Statistical Models & Computing Methods

Lecture 11: Autoregressive Models and VAE

Cheng Zhang

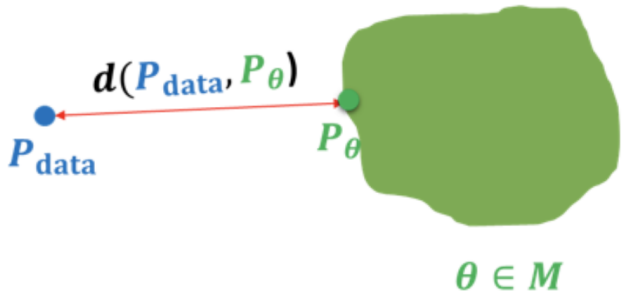
School of Mathematical Sciences, Peking University

December 17, 2020

- Statistical models and inference methods allow us to learn and explain the generative process of the observed data.
- However, real data distributions are often too complicated to be handled by standard statistical models in a satisfactory manner.
- In this lecture, we will introduce some recent techniques that combine deep neural networks and statistical inference methods for expressive generative models.
- The material for this lecture is mainly adapted from Ermon and Grover, 2019.

We are given a training set of examples, e.g., images of dogs

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



Goal: learn a probability distribution $p(x)$ over x such that

- Generation: If we sample $x_{\text{new}} \sim p(x)$, x_{new} should look like a real image.
- Density estimation: $p(x)$ should be high if x looks like a real image, and low otherwise (anomaly detection).
- Unsupervised representation learning: We should be able to learn high level features of these images, e.g., ears, tail, etc.

Two key questions: (1) How to construct $p(x)$? (2) How to learn $p(x)$?

Chain Rule in Probability

We can decompose the joint probability using Chain Rule

$$p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2)p(x_4|x_1, x_2, x_3)$$

Fully general (exponential size, no free lunch)

- Bayes Net: assumes conditional independencies; tabular representations via conditional probability tables (CPT)

$$p(x_1, x_2, x_3, x_4) \approx p(x_1)p(x_2|x_1)p(x_3|x_2)p(x_4|x_3)$$

- Neural Models: assume specific functional form for the conditionals. A sufficiently deep neural net can approximate any function.

$$p(x_1, x_2, x_3, x_4) \approx p(x_1)p(x_2|x_1)p_{\text{Neural}}(x_3|x_1, x_2) \\ p_{\text{Neural}}(x_4|x_1, x_2, x_3)$$

- Input features $X \in \{0,1\}^n$, response variable $Y \in \{0,1\}$.
- For classification, we care about $p(Y|x)$, and assume that

$$p(Y = 1|x; \alpha) = f(x, \alpha)$$

- Logistic regression: let $z(\alpha, x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i x_i$

$$p_{\text{logit}}(Y = 1|x; \alpha) = \sigma(z(\alpha, x)), \quad \sigma(z) = \frac{1}{1 + \exp(-z)}$$

- Neural Nets: let $h(x; A, b)$ be a non-linear transformation of the input features.

$$p_{\text{Neural}}(Y = 1|x; \alpha, A, b) = \sigma(z(\alpha, h))$$

More parameters $\Rightarrow$ more flexibility. Can repeat multiple times to get a multilayer perceptron.

Example: MNIST

Consider a dataset $\mathcal{D}$ of handwritten digits (binarized MNIST)

- Each image has $n = 28 \times 28 = 784$ pixels. Each pixel can either be black (0) or white (1).
- **Goal:** Learn a probability distribution $p(x) = p(x_1, \ldots, x_{784})$ over $x \in \{0, 1\}^{784}$ such that $x \sim p(x)$ looks like a handwritten digit.
- Two step process as mentioned before:
  - Parameterize a family of flexible models $\{p_\theta(x), \theta \in \Theta\}$
  - Search for model parameters θ based on training data $\mathcal{D}$

We start with the first step.

- Pick an order of all random variables, i.e., raster scan order of pixels from top-left (x_1) to bottom-right ($x_{n=784}$)
- Without loss of generality, we can use chain rule for factorization

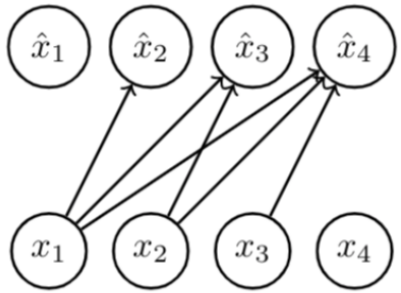
$$p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \cdots p(x_n|x_1, \ldots, x_{n-1})$$

- However, the above parameterization is too heavy to be practical. We can use neural models to simplify it

$$p(x_1, \ldots, x_{784}) = p(x_1; \alpha^1)p_{\text{logit}}(x_2|x_1; \alpha^2) \cdots p_{\text{logit}}(x_n|x_1, \ldots, x_{n-1}; \alpha^n)$$

Remark: This is a modeling assumption. We are using parameterized functions to predict next pixel given all the previous ones (autoregressive models).

Fully Visible Sigmoid Belief Network



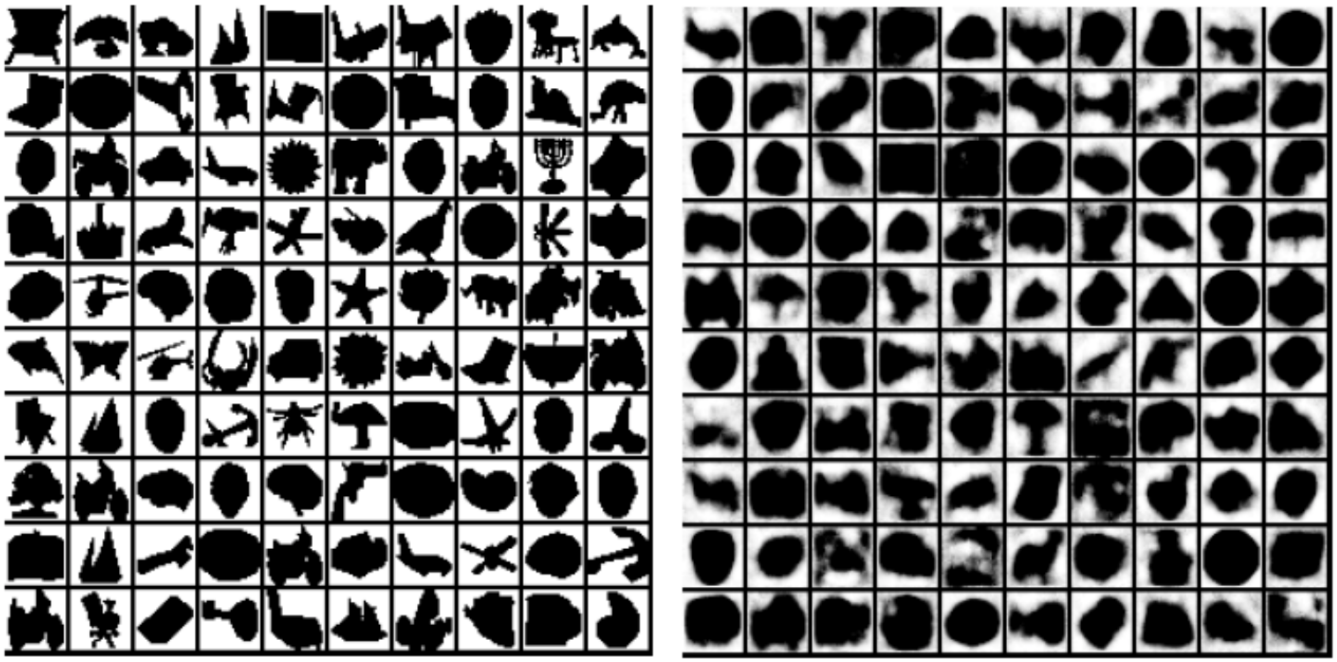
FVSBN

- The conditional distributions $X_i|X_{<i}$ are Bernoulli with parameters

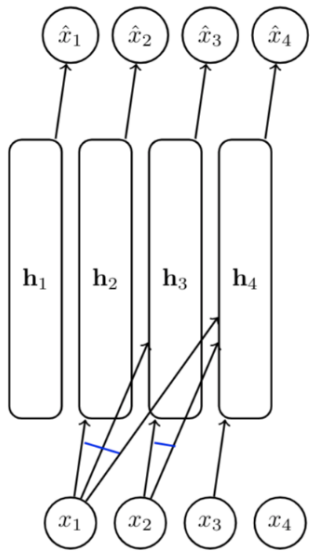
$$p(x_i = 1|x_{<i}; \alpha^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

- We can evaluate $p(x)$ as a product of all the conditionals.
- How to sample from $p(x)$? Sequential sampling!
  - Sample $\bar{x}_1 \sim p(x_1)$
  - Sample $\bar{x}_i \sim p(x_i|x_{<i}), \quad i = 2, \ldots, n$
- How many parameters do we have? $\sum_{i=1}^{n} i = O(n^2)$

Training data on the left (*Caltech 101 Silhouettes*). Samples from the model on the right.



