Deep Probabilistic Models (2)

Piyush Rai

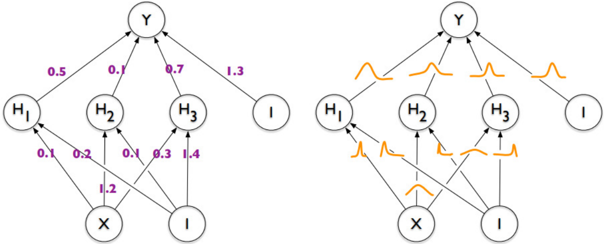
Probabilistic Machine Learning (CS772A)

Nov 2, 2017

Recap: Bayesian Neural Networks

- Responses modeled via a suitable prob. distribution whose params are outputs of an NN, e.g.,

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \qquad \text{(for real-valued responses)}$$



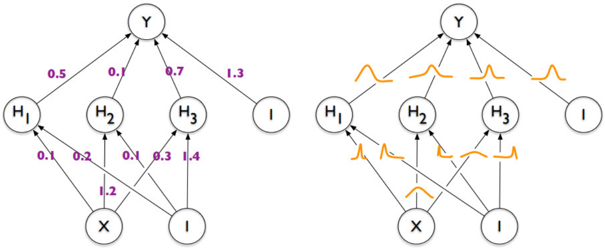
- $\text{NN}(\boldsymbol{x}_n; \mathbf{W})$ is a neural network with features $\boldsymbol{x}_n$ as its inputs and parameters $\mathbf{W}$

† "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Recap: Bayesian Neural Networks

- Responses modeled via a suitable prob. distribution whose params are outputs of an NN, e.g.,

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \qquad \text{(for real-valued responses)}$$

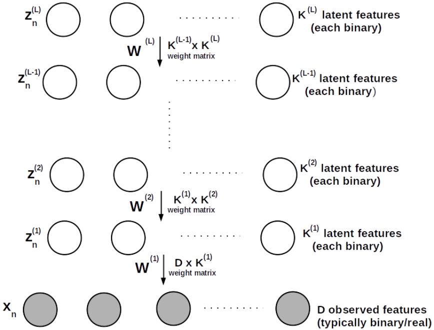


- $\text{NN}(\boldsymbol{x}_n; \mathbf{W})$ is a neural network with features $\boldsymbol{x}_n$ as its inputs and parameters $\mathbf{W}$
- Unlike standard neural networks, we learn the posterior over the unknowns
  - Non-conjugate model. MCMC or VB with Monte Carlo approximations typically used

† "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

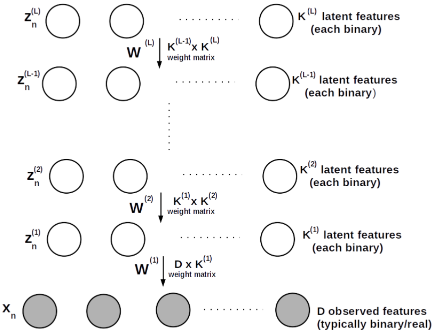
Recap: Sigmoid Belief Network

- An unsupervised generative model for the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Assumes data generated by successive nonlinear transformations of multiple layers of latent features



Recap: Sigmoid Belief Network

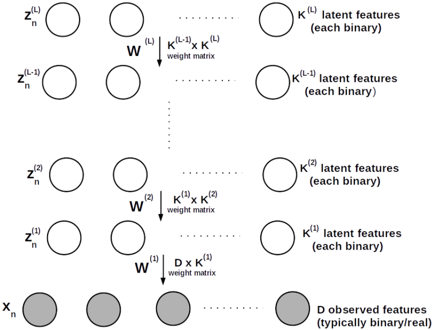
- An unsupervised generative model for the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Assumes data generated by successive nonlinear transformations of multiple layers of latent features



- In SBN, the latent features in each hidden layer are assumed to be binary-valued

Recap: Sigmoid Belief Network

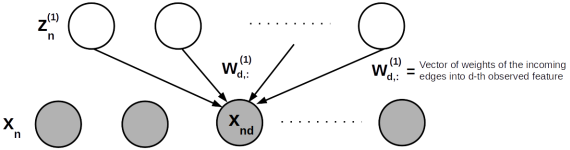
- An unsupervised generative model for the inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$
- Assumes data generated by successive nonlinear transformations of multiple layers of latent features



- In SBN, the latent features in each hidden layer are assumed to be binary-valued
- The goal is to infer $\mathbf{Z}^{(1)}, \ldots, \mathbf{Z}^{(L)}$ and the other parameters of the network (MCMC or VB with Monte Carlo approximations is needed since the model is non-conjugate)

Recap: Sigmoid Belief Network (A Zoomed-in Look)

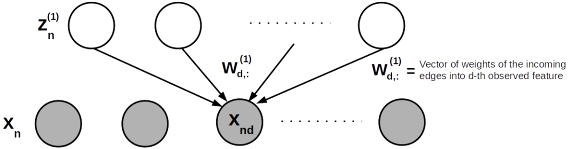
- Layer 1 hidden nodes (latent features) generate each observed feature (assuming binary) as



$$p(x_{nd} = 1 | \boldsymbol{z}_n^{(1)}) = \text{sigmoid}(\boldsymbol{w}_{d,:}^{(1)\top} \boldsymbol{z}_n^{(1)} + b_d)$$

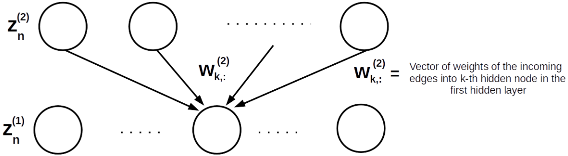
Recap: Sigmoid Belief Network (A Zoomed-in Look)

- Layer 1 hidden nodes (latent features) generate each observed feature (assuming binary) as



$$p(x_{nd} = 1|\boldsymbol{z}_n^{(1)}) \quad = \quad \text{sigmoid}(\boldsymbol{w}_{d,:}^{(1)\top}\boldsymbol{z}_n^{(1)} + b_d)$$

