# Summer Research Internship 2018 at Speech and Vision Lab, IIT Madras

Report submitted by

*Bhargav D*
Undergraduate Student at IIT - Kharagpur

# TABLE OF CONTENTS

# 1. Introduction

## 1.1 Mathematical Background

First to get a more concrete mathematical background of the concepts of Machine Learning and Deep Learning in detail, I read specific Chapters concerning the work I would be doing from the books: Pattern Recognition and Machine Learning by C.M. Bishop and Deep Learning by Courville, Goodfellow and Bengio.

## 1.2 Caption/Description Generation for Multimedia

Then I read certain Papers about generating Captions/Descriptions for Multimedia (mainly Images).

### 1.2.1 Describing Multimedia Content using Attention - based Encoder-Decoder Networks

In this paper, they described the attention-based encoder–decoder architecture for describing multimedia content. Background materials on recurrent neural networks (RNN) and convolutional networks (CNN) which form the building blocks of the encoder–decoder architecture. Emphasis was given on specific variants of those networks that are often used in the encoder–decoder model; a conditional language model based on RNNs (a conditional RNN-LM) and a pre-trained CNN for transfer learning. Then, the simple encoder–decoder model followed by the attention mechanism, which together form the central topic of this paper, the attention-based encoder–decoder model was described. Then applications for the proposed attention-based encoder–decoder model were also described.

### 1.2.2 Show and Tell Lessons learned from 2015 MSCOCO Image Captioning Challenges

It presented NIC, an end-to-end neural network system that can automatically view an image and generate a reasonable description in plain English. NIC is based on a convolution neural network that encodes an image into a compact representation, followed by a recurrent neural network that generates a corresponding sentence. The model is trained to maximize the likelihood of the sentence given the image. The robustness of NIC in terms of qualitative results (the generated sentences are very reasonable) and quantitative evaluations, using either ranking metrics or BLEU, a metric used in machine translation to evaluate the quality of generated sentences was shown based on Experiments on various Datasets. The various improvements made to the basic NIC model were described which was used to win the MSCOCO Image Captioning Challenge.

#### 1.2.3 Deep Visual Semantic Alignments for Generating Image Descriptions

A model that generates natural language descriptions of image regions based on weak labels in form of a dataset of images and sentences with very few hard-coded assumptions was introduced. The approach featured a novel ranking model that aligned parts of visual and language modalities through a common, multimodal embedding. This model provided state of the art performance on image-sentence ranking experiments. Another model described a Multimodal Recurrent Neural Network architecture that generated descriptions of visual data. Its performance on both full-frame and region-level experiments showed that in both cases the Multimodal RNN outperformed retrieval baselines.

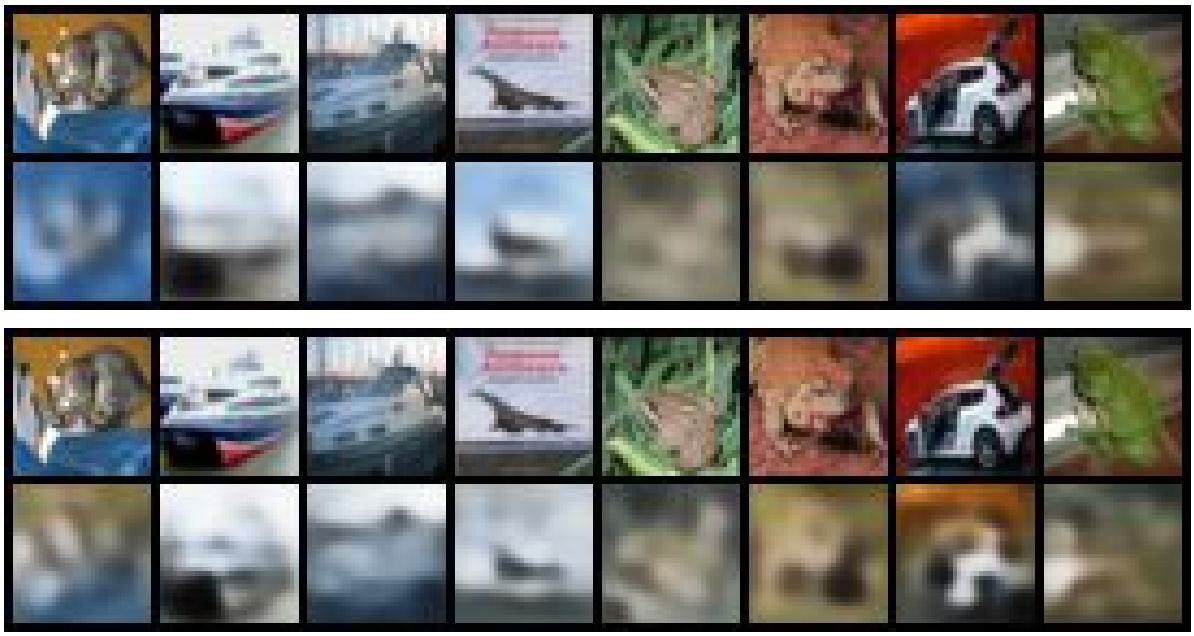
# 2. Pytorch - The Researcher's Framework

