Course Materials for GEN-AI

Northeastern University

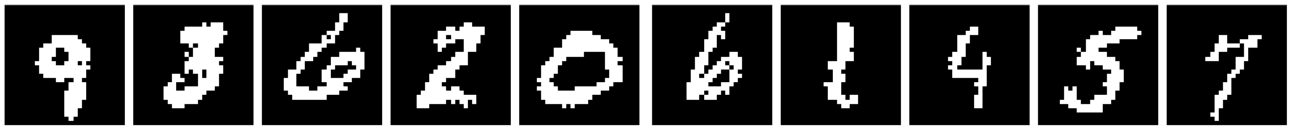
Instructor: Ramin Mohammadi
`r.mohammadi@northeastern.edu`

Thank you for your understanding and collaboration.

Autoregressive Models

Motivating Example: MNIST

Suppose we have a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST) and we want to learn a generative model over these images.



- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Goal:** We want to learn a probability distribution $p(x) = p(x_1, \cdots, x_{784})$ over $v \in \{0,1\}^{784}$ such that when $x \sim p(x)$, x looks like a digit.
- **Two Step Process:**
  - Define a model family of parametrized distribution $\{p_\theta(x), \theta \in \Theta\}$.
  - then pick a good one based on training data $\mathcal{D}$ by searching over parameter space and pick the best θ that optimizes the learning problem. (more on that later).
- **How to parameterize $p_\theta(x)$?**
- We can use autoregressive to define this probability distribution.

Autoregressive Models

To use autoregressive to define this probability distribution. First, We can pick an ordering, i.e., order variables (pixels) from top-left (X_1) to bottom-right ($X_{n=784}$).
For image is not that obvious what the ordering should be, as there is not an obvious kind of causal structure. We are not modeling time-series where you might expect that there is some causal structure and predicting the future given the past is easier than going backwards. However, any ordering might work in principal. Ex: Raster scanning over an image.
Use chain rule factorization without loss of generality:

- $$p(x_1, \cdots, x_{784}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_n|x_1, \cdots, x_{n-1})$$
- This is a common way to break down a generative modeling problem into sequence, small numbers of classification/regression.

Some conditionals are too complex to be stored in tabular form (CPT) so instead we formulate these conditionals using some kind of NNs model. So we assume:

$$p(x_1, \cdots, x_{784}) = p_{\text{CPT}}(x_1; \alpha_1) p_{\text{logit}}(x_2|x_1; \alpha_2) p_{\text{logit}}(x_3|x_1, x_2; \alpha_3) \cdots p_{\text{logit}}(x_n|x_1, \cdots, x_{n-1}; \alpha_n)$$

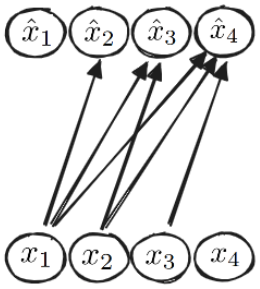
More explicitly:

- The first random variable can be defined using CPT as :
$$p_{\text{CPT}}(x_1 = 1; \alpha_1) = \alpha_1, \quad p(x_1 = 0) = 1 - \alpha_1$$
- One simple idea is to just use Logistic Regression to formulate conditional for each pixel.
$$p_{\text{logit}}(x_2 = 1|x_1; \alpha_2) = \sigma(\alpha_2^0 + \alpha_2^1 x_1)$$
- $$p_{\text{logit}}(x_3 = 1|x_1, x_2; \alpha_3) = \sigma(\alpha_3^0 + \alpha_3^1 x_1 + \alpha_3^2 x_2)$$
- More explicity:
$$p_{\text{CPT}}(X_1 = 1; \alpha^1) = \alpha^1, \quad p(X_1 = 0) = 1 - \alpha^1$$
$$p_{\text{logit}}(X_2 = 1 \mid x_1; \alpha^2) = \sigma(\alpha_0^2 + \alpha_1^2 x_1)$$
$$p_{\text{logit}}(X_3 = 1 \mid x_1, x_2; \alpha^3) = \sigma(\alpha_0^3 + \alpha_1^3 x_1 + \alpha_2^3 x_2)$$

Note: This is a modeling assumption (e.g., Using Logistic Regression as a parametrized function). Whether or not this modeling assumption works depends on if its easy to predict the value of a pixel given the previous ones. We can look at this model as a **autoregressive** model as we are regressing given previous data parts.
This kind of assumption has been tried before and this kind of model is called "Fully Visible Sigmoid Belief Network".

