Variational Autoencoders

Vahid Tarokh

ECE 685D, Fall 2025

Generative Models-Review

- We discussed about two learning paradigms : generative and discriminative
- In generative paradigm the learner tries to learn a joint distribution over all the variables.
- A generative model simulates how the data is generated in the real world.
- Why do we need generative model:
  - Encoding the laws of physics and other constraints into the generative process
  - Expresses causal relations of the world, so we can generalize better to new situations than mere correlations.
  - Building useful abstractions of the world
  - Useful for unsupervised representation learning

VAE-Overview

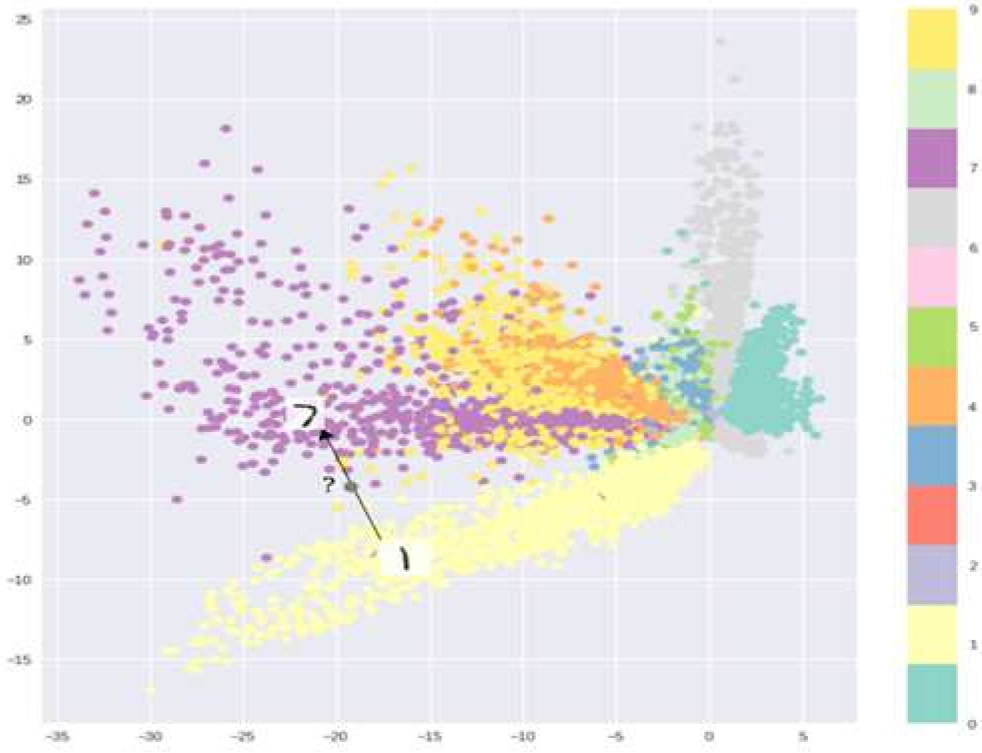
- Variational Autoencoders:
  - Allow us to design complex generative models of data and fit them to large datasets.
  - Can generate fake data/images (e.g., fake celebrity faces) and fake high-resolution digital artwork.

VAE-Overview

- These models also yield state-of-the-art machine learning results in image generation.
- VAEs bridges graphical models and the deep learning
- The general construction is based on generative probability models and does not really need to be implemented using Neural Networks.
- Variational Autoencoders (VAEs) based on Deep Learning were proposed in 2013.
- But since this is a course on Neural Networks, we will mostly implement them using Neural Networks.

Ideal Generative Models

- The fundamental problem with autoencoders:
  - ➢latent encoded set where their encoded vectors lie, may not be contiguous, or allow easy interpolation.



Optimizing purely for reconstruction loss

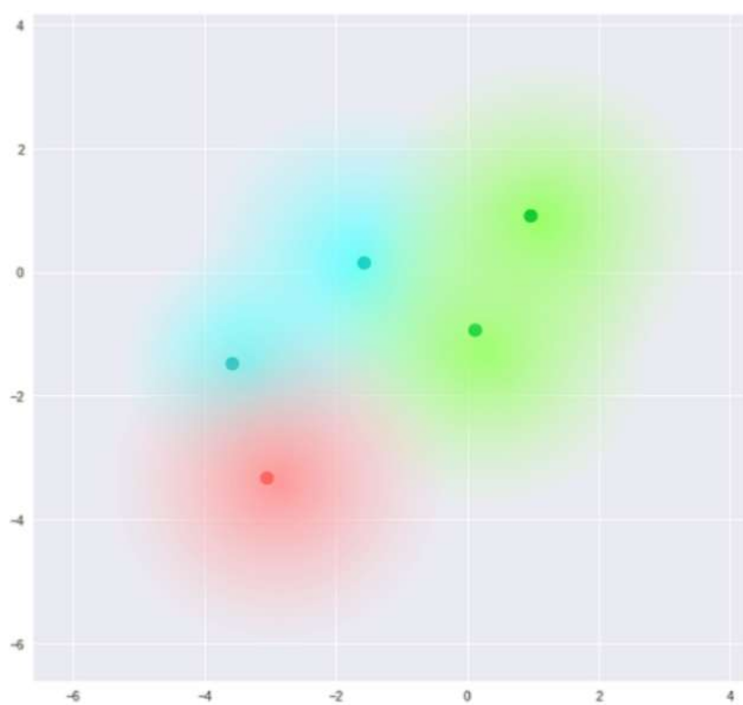
- Training an auto-encoder on the MNIST dataset
- Visualizing the encodings from a 2D latent space
- The formation of distinct clusters.

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

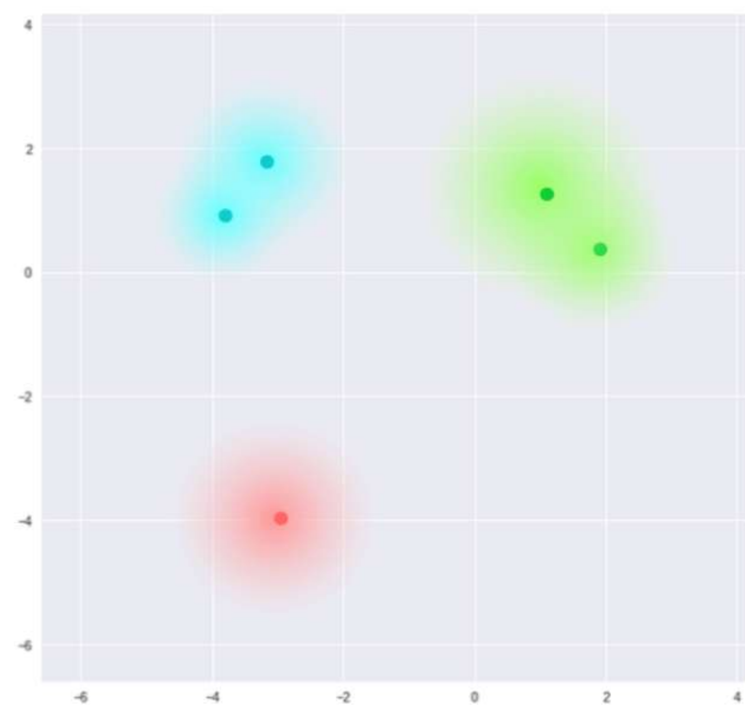
Optimizing only based on the maximum likelihood (reconstruction loss)

Ideal Generative Models

- What we may ideally want: the encodings are close to being "contiguous" (intuitively speaking) while still being distinct,
- This allows smooth interpolation and enables construction of *new* samples.



Desired encoding

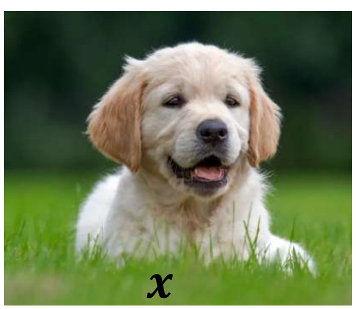


Undesired encoding

Parameterization of conditional distributions with Neural Networks

- Probabilistic models based on neural networks are computationally scalable
  - Using stochastic gradient-based optimization
  - Scaling to large models and large datasets
- An example:
  - Parameterizing a categorical distribution (cat), $P_{\boldsymbol{\theta}}(y|\boldsymbol{x})$ using neural networks (NN)
    - y : class label, conditioned on
    - $\boldsymbol{x}$: input image
    - θ : the parameter vector of the distribution

$$\theta = NN(\boldsymbol{x})$$
$$p_\theta(y|\boldsymbol{x})$$



$$\xrightarrow{NN(\boldsymbol{x})}$$

$$p(y = dog|\boldsymbol{x}) = 0.979$$
$$p(y = cat|\boldsymbol{x}) = 0.02$$
$$p(y = deer|\boldsymbol{x}) = 0.001$$
$$\vdots$$

Deep Latent-Variable Model (DLVM)

- DLVM denote a latent variable model $p_\theta(\boldsymbol{x}, \boldsymbol{z})$ where distributions are parameterized by neural networks.
- Advantage of DLVM:
  - Even having simple factors (prior or conditional distribution) such as conditional Gaussian, the **marginal likelihood**, $p_\theta(\boldsymbol{x})$ can be arbitrarily complex

DLVM is attractive for approximating complicated underlying distributions

Example:
DLVM for multivariate Bernoulli data

- An example from (Kingma and Welling, 2014) for binary data $\boldsymbol{x}$ and with a spherical Gaussian latent space,

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{z}; 0, \mathbf{I})$$

Latent variable distribution

$$\mathbf{p} = \text{DecoderNeuralNet}_{\boldsymbol{\theta}}(\mathbf{z})$$

Parametrized Bernoulli with NN

$$\log p(\mathbf{x}|\mathbf{z}) = \sum_{j=1}^{D} \log p(x_j|\mathbf{z}) = \sum_{j=1}^{D} \log \text{Bernoulli}(x_j; p_j)$$

$$= \sum_{j=1}^{D} x_j \log p_j + (1 - x_j) \log(1 - p_j)$$

D denotes the dimension of x and $0 \leq p_j$'s ≤ 1.

