Autoregressive models

We begin our study into generative modeling with autoregressive models. As before, we assume we are given access to a dataset $\mathcal{D}$ of n-dimensional datapoints $\mathbf{x}$. For simplicity, we assume the datapoints are binary, i.e., $\mathbf{x} \in \{0,1\}^n$.

Representation

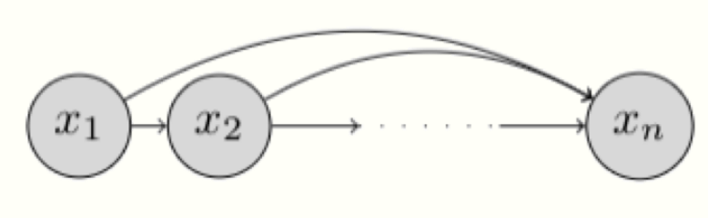
Original Probability distribution (chain rule)

By the chain rule of probability, we can factorize the joint distribution over the n-dimensions as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|x_1, x_2, \ldots, x_{i-1}) = \prod_{i=1}^{n} p(x_i|\mathbf{x}_{<i})$$

where $\mathbf{x}_{<i} = [x_1, x_2, \ldots, x_{i-1}]$ denotes the vector of random variables with index less than i.

The chain rule factorization can be expressed graphically as a Bayesian network.



Graphical model for an autoregressive Bayesian network with no conditional independence assumptions.

Such a Bayesian network that makes no conditional independence assumptions is said to obey the *autoregressive* property. The term *autoregressive* originates from the literature on time-series models where observations from the previous time-steps are used to predict the value at the current time step. Here, we fix an ordering of the variables $x_1, x_2, \ldots, x_n$ and the distribution for the i-th random variable depends on the values of all the preceeding random variables in the chosen ordering $x_1, x_2, \ldots, x_{i-1}$.

Autoregressive Model : Bayesian Network에 "No conditional independence" 가정을 추가한 것이다.

만약 Probability를 이러한 가정없이 chain rule로 계산한다면,
$P(x_i|X_{<i})$를 계산하기 위해서 최상 model은 2^{i-1}개의 parameter가 필요하며 n-dimensional data의 경우
total $1+2^1+2^2+\cdots+2^{n-1}=2^n-1$ 개의 parameter가 필요하다.

Original chain rule.
$p(x_1,x_2,x_3,x_4) = p(x_1)\cdot p(x_2|x_1)\cdot p(x_3|x_1,x_2)\cdot p(x_4|x_1,x_2,x_3)$

Autoregressive Models.
$P(x_1,x_2,x_3,x_4) \approx P_{CPT}(x_1;\alpha^1)\, P_{logit}(x_2|x_1;\alpha^2)\, P_{logit}(x_3|x_1,x_2;\alpha^3)\, P_{logit}(x_4|x_1,x_2,x_3;\alpha^4)$

If we allow for every conditional $p(x_i|\mathbf{x}_{<i})$ to be specified in a tabular form, then such a representation is fully general and can represent any possible distribution over n random variables. However, the space complexity for such a representation grows exponentially with n.

따라서 Autoregressive generative model은 제한된 수의 parameters로 이러한 conditional distribution을 표시해내는 것을 가정한다.

$P_{\theta_i}(x_i|X_{<i}) = Bern(f_i(x_1,x_2,\cdots,x_{i-1}))$
θ_i denotes the set of parameters
$f_i : \{0,1\}^{i-1} \longrightarrow [0,1]$
즉, f_i function의 output이 의미하는 바는 "1"이 될 확률

To see why, let us consider the conditional for the last dimension, given by $p(x_n|\mathbf{x}_{<n})$. In order to fully specify this conditional, we need to specify a probability for 2^{n-1} configurations of the variables $x_1, x_2, \ldots, x_{n-1}$. Since the probabilities should sum to 1, the total number of parameters for specifying this conditional is given by $2^{n-1} - 1$. Hence, a tabular representation for the conditionals is impractical for learning the joint distribution factorized via chain rule.

