Deep Probabilistic Models (1)

Piyush Rai

Probabilistic Machine Learning (CS772A)

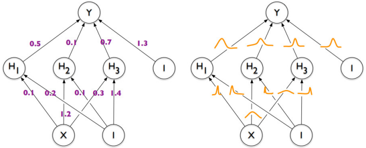
Oct 31, 2017

Plan (today and next class)

- Will look at some fundamental/popular deep probabilistic models

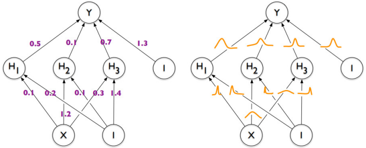
Plan (today and next class)

- Will look at some fundamental/popular deep probabilistic models
  - Supervised Learning: Probabilistic/Bayesian neural networks

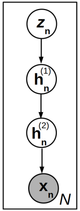


Plan (today and next class)

- Will look at some fundamental/popular deep probabilistic models
  - Supervised Learning: Probabilistic/Bayesian neural networks

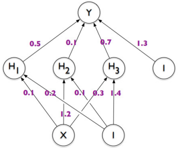


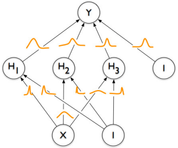
  - Unsupervised Learning: Deep Generative Models



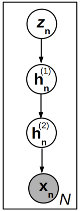
Plan (today and next class)

- Will look at some fundamental/popular deep probabilistic models
  - Supervised Learning: Probabilistic/Bayesian neural networks





  - Unsupervised Learning: Deep Generative Models



- Disclaimer: Very rapidly evolving area, deserves an entire course. Will only cover the basic principles underlying some of the representative models (from probabilistic/Bayesian perspective)

Deep Probabilistic Models

- Deep probabilistic models for supervised learning

$$\begin{aligned} y_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \quad \text{(for real-valued responses)} \\ y_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{x}_n; \mathbf{W})) \quad \text{(for other types of responses)} \end{aligned}$$

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

Deep Probabilistic Models

- Deep probabilistic models for supervised learning

$$
\begin{aligned}
y_n &\sim \mathcal{N}(\mathrm{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \quad \text{(for real-valued responses)} \\
y_n &\sim \mathrm{ExpFam}(\mathrm{NN}(\boldsymbol{x}_n; \mathbf{W})) \quad \text{(for other types of responses)}
\end{aligned}
$$

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

- Deep probabilistic model for unsupervised learning

$$
\begin{aligned}
\boldsymbol{x}_n &\sim \mathcal{N}(\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W}), \sigma^2 \mathbf{I}_D) \quad \text{(for real-valued features)} \\
\boldsymbol{x}_n &\sim \mathrm{ExpFam}(\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W})) \quad \text{(for other types of features)}
\end{aligned}
$$

where $\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W})$ is a neural network with latent variables $\boldsymbol{z}_n$ as inputs and and parameters $\mathbf{W}$

Deep Probabilistic Models

- Deep probabilistic models for supervised learning

$$\begin{aligned} y_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \quad \text{(for real-valued responses)} \\ y_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{x}_n; \mathbf{W})) \quad \text{(for other types of responses)} \end{aligned}$$

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

- Deep probabilistic model for unsupervised learning

$$\begin{aligned} \boldsymbol{x}_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{z}_n; \mathbf{W}), \sigma^2 \mathbf{I}_D) \quad \text{(for real-valued features)} \\ \boldsymbol{x}_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{z}_n; \mathbf{W})) \quad \text{(for other types of features)} \end{aligned}$$

  where $\text{NN}(\boldsymbol{z}_n; \mathbf{W})$ is a neural network with latent variables $\boldsymbol{z}_n$ as inputs and and parameters $\mathbf{W}$

- In both cases, these models learn a nonlinear mapping from inputs to outputs

Deep Probabilistic Models

- Deep probabilistic models for supervised learning

$$\begin{aligned} y_n &\sim \mathcal{N}(\mathrm{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \quad \text{(for real-valued responses)} \\ y_n &\sim \mathrm{ExpFam}(\mathrm{NN}(\boldsymbol{x}_n; \mathbf{W})) \quad \text{(for other types of responses)} \end{aligned}$$