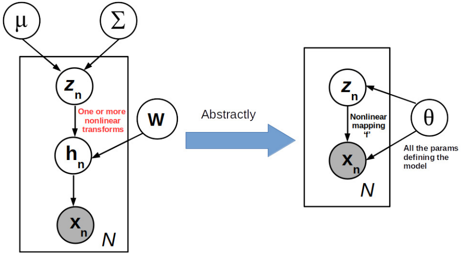
- Each hidden layer generates the nodes of hidden layer below it as (e.g., L2 $\rightarrow$ L1 in fig. below)



$$p(z_{nk}^{(1)} = 1|\boldsymbol{z}_n^{(2)}) \quad = \quad \text{sigmoid}(\boldsymbol{w}_{k,:}^{(2)\top}\boldsymbol{z}_n^{(2)} + c_k)$$

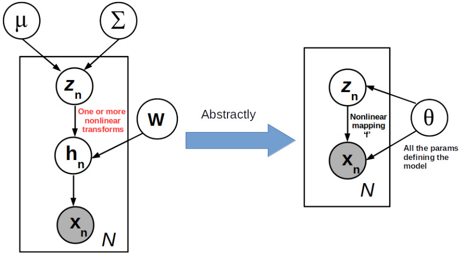
Recap: Deep Latent Gaussian Models

- Assume that a Gaussian latent variable $\boldsymbol{z}_n$ is pushed through a (deep) NN to generate $\boldsymbol{x}_n$



Recap: Deep Latent Gaussian Models

- Assume that a Gaussian latent variable $\boldsymbol{z}_n$ is pushed through a (deep) NN to generate $\boldsymbol{x}_n$



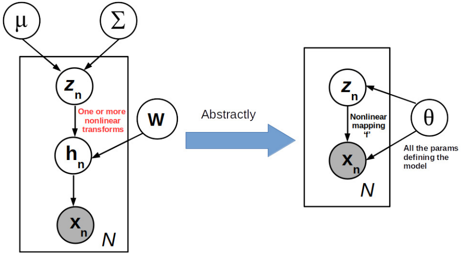
- A simple example with $\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ and assuming $\boldsymbol{x}_n \in \mathbb{R}^D$ with Gaussian likelihood

$$\boldsymbol{x}_n \sim \mathcal{N}(\boldsymbol{h}_n, \sigma^2 \mathbf{I}_D)$$

where $\boldsymbol{h}_n$ is a deterministic nonlinear transform of $\boldsymbol{z}_n$, e.g., $\boldsymbol{h}_n = \mathbf{W}\underbrace{\sigma(\mathbf{V}\boldsymbol{z}_n)}_{\text{sigmoid}}$ or $\boldsymbol{h}_n = \mathbf{W}\underbrace{\max\{\mathbf{0}, \mathbf{V}\boldsymbol{z}_n\}}_{\text{ReLU}}$

Recap: Deep Latent Gaussian Models

- Assume that a Gaussian latent variable $\boldsymbol{z}_n$ is pushed through a (deep) NN to generate $\boldsymbol{x}_n$



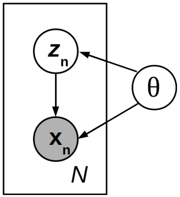
- A simple example with $\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ and assuming $\boldsymbol{x}_n \in \mathbb{R}^D$ with Gaussian likelihood

$$\boldsymbol{x}_n \sim \mathcal{N}(\boldsymbol{h}_n, \sigma^2 \mathbf{I}_D)$$

where $\boldsymbol{h}_n$ is a deterministic nonlinear transform of $\boldsymbol{z}_n$, e.g., $\boldsymbol{h}_n = \mathbf{W} \underbrace{\sigma(\mathbf{V}\boldsymbol{z}_n)}_{\text{sigmoid}}$ or $\boldsymbol{h}_n = \mathbf{W} \underbrace{\max\{\mathbf{0}, \mathbf{V}\boldsymbol{z}_n\}}_{\text{ReLU}}$

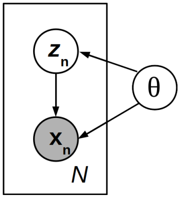
- The goal is to infer the latent variables $\boldsymbol{z}_n$'s and the other model params θ (MCMC or VB with Monte Carlo approximations is needed since the model is non-conjugate)

Inference for Deep Latent Gaussian Models



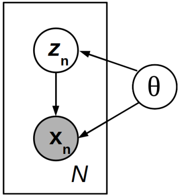
- The usual approach for inference in such models (as in most probabilistic models) is iterative

Inference for Deep Latent Gaussian Models



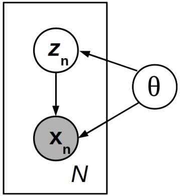
- The usual approach for inference in such models (as in most probabilistic models) is iterative, e.g.,
- Initialize θ. Then iterate until converence
  - For $n = 1, \ldots, N$
    - Infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ using MCMC. If doing VB, update variational parameters ϕ_n of $q(\boldsymbol{z}_n|\phi_n)$

Inference for Deep Latent Gaussian Models



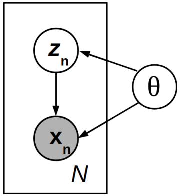
- The usual approach for inference in such models (as in most probabilistic models) is iterative, e.g.,
- Initialize θ. Then iterate until converence
  - For $n = 1, \ldots, N$
    - Infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ using MCMC. If doing VB, update variational parameters ϕ_n of $q(\boldsymbol{z}_n|\phi_n)$
  - Infer θ (its full posterior using MCMC or VB, or a point estimate)

Inference for Deep Latent Gaussian Models



- The usual approach for inference in such models (as in most probabilistic models) is iterative, e.g.,
- Initialize θ. Then iterate until converence
  - For $n = 1, \ldots, N$
    - Infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ using MCMC. If doing VB, update variational parameters ϕ_n of $q(\boldsymbol{z}_n|\phi_n)$
  - Infer θ (its full posterior using MCMC or VB, or a point estimate)
- This iterative approach can be slow for large N

