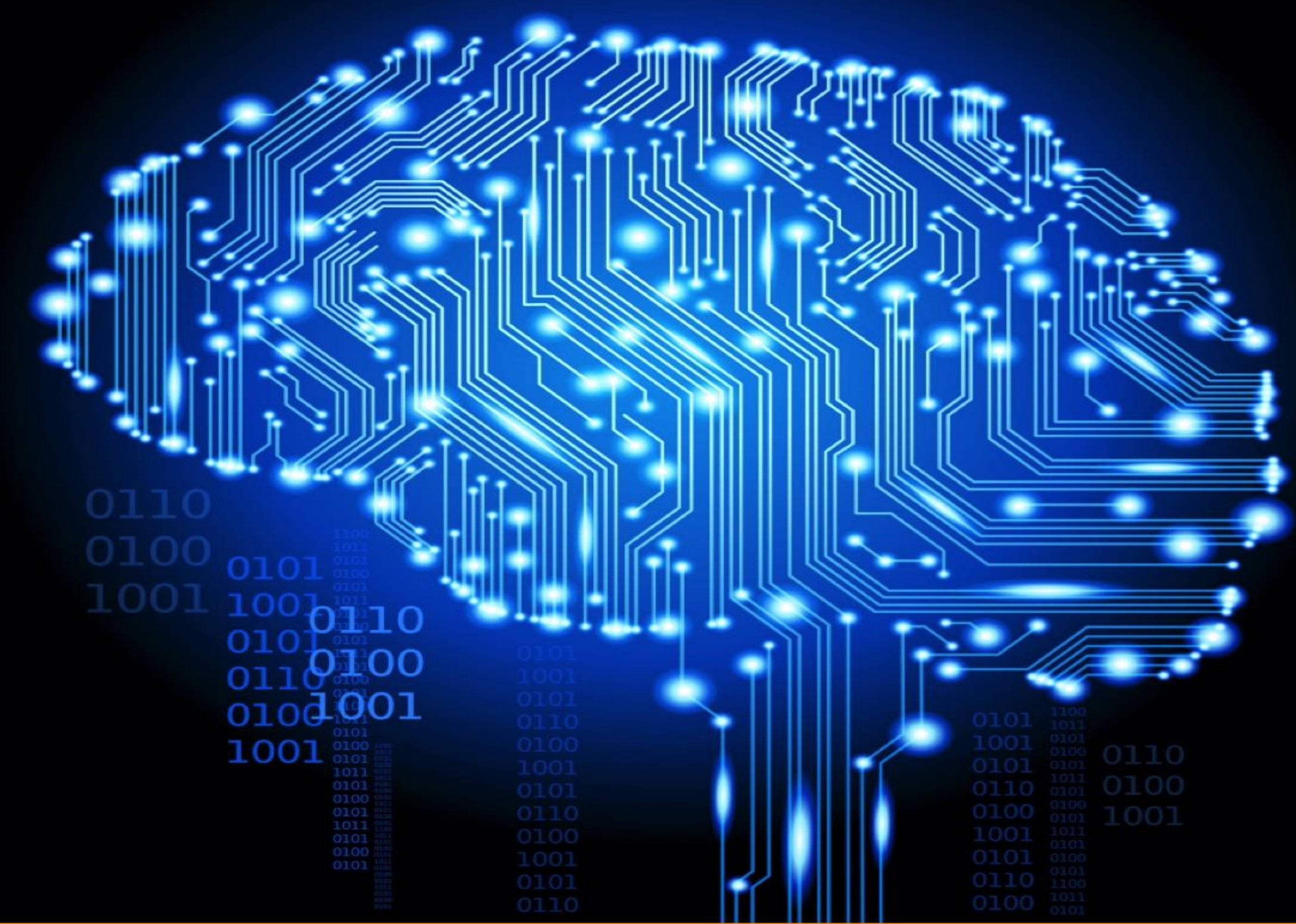


Lecture 9: Deep Generative Models

Efstratios Gavves

Lecture overview

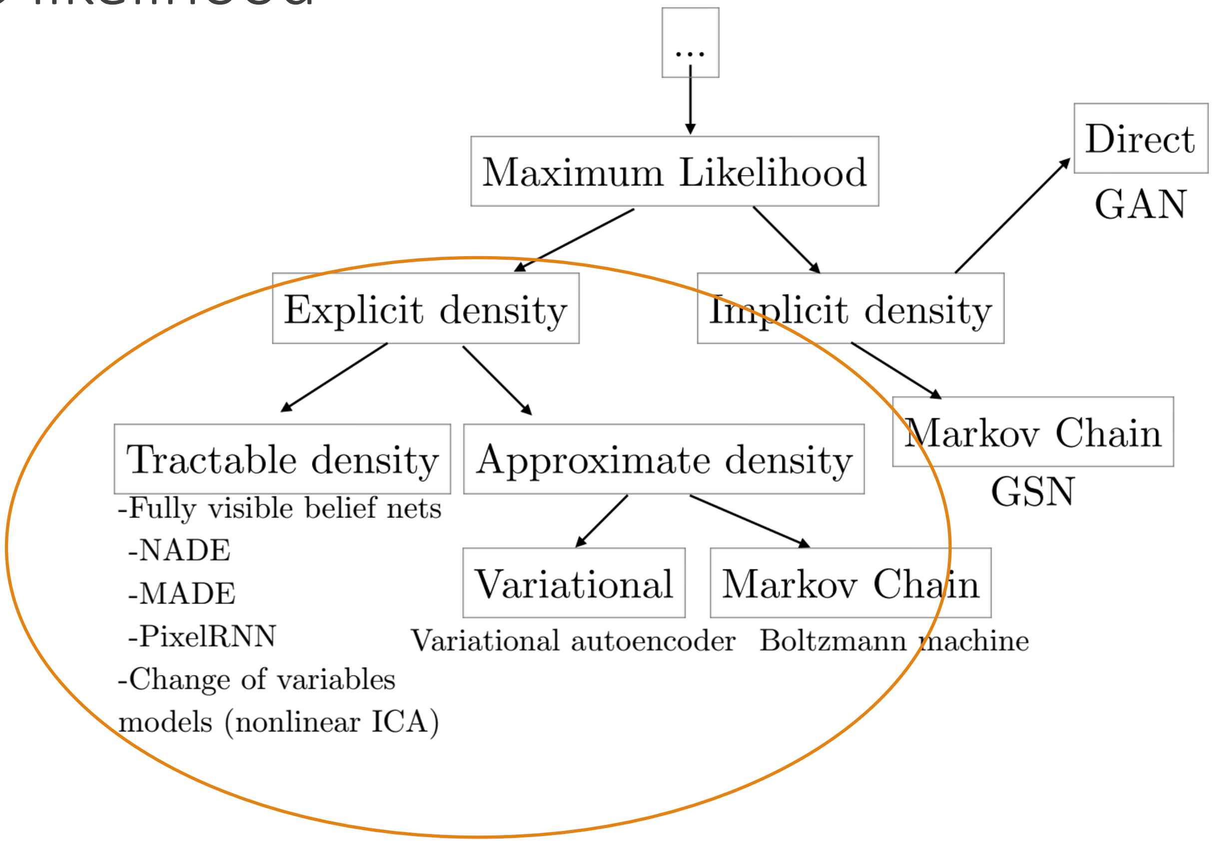
- Early Generative Models
- Restricted Boltzmann Machines
- Deep Boltzmann Machines
- Deep Belief Network
- Contrastive Divergence
- Gentle intro to Bayesian Modelling and Variational Inference
- Variational Autoencoders
- Normalizing Flows

Explicit density models

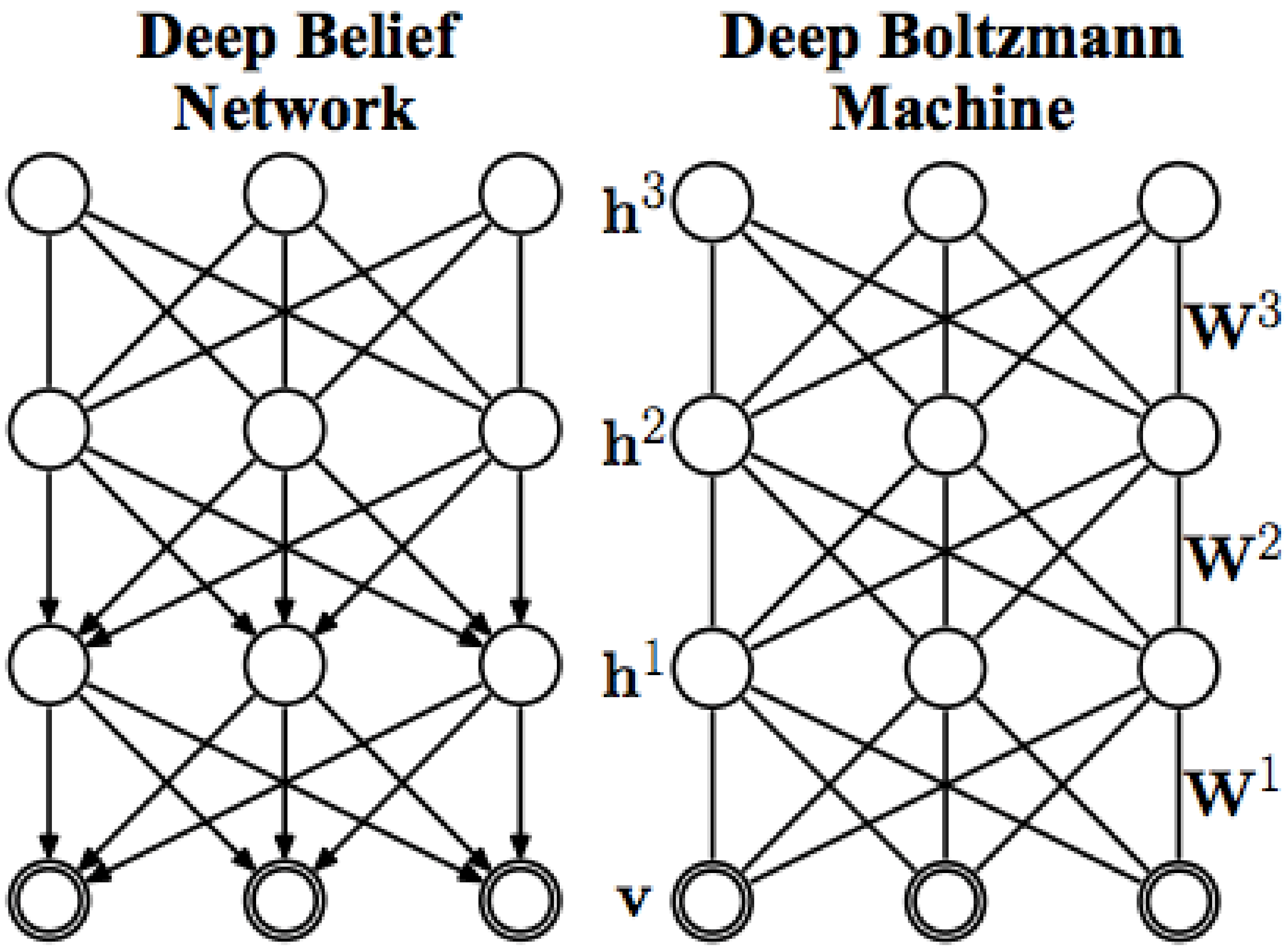
- Plug in the model density function to likelihood
- Then maximize the likelihood

Problems

- Design complex enough model that meets data complexity
- At the same time, make sure model is computationally tractable
- More details in the next lecture



Restricted Boltzmann Machines
Deep Boltzmann Machines
Deep Belief Nets



How to define a generative model?

- We can define an explicit density function over all possible relations ψ_c between the input variables x_c

$$p(x) = \prod_c \psi_c\,(x_c)$$

- Quite inefficient → think of all possible relations between $256 \times 256 = 65K$ input variables
  - Not just pairwise

- Solution: Define an energy function to model these relations

Boltzmann Distribution

- First, define an energy function $-E(x)$ that models the joint distribution

$$p(x) = \frac{1}{Z} \exp(-E(x))$$

- Z is a normalizing constant that makes sure $p(x)$ is a pdf: $\int p(x) = 1$

$$Z = \sum_x \exp(-E(x))$$

Why Boltzmann?

- Well understood in physics, mathematics and mechanics
- A Boltzmann distribution (also called Gibbs distribution) is a probability distribution, probability measure, or frequency distribution of particles in a system over various possible states
- The distribution is expressed in the form

$$F(state) \propto \exp(-\frac{E}{kT})$$

- E is the state energy, k is the Boltzmann constant, T is the thermodynamic temperature

https://en.wikipedia.org/wiki/Boltzmann_distribution

Problem with Boltzmann Distribution?

Problem with Boltzmann Distribution?

- Assuming binary variables x the normalizing constant has very high computational complexity
- For n-dimensional x we must enumerate all possible 2^n operations for Z
- Clearly, gets out of hand for any decent n
- Solution: Consider only pairwise relations

Boltzmann Machines

- The energy function becomes

$$E(x) = -x^T W x - b^T x$$

- x is considered binary
- $x^T W x$ captures correlations between input variables
- $b^T x$ captures the model prior
  - The energy that each of the input variable contributes itself

Problem with Boltzmann Machines?

Problem with Boltzmann Machines?

- Still too complex and high-dimensional
- If x has $256 \times 256 = 65536$ dimensions
- The pairwise relations need a huge W: 4.2 billion dimensions

- <u>Just for connecting two layers!</u>

- Solution: Consider latent variables for model correlations

Restricted Boltzmann Machines

- Restrict the model energy function further to a bottleneck over latents h

$$E(x) = -x^T W h - b^T x - c^T h$$

Restricted Boltzmann Machines

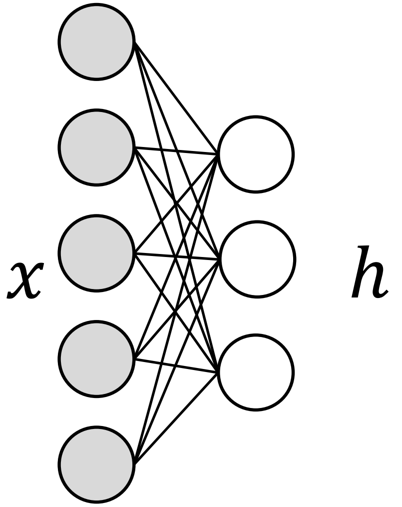
- $E(x) = -x^T W h - b^T x - c^T h$
- The $x^T W h$ models correlations between x and the latent activations via the parameter matrix W
- The $b^T x, c^T h$ model the priors
- Restricted Boltzmann Machines (RBM) assume x, h to be binary

Restricted Boltzmann Machines

- Energy function: $E(x) = -x^T W h - b^T x - c^T h$

$$p(x) = \frac{1}{Z} \sum_h \exp(-E(x,h))$$

  - Not in the form $\propto \exp(\mathrm{x})/\mathrm{Z}$ because of the $\sum$



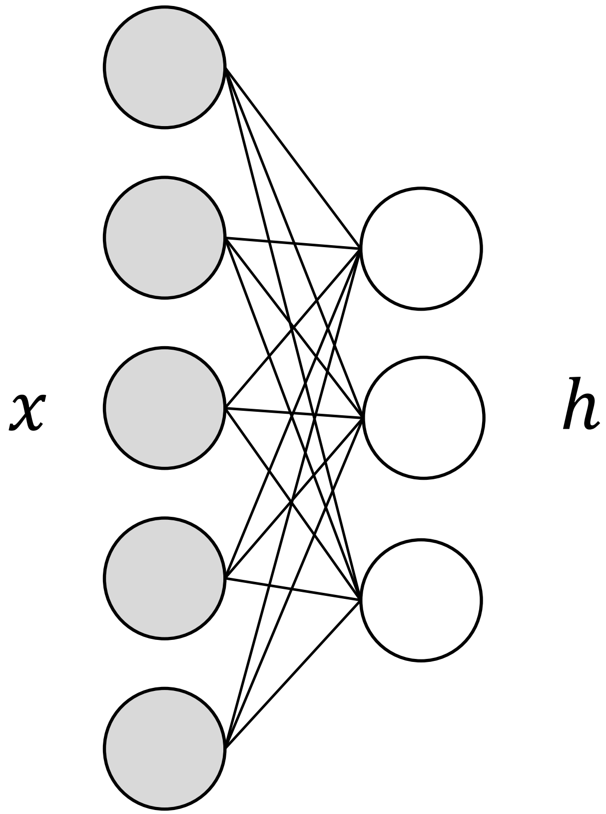
- Free energy function: $F(x) = -b^T x - \sum_i \log \sum_{h_i} \exp(h_i(c_i + W_i x))$

$$p(x) = \frac{1}{Z} \exp(-F(x))$$

$$Z = \sum_x \exp(-F(x))$$

Restricted Boltzmann Machines

- The $F(x)$ defines a bipartite graph with undirected connections
  - Information flows forward and backward