## 2.1 Building an efficient Neural Network Classifier

- Next, to get familiar with building of Neural networks with Pytorch, I was asked to work on the CIFAR10 dataset to build a good Neural net Classifier.
- For Starters, a very basic architecture ( Conv2d, MaxPool, Conv2d, fc1, fc2, fc3 ) was able to achieve **61%** on the test set. Then I increased the complexity of the my network and also did Hyper-parameter tuning. The best accuracy I got was implementing the ResNet architecture with hyper-parameter tuning was **91%** on the test set.

## 2.2 Building an efficient Stacked AutoEncoder

- The next task was to build a Stacked Auto-Encoder (SAE) using Pytorch on the CIFAR10 dataset. The idea was to pass the an Image through an encoder to generate an Embedding and then re-create the image from it using a Decoder Network.
- Experiments were run on the SAE by changing the intermediate embedding size and also a latent space was also introduced.
- It was clearly seen that as the size decreases the Generated Images become Blurrier or conversely as the size increases the Generated Images becomes Clearer.
- When a Latent Space in-between the Encoder and the Decoder part of the SAE with Intermediate Size of 128 was introduced it degraded the quality of the Generated Images as compared to without the latent space of size 128.



**Top Set of Images** : Embedding Size of 128 with Latent Space
**Bottom Set of Images** : Embedding Size of 128 without Latent Space
**Note** : In each pair of Images, the top one corresponds to the original one and the bottom corresponds to the recreated one

# 3. Deep Embedded Clustering

## 3.1 Introduction

Deep Embedded Clustering (DEC) is an end-to-end clustering system. It simultaneously learns feature representations and cluster assignments using deep neural networks. The DEC learns a mapping from the input feature space to a lower-dimensional feature space in which it iteratively optimizes the clustering objective. The complexity of the model increases linearly with the number of data points rather than exponentially. The DEC has two phases: (1) parameter initialization with deep stacked autoencoder (2) parameter optimization with clustering.

## 3.2 Parameter Initialization with deep Stacked AutoEncoder

### 3.2.1 Why SAE ?

The DEC contains a Stacked AutoEncoder (SAE), to transform the data from the input space X to the latent feature space Z (dimension(Z) <= dimension(X)). The SAE is chosen for this task because it consistently produces semantically meaningful and well-separated representations. This is conducive for clustering task.

### 3.2.2 SAE Training

The SAE is trained to reconstruct the data using the full encoder-decoder network. First, each autoencoder is trained greedily. In this greedy layer-wise training each autoencoder is trained to reconstruct its previous layer's output. This can be described as follows:

$$xi^* = Dropout(xi) \quad (4.1)$$

$$hi = g1(Wi1.xi^* + bi1) \quad (4.2)$$

$$hi^* = Dropout(hi) \quad (4.3)$$

$$yi = g2(Wi2.hi^* + bi2) \quad (4.4)$$

Set of learnable parameters: Wi1, Wi2, bi1 and bi2
Activation Functions : g1 for encoder and g2 for decoder
xi* is constructed by randomly replacing xi values to zero using Dropout

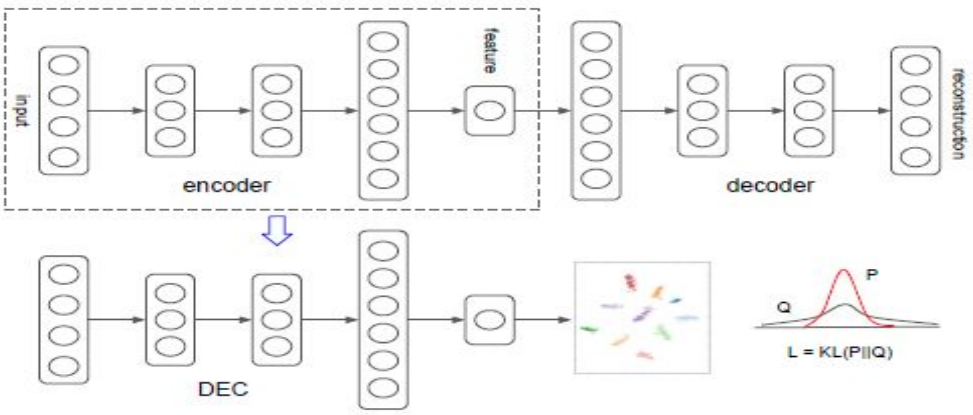



Figure. DEC Network Structure

## 3.3 Parameter Optimization with Clustering

Now, each AEi is trained to minimize the least squared error between xi and yi. After training AEi, its encoder output hi is given as input to the next autoencoder AEi+1. Now, AEi+1 is trained to reconstruct hi using a similar architecture. After greedy layer-wise training, SAE is constructed by cascading all encoder modules of trained autoencoders followed by reverse cascading of all decoder modules. This forms multilayer deep encoder-decoder structure. Further, SAE is trained to reconstruct the input with the whole architecture. The final layer output of the encoder gives the features for clustering task as shown in Figure. After training the SAE to reconstruct input, the decoder module of SAE is discarded and the feature representations at the bottleneck layer are collected for the whole input data and are used to initialize the initial cluster centroids. Initial

