10-708: Probabilistic Graphical Models, Spring 2018

Lecture 21 : A Hybrid: Deep Learning and Graphical Models

Lecturer: Kayhan Batmanghelich *Scribes: Paul Liang, Anirudha Rayasam*

1 Introduction and Motivation

Deep learning and graphical models despite sharing several common features like having the same structure, modeling objectives, energy etc. have very key differences which differentiate them making one more suitable than the other depending on the problem at hand. The main objective in deep learning is mostly to learn a model to perform tasks of classification or feature extraction and the research effort is directed to experimenting with the various architectures, gate functions and the activations which compose the network. These networks have proven to be very good at fitting the data discriminatively to make good predictions. Back-propagation is the backbone of the learning in deep learning networks. Graphical models assist in encoding the domain knowledge in a structured fashion allowing easy incorporation of already known information for other downstream tasks. The focus in graphical models has been toward improving the inference accuracy and convergence speed. Most notable applications of graphical models in recent times have been in transfer learning and latent variable inference.

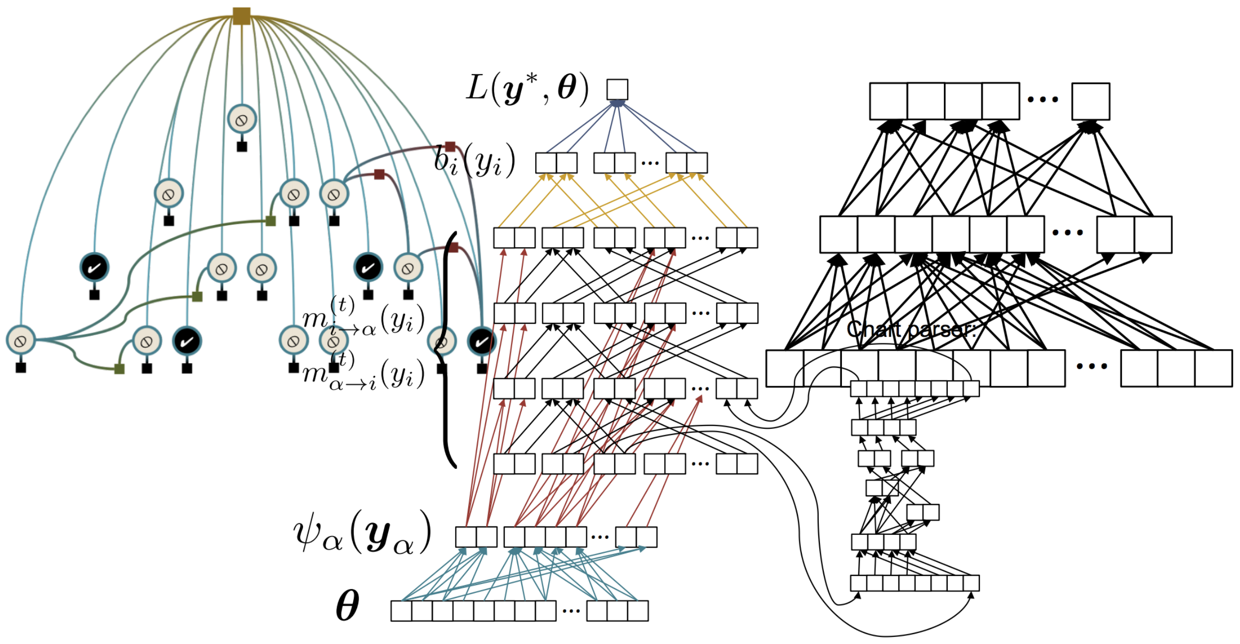

$L(\boldsymbol{y}^*, \boldsymbol{\theta})$
$b_i(y_i)$
$m^{(t)}_{i\to\alpha}(y_i)$
$m^{(t)}_{\alpha\to i}(y_i)$
$\psi_\alpha(\boldsymbol{y}_\alpha)$
$\boldsymbol{\theta}$
Chart parser:

Figure 1: Example of a typical hybrid model: neural network + graphical model

In recent times deep learning has proven to be very efficient for learning tasks associated with massive datasets and is known to learn an approximation for almost any function. But the complex architectures being utilized have compromised the interpretability of these models and during the evaluation it is hard to isolate conclusively the factors responsible for the performance boost. For example, to understand if the improvement is from the tweaked architecture, better algorithm, quality of data and preprocessing etc. Additionally, the concept of training error is perfectly valid when we have a classifier with no hidden states and hence inference and inference accuracy is not an issue. However, deep neural networks are not just classifiers and possess many hidden states, so inference quality is something that should be looked at more carefully. This is often not the case in current deep learning research. Hence, in graphical models, lots of efforts are directed towards improving inference accuracy and convergence. The fusion of these two techniques have been explored for various tasks and have proven very effective, opening up frontiers for effective convergence between deep learning and graphical models. Example of a typical hybrid neural network and graphical can be seen in Figure 1.

2 Hybrid Models

The goal of hybrid models is to examine whether we can combine efficiently the best of these two paradigms, resulting in defining a neural network that incorporates domain knowledge enabling it to learn features for a graphical model, and then training the entire model using back-propagation.

2.1 Neural Network + Hidden Markov Models (HMM)

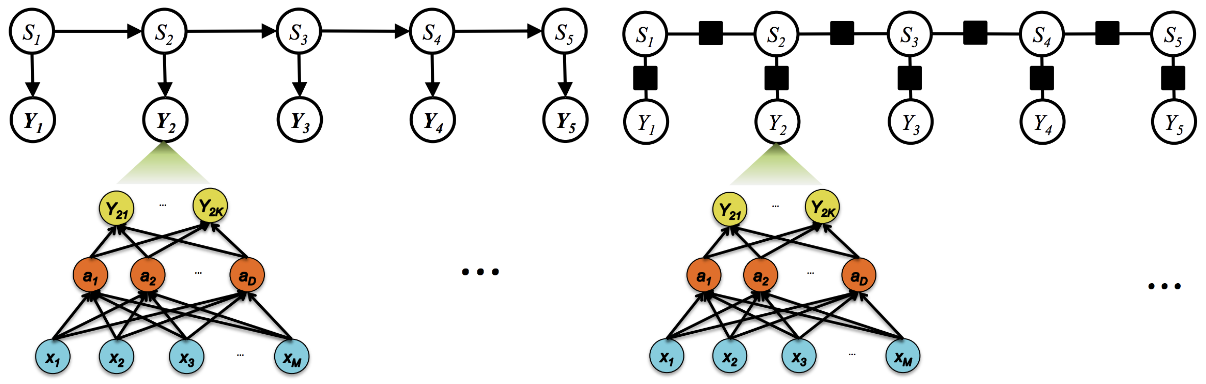


Figure 2: Example of Hybrid NN + HMM for Phoneme Recognition

A hybrid model of neural networks and HMMs is discussed with a case study of task of Phoneme Recognition, as seen in Figure 2 (normal and factor graph notation). The hidden states, $S_1, ..., S_N$ are phonemes, i.e., $St \in /p/, /t/, /k/, ..., /g/$, and emissions, $Y_1, ..., Y_T$, where $Y_t \in R^k$. In this example the HMM has Gaussian emissions at each state - it is a mixture of Gaussian given by:

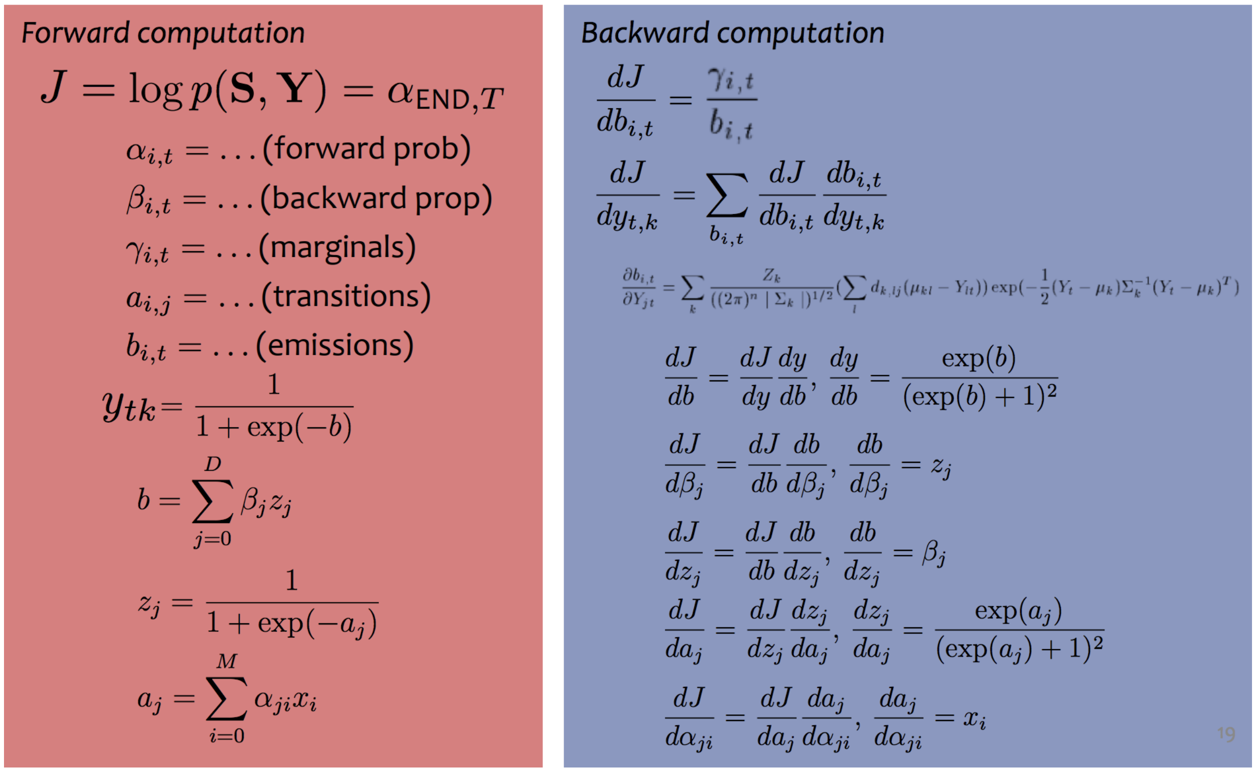
$$p(Y_t \mid S_t = i) = \sum_k \frac{Z_k}{\sqrt{(2\pi)^n |\Sigma_k|}} exp\left(-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T\right)$$

Each of the emission nodes in HMM is attached to the same neural network. Input to each network is a fixed number of overlapping frames of the speech signal and the output is the learned features. The emissions are simultaneously generated in a top-down fashion by the HMM and in a bottom-up fashion by the NN. Emissions are not actually observed but only appear as a function of the observed speech signal. At test time, speech signal are fed as input into the NN and resulting features are used for each timestep as observations in the HMM. Finally, the forward-backward algorithm is run on the HMM to compute the alpha-beta probabilities and the gamma (marginal) probabilities.

$$\alpha_{i,t} = P(Y_1^t, S_t = i \mid model) = b_{i,t} \sum_j a_{j,i}\alpha_{j,t-1}$$
$$\beta_{i,t} = P(Y_{i+1}^T \mid S_t = i, model) = \sum_j a_{j,i} b_{j,t+1}\beta_{j,t+1}$$
$$\gamma_{i,t} = P(S_t = i \mid Y_1^t, model) = \alpha_{i,t}\beta_{i,t}$$

where, $a_{i,j} = P(S_t = 1|S_{t-1} = j)$ and $b_{i,t} =, P(Y_t|S_t = i)$ are the transition and emission probabilities respectively. Log-likelihood for feed-forward objective function is given by $\log p(S,Y) = \alpha_{end,T}$.

All the feed-forward computations are differentiable since there is no longer an argmax operation but only summations and products to compute the αs and β. The gradient for updating the network is computed by using the regular chain rule. We compute the gradient as the partial derivatives for the graphical model and partial derivatives for the neural network, i.e, partial derivative of the log-likelihood with respect to the emissions and partial derivative of the emission with respect to the neural network parameters. A summary of the end-to-end training of this hybrid model is given below for reference.



