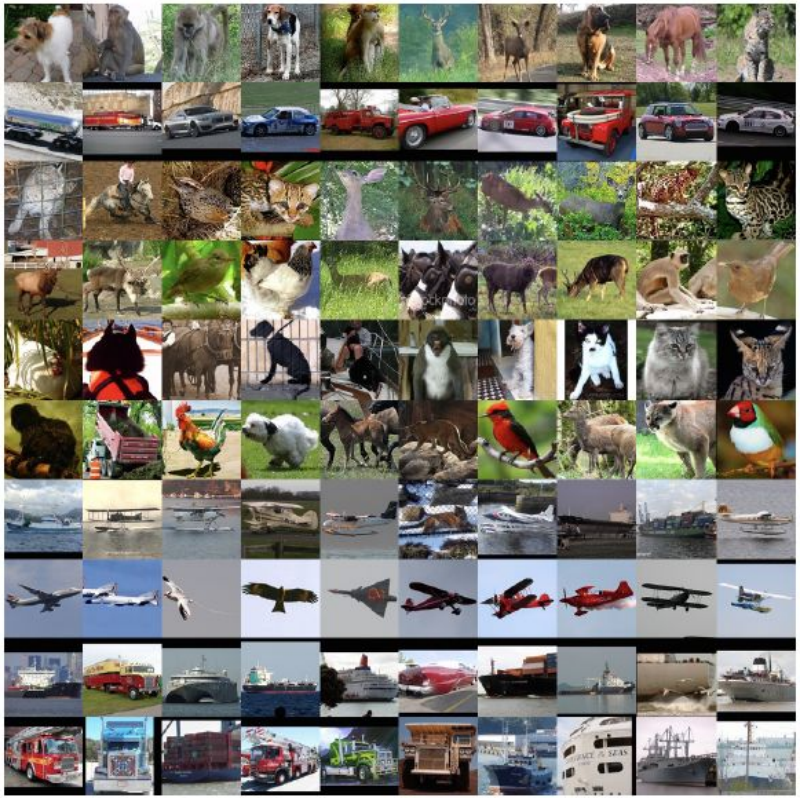


(f) Accuracy vs. epochs

Visualization: t-SNE (?)



(a) MNIST



(b) STL-10

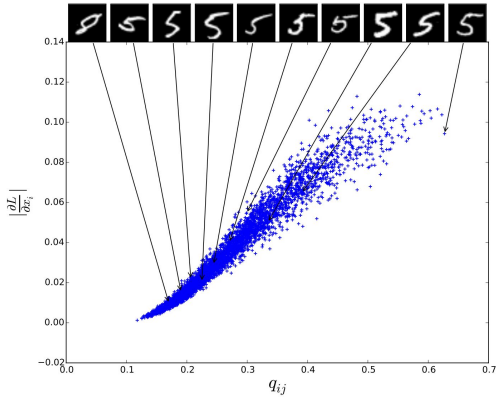
Each row: The 10 top scoring elements from a cluster.

Comment

The underlying assumption of DEC is that the initial classifier's high confidence predictions are mostly correct. (Section 5.1)

True for the datasets considered in the study: dependency of gradient on the soft assignment q_{ij} for a random MNIST cluster, at the beginning of optimization:

- Examples closer to the cluster center (higer q_{ij}) contribute more to gradient



My personal experience: Much harder to achieve good results on demanding problems, with less compact clusters.

Non-clustering loss components (a.k.a. cluster regularization)

Controlling cluster size distribution

Promoting balanced clusters:

$$L_{ba} = \mathrm{KL}(G \| U)$$

U: uniform distribution, *G*: probability of distribution of assigning points to clusters:

$$g_k = P(y = k) = \frac{1}{N} \sum_i q_{ik}$$

Promoting a specific (non-uniform) distribution of cluster sizes:

- Replace *U* with a distribution of target clusters, based on some premises.
- E.g., Zipf's law: The most frequent word occurs approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.
  The rank-frequency distribution is an inverse relation
  (k: rank, s: exponent)

$$f(k; s, N) = \frac{1/k^s}{\sum_{n=1}^{N} (1/n^s)}$$

Locality-preserving loss

Meant to preserve the locality of the mapping from the original space (X) to the latent space (Z). Possible implementation:

$$L_{lp} = \sum_{i} \sum_{j \in N_k(i)} s(x_i, x_j) \|z_i - z_j\|^2$$

- $N_k(i)$: set of k nearest neighbors of x_i
  - Or, in the simplest case, all xs in the training set.
- $s(x_i,x_j)$: similarity measure between x_i and x_j (e.g., images)

Note:

- Does not explicitly refer to cluster centers.
- Measuring similarity in X is in general challenging.