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

- Deep probabilistic model for unsupervised learning

$$\begin{aligned} \boldsymbol{x}_n &\sim \mathcal{N}(\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W}), \sigma^2 \mathbf{I}_D) \quad \text{(for real-valued features)} \\ \boldsymbol{x}_n &\sim \mathrm{ExpFam}(\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W})) \quad \text{(for other types of features)} \end{aligned}$$

  where $\mathrm{NN}(\boldsymbol{z}_n; \mathbf{W})$ is a neural network with latent variables $\boldsymbol{z}_n$ as inputs and and parameters $\mathbf{W}$

- In both cases, these models learn a nonlinear mapping from inputs to outputs
  - The nonlinear nature of the NN architecture facilitates learning these nonlinear mappings

Deep Probabilistic Models

- Deep probabilistic models for supervised learning

$$
\begin{aligned}
y_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2) \quad \text{(for real-valued responses)} \\
y_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{x}_n; \mathbf{W})) \quad \text{(for other types of responses)}
\end{aligned}
$$

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

- Deep probabilistic model for unsupervised learning

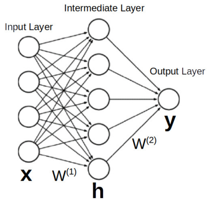
$$
\begin{aligned}
\boldsymbol{x}_n &\sim \mathcal{N}(\text{NN}(\boldsymbol{z}_n; \mathbf{W}), \sigma^2 \mathbf{I}_D) \quad \text{(for real-valued features)} \\
\boldsymbol{x}_n &\sim \text{ExpFam}(\text{NN}(\boldsymbol{z}_n; \mathbf{W})) \quad \text{(for other types of features)}
\end{aligned}
$$

  where $\text{NN}(\boldsymbol{z}_n; \mathbf{W})$ is a neural network with latent variables $\boldsymbol{z}_n$ as inputs and and parameters $\mathbf{W}$

- In both cases, these models learn a nonlinear mapping from inputs to outputs
  - The nonlinear nature of the NN architecture facilitates learning these nonlinear mappings
- Note: Such models are not limited to exp-family distributions only

Neural Network

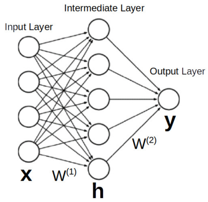
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes

Neural Network

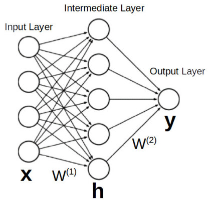
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,

Neural Network

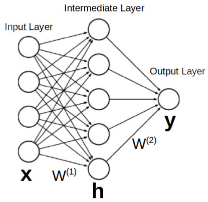
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,
  - Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid)

Neural Network

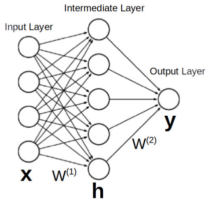
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,
  - Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid)
  - A nonlinear function modeled by a Gaussian Process

Neural Network

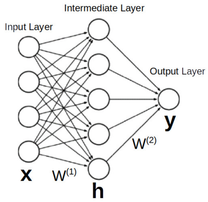
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,
  - Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid)
  - A nonlinear function modeled by a Gaussian Process
- An NN with one hidden layer with infinite many nodes can approximate any function

Neural Network

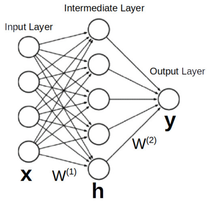
- A simple neural network with one intermediate (also called "hidden") layer and a single output



- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,
  - Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid)
  - A nonlinear function modeled by a Gaussian Process
- An NN with one hidden layer with infinite many nodes can approximate any function
  - Note: A GP is equivalent to such an NN if the NN weights have Gaussian priors†

Neural Network

- A simple neural network with one intermediate (also called "hidden") layer and a single output