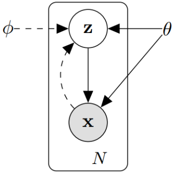
Inference for Deep Latent Gaussian Models



- The usual approach for inference in such models (as in most probabilistic models) is iterative, e.g.,
- Initialize θ. Then iterate until converence
  - For $n = 1, \ldots, N$
    - Infer $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ using MCMC. If doing VB, update variational parameters ϕ_n of $q(\boldsymbol{z}_n|\phi_n)$
  - Infer θ (its full posterior using MCMC or VB, or a point estimate)
- This iterative approach can be slow for large N
- Also, inferring $\boldsymbol{z}$ for new data point(s) $\boldsymbol{x}$ would require using the same iterative procedure

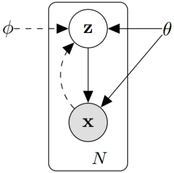
Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



Variational Auto-encoder (VAE)

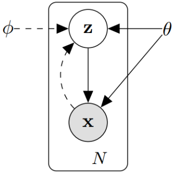
- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$

Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)

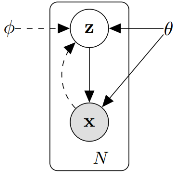


- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$
- Key idea: For each point $\boldsymbol{x}_n$, instead of learning a separate $q(\boldsymbol{z}_n|\phi_n)$ with local params ϕ_n, assume

$$q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\text{NN}(\boldsymbol{x}_n;\phi))$$

Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



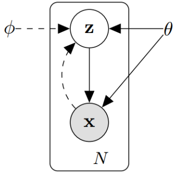
- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$
- Key idea: For each point $\boldsymbol{x}_n$, instead of learning a separate $q(\boldsymbol{z}_n|\phi_n)$ with local params ϕ_n, assume

$$q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\text{NN}(\boldsymbol{x}_n;\phi))$$

so, basically, each ϕ_n is computed by a neural net with global parameters ϕ and input $\boldsymbol{x}_n$

Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$
- Key idea: For each point $\boldsymbol{x}_n$, instead of learning a separate $q(\boldsymbol{z}_n|\phi_n)$ with local params ϕ_n, assume

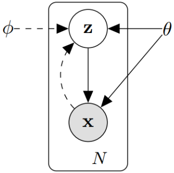
$$q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\text{NN}(\boldsymbol{x}_n;\phi))$$

  so, basically, each ϕ_n is computed by a neural net with global parameters ϕ and input $\boldsymbol{x}_n$

- Once ϕ is learned, we can get $q(\boldsymbol{z}_*|\boldsymbol{x}_*) = q(\boldsymbol{z}_*|\phi_*)$ for any $\boldsymbol{x}_*$ by just using $\phi_* = \text{NN}(\boldsymbol{x}_*;\phi)$

Variational Auto-encoder (VAE)

- Esseentially a DLGM, i.e., the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping $p(\boldsymbol{x}|\boldsymbol{z})$ is defined by a neural net. Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- VAE uses VB for inference but has a fast, non-iterative way of computing $\boldsymbol{z}_n$ for a data point $\boldsymbol{x}_n$
- Key idea: For each point $\boldsymbol{x}_n$, instead of learning a separate $q(\boldsymbol{z}_n|\phi_n)$ with local params ϕ_n, assume

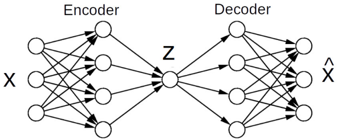
$$q(\boldsymbol{z}_n|\phi_n) = q(\boldsymbol{z}_n|\text{NN}(\boldsymbol{x}_n;\phi))$$

  so, basically, each ϕ_n is computed by a neural net with global parameters ϕ and input $\boldsymbol{x}_n$

- Once ϕ is learned, we can get $q(\boldsymbol{z}_*|\boldsymbol{x}_*) = q(\boldsymbol{z}_*|\phi_*)$ for any $\boldsymbol{x}_*$ by just using $\phi_* = \text{NN}(\boldsymbol{x}_*;\phi)$
- $p(\boldsymbol{x}|\boldsymbol{z})$ is known as decoder and $q(\boldsymbol{z}|\boldsymbol{x})$ is known as encoder

Standard Auto-encoder vs Variational Auto-encoder

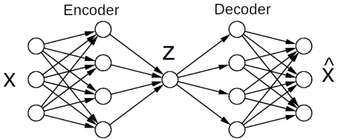
- A standard auto-encoder learns to (nonlinearly) compress and uncompress an input



- Model is trained to minimize the reconstruction error (difference b/w $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$)

Standard Auto-encoder vs Variational Auto-encoder

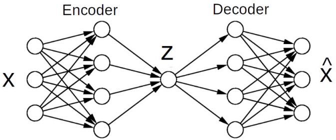
- A standard auto-encoder learns to (nonlinearly) compress and uncompress an input



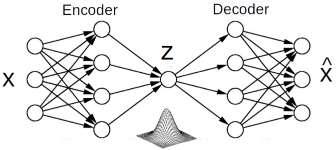
- Model is trained to minimize the reconstruction error (difference b/w $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$)
- However, it can't "generate" a "realistic" input from a random $\boldsymbol{z}$ (the model isn't <u>trained</u> for that)

Standard Auto-encoder vs Variational Auto-encoder

- A standard auto-encoder learns to (nonlinearly) compress and uncompress an input

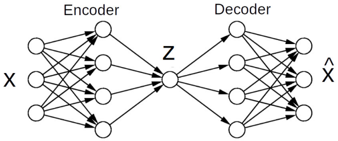


- Model is trained to minimize the reconstruction error (difference b/w $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$)
- However, it can't "generate" a "realistic" input from a random $\boldsymbol{z}$ (the model isn't trained for that)
- VAE allows this by assuming a distribution (e.g., Gaussian) over $\boldsymbol{z}$ and learning to generate $\boldsymbol{x}$ from random $\boldsymbol{z}$'s drawn from that distribution (so the model is trained to do this!)