centroids are obtained by using the k-means clustering algorithm on feature representations.
With non-linear mapping f from the encoder of SAE and the initial cluster centroids, the DEC improves clustering performance using an unsupervised clustering algorithm with two steps. In the first step, soft assignments between embedded points and cluster centroids are computed. In the second step, the mapping f and centroids are recomputed by learning from the current high confidence assignments using an auxiliary target distribution. This process is repeated until there is no significant change in assignments in two successive iterations.
**Soft Assignment** : DEC uses Student's t-distribution as a kernel to measure the similarity between an embedded point zi and centroid j
**Target Auxiliary Distribution :** The assumption in DEC is that the initial high confidence predictions are mostly correct. The DEC learns from its initial high confidence soft assignments by using a target auxiliary distribution P. So, the choice of P must have following properties: (a) strengthen predictions (improve cluster purity), (b) give importance to the data points with high assigned confidence, and (c) normalize loss contribution of each centroid to prevent large clusters from distorting the hidden feature space. It is constructed from soft assignments.
**KL Divergence Minimization :** The DEC iteratively refines the cluster centroids and the mapping f by minimizing KL divergence between soft assignments Q and target distribution P.
**Optimization :** The DEC jointly optimizes the cluster centers and DNN parameters using backpropagation of gradients.

## 3.4 Evaluation Metric - Unsupervised Clustering Accuracy

**Unsupervised Clustering Accuracy :** The performance of the model is evaluated using the unsupervised clustering accuracy defined as follows:

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

Here, li and ci are the ground truth label and the clustering label of image xi respectively,
m is a one-to-one mapping from the cluster labels to the ground truth labels and n is the number of examples. The mapping m ranges over all possible one-to-one mappings from the cluster labels to the ground truth labels.

## 3.5 Training Phase with optimized set of Hyper-Parameters

***Input :*** For training normal AutoEncoder, flattened image pixels or Scale-Invariant Feature Transform are given as input, while for convolutional autoencoder, direct images are given as input.

SAE is first trained with greedy layer-wise pre-training. Each layer is trained to reconstruct its previous layer's output.

Training of each layer : 250 epochs with a dropout rate of 0.2. Then the constructed SAE is fine-tuned to reconstruct the input for 500 epochs without any dropout. During both layerwise training and fine-tuning, the learning rate is set to 0.1 decaying by factor 10 for every 70 epochs and batch size is set to 256.

After training SAE, the decoder module is discarded. Embeddings for the whole data are extracted and used for cluster centroid initialization with standard k-means clustering. The number of clusters is set to 10 in k-means clustering.

In the KL divergence minimization phase, the DEC is trained with a fixed learning rate of 0.01 with no dropout. Convergence threshold is set to tol = 0:1%.

In case of stacked convolutional autoencoder, the training process is similar but the initial learning rate while training the network for reconstruction is set to 0.01. In the KL divergence minimization phase, the learning rate is set to 0.001.

## 3.6 Results on Data-Sets

**DEC with AutoEncoder**

| Dataset | ACC(%) |
|---|---|
| MNIST | 84.8% |
| Fashion-MNIST | 44.8% |
| CIFAR-10 | 23.3% |

**DEC with Conv AutoEncoder**

| Dataset | ACC(%) |
|---|---|
| MNIST | 92% |
| Fashion-MNIST | 66% |
| CIFAR-10 | 29% |

# 4. Improved Deep Embedded Clustering

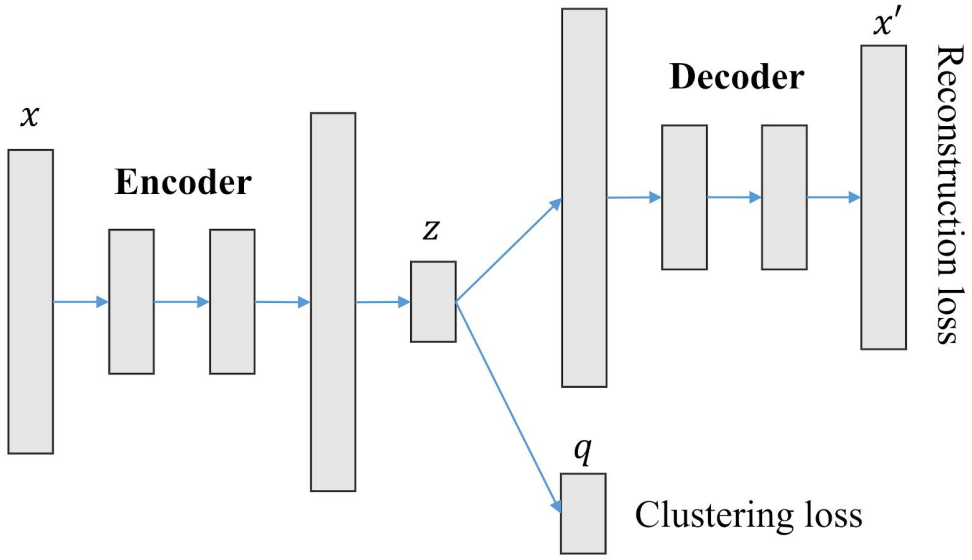
## 4.1 Issue with Deep Embedded Clustering