Other non-clustering loss terms used in clustering

- Autoencoder reconstruction loss:

$$L = d_{\mathrm{AE}}(x_i, f(x_i)) = \sum_i \|x_i - f(x_i)\|^2,$$

- Self-Augmentation Loss: Hu et al. (2017): pushes together the representation of the original sample and augmentations of the examples from that sample:

$$L = -\frac{1}{N} \sum_N s(f(x), f(T(x))),$$

- where T is an augmentation operation.

Combining loss terms

In general:

$$L(\theta) = \alpha L_c(\theta) + (1 - \alpha) L_n(\theta),$$

L_c: clustering loss, L_n: non-clustering loss.

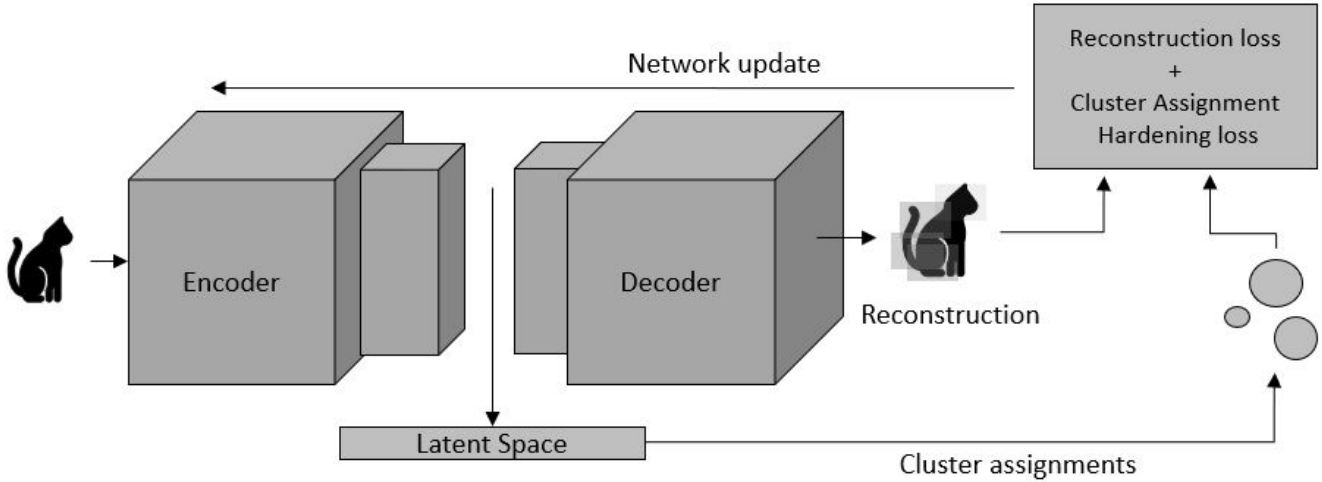
Ways of determining and scheduling the changes of α:

- Pre-training and fine-tuning: start with α=0, then set α=1
- Joint training, e.g. α=0.5
- Variable schedule: e.g., start with low α and increase it gradually.

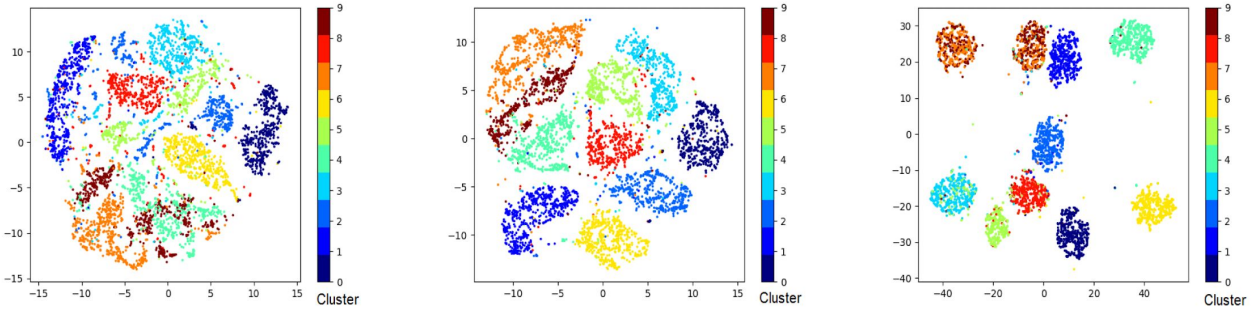
Follow-ups of Xie et al.

Aljalbout et al.:

- Don’t discard the decoder: use the reconstruction loss too.
- Motivation: reconstruction loss forces the latent representation to maintain its high information content.
  - Lower risk of features degeneration under the pressure of clustering loss.



Results on MNIST and COIL-20



(a) k-Means (b) Autoencoder + k-Means (c) Proposed

Figure 3: t-SNE visualizations for clustering on MNIST dataset in (a) Original pixel space, (b) Autoencoder hidden layer space and (c) Autoencoder hidden layer space with the proposed method.