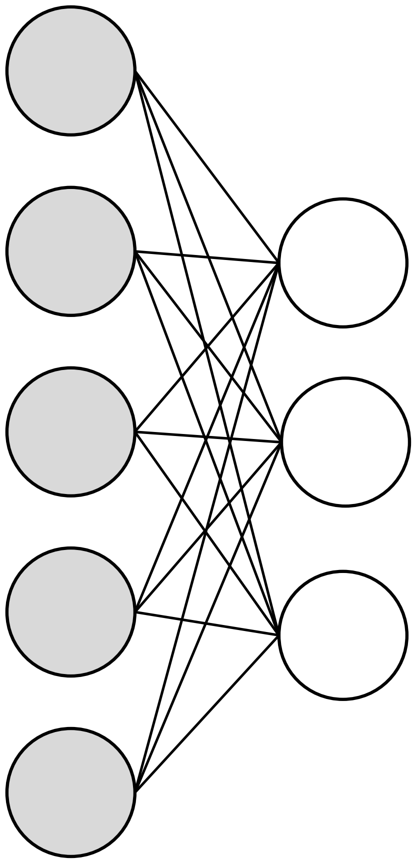
Restricted Boltzmann Machines

- The hidden units h_j are independent to each other conditioned on the visible units

$$p(h|x) = \prod_j p\left(h_j|x, \theta\right)$$

- The hidden units x_i are independent to each other conditioned on the visible units

$$p(x|h) = \prod_i p(x_i|h, \theta)$$



Training RBMs

- The conditional probabilities are defined as sigmoids

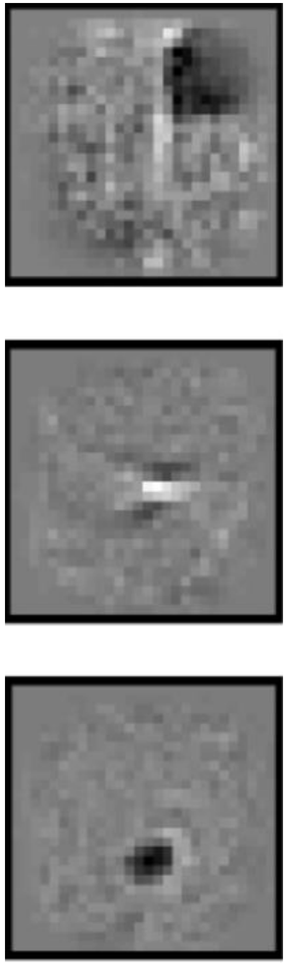
$$p(h_j|x,\theta) = \sigma(W_{\cdot j}x + b_j)$$
$$p(x_i|h,\theta) = \sigma(W_{\cdot i}x + c_i)$$

- Maximize log-likelihood

$$\mathcal{L}(\theta) = \frac{1}{\mathrm{N}} \sum_n \log p(x_n|\theta)$$

and

$$p(x) = \frac{1}{Z} \exp(-F(x))$$



Hidden unit (features)

Training RBMs

- Let's take the gradients

$$\frac{\partial \log p(x_n|\theta)}{\partial \theta} = -\frac{\partial F(x_n)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta}$$

$$= -\sum_h p(h|x_n,\theta)\frac{\partial E(x_n|h,\theta)}{\partial \theta} + \sum_{\tilde{x},h} p(\tilde{x},h|\theta)\frac{\partial E(\tilde{x},h|\theta)}{\partial \theta}$$

Hidden unit (features)

Training RBMs

- Let's take the gradients

$$\frac{\partial \log p(x_n|\theta)}{\partial \theta} = -\frac{\partial F(x_n)}{\partial \theta} - \frac{\partial \log Z}{\partial \theta}$$

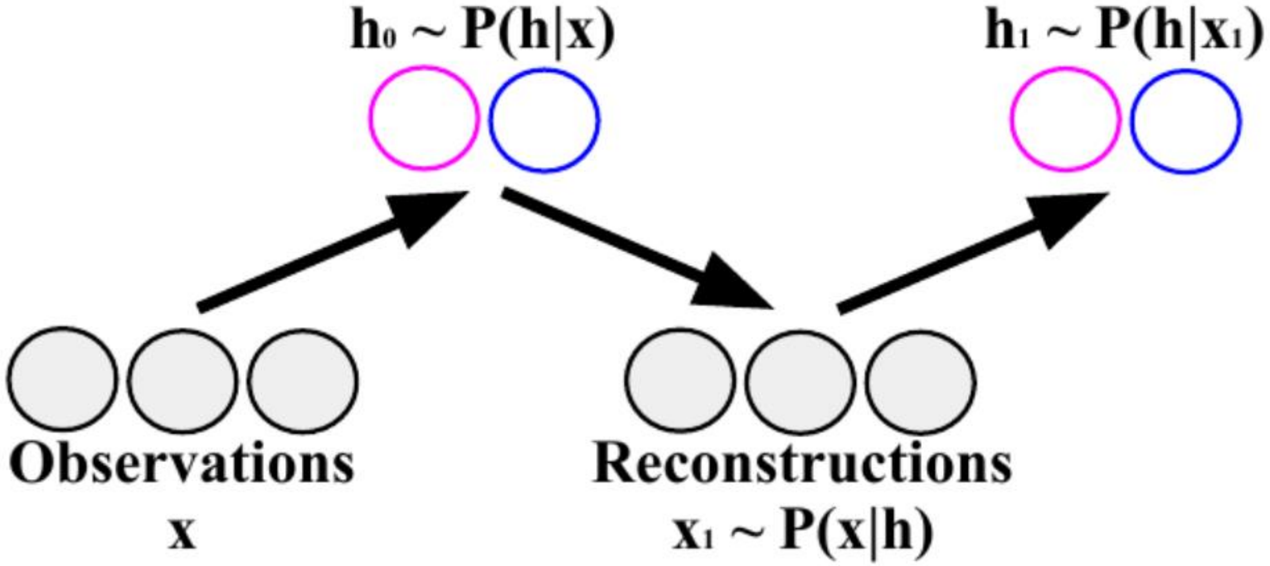
$$= -\sum_h p(h|x_n,\theta)\frac{\partial E(x_n|h,\theta)}{\partial \theta} + \sum_{\tilde{x},h} p(\tilde{x},h|\theta)\frac{\partial E(\tilde{x},h|\theta)}{\partial \theta}$$

- Easy because we just substitute in the definitions the x_n and sum over h
- Hard because you need to sum over both $\tilde{x}, h$ which can be huge
  - It requires approximate inference, *e.g.,* MCMC

Training RBMs with Contrastive Divergence

- Approximate the gradient with Contrastive Divergence
- Specifically, apply Gibbs sampler for k steps and approximate the gradient

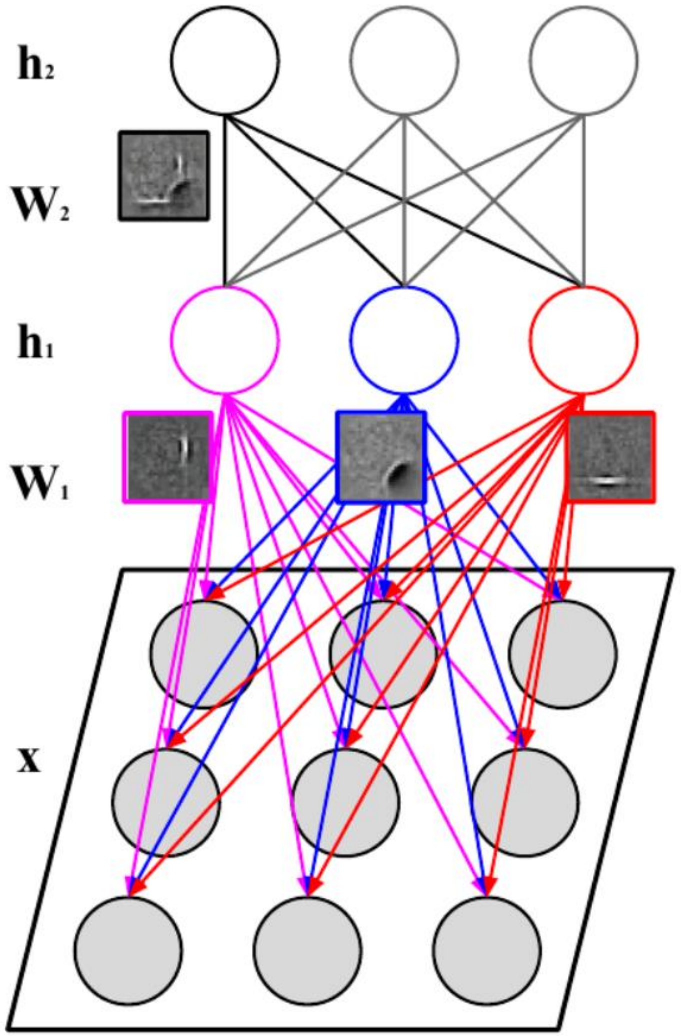
$$\frac{\partial \log p(x_n|\theta)}{\partial \theta} = -\frac{\partial E(x_n, h_0|\theta)}{\partial \theta} - \frac{\partial E(x_k, h_k|\theta)}{\partial \theta}$$



Hinton, *Training Products of Experts by Minimizing Contrastive Divergence*, Neural Computation, 2002

Deep Belief Network

- RBMs are just one layer
- Use RBM as a building block
- Stack multiple RBMs one on top of the other
$$p(x, h_1, h_2) = p(x|h_1) \cdot p(h_1|h_2)$$
- Deep Belief Networks (DBN) are directed models
  - The layers are densely connected and have a single forward flow
  - This is because the RBM is directional, $p(x_i|h, \theta) = \sigma(W_{.i}x + c_i)$, namely the input argument has only variable only from below

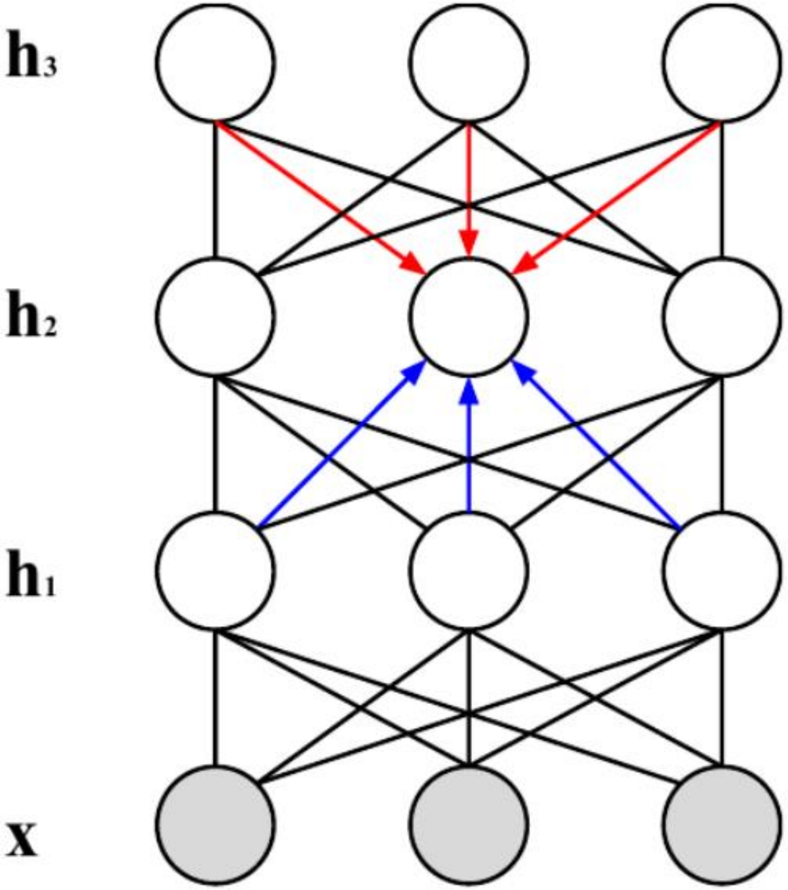


Deep Boltzmann Machines

- Stacking layers again, but now with connection from the above and from the below layers
- Since it's a Boltzmann machine, we need an energy function

$$E(x, h_1, h_2|\theta) = x^T W_1 h_1 + h_1^T W_2 h_2 + h_2^T W_3 h_3$$

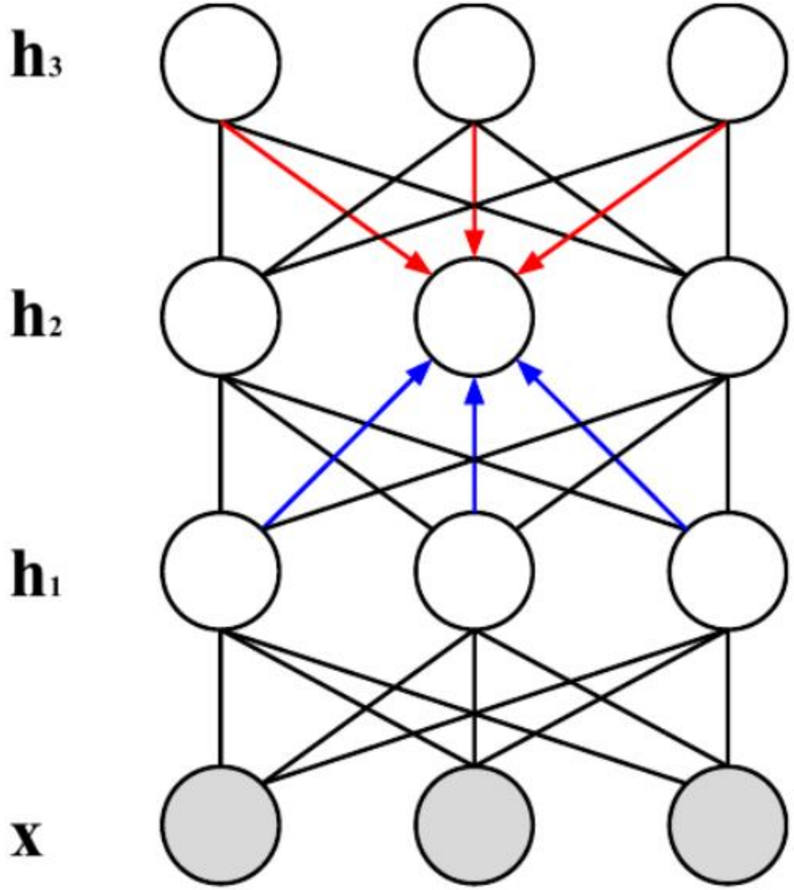
$$p(h_2^k|h_1, h_3) = \sigma(\sum_j W_1^{jk} h_1^j + \sum_l W_3^{kl} h_3^k)$$