Adapted from Gan et al., 2015

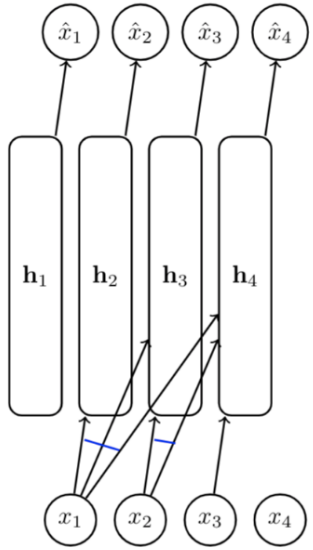


NADE

- Use one layer neural network instead of logistic regression

$$p(x_i = 1|x_{<i}; A_i, c_i, \alpha_i, b_i) = \sigma(\alpha_i^T h_i + b_i), \quad h_i = \sigma(A_i x_{<i} + c_i)$$

- For example: $h_2 = \sigma\left(A_2 x_1 + c_2\right)$, $h_2 = \sigma\left(A_3 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + c_3\right)$



NADE

- Tie weights to reduce the number of parameters

  $$p(x_i = 1|x_{<i}; A_i, c_i, \alpha_i, b_i) = \sigma(\alpha_i^T h_i + b_i), \quad h_i = \sigma(W_{.,<i} x_{<i} + c)$$

- For example: $h_2 = \sigma\left([w_1] x_1 + c\right)$, $h_3 = \sigma\left([w_1\ w_2]\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + c\right)$

- If $h_i \in \mathbb{R}^d$, weights $W \in \mathbb{R}^{d \times n}$, biases $c \in \mathbb{R}^d$, and n logistic regression coefficient $\alpha_i, b_i \in \mathbb{R}^{d+1}$. Probability is evaluated in $O(nd)$.

Samples on the left. Conditional probabilities on the right.

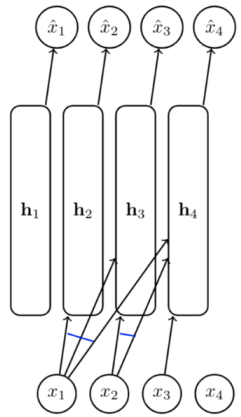
Adapted from Larochelle and Murray, 2011

- What about multi-class discrete random variables $X_i \in \{1, \ldots, K\}$? E.g., pixel intensities varying from 0 to 255
- One solution: use categorical distribution instead of a binary one

$$x_i | x_{<i} \sim \text{Cat}(\pi_i), \quad \pi_i = \text{softmax}(\alpha_i^T h_i + b_i)$$

- Softmax generalizes the sigmoid/logistic function $\sigma(\cdot)$ and transforms a vector of K numbers into a vector of K probabilities

$$\text{softmax}(a) = \left(\frac{\exp(a_1)}{\sum_i \exp(a_i)}, \cdots, \frac{\exp(a_K)}{\sum_i \exp(a_i)} \right)$$



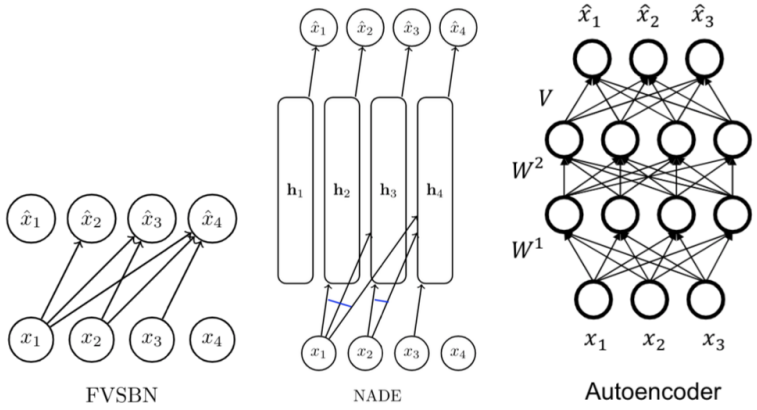
NADE

- How to model continuous random variables $X_i \in \mathbb{R}$? E.g., speech signals.
- Solution: Use a continuous distribution instead! For example, a mixture of K Gaussians

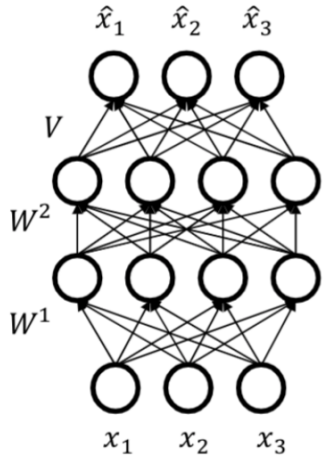
$$p(x_i|x_{<i}) = \frac{1}{K}\sum_{j=1}^{K}\mathcal{N}\left(\mu_i^j, (\sigma_i^j)^2\right)$$

where μ_i^j, σ_i^j can be functions of h_i, e.g., neural networks. Can use exponential function to ensure $\sigma_i^j > 0$.

Autoregressive Models vs Autoencoders

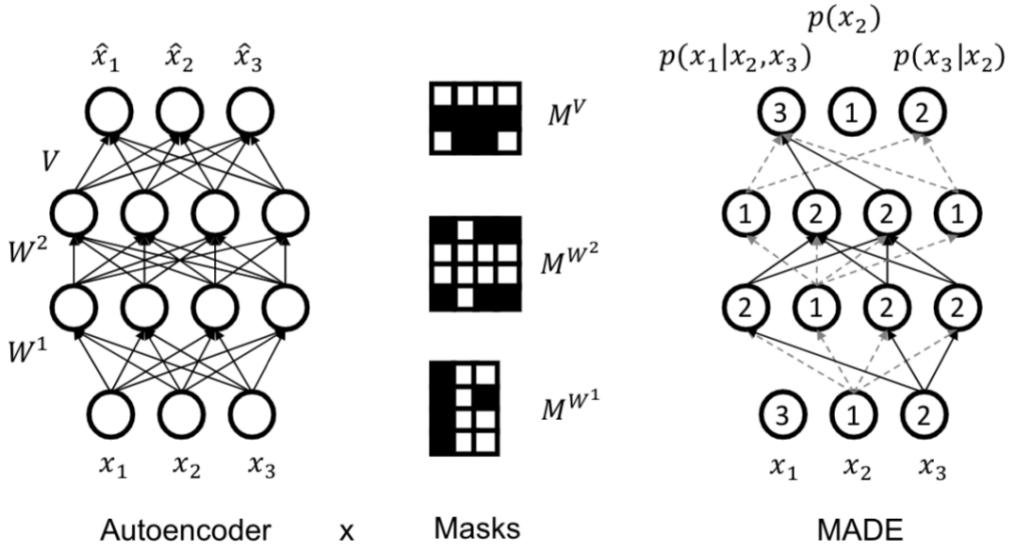


- FVSBN and NADE look similar to an autoencoder.
- Encoder $e(\cdot)$. E.g., $e(x) = \sigma(W^2(W^1 x + b^1) + b^2)$.
- Decoder such that $d(e(x)) \approx x$. E.g., $d(h) = \sigma(Vh + c)$.
- Autoencoder can be trained by minimizing some loss function, e.g., cross-entropy/mean square error.
- In practice, e and d are often constrained so that we don't learn identity mappings. Hopefully, $e(x)$ would be a meaningful, compressed representation of x.
- Note that a vanilla autoencoder is not a generative model



Autoencoder

- Can we get a generative model from an autoencoder?
- We need to make sure it corresponds to an autoregressive architecture, which requires a pre-specified order, say $x_1, x_2, \ldots, x_n$, then $\hat{x}_i$ can only depend on $x_{<i}$, $\forall i$.
- Benefit: we can use a single neural network to produce all the parameters, In contrast, NADE requires n passes. Much more efficient on modern hardware.



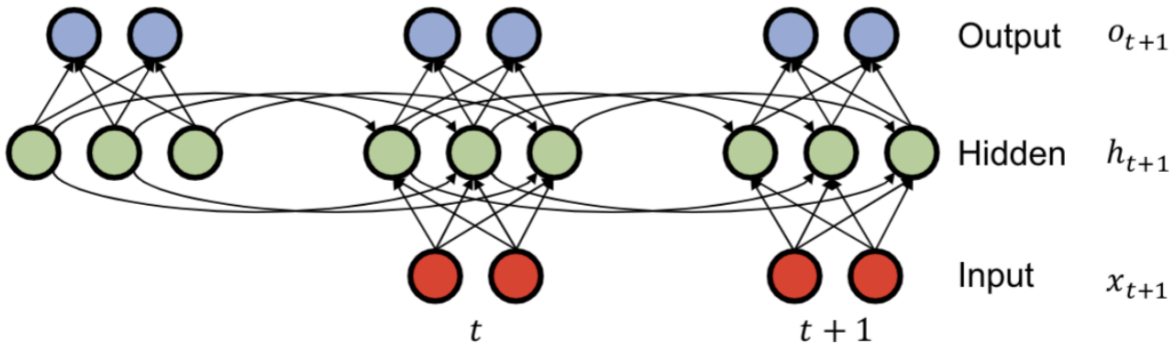
- Challenge: An autoencoder that is autoregressive.
- Solution: use mask to disallow certain paths (Germain et al., 2015).

$$h^\ell(x) = \sigma((W^\ell \odot M^{W^\ell})h^{\ell-1}(x) + b^\ell), \quad \ell = 1, \ldots, L$$

where the masks satisfies

$$M^{W^\ell}_{k',k} = \mathbf{1}_{m^\ell(k') \geq m^{\ell-1}(k)}, \; 1 \leq \ell \leq L, \quad M^V_{d,k} = \mathbf{1}_{d > m^L(k)}.$$

- Challenge: In autoregressive models, the history $x_{1:t-1}$ in conditional distributions $p(x_t|x_{<t};\alpha^t)$ keeps getting longer.
- Idea: keep a summary and recursively update it