Disadvantage of DLVM

- Intractability of Marginal likelihood (evidence)
$$\int p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})\boldsymbol{dz}$$
  - ➢So, we cannot differentiate it w.r.t. its parameters and optimize it
- This means the intractability of the posterior, $p(\boldsymbol{z}|\boldsymbol{x})$
- Solution: Estimate posterior distribution using Variational Inference (variational Bayes)

VAE Framework

- Using Variational Autoencoders (VAEs), we can efficiently optimize DLVMs jointly with a corresponding inference model using SGD.
- In the probability model framework, a variational auto-encoder (VAE) is assumed to be modeled by a specific probability model of data $\boldsymbol{x}$ and latent variables $\boldsymbol{z}$.
- We can write the joint probability of the model as $p_\theta(\boldsymbol{x}, \boldsymbol{z}) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\, p(\boldsymbol{z})$
- The generative process can be written as follows:
  - For each data-point i:
    - Draw latent variables $\boldsymbol{z_i} \sim p(\boldsymbol{z})$
    - Draw data-point $\boldsymbol{x_i} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$

The Goal

- To get around of directly computing the intractable posterior, one can introduce parametric *inference model* $q_\lambda(z|x)$ to approximate the true posterior:

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \approx p_\theta(\boldsymbol{z}|\boldsymbol{x})$$

  - ➢ This model is also called an *encoder* or *recognition model*.
  - ➢ The parameters of this inference model, λ also called the *variational parameters*.
  - ➢ Using the above approximation, we can optimize the marginal likelihood.

- For instance if $q_\lambda(z|x)$ are Gaussian, the λ is the mean vector and co-variance matrix.

VAE Framework

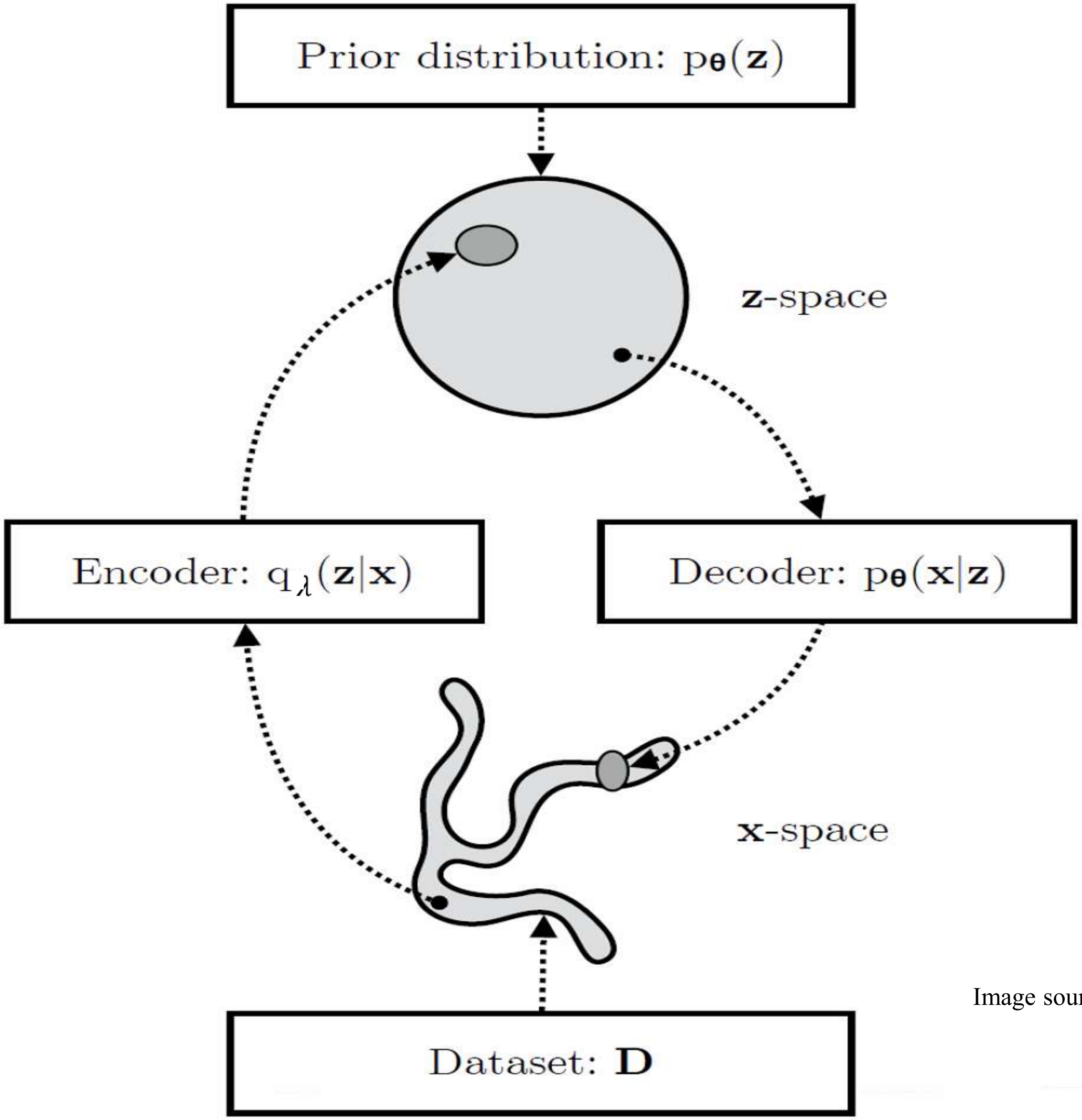


Image source: Kingma and Welling, 2019

Variational Parameters

- Parameterizing the distribution of approximate posterior, $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ by a deep neural network encoder:
  - The variational parameters include the weights and biases of the neural network:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = EncoderNN_\phi(\boldsymbol{x})$$
$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, diag(\boldsymbol{\sigma}))$$

  - Using a single encoder neural network to perform posterior inference over all datapoints (shared parameters.
    - This is called *amortized variational inference*
  - In traditional approximate inference models, variational parameters are not shared

Question: How to *choose* λ?

- The optimization objective is reverse KL:
  - The *evidence lower bound*, abbreviated as ELBO (*variational lower bound*)

$$D(q_\lambda(\boldsymbol{z}|\boldsymbol{x})||p_\theta(z|x))$$
$$= E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ] + \log\left(\mathrm{p}_\theta(\boldsymbol{x})\right)$$
$$-\ E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ p_\theta\ (\boldsymbol{x},\boldsymbol{z})\ ]$$

- Hence:

$$log\left(p(\boldsymbol{x})\right) = D(q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ||p_\theta(\ \boldsymbol{z}\mid\boldsymbol{x}\ )) + E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ p_\theta\ (\boldsymbol{x},\boldsymbol{z})\ ] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ]$$

Evidence Lower Bound (ELBO)

- This means that minimizing the KL-distance is equivalent to maximizing

$$ELBO = E_{q_\lambda(z|x)}[\log p_\theta(x,z)] - E_{q_\lambda(z|x)}[\log q_\lambda(z|x)]$$

- Assume that no two data points share their latent variables with each other then ELBO decomposes into the sum of

$$ELBO_i = E_{q_\lambda(z|x_i)}[\log p_\theta(x,z)] - E_{q_\lambda(z|x_i)}[\log q_\lambda(z|x_i)]$$

The Main Message

- This is equal to

$$ELBO_i = \underbrace{E_{q_\lambda(z|x_i)}[\log p_\theta(x_i \mid z)]}_{\text{Likelihood term (reconstruction part)}} - \underbrace{D(q_\lambda(z|x_i) \,||\, p(z))}_{\text{Closeness of encoding to p(z) (typically Gaussian)}}$$

- Typically $p(z)$ is selected to be standard normal distribution. Then if we have a parametric model class for $p_\theta(x_i \mid z)$, we can maximize the objective function $\sum_i ELBO_i$ over parameters θ and λ using Stochastic Gradient Ascent.

- This process is true regardless of if model classes $p_\theta(x_i|z)$ and $q_\lambda(z|x)$ are given by deep Neural Networks or not!