In an *autoregressive generative model*, the conditionals are specified as parameterized functions with a fixed number of parameters. That is, we assume the conditional distributions $p(x_i|\mathbf{x}_{<i})$ to correspond to a Bernoulli random variable and learn a function that maps the preceeding random variables $x_1, x_2, \ldots, x_{i-1}$ to the mean of this distribution. Hence, we have

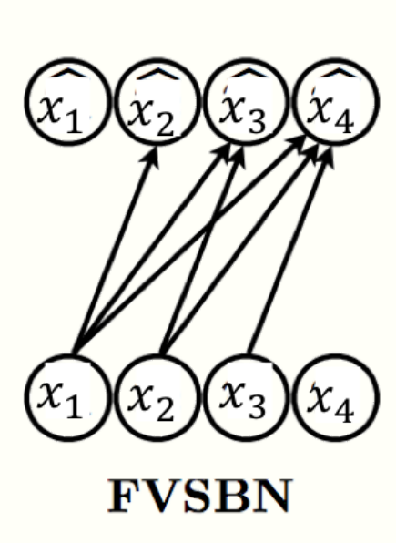
$$\boxed{p_{\theta_i}(x_i|\mathbf{x}_{<i}) = \mathrm{Bern}(f_i(x_1, x_2, \ldots, x_{i-1}))}$$

where θ_i denotes the set of parameters used to specify the mean function $f_i : \{0,1\}^{i-1} \to [0,1]$.

For n-dimensional data.
• Chain Rule로 Probability distribution을 표시할 때 필요한 parameter 수.
: $1+2^1+2^2+\cdots+2^{n-1} = 2^n-1$
• Autoregressive Model로 Probability distribution 표시할 때 parameter 수.
: $\sum_{i=1}^{n}|\theta_i|$

The number of parameters of an autoregressive generative model are given by $\sum_{i=1}^{n} |\theta_i|$. As we shall see in the examples below, the number of parameters are much fewer than the tabular setting considered previously. Unlike the tabular setting however, an autoregressive generative model cannot represent all possible distributions. Its expressiveness is limited by the fact that we are limiting the conditional distributions to correspond to a Bernoulli random variable with the mean specified via a restricted class of parameterized functions.

Autoregressive Model을 사용할 경우 Probability distribution을 표시하는데 필요한 parameter 수는 감소하지만,
그에 상응하는 대가로 Autoregressive model이 모든 발생가능한 경우들에 대한 distribution을 표현할 수 없음을 이해해야 한다.



FVSBN의 표현하는 Conditional distribution은 아래의 수식처럼 가정하여 계산된다.

A fully visible sigmoid belief network over four variables. The conditionals are denoted by $\widehat{x}_1, \widehat{x}_2, \widehat{x}_3, \widehat{x}_4$ respectively.

In the simplest case, we can specify the function as a linear combination of the input elements followed by a sigmoid non-linearity (to restrict the output to lie between 0 and 1). This gives us the formulation of a *fully-visible sigmoid belief network* (FVSBN).

$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\alpha_0^{(i)} + \alpha_1^{(i)} x_1 + \ldots + \alpha_{i-1}^{(i)} x_{i-1})$$

where σ denotes the sigmoid function and $\theta_i = \{\alpha_0^{(i)}, \alpha_1^{(i)}, \ldots, \alpha_{i-1}^{(i)}\}$ denote the parameters of the mean function. The conditional for variable i requires i parameters, and hence the total number of parameters in the model is given by $\sum_{i=1}^{n} i = O(n^2)$. Note that the number of parameters are much fewer than the exponential complexity of the tabular case.

FVSBN 관련설명
NADE 관련설명

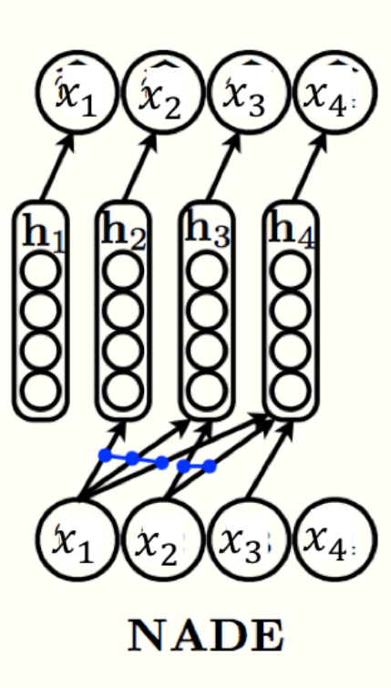
A natural way to increase the expressiveness of an autoregressive generative model is to use more flexible parameterizations for the mean function e.g., multi-layer perceptrons (MLP). For example, consider the case of a neural network with 1 hidden layer. The mean function for variable i can be expressed as

expressiveness를 높이기 위해 mean function을 MLP로 교체
• 기존: $\alpha_0^{(i)} + \alpha_1^{(i)}x_1 + \alpha_2^{(i)}x_2 + \cdots + \alpha_{i-1}^{(i)}x_{i-1}$
• 변경: $h_i = \sigma(A_i x_{<i} + c_i)$
$f_i(x_1,x_2,\cdots,x_{i-1}) = \sigma(\alpha^{(i)} h_i + b_i)$

간단한 예시 1 hidden layer 가정.