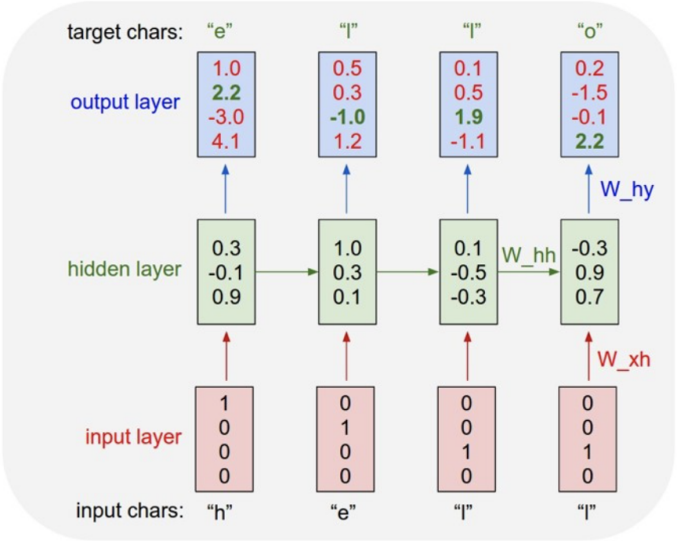


$$\text{update rule: } h_{t+1} = \tanh(W_{hh}h_t + W_{xh}x_{t+1})$$

$$\text{output: } o_{t+1} = W_{hy}h_{t+1}$$

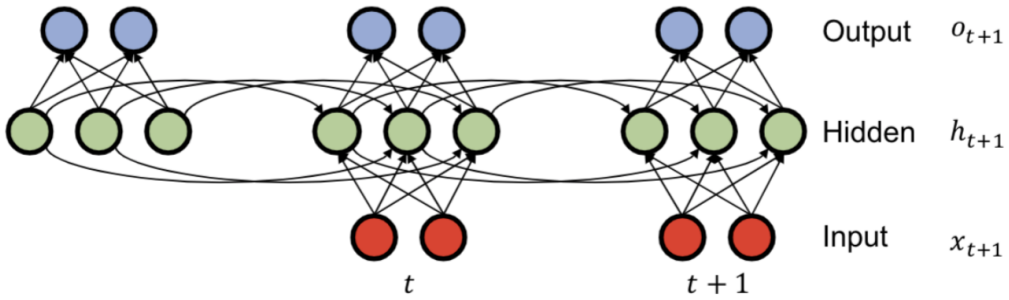
$$\text{initialization: } h_0 = b_0$$

- h_t is a summary of the inputs seen till time t
- o_{t-1} specifies parameters for conditional $p(x_t|x_{<t})$
- Parameterized by b_0, and matrices W_{hh}, W_{xh}, W_{hy}. Constant number of parameters w.r.t. n.



- Suppose $x_i \in \{h, e, l, o\}$. Use one-hot encoding (e.g., h encoded as $[1, 0, 0, 0]$, e encoded as $[0, 1, 0, 0]$.
- Autoregressive modeling: $p(x = hello) = p(x_1 = h)p(x_2 = e|x_1 = h) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
- For example:

$$p(x_2 = e|x_1 = h) = \text{softmax}(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.0)}$$



Pros:

- Can be applied to sequences of arbitrary length.
- Very general: for every computable function, there exists a finite RNN that can compute it.

Cons:

- Still requires an ordering
- Sequential likelihood evaluation (very slow for training)
- Sequential generation (unavoidable in an autoregressive model)
- Can be difficult to train (vanishing/exploding gradients)

Examples: Character RNN

Train 3-layer RNN with 512 hidden nodes on all the works of Shakespeare. Then sample from the model:

> KING LEAR: O, if you were a feeble sight, the courtesy of your law,
> Your sight and several breath, will wear the gods
> With his heads, and my hands are wonder'd at the deeds,
> So drop upon your lordship's head, and your opinion
> Shall be against your honour.

Remark: generation happens character by character. Needs to learn valid words, grammar, punctuation, etc.

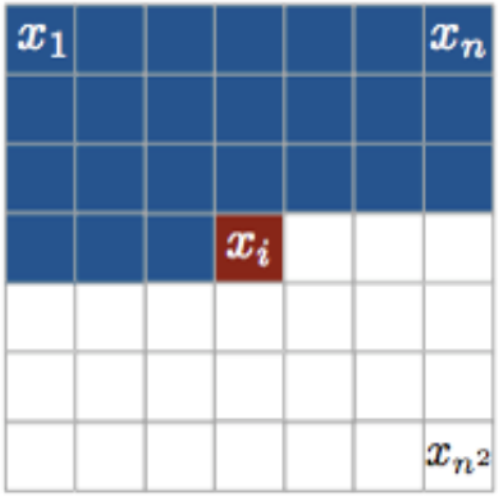
Train on Wikipedia. Then sample from the model:

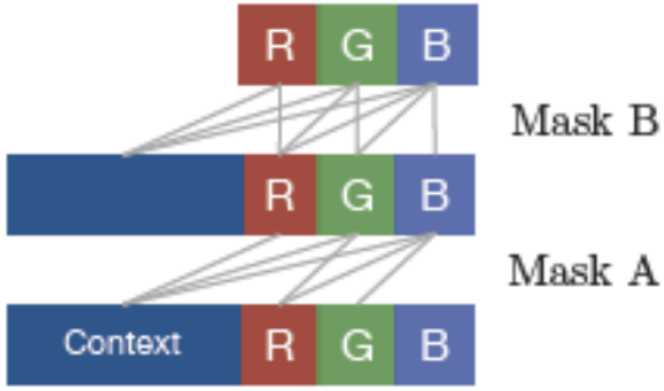
> Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25—21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict.

Remark: correct Markdown syntax. Opening and closing of brackets [[·]]

Train on data set of baby names. Then sample from the model:

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacrie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria Ellia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Velen Daurenge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina



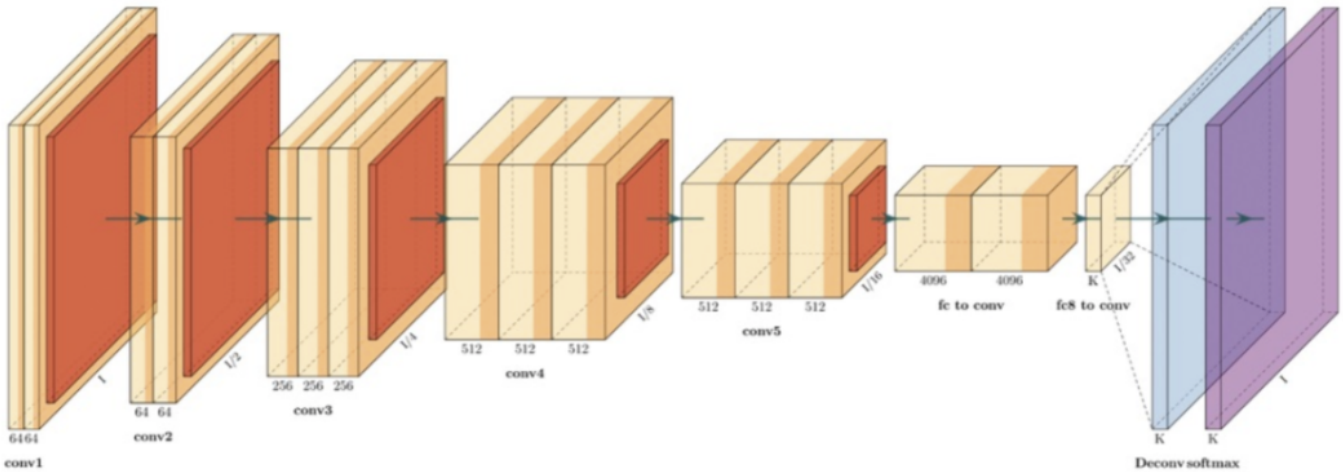


- Model images pixel by pixel using raster scan order
- Each pixel conditional $p(x_t|x_{1:t-1})$ needs to specify 3 colors

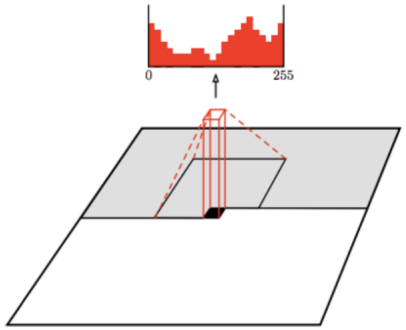
$$p(x_t|x_{1:t-1}) = p(x_t^{\text{red}}|x_{1:t-1})p(x_t^{\text{green}}|x_{1:t-1}, x_t^{\text{red}})p(x_t^{\text{blue}}|x_{1:t-1}, x_t^{\text{red}}, x_t^{\text{green}})$$

- Conditionals modeled using RNN variants. LSTM + Masking (like MADE)

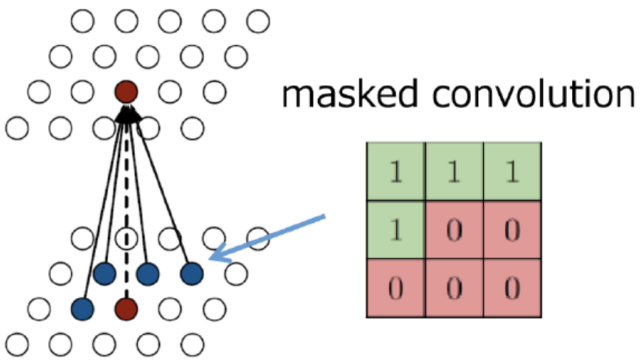
Results on downsampled ImageNet. Very slow: sequential likelihood evaluation.

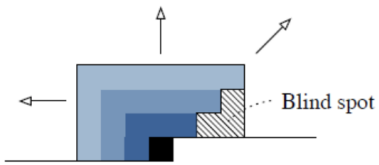


Convolutions are natural for image data and easy to parallelize on modern hardware.