Fully Visible Sigmoid Belief Network - (FVSBN)

relatively simple, early type of generative model that don't work particularly well but is good for understanding.



The conditional variables $x_i|x_1, \cdots, x_{i-1}$ are Bernoulli with parameters:

$$\hat{x}_i = p(x_i = 1|x_1, \cdots, x_{i-1}; \alpha^i) = p(x_i = 1|x_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

How to evaluate $p(x_1, \cdots, x_{784})$**?** How likely is a given data-point according to a our model?
Multiply all the conditionals (factors):
In the above example:

$$p(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0) = (1 - \hat{x}_1) \times \hat{x}_2 \times \hat{x}_3 \times (1 - \hat{x}_4)$$
$$= (1 - \hat{x}_1) \times \hat{x}_2(x_1 = 0) \times \hat{x}_3(x_1 = 0, x_2 = 1) \times (1 - \hat{x}_4(x_1 = 0, x_2 = 1, x_3 = 1))$$

How to sample from $p(x_1, \cdots, x_{784})$**?** The good point about autoregressive models is that you can start sample/generate one point at the time given the past.

1. Sample $x_1 \sim p(x_1)$ `(np.random.choice([1,0],p=[`$\hat{x}_1$`, 1-`$\hat{x}_1$`]))`
2. Sample $x_2 \sim p(x_2|x_1 = x_1)$
3. Sample $x_3 \sim p(x_3|x_1 = x_1, x_2 = x_2) \cdots$

How many parameters? $1 + 2 + 3 \cdots + n \approx \frac{n^2}{2}$.

FVSBN Results

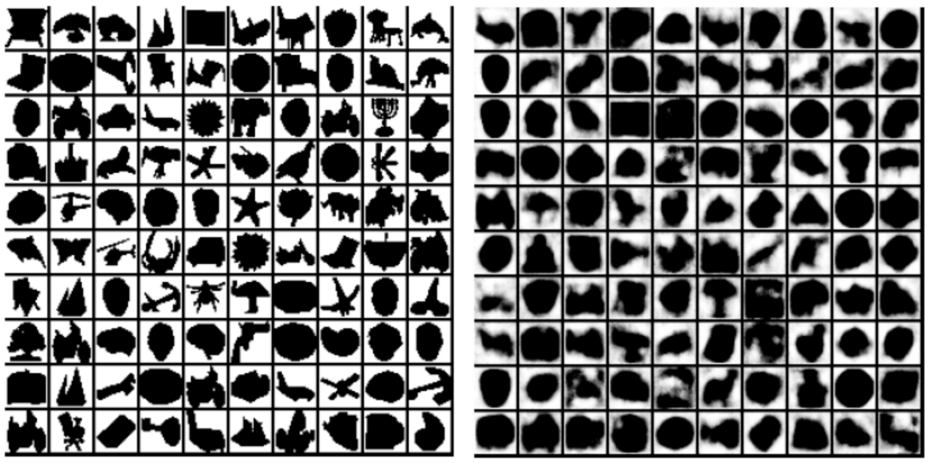


Figure: From *Learning Deep Sigmoid Belief Networks with Data Augmentation, 2015.* Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right. Best-performing model they tested on this dataset in 2015.

So how can make things better?

NADE: Neural Autoregressive Density Estimation

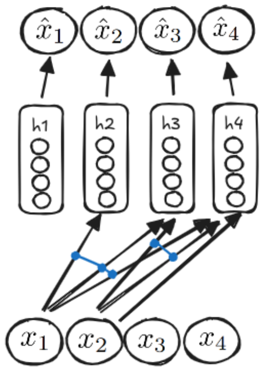
In the Neural Autoregressive Density Estimation (NADE) framework, sampling a new image involves generating each pixel value sequentially, conditioned on the previously generated pixels. Here's how this process works:

- Autoregressive Model The joint probability of the image's pixels $p(x_1, x_2, \ldots, x_d)$ is factorized as:

$$p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots p(x_d \mid x_1, x_2, \ldots, x_{d-1}).$$

  NADE models each conditional probability $p(x_i \mid x_1, x_2, \ldots, x_{i-1})$ using neural networks.