Deep Boltzmann Machines

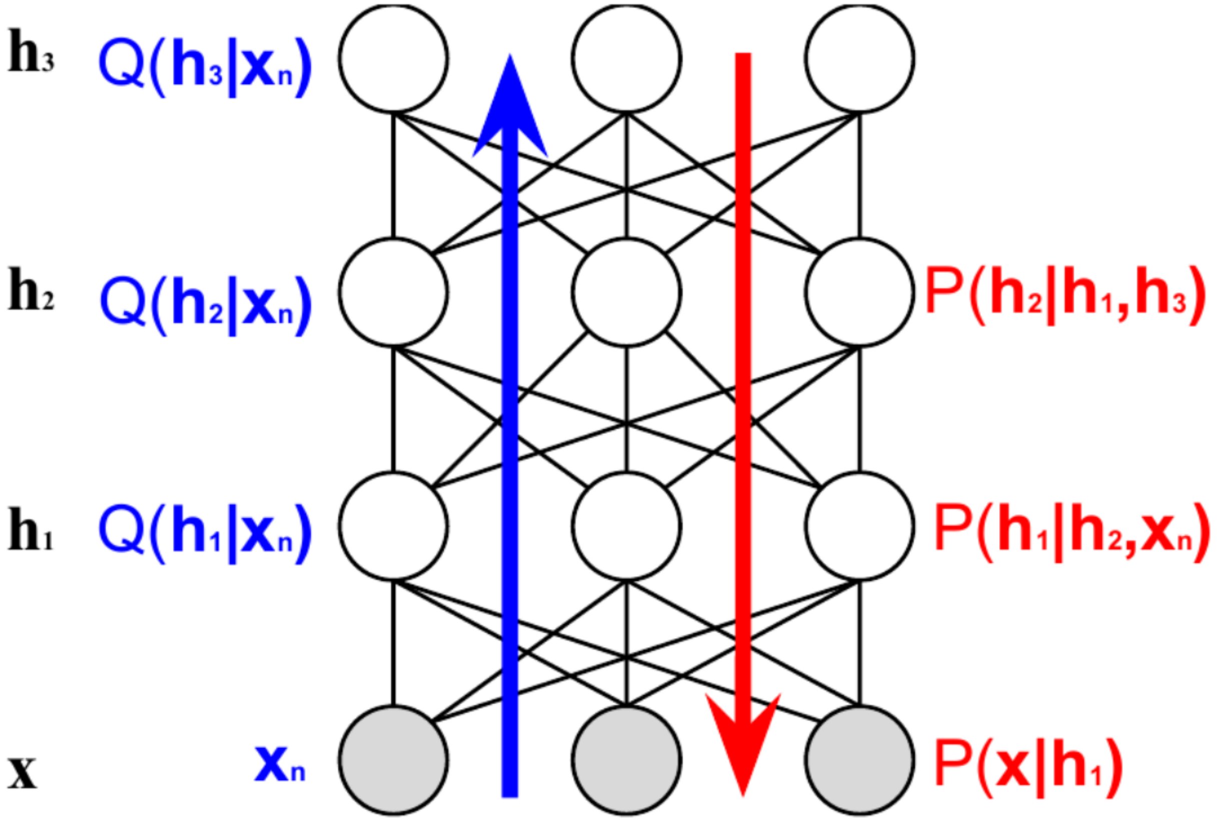
- Schematically similar to Deep Belief Networks
- But, Deep Boltzmann Machines (DBM) are undirected models
  - Belong to the Markov Random Field family
- So, two types of relationships: bottom-up and up-bottom

$$p(h_2^k | h_1, h_3) = \sigma(\sum_j W_1^{jk} h_1^j + \sum_l W_3^{kl} h_3^k)$$

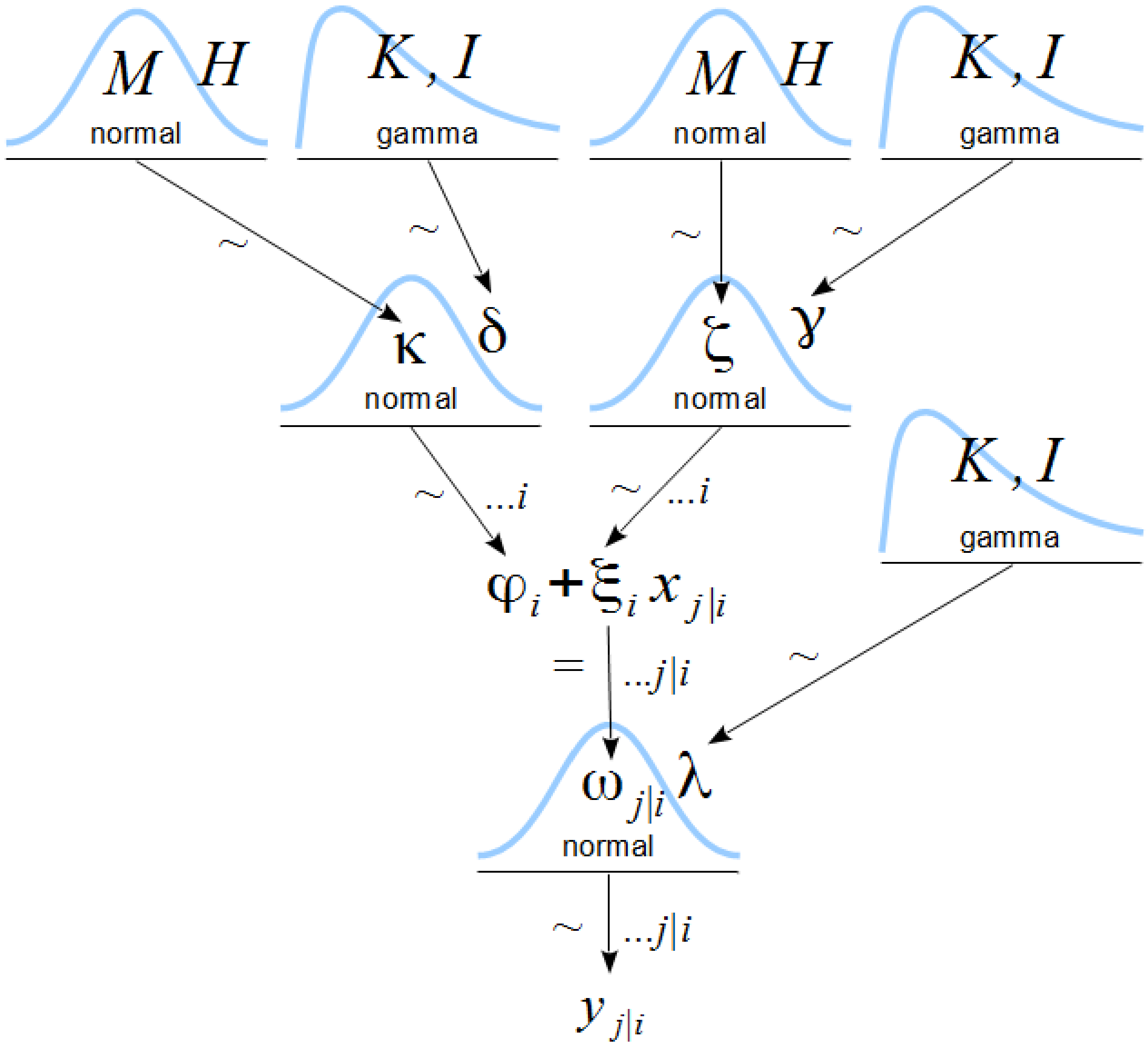


Training Deep Boltzmann Machines

- Computing gradients is intractable
- Instead, variational methods (mean-field) or sampling methods are used



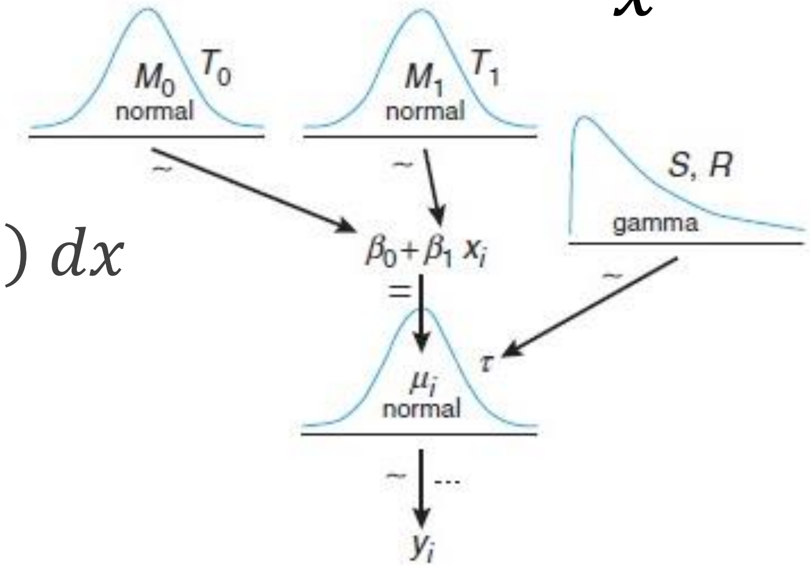
Variational Inference



Some (Bayesian) Terminology

- Observed variables x
- Latent variables θ
  - Both unobservable model parameters w and unobservable model activations z
  - $\theta = \{w, z\}$
- Joint probability density function (pdf): $p(x, \theta)$
- Marginal pdf: $p(x) = \int_\theta p(x, \theta)\, d\theta$
- Prior pdf → marginal over input: $p(\theta) = \int_x p(x, \theta)\, dx$
  - Usually a user defined pdf
- Posterior pdf: $p(\theta|x)$
- Likelihood pdf: $p(x|\theta)$

x



Bayesian Terminology

- Posterior pdf

$$
\begin{aligned}
p(\theta|x) &= \frac{p(x,\theta)}{p(x)} && \leftarrow \text{Conditional probability} \\
&= \frac{p(x|\theta)\, p(\theta)}{p(x)} && \leftarrow \text{Bayes Rule} \\
&= \frac{p(x|\theta)\, p(\theta)}{\int_{\theta'} p(x,\theta')\; d\theta'} && \leftarrow \text{Marginal probability} \\
&\propto p(x|\theta)\, p(\theta) && \leftarrow p(x\,) \text{ is constant}
\end{aligned}
$$

- Posterior Predictive pdf

$$p(y_{new}|y) = \int_{\theta} p(y_{new}|\theta)\, p(\theta|y)\, d\theta$$

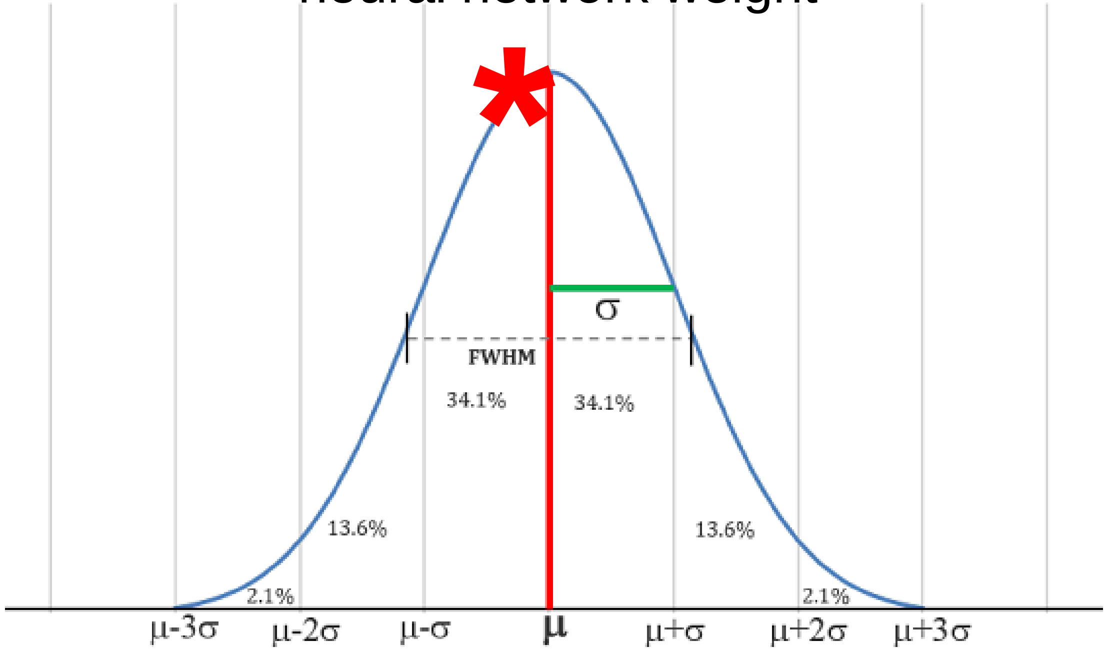
Bayesian Terminology

- Conjugate priors
  - when posterior and prior belong to the same family, so the joint pdf is easy to compute
- Point estimate approximations of latent variables
  - instead of computing a distribution over all possible values for the variable
  - compute one point only
  - e.g. the most likely (maximum likelihood or max a posteriori estimate)

$$\theta^* = \arg_\theta \max p(x|\theta)p(\theta) \ (MAP)$$
$$\theta^* = \arg_\theta \max p(x|\theta) \qquad (MLE)$$

  - Quite good when the posterior distribution is peaky (low variance)

Point estimate of your neural network weight



Bayesian Modelling

- Estimate the posterior density $p(\theta|x)$ for your training data x
- To do so, need to define the prior $p(\theta)$ and likelihood $p(x|\theta)$ distributions
- Once the $p(\theta|x)$ density is estimated, Bayesian Inference is possible
  - $p(\theta|x)$ is a (density) function, not just a single number (point estimate)

- But how to estimate the posterior density?
  - Markov Chain Monte Carlo (MCMC) → Simulation-like estimation
  - Variational Inference → Turn estimation to optimization

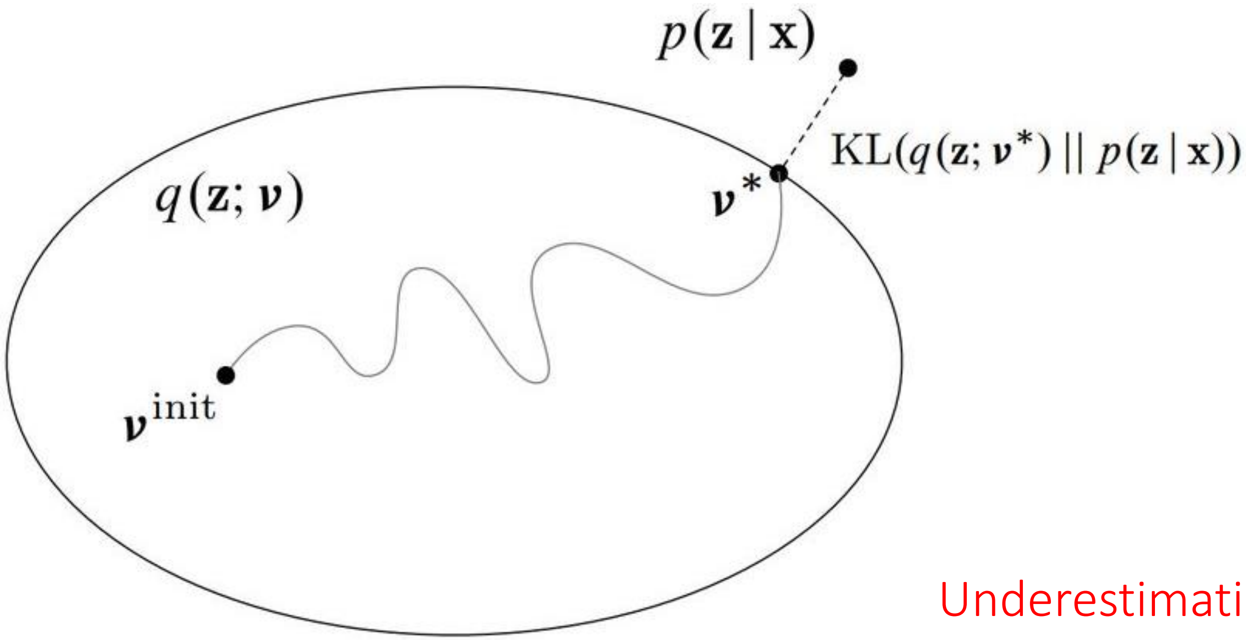
Variational Inference

- Estimating the true posterior $p(\theta|x)$is not always possible
  - especially for complicated models like neural networks
- Variational Inference assumes another function $q(\theta|\varphi)$ with which we want to approximate the true posterior $p(\theta|x)$
  - $q(\theta|\varphi)$ is the approximate posterior
  - Note that the approximate posterior does not depend on the observable variables x
- We approximate by minimizing the **reverse** KL-divergence w.r.t. φ

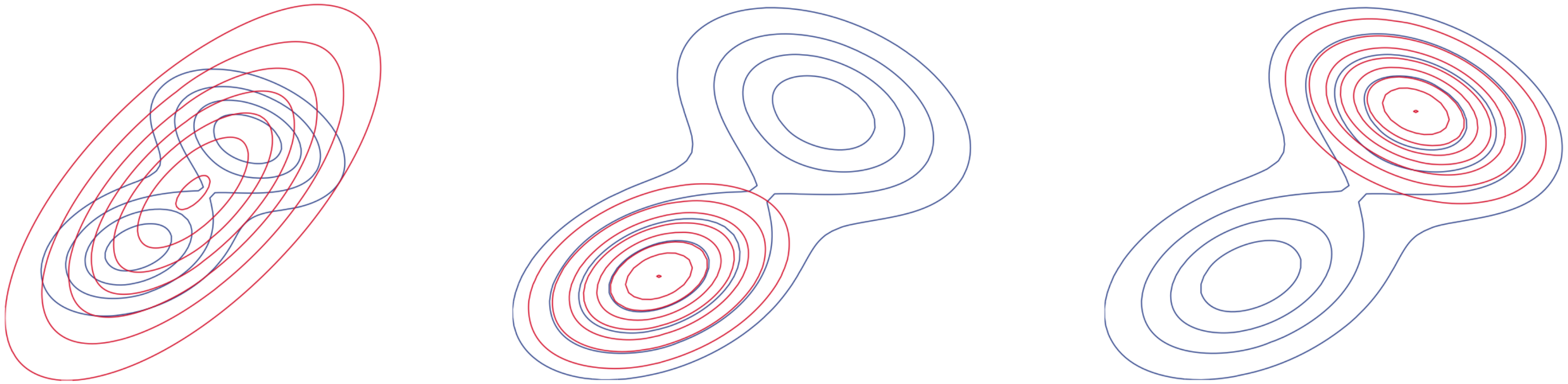
$$\varphi^* = \arg\min_{\varphi} KL(q(\theta|\varphi)||p(\theta|x))$$