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i) \quad \mathbb{R}^d$$
$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\boldsymbol{\alpha}^{(i)} \mathbf{h}_i + b_i)$$

where $\mathbf{h}_i \in \mathbb{R}^d$ denotes the hidden layer activations for the MLP and $\theta_i = \{A_i \in \mathbb{R}^{d\times(i-1)}, \mathbf{c}_i \in \mathbb{R}^d, \boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d, b_i \in \mathbb{R}\}$ are the set of parameters for the mean function $\mu_i(\cdot)$. The total number of parameters in this model is dominated by the matrices A_i and given by $O(n^2 d)$.



A neural autoregressive density estimator over four variables. The conditionals are denoted by $\widehat{x}_1, \widehat{x}_2, \widehat{x}_3, \widehat{x}_4$ respectively. The blue connections denote the tied weights $W[.,i]$ used for computing the hidden layer activations.

The *Neural Autoregressive Density Estimator* (NADE) provides an alternate MLP-based parameterization that is more statistically and computationally efficient than the vanilla approach. In NADE, parameters are shared across the functions used for evaluating the conditionals. In particular, the hidden layer activations are specified as

W[:, :i] slicing.

$$\mathbf{h}_i = \sigma(W_{.,<i} \mathbf{x}_{<i} + \mathbf{c})$$
$$f_i(x_1, x_2, \ldots, x_{i-1}) = \sigma(\boldsymbol{\alpha}^{(i)} \mathbf{h}_i + b_i)$$

where $\theta = \{W \in \mathbb{R}^{d\times n}, \mathbf{c} \in \mathbb{R}^d, \{\boldsymbol{\alpha}^{(i)} \in \mathbb{R}^d\}_{i=1}^n, \{b_i \in \mathbb{R}\}_{i=1}^n\}$ is the full set of parameters for the mean functions $f_1(\cdot), f_2(\cdot), \ldots, f_n(\cdot)$. The weight matrix W and the bias vector $\mathbf{c}$ are shared across the conditionals. Sharing parameters offers two benefits:

1. The total number of parameters gets reduced from $O(n^2 d)$ to $O(nd)$ [readers are encouraged to check!].
2. The hidden unit activations can be evaluated in $O(nd)$ time via the following recursive strategy:

$$\mathbf{h}_i = \sigma(\mathbf{a}_i)$$
$$\mathbf{a}_{i+1} = \mathbf{a}_i + W[., i] x_i$$

with the base case given by $\mathbf{a}_1 = \mathbf{c}$.

즉, 1. parameter 수 감소
2. computation cost 감소.

Extensions to NADE

The RNADE algorithm extends NADE to learn generative models over real-valued data. Here, the conditionals are modeled via a continuous distribution such as a equi-weighted mixture of K Gaussians. Instead of learning a mean function, we know learn the means $\mu_{i,1}, \mu_{i,2}, \ldots, \mu_{i,K}$ and variances $\Sigma_{i,1}, \Sigma_{i,2}, \ldots, \Sigma_{i,K}$ of the K Gaussians for every conditional. For statistical and computational efficiency, a single function $g_i : \mathbb{R}^{i-1} \to \mathbb{R}^{2K}$ outputs all the means and variances of the K Gaussians for the i-th conditional distribution.

Notice that NADE requires specifying a single, fixed ordering of the variables. The choice of ordering can lead to different models. The EoNADE algorithm allows training an ensemble of NADE models with different orderings.

Learning and inference

KL divergence는 두 distribution 간의 closeness를 판단하는 metrics로 사용된 경우.

Recall that learning a generative model involves optimizing the closeness between the data and model distributions. One commonly used notion of closeness in the KL divergence between the data and the model distributions.

즉, distance metric은 KL divergence.