Computational Flow In a Variational Autoencoder

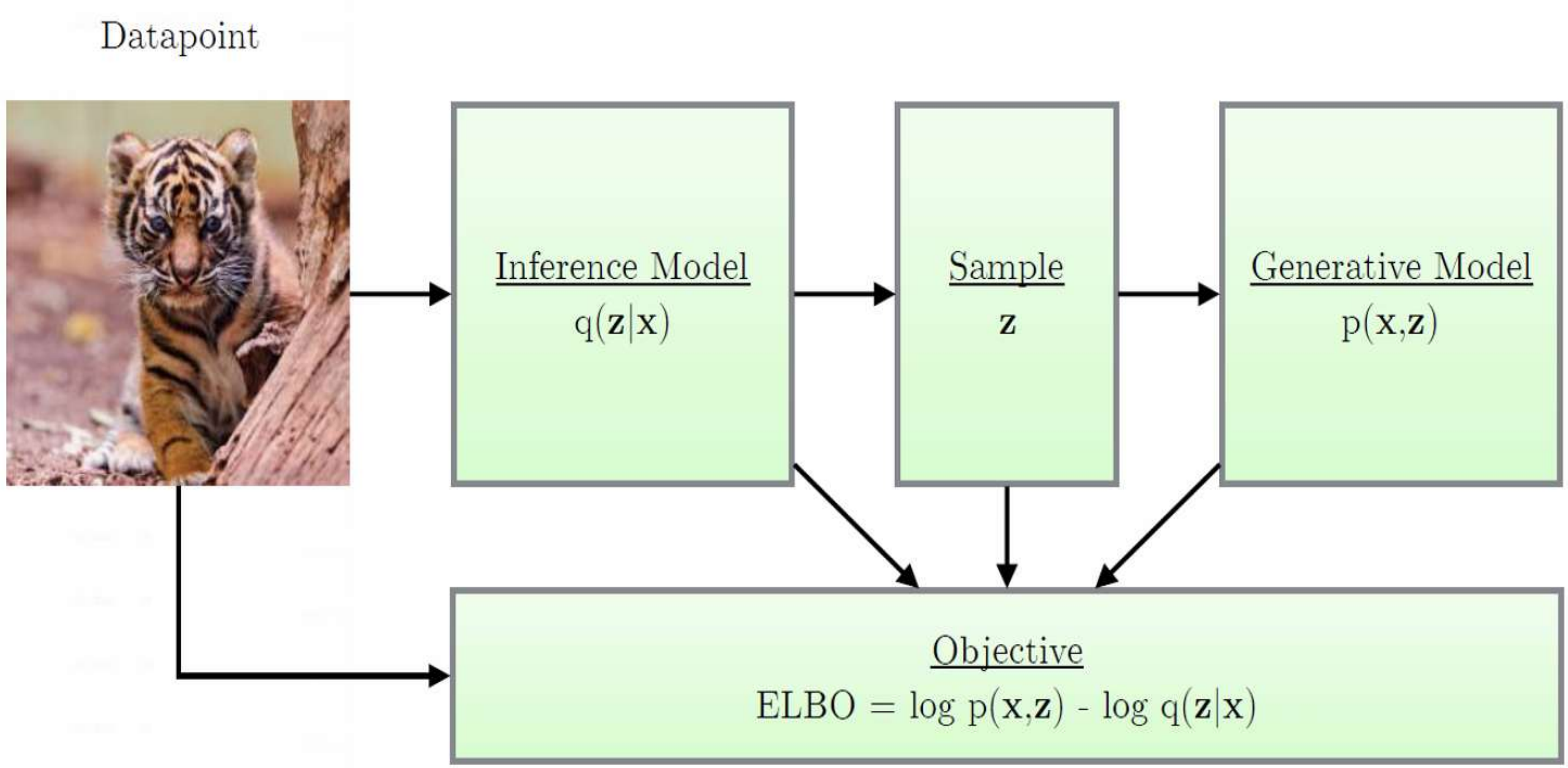


Image source: Kingma and Welling, 2019

Generation of New Data

- Let us introduce some names:
  - $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ is referred to as the encoder
  - $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is referred to as the decoder

- If we have an encoder and decoder designed (optimized as before) and have calculated all the optimized parameters, then to make a new value of x (generate data)

  - Generate z according to $p(\boldsymbol{z})$ (Typically standard Gaussian)
  - Generate x according to $p_\theta(\boldsymbol{x}|\mathbf{z})$

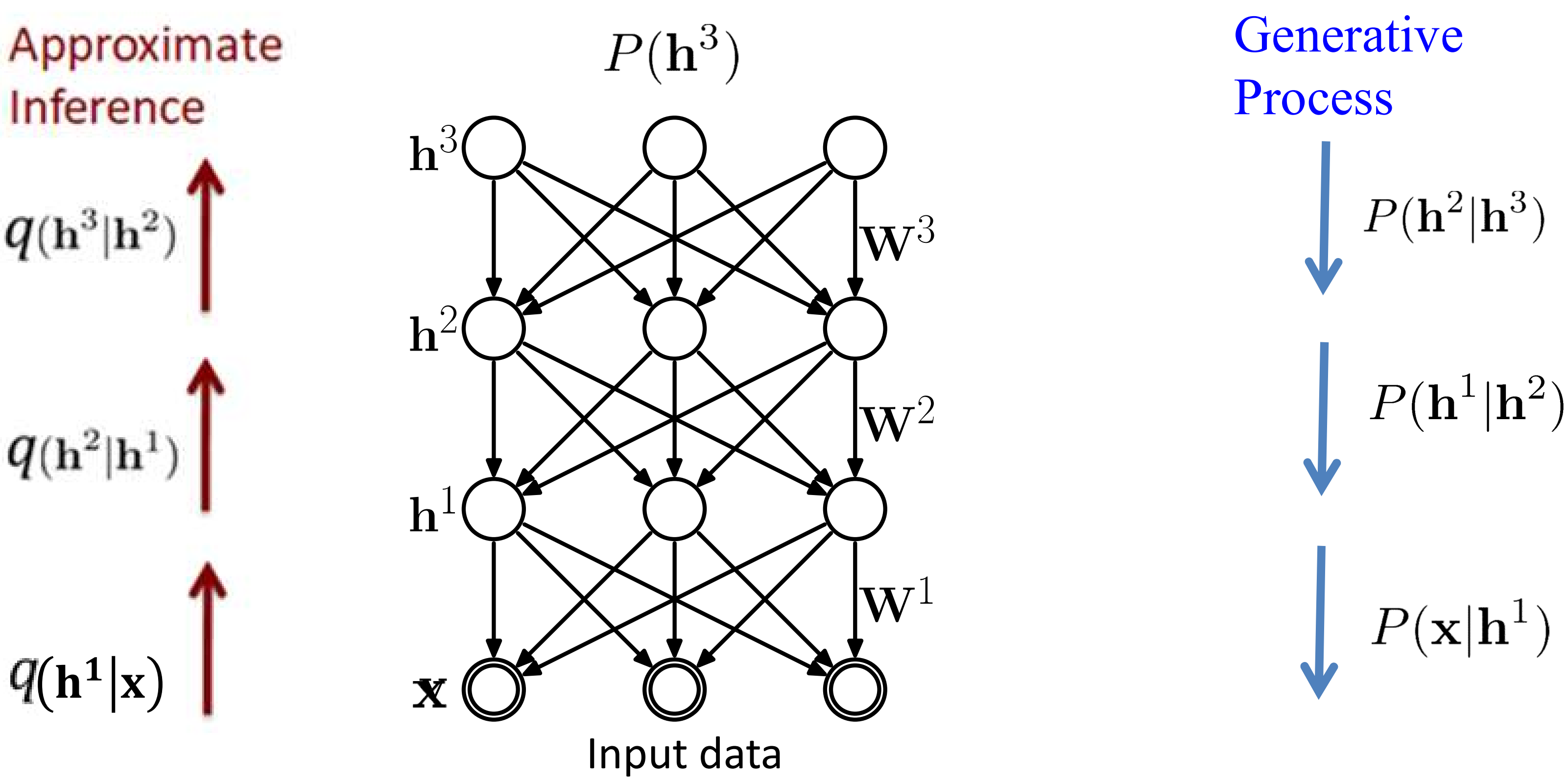
Connecting to Neural Networks

- **But this course is about Neural Networks**.
- Idea: Use DNNs somehow for encoders and Decoders.
  - Encoding Neural Network: Up on observing x, the neural network outputs parameters λ
  - Decoding Neural Network: Up on observing z, the neural network outputs parameters θ
- Equivalently we will have to learn the weights λ and θ of encoding and decoding DNNs using SGD to minimize $-ELBO$.
- We can present this in a fancier way if we wish.

VAEs from Multilayer DNNs

Deep VAE

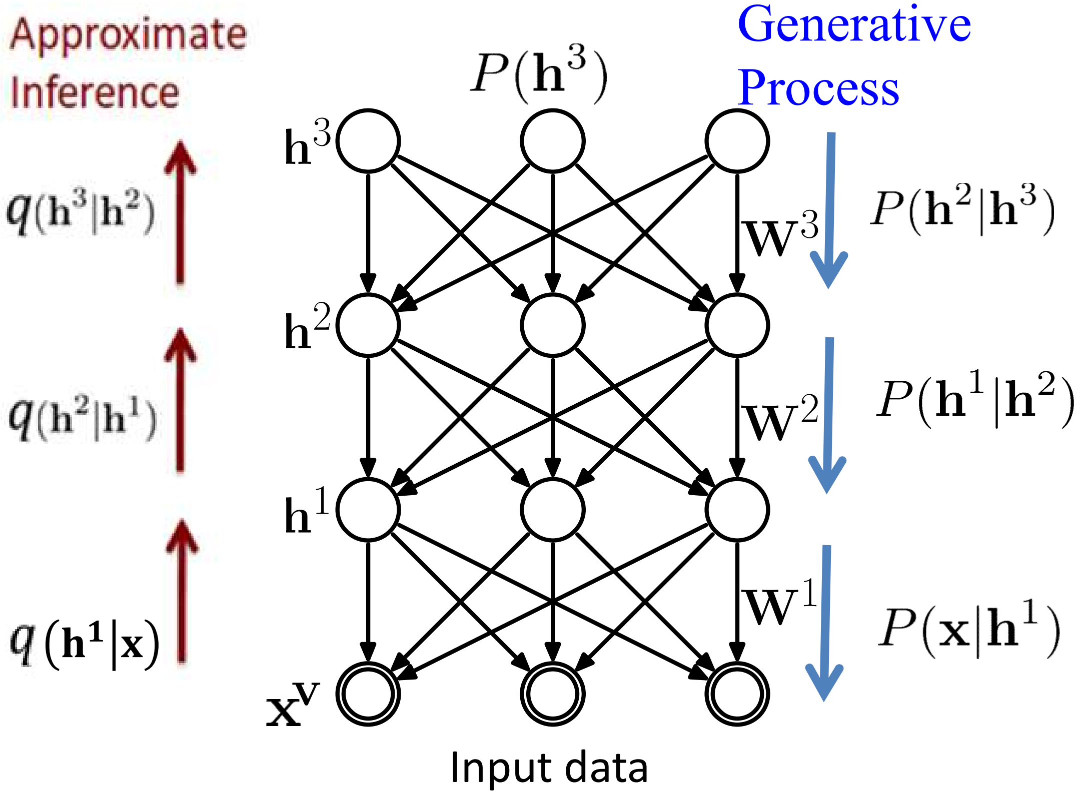
- **Consider multilevel features:**



Generative (Decoder) Network

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{\mathbf{h}^1,\ldots,\mathbf{h}^L} p(\mathbf{h}^L|\boldsymbol{\theta})p(\mathbf{h}^{L-1}|\mathbf{h}^L,\boldsymbol{\theta})\cdots p(\mathbf{x}|\mathbf{h}^1,\boldsymbol{\theta})$$