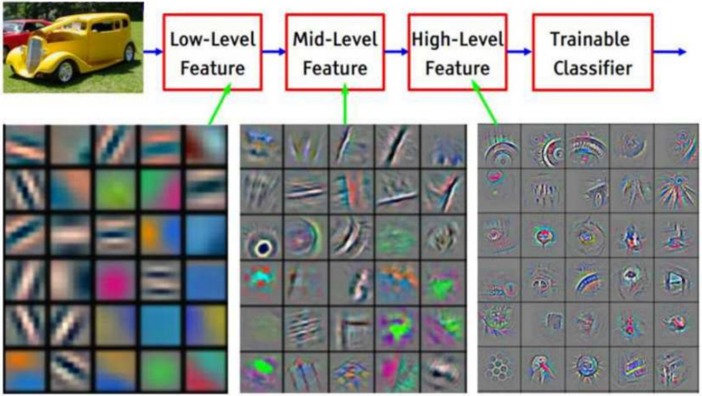


- Each intermediate layer computes a nonlinear transformation of its previous layer's nodes
- The nonlinear transform that computes $\boldsymbol{h}$ can be defined in various ways, e.g.,
  - Linear transform (e.g., $\mathbf{W}^{(1)}\boldsymbol{x}$ in the above picture) followed by a nonlinearlity (e.g., sigmoid)
  - A nonlinear function modeled by a Gaussian Process
- An NN with one hidden layer with infinite many nodes can approximate any function
  - Note: A GP is equivalent to such an NN if the NN weights have Gaussian priors†

† Priors for Infinite Networks (Neal, 1995)

Deep Models as Feature Extractors

Deep models can learn features at different levels of abstraction



Example: Lowest layer learns edges, higher layers learns (parts of) objects

Deep Probabilistic Models for Supervised Learning

Deep Probabilistic Models for Supervised Learning

- Consider a regression (supervised learning) problem given training data $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2)$$

Pic courtesy: "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Deep Probabilistic Models for Supervised Learning

- Consider a regression (supervised learning) problem given training data $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2)$$

- The goal is to learn the parameters $\mathbf{W}$. Point estimation (MLE/MAP) via gradient methods

Pic courtesy: "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Deep Probabilistic Models for Supervised Learning

- Consider a regression (supervised learning) problem given training data $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2)$$

- The goal is to learn the parameters $\mathbf{W}$. Point estimation (MLE/MAP) via gradient methods
    - Regularization important (a huge number of parameters); amounts to using a prior on $\mathbf{W}$

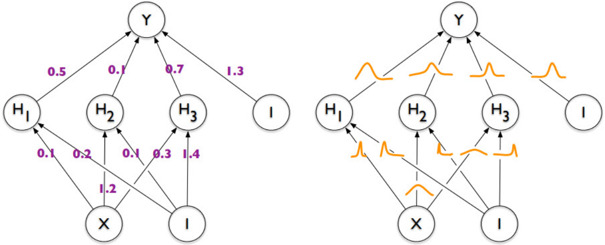
Pic courtesy: "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Deep Probabilistic Models for Supervised Learning

- Consider a regression (supervised learning) problem given training data $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2)$$

- The goal is to learn the parameters $\mathbf{W}$. Point estimation (MLE/MAP) via gradient methods
  - Regularization important (a huge number of parameters); amounts to using a prior on $\mathbf{W}$
- Overfitting a concern. Fully Bayesian inference is desirable (i.e., a Bayesian neural network)



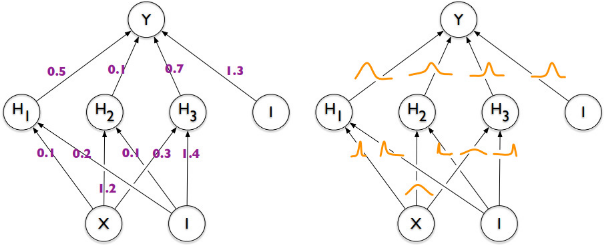
Pic courtesy: "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

Deep Probabilistic Models for Supervised Learning

- Consider a regression (supervised learning) problem given training data $\mathcal{D} = \{\boldsymbol{x}_n, y_n\}_{n=1}^N$

$$y_n \sim \mathcal{N}(\text{NN}(\boldsymbol{x}_n; \mathbf{W}), \sigma^2)$$

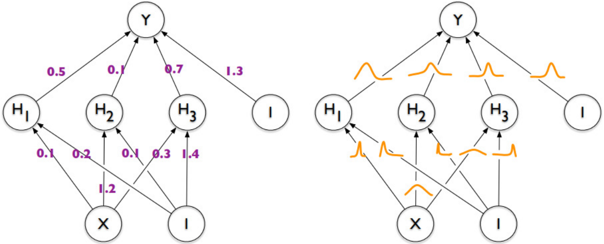
- The goal is to learn the parameters $\mathbf{W}$. Point estimation (MLE/MAP) via gradient methods
    - Regularization important (a huge number of parameters); amounts to using a prior on $\mathbf{W}$
- Overfitting a concern. Fully Bayesian inference is desirable (i.e., a Bayesian neural network)