- Turn inference into optimization

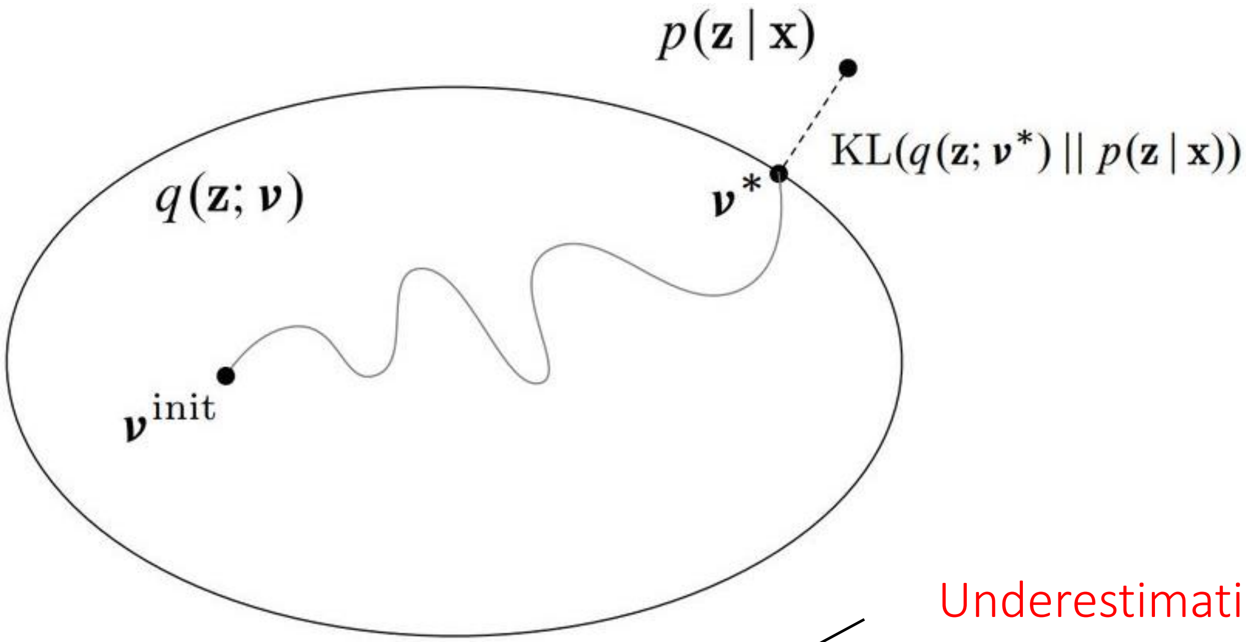
Variational Inference (graphically)



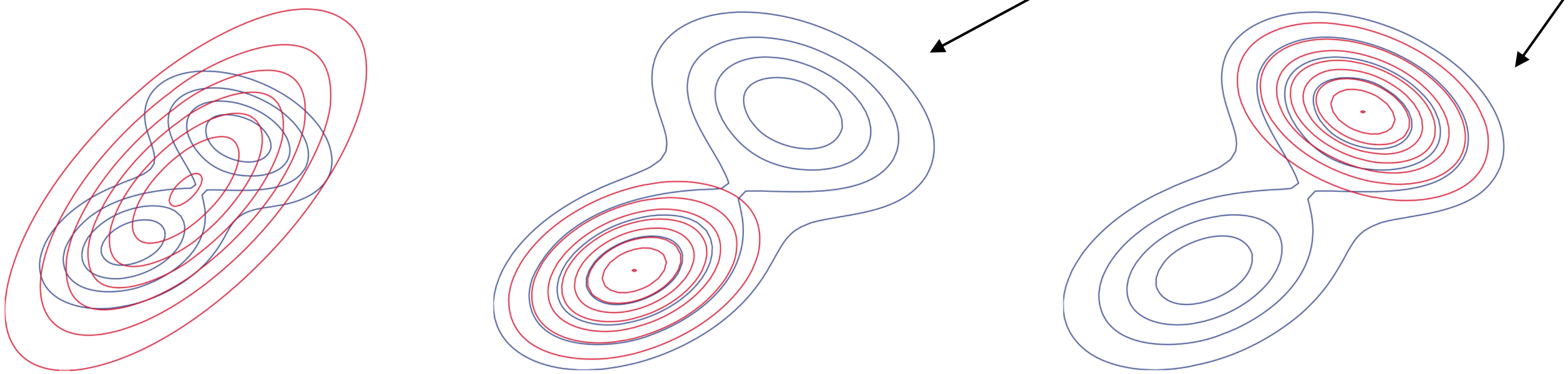
Underestimating variance. Why?



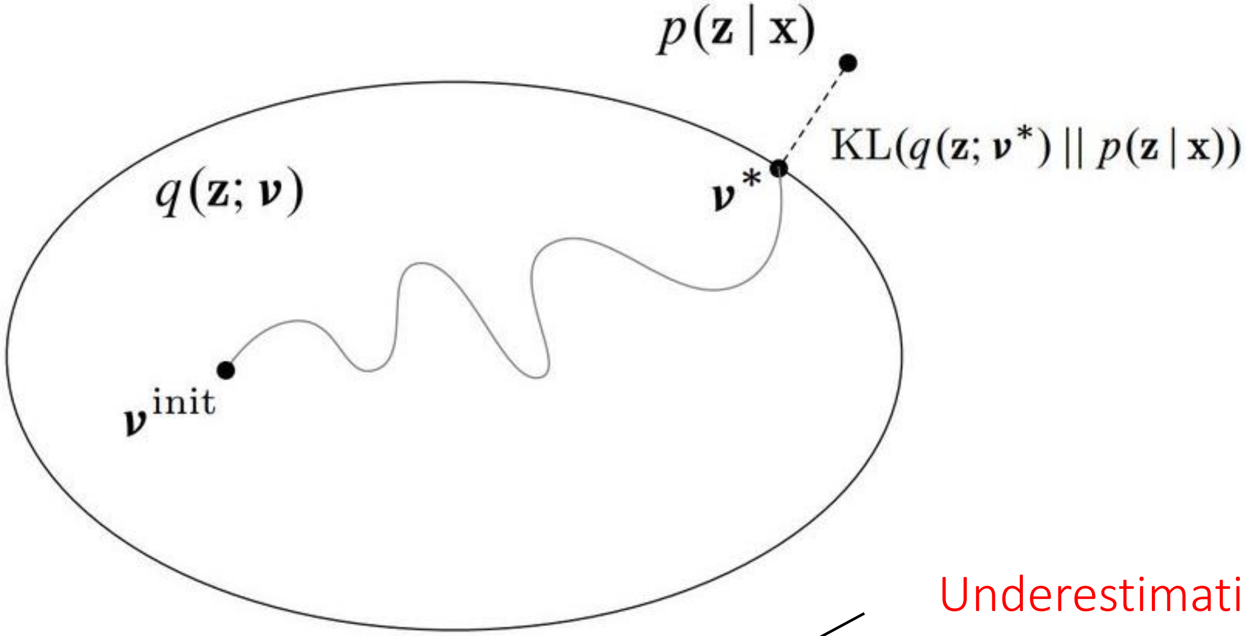
Variational Inference (graphically)



Underestimating variance. Why?

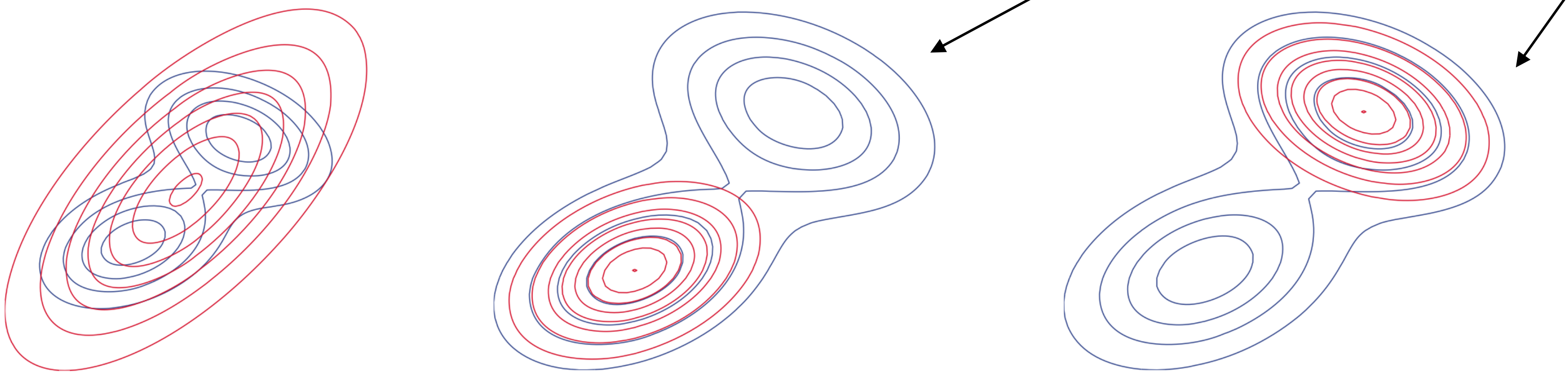


Variational Inference (graphically)

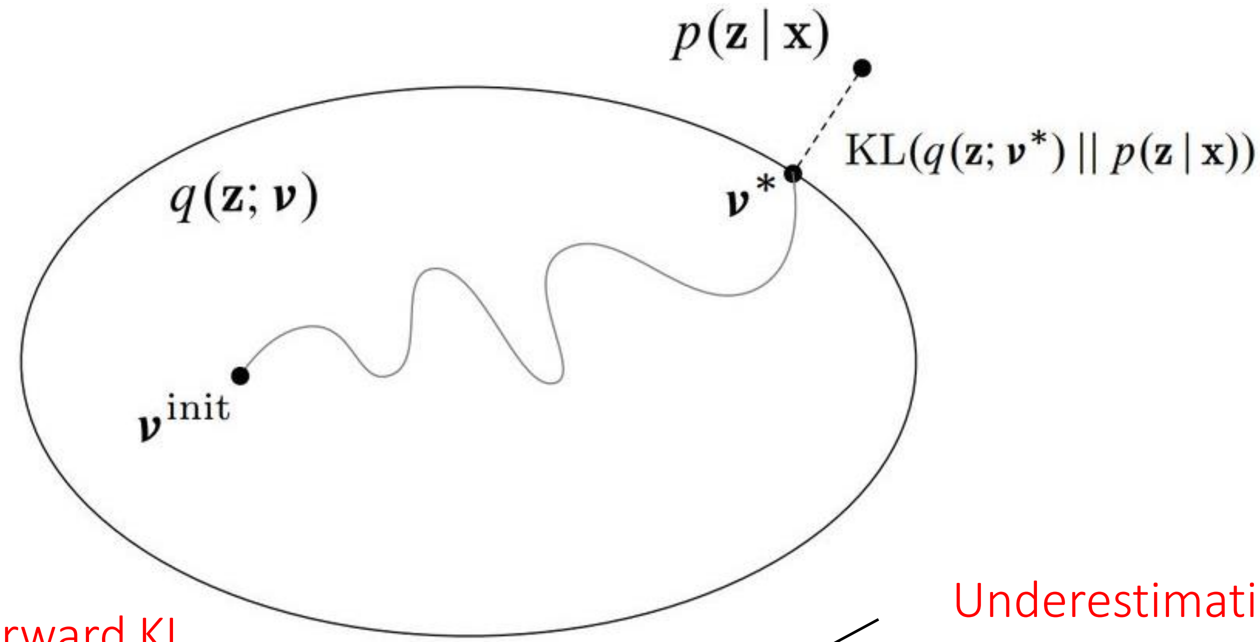


Underestimating variance. Why?

How to overestimate variance?

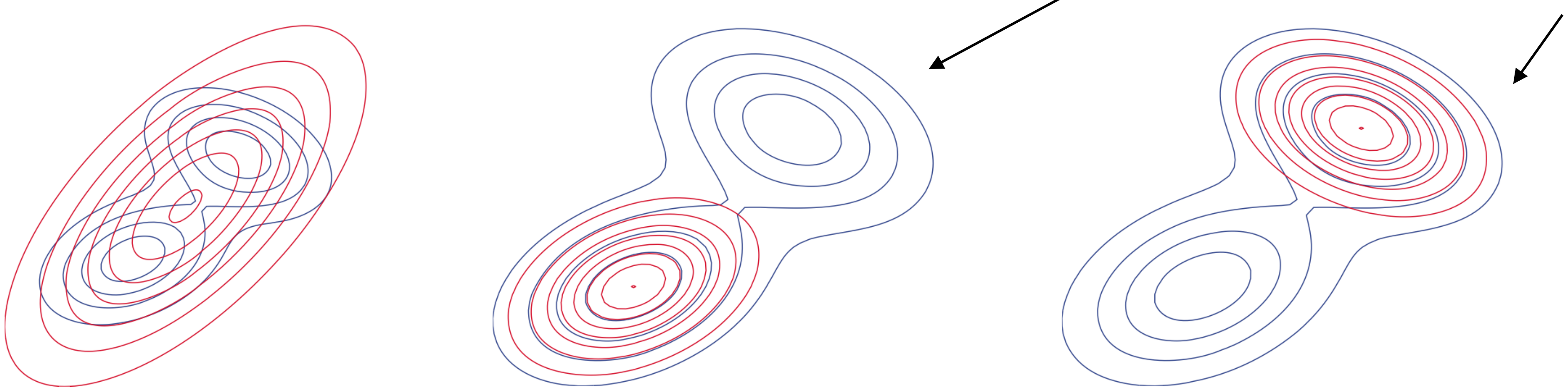


Variational Inference (graphically)



How to overestimate variance? Forward KL

Underestimating variance. Why?



Variational Inference - Evidence Lower Bound (ELBO)

- Given latent variables θ and the approximate posterior

$$q_{\varphi}(\theta) = q(\theta|\varphi)$$

- What about the log marginal $\log p(x)$?

Variational Inference - Evidence Lower Bound (ELBO)

- Given latent variables θ and the approximate posterior

$$q_\varphi(\theta) = q(\theta|\varphi)$$

- We want to maximize the marginal $p(x)$ (or the log marginal $\log p(x)$

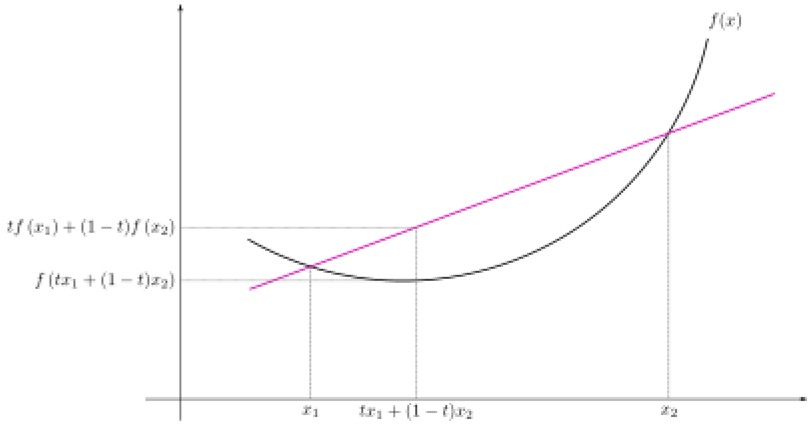
$$\log p(x) \geq \mathbb{E}_{q_\varphi(\theta)}\left[\log \frac{p(x,\theta)}{q_\varphi(\theta)}\right]$$

Evidence Lower Bound (ELBO): Derivations

Evidence Lower Bound (ELBO): Derivations

- Given latent variables θ and the approximate posterior
$$q_\varphi(\theta) = q(\theta|\varphi)$$
- The log marginal is

$$\log p(x) = \log \int_\theta p(x,\theta)\, d\theta$$
$$= \log \int_\theta p(x,\theta) \frac{q_\varphi(\theta)}{q_\varphi(\theta)}\, d\theta$$
$$= \log \mathbb{E}_{q_\varphi(\theta)}\left[\frac{p(x,\theta)}{q_\varphi(\theta)}\right]$$
$$\geq \mathbb{E}_{q_\varphi(\theta)}\left[\log \frac{p(x,\theta)}{q_\varphi(\theta)}\right]$$



[Jensen Inequality]

- $\varphi\big(\mathbb{E}([\mathrm{x}])\big) \leq \mathbb{E}[\varphi(x)]$

for convex φ
- $\log$ is convave

ELBO: A second derivation

$$
\begin{aligned}
KL\left[q(Z)\|p(Z|X)\right] &= \int_Z q(Z)\log\frac{q(Z)}{p(Z|X)} \\
&= -\int_Z q(Z)\log\frac{p(Z|X)}{q(Z)} \\
&= -\left(\int_Z q(Z)\log\frac{p(X,Z)}{q(Z)} - \int_Z q(Z)\log p(X)\right) \\
&= -\int_Z q(Z)\log\frac{p(X,Z)}{q(Z)} + \log p(X)\int_Z q(Z) \\
&= -L + \log p(X)
\end{aligned}
$$

ELBO: Formulation 1

$$
\begin{aligned}
&\geq \mathbb{E}_{q_\varphi(\theta)}\left[\log \frac{p(x,\theta)}{q_\varphi(\theta)}\right] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)] + \mathbb{E}_{q_\varphi(\theta)}[\log p(\theta)] - \mathbb{E}_{q_\varphi(\theta)}[\log q_\varphi(\theta)] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)] - \mathrm{KL}(q_\varphi(\theta)||\mathrm{p}(\theta)) \\
&= \mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})
\end{aligned}
$$