Each term may denote a complicated nonlinear relationship



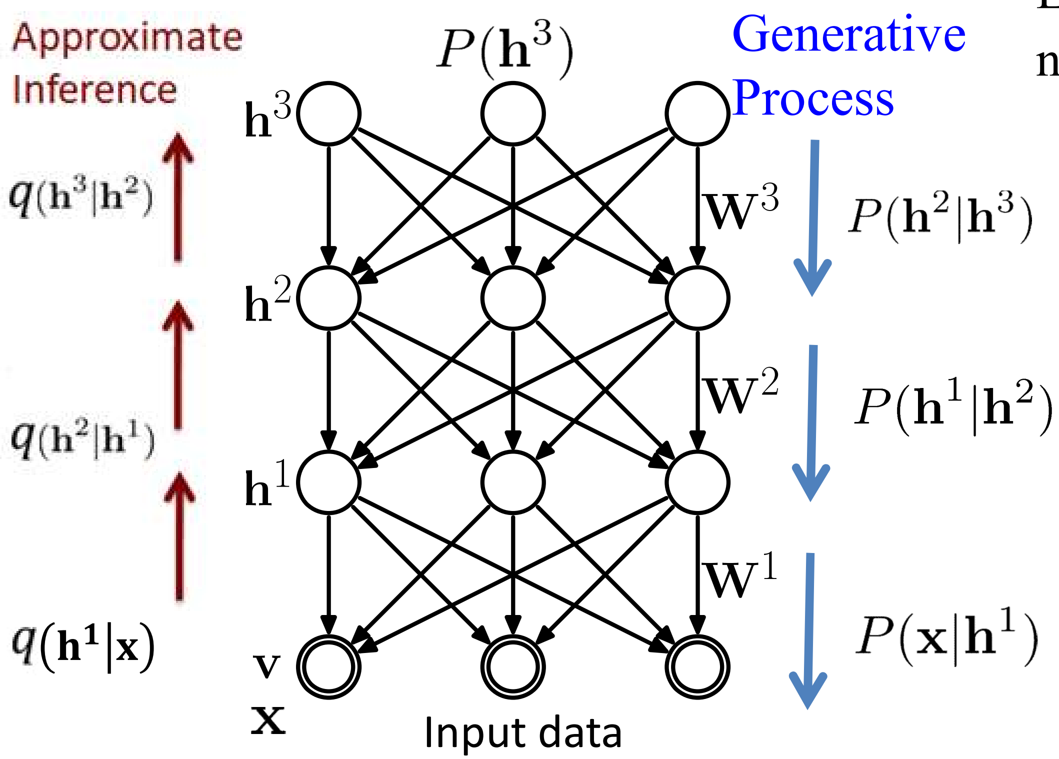
- $\boldsymbol{\theta}$ denotes parameters of decoder.
- L is the number of **stochastic** layers.
- Sampling and probability evaluation is tractable for each $p(\mathbf{h}^\ell|\mathbf{h}^{\ell+1})$.

Recognition (Encoder) Network

- The recognition model (**encoder**) is defined in terms of an analogous factorization:

$$q(\mathbf{h}|\mathbf{x},\boldsymbol{\lambda}) = q(\mathbf{h}^1|\mathbf{x},\boldsymbol{\lambda})\, q(\mathbf{h}^2|\mathbf{h}^1,\boldsymbol{\lambda}) \ldots q(\mathbf{h}^{\mathrm{L}}|\mathbf{h}^{\mathrm{L}-1},\boldsymbol{\lambda})$$

Each term may denote a complicated nonlinear relationship



We assume that

$$\mathbf{h}^L \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

The conditionals:

$$p(\mathbf{h}^\ell|\ \mathbf{h}^{\ell+1})$$
$$q(\mathbf{h}^\ell|\mathbf{h}^{\ell-1})$$

Using SGD for training Both Networks

- We can train decoder and encoder networks using SGD algorithm for minimizing the $\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) \triangleq$ -ELBO ($\boldsymbol{\mathcal{D}}$ denotes a mini-batch):

$$\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) = \sum_{\boldsymbol{x}\in\boldsymbol{\mathcal{D}}} \mathcal{L}_{\theta,\lambda}(\boldsymbol{x})$$

$$\mathcal{L}_{\theta,\lambda}(\boldsymbol{x}) = E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})\,]$$

- Obtaining Unbiased gradient w.r.t the decoder parameters, θ is simple:

$$\begin{aligned}
\nabla_\theta(\mathcal{L}_{\theta,\lambda}(\boldsymbol{x})) &= \nabla_\theta(E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})\,]) \\
&= E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[\nabla_\theta(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})] \\
&\simeq \nabla_\theta\;(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\, q_\lambda(\boldsymbol{z}|\boldsymbol{x})) \\
&= \nabla_\theta(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}))
\end{aligned}$$

Using SGD for training Both Networks

- Unbiased gradient w.r.t the variational parameters, λ is more difficult:
  - The Expectation is taken w.r.t the $q_\lambda(\mathbf{z}|\mathbf{x})$, which is a function of λ !!!

$$\nabla_\lambda(\mathcal{L}_{\theta,\lambda}(\boldsymbol{x})) = \nabla_\lambda(E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})] - E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\; q_\lambda(\boldsymbol{z}|\boldsymbol{x})\,])$$

$$\neq E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}\,[\nabla_\lambda(log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - log\; q_\lambda(\boldsymbol{z}|\boldsymbol{x})\,]$$

- Fortunately, for the continuous r.v's, the unbiased estimator of the gradient can be obtained through the reparameterization trick.

Reparameterization Trick

- This trick is essentially the application of change of variables:
  - Let z be a r.v distributed as $q_\lambda(z|x)$
  - Assume, there is a differentiable and invertible function h such that

$$z = h(\epsilon, x, \lambda)$$

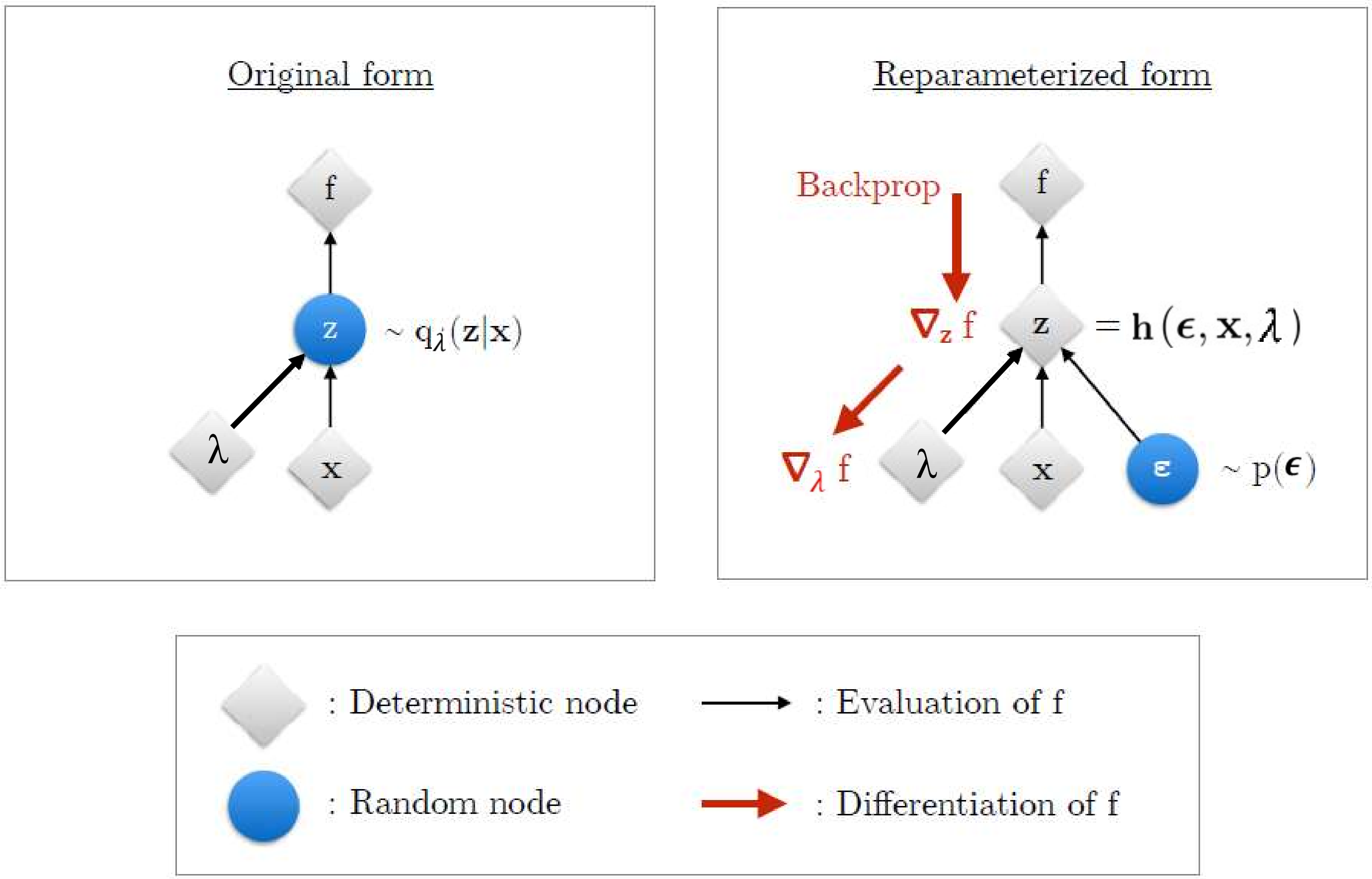
- Where $\varepsilon \sim p(\varepsilon)$ is another random variable which is independent of x and λ. Hence:

$$E_{q(z|x)}[f(z)] = E_{p(\varepsilon)}[f(z)]$$

- Now, we can safely exchange the gradient and expectation (f(z) $= log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})$):

$$\nabla_\lambda\ E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})\ ] = E_{p(\varepsilon)}\ [\nabla_\lambda(log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}))]$$
$$= \nabla_\lambda E_{p(\varepsilon)}[log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})] \simeq \nabla_\lambda(log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}))$$