- Network Parameters At each step i, the neural network computes the parameters (e.g., logits) of the distribution $p(x_i \mid x_1, x_2, \ldots, x_{i-1})$. The parameters W, c, α, b and the activations h_i are used to compute $\hat{x}_i$, the predicted probability for pixel x_i.



Improvement: Use one-layer neural networks instead of logistic regression:

$$h_i = \sigma(A_i x_{<i} + c_i)$$

$$\hat{x}_i = p(x_i|x_1, \cdots, x_{i-1}; A_i, c_i, \alpha_i, b_i) = \sigma(\alpha_i h_i + b_i)$$

where:

$$A_i, c_i, \alpha_i, b_i$$

are the parameters of the model.

$$h_2 = \sigma\left(\underbrace{\begin{pmatrix}\vdots\end{pmatrix}}_{A_1}(x_1) + c\right)$$

$$h_3 = \sigma\left(\underbrace{\begin{pmatrix}\vdots & \vdots\end{pmatrix}}_{A_2}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \underbrace{\begin{pmatrix}\vdots\end{pmatrix}}_{c_2}\right)$$

Total number of parameters:

- $W \in R^{d*N}$.
- $c \in R^d$
- n logistic regression parameters each $\alpha_i \in R^{d*1}$ and $b_i \in R^{d*1}$

How could we reduce number of parameters and avoid learning n logistic regression models?

Tie weights: Reduce parameters and improve computation:

$$h_i = \sigma(W_{\cdot,<i} x_{<i} + c)$$

$$\hat{x}_i = p(x_i|x_1, \cdots, x_{i-1}) = \sigma(\alpha_i h_i + b_i)$$

Example:

$$h_2 = \sigma\left(W_{\cdot,<i} x_{<i} + c\right)$$

Example:

$$h_2 = \sigma\left(\begin{pmatrix}w_1\\ \vdots\end{pmatrix}(x_1) + c\right), \quad W_{\cdot,<2}$$

$$h_3 = \sigma\left(\begin{pmatrix}w_1 & w_2\\ \vdots & \vdots\end{pmatrix}\begin{pmatrix}x_1\\x_2\end{pmatrix} + \begin{pmatrix}\vdots\end{pmatrix}c\right), \quad W_{\cdot,<3}$$

$$h_4 = \sigma\left(\begin{pmatrix}w_1 & w_2 & w_3\\ \vdots & \vdots & \vdots\end{pmatrix}\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix} + \begin{pmatrix}\vdots\end{pmatrix}c\right), \quad W_{\cdot,<4}$$

If $h_i \in \mathbb{R}^d$, how many total parameters? Linear in n: weights W $\in \mathbb{R}^{n\times d}$, biases $c \in \mathbb{R}^d$, and n logistic regression coeffic
$\alpha_i, b_i \in \mathbb{R}^{d+1}$. Probability is evaluated in O(nd).

NADE Steps

- Sampling Procedure To generate a new image:
  - Start with an empty canvas: Assume no pixels are known at the start.
  - Sample the first pixel x_1: - Use the network to compute $p(x_1)$ based on initial biases or prior knowledge. - Draw a sample x_1 from the predicted distribution $p(x_1)$ (e.g., Bernoulli for binary images or a Gaussian/softmax for continuous values).
  - Iteratively condition on generated pixels: - After sampling $x_1, x_2, \dots, x_{i-1}$, compute h_i, which encodes the dependencies on the previous pixels:

    $$h_i = \sigma(A_i x_{<i} + c_i),$$

    where A_i, c_i are parameters specific to step i. - Use h_i to compute the probability distribution $p(x_i \mid x_{<i})$. - Draw a sample x_i from $p(x_i \mid x_{<i})$.
  - Repeat until all pixels are sampled: Continue this process until all d pixels are generated.