- Maximize reconstruction accuracy $\mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)]$
- While minimizing the KL distance between the prior $\mathrm{p}(\theta)$ and the approximate posterior $q_\varphi(\theta)$

ELBO: Formulation 2

$$
\begin{aligned}
&\geq \mathbb{E}_{q_\varphi(\theta)}\left[\log \frac{p(x,\theta)}{q_\varphi(\theta)}\right] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x,\theta)] - \mathbb{E}_{q_\varphi(\theta)}[\log q_\varphi(\theta)] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x,\theta)] + \mathrm{H}(\theta) \\
&= \mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})
\end{aligned}
$$

- Maximize something like negative Boltzmann energy $\mathbb{E}_{q_\varphi(\theta)}[\log p(x,\theta)]$
- While maximizing the entropy the approximate posterior $q_\varphi(\theta)$
  - Avoid collapsing latents θ to a single value (like for MAP estimates)

ELBO vs. Marginal

- It is easy to see that the ELBO is directly related to the marginal

$$\log p(x) = \mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) + KL(q_{\varphi}(\theta)||p(\theta|x))$$

- You can also see $\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})$ as Variational Free Energy

ELBO vs. Marginal: Derivations

- It is easy to see that the ELBO is directly related to the marginal

$$\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) =$$

ELBO vs. Marginal: Derivations

- It is easy to see that the ELBO is directly related to the marginal

$$
\begin{aligned}
&\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) = \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x,\theta)] - \mathbb{E}_{q_\varphi(\theta)}\left[\log q_\varphi(\theta)\right] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(\theta|x)] + \mathbb{E}_{q_\varphi(\theta)}[\log p(x)] - \mathbb{E}_{q_\varphi(\theta)}\left[\log q_\varphi(\theta)\right] \\
&= \mathbb{E}_{q_\varphi(\theta)}[\log p(x)] - KL(q_\varphi(\theta)||p(\theta|x)) \\
&= \log p(x) - KL(q_\varphi(\theta)||p(\theta|x)) \\
&\Rightarrow \\
&\log p(x) = \mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) + KL(q_\varphi(\theta)||p(\theta|x))
\end{aligned}
$$

$\log p(x)$ does not depend on $q_\varphi(\theta)$

$\mathbb{E}_{q_\varphi(\theta)}[1]$=1

- You can also see $\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})$ as Variational Free Energy

ELBO interpretations

- $\log p(x) = \mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) + KL(q_\varphi(\theta)||p(\theta|x))$
- The log-likelihood $\log p(x)$ constant → does not depend on any parameter
- Also, $\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x}) > 0$ and $KL(q_\varphi(\theta)||p(\theta|x)) > 0$

1. The higher the Variational Lower Bound $\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})$, the smaller the difference between the approximate posterior $q_\varphi(\theta)$ and the true posterior $p(\theta|x)$ → better latent representation
2. The Variational Lower Bound $\mathrm{ELBO}_{\theta,\varphi}(\mathrm{x})$ approaches the log-likelihood → better density model

Amortized Inference

- The variational distribution $q(\theta|\varphi)$ does not depend directly on data
  - Only indirectly, via minimizing its distance to the true posterior $KL(q(\theta|\varphi)||p(\theta|x))$
- So, with $q(\theta|\varphi)$ we have a major optimization problem
- The approximate posterior must approximate the whole dataset $x = [x_1, x_2, \dots, x_N]$ jointly
- Different neural network weights for each data point x_i

Amortized Inference

- Better share weights and “amortize” optimization between individual data points

$$q(\theta|\varphi) = q_{\varphi}(\theta|x)$$

- Predict model parameters θ using a φ-parameterized model of the input x
- Use amortization for data-dependent parameters that depend on data
  - E.g., the latent activations that are the output of a neural network layer: $\mathrm{z} \sim q_{\varphi}(z|x)$

Amortized Inference (Intuitively)

- The original view on Variational Inference is that $q(\theta|\varphi)$ describes the approximate posterior of the dataset as a whole
- Imagine you don't want to make a practical model that returns latent activations for a specific input
- Instead, you want to optimally approximate the true posterior of the unknown weights with an model with latent parameters
- It doesn't matter if these parameters are "latent activations" z or "model variables" w

Variational Autoencoders

- Let's rewrite the ELBO a bit more explicitly

$$\mathrm{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)] - \mathrm{KL}(q_\varphi(\theta)||\mathrm{p}(\theta))$$
$$= \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- $p_\theta(x|z)$ instead of $p(x|\theta)$

- I.e., the likelihood model $p_\theta(x|z)$ has weights parameterized by θ
- Conditioned on latent model activations parameterized by z

Variational Autoencoders

- Let's rewrite the ELBO a bit more explicitly

$$\mathrm{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)] - \mathrm{KL}(q_\varphi(\theta)||\mathrm{p}(\theta))$$
$$= \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- $\mathrm{p}_\lambda(\mathrm{z})$ instead of $\mathrm{p}(\theta)$

- I.e., a λ-parameterized prior only on the latent activations $\mathbf{z}$
- <u>Not on model weights</u>

Variational Autoencoders

- Let's rewrite the ELBO a bit more explicitly

$$\mathrm{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(\theta)}[\log p(x|\theta)] - \mathrm{KL}(q_\varphi(\theta)||\mathrm{p}(\theta))$$
$$= \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- $q_\varphi(z|x)$ instead of $q(\theta|\varphi)$

- The model $q_\varphi(z|x)$ approximates the posterior density of the latents z
- The model weights are parameterized by φ

Variational Autoencoders

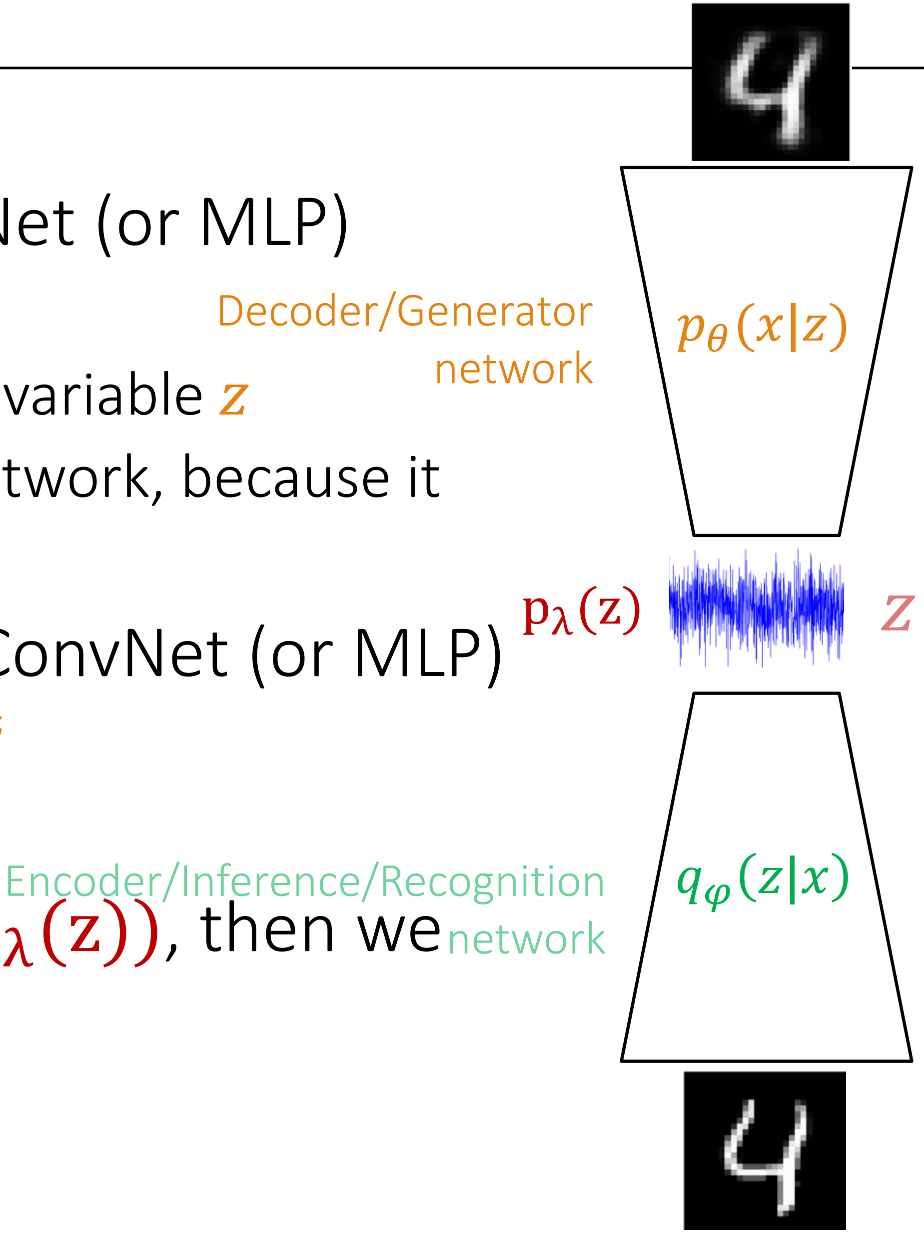
- $\mathrm{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_{\varphi}(z|x)}[\log p_{\theta}(x|z)] - \mathrm{KL}(q_{\varphi}(z|x)||\mathrm{p}_{\lambda}(\mathrm{z}))$
- How to model $p_{\theta}(x|z)$ and $q_{\varphi}(z|x)$?

Variational Autoencoders

- $\mathrm{ELBO}_{\theta,\varphi}(x) = \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$
- How to model $p_\theta(x|z)$ and $q_\varphi(z|x)$?
- What about modelling them as neural networks

Variational Autoencoders

- The approximate posterior $q_\varphi(z|x)$ is a CovnNet (or MLP)
  - Input x is an image
  - Given input the output is a feature map from a latent variable z
  - Also known as encoder or inference or recognition network, because it infers/recognizes the latent codes
- The likelihood density $p_\theta(x|z)$ is an inverted ConvNet (or MLP)
  - Given the latent z as input, it reconstructs the input $\tilde{x}$
  - Also known as decoder or generator network
- If we ignore the distribution of the latents z, $\mathrm{p}_\lambda(\mathrm{z})$), then we get the Vanilla Autoencoder



Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- How to we optimize the ELBO?

Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(Z|x)}[\log p_\theta(x|Z)] - \mathrm{KL}(q_\varphi(Z|x)||\mathrm{p}_\lambda(\mathrm{Z}))$$
$$= \int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz - \int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$$

- Forward propagation → compute the two terms
- The first term is an integral (expectation) that we cannot solve analytically. So, we need to sample from the pdf instead
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically

Training Variational Autoencoders

- Maximize the Evidence Lower Bound (ELBO)
  - Or minimize the negative ELBO

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{q_\varphi(Z|x)}[\log p_\theta(x|Z)] - \mathrm{KL}(q_\varphi(Z|x)||\mathrm{p}_\lambda(\mathrm{Z}))$$
$$= \int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz - \int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$$

- Forward propagation → compute the two terms
- The first term is an integral (expectation) that we cannot solve analytically.
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically
- So, we need to sample from the pdf instead
- VAE is a stochastic model
- The second term is the KL divergence between two distributions that we know

Training Variational Autoencoders

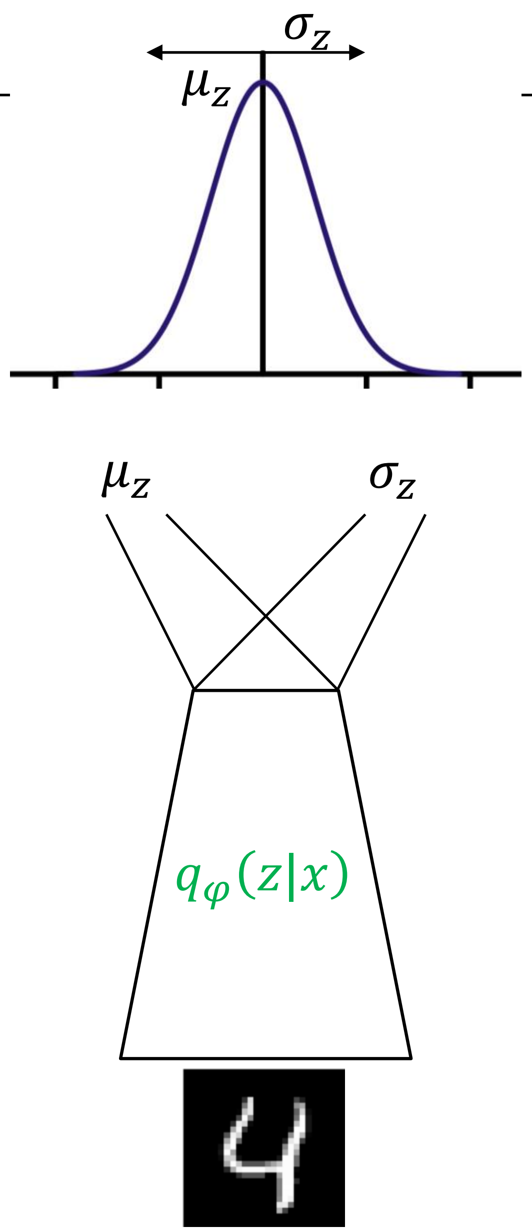
- $\int_z q_\varphi(z|x) \log p_\theta(x|z)\, dz$
- The first term is an integral (expectation) that we cannot solve analytically.
  - When $p_\theta(x|z)$ contains even a few nonlinearities, like in a neural network, the integral is hard to compute analytically
- As we cannot compute analytically, we sample from the pdf instead
  - Using the density $q_\varphi(z|x)$ to draw samples
  - Usually one sample is enough → stochasticity reduces overfitting
- VAE is a stochastic model
- The second term is the KL divergence between two distributions that we know

Training Variational Autoencoders

- $\int_z q_\varphi(z|x) \log \frac{q_\varphi(z|x)}{p_\lambda(z)} dz$
- The second term is the KL divergence between two distributions that we know
- E.g., compute the KL divergence between a centered $N(0, 1)$ and a non-centered $N(\mu, \sigma)$ gaussian

Typical VAE

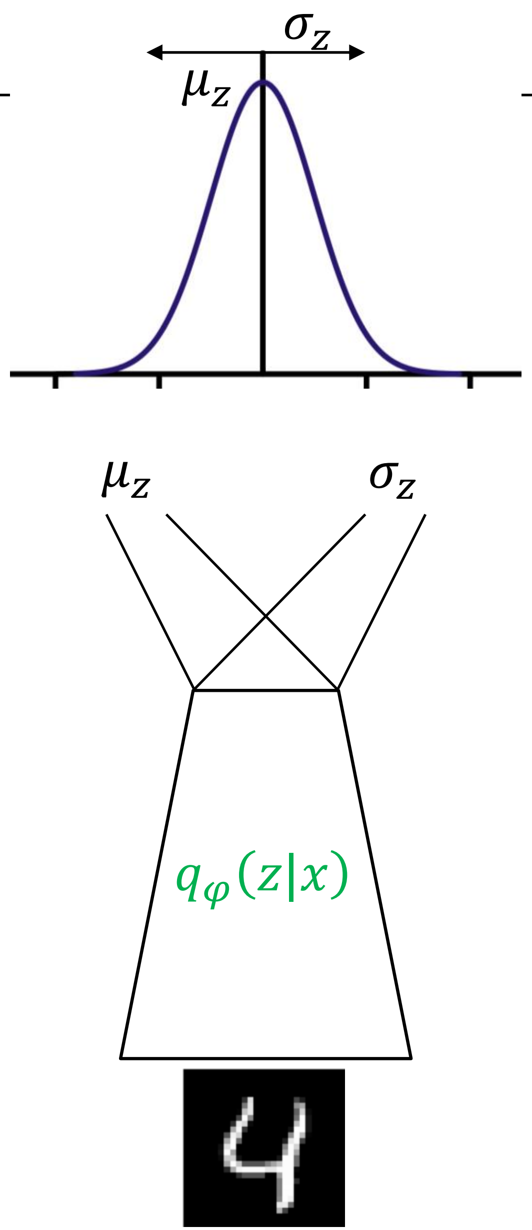
- We set the prior $p_\lambda(z)$ to be the unit Gaussian
$$p(z) \sim N(0,1)$$
- We set the likelihood to be a Bernoulli for binary data
$$p(x|z) \sim Bernoulli(\pi)$$
- We set $q_\varphi(z|x)$ to be a neural network (MLP, ConvNet), which maps an input $\mathbf{x}$ to the Gaussian distribution, specifically it's mean and variance
  - $\mu_z, \sigma_z \sim q_\varphi(z|x)$
  - The neural network has two outputs, one is the mean μ_x and the other the σ_x, which corresponds to the covariance of the Gaussian