Illustration of Reparameterization Trick



- The reparametrized ELBO estimator is referred to as the **Stochastic Gradient Variational Bayes (SGVB)** estimator (Kingma and Welling, 2014)

Image source: Kingma and Welling, 2019

Computing the Distribution of The Approximate Posterior After Change of Variables

- The goal is to compute the distribution of $log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ by changing z as a function of ε .
- This is easy task as long as the function h is chosen appropriately
- Using the change of variable rule in probability:

$$log\ q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \log p(\boldsymbol{\epsilon}) - \log|\det(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}})|$$

- $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}$ denotes the Jacobian matrix which is computed through $\boldsymbol{z} = h(\boldsymbol{\epsilon}, \boldsymbol{x}, \boldsymbol{\lambda})$.

Two common and simple choices for function h

1. Factorized Gaussian posterior:

- We can assume that $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ is given by

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, diag(\boldsymbol{\sigma}^2))$$

- In this case, variational parameters, and the approximate posterior, $q_\lambda(\boldsymbol{z}|\boldsymbol{x})$ are given by:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}) = \text{EncoderNN}_\lambda(\mathbf{x})$$

$$q_\lambda(\boldsymbol{z}|\boldsymbol{x}) = \prod_i \mathcal{N}(z_i; \mu_i, \sigma_i^2)$$

Univariate Gaussian Distribution

- We take h as an affine function of $\boldsymbol{\varepsilon}$ as follows:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$$

Jacobian: $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = diag(\boldsymbol{\sigma}) \xrightarrow{\text{yields}} \log \left|\det\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}\right)\right| = \sum_i log(\sigma_i)$

Two common and simple choices for function h

2. Full-covariance Gaussian posterior

- We can assume that $q_\lambda(z|\boldsymbol{x})$ is given by

$$q_\lambda(z|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$$

Cholesky Decomposition

- $\mathbf{L}$ is a lower or upper triangle matrix with non-zero entries in the diagonal.

- In this case, variational parameters is given by:

$$(\boldsymbol{\mu}, \log \boldsymbol{\sigma}, \boldsymbol{L}') = \text{EncoderNN}_\lambda(\mathbf{x})$$

$$\boldsymbol{L} = \boldsymbol{L}_{mask} \odot \mathbf{L}' + \operatorname{diag}(\boldsymbol{\sigma})$$

- L_{mask} is a 0-1 matrix with zero's on and above (lower) diagonal.

- We take h as an affine function of $\boldsymbol{\varepsilon}$ as follows:

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{L\epsilon}$$

$$\text{Jacobian: } \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}} = \boldsymbol{L} \xrightarrow{\text{yields}} \log \left| \det\left(\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{\epsilon}}\right) \right| = \sum_i log|L_{ii}|$$

Computing the Gradients

- The gradient w.r.t the parameters: both recognition and generative:

$$\nabla_{\theta,\lambda}\mathbb{E}_{z\sim q(z|\mathbf{x},\lambda)}\left[\log\frac{p(\mathbf{x},z|\boldsymbol{\theta})}{q(z|\mathbf{x},\lambda)}\right]$$

$$=\nabla_{\theta,\lambda}\mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\log\frac{p(\mathbf{x},h(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\theta})|\boldsymbol{\theta})}{q(\mathbf{h}(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\lambda})|\mathbf{x},\boldsymbol{\lambda})}\right]$$

$$=\mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\nabla_{\theta,\lambda}\log\frac{p(\mathbf{x},h(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\theta})|\boldsymbol{\theta})}{q(\mathbf{h}(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\lambda})|\mathbf{x},\boldsymbol{\lambda})}\right]$$

Gradients can be computed by backprop

The mapping $\boldsymbol{h}$ is a deterministic neural net for fixed $\boldsymbol{\epsilon}$.

Computing the Gradients

- The gradient w.r.t the parameters: recognition and generative:

is h, not g

$$\nabla_{\theta,\lambda}\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x},\lambda)}\left[\log\frac{p(\mathbf{x},\mathbf{z}|\boldsymbol{\theta})}{q(\mathbf{z}|\mathbf{x},\lambda)}\right] = \mathbb{E}_{\boldsymbol{\epsilon}^1,\ldots,\boldsymbol{\epsilon}^L\sim\mathcal{N}(\mathbf{0},\boldsymbol{I})}\left[\nabla_{\theta,\lambda}\log\frac{p(\mathbf{x},g(\boldsymbol{\epsilon},\mathbf{x},\boldsymbol{\theta})|\boldsymbol{\theta})}{q(h(\boldsymbol{\epsilon},\mathbf{x},\lambda)|\mathbf{x},\lambda)}\right]$$

- Approximate expectation by generating k samples from $\boldsymbol{\epsilon}$:

$$\frac{1}{k}\sum_{i=1}^{k}\nabla_{\theta,\lambda}\log w\left(\mathbf{x},\mathbf{h}(\boldsymbol{\epsilon}_i,\mathbf{x},\boldsymbol{\lambda}),\boldsymbol{\theta}\right)$$

- Where we defined unnormalized importance weights:

$$w(\mathbf{x},\mathbf{h},\boldsymbol{\theta}) = p(\mathbf{x},\mathbf{h}|\boldsymbol{\theta})/q(\mathbf{h}|\mathbf{x},\boldsymbol{\lambda})$$

Relation between ML and ELBO

- With i.i.d samples from dataset $\mathcal{D}$, the maximum likelihood principle is given by:

$$log\ p_\theta(\mathcal{D}) = \frac{1}{N}\sum_{\boldsymbol{x}\in\mathcal{D}} log\ p_\theta(\boldsymbol{x}) = E_{q_{\mathcal{D}}(\boldsymbol{x})} \log p_\theta(\boldsymbol{x})$$

- $q_{\mathcal{D}}(\boldsymbol{x})$ is the empirical distribution of data
- Recall that

$$Max_\theta\ log\ p_\theta(\mathcal{D}) = \operatorname*{Min}_\theta D_{kl}(q_{\mathcal{D}}(\boldsymbol{x})||p_\theta(\boldsymbol{x}))$$

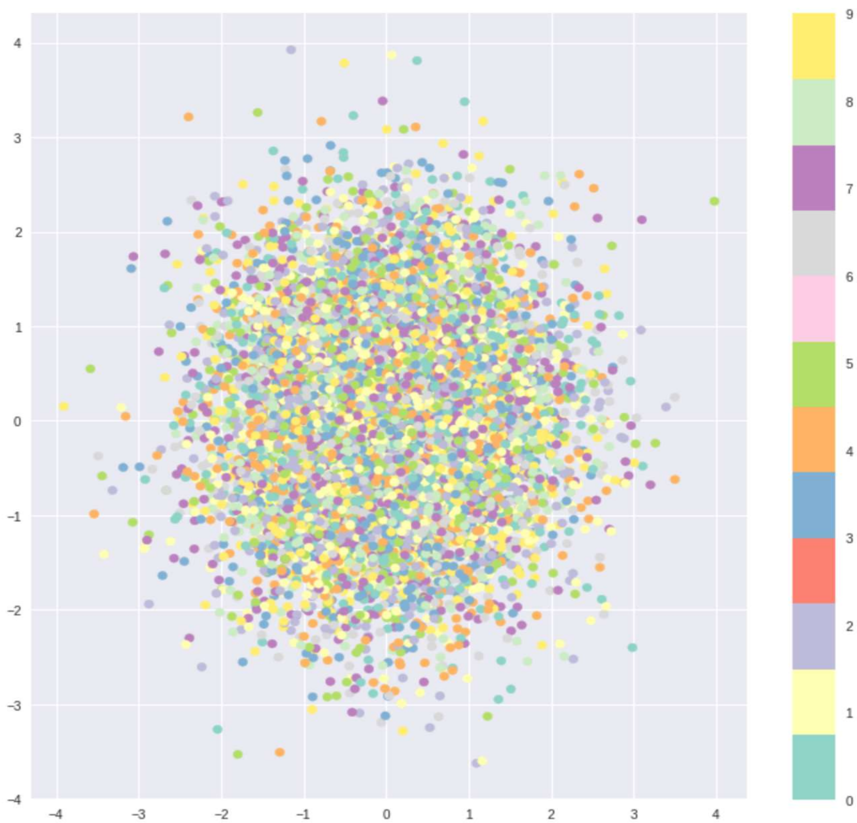
- Combining the empirical distribution of data and the inference model in VAE, $q_{\lambda,\mathcal{D}}(\boldsymbol{x},\boldsymbol{z})$=$q_{\mathcal{D}}(\boldsymbol{x})q_\lambda(\boldsymbol{z}|\mathrm{x})$, one can show that

$$D_{kl}(q_{\lambda,\mathcal{D}}(\boldsymbol{x},\boldsymbol{z})||p_\theta(\boldsymbol{x},\boldsymbol{z})) \geq D_{kl}(q_{\mathcal{D}}(\boldsymbol{x})||p_\theta(\boldsymbol{x}))$$