Forward computation

$$J = \log p(\mathbf{S}, \mathbf{Y}) = \alpha_{\text{END},T}$$
$$\alpha_{i,t} = \ldots \text{(forward prob)}$$
$$\beta_{i,t} = \ldots \text{(backward prop)}$$
$$\gamma_{i,t} = \ldots \text{(marginals)}$$
$$a_{i,j} = \ldots \text{(transitions)}$$
$$b_{i,t} = \ldots \text{(emissions)}$$
$$y_{tk} = \frac{1}{1+\exp(-b)}$$
$$b = \sum_{j=0}^{D} \beta_j z_j$$
$$z_j = \frac{1}{1+\exp(-a_j)}$$
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

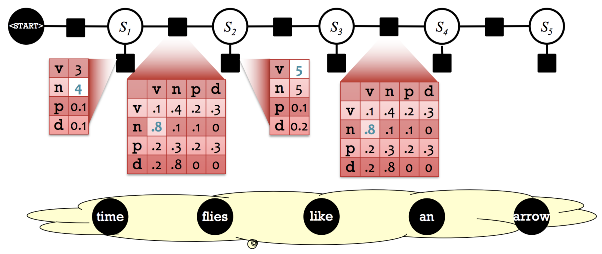
Backward computation

$$\frac{dJ}{db_{i,t}} = \frac{\gamma_{i,t}}{b_{i,t}}$$
$$\frac{dJ}{dy_{t,k}} = \sum_{b_{i,t}} \frac{dJ}{db_{i,t}} \frac{db_{i,t}}{dy_{t,k}}$$
$$\frac{\partial b_{i,t}}{\partial Y_{jt}} = \sum_k \frac{Z_k}{((2\pi)^n \mid \Sigma_k \mid)^{1/2}} (\sum_l d_{k,lj}(\mu_{kl} - Y_{lt})) \exp(-\frac{1}{2}(Y_t - \mu_k)\Sigma_k^{-1}(Y_t - \mu_k)^T)$$
$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \ \frac{dy}{db} = \frac{\exp(b)}{(\exp(b)+1)^2}$$
$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \ \frac{db}{d\beta_j} = z_j$$
$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \ \frac{db}{dz_j} = \beta_j$$
$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \ \frac{dz_j}{da_j} = \frac{\exp(a_j)}{(\exp(a_j)+1)^2}$$
$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \ \frac{da_j}{d\alpha_{ji}} = x_i$$

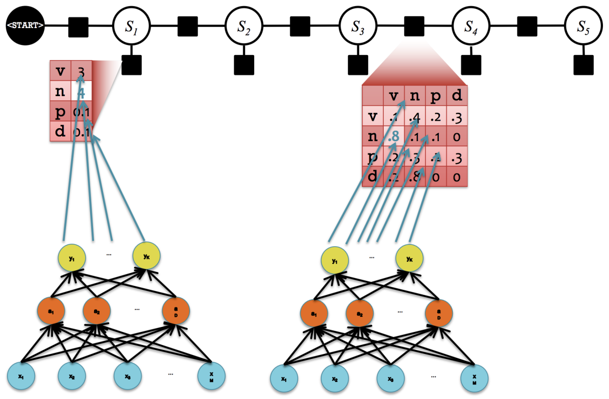
19

2.2 Neural Network (CNN) + Conditional Random Field (CRF)

In a standard CRF, each of the factor cells is a parameter (e.g. transition or emission). In the hybrid model, these values are computed by a neural network with its own parameters. An example of hybrid model with a CNN and CRF is presented in Figure 4. This model allows us to compute the values of the factors using a CNN. We allow each factor to have its own parameters (i.e. its own CNN). This hybrid model is particularly suited for Natural Language Processing applications such as part-of-speech tagging, noun-phrase and verb-phrase chunking, named-entity recognition, and semantic role labeling. In such NLP tasks, the CNN is 1-dimensional. The output of the CNN is fed into the input of the CRF. Experiments on these tasks show that the CNN+HMM model gives results close to the state of the art on benchmark systems. In addition, the advantage of this model is that it does not require hand-engineered features.