$d(P_{data}, P_\theta)$ P_{data} $\theta \in \mathcal{M}$ Model family

$$\min_{\theta\in\mathcal{M}} d_{KL}(p_{\text{data}}, p_\theta) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log p_{\text{data}}(\mathbf{x}) - \log p_\theta(\mathbf{x})\right]$$

KL Divergence.
$d_{KL}(P,Q) = \sum_{x\in X} P(x) \log_b \frac{P(x)}{Q(x)} = \mathbb{E}_{x\in X}[\log_b P(x) - \log_b Q(x)]$
$= \mathbb{E}_{x\in X}[\log_b P(x)] - \mathbb{E}_{x\in X}[\log_b Q(x)]$

Before moving any further, we make two comments about the KL divergence. First, we note that 1 the KL divergence between any two distributions is asymmetric. As we navigate through this chapter, the reader is encouraged to think what could go wrong if we decided to optimize the reverse KL divergence instead. Secondly, 2 the KL divergences heavily penalizes any model distribution p_θ (which assigns low probability to a datapoint that is likely to be sampled under p_{data}.) In the extreme case, if the density $p_\theta(\mathbf{x})$ evaluates to zero for a datapoint sampled from p_{data}, the objective evaluates to $+\infty$.

일반적으로 P_{data}와 P_{model} data distribution 사이의 유사성을 검토하기 위해 KL divergence를 많이 이용할 수 있다.
하지만 KL divergence는 위 2개의 문제를 갖는다.
1. Asymmetric
2. KL divergence는 P_{data} (실제 분포)상에서 존재할 확률이 있는 (여기서는) data point에 대해 $P_\theta(x)$가 낮은확률을 보일 때 heavily penalizes한다.
ex) $P_\theta(x)=0,\ P_{data}(x)\neq 0 \rightarrow \log(0) = -\infty \rightarrow \infty$. objective function $\rightarrow \infty$.

Since p_{data} does not depend on θ, we can equivalently recover the optimal parameters via maximizing likelihood estimation.

$$\max_{\theta\in\mathcal{M}} \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}\left[\log p_\theta(\mathbf{x})\right].$$

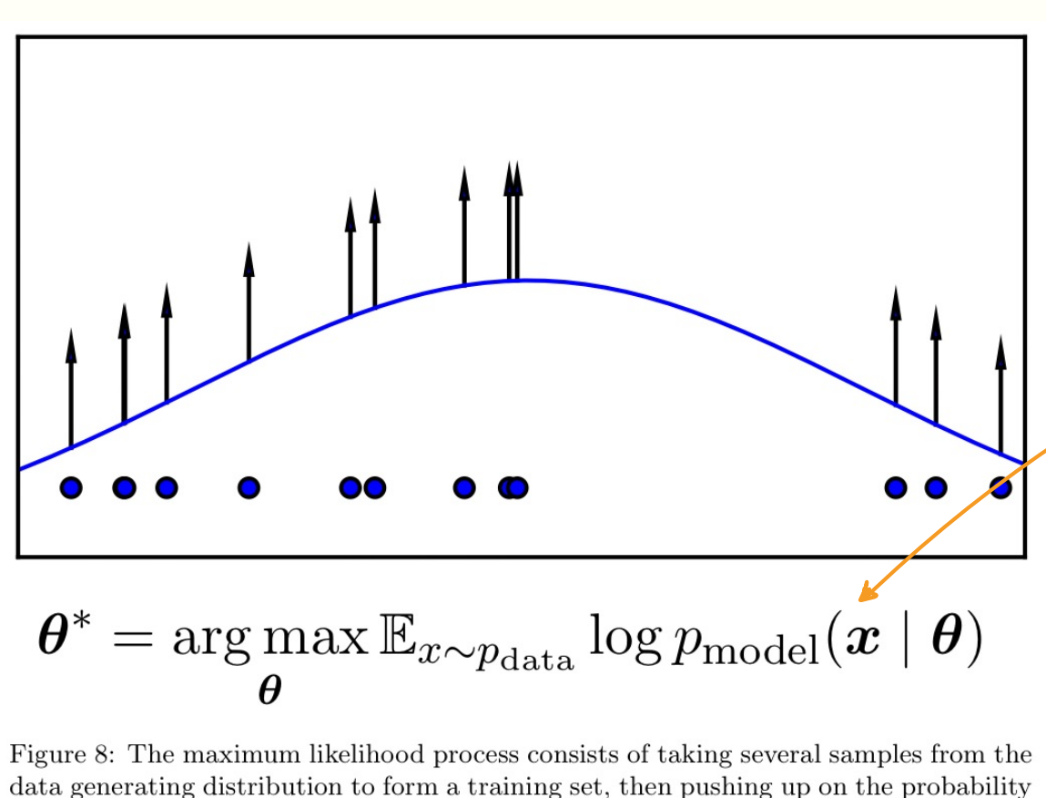


Figure 8: The maximum likelihood process consists of taking several samples from the data generating distribution to form a training set, then pushing up on the probability the model assigns to those points, in order to maximize the likelihood of the training data. This illustration shows how different data points push up on different parts of the density function for a Gaussian model applied to 1-D data. The fact that the density function must sum to 1 means that we cannot simply assign infinite likelihood to all points; as one point pushes up in one place it inevitably pulls down in other places. The resulting density function balances out the upward forces from all the data points in different locations.