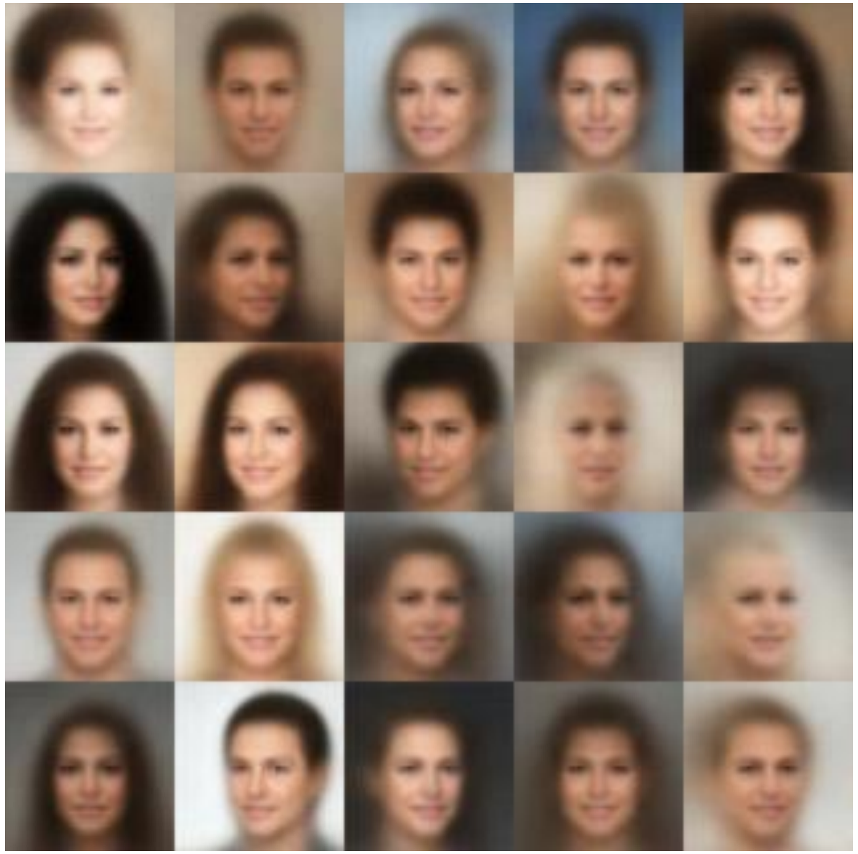
Typical VAE

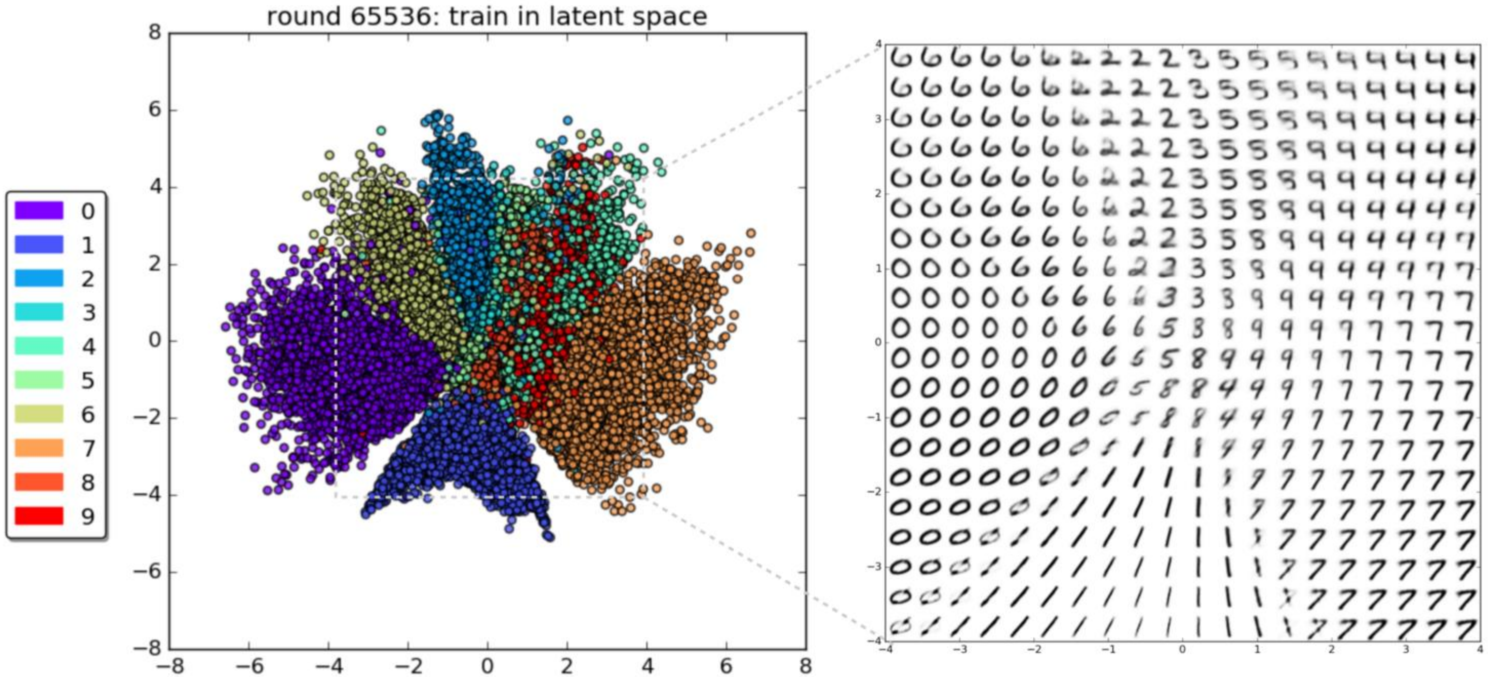
- We set $p_\theta(\mathrm{x}|\mathrm{z})$ to be an inverse neural network, which maps $\mathbf{Z}$ to the Bernoulli distribution if our outputs binary (e.g. Binary MNIST)

- Good exercise: Derive the ELBO for the standard VAE



VAE: Interpolation in the latent space





Forward propagation in VAE

- Sample z from the approximate posterior density $z \sim q_{\varphi}(Z|x)$
  - As q_{φ} is a neural network that outputs values from a specific and known parametric pdf, e.g. a Gaussian, sampling from it is rather easy
  - Often even a single draw is enough
- Second, compute the $\log p_{\theta}(x|Z)$
  - As p_{θ} is a a neural network that outputs values from a specific and known parametric pdf, e.g. a Bernoulli for white/black pixels, computing the log-prob is easy
- Computing the ELBO is rather straightforward in the standard case
- How should we optimize the ELBO?

Forward propagation in VAE

- Sample z from the approximate posterior density $z \sim q_\varphi(Z|x)$
  - As q_φ is a neural network that outputs values from a specific and known parametric pdf, e.g. a Gaussian, sampling from it is rather easy
  - Often even a single draw is enough
- Second, compute the $\log p_\theta(x|Z)$
  - As p_θ is a a neural network that outputs values from a specific and known parametric pdf, e.g. a Bernoulli for white/black pixels, computing the log-prob is easy
- Computing the ELBO is rather straightforward in the standard case
- How should we optimize the ELBO? Backpropagation?

Backward propagation in VAE

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z\sim q_{\varphi}(Z|x)}[\log p_{\theta}(x|z)] - \mathrm{KL}(q_{\varphi}(Z|x)||\mathrm{p}_{\lambda}(\mathrm{Z}))$$

- We must take the gradients with respect to the trainable parameters
- The generator network parameters θ
- The inference network/approximate posterior parameters φ

Monte Carlo Integration

- Let's try to compute the following integral

$$\mathbb{E}(f) = \int_x p(x) f(x)$$

  where $p(x)$ is a probability density function for x
  - Often complex if $p(x)$ and $f(x)$ is slightly complicated
- Instead, we can approximate the integral as a summation

$$\mathbb{E}(f) = \int_x p(x) f(x) \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i), x_i \sim p(x) = \hat{f}$$

- The estimator is unbiased: $\mathbb{E}(f) = \mathbb{E}(\hat{f})$ and its variance

$$Var(\hat{f}) = \frac{1}{N} \mathbb{E}[(f - \mathbb{E}(\hat{f}))]$$

- So, if we have an easy to sample from probability density function in the integral we can do Monte Carlo integration to approximate the integral

Gradients w.r.t. the generator parameters θ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta,\varphi)=\mathbb{E}_{z\sim q_{\varphi}(Z|x)}[\log p_{\theta}(x|z)]-\mathrm{KL}(q_{\varphi}(Z|x)||\mathrm{p}_{\lambda}(\mathrm{Z}))$$

Gradients w.r.t. the generator parameters θ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z\sim q_{\varphi}(z|x)}[\log p_{\theta}(x|z)] - \mathrm{KL}(q_{\varphi}(z|x)||\mathrm{p}_{\lambda}(\mathrm{z}))$$

with respect to θ and φ

- $\nabla_{\theta}\mathcal{L} = \mathbb{E}_{z\sim q_{\varphi}(z|x)}[\nabla_{\theta} \log p_{\theta}(x|z)]$
- The expectation and sampling in $\mathbb{E}_{z\sim q_{\varphi}(z|x)}$ do not depend on θ
- Also, the KL does not depend on θ, so no gradient from over there!
- So, no problem → Just Monte-Carlo integration using samples z drawn from $q_{\varphi}(z|x)$

Gradients w.r.t. the recognition parameters φ

- Backpropagation → compute the gradients of

$$\mathcal{L}(\theta, \varphi) = \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}(q_\varphi(z|x)||\mathrm{p}_\lambda(\mathrm{z}))$$

- Our latent variable z is a Gaussian (in standard VAE) represented by μ_z, σ_z
- So, we can train by sampling randomly from that Gaussian $z\sim N(\mu_Z, \sigma_Z)$
- Problem?
- Sampling $z\sim q_\varphi(z|x)$ is not differentiable
  - And after sampling z, it's a fixed value (not a function), so we cannot backprop
- Not differentiable → no gradients
- No gradients → No backprop → No training!

Solution: Monte Carlo Differentiation?

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta\,(x|z)dz$

$$= \int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta\,(x|z)dz$$

- Problem: Monte Carlo integration not possible anymore
  - No density function inside the integral
  - Only the gradient of a density function
- Similar to Monte Carlo integration, we want to have an expression where there is a density function inside the summation
- That way we can express it again as Monte Carlo integration

Solution: Monte Carlo Differentiation?

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta(x|z) dz$

$$= \int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz$$

- $\int_z \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz =$

$$= \int_z \frac{q_\varphi(z|x)}{q_\varphi(z|x)} \nabla_\varphi [q_\varphi(z|x)] \log p_\theta(x|z) dz$$

NOTE: $\nabla_x \log f(x) = \frac{1}{f(x)} \nabla_x f(x)$

$$= \int_z q_\varphi(z|x) \nabla_\varphi [\log q_\varphi(z|x)] \log p_\theta(x|z) dz$$

$$= \mathbb{E}_{z\sim q_\varphi(z|x)}[\nabla_\varphi [\log q_\varphi(z|x)] \log p_\theta(x|z)]$$

Solution: Monte Carlo Differentiation == REINFORCE

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \mathbb{E}_{z\sim q_\varphi(z|x)}\left[\nabla_\varphi\left[\log q_\varphi(z|x)\right]\log p_\theta(x|z)\right]$
$$= \sum_i \nabla_\varphi\left[\log q_\varphi(z_i|x)\right]\log p_\theta(x|z_i),\, z_i\sim q_\varphi(z|x)$$
- Also known as REINFORCE or score-function estimator
  - $\log p_\theta(x|z)$ is called score function
  - Used to approximate gradients of non-differentiable function
  - Highly popular in Reinforcement Learning, where we also sample from policies
- Problem: Typically high-variance gradients →
- → Slow and not very effective learning

To sum up

- So, our latent variable z is a Gaussian (in the standard VAE) represented by the mean and variance μ_z, σ_z, which are the output of a neural net
- So, we can train by sampling randomly from that Gaussian

$$z \sim N(\mu_Z, \sigma_Z)$$

- Once we have that z, however, it's a fixed value (not a function), so we cannot backprop
- We can use REINFORCE algorithm to compute an approximation to the gradient
  - High-variance gradients → slow and not very effective learning

Solution: Reparameterization trick

- Remember, we have a Gaussian output $z \sim N(\mu_Z, \sigma_Z)$
- For certain pdfs, including the Gaussian, we can rewrite their random variable z as deterministic transformations of an auxiliary and simpler random variable ε

$$z \sim N(\mu, \sigma) \;\Leftrightarrow\; z = \mu + \varepsilon \cdot \sigma, \qquad \varepsilon \sim N(0, 1)$$

- μ, σ are deterministic (not random) values
- Long story short:
- We can model μ, σ by our NN encoder/recognition
- And ε comes externally

What do we gain?

- Change of variables

$$z = g(\varepsilon)$$
$$\mathrm{p}(z)dz = p(\varepsilon)d\varepsilon$$

  - Intuitively, think that the probability mass must be invariant after the transformation

- In our case

$$\varepsilon \sim q(\varepsilon) = \mathrm{N}(0, 1), z = g_\varphi(\varepsilon) = \mu_\varphi + \varepsilon \cdot \sigma_\varphi$$

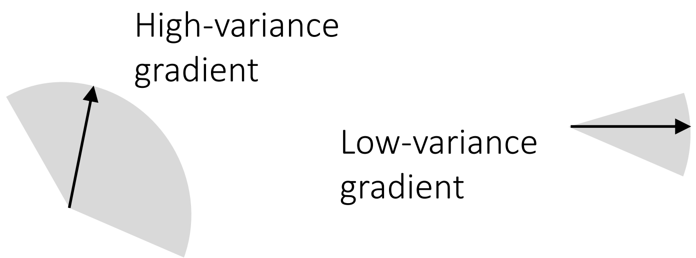
- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] = \nabla_\varphi \int_z q_\varphi(z|x) \log p_\theta\,(x|z)dz$

$$= \nabla_\varphi \int_\varepsilon q(\varepsilon) \log p_\theta\left(x\middle|\mu_\varphi, \sigma_\varphi, \varepsilon\right)d\varepsilon$$
$$= \int_\varepsilon q(\varepsilon)\nabla_\varphi \log p_\theta\left(x\middle|\mu_\varphi, \sigma_\varphi, \varepsilon\right)d\varepsilon$$

- $\nabla_\varphi \mathbb{E}_{z\sim q_\varphi(z|x)}[\log p_\theta(x|z)] \approx \sum_i \nabla_\varphi \log p_\theta\left(x\middle|\mu_\varphi, \sigma_\varphi, \varepsilon_i\right), \varepsilon_i \sim N(0, 1)$
  - The Monte Carlo integration does not depend on the parameter of interest φ anymore

Solution: Reparameterization trick

- Sampling directly from $\varepsilon \sim N(0,1)$ leads to low-variance estimates compared to sampling directly from $z \sim N(\mu_Z, \sigma_Z)$
  - Why low variance? Exercise for the interested reader
- Remember: since we are sampling for z, we are also sampling gradients
  - Stochastic gradient estimator
- More distributions beyond Gaussian possible: Laplace, Student-t, Logistic, Cauchy, Rayleight, Pareto



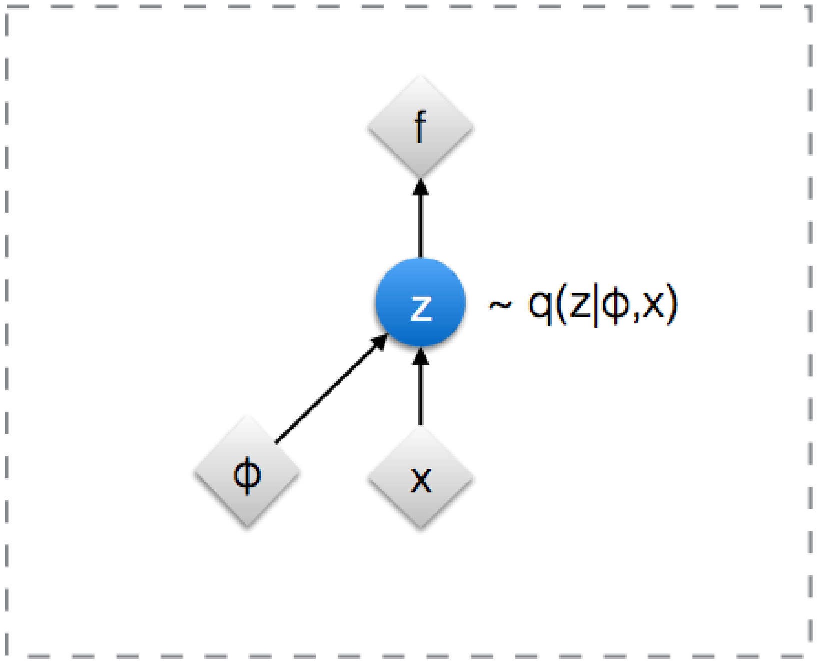
http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/

Once more: what is random in the reparameterization trick?