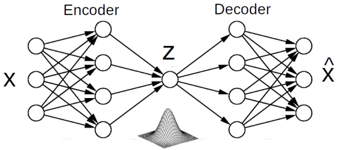


Standard Auto-encoder vs Variational Auto-encoder

- A standard auto-encoder learns to (nonlinearly) compress and uncompress an input



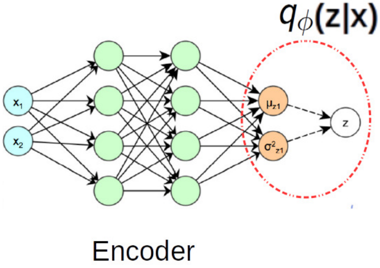
- Model is trained to minimize the reconstruction error (difference b/w $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$)
- However, it can't "generate" a "realistic" input from a random $\boldsymbol{z}$ (the model isn't trained for that)
- VAE allows this by assuming a distribution (e.g., Gaussian) over $\boldsymbol{z}$ and learning to generate $\boldsymbol{x}$ from random $\boldsymbol{z}$'s drawn from that distribution (so the model is trained to do this!)



- Note: Simple generative models like PPCA or factor analysis also have this ability to generate data from random $\boldsymbol{z}$ but the linear map from $\boldsymbol{z}$ to $\boldsymbol{x}$ limits the type of data that can be generated well

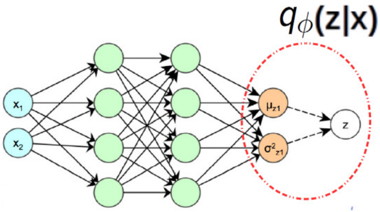
VAE: The Encoder

- Role of encoder: Take $\boldsymbol{x}$ as input and generate an encoding $\boldsymbol{z}$



VAE: The Encoder

- Role of encoder: Take $\boldsymbol{x}$ as input and generate an encoding $\boldsymbol{z}$

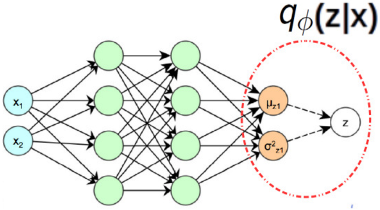


Encoder

- Unlike standard autoencoders, for each $\boldsymbol{x}$, VAE gives us a distribution $q(\boldsymbol{z}|\boldsymbol{x})$ over its encoding

VAE: The Encoder

- Role of encoder: Take $\boldsymbol{x}$ as input and generate an encoding $\boldsymbol{z}$



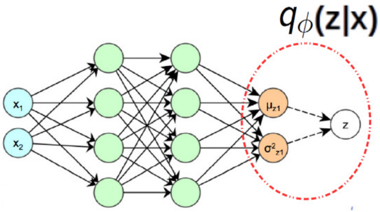
Encoder

- Unlike standard autoencoders, for each $\boldsymbol{x}$, VAE gives us a distribution $q(\boldsymbol{z}|\boldsymbol{x})$ over its encoding
- Assume $q(\boldsymbol{z}|\boldsymbol{x})$ to be Gaussian whose mean/var are computed by a NN with global params ϕ

$$\mu_z = \text{NN}(\boldsymbol{x};\phi) \qquad \sigma_z^2 = \text{NN}(\boldsymbol{x};\phi)$$

VAE: The Encoder

- Role of encoder: Take $\boldsymbol{x}$ as input and generate an encoding $\boldsymbol{z}$



Encoder

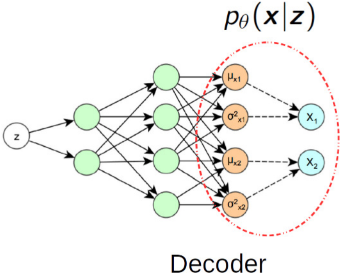
- Unlike standard autoencoders, for each $\boldsymbol{x}$, VAE gives us a distribution $q(\boldsymbol{z}|\boldsymbol{x})$ over its encoding
- Assume $q(\boldsymbol{z}|\boldsymbol{x})$ to be Gaussian whose mean/var are computed by a NN with global params ϕ

$$\mu_z = \text{NN}(\boldsymbol{x};\phi) \qquad \sigma_z^2 = \text{NN}(\boldsymbol{x};\phi)$$

- Since μ_z, σ_z are outputs of neural networks, the $\boldsymbol{x}$ to $\boldsymbol{z}$ mapping is nonlinear

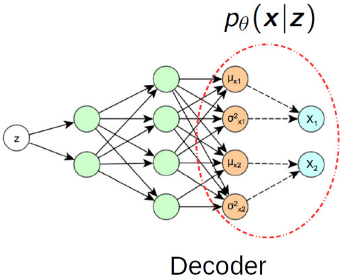
VAE: The Decoder

- Role of decoder: Generate $\boldsymbol{x}$ given $\boldsymbol{z}$. Defined by the likelihood model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$



VAE: The Decoder

- Role of decoder: Generate $\boldsymbol{x}$ given $\boldsymbol{z}$. Defined by the likelihood model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$

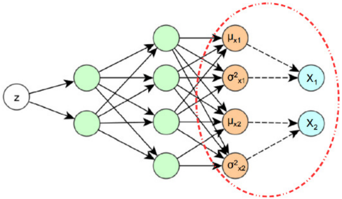


- Unlike PPCA (and like GPLVM), the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping is nonlinear (modeled by a neural network)

VAE: The Decoder

- Role of decoder: Generate $\boldsymbol{x}$ given $\boldsymbol{z}$. Defined by the likelihood model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$

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



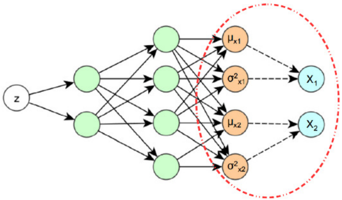
Decoder

- Unlike PPCA (and like GPLVM), the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping is nonlinear (modeled by a neural network)
- Assume $p(\boldsymbol{x}|\boldsymbol{z})$ to be Gaussian whose mean/var are computed by a NN with global params θ

$$\mu_x = \text{NN}(\boldsymbol{z};\theta) \qquad \sigma_x^2 = \text{NN}(\boldsymbol{z};\theta)$$