Here, $\log p_\theta(\mathbf{x})$ is referred to as the log-likelihood of the datapoint $\mathbf{x}$ with respect to the model distribution p_θ.

To approximate the expectation over the unknown p_{data}, we make an assumption: points in the dataset $\mathcal{D}$ are sampled i.i.d. from p_{data}. This allows us to obtain an unbiased Monte Carlo estimate of the objective as

$$\max_{\theta\in\mathcal{M}} \frac{1}{|D|} \sum_{\mathbf{x}\in\mathcal{D}} \log p_\theta(\mathbf{x}) = \mathcal{L}(\theta|\mathcal{D}).$$

The maximum likelihood estimation (MLE) objective has an intuitive interpretation: pick the model parameters $\theta \in \mathcal{M}$ that maximize the log-probability of the observed datapoints in $\mathcal{D}$.

In practice, we optimize the MLE objective using mini-batch gradient ascent. The algorithm operates in iterations. At every iteration t, we sample a mini-batch $\mathcal{B}_t$ of datapoints sampled randomly from the dataset ($|\mathcal{B}_t| < |\mathcal{D}|$) and compute gradients of the objective evaluated for the mini-batch. These parameters at iteration $t+1$ are then given via the following update rule

$$\theta^{(t+1)} = \theta^{(t)} + r_t \nabla_\theta \mathcal{L}(\theta^{(t)}|\mathcal{B}_t)$$

where $\theta^{(t+1)}$ and $\theta^{(t)}$ are the parameters at iterations $t+1$ and t respectively, and r_t is the learning rate at iteration t. Typically, we only specify the initial learning rate r_1 and update the rate based on a schedule. Variants of stochastic gradient ascent, such as RMS prop and Adam, employ modified update rules that work slightly better in practice.

From a practical standpoint, 1 we must think about how to choose hyperparameters (such as the initial learning rate) and a 2 stopping criteria for the gradient descent. For both these questions, we follow the standard practice in machine learning of monitoring the objective on a validation dataset. Consequently, we choose the hyperparameters with the best performance on the validation dataset and stop updating the parameters when the validation log-likelihoods cease to improve[1].

Now that we have a well-defined objective and optimization procedure, the only remaining task is to evaluate the objective in the context of an autoregressive generative model. To this end, we substitute the factorized joint distribution of an autoregressive model in the MLE objective to get

$$\max_{\theta\in\mathcal{M}} \frac{1}{|D|} \sum_{\mathbf{x}\in\mathcal{D}} \sum_{i=1}^{n} \log p_{\theta_i}(x_i|\mathbf{x}_{<i})$$

where $\theta = \{\theta_1, \theta_2, \ldots, \theta_n\}$ now denotes the collective set of parameters for the conditionals.

Inference in an autoregressive model is straightforward. For density estimation of an arbitrary point $\mathbf{x}$, we simply evaluate the log-conditionals $\log p_{\theta_i}(x_i|\mathbf{x}_{<i})$ for each i and add these up to obtain the log-likelihood assigned by the model to $\mathbf{x}$. Since we know conditioning vector $\mathbf{x}$, each of the conditionals can be evaluated in parallel. Hence, density estimation is efficient on modern hardware.

* Sampling from an autoregressive model is a sequential procedure. Here, we first sample x_1, then we sample x_2 conditioned on the sampled x_1, followed by x_3 conditioned on both x_1 and x_2 and so on until we sample x_n conditioned on the previously sampled $\mathbf{x}_{<n}$. For applications requiring real-time generation of high-dimensional data such as audio synthesis, the sequential sampling can be an expensive process. Later in this course, we will discuss how parallel Wavenet, an autoregressive model sidesteps this expensive sampling process.

Finally, an autoregressive model does not directly learn unsupervised representations of the data. In the next few set of lectures, we will look at latent variable models (e.g., variational autoencoders) which explicitly learn latent representations of the data.

Footnotes

1. Given the non-convex nature of such problems, the optimization procedure can get stuck in local optima. Hence, early stopping will generally not be optimal but is a very practical strategy. ↩