NADE results

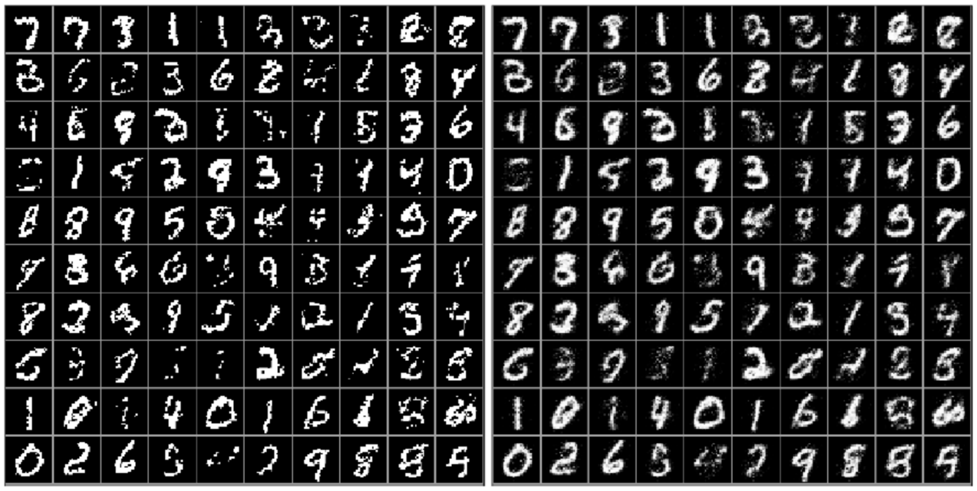


Figure from The Neural Autoregressive Distribution Estimator, 2011. Samples from the model trained on MNIST on the left. Conditional probabilities corresponding to these samples $\hat{x_i}$ on the right.
In the sampling process for NADE, the *right image (probabilistic reconstruction)* is generated first, and then the *left image (hard samples)* is generated based on it. Here's the detailed reasoning:

- **Sequential Sampling Process for NADE:**
  - **Interplay Between Right and Left Images:**
    - The model begins by calculating the conditional probability for the first pixel:

      $$p(x_1),$$

      visualized as a grayscale value in the *right image (Probabilistic Reconstruction).*
    - Based on this probability, a hard value for pixel 1 is sampled and fixed:

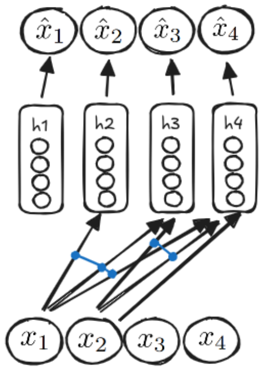
      $$x_1 \sim p(x_1),$$

      displayed in the *left image (Hard Samples).*
    - This hard value for pixel 1 then influences the computation of the conditional probability for the next pixel:

      $$p(x_2 \mid x_1),$$

      again shown in the right image. The process continues sequentially for each pixel, alternating between generating a soft probability in the right image and a hard pixel value in the left image, using the previously determined hard values.

General Discrete Distributions



How to model non-binary discrete random variables $x_i \in \{1, \cdots, K\}$ (e.g., color images)? Solution: let $\hat{x}_i$ parameterize a categorical distribution

$$h_i = \sigma(W_{<i}x_{<i} + c)$$

$$p(x_i \mid x_1, \cdots, x_{i-1}) = \text{Categorical}(p_i^1, \cdots, p_i^K)$$

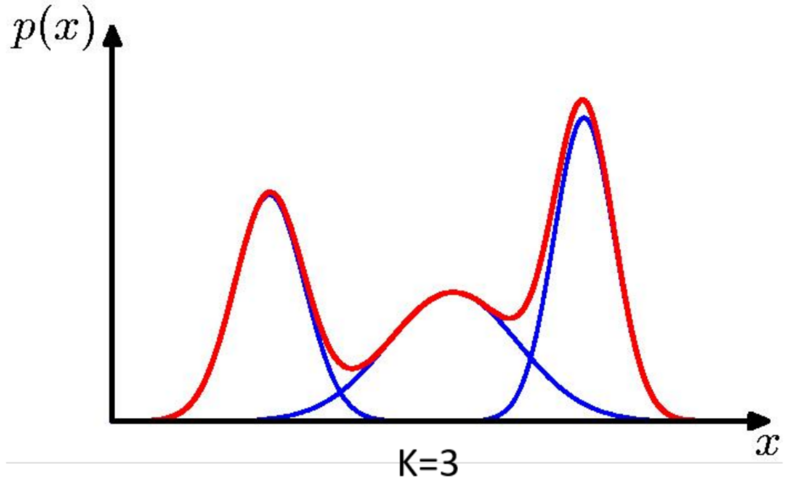
$$\hat{x}_i = (p_i^1, \cdots, p_i^K) = \text{softmax}(x_i h_i + b_i)$$

Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K probabilities (non-negative, sum to 1).

$$\text{softmax}(\mathbf{a}) = \text{softmax}(a^1, \cdots, a^K) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \cdots, \frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$

Real-Valued Neural Autoregressive Density-Estimator - RNADE

How to model continuous random variables $x_i \in \mathbb{R}$ (e.g., speech signals)? Solution: let $\hat{x}_i$ parameterize a continuous distribution (e.g., mixture of K Gaussians). We want our model to learns the parameters of K different Gaussians.



$$p(x_i \mid x_1, \cdots, x_{i-1}) = \frac{1}{K} \sum_{j=1}^{K} \mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$

$$h_i = \sigma(W_{<i} x_{<i} + c)$$

$$\hat{x}_i = (\mu_i^1, \cdots, \mu_i^K, \sigma_i^1, \cdots, \sigma_i^K) = f(h_i)$$

$\hat{x}_i$ defines the mean and variance of each Gaussian (μ_i^j, σ_i^j). Can use exponential $\exp(\cdot)$ to ensure variance is non-negative.

Autoregressive models vs. AutoEncoder

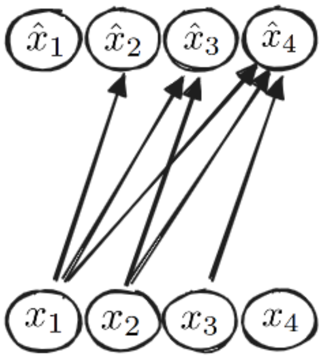


Figure 1: FVSBN

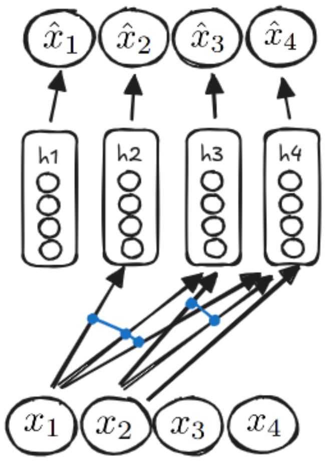


Figure 2: NADE

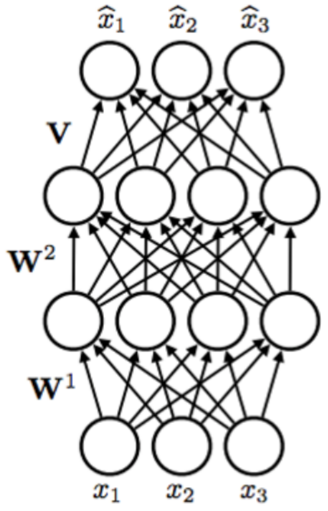


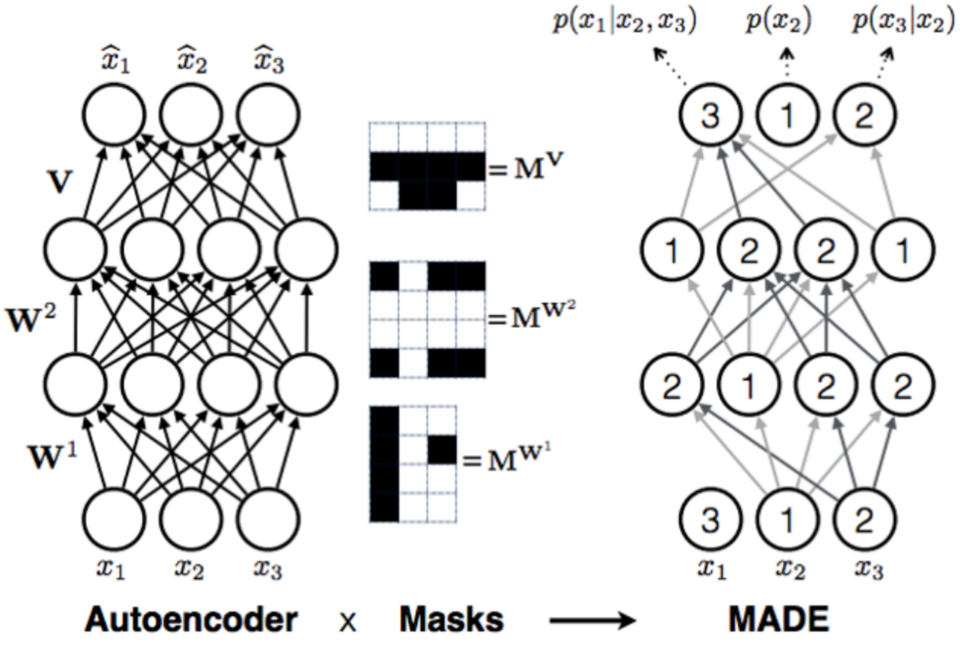
Figure 3: AutoEncoder

On the surface, FVSBN and NADE look similar to an **autoencoder**:

- An **encoder** $e(\cdot)$. For example,
$$e(x) = h = \sigma\left(W^2\left(W^1 x + b^1\right) + b^2\right)$$
- A **decoder** such that $d(e(x)) \approx x$. For example,
$$d(h) = \sigma\left(Vh + c\right).$$
- Binary case:
$$\min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i \left(-x_i \log \hat{x}_i - (1 - x_i) \log(1 - \hat{x}_i)\right)$$
- Continuous case:
$$\min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i \left(x_i - \hat{x}_i\right)^2$$
- e and d are constrained so that we don't learn identity mappings. The hope is that $e(x)$ is a meaningful, compressed representation of x (feature learning).
- A vanilla autoencoder is **not** a generative model: it does not define a distribution over x that we can sample from to generate new data points.