VAE: The Decoder

- Role of decoder: Generate $\boldsymbol{x}$ given $\boldsymbol{z}$. Defined by the likelihood model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$



Decoder

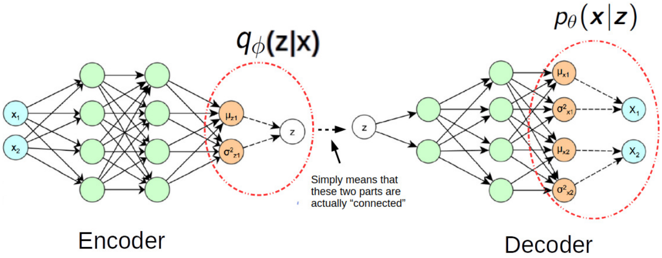
- Unlike PPCA (and like GPLVM), the $\boldsymbol{z}$ to $\boldsymbol{x}$ mapping is nonlinear (modeled by a neural network)
- Assume $p(\boldsymbol{x}|\boldsymbol{z})$ to be Gaussian whose mean/var are computed by a NN with global params θ

$$\mu_x = \text{NN}(\boldsymbol{z};\theta) \qquad \sigma_x^2 = \text{NN}(\boldsymbol{z};\theta)$$

- Thus in the VAE, both $\boldsymbol{x}$ to $\boldsymbol{z}$ (encoder) and $\boldsymbol{z}$ to $\boldsymbol{x}$ (decoder) mappings are nonlinear

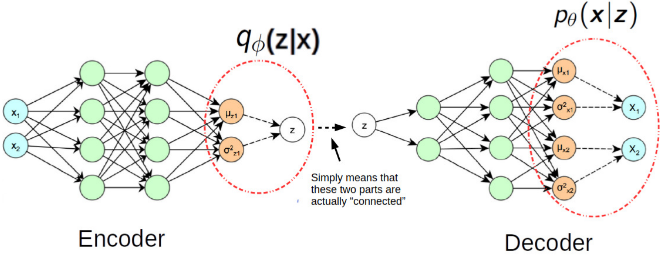
Inference for VAE

- VAE uses variational inference (hence the name!) to learn the model parameters θ and ϕ



Inference for VAE

- VAE uses variational inference (hence the name!) to learn the model parameters θ and ϕ

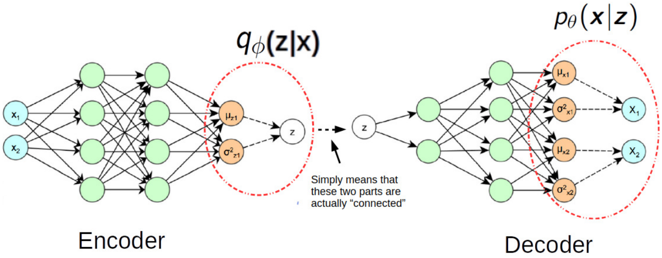


- Typically a prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ is assumed on $\boldsymbol{z}$. The ELBO for a single $\boldsymbol{x}_n$ will be

$$\text{ELBO} \quad = \quad \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \qquad \text{(note: } q_\phi \text{ and } q(\boldsymbol{z}_n|\boldsymbol{x}_n) \text{ mean the same)}$$

Inference for VAE

- VAE uses variational inference (hence the name!) to learn the model parameters θ and ϕ

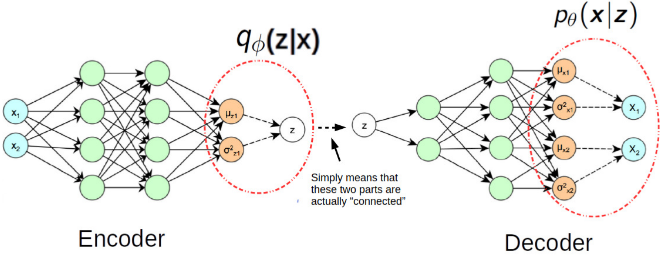


- Typically a prior $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ is assumed on $\mathbf{z}$. The ELBO for a single $\mathbf{x}_n$ will be

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\mathbf{x}_n, \mathbf{z}_n|\theta) - \log q(\mathbf{z}_n|\mathbf{x}_n)] \qquad \text{(note: } q_\phi \text{ and } q(\mathbf{z}_n|\mathbf{x}_n) \text{ mean the same)} \\ &= {\color{red}\mathbb{E}_{q_\phi}[\log p(\mathbf{x}_n|\mathbf{z}_n)]} - {\color{blue}\text{KL}(q(\mathbf{z}_n|\mathbf{x}_n)||p(\mathbf{z}_n))} \end{aligned}$$

Inference for VAE

- VAE uses variational inference (hence the name!) to learn the model parameters θ and ϕ



- Typically a prior $p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ is assumed on $\boldsymbol{z}$. The ELBO for a single $\boldsymbol{x}_n$ will be

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \qquad \text{(note: } q_\phi \text{ and } q(\boldsymbol{z}_n|\boldsymbol{x}_n) \text{ mean the same)} \\ &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)] - \text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n)) \end{aligned}$$

- ELBO intuition: Maximizing it will learn latent code $\boldsymbol{z}_n$ that will give good reconstruction for $\boldsymbol{x}_n$ (in expectation) and will keep $q(\boldsymbol{z}_n|\boldsymbol{x}_n)$ to be close to the prior $p(\boldsymbol{z}_n)$ (i.e., regularization)

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned}\text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)] - \text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))\end{aligned}$$

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= {\color{red}\mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)]} - {\color{blue}\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))} \end{aligned}$$

- ELBO is intractable because it requires computing expectations w.r.t. $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)] - \text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n)) \end{aligned}$$