Deep clustering learns deep feature representations that favor clustering task using neural networks. A vital ingredient has been overlooked by these work that the defined clustering loss may corrupt feature space, which leads to non-representative meaningless features and this in turn hurts clustering performance.

## 4.2 Why Improved Deep Embedded Clustering ?

To address this issue, the Improved Deep Embedded Clustering (IDEC) algorithm takes care of data structure preservation. Specifically, we manipulate feature space to scatter data points using a clustering loss as guidance. To constrain the manipulation and maintain the local structure of data generating distribution, an under-complete autoencoder is applied. By integrating the clustering loss and autoencoder reconstruction loss, IDEC can jointly optimize cluster labels assignment and learn features that are suitable for clustering with local structure preservation. The resultant optimization problem can be effectively solved by mini-batch stochastic gradient descent and backpropagation.

## 4.3 Difference between DEC and IDEC - Network Structure




**The network structure of IDEC :** The encoder and decoder are composed of fully connected layers. Clustering loss is used to scatter the embedded points z and the

reconstruction loss makes sure that the embedded space preserves local structure of data generating distribution.

## 4.4 Clustering and Reconstruction Loss

**Objective :** It is defined as L = Lr + t*Lc where Lr and Lc are reconstruction loss and clustering loss respectively, and t> 0 is a coefficient that controls the degree of distorting embedded space. When t = 1 and Lr = 0, reduces to the objective of DEC.
**Reconstruction Loss :** The reconstruction loss is measured by Mean Squared Error (MSE) between xi and gW'(zi) where zi = fW(xi) and fW and gW' are encoder and decoder mappings respectively. Autoencoders can preserve local structure of data generating distribution. Under this condition, manipulating embedded space slightly using clustering loss will not cause corruption. So the coefficient is better to be less than 1.

## 4.5 Parameter Optimization and Updation in training

**Optimization :** We optimize the Loss Function using mini-batch stochastic gradient descent (SGD) and backpropagation. There are three kinds of parameters to optimize or update: autoencoder weights, cluster centers and target distribution P.
**Update target distribution :** The target distribution P serves as "groundtruth" soft label but also depends on predicted soft label. Therefore, to avoid instability, P should not be updated at each iteration (one update for autoencoder weights using a mini-batch of samples is called an iteration) using only a batch of data. In practice, we update target distribution using all embedded points every T iterations. When update target distribution, the label assigned to xi is obtained by **si = arg max j (qij)**. Training will be stopped if label assignment change (in percentage) between two consecutive updates for target distribution is less than a threshold.
**Running Time:** The time complexity of IDEC algorithm is O(nD2+ndK), where D, d and K are maximum number of neurons in hidden layers, dimension of embedding layer and number of clusters. Generally K <= d <= D holds, so the time complexity is O(nD2).
**Training :** It was done in the same way as DEC with hyper-parameter optimization.

## 4.6 Results on Data-Sets

**IDEC with AutoEncoder**

| Dataset | ACC(%) |
|---|---|
| MNIST | 88% |
| Fashion-MNIST | 49.8% |

| | |
|---|---|
| CIFAR-10 | 28.3% |

**IDEC with Conv AutoEncoder**

| Dataset | ACC(%) |
|---|---|
| MNIST | 96.5% |
| Fashion-MNIST | 70.5% |
| CIFAR-10 | 34.5% |

# 5. IDEC/DEC for Speaker Diarization

The IDEC/DEC can be extended to the task of speaker diarization. Given a set of speech segments, DEC clusters those segments into clusters. However, the number of clusters is to be specified explicitly since DEC doesn't perform agglomerative clustering. We can also extend DEC to speaker diarization by using it as a feature extractor. The DEC learns a mapping function f through SAE which is more conducive for clustering task. The number of clusters also need not be specified in this case since we perform agglomerative clustering on DEC features. But the number of clusters in DEC training should be much more than the expected number of clusters.
Both DEC as well as IDEC were run on Speech Data [ MFCC and SCP Data Files ]. The mfcc vector corresponding to each and every time-step was given one of the cluster labels at the end of clustering.