On the surface, FVSBN and NADE look similar to an **autoencoder** but we can not get generative model from an autoencoder as the general autoencoder doesn't have an ordering . So how can we get a generative model from an autoencoder?

- We need to make sure it corresponds to a valid Bayesian Network (DAG structure), i.e., we need an **ordering**. If the ordering is $1, 2, 3$, then:
  - $\hat{x}_1$ cannot depend on any input x. Then at generation time we don't need any input to get started.
  - $\hat{x}_2$ can only depend on x_1.
  - $\cdots$
- **Bonus:** We can use a single neural network (with n outputs) to produce all the parameters. In contrast, NADE requires n passes. This is much more efficient on modern hardware.
- Haven't been done practically.



- **Solution:** Use masks to disallow certain paths (Germain et al., 2015). Suppose the ordering is x_2, x_3, x_1:
  1. The unit producing the parameters for $p(x_2)$ is not allowed to depend on any input. The unit for $p(x_3 \mid x_2)$ can only depend on x_2, and so on.
  2. For each unit, pick a number i in $[1, n-1]$. That unit is only allowed to depend on the first i inputs (according to the chosen ordering).
  3. Add a mask to preserve this invariant: Connect to all units in the previous layer with a smaller or equal assigned number (strictly $<$ in the final layer).
  4. similar loss function as autoregressive models.

Autoregressive Summary

- **Easy to sample from:**
  1. Sample $\bar{x}_0 \sim p(x_0)$.
  2. Sample $\bar{x}_1 \sim p(x_1 \mid x_0 = \bar{x}_0)$.
  3. $\cdots$
- **Easy to compute probability $p(x = \bar{x})$:**

1. Compute $p(x_0 = \bar{x}_0)$.
2. Compute $p(x_1 = \bar{x}_1 \mid x_0 = \bar{x}_0)$.
3. Multiply together (or sum their logarithms).
4. $\cdots$
5. Ideally, can compute all these terms in parallel for fast training.

- **Easy to extend to continuous variables.** For example, can choose Gaussian conditionals:

$$p(x_t \mid x_{<t}) = \mathcal{N}(\mu_\theta(x_{<t}), \Sigma_\theta(x_{<t}))$$

  or a mixture of logistics.

- **No natural way to get features, cluster points, or do unsupervised learning.**
- An alternative way to approach generative problems is using architectures like RNN.

References

[1] Stefano Ermon. *CS236*.