- Fully Bayesian treatment for such neural nets also has other benefits (e.g., Active Learning)

Pic courtesy: "Weight Uncertainty in Neural Networks" (Blundell et al, 2015)

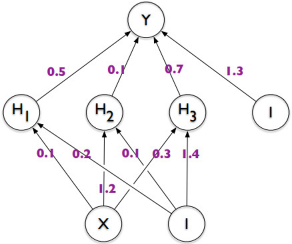
Learning Bayesian Neural Networks

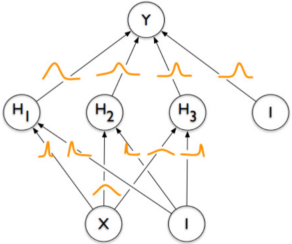


- Even if $p(\mathbf{W})$ is Gaussian, the model is not conjugate due to the nonlinearities

† "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks" (Li et al, 2016)

Learning Bayesian Neural Networks

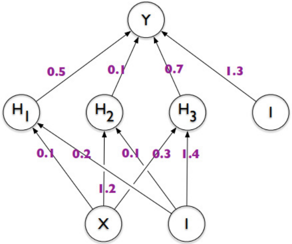


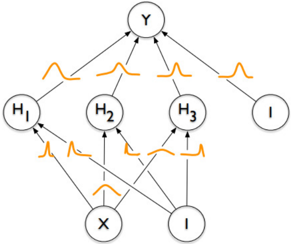


- Even if $p(\mathbf{W})$ is Gaussian, the model is not conjugate due to the nonlinearities
- MCMC methods can be used to learn the posterior $p(\mathbf{W}|\mathcal{D})$ but can be slow

† "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks" (Li et al, 2016)

Learning Bayesian Neural Networks

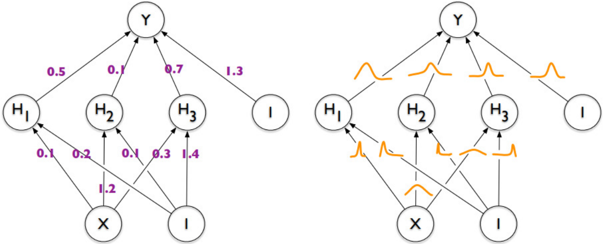




- Even if $p(\mathbf{W})$ is Gaussian, the model is not conjugate due to the nonlinearities
- MCMC methods can be used to learn the posterior $p(\mathbf{W}|\mathcal{D})$ but can be slow
    - However, methods such as SGLD[†] allow efficient MCMC inference for such models (recall that SGLD only requires gradient expressions of the log-joint probability $\log p(\mathcal{D}, \mathbf{W})$ of the model)

[†] "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks" (Li et al, 2016)

Learning Bayesian Neural Networks



- Even if $p(\mathbf{W})$ is Gaussian, the model is not conjugate due to the nonlinearities
- MCMC methods can be used to learn the posterior $p(\mathbf{W}|\mathcal{D})$ but can be slow
  - However, methods such as SGLD[†] allow efficient MCMC inference for such models (recall that SGLD only requires gradient expressions of the log-joint probability $\log p(\mathcal{D}, \mathbf{W})$ of the model)
- Variational inference is another popular alternative to MCMC for such models

[†] "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks" (Li et al, 2016)

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$\hat{\phi} \quad = \quad \arg\min_{\phi} \text{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D}))$$

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$\begin{aligned}\hat{\phi} &= \arg\min_{\phi} \mathrm{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D})) \\ &= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D},\mathbf{W}) - \log q(\mathbf{W}|\phi)] \qquad \text{(The ELBO)}\end{aligned}$$

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$\begin{aligned}\hat{\phi} &= \arg\min_{\phi} \mathrm{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D})) \\ &= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D},\mathbf{W}) - \log q(\mathbf{W}|\phi)] \qquad \text{(The ELBO)} \\ &= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{W}) + \log p(\mathbf{W}) - \log q(\mathbf{W}|\phi)]\end{aligned}$$