- ELBO is intractable because it requires computing expectations w.r.t. $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$
- Note: If using the second form of the ELBO expression (the one with the KL term), the KL term has an analytic expression if $q(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ are Gaussians, and need not be approximated

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned}\text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)] - \text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))\end{aligned}$$

- ELBO is intractable because it requires computing expectations w.r.t. $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$
- Note: If using the second form of the ELBO expression (the one with the KL term), the KL term has an analytic expression if $q(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ are Gaussians, and need not be approximated
    - E.g., if $q(\boldsymbol{z}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}_n, \text{diag}(\sigma_{n1}^2, \ldots, \sigma_{nK}^2))$ and $p(\boldsymbol{z}_n) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ then

$$-\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n)) = \frac{1}{2}\sum_{k=1}^{K}(1 + \log(\sigma_{nk}^2) - \mu_{nk}^2 - \sigma_{nk}^2)$$

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned}\text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= \color{red}{\mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)]} - \color{blue}{\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))}\end{aligned}$$

- ELBO is intractable because it requires computing expectations w.r.t. $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$
- Note: If using the second form of the ELBO expression (the one with the KL term), the KL term has an analytic expression if $q(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ are Gaussians, and need not be approximated
  - E.g., if $q(\boldsymbol{z}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}_n, \text{diag}(\sigma_{n1}^2, \ldots, \sigma_{nK}^2))$ and $p(\boldsymbol{z}_n) = \mathcal{N}(\mathbf{0}, \mathbf{I}_K)$ then

$$-\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n)) = \frac{1}{2}\sum_{k=1}^{K}(1 + \log(\sigma_{nk}^2) - \mu_{nk}^2 - \sigma_{nk}^2)$$

- Since we basically want derivative of the ELBO, one option is to use Monte Carlo samples $\boldsymbol{z}_n^{(1)}, \ldots, \boldsymbol{z}_n^{(L)}$ from $q(\boldsymbol{z}_n|\phi_n)$ and approximate the ELBO derivative as (recall BBVI discussion)

Inference for VAE

- The ELBO is maximized w.r.t. the variational parameters ϕ

$$\begin{aligned}\text{ELBO} &= \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q(\boldsymbol{z}_n|\boldsymbol{x}_n)] \\ &= \color{red}{\mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n|\boldsymbol{z}_n)]} - \color{blue}{\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n))}\end{aligned}$$

- ELBO is intractable because it requires computing expectations w.r.t. $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$
- Note: If using the second form of the ELBO expression (the one with the KL term), the KL term has an analytic expression if $q(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ are Gaussians, and need not be approximated
    - E.g., if $q(\boldsymbol{z}_n|\boldsymbol{x}_n) = \mathcal{N}(\boldsymbol{\mu}_n, \text{diag}(\sigma_{n1}^2, \ldots, \sigma_{nK}^2))$ and $p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{0}, \mathbf{I}_K)$ then

$$-\text{KL}(q(\boldsymbol{z}_n|\boldsymbol{x}_n)||p(\boldsymbol{z}_n)) = \frac{1}{2}\sum_{k=1}^{K}(1 + \log(\sigma_{nk}^2) - \mu_{nk}^2 - \sigma_{nk}^2)$$

- Since we basically want derivative of the ELBO, one option is to use Monte Carlo samples $\boldsymbol{z}_n^{(1)}, \ldots, \boldsymbol{z}_n^{(L)}$ from $q(\boldsymbol{z}_n|\phi_n)$ and approximate the ELBO derivative as (recall BBVI discussion)

$$\nabla_\phi \mathbb{E}_q[f(\boldsymbol{z}_n)] = \mathbb{E}_q[f(\boldsymbol{z}_n)\nabla_\phi \log q(\boldsymbol{z}_n)] \approx \frac{1}{L}\sum_{\ell=1}^{L} f(\boldsymbol{z}_n^{(\ell)})\nabla_\phi \log q(\boldsymbol{z}_n^{(\ell)})$$

where $f(\boldsymbol{z}_n)$ denotes all the difficult terms in the ELBO expression

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} \quad = \quad \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} = \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

- Reparametrization Trick (Kingma & Welling, 2014): Make the randomness in $\boldsymbol{z}$ independent of ϕ

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} = \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

- Reparametrization Trick (Kingma & Welling, 2014): Make the randomness in $\boldsymbol{z}$ independent of ϕ
- Suppose our variational distribution $q_\phi(z|x) = \mathcal{N}(z|\mu, \sigma^2)$, where μ, σ are functions of $\boldsymbol{x}$

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} = \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

- Reparametrization Trick (Kingma & Welling, 2014): Make the randomness in $\boldsymbol{z}$ independent of ϕ
- Suppose our variational distribution $q_\phi(z|x) = \mathcal{N}(z|\mu, \sigma^2)$, where μ, σ are functions of $\boldsymbol{x}$
- We can reparametrize z as $z = g(\epsilon, \phi, x) = \mu + \sigma\epsilon, \quad$ where $\epsilon \sim \mathcal{N}(\epsilon|0, 1)$

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} = \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

- Reparametrization Trick (Kingma & Welling, 2014): Make the randomness in $\boldsymbol{z}$ independent of ϕ
- Suppose our variational distribution $q_\phi(z|x) = \mathcal{N}(z|\mu, \sigma^2)$, where μ, σ are functions of $\boldsymbol{x}$
- We can reparametrize z as $z = g(\epsilon, \phi, x) = \mu + \sigma\epsilon, \quad$ where $\epsilon \sim \mathcal{N}(\epsilon|0, 1)$
- Now the above ELBO can be replaced by

$$\text{ELBO} = \mathbb{E}_{p(\epsilon)}[\log p(\boldsymbol{x}_n, g(\epsilon, \phi, \boldsymbol{x}_n)|\theta) - \log q_\phi(g(\epsilon, \phi, \boldsymbol{x}_n))|\boldsymbol{x}_n)]$$