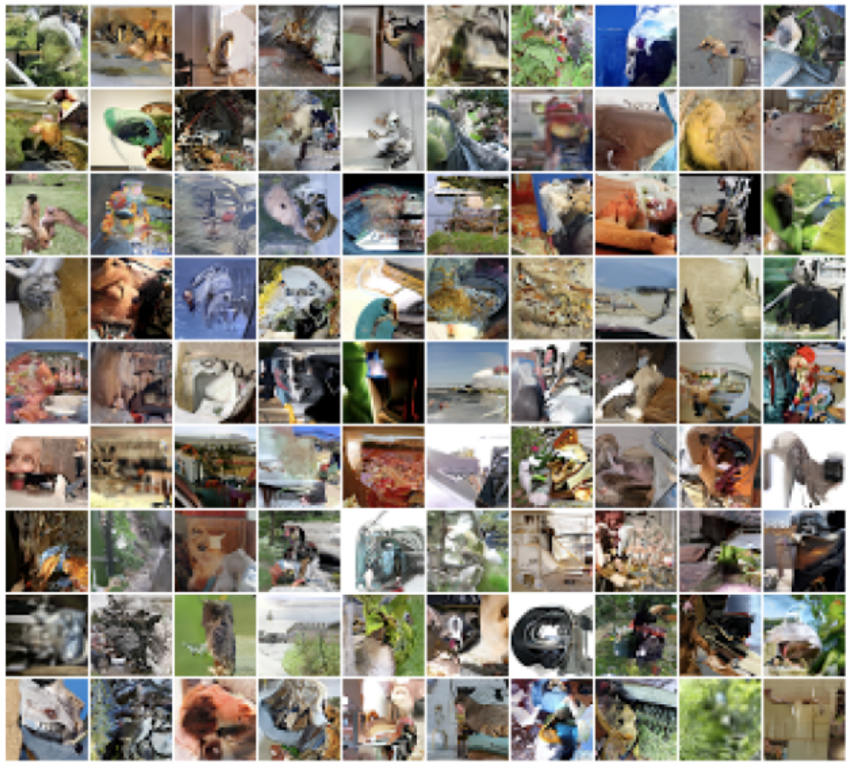


Idea: use convolutional architecture to predict next pixel given context (a neighborhood of pixels)
Challenge: Has to be autoregressive. Masked convolutions preserve raster scan order. Additional masking for colors order.



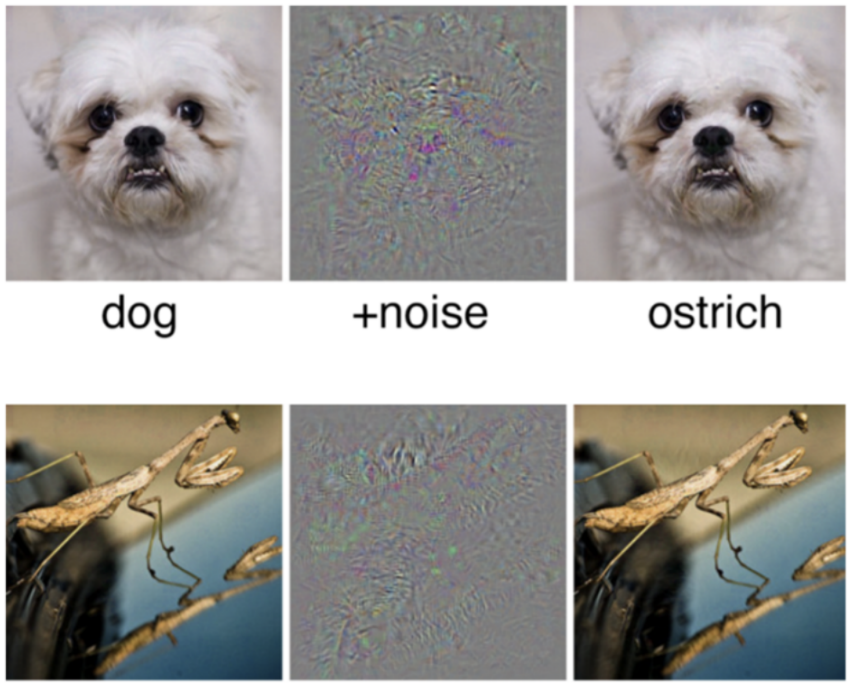


Examples: Pixel CNN

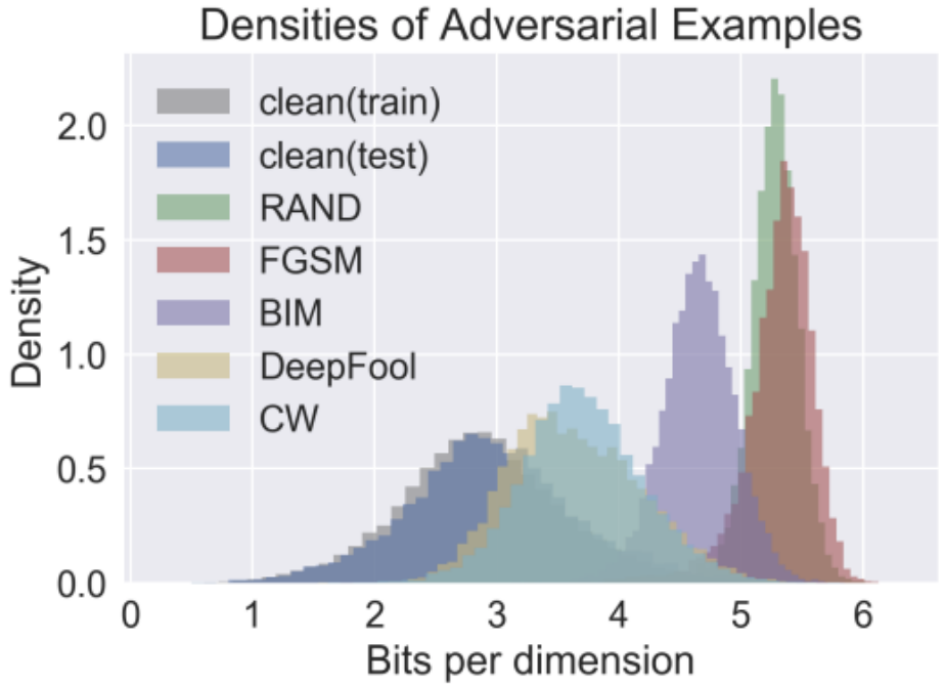


Samples from the PixelCNN model trained on Imagenet (32×32 pixels). Similar performance to PixelRNN, but much faster.

Machine learning methods are vulnerable to adversarial examples

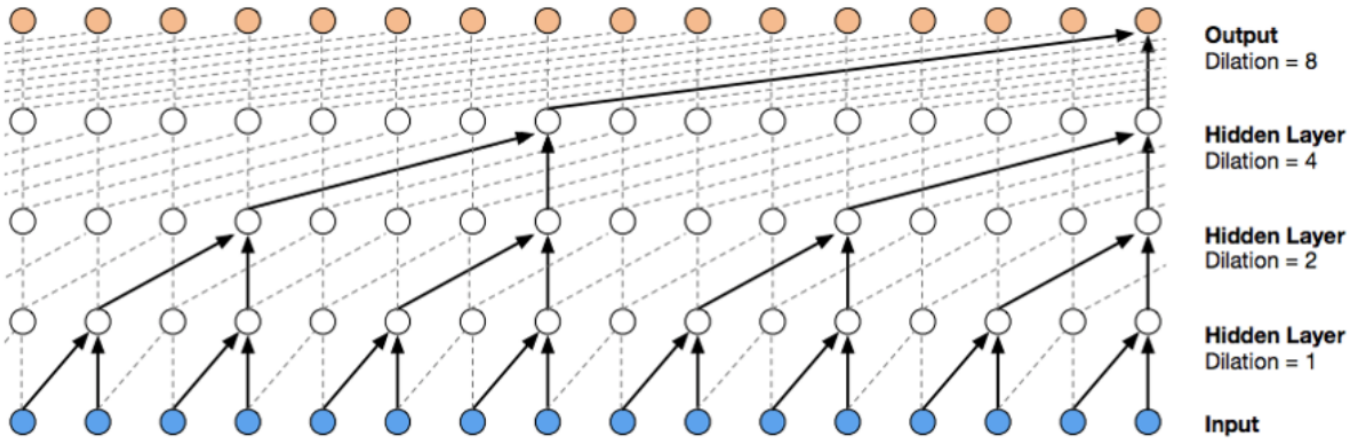


Can we detect them?



- Train a generative model $p(x)$ on clean inputs (PixelCNN)
- Given a new input $\bar{x}$, evaluate $p(\bar{x})$
- Adversarial examples are significantly less likely under $p(x)$

State of the art model for speech:



Dilated convolutions increases the receptive field: kernel only touches the signal at every 2^d entries.