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$
\begin{aligned}
\hat{\phi} &= \arg\min_{\phi} \mathrm{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D})) \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}, \mathbf{W}) - \log q(\mathbf{W}|\phi)] \qquad \text{(The ELBO)} \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{W}) + \log p(\mathbf{W}) - \log q(\mathbf{W}|\phi)]
\end{aligned}
$$

- The expectation above can be approximated using Monte Carlo samples from $q(\mathbf{W})$

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$\begin{aligned}
\hat{\phi} &= \arg\min_{\phi} \mathrm{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D})) \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}, \mathbf{W}) - \log q(\mathbf{W}|\phi)] \qquad \text{(The ELBO)} \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{W}) + \log p(\mathbf{W}) - \log q(\mathbf{W}|\phi)]
\end{aligned}$$

- The expectation above can be approximated using Monte Carlo samples from $q(\mathbf{W})$
  - Note: Variance reduction methods typically employed to reduce the variance in the estimate

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

Variational Inference for Bayesian Neural Networks

- Standard VI doesn't work for Bayesian neural networks
- Reason: The ELBO is intractable (due to non-conjugacy)
  - To do VI, ELBO has to be approximated, e.g., using Monte Caro methods[†]
- Suppose $p(\mathbf{W}|\mathcal{D})$ is the true posterior, $q(\mathbf{W}|\phi)$ is the variational approximation

$$
\begin{aligned}
\hat{\phi} &= \arg\min_{\phi} \text{KL}(q(\mathbf{W}|\phi)||p(\mathbf{W}|\mathcal{D})) \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}, \mathbf{W}) - \log q(\mathbf{W}|\phi)] \qquad \text{(The ELBO)} \\
&= \arg\max_{\phi} \mathbb{E}_q[\log p(\mathcal{D}|\mathbf{W}) + \log p(\mathbf{W}) - \log q(\mathbf{W}|\phi)]
\end{aligned}
$$

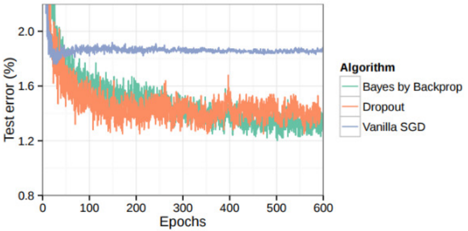
- The expectation above can be approximated using Monte Carlo samples from $q(\mathbf{W})$
  - Note: Variance reduction methods typically employed to reduce the variance in the estimate
- Black-box VI can be another alternative (again based on Monte Carlo approximation)

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

Bayesian Neural Networks on MNIST

- Some results on MNIST† (using standard feedforward network as the model; no convolution etc.)

Method	# Units/Layer	# Weights	Test Error
SGD, no regularisation (Simard et al., 2003)	800	1.3m	1.6%
SGD, dropout (Hinton et al., 2012)			≈ 1.3%
SGD, dropconnect (Wan et al., 2013)	800	1.3m	**1.2%***
SGD	400	500k	1.83%
	800	1.3m	1.84%
	1200	2.4m	1.88%
SGD, dropout	400	500k	1.51%
	800	1.3m	1.33%
	1200	2.4m	1.36%
Bayes by Backprop, Gaussian	400	500k	1.82%
	800	1.3m	1.99%
	1200	2.4m	2.04%
Bayes by Backprop, Scale mixture	400	500k	1.36%
	800	1.3m	1.34%
	1200	2.4m	**1.32%**



- Note: The scale mixture prior is a sparsity inducing prior of the form

$$p(\boldsymbol{w}) = \prod_j \pi \mathcal{N}(w_j|0,\sigma_1^2) + (1-\pi)\mathcal{N}(w_j|0,\sigma_2^2) \qquad (\sigma_1^2 \gg \sigma_2^2)$$

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

Bayesian Neural Networks: Summary

- Only scratched the surface. Many advanced variants exist
- Bayesian convolutional neural network[†] (CNN) for image data
- Since we get uncertainty in model and its predictions, can be extended to active learning[‡]
- Can also be extended to do semi-supervised learning when combined with deep generative models (we will look at DGMs)
- A lot of recent work on scalable Bayesian inference for Bayesian neural networks
  - Both MCMC and variational inference

[†] Bayesian convolutional neural networks with Bernoulli approximate variational inference (Gal and Ghahramani, 2016)
[‡] Deep Bayesian Active Learning with Image Data (Gal and Ghahramani, 2017)