# 6. Speaker Diarization

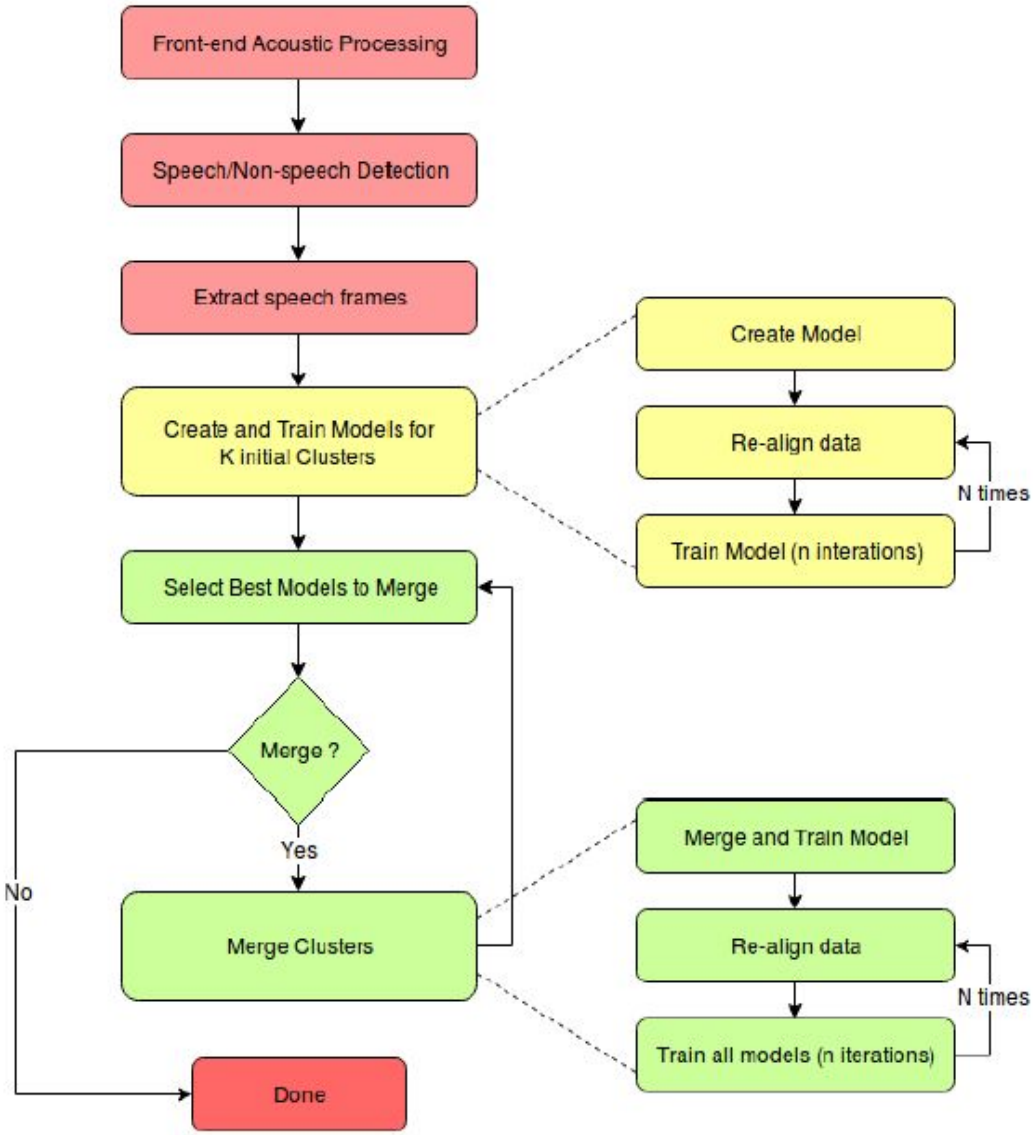
I worked with approaches based on hidden Markov Model/Gaussian Mixture Model (HMM/GMM) and Information Bottleneck (IB) for Speaker Diarization.

## 6.1 HMM/GMM Speaker Diarization System

HMM/GMM speaker diarization system is the classical approach to the diarization task. It mainly consists of 3 phases: (a) Front-end acoustic processing, (b)Speech/Non-speech detection, and (c) Diarization.
**6.1.1 Front-end Acoustic Processing:** In this phase, the input speech signal is filtered to remove any noise in the signal and Mel Frequency Cepstrum Coefficients (MFCC) features are extracted from the filtered signal.

**6.1.2 Speech/Non-speech Detection:** In this phase, non-speech part of the input is eliminated. So, the model is trained only using speech data from the input signal.
**6.1.3 Diarization:** In this phase, an agglomerative clustering technique is applied on the data to form clusters. Initially, the system splits the data into K clusters (where K > number of ground truth speakers) then it iteratively selects (according to a metric based On BIC ) a pair of most similar clusters and merges them. This iterative merging is done until no such pair of clusters is found. The data is modeled using an HMM, where the initial number of states is equal to the initial number of clusters (K).