Reparametrization Trick

- The Monte Carlo estimator of ELBO's gradient (previous slide) suffers from high variance
- Reason: The drawn samples $\boldsymbol{z}_n^{(\ell)} \sim q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ depend on ϕ and ELBO itself depends on ϕ

$$\text{ELBO} = \mathbb{E}_{q_\phi}[\log p(\boldsymbol{x}_n, \boldsymbol{z}_n|\theta) - \log q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)]$$

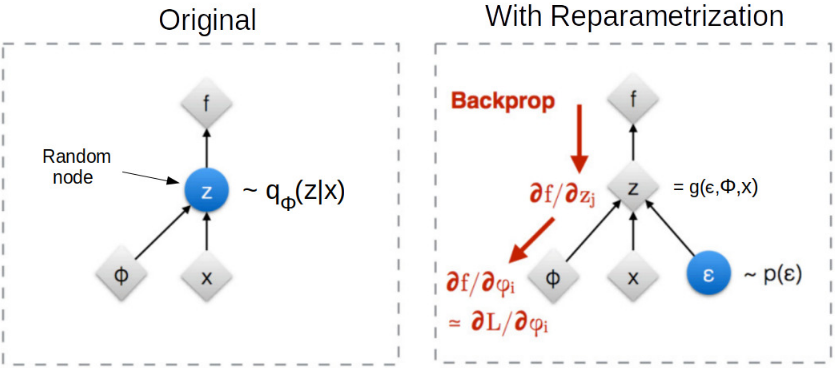
- Reparametrization Trick (Kingma & Welling, 2014): Make the randomness in $\boldsymbol{z}$ independent of ϕ
- Suppose our variational distribution $q_\phi(z|x) = \mathcal{N}(z|\mu, \sigma^2)$, where μ, σ are functions of $\boldsymbol{x}$
- We can reparametrize z as $z = g(\epsilon, \phi, x) = \mu + \sigma\epsilon, \quad$ where $\epsilon \sim \mathcal{N}(\epsilon|0, 1)$
- Now the above ELBO can be replaced by

$$\text{ELBO} = \mathbb{E}_{p(\epsilon)}[\log p(\boldsymbol{x}_n, g(\epsilon, \phi, \boldsymbol{x}_n)|\theta) - \log q_\phi(g(\epsilon, \phi, \boldsymbol{x}_n))|\boldsymbol{x}_n)]$$

- Given samples from $p(\epsilon)$, can easily approximate the ELBO (these samples don't depend on ϕ)

Reparametrization Trick

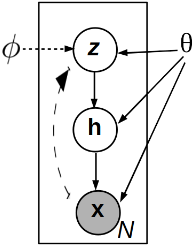
- Decoupling the randomness of $\boldsymbol{z}$ from ϕ using the reparametrization $z = g(\epsilon, \phi, x)$ also helps backpropagate easily through $\boldsymbol{z}$ when taking derivatives



VAE Architecture: An Example

- Assume a generative model (decoder) of the form $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ as specified below

$$
\begin{aligned}
\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \\
\text{where} \quad \boldsymbol{\mu} &= \mathbf{W}_4\mathbf{h} + \mathbf{b}_4 \\
\log \boldsymbol{\sigma}^2 &= \mathbf{W}_5\mathbf{h} + \mathbf{b}_5 \\
\mathbf{h} &= \tanh(\mathbf{W}_3\mathbf{z} + \mathbf{b}_3)
\end{aligned}
$$

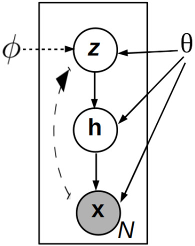


where θ consists of all the weights and bias terms

VAE Architecture: An Example

- Assume a generative model (decoder) of the form $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ as specified below

$$\begin{aligned}
\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \\
\text{where} \quad \boldsymbol{\mu} &= \mathbf{W}_4\mathbf{h} + \mathbf{b}_4 \\
\log \boldsymbol{\sigma}^2 &= \mathbf{W}_5\mathbf{h} + \mathbf{b}_5 \\
\mathbf{h} &= \tanh(\mathbf{W}_3\mathbf{z} + \mathbf{b}_3)
\end{aligned}$$



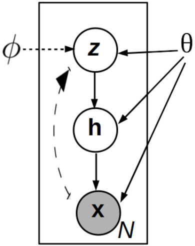
where θ consists of all the weights and bias terms

- The encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can also be defined to have the same form (swap $\boldsymbol{z}$ and $\boldsymbol{x}$ above) and the ϕ's will be the weights and bias terms

VAE Architecture: An Example

- Assume a generative model (decoder) of the form $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ as specified below

$$\begin{aligned}\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2\mathbf{I}) \\ \text{where } \boldsymbol{\mu} &= \mathbf{W}_4\mathbf{h} + \mathbf{b}_4 \\ \log \boldsymbol{\sigma}^2 &= \mathbf{W}_5\mathbf{h} + \mathbf{b}_5 \\ \mathbf{h} &= \tanh(\mathbf{W}_3\mathbf{z} + \mathbf{b}_3)\end{aligned}$$

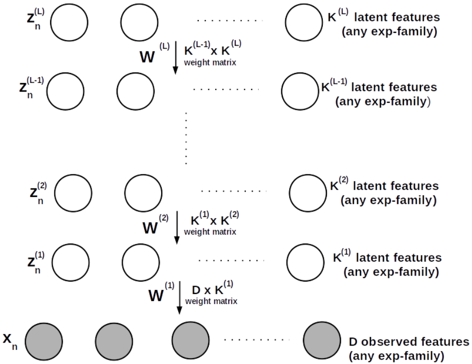


  where θ consists of all the weights and bias terms

- The encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ can also be defined to have the same form (swap $\boldsymbol{z}$ and $\boldsymbol{x}$ above) and the ϕ's will be the weights and bias terms
- Note: Recent work on VAE uses richer priors $p(\boldsymbol{z})$ as well as richer variational approx. $q(\boldsymbol{z}|\boldsymbol{x})$
  - For standard VAEs, these are simple Gaussians with diagonal covariances