(a) Traditional CRF



(b) Hybrid model: CNN + CRF

3 Variational Autoencoder

Variational Autoencoder (VAE) is an unsupervised learning method. Recall from our lectures on approximate inference that when we are unable to perform exact learning, we can cast the learning problem as an optimization problem over the evidence lower bound (ELBO), which is given as follows:

$$F(y,q) = \mathbb{E}_{q(z)}[\log p(y|z)] - KL(q(z)||p(z))$$

where $q(z)$ the approximate posterior distribution, in other words the best match to true posterior $p(z|y)$, one of the unknown inferential quantities of interest to us. $q(z)$ is a member of a parametrized family of distributions that are easy to compute. $\mathbb{E}_{q(z)}[\log p(y|z)]$ is the reconstruction Cost, or the expected log-likelihood to measure how well samples from $q(z)$ are able to explain the data. $KL(q(z)||p(z))$ is the penalty and ensures the the explanation of the data using approximate posterior $q(z)$ doesnt deviate too far from your beliefs $p(z)$.

From the encoder decoder view, given data x, the encoder represents a variational distribution $q_\phi(z|x)$ and the decoder represents the distribution $p_\theta(x|z)$. The goal is to optimize the network such that $q_\phi(z|x)$ and $p_\theta(x|z)$ are close: i.e. finding an approximate posterior that is as close as possible to the true posterior.

In an VAE we use the mean field approximation that the approximate posterior is a member of the mean field family of distributions:

$$q_\phi(z|x) = q_\phi(z_1, ..., z_M|x) = \prod_{j=1}^{M} q_\phi(z_j|Pa(z_j), x)$$

We can further force all factors to share parameters and this is called amortization. For example, we can parametrize the encoder as follows:

$$(\mu, \log \sigma) = \text{EncoderNeuralNet}(x)$$

$$q_\phi(z|x) = N(z; \mu, diag(\sigma))$$

Although this is a simple method it can actually give rise to a large family of distributions. Note that although $q_\phi(z|x)$ is Gaussian, the parameters of this Gaussian are functions of x i.e. $\mu(x), \sigma(x)$. This implies that $q_\phi(z)$ can represent a large family of distributions, not necessarily just a Gaussian distribution.

Overall, VAE learns a mapping that transforms a simple distribution, $q(z)$, to a complicated distribution, $q(z|x)$ so that the empirical distribution (in the x-space) is matched.

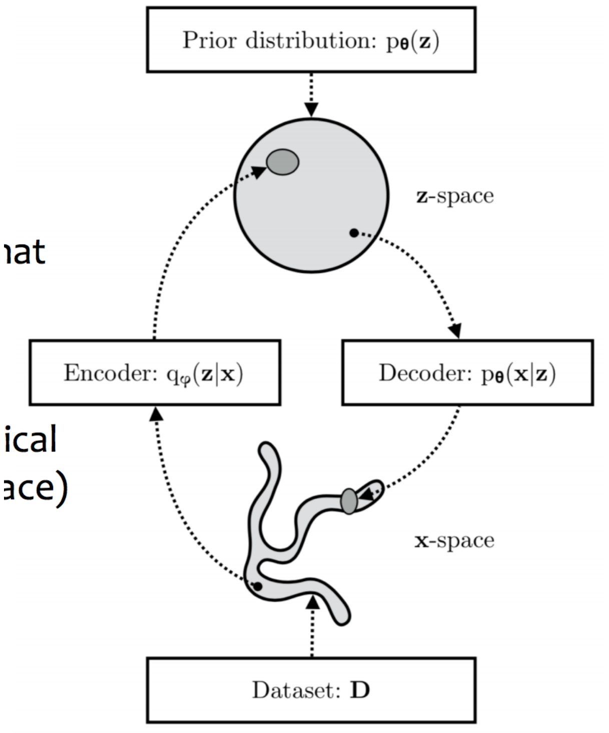

Prior distribution: $p_{\boldsymbol{\theta}}(\mathbf{z})$
$\mathbf{z}$-space
ıat
Encoder: $q_{\varphi}(\mathbf{z}|\mathbf{x})$
Decoder: $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$
ical
ace)
$\mathbf{x}$-space
Dataset: $\mathbf{D}$

Figure 4: VAE encoder and decoder map distributions from input space (x) to latent space z.