Summary of Autoregressive Models

- Easy to sample from via sequential sampling

$$x_0 \sim p(x_0),\ x_1 \sim p(x_1|x_0),\ \ldots,\ x_n \sim p(x_n|x_{<n})$$

- Easy to compute probability $p(x)$

$$p(x) = p(x_0)p(x_1|x_0)\cdots p(x_n|x_{<n})$$

  Ideally, these conditional distributions can be computed in parallel for fast training

- Easy to extend to continuous variables. For example, $p(x_t|x_{<t}) = \mathcal{N}(\mu_\theta(x_{<t}), \Sigma_\theta(x_{<t})))$ or mixture of logistics
- No natural way to get features, cluster points, do unsupervised learning
- Next, we will discuss learning methods for autoregressive models

- Assume that the domain is governed by some underlying distribution p_{data}.
- We are given a dataset $\mathcal{D}$ of m samples from p_{data}. Each sample is an assignment of values to the variables, e.g., $X_{\text{bank}} = 1, X_{\text{dollar}} = 0, \ldots, Y = 1$ or pixel intensities.
- The standard assumption is that the data instances are independent and identically distributed (IID)
- We are also given a family of models $\mathcal{M}$, and our task is to learn some "good" model $\hat{\mathcal{M}} \in \mathcal{M}$ that defines a distribution $p_{\hat{\mathcal{M}}}$. For example
  - All Bayes nets with a given graph structure, for all possible choices of the CPD tables
  - A FVSBN for all possible choice of the logistic regression parameters. $\mathcal{M} = \{p_\theta; \theta \in \Theta\}$, where θ is the concatenation of all logistic regression coefficients.

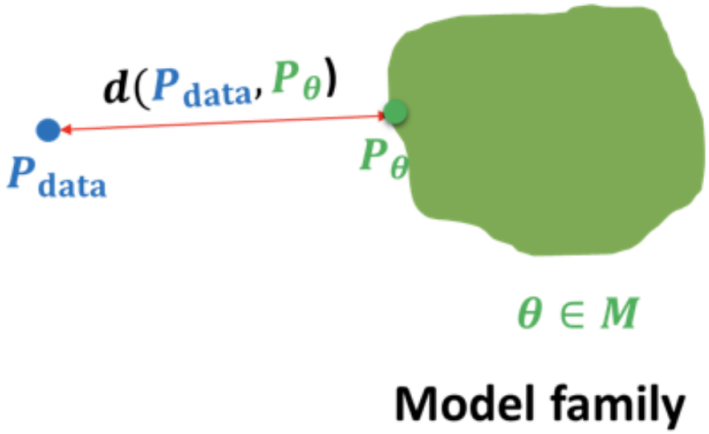
Goal of Learning

- The goal of learning is to return a model $\hat{\mathcal{M}}$ that precisely captures the distribution p_{data} from which our data was sampled
- This is in general not achievable because of
  - limited data only provides a rough approximation of the true underlying distribution
  - can not handle too complicated models due to computational reasons
- Binary MNIST Example: The number of possible states is $2^{784} \approx 10^{236}$. Even 10^7 training examples provide extremely sparse coverage!
- We want to select $\hat{\mathcal{M}}$ to provide the "best" approximation to the underlying distribution p_{data}
- So, what is the "best"?

Learning as Density Estimation

- If our goal is to learn the full distribution so that later we can answer any probabilistic inference query, we can view the learning problem as density estimation.
- Therefore, we want to construct p_θ as "close" as possible to p_{data} (where we assume the dataset $\mathcal{D}$ come from)

$x_i \sim P_{\text{data}}$
$i = 1, 2, \ldots, n$



- How do we measure "closeness"?

- One possibility is to use KL-divergence

$$\begin{aligned}\mathrm{KL}(p_{\text{data}}\|p_\theta) &= \mathbb{E}_{x\sim p_{\text{data}}}\left(\log\frac{p_{\text{data}}(x)}{p_\theta(x)}\right)\\ &= \sum_x p_{\text{data}}(x)\log\frac{p_{\text{data}}(x)}{p_\theta(x)}\end{aligned}$$

- Minimizing KL divergence is equivalent to maximizing the expected log-likelihood

$$\arg\min_{p_\theta}\mathrm{KL}(p_{\text{data}}\|p_\theta) = \arg\max_{p_\theta}\mathbb{E}_{x\sim p_{\text{data}}}\log p_\theta(x)$$

  - Ask that p_θ assign high probability to instances sampled from p_{data} so as to reflect the true distribution
  - Heavily penalize samples x where $p_\theta(x)\approx 0$

- Remark: we do not know how close we are to the data distribution since we do not know p_{data}

- Log-likelihood of an autoregressive model

$$\ell(\theta) = \log p(\theta, \mathcal{D}) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} | \text{pa}(x_i)^{(j)}; \theta_i)$$

- This is an empirical version of $\mathbb{E}_{x \sim p_{\text{data}}} \log p_\theta(x)$. Its negative value can be taken as an Empirical Risk
- Can be trained via gradient ascent

$$\theta^{t+1} = \theta^{(t)} + \alpha_t \nabla_\theta \ell(\theta^t)$$

- When the data size m is large, we can use stochastic gradient ascent

$$\nabla_\theta \ell(\theta) \approx m \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \text{pa}(x_i)^{(j)}; \theta_i), \quad x^{(j)} \sim \mathcal{D}$$

- Empirical risk minimization can easily overfit the data. One extreme case is that the model just memorizes all the training data.
- In practice, people usually care more about generalization: how the model performs on samples that have not yet been seen.
- Thus, we typically restrict the hypothesis space of distributions that we search over, which involves a Bias-Variance trade off
  - Limited hypothesis space might not be able to represent p_{data}, leading to large bias
  - Highly expressive hypothesis space learns too much from the dataset $\mathcal{D}$ (together with random noises), and small perturbations on $\mathcal{D}$ can result in very different estimates, i.e., large variance

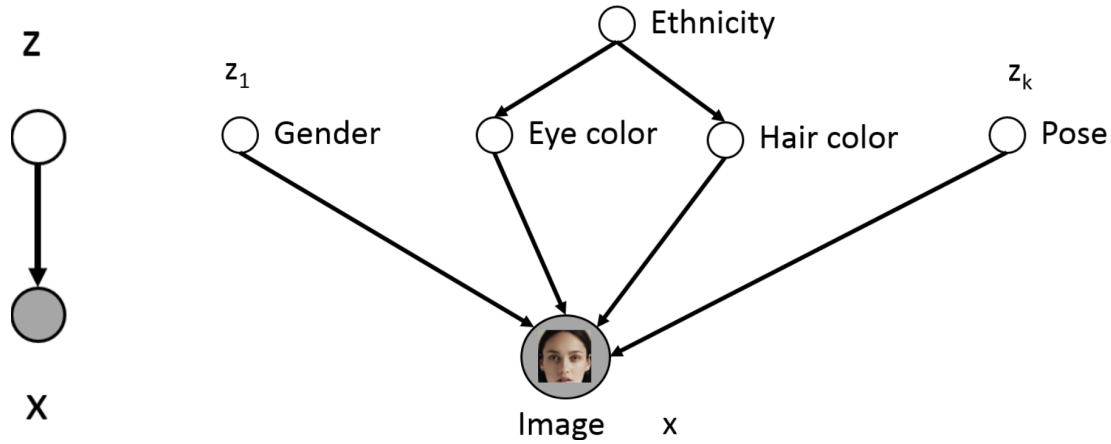
Recap of Autoregressive Models

- Autoregressive models:
  - Chain rule based factorization is fully general
  - Compact representation via conditional independence and /or neural parameterizations
- Pros:
  - Easy to evaluate likelihoods
  - Easy to train
- Cons:
  - Requires an ordering
  - Generation is sequential
  - Cannot learn features in an unsupervised way

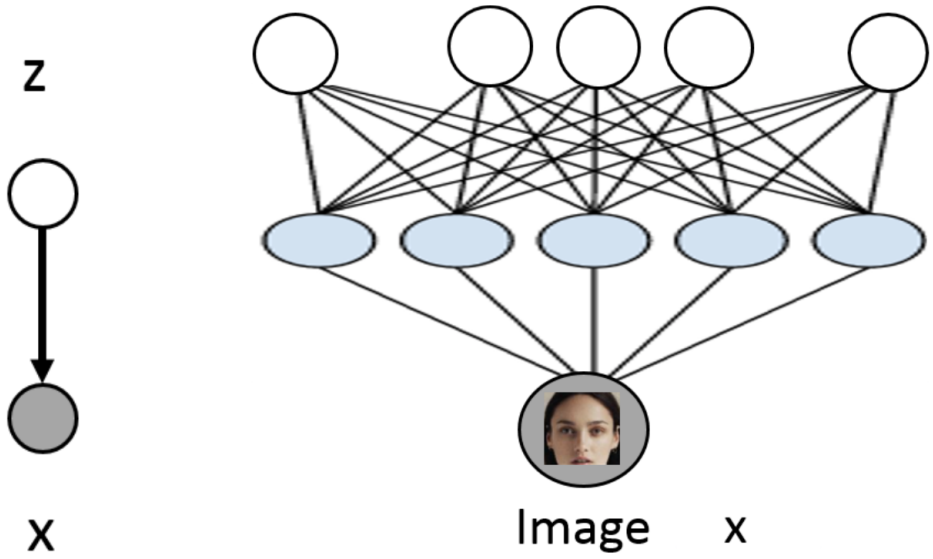
Latent Variable Models: Motivation

- Lots of variability in images x due to gender, eye color, hair color, pose, etc. However, unless images are annotated, these factors of variation are not explicitly available (latent)
- Idea: explicitly model these factors using latent variables z