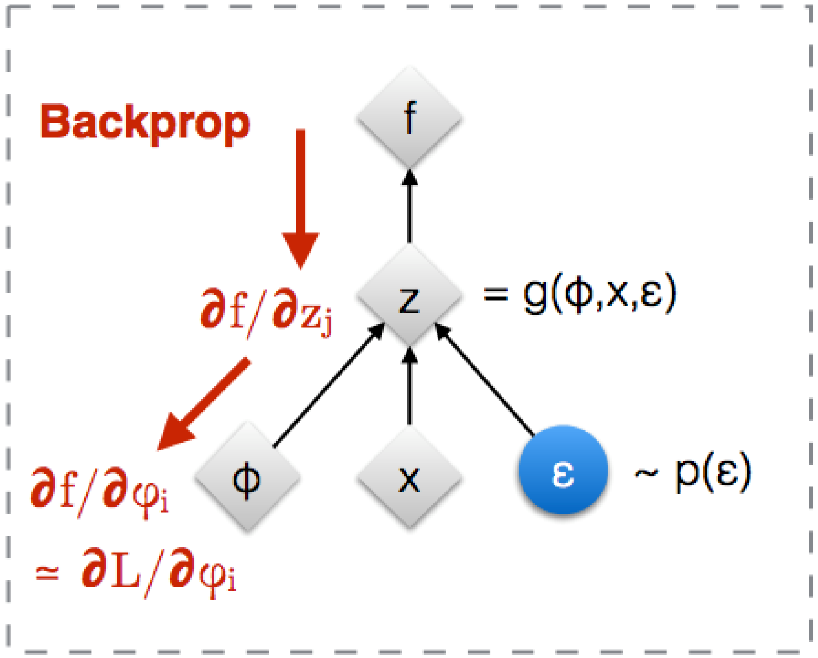
- Again, the latent variable is $z = \mu_\varphi + \varepsilon \cdot \sigma_\varphi$
- μ_φ and σ_φ are deterministic functions (via the neural network encoder)
- ε is a random variable, which comes **externally**
- The z as a result is itself a random variable, because of ε
- However, now the randomness is not associated with the neural network and its parameters that we have to learn
  - The randomness instead comes from the external ε
  - The gradients flow through μ_φ and σ_φ

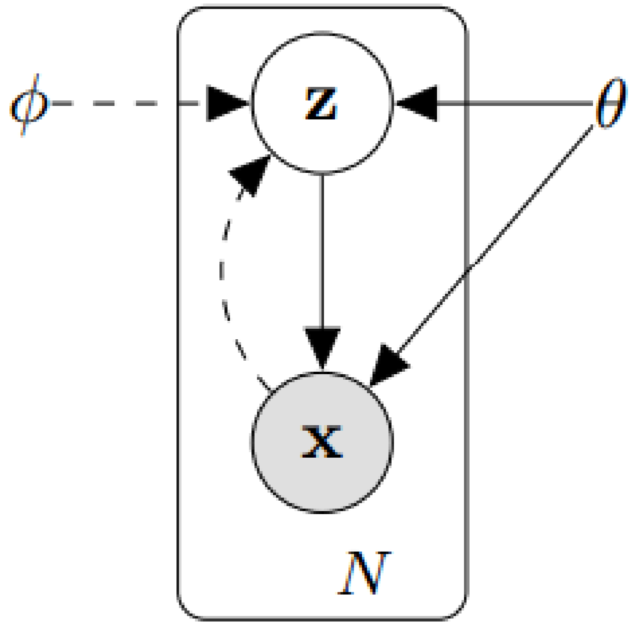
Reparameterization Trick (graphically)

Original form



Reparameterised form



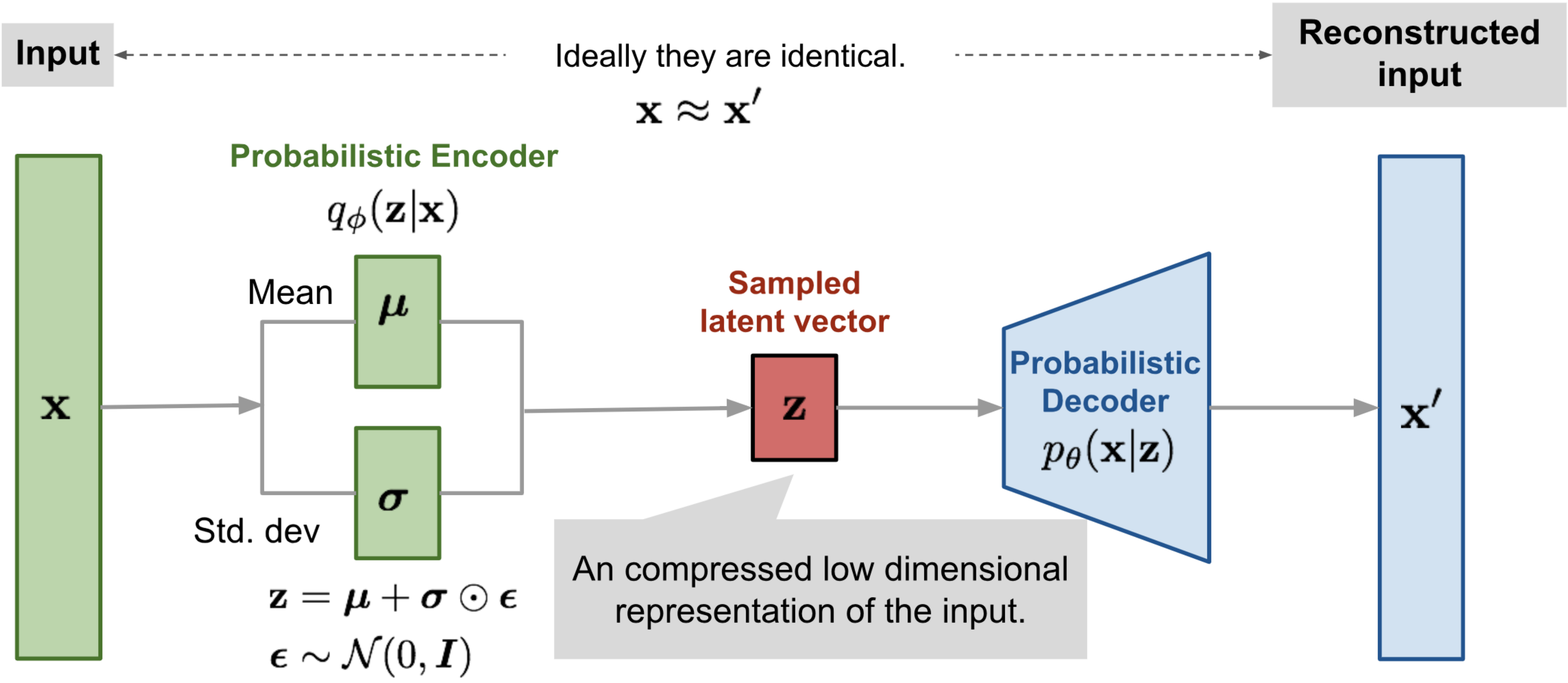


: Deterministic node

: Random node

[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Variational Autoencoders



https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

VAE Training Pseudocode

Data:
  $\mathcal{D}$: Dataset
  $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$: Inference model
  $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$: Generative model
Result:
  $\boldsymbol{\theta}, \boldsymbol{\phi}$: Learned parameters

$(\boldsymbol{\theta}, \boldsymbol{\phi}) \leftarrow$ Initialize parameters
while *SGD not converged* **do**
  $\mathcal{M} \sim \mathcal{D}$ (Random minibatch of data)
  $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$ (Random noise for every datapoint in $\mathcal{M}$)
  Compute $\tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$ and its gradients $\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \tilde{\mathcal{L}}_{\boldsymbol{\theta},\boldsymbol{\phi}}(\mathcal{M}, \boldsymbol{\epsilon})$
  Update $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ using SGD optimizer
end

The ELBO's gradients

VAE for NLP

“ i want to talk to you . ”
“i want to be with you . ”
“i do n’t want to be with you . ”
i do n’t want to be with you .
she did n’t want to be with him .
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Figure 2.D.2: An application of VAEs to interpolation between pairs of sentences, from [Bowman et al., 2015]. The intermediate sentences are grammatically correct, and the topic and syntactic structure are typically locally consistent.

VAE for Image Resynthesis

Smile vector:
mean smiling faces − mean no-smile faces

Latent space arithmetic

Figure 2.D.3: VAEs can be used for image re-synthesis. In this example by White [2016], an original image (left) is modified in a latent space in the direction of a *smile vector*, producing a range of versions of the original, from smiling to sadness. Notice how changing the image along a single vector in latent space, modifies the image in many subtle and less-subtle ways in pixel space.

VAE for designing chemical compounds

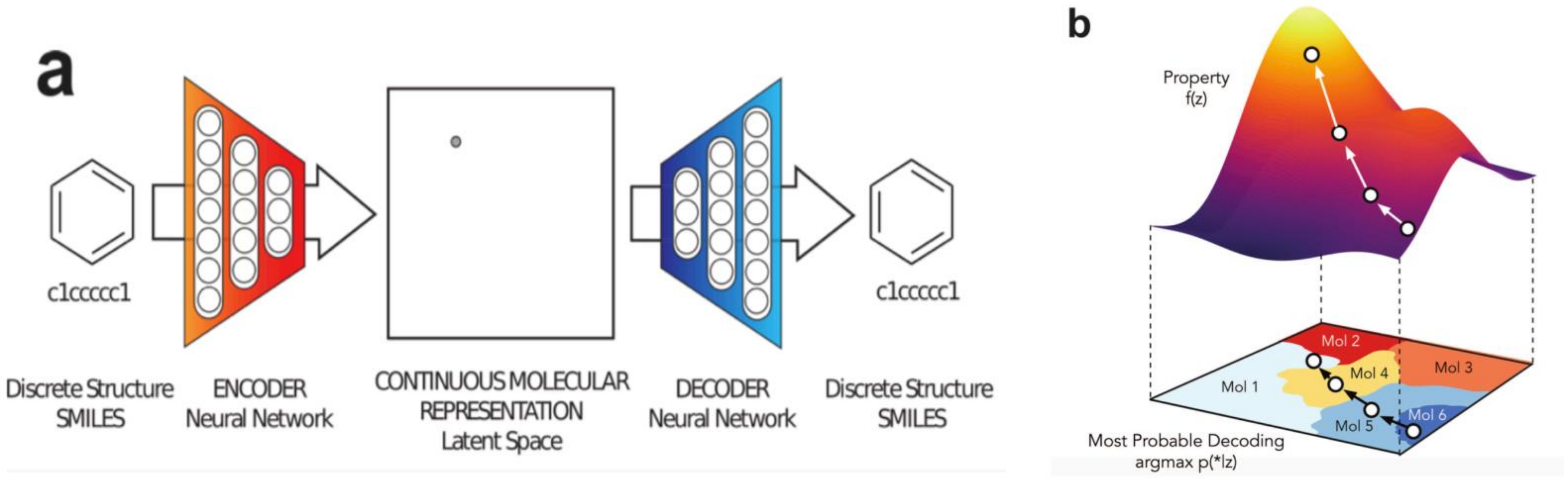
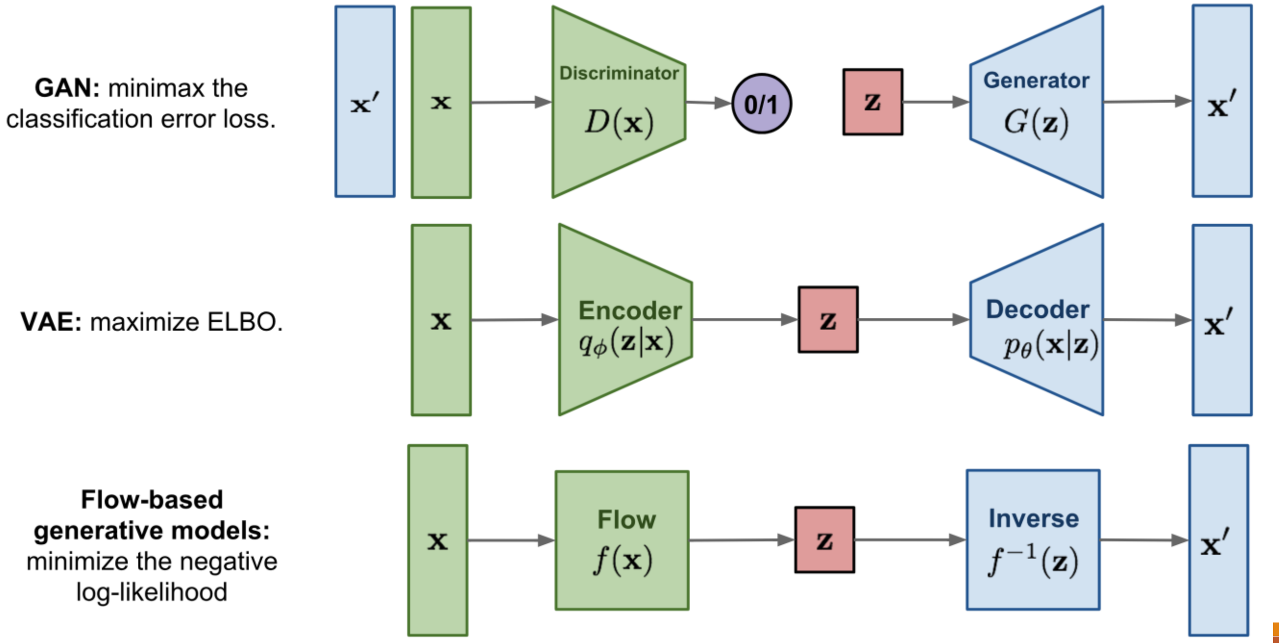


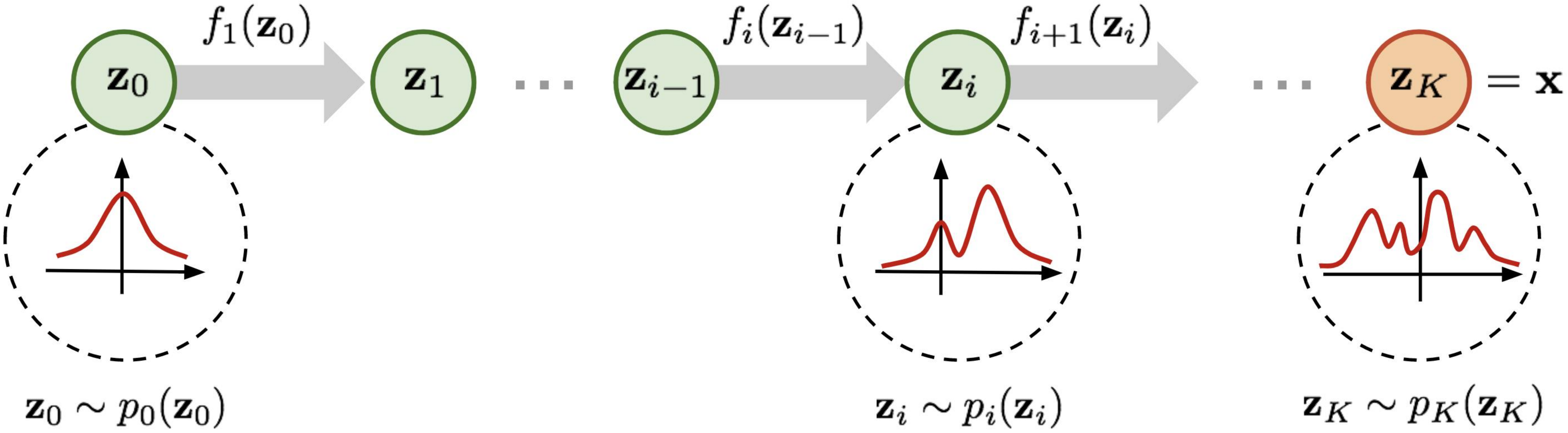
Figure 2.D.1: Example application of a VAE in [Gómez-Bombarelli et al., 2016]: design of new molecules with desired chemical properties. (a) A latent continuous representation $\mathbf{z}$ of molecules is learned on a large dataset of molecules. (b) This continuous representation enables gradient-based search of new molecules that maximizes some chosen desired chemical property given by objective function $f(\mathbf{z})$.