Learning an VAE involves optimizing the following loss function:

$$\mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)]$$

Taking the gradients with respect to θ and ϕ:

$$\nabla_\theta \mathcal{L}_{\theta,\phi}(x) = \mathbb{E}_{q_\phi(z|x)}[\nabla_\theta \log p_\theta(x, z)]$$

$$\nabla_\phi \mathcal{L}_{\theta,\phi}(x) = \nabla_\phi \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)]$$

The first term is easy but the second term is harder to compute: we cannot bring the ∇_ϕ into the $\mathbb{E}_{q_\phi(z|x)}$ term since the expectation is a function of ϕ. To compute this we use the reparametrization trick. Suppose $\epsilon \sim p(\epsilon)$. Then define the change of variables $z = g(\epsilon, \phi, x)$. As as example, if $\epsilon \sim N(0, 1)$ and $z \sim N(\mu, \sigma^2)$

then the change of variables reparametrization would yield $z = \mu + \sigma\epsilon$. And in the multivariate case if $\epsilon \sim N(0, I)$, $z \sim N(\mu, \Sigma)$ then change of variables reparametrization would yield $z = \mu + \Sigma^{1/2}\epsilon$.

Applying this to our problem, we use the reparametrization trick as follows:

$$\nabla_\phi \mathbb{E}_{q_\phi(z|x)}[f(z)] = \nabla_\phi \mathbb{E}_{p(\epsilon)}[f(z)] = \mathbb{E}_{p(\epsilon)}[\nabla_\phi f(z)]$$

where the first equality follows since z is now written as a deterministic function of ϵ: $z = g(\epsilon, \phi, x)$ and the second equality follows since ϵ is not a function of ϕ. Finally $\mathbb{E}_{p(\epsilon)}[\nabla_\phi f(z)]$ can then be approximated by sampling over different values of ϵ.

Going back to the learning, we have that:

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x, z) - \log q_\phi(z|x)] \\ &= \mathbb{E}_{p(\epsilon)}[\log p_\theta(x, z) - \log q_\phi(z|x)]\end{aligned}$$

using the reparametrization trick. Now can we improve the estimation of $\log q_\phi(z|x)$ by reducing the variance? Let's revisit the change of variables for the second term:

$$\log q_\phi(z|x) = \log p(\epsilon) - \log \left| det\left(\frac{\partial z}{\partial \epsilon}\right)\right|$$

In some cases we can write the $\log \left| det\left(\frac{\partial z}{\partial \epsilon}\right)\right|$ exactly. As an example, if $\epsilon \sim N(0, I)$, $(\mu, \log \sigma) = \text{EncoderNeuralNet}(x)$ and $z = \mu + \sigma \odot \epsilon$, then we can compute $\log \left| det\left(\frac{\partial z}{\partial \epsilon}\right)\right|$ exactly as $\sum_i \sigma_i$.

There are some examples at `http://www.dpkingma.com/sgvb_mnist_demo/demo.html` and variants of VAE have been proposed using convolutional and recurrent encoders and decoders for image and language applications respectively.

We can use the VAE technique in graphical models. As an example, consider the mixture model case where instead of a simple mixture of Gaussians we have a mixture of more complex distributions:

$$\pi \sim Dir(\alpha)$$

$$(\mu_k, \Sigma_k) \sim NIW(\lambda)$$

$$z_n|\pi \sim \pi$$

$$y_n|z_n\{(\mu_k, \Sigma_k)_{k=1}^K\} \sim N(\mu_{z_n}, \Sigma_{z_n})$$

Then we could use a VAE that learns generation of the data through the following:

$$\pi \sim Dir(\alpha)$$

$$(\mu_k, \Sigma_k) \sim NIW(\lambda)$$

$$z_n|\pi \sim \pi$$

$$x_n \sim N(\mu(z_n), \Sigma(z_n))$$

$$y_n|x_n \sim N(\mu(x_n; \gamma), \Sigma(x_n; \gamma))$$

Here only the last 2 lines have changed: we have introduced auxiliary random variables x_n which pass through a complex decoder to generate $y_n|x_n$. This complex decoder allows us to model complex distributions in input space y_n while the distribution of x_n is a simple Gaussian distribution. For a video demonstration, see `https://www.youtube.com/watch?v=btr1poCYIzw&t=60s`.

4 Generative Adversarial Network

(there was not enough time and this was covered in the next class)