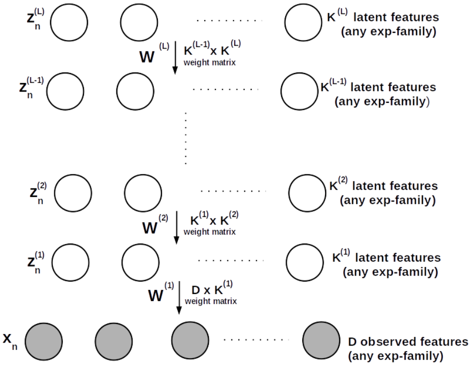
Other Architectures: Deep Exponential Families

- Similar in architecture to sigmoid belief networks
- However, latent variables in every layer, as well as observations, are from exp. family distributions



Other Architectures: Deep Exponential Families

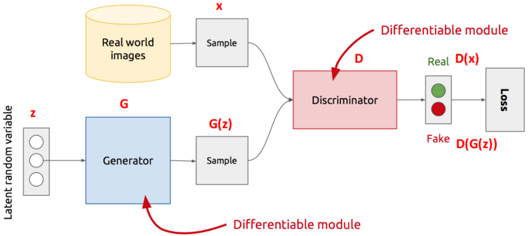
- Similar in architecture to sigmoid belief networks
- However, latent variables in every layer, as well as observations, are from exp. family distributions



- Overall model not conjugate but BBVI (Ranganath et al, 2013) or MCMC methods can be used

Other Architectures: Generative Adversarial Networks

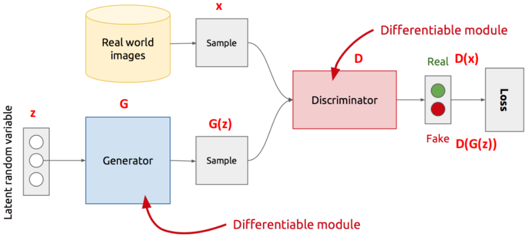
- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)



https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Other Architectures: Generative Adversarial Networks

- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)

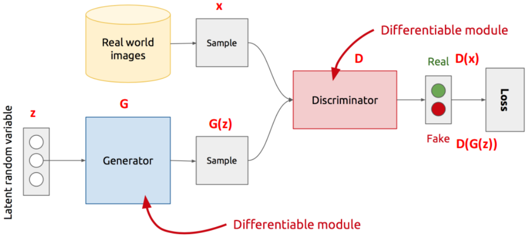


- Generator (a neural net) generates realistic looking "fake" data $\boldsymbol{x}$ from random $\boldsymbol{z}$

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Other Architectures: Generative Adversarial Networks

- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)

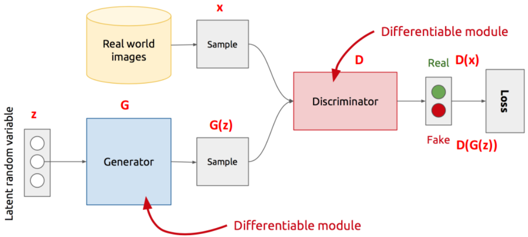


- Generator (a neural net) generates realistic looking "fake" data $\boldsymbol{x}$ from random $\boldsymbol{z}$
- Discriminator tries to detect fake data from real data

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Other Architectures: Generative Adversarial Networks

- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)

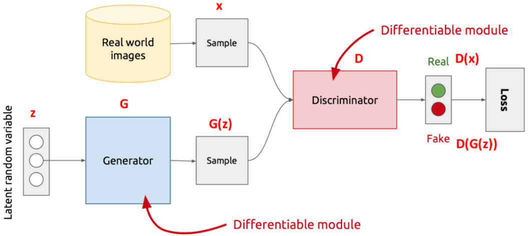


- Generator (a neural net) generates realistic looking "fake" data $\boldsymbol{x}$ from random $\boldsymbol{z}$
- Discriminator tries to detect fake data from real data
- At game's equilibrium, the $p_{gen} = p_{data}$ and success rate of discriminator = 50% (i.e., random)

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Other Architectures: Generative Adversarial Networks

- Based on a game between a generator and a discriminator (Goodfellow et al, 2013)



- Generator (a neural net) generates realistic looking "fake" data $\boldsymbol{x}$ from random $\boldsymbol{z}$
- Discriminator tries to detect fake data from real data
- At game's equilibrium, the $p_{gen} = p_{data}$ and success rate of discriminator = 50% (i.e., random)
- Originally designed mainly for synthetic data generation tasks but recent work extends GANs for many other problems such as latent variable inference, semi-supervised learning, etc.

---

https://www.slideshare.net/xavigiro/deep-learning-for-computer-vision-generative-models-and-adversarial-training-upc-2016

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)
- Several other models such as Deep Boltzmann Machines, Neural Autoregressive Density Estimator, etc. that we didn't cover here

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)
- Several other models such as Deep Boltzmann Machines, Neural Autoregressive Density Estimator, etc. that we didn't cover here
- An important distinction between explicit and implicit generative models

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)
- Several other models such as Deep Boltzmann Machines, Neural Autoregressive Density Estimator, etc. that we didn't cover here
- An important distinction between explicit and implicit generative models
  - Models like PPCA, FA, DLGM, SBN, VAE, etc. have an explicit likelihood model for data

Summary

- Probabilistic modeling allows developing very flexible deep learning models
- Much of the recent progress is fuelled by advances in probabilistic modeling and inference
- State-of-the-art results on a variety of tasks such as
  - Representation Learning (latent variables used as a new learned representation of data)
  - Density Estimation (i.e., $p(\boldsymbol{x})$)
  - Data generation (models like VAE and GAN can generate very realistic looking synthetic data)
  - Semi-supervised learning (models like VAE and GAN can be combined with supervised learning)
- Several other models such as Deep Boltzmann Machines, Neural Autoregressive Density Estimator, etc. that we didn't cover here
- An important distinction between explicit and implicit generative models
  - Models like PPCA, FA, DLGM, SBN, VAE, etc. have an explicit likelihood model for data
  - A model like GAN only defines $p(\boldsymbol{x})$ implicitly (no "likelihood" model for data)