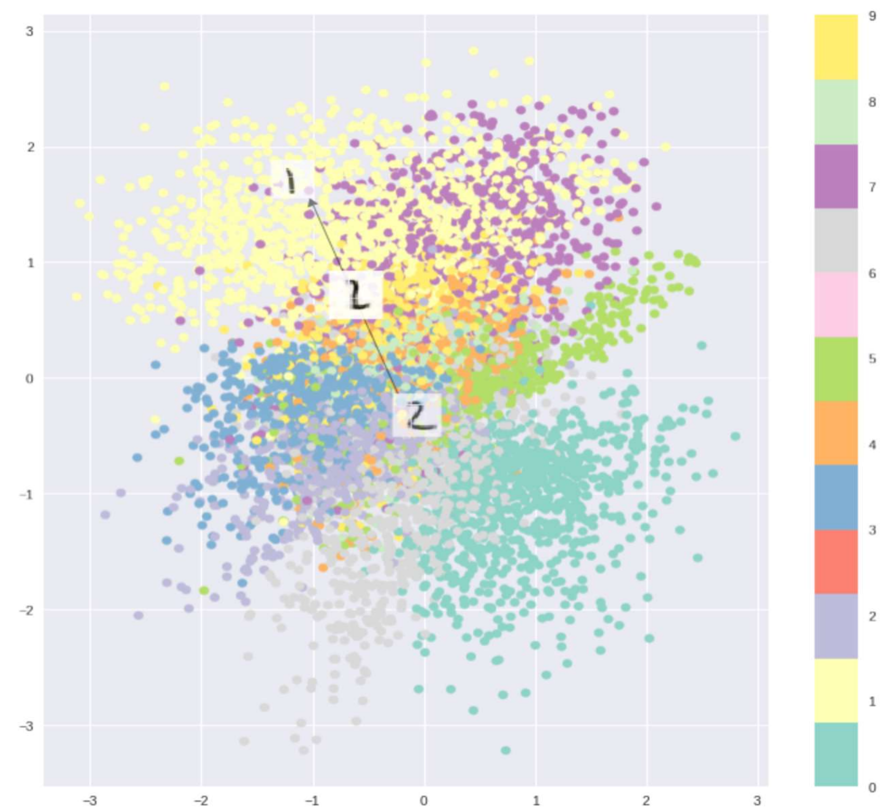
- ELBO can be thought as a maximum likelihood objective *in an augmented space,* $(\boldsymbol{x},\boldsymbol{z})$

Trade-off in the VAE loss

$$E_{q_\lambda(z|x_i)}[\log p(x_i \mid z)] - D(q_\lambda(z|x) \,||\, p(z))$$



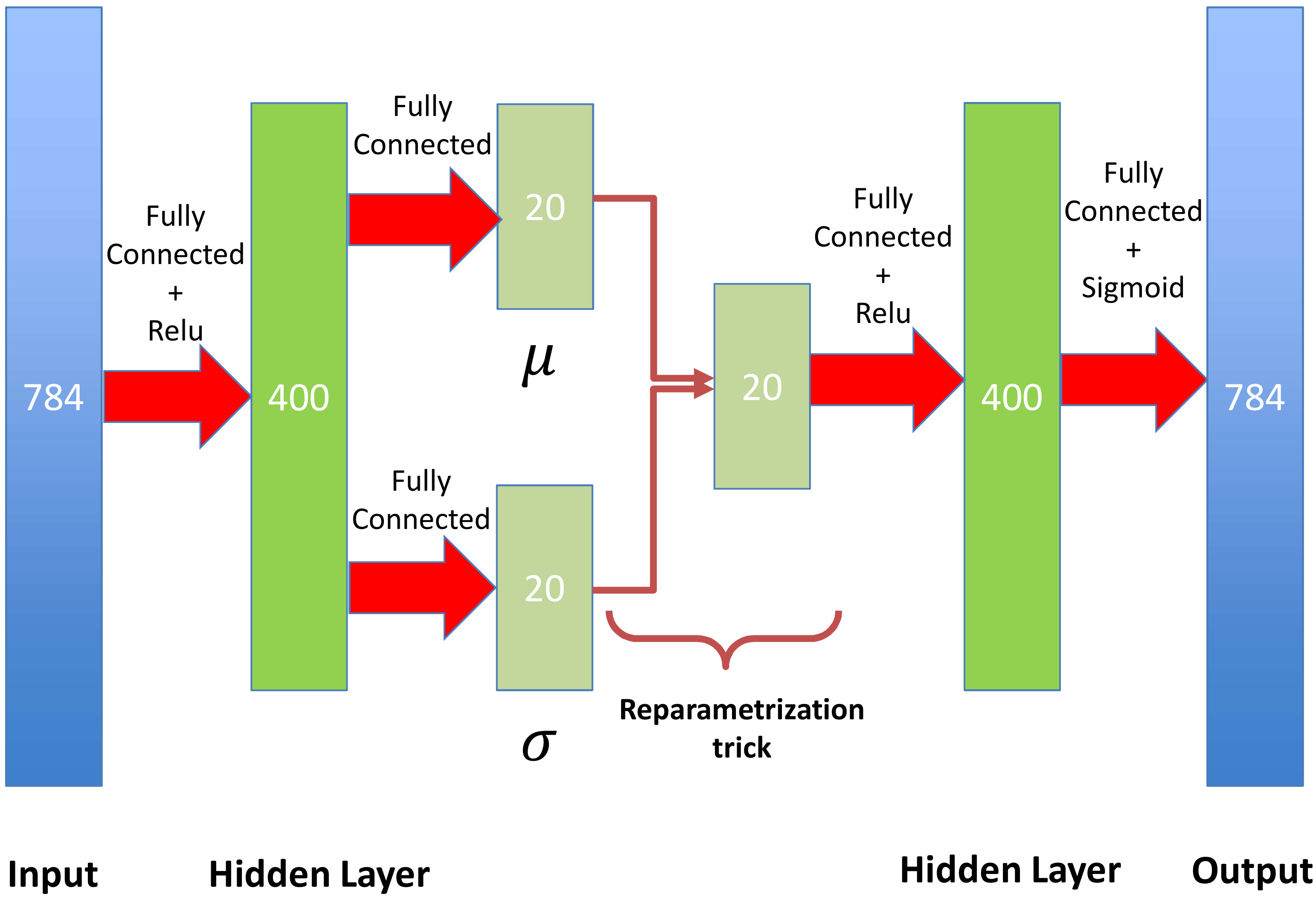
Optimizing using the second term (KL divergence) in the loss



Optimizing using both reconstruction loss (likelihood term) and KL divergence term in the loss

https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf

VAE Architectures for MNIST



MNIST Experiment

Reconstruction

1st epoch

5th epoch

10th epoch

- Adam optimizer, learning rate=0.001, batch size = 128, 10 epochs, no image normalization
- Reconstruction of images in the output of VAE in different epochs

MNIST Experiment

Sampling (generating)

1st epoch

5th epoch

10th epoch

- Generating of images in the output of VAE in different epochs by sampling a fixed random vector $Z \in \mathbb{R}^{128\times 20}$ $with\ Z_i \sim \mathcal{N}(0, I_{20\times 20}),\quad i = 1,2,\ldots,128$

Other Variants

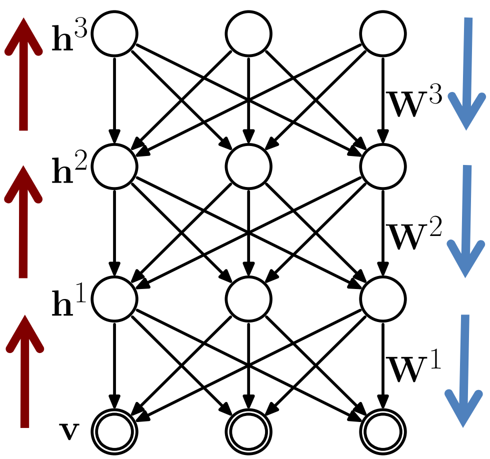
Importance Weighted Autoencoders

- Consider the following k-sample importance weighting of the log-likelihood ($z_i = h_i = h(\epsilon_i, \mathbf{x}, \lambda)$):

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{1}{k}\sum_{i=1}^{k}\frac{p(\mathbf{x},\mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}\right]$$

$$= \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})}\left[\log \frac{1}{k}\sum_{i=1}^{k} w_i\right]$$

Unnormalized importance weights



Input data

where $\mathbf{h}_1, \ldots, \mathbf{h}_k$ are sampled from the recognition network.

Importance Weighted Autoencoders

- Consider the following k-sample importance weighting of the log-likelihood ($z_i = h_i = h(\epsilon_i, \mathbf{x}, \lambda)$):

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k \sim q(\mathbf{h}|\mathbf{x})} \left[\log \frac{1}{k} \sum_{i=1}^{k} \frac{p(\mathbf{x}, \mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})} \right]$$

- This is a lower bound on the marginal log-likelihood:

$$\mathcal{L}_k(\mathbf{x}) = \mathbb{E}\left[\log \frac{1}{k} \sum_{i=1}^{k} w_i \right] \leq \log \mathbb{E}\left[\frac{1}{k} \sum_{i=1}^{k} w_i \right] = \log p(\mathbf{x})$$

Calculate this

- Special Case of k=1: Same as standard VAE objective.