Normalizing Flows

- VAE cannot model $p(x)$ directly because of intractable formulation
- Normalizing Flows solves exactly that problem
- It does that by series of <u>invertible transformations</u> that allow for much more complex latent distributions (beyond Gaussians)
- The loss is the negative log-likelihood (not ELBO and so on)



Series of invertible transformations

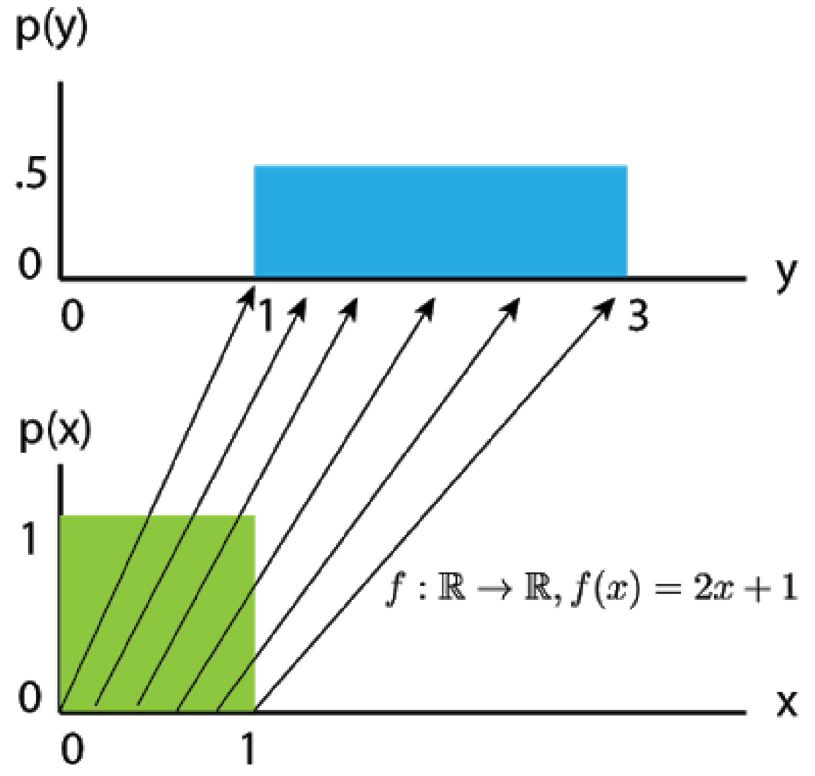


https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing Flows

https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf
https://blog.evjang.com/2018/01/nf1.html
https://arxiv.org/pdf/1505.05770.pdf

- Using simple pdfs, like a Gaussian, for the approximate posterior limits the expressivity of the model
- Better make sure the approximate posterior comes from a class of models that can <u>even</u> contain the true posterior
- Use a series of K invertible transformations to construct the approximate posterior
  - $z_k = f_k \circ f_{k-1} \circ \cdots f_1(z_0)$
  - Rule of change for variables



Changing from the x variable to y using the transformation $y = f(x) = 2x + 1$

Normalizing Flows: Log-likelihood

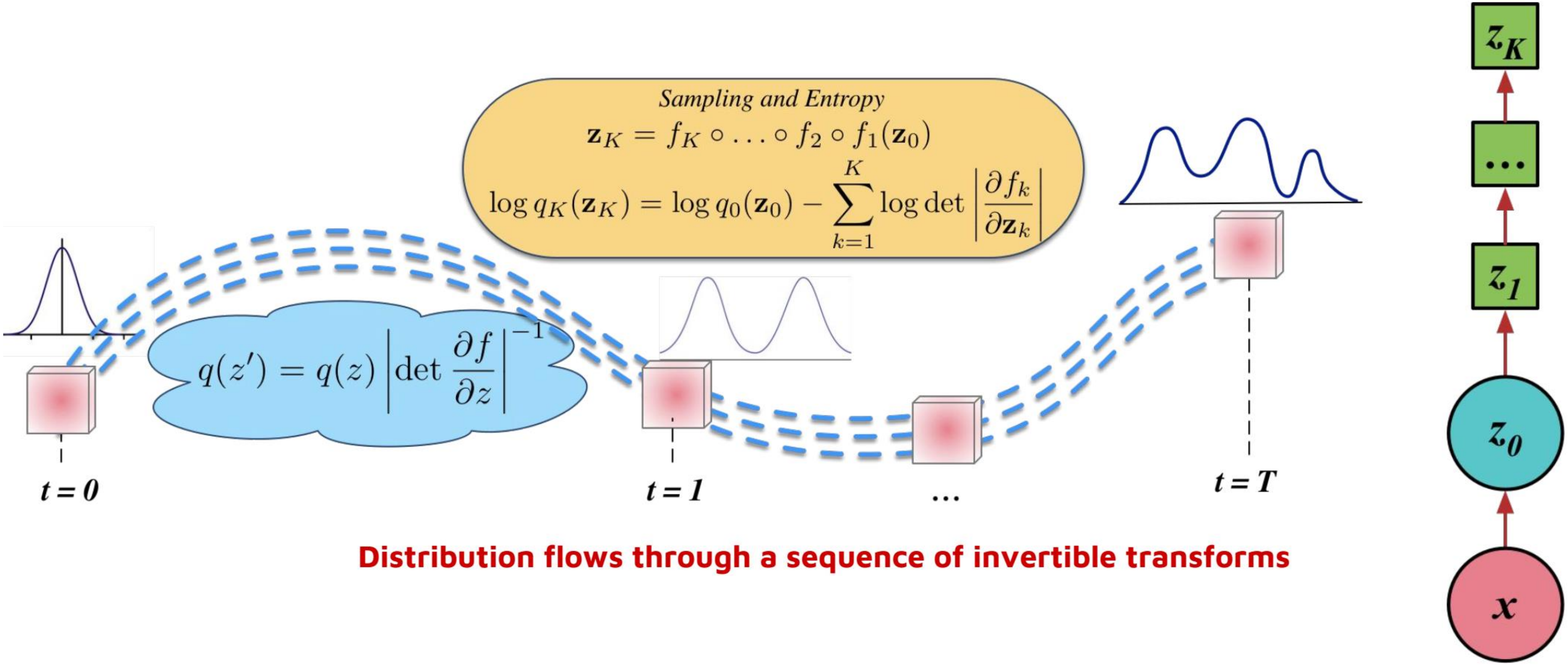
- $x = z_k = f_k \circ f_{k-1} \circ \cdots f_1(z_0) \quad \rightarrow \quad z_i = f_i(z_{i-1})$
- Again, change of variables (multi-dimensional): $p_i(z_i) = p_{i-1}\left(f_i^{-1}(z_i)\right) | \det \frac{df_i^{-1}}{dz_i} |$
- $$\log p(x) = \log \pi_K(z_K) = \log \pi_{K-1}(z_{K-1}) - \log | \det \frac{df_K}{df_{K-1}} |$$
$$= \cdots$$
$$= \log \pi_0(z_0) - \sum_i^K \log \left| \det \frac{df_i}{dz_{i-1}} \right|$$
- Two requirements

1. f_i must be easily invertible
2. The Jacobian of f_i must be easy to compute

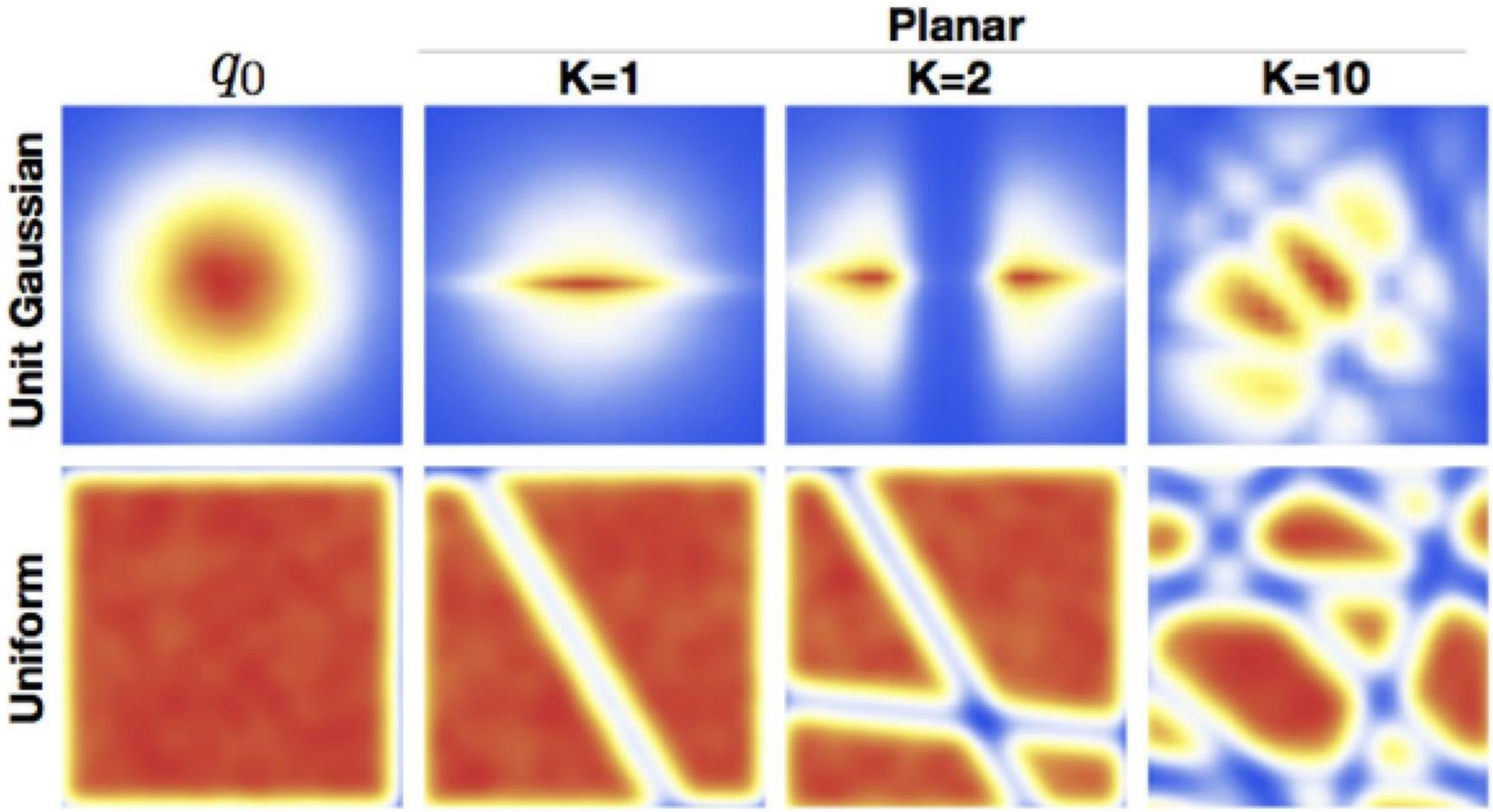
https://lilianweng.github.io/lil-log/2018/10/13/flow-based-deep-generative-models.html

Normalizing Flows

https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf
https://blog.evjang.com/2018/01/nf1.html
https://arxiv.org/pdf/1505.05770.pdf



Normalizing Flows



https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

Normalizing Flows on Non-Euclidean Manifolds

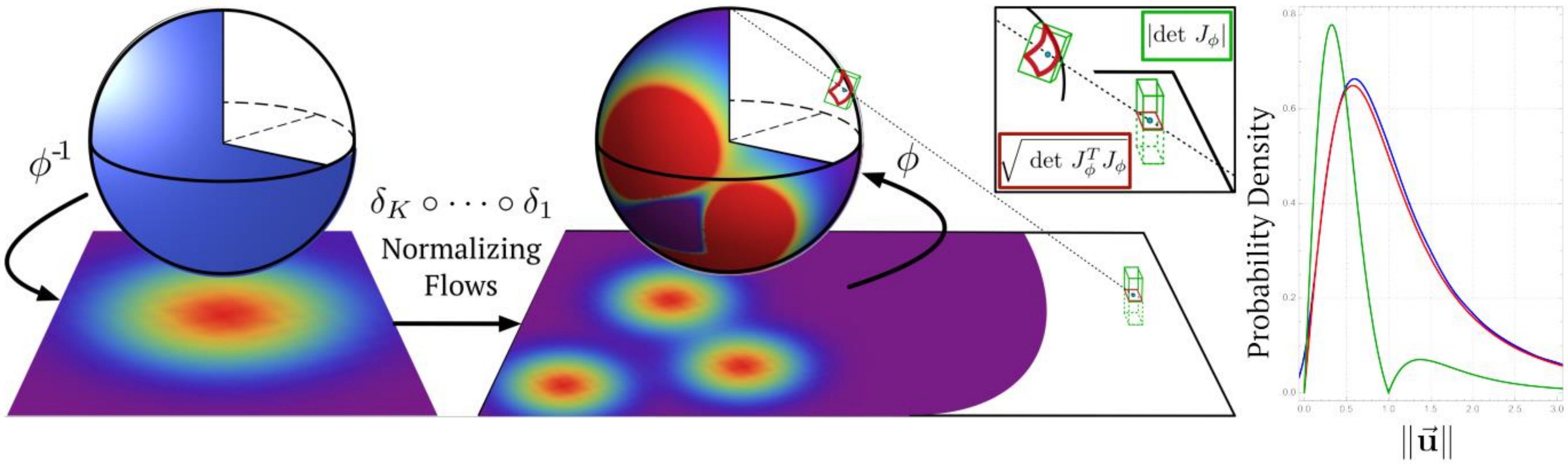


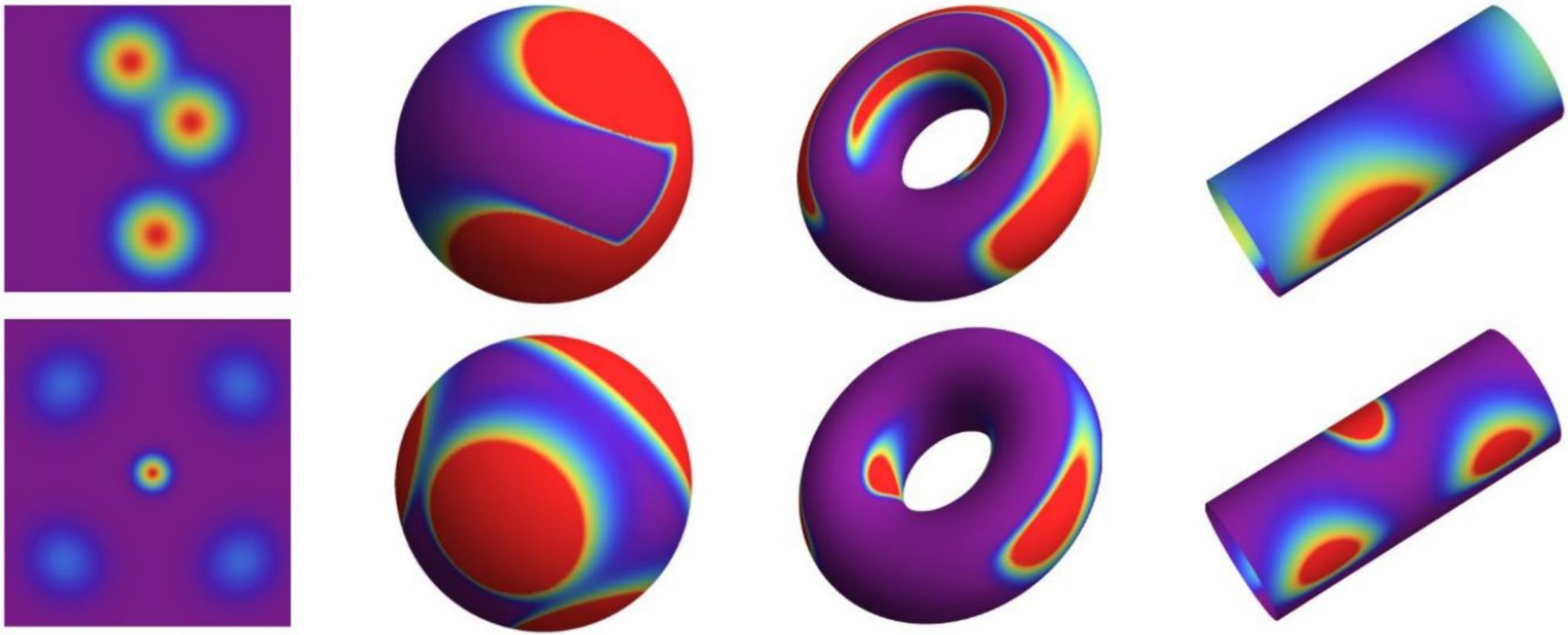
Figure 1: Left: Construction of a complex density on $\mathbf{S}^n$ by first projecting the manifold to $\mathbf{R}^n$, transforming the density and projecting it back to $\mathbf{S}^n$. Right: Illustration of transformed ($\mathbf{S}^2 \to \mathbf{R}^2$) densities corresponding to an uniform density on the sphere. Blue: empirical density (obtained by Monte Carlo); Red: Analytical density from equation (4); Green: Density computed ignoring the intrinsic dimensionality of $\mathbf{S}^n$.

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \frac{1}{2}\sum_{k=1}^{K} \log \det \left| \mathbf{J}_\phi{}^\top \mathbf{J}_\phi \right|$$

Gemici et al., 2016

https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

Normalizing Flows on Non-Euclidean Manifolds



Summary

- Gentle intro to Bayesian Modelling and Variational Inference
- Restricted Boltzmann Machines
- Deep Boltzmann Machines
- Deep Belief Network
- Contrastive Divergence
- Variational Autoencoders
- Normalizing Flows