## HMM/GMM Speaker Diarization System Flow-Chart



**6.1.4 Delta Bayesian Information Criterion** is a criterion for selecting a model from a finite set of models. Let, M = M1; M2; : : : ; MN be a set of models trained on set of data segments X = X1; X2; : : : ; XN. To select a pair similar models from M, then we can use BIC as a metric to identify them. It is calculated as follows:

$$BIC(i, j) = \log(Pr(Xij \mid Mij)) - \log(Pr(Xi \mid Mi)) - \log(Pr(Xj \mid Mj))$$

Here, Xij is obtained by combining Xi and Xj segments and model Mij is trained on Xij . After creating and training the initial K models for K speaker clusters, BIC score is calculated for all possible pairs of models and a pair is selected based on maximum BIC score (and > some threshold to avoid over-segmentation). These two models are then merged into a single model and trained with merged data. This process is repeated until no such pair found.

**6.1.5 The Evaluation Metric** used is Speaker Error Rate ( SER ) to evaluate the performance of the model.

$$SER = \frac{\sum_{s=1}^{S} dur(s).(min(N_{ref}(s), N_{sys}(s)) - N_{correct}(s))}{T_{score}}$$

Here, N ref (s) and N sys (s) indicate the actual and predicted number of speakers speaking in segment s respectively. N correct (s) indicates the number of speakers that speak in segment s and have been correctly matched. T score is the total scoring time.

## 6.1.6 Training Phase with optimized set of Hyper-Parameters

**Input**: The length of each frame is 25 milliseconds and with a shift of 10 milliseconds. For each frame, 19-dimensional MFCC features are extracted.

**Training:** The stacked autoencoder, DEC/IDEC are trained as specified above.
The input to the stacked autoencoder is a flattened speech segment of 5 frames. So, the input feature dimension is 95. Then, feature embeddings for whole data are extracted from the SAE in the trained DEC/IDEC. These feature embeddings are used as input features for HMM/GMM and IB diarization systems.

**Hyper-Parameters:** Hyper-Parameter Optimization is done for the various parameters of the HMM/GMM System. Below are the Optimized parameters by audio file class.

| Audio File Class | Initial number of mixtures | Initial segmentation and polling iterations | Iterations for segmentation and polling while clustering |
|---|---|---|---|
| CMU | 3 | 10 | 2 |
| ICSI | 5 | 10 | 2 |
| NIST | 7 | 10 | 3 |
| LDC | 10 | 10 | 3 |

Dimensionality of Input Feature Vectors : 19 & Initial Number of Clusters : 20

#### 6.1.7 Results on Data

| File Name | SER ( in % ) | File Name | SER ( in % ) |
|---|---|---|---|
| CMU_20020319-1400 | 36 | CMU_20020320-1500 | 9 |
| ICSI_20010208-1430 | 14.2 | ICSI_20010322-1450 | 31 |
| LDC_20011116-1400 | 9.8 | LDC_20011116-1500 | 24 |
| NIST_20020214-1148 | 33 | NIST_20020305-1007 | 40 |

Overall SER for NIST RT04 dataset (all the above data files) = 25%

### 6.2 Information Bottleneck based Speaker Diarization

The agglomerative Information Bottleneck approach performs bottom-up clustering based on information bottleneck principle. Let X represent a set of segments in a speech file, Y represent a set of variables of interest that give meaningful information about the active speaker in each segment, and C represent a clustering solution to X. The IB based approach converts the set of segments X into a set of clusters C that conveys most of the information possible about Y. Each segment x is assumed to be short and contains only one active speaker and it can be modeled by a Gaussian. The IB principle states that "the clustering solution C should preserve all the information possible about the variables of interest Y (i.e., maximize I(Y;C)) under a constraint on the mutual information between X and C (i.e, minimize I(C;X))". The stop condition for the iterative merging is given by a threshold on the normalized mutual information(NMI). After the merging completed, the output is re-segmented using KL-HMM or HMM/GMM. This system performs much faster than HMM/GMM based speaker diarization system and achieves similar performance.

# 7. Viterbi Realignment

## 7.1 Introduction

The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states called the Viterbi path. In this part, I integrated this algorithm into the 2 Speaker Diarization Systems we have seen so far.

## 7.2 Integration into IB based Speaker Diarization System

The Viterbi algorithm can be integrated into the IB System by performing the re-segmentation using KL-HMM or HMM/GMM after each iteration in the AIB Clustering till the stopping criterion is reached. The system now takes much longer than the original IB based Speaker Diarization System in terms of running time. This is because the re-alignment part usually takes the longest duration in the IB Diarization System and since now we are doing the re-alignment after each and every iteration, the running time increases. But this comes with the added advantage of better performance, i.e., lower SER. For Example, the SER for CMU_20020319-1400 we get is 33% which is 3% less than what I got for using the HMM/GMM system for the same sample file.