- Using more samples → Improves the tightness of the bound.

Tighter Lower Bound

- Using more samples can only improve the tightness of the bound.

- For all k, the lower bounds satisfy:

$$\log p(\mathbf{x}) \geq \mathcal{L}_{k+1}(\mathbf{x}) \geq \mathcal{L}_k(\mathbf{x})$$

- Moreover if $p(\mathbf{h}, \mathbf{x})/q(\mathbf{h}|\mathbf{x})$ is bounded, then:

$$\mathcal{L}_k(\mathbf{x}) \to \log p(\mathbf{x}), \quad \text{as} \quad k \to \infty$$

Computing the Gradients

- We can use the unbiased estimate of the gradient using reparameterization trick:

$$\nabla_{\theta,\lambda}\mathcal{L}_k(\mathbf{x}) = \nabla_{\theta,\lambda}\mathbb{E}_{\mathbf{h}_1,\ldots,\mathbf{h}_k\sim q(\mathbf{h}|\mathbf{x})}\left[\log\frac{1}{k}\sum_{i=1}^{k} w_i\right]$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_k}\left[\nabla_{\theta,\lambda}\log\frac{1}{k}\sum_{i=1}^{k} w(\mathbf{x}, h(\boldsymbol{\epsilon}_i,\mathbf{x},\lambda),\boldsymbol{\theta})\right]$$

$$= \mathbb{E}_{\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_k}\left[\sum_{i=1}^{k}\widetilde{w}_i\nabla_{\theta,\lambda}\log w(\mathbf{x}, h(\boldsymbol{\epsilon}_i,\mathbf{x},\lambda),\boldsymbol{\theta})\right]$$

- Where we define normalized importance weights:

$$\widetilde{w}_i = w_i/\sum_{i=1}^{k} w_i, \quad \text{where } w_i = \frac{p(\mathbf{x},\mathbf{h}_i)}{q(\mathbf{h}_i|\mathbf{x})}$$

IWAEs vs. VAEs

- Draw k-samples form the recognition network $q_\lambda(\mathbf{h}|\mathbf{x})$
  - or k-sets of auxiliary variables ϵ.

- Obtain the following Monte Carlo estimate of the gradient (IWAE):

$$\nabla_{\theta,\lambda}\mathcal{L}_k(\mathbf{x}) \approx \sum_{i=1}^{k} \widetilde{w}_i \boxed{\nabla_{\theta,\lambda} \log w(\mathbf{x}, \mathbf{h}(\boldsymbol{\epsilon}_i, \mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\theta})}$$

- Compare this to the VAE's estimate of the gradient (VAE):

$$\nabla_{\theta,\lambda}\mathcal{L}(\mathbf{x}) \approx \frac{1}{k}\sum_{i=1}^{k} \boxed{\nabla_{\theta,\lambda} \log w(\mathbf{x}, \mathbf{h}(\boldsymbol{\epsilon}_i, \mathbf{x}, \boldsymbol{\lambda}), \boldsymbol{\theta})}$$

Conditional VAE

Conditional VAE (CWAE)

- No control on the data generation process on VAE
  - e.g., We want to generate only a digit 2
- Conditioning all the distributions on what we want, the objective of CWAE (conditional ELBO):

$$E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \mathbf{z}, \mathbf{c}) - D_{kl}(q_\lambda(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{c}) \,||\, p(\boldsymbol{z}|\boldsymbol{c}))]$$

- $\boldsymbol{C}$ denotes the conditioning vector (e.g., a code for digit 2).

CWAE in Practice

- Conditioned image generation

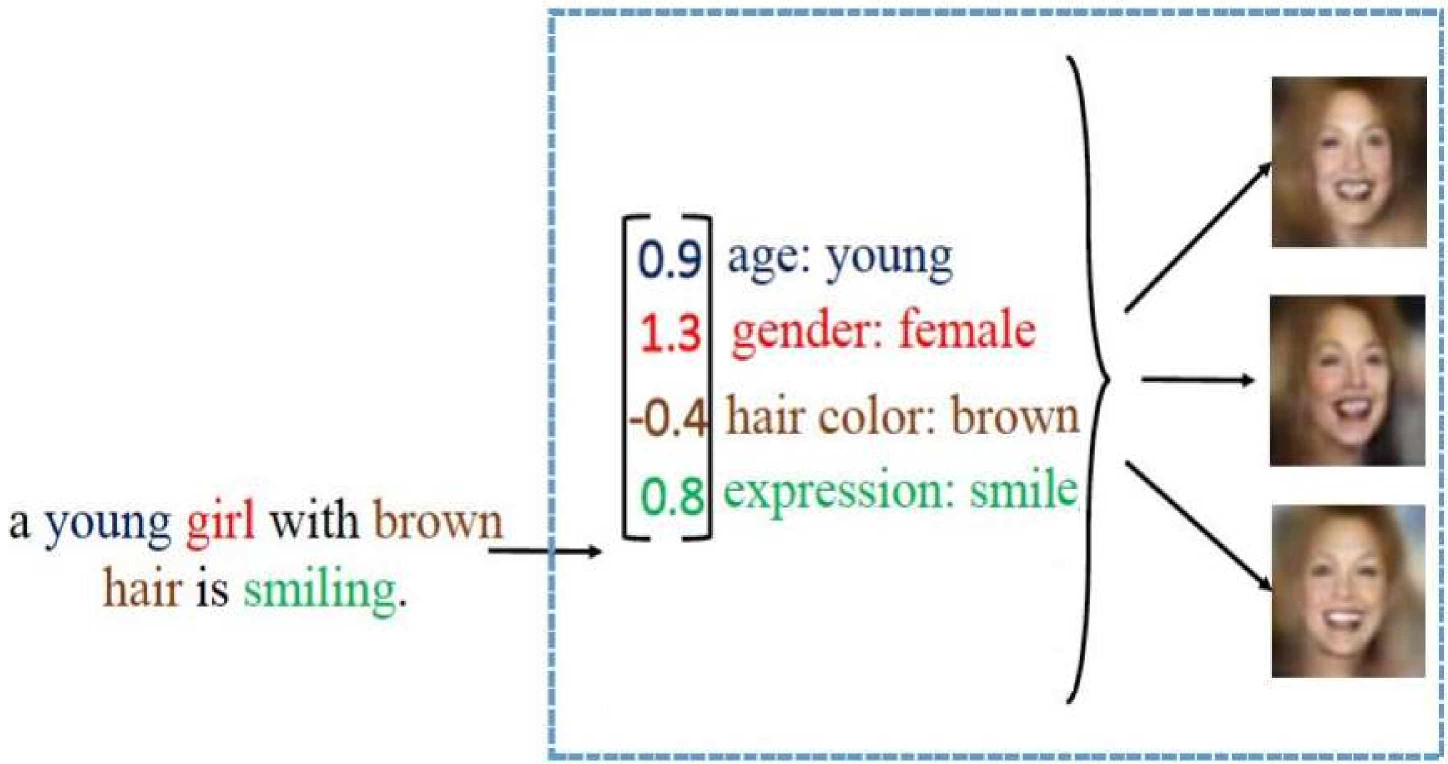


Image source: Yeh et al., lecture notes, 2017

β-VAE

Disentangled Latent Features—β-VAE

- A latent variable z is called disentangled factor if it is
  - Only sensitive to one single generative factor
  - Relatively invariant to other factors
- Advantage:
  - Interpretability
  - Easy generalization to a different task
- Example: Disentangled factors in Human face
  - Skin color
  - Hair length
  - Having glass or not

Disentangled Latent Features—$\boldsymbol{\beta}$-VAE

- β-VAE was proposed by Higgins et al., 2017 to discover the disentangled latent representation:

$$\underset{\theta,\lambda}{\text{Max}}\, E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z})]$$
$$s.t.\;\; D_{kl}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z})) < \delta$$

- If β=1, this is regular VAE
- If $\beta > 1$, a stronger constraint on the latent factors; hence, limiting the representation capacity of z
- The larger β, the better discovering of disentangled latent features

Using Lagrangian form

$$\underset{\theta,\lambda}{\text{Max}}\, E_{q_\lambda(\boldsymbol{z}|\boldsymbol{x})}[log\, p_\theta(\boldsymbol{x},\boldsymbol{z}) - \beta D_{kl}(q_{\boldsymbol{\lambda}}(\boldsymbol{z}|\boldsymbol{x}) \,||\, p(\boldsymbol{z}))]$$

β-VAE in Practice

- β-VAE: $\beta = 250$
- VAE: $\beta = 1$

- Disentangled latent factors: Azimuth, Emotion, Hair
- β-VAE learns to **disentangle** factors
- VAE learns an **entangled** representation
  - ➢ Entangling azimuth with emotion
  - ➢ presence of glasses and gender