Deep Probabilistic Models for Unsupervised Learning

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction
- Good models for estimating the probability density $p(\boldsymbol{x})$ of data (akin to GMM, PPCA, etc)

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction
- Good models for estimating the probability density $p(\boldsymbol{x})$ of data (akin to GMM, PPCA, etc)
  - Using deep neural nets in these models allows learning very complex densities

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction
- Good models for estimating the probability density $p(\boldsymbol{x})$ of data (akin to GMM, PPCA, etc)
  - Using deep neural nets in these models allows learning very complex densities
- Good models for generating new data from the learned model

Why Deep Probabilistic Models for Unsupervised Learning?

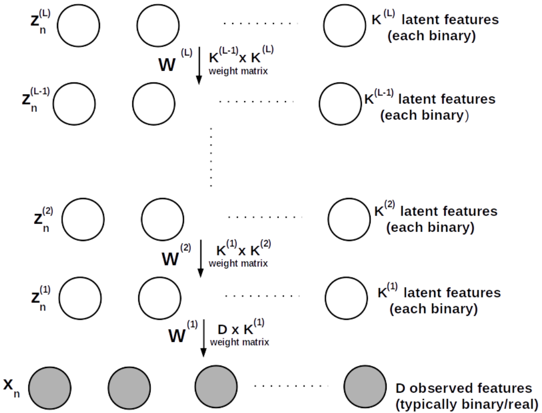
- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction
- Good models for estimating the probability density $p(\boldsymbol{x})$ of data (akin to GMM, PPCA, etc)
  - Using deep neural nets in these models allows learning very complex densities
- Good models for generating new data from the learned model
- Useful for semi-supervised learning

Why Deep Probabilistic Models for Unsupervised Learning?

- Useful in many ways
- Unsupervised feature learning from raw data (images, text, etc.)
  - These features can be useful in supervised learning tasks or for dimensionality reduction
- Good models for estimating the probability density $p(\boldsymbol{x})$ of data (akin to GMM, PPCA, etc)
  - Using deep neural nets in these models allows learning very complex densities
- Good models for generating new data from the learned model
- Useful for semi-supervised learning
- Most of these models contain latent variable but some do not (some examples in next lecture)

Some Early Architectures: Sigmoid Belief Network (Neal 1992)

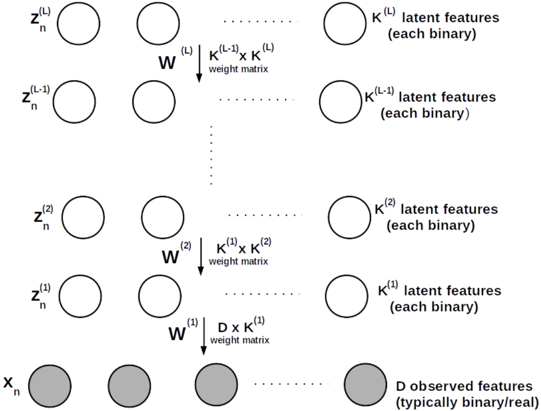
- Assumes data generated by successive nonlinear transformations of multiple layers of latent features



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

Some Early Architectures: Sigmoid Belief Network (Neal 1992)

- Assumes data generated by successive nonlinear transformations of multiple layers of latent features

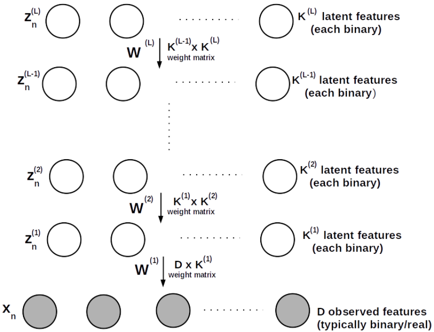


- in SBN, the latent features in each hidden layer are assumed to be binary-valued
- Unlike a Bayesian NN, this is a generative model for the data (which here are the only inputs $\boldsymbol{x}_n$'s)

Some Early Architectures: Sigmoid Belief Network (Neal 1992)