Latent Variable Models: Motivation

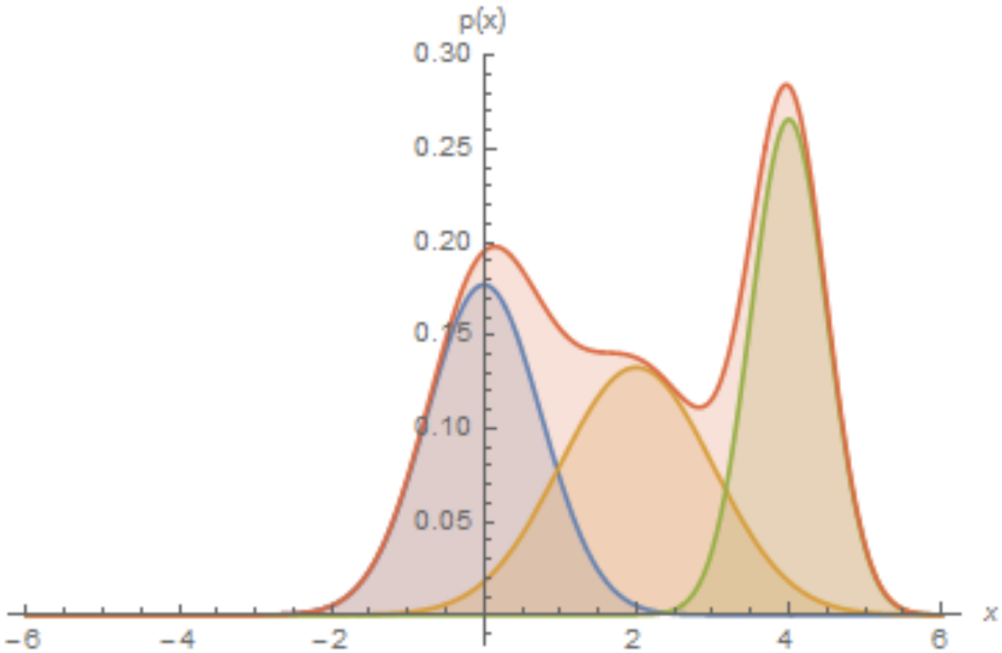


- Only shaded variables x are observed in the data (pixel values)
- Latent variables z correspond to high level features
  - If z chosen properly, $p(x|z)$ could be much simpler than $p(x)$
  - If we had trained this model, then we could identify features via $p(z|x)$, e.g., $p(\text{EyeColor} = \text{Blue}|x)$
- **Challenge**: Very difficult to specify these conditionals by hand

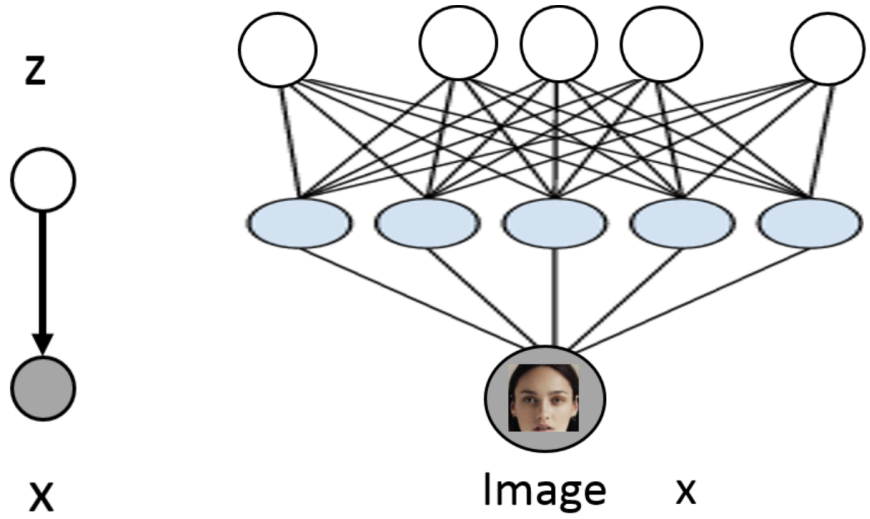


- $z \sim \mathcal{N}(0, I)$
- $p(x|z) = \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$ where $\mu_\theta, \Sigma_\theta$ are neural networks
- Hope that after training, z will correspond to meaningful latent factors of variation (features). Unsupervised representation learning
- As before, features can be computed via $p(z|x)$

Combine simple models into a more complex and expressive one



$$p(x) = \sum_z p(x, z) = \sum_z p(z)p(x|z) = \sum_{k=1}^{K} p(z = k)\mathcal{N}(x; \mu_k, \Sigma_k)$$



A mixture of infinite many Gaussians

- $z \sim \mathcal{N}(0, I)$
- $p(x|z) = \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$ where $\mu_\theta, \Sigma_\theta$ are neural networks
- Even though $p(x|z)$ is simple, the marginal $p(x)$ could be very complex/flexible

$$p_\theta(x) = \int_z p_\theta(x, z)dz = \int_z p_\theta(x|z)p(z)dz$$

- Allow us to define complex models $p(x)$ in terms of simple building blocks $p(x|z)$
- Natural for unsupervised learning tasks (clustering, unsupervised representation learning, etc)
- No free lunch: much more difficult to learn compared to fully observed autoregressive models

$$p_\theta(x) = \mathbb{E}_{z\sim p(z)} p_\theta(x|z), \quad \nabla_\theta p_\theta(x) = \mathbb{E}_{z\sim p(z)} \nabla_\theta p_\theta(x|z)$$

We can use Monte Carlo estimate for the marginal likelihood and its gradient

- Sample $z^{(1)}, \cdots, z^{(k)}$ from the prior $p(z)$
- Approximate expectation with sample average

$$p_\theta(x) \approx \frac{1}{k}\sum_{i=1}^{k} p_\theta(x|z^{(i)}), \quad \nabla_\theta p_\theta(x) \approx \frac{1}{k}\sum_{i=1}^{k} \nabla_\theta p_\theta(x|z^{(i)})$$

Remark: work in theory but not in practice. For most $z \sim p(z)$, $p_\theta(x|z)$ is very low, i.e., mismatch between the prior and posterior. This leads to large variance for the Monte Carlo estimates. We need a clever way to select $z^{(i)}$ to reduce the variance of the estimator.

Second Attempt: Importance Sampling

We can use importance sampling to reduce the variance

$$p_\theta(x) = \int_z p_\theta(x|z)p(z)dz = \int_z q(z)\frac{p_\theta(x,z)}{q(z)}dz = \mathbb{E}_{z\sim q(z)}\frac{p_\theta(x,z)}{q(z)}$$

Similarly, we can use Monte Carlo estimate

- Sample $z^{(1)}, \cdots, z^{(k)}$ from the important distribution $q(z)$
- Approximate expectation with sample average

$$p_\theta(x) \approx \frac{1}{k}\sum_{i=1}^{k}\frac{p_\theta(x,z^{(i)})}{q(z^{(i)})}$$

Remark: What is a good choice for $q(z)$?

- Evidence Lower Bound (ELBO)

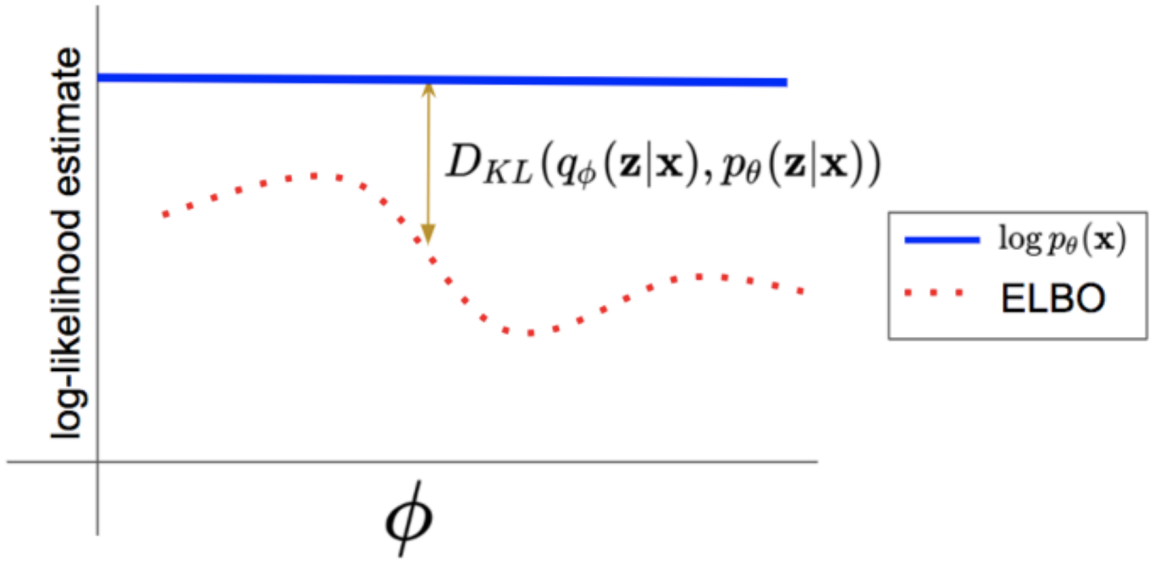
$$\begin{aligned}\log p_\theta(x) &\geq \mathbb{E}_{z\sim q(z)} \log \frac{p_\theta(x,z)}{q(z)} \\ &= \mathbb{E}_{z\sim q(z)} \log p_\theta(x,z) - \mathbb{E}_{z\sim q(z)} \log q(z) \\ &= \mathbb{E}_{z\sim q(z)} \log p_\theta(x,z) + H(q)\end{aligned}$$

- Equality holds when $q(z) = p(z|x;\theta)$

$$\log p_\theta(x) = \mathbb{E}_{z\sim p(z|x;\theta)} \log p_\theta(x,z) + H(p(z|x;\theta))$$