## 7.3 Integration into HMM/GMM Speaker Diarization System

### 7.3.1 Modifications

The IB based Speaker Diarization System with the integrated Viterbi algorithm was modified with new auxiliary functions and different Loss Objectives such that it behaves likes a HMM/GMM Speaker Diarization System. Running time remained similar.
The system was ran with the same parameters and results were obtained. In general, where there was scope for improvement, i.e., high SER, there seems to better improvement as compared to cases where the SER was already low. [ Comparison with HMM/GMM Speaker Diarization System without Viterbi Integration ]

### 7.3.2 Results Obtained

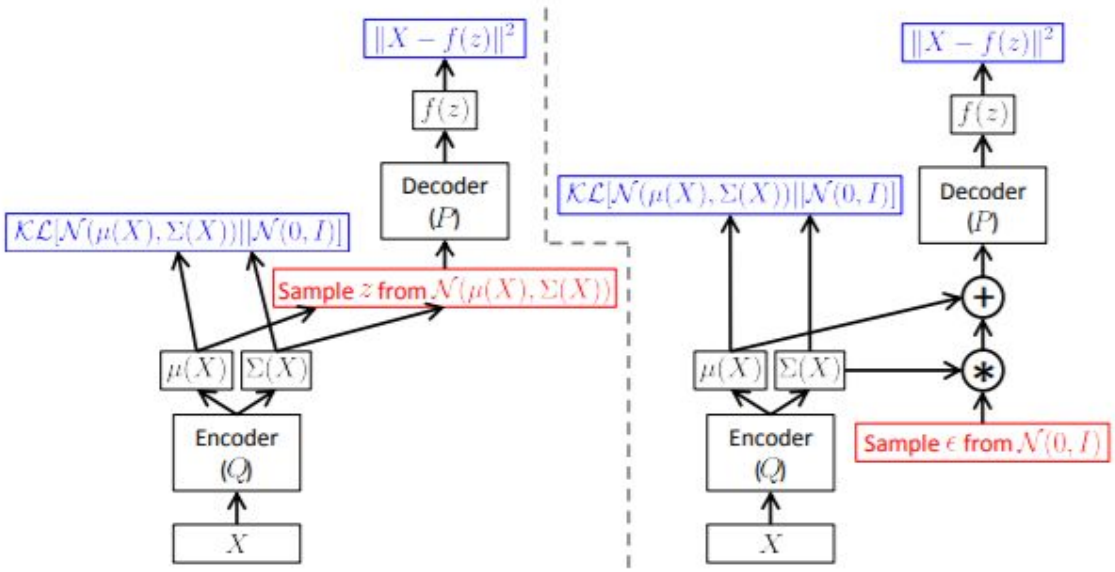
| File Name | SER ( in % ) | File Name | SER ( in % ) |
|---|---|---|---|
| CMU_20020319-1400 | 28 | CMU_20020320-1500 | 8.68 |
| ICSI_20010208-1430 | 13.3 | ICSI_20010322-1450 | 27 |
| LDC_20011116-1400 | 8.2 | LDC_20011116-1500 | 22 |
| NIST_20020214-1148 | 29 | NIST_20020305-1007 | 37.9 |

# 8. Variational Autoencoders

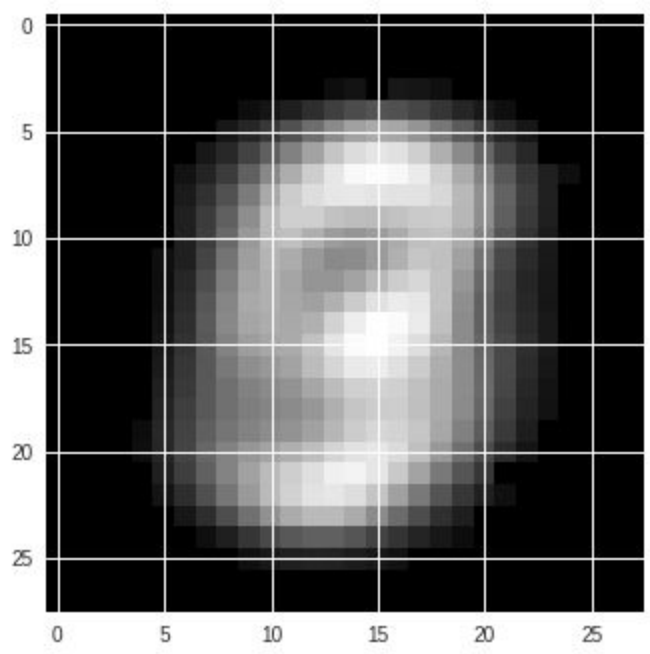
## 8.1. Introduction

Variational autoencoder models inherit autoencoder architecture, but make strong assumptions concerning the distribution of latent variables. They use variational approach for latent representation learning, which results in an additional loss component. It assumes that the data is generated by a directed graphical model p(x|z) and that the encoder is learning an approximation q'(x|z) to the posterior distribution p'(x|z). The objective contains the Kullback–Leibler divergence and the prior over the latent variables.

