Testing the Deep Autoencoding Gaussian Mixture Model

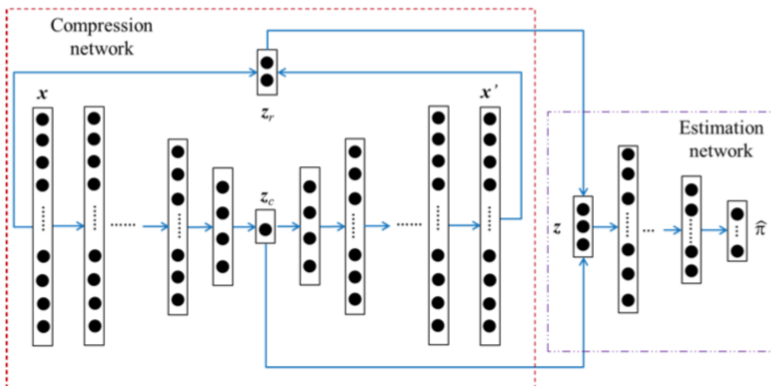


Figure 2: An overview on Deep Autoencoding Gaussian Mixture Model

0.1 First Step: Dimension Reduction

The network takes as input a sample x and operates as shown below:

$$\mathbf{z}_c = h\left(\mathbf{x}; \theta_e\right)$$

$$\mathbf{x}' = g\left(\mathbf{z}_c; \theta_d\right)$$

$$\mathbf{z}_r = f\left(\mathbf{x}, \mathbf{x}'\right)$$

$$\mathbf{z} = \left[\mathbf{z}_c, \mathbf{z}_r\right]$$

Here z_c is the low-dimensional embedding and $x^{'}$ is the reconstructed form of x produced by the encoder $h(\cdot)$ and decoder $g(\cdot)$. The variables θ_e and θ_d are parameters of the autoencoder. The function $f(\cdot)$ computes the reconstruction feature z_r. Observe that the value that will be passed to the estimation network is a matrix that is a combination of z_c and

0.2 Second Step: Density Estimation

Next, the equivalence between log-linear models and the posterior form of a Gaussian is exploited to perform density estimation via a simple feed-forward network which is referred to as the estimation network. The estimation network acts as a softmax classifier, providing a K-dimensional vector representing the membership probabilities:

$$\hat{\gamma} = softmax(\mathbf{MLN}\left(\mathbf{z}; \theta_{\mathbf{m}}\right)) \tag{1}$$

where θ_m is the parameter of the MLN.

Next, the parameters are adjusted following the same calculations as the typical EM algorithm.

$$\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$$

$$\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} \mathbf{z}_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

$$\hat{\mathbf{\Sigma}}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} \left(\mathbf{z}_i - \hat{\mu}_k\right) \left(\mathbf{z}_i - \hat{\mu}_k\right)^T}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$$

$$E(\mathbf{z}) = -\log \left(\sum_{k=1}^{K} \hat{\phi}_k \frac{\exp\left(-\frac{1}{2} \left(\mathbf{z} - \hat{\mu}_k\right)^T \hat{\mathbf{\Sigma}}_k^{-1} \left(\mathbf{z} - \hat{\mu}_k\right)\right)}{\sqrt{\left|2\pi \hat{\mathbf{\Sigma}}_k\right|}} \right)$$

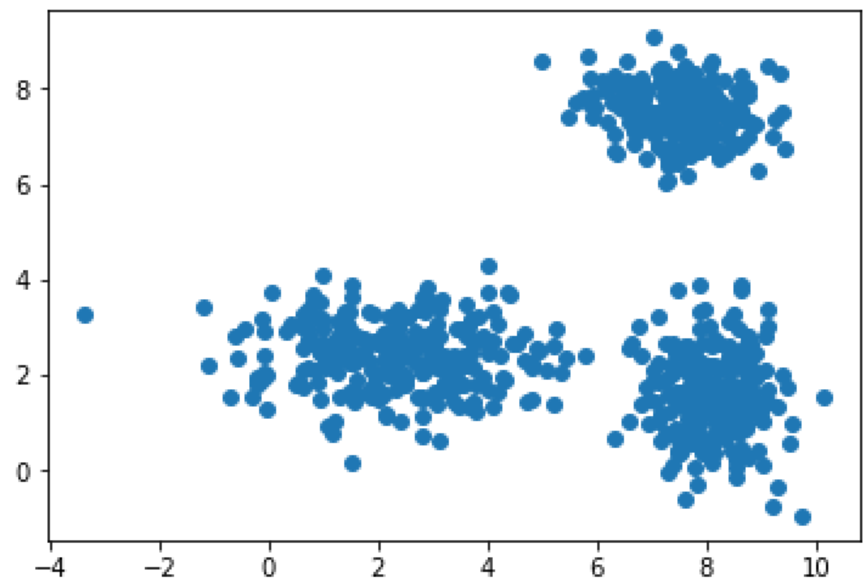
0.3 Objective Function

Finally we train the entire neural net with this somewhat mysterious objective function.

$$J\left(\theta_e, \theta_d, \theta_m\right) = \frac{1}{N} \sum_{i=1}^{N} L\left(\mathbf{x}_i, \mathbf{x}_i'\right) + \frac{\lambda_1}{N} \sum_{i=1}^{N} E\left(\mathbf{z}_i\right) + \lambda_2 P(\hat{\mathbf{\Sigma}})$$

0.4 Explaining Quick Experiment

To get some intuition on how well the architecture is able to approximate the original distribution of the input I began by first coming up with my distributions with constant μ_k and Σ_k. An example is pictured below, depicting many points sampled from three Gaussians.



This example is somewhat unusual, and almost defeats the purpose of modeling the scenario as a GMM and was mostly done to see how well the model performs in certain edge-cases.

After this I feed the inputs into the neural-net. This required some tuning of the hyper parameters λ_1 and λ_2. After training the model, I hope to find a suitable benchmark for comparing how well the model was able to reproduce the original distribution. For this I was considering simply implementing some standard clustering algorithms and comparing results.