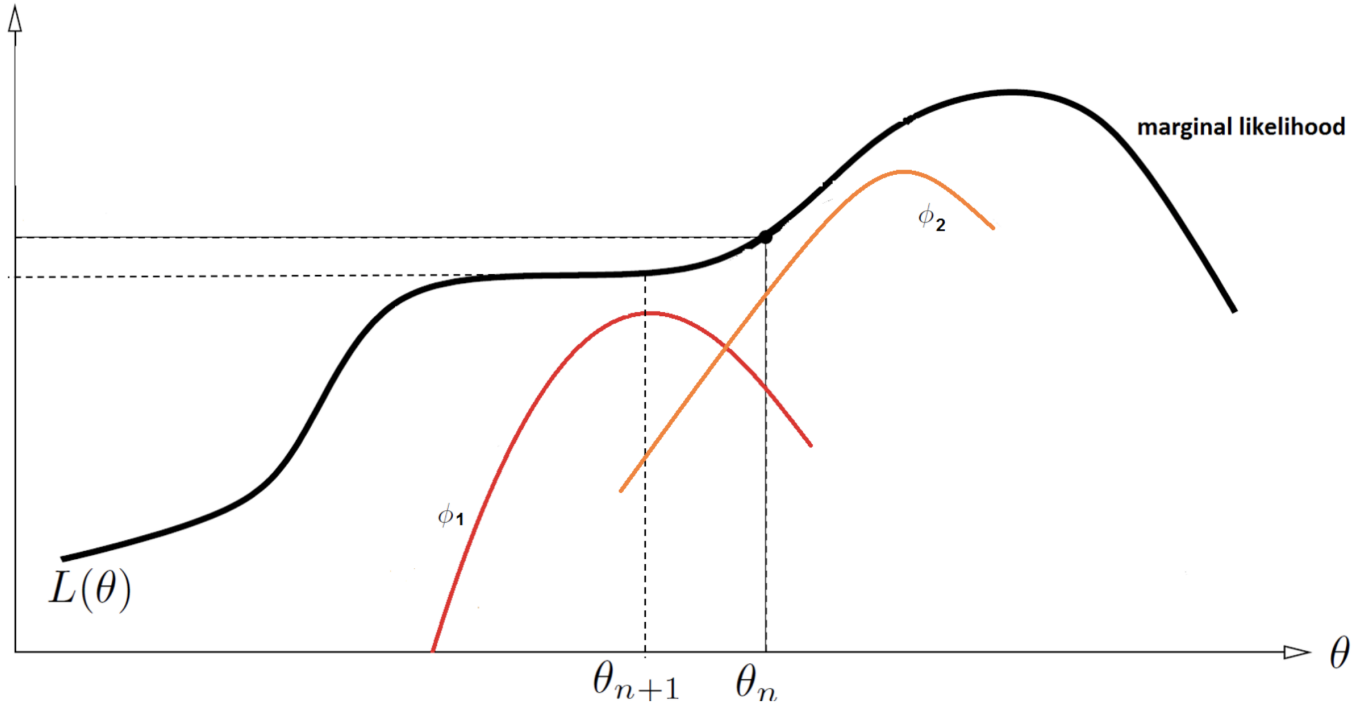
This is the E-step in EM!

- In practice, $p(z|x,\theta)$ is usually intractable. We can find the "best" $q(z)$ by maximizing the ELBO in a parameterized family of $\{q_\phi(z) : \phi \in \Phi\}$



$$\log p_\theta(x) \geq \int_z q_\phi(z|x) \log \frac{p_\theta(x,z)}{q_\phi(z|x)} = \mathcal{L}(x;\theta,\phi)$$
$$= \mathcal{L}(x;\theta,\phi) + \mathrm{KL}(q_\phi(z|x) \| p(z|x;\theta))$$

The better $q_\phi(z|x)$ can approximate the posterior $p(z|x;\theta)$, the closer ELBO will be to the $\log p_\theta(x)$. We then jointly optimize over θ and ϕ to maximize the ELBO over a dataset.



$\mathcal{L}(x;\theta,\phi_1)$ and $\mathcal{L}(x;\theta,\phi_2)$ are both lower bounds, we want to jointly optimize θ and ϕ.

ELBO for The Entire Dataset

- For each data point x, ELBO holds

$$\log p_\theta(x) \geq \int_z q_\phi(z|x) \log p_\theta(x, z) + H(q_\phi(z|x)) = \mathcal{L}(x; \theta, \phi)$$

- Maximum likelihood learning over the entire dataset

$$\ell(\theta; \mathcal{D}) = \sum_{x^i \in \mathcal{D}} \log p_\theta(x^i) \geq \sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i; \theta, \phi^i)$$

- Therefore

$$\max_\theta \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi^1, \cdots, \phi^M} \sum_{i=1}^{M} \mathcal{L}(x^i; \theta, \phi^i)$$

- Note that we use different *variational parameters* ϕ^i for every data point x^i, because the true posterior $p_\theta(z|x^i)$ is different across data points x^i

- Assume $p_\theta(z, x^i)$ is close to $p_{\text{data}}(z, x^i)$. Suppose z captures information such as digit identity (label), style, etc. For simplicity, assume $z \in \{0, 1, \ldots, 9\}$
- Suppose $q_{\phi^i}(z)$ is a probability distribution over the hidden variable z parameterized by $\phi^i = (p_0, \ldots, p_9)$
- If $\phi^i = (0, 0, 0, 1, \ldots, 0)$, is $q_{\phi^i}(z)$ a good approximation of $p_\theta(z|x^1)$(x^1 is the leftmost datapoint)? Yes
- If $\phi^i = (0, 0, 0, 1, \ldots, 0)$, is $q_{\phi^i}(z)$ a good approximation of $p_\theta(z|x^3)$(x^3 is the rightmost datapoint)? No
- For each x^i, need to find a good $\phi^{i,*}$ via optimization, can be expensive

- Optimizing $\sum_{x^i \in \mathcal{D}} \mathcal{L}(x^i; \theta, \phi^i)$ as a function of $\theta, \phi^1, \ldots, \phi^M$ using stochastic gradient ascent

$$L(\mathcal{D}; \theta, \phi^{1:M}) = \sum_{i=1}^{M} \mathbb{E}_{q_{\phi^i}(z^i)} \left(\log p_\theta(x^i, z) - \log q_{\phi^i}(z^i) \right)$$

1. Initialize $\theta, \phi^1, \cdots, \phi^M$
2. Randomly sample a data point x^i from $\mathcal{D}$
3. Optimize $\mathcal{L}(x^i; \theta, \phi^i)$ as a function of ϕ^i, e.g., local gradient update
4. Compute $\nabla_\theta \mathcal{L}(x^i; \theta, \phi^{i,*})$
5. Update θ in the gradient direction. Go to step 2

- How to compute the gradients? Often no close form solution for the expectations. Use Monte Carlo estimates!

$$\mathcal{L}(x;\theta,\phi) = \mathbb{E}_{q_\phi(z)}\left(\log p_\theta(x,z) - \log q_\phi(z)\right)$$

- Similarly as in VI, we assume $q_\phi(z)$ is tractable, i.e., easy to sample from and evaluate
- Suppose $z^1, \ldots, z^k$ are samples from $q_\phi(z)$
- The gradient with respect to θ is easy

$$\begin{aligned}\nabla_\theta \mathcal{L}(x;\theta,\phi) &= \nabla_\theta \mathbb{E}_{q_\phi(z)}\left(\log p_\theta(x,z) - \log q_\phi(z)\right) \\ &= \mathbb{E}_{q_\phi(z)} \nabla_\theta \log p_\theta(x,z) \\ &\approx \frac{1}{k}\sum_{i=1}^{k} \nabla_\theta \log p_\theta(x,z^i)\end{aligned}$$

- The gradient with respect to ϕ is more complicated because the expectation depends on ϕ
- We can use score function estimator (or REINFORCE) with *control variates*. When $q_\phi(z)$ is reparameterizable, we can also use the reparameterization trick.
- If these exists g_ϕ and q_ϵ, s.t. $z = g_\phi(\epsilon), \epsilon \sim q_\epsilon \Rightarrow z \sim q_\phi(z)$

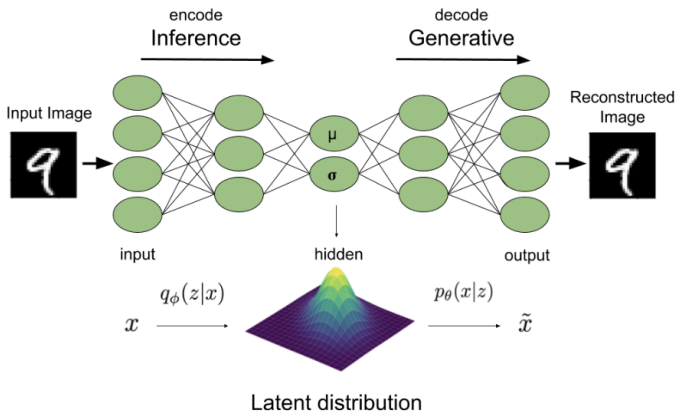
$$\begin{aligned}\nabla_\phi \mathcal{L}(x;\theta,\phi) &= \nabla_\phi \mathbb{E}_{q_\epsilon(\epsilon)} \left(\log p_\theta(x, g_\phi(\epsilon)) - \log q_\phi(g_\phi(\epsilon))\right) \\ &= \mathbb{E}_{q_\epsilon(\epsilon)} \left(\nabla_\phi \log p_\theta(x, g_\phi(\epsilon)) - \nabla_\phi \log q_\phi(g_\phi(\epsilon))\right) \\ &\approx \frac{1}{k} \sum_{i=1}^{k} \left(\nabla_\phi \log p_\theta(x, g_\phi(\epsilon^i)) - \nabla_\phi \log q_\phi(g_\phi(\epsilon^i))\right)\end{aligned}$$

  where $\epsilon^i \sim q_\epsilon(\epsilon), i = 1, \ldots, k$
- Example: $z = \mu + \sigma\epsilon, \epsilon \sim \mathcal{N}(0,1) \Leftrightarrow z \sim \mathcal{N}(\mu, \sigma^2) = q_\phi(z)$

$$\max_{\theta} \ell(\theta; \mathcal{D}) \geq \max_{\theta, \phi^{1:M}} \sum_{i=1}^{M} \mathcal{L}(x^i; \theta, \phi^i)$$