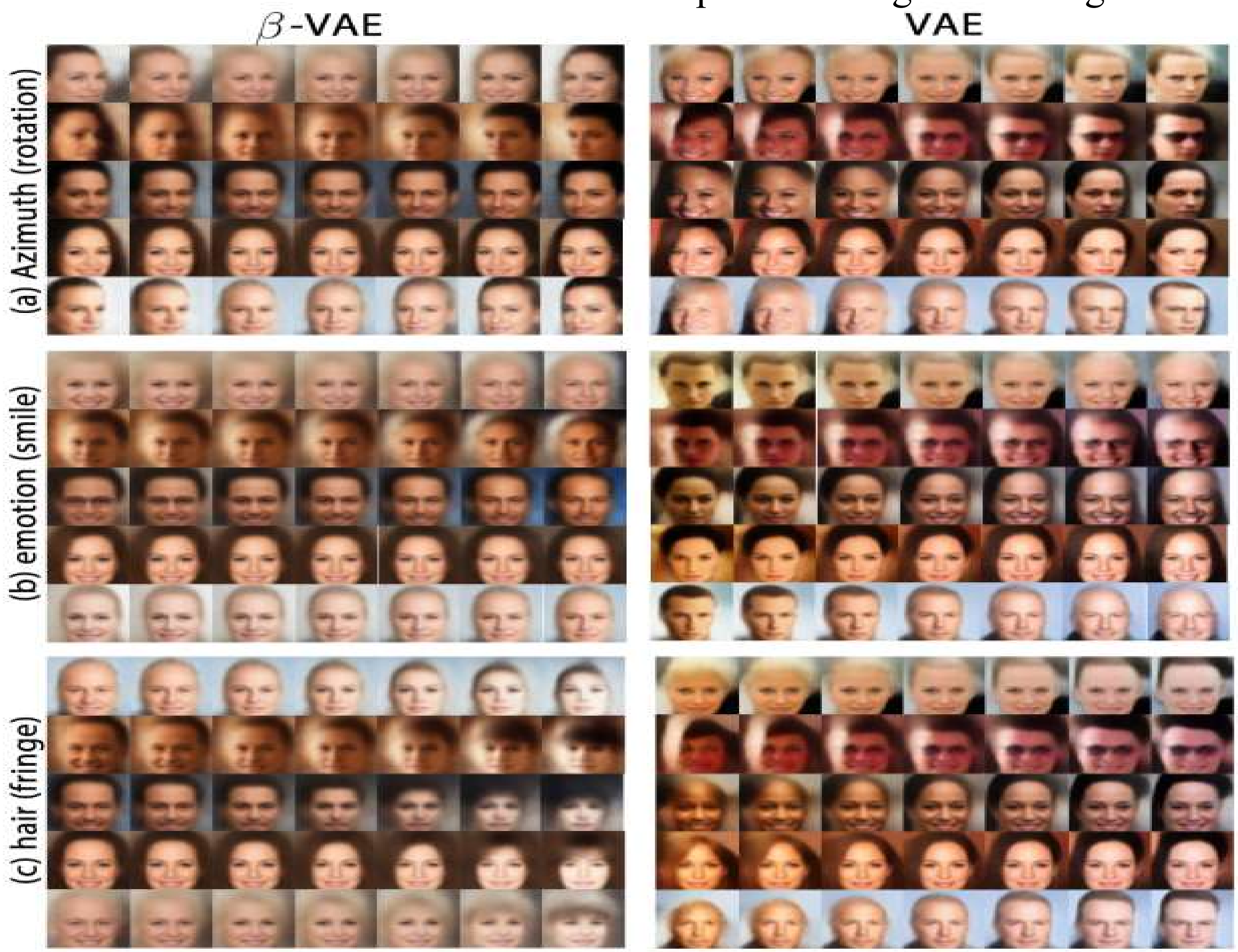


Image source: Higgins et al., 2017

β-VAE in Practice

- β-VAE: $\beta = 5$
- VAE: $\beta = 1$

- Disentangled latent factors: Azimuth, Chair Width, Leg Style
- β-VAE learns to **disentangle** factors
- VAE learns an **entangled** representation
  - ➢ Entangling chair width with azimuth and leg style

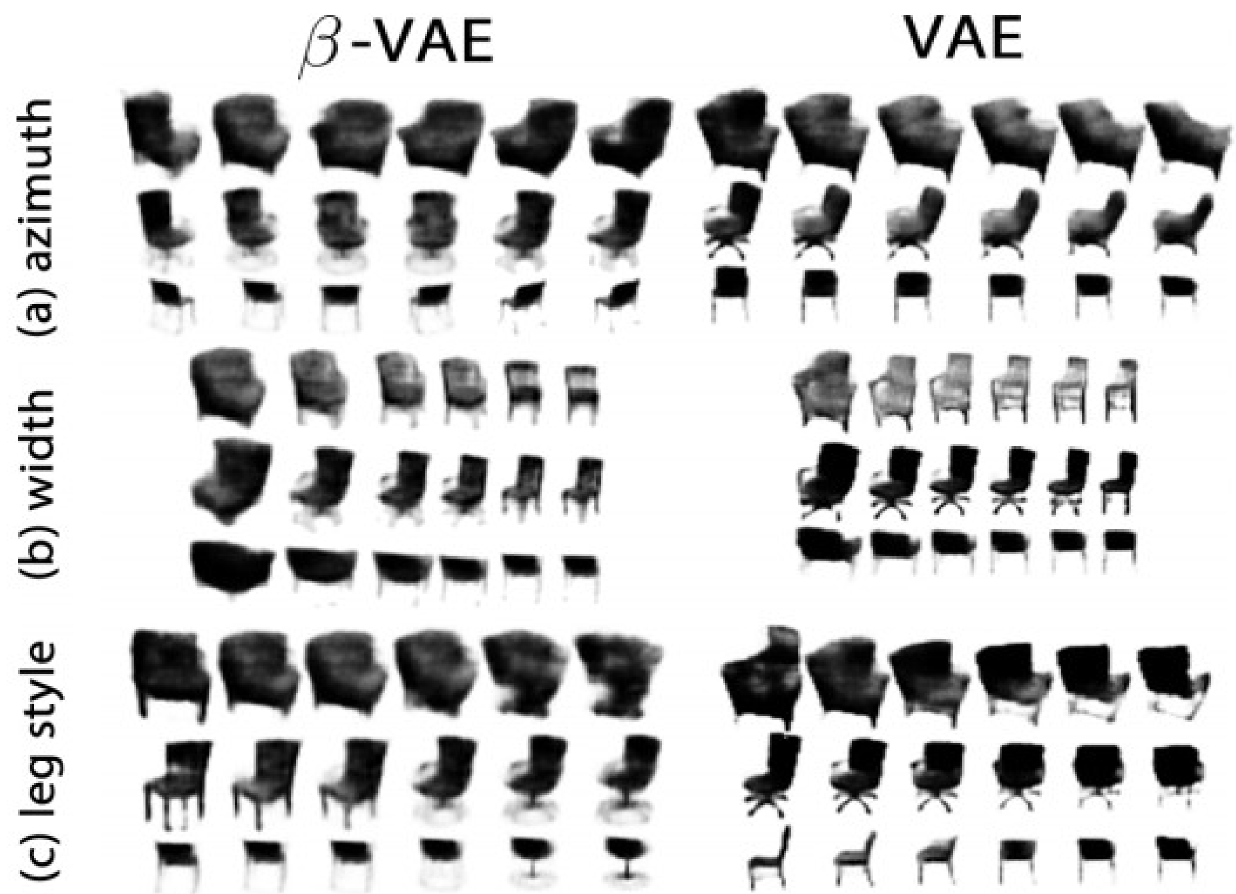
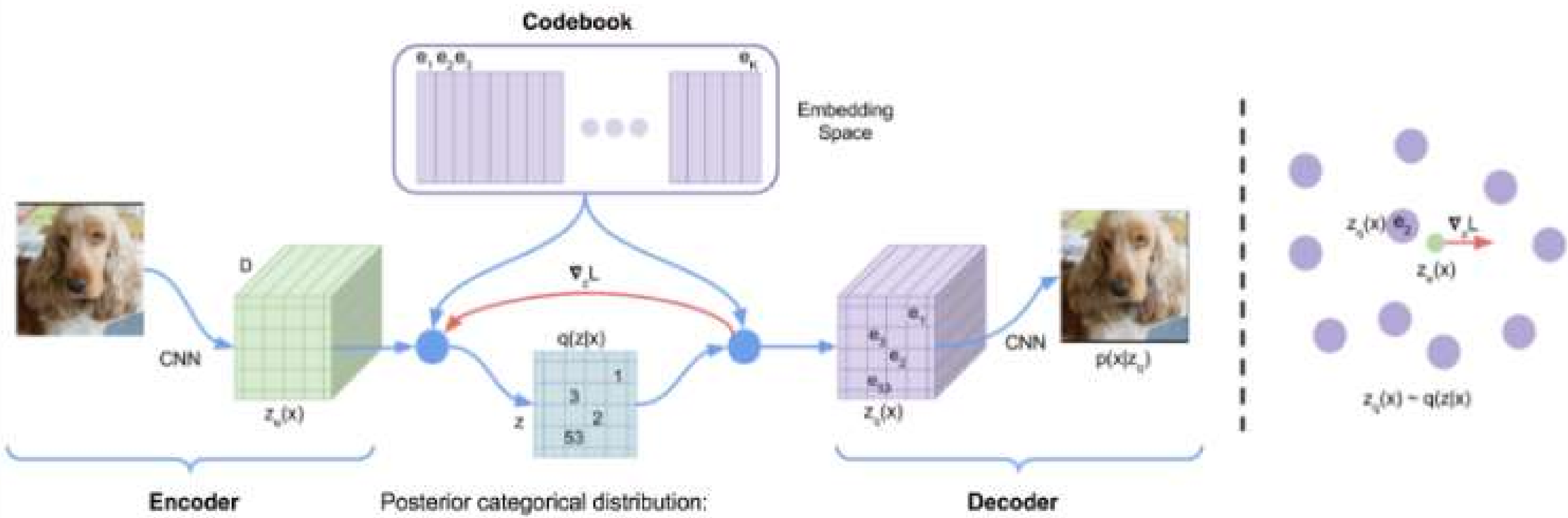


Image source: Higgins et al., 2017

Vector Quantized-VAE

Vector Quantized-Variational AutoEncoder (VQ-VAE)

- Learning a discrete latent variable by the encoder



Posterior categorical distribution:

$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

- Mapping K-dimensional vectors into a finite set of "code" vectors.
- Similar to K-means algorithm

Image source: van den Oord, et al. 2017