## 8.2 VAE network Structure



A training-time variational autoencoder implemented as a feedforward neural network, where P(X|z) is Gaussian. Left is without the “reparameterization trick”, and right is with it. Red shows sampling operations that are non-differentiable. Blue shows loss layers. The feedforward behavior of these networks is identical, but backpropagation can be applied only to the right network.




**8.3 Implementation & Images generated :** I implemented a simple VAE Architecture with 2 linear encoding and decoding layers with the corresponding sampling function, loss function and feed-forward function. MSE Loss and ADAM Optimizer were used. It was trained on MNIST dataset with batch-size of 32 for 100 training epochs. Then an image was generated by sampling. The produced image seems to be a mix of 3 and 9, better results can be obtained from sophisticated architectures and training for longer epochs.

# 9. Generative Adversarial Nets

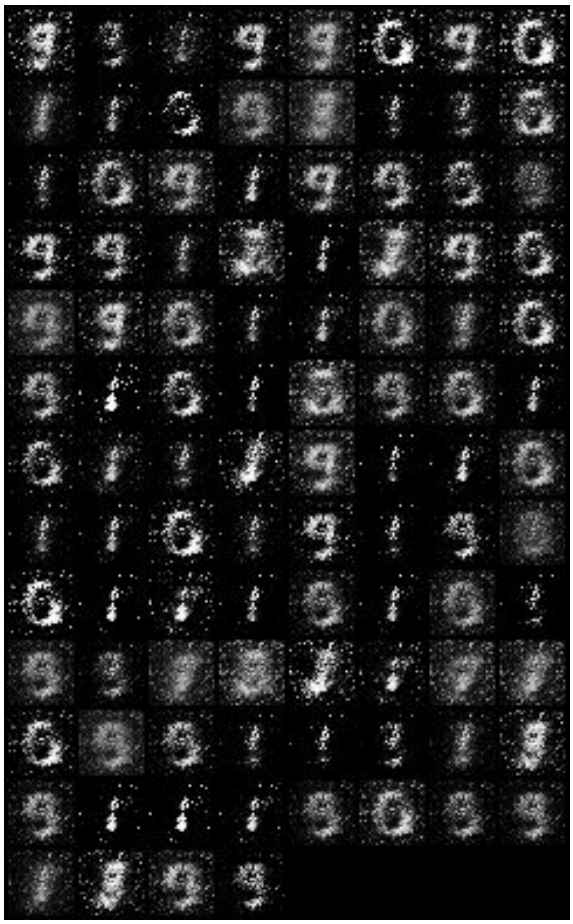
## 9.1 Introduction

It’s a framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation.

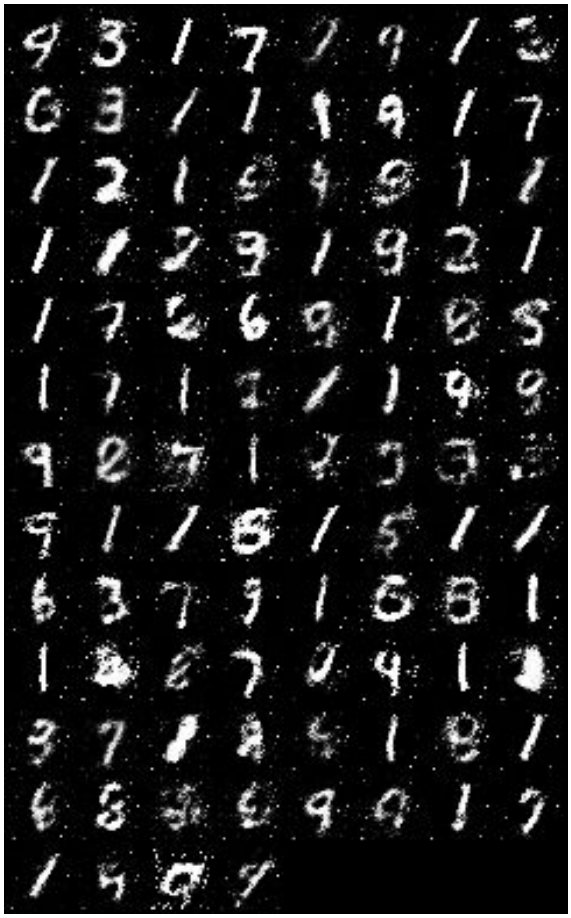
## 9.2 Implementation Details

I implemented a simple GAN in which the Discriminator (D) consists of Linear and LeakyRelu units in pairs and the Generator (G) consists of Linear and Relu units in pairs. BCE Loss and ADAM Optimizer were used. It was trained on MNIST dataset with batch-size of 100 for 50 training epochs. Training was done on MNIST dataset and Fake Images were generated at the end of each training epoch. Hyper-parameters such as latent size and hidden size can be tuned for better Images.

## 9.3 Images generated



Epoch 1



Epoch 50

From the above two images we can see the stark difference in the fake images created at the end of epochs 1 and 50 respectively. Better results can be obtained from sophisticated architectures and training for longer epochs and hyper-parameter optimization. Also GANs seem to better at Fake image generation than VAEs.