- So far we have used a set of variational parameters ϕ^i for each data point x^i. Unfortunately, this does not scale to large datasets.
- **Amortization**: Learn a single parameteric function f_λ that maps each x to a set of variational parameters. Like doing regression $x^i \mapsto \phi^{i,*}$
  - For example, if $q(z|x^i)$ are Gaussians with different means $\mu^1, \ldots, \mu^m$, we learn a single neural network f_λ mapping x^i to μ^i
- We approximate the posteriors $q(z|x^i)$ using this distribution $q_\lambda(z|x^i)$

- Assume $p_\theta(z, x^i)$ is close to $p_{\text{data}}(z, x^i)$. Suppose z captures information such as digit identity (label), style, etc.
- Suppose $q_{\phi^i}(z)$ is a probability distribution over the hidden variable z parameterized by ϕ^i
- For each x^i, need to find a good $\phi^{i,*}$ via optimization, expensive for large dataset
- Amortized Inference: learn how to map x^i to a good set of parameters ϕ^i via $q(z; f_\lambda(x^i))$. f_λ learns how to solve the optimization problem for you, jointly across all datapoints.
- In the literature, $q(z; f_\lambda(x^i))$ often denoted as $q_\phi(z|x^i)$

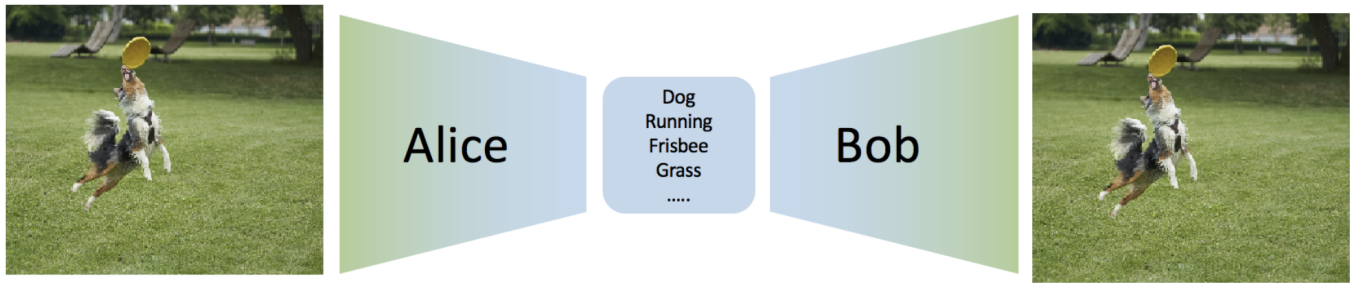


$$\begin{aligned}\mathcal{L}(x;\theta,\phi) &= \mathbb{E}_{q_\phi(z|x)}\left(\log p_\theta(x,z) - \log q_\phi(z|x)\right)\\ &= \mathbb{E}_{q_\phi(z|x)}\left(\log p_\theta(x|z) + \log p(z) - \log q_\phi(z|x)\right)\\ &= \mathbb{E}_{q_\phi(z|x)} \log p(x|z;\theta) - \mathrm{KL}\left(q_\phi(z|x)\|p(z)\right)\end{aligned}$$

Take a data point $x^i \rightarrow$ Map it to $\hat{z}$ by sampling from $q_\phi(z|x^i)$ (encoder) $\rightarrow$ Reconstruct $\hat{x}$ by sampling from $p(x|\hat{z};\theta)$ (decoder)

What does the training objective $\mathcal{L}(x;\theta,\phi)$ do?

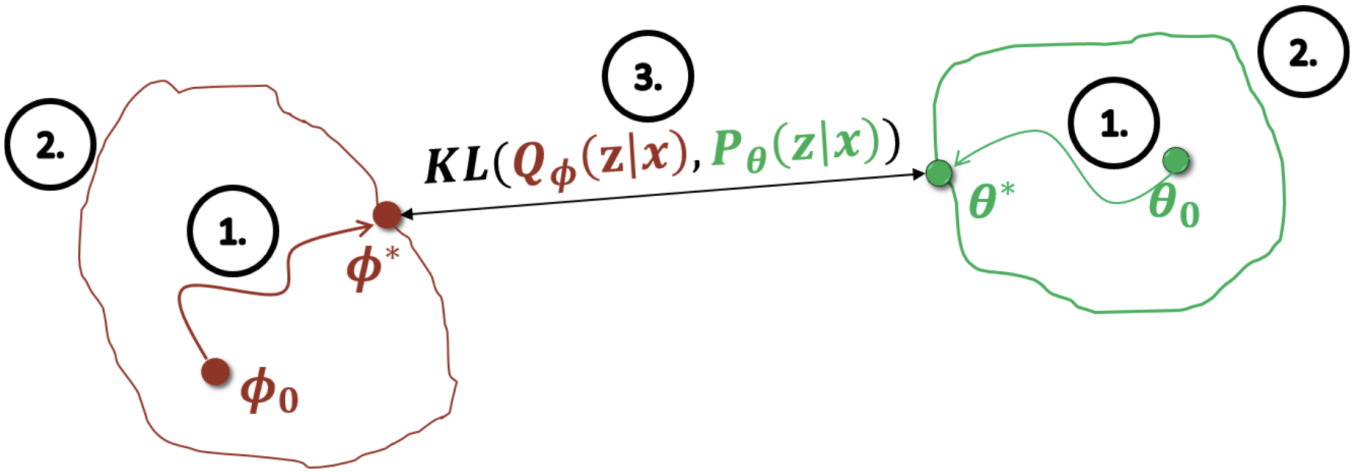
- First term encourages $\hat{x} \approx x^i$ (x^i likely under $p(x|\hat{z};\theta)$)
- Second term encourages $\hat{z}$ to be likely under the prior $p(z)$



- Alice goes on a space mission and needs to send images to Bob. Given an image x^i, she (stochastically) compress it using $\hat{z} \sim q_\phi(z|x^i)$ obtaining a message $\hat{z}$. Alice sends the message $\hat{z}$ to Bob
- Given $\hat{z}$, Bob tries to reconstruct the image using $p_\theta(x|\hat{z})$
  - This scheme works well if $\mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z)$ is large
  - The term $\mathrm{KL}\left(q_\phi(z|x)||p(z)\right)$ forces the distribution over messages to have a specific shape $p(z)$. If Bob knows $p(z)$, he can generate realistic messages $\hat{z} \sim p(z)$ and the corresponding image, as if he had received them from Alice!

Summary on Latent Variable Models

- Combine simple models to get a more flexible one (e.g., mixture of Gaussians)
- Directed model permits ancestral sampling (efficient generation): $z \sim p(z), \quad x \sim p_\theta(x|z)$
- However, log-likelihood is generally intractable, hence learning is difficult (compared to autoregressive models)
- Joint learning of a model (θ) and an amortized inference component ϕ to achieve tractability via ELBO optimization
- Latent representations for any x can be inferred via $q_\phi(z|x)$



Improving variational learning via

- Better optimization techniques
- More expressive approximating families
- Alternate loss functions

Amortization (Gershman & Goodman, 2015; Kingma; Rezende;..)

- Scalability: efficient learning and inference on massive datasets
- Regularization effect: due to joint training, it also implicitly regularizes the model θ (Shu et al., 2018)

Augmenting variational posteriors

- Monte Carlo methods: Importance sampling (Burda et al., 2015), MCMC (Salimans et al., 2015, Hoffman, 2017, Levy et al., 2018), Sequential Monte Carlo (Maddison et al., 2017, Le et al., 2018, Naesseth et al., 2018), Rejection sampling (Grover et al., 2018)
- Normalizing flwos (Rezende & Mohammed, 2015, Kingma et al., 2016)

Tighter ELBO does not imply:

- Better samples: sample quality and likelihoods are uncorrelated (Theis et al., 2016)
- Informative latent codes: powerful decoders can ignore latent codes due to tradeoff in minimizing reconstruction error vs KL prior penalty (Bowman et al., 2015, Chen et al., 2016, Zhao et al., 2017, Alemi et al., 2018)

Alternatives to the KL divergence:

- Renyi's alpha-divergences (Li & Turner, 2016)
- Integral probability metrics such as maximum mean discrepancy, Wasserstein distance (Dziugaite et al., 2015, Zhao et al., 2017, Tolstikhin et al., 2018)

References

- Zhe Gan, Ricardo Henao, David E. Carlson, and Lawrence Carin. 2015. Learning deep sigmoid belief networks with data augmentation. In Proceedings of the AISTATS.
- Larochelle, H. and Murray, I. The neural autoregressive distribution estimator. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, pp. 29–37, 2011.
- Uria, B., Murray, I., and Larochelle, H. Rnade: The realvalued neural autoregressive density-estimator. In Advances in Neural Information Processing Systems, pp. 2175–2183, 2013.
- M. Germain, K. Gregor, I. Murray, H. Larochelle, "MADE: Masked autoencoder for distribution estimation" in 32nd International Conference on Machine Learning, ICML 2015, vol. 2, pp. 881–889

- Aäron Van Den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In Proceedings of the 33rd International Conference on International Conference on Machine Learning, pages 1747–1756. JMLR. org, 2016.
- A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. 2016. Conditional image generation with PixelCNN decoders. In Advances in Neural Information Processing Systems. 4790–4798.
- Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In International Conference on Learning Representations, 2018.

References

- A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A generative model for raw audio," arXiv preprint arXiv:1609.03499, 2016.
- Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.