- The SBN generative model is

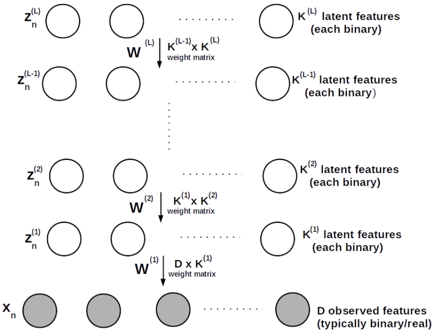
$$\begin{aligned} p(x_{nd} = 1|\boldsymbol{z}_n^{(1)}) &= \text{sigmoid}(\boldsymbol{w}_{d,:}^{(1)\top}\boldsymbol{z}_n^{(1)} + b_d) \qquad \text{(if features are binary)} \\ p(z_{nk}^{(1)} = 1|\boldsymbol{z}_n^{(2)}) &= \text{sigmoid}(\boldsymbol{w}_{k,:}^{(2)\top}\boldsymbol{z}_n^{(2)} + c_k) \end{aligned}$$



Some Early Architectures: Sigmoid Belief Network (Neal 1992)

- The SBN generative model is

$$\begin{aligned} p(x_{nd} = 1 | \boldsymbol{z}_n^{(1)}) &= \text{sigmoid}(\boldsymbol{w}_{d,:}^{(1)\top} \boldsymbol{z}_n^{(1)} + b_d) \qquad \text{(if features are binary)} \\ p(z_{nk}^{(1)} = 1 | \boldsymbol{z}_n^{(2)}) &= \text{sigmoid}(\boldsymbol{w}_{k,:}^{(2)\top} \boldsymbol{z}_n^{(2)} + c_k) \end{aligned}$$



- Not a conjugate model. Inference using MCMC (Neal, 1992[†]; Gan et al, 2015[‡])

[†] Connectionist learning of belief networks (Neal, 1992)

[‡] Learning Deep Sigmoid Belief Networks with Data Augmentation (Gan et al, 2015)

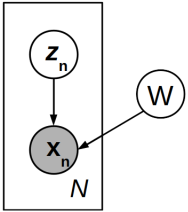
Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n | 0, \mathbf{I}_K)$$

$$p(\boldsymbol{x}_n | \boldsymbol{z}_n) = \text{ExpFam}(\boldsymbol{x}_n | \mathbf{W}\boldsymbol{z}_n)$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam



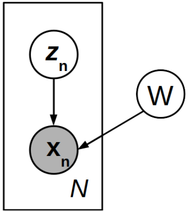
Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n | 0, \mathbf{I}_K)$$

$$p(\boldsymbol{x}_n|\boldsymbol{z}_n) = \text{ExpFam}(\boldsymbol{x}_n|\mathbf{W}\boldsymbol{z}_n)$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam



- These models are popularly known as a Latent Gaussian Models

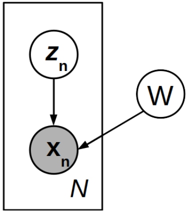
Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n | 0, \mathbf{I}_K)$$

$$p(\boldsymbol{x}_n | \boldsymbol{z}_n) = \text{ExpFam}(\boldsymbol{x}_n | \mathbf{W}\boldsymbol{z}_n)$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam



- These models are popularly known as a Latent Gaussian Models
- Many generative models have this form

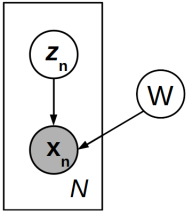
Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n | 0, \mathbf{I}_K)$$

$$p(\boldsymbol{x}_n | \boldsymbol{z}_n) = \text{ExpFam}(\boldsymbol{x}_n | \mathbf{W}\boldsymbol{z}_n)$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam



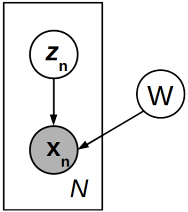
- These models are popularly known as a Latent Gaussian Models
- Many generative models have this form, e.g.,
  - Factor Analysis and Probabilistic PCA (e.g., when "Expon" = Gaussian)

Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$
\begin{aligned}
p(\boldsymbol{z}_n) &= \mathcal{N}(\boldsymbol{z}_n | 0, \mathbf{I}_K) \\
p(\boldsymbol{x}_n | \boldsymbol{z}_n) &= \text{ExpFam}(\boldsymbol{x}_n | \mathbf{W}\boldsymbol{z}_n)
\end{aligned}
$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam



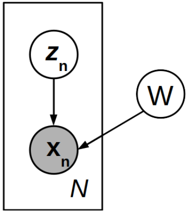
- These models are popularly known as a Latent Gaussian Models
- Many generative models have this form, e.g.,
  - Factor Analysis and Probabilistic PCA (e.g., when "Expon" $=$ Gaussian)
  - $\boldsymbol{z}_n$'s can also be defined as $p(\boldsymbol{z}_n | \boldsymbol{z}_{n-1}) = \mathcal{N}(\boldsymbol{z}_n | \mathbf{A}\boldsymbol{z}_{n-1}, \mathbf{I}_K)$ [linear dynamical systems; will see later)

Another Architecture: Latent Gaussian Models

- Observed data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$
- Assume each observation $\boldsymbol{x}_n$ to be generated by a Gaussian latent variable $\boldsymbol{z}_n \in \mathbb{R}^K$ as follows

$$p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K)$$
$$p(\boldsymbol{x}_n|\boldsymbol{z}_n) = \text{ExpFam}(\boldsymbol{x}_n|\mathbf{W}\boldsymbol{z}_n)$$

$\mathbf{W}\boldsymbol{z}_n$: Natural parameters of exp-fam

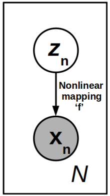


- These models are popularly known as a Latent Gaussian Models
- Many generative models have this form, e.g.,
  - Factor Analysis and Probabilistic PCA (e.g., when "Expon" = Gaussian)
  - $\boldsymbol{z}_n$'s can also be defined as $p(\boldsymbol{z}_n|\boldsymbol{z}_{n-1}) = \mathcal{N}(\boldsymbol{z}_n|\mathbf{A}\boldsymbol{z}_{n-1}, \mathbf{I}_K)$ [linear dynamical systems; will see later)
- A key limitation: $\boldsymbol{z}_n$ to $\boldsymbol{x}_n$ mapping is **linear** (note that the natural params = $\mathbf{W}\boldsymbol{z}_n$)

Nonlinear Latent Gaussian Models

- Let's assume $\boldsymbol{z}_n$ goes through a nonlinear transformation f before it generates $\boldsymbol{x}_n$

$$
\begin{aligned}
\boldsymbol{z}_n &\sim \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K) \\
\boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|f(\boldsymbol{z}_n))
\end{aligned}
$$

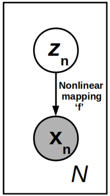


- We now have a Nonlinear version of the Latent Gaussian Model

Nonlinear Latent Gaussian Models

- Let's assume $\boldsymbol{z}_n$ goes through a nonlinear transformation f before it generates $\boldsymbol{x}_n$

$$\begin{aligned}\boldsymbol{z}_n &\sim \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K) \\ \boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|f(\boldsymbol{z}_n))\end{aligned}$$

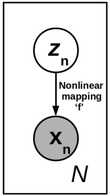


- We now have a Nonlinear version of the Latent Gaussian Model
- How should we model the nonlinear mapping f?

Nonlinear Latent Gaussian Models

- Let's assume $\boldsymbol{z}_n$ goes through a nonlinear transformation f before it generates $\boldsymbol{x}_n$

$$\begin{aligned} \boldsymbol{z}_n &\sim \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K) \\ \boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|f(\boldsymbol{z}_n)) \end{aligned}$$

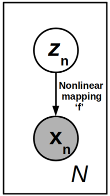


- We now have a Nonlinear version of the Latent Gaussian Model
- How should we model the nonlinear mapping f? Some options
  - Model f using a Gaussian Process (recall GPLVM)

Nonlinear Latent Gaussian Models

- Let's assume $\boldsymbol{z}_n$ goes through a nonlinear transformation f before it generates $\boldsymbol{x}_n$

$$\begin{aligned}\boldsymbol{z}_n &\sim \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K) \\ \boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|f(\boldsymbol{z}_n))\end{aligned}$$



- We now have a Nonlinear version of the Latent Gaussian Model
- How should we model the nonlinear mapping f? Some options
  - Model f using a Gaussian Process (recall GPLVM)
  - Model f using a Deep Neural Network (we will look at these)

Nonlinear Latent Gaussian Models

- Assume that a Gaussian drawn $\boldsymbol{z}_n$ is pushed through an NN to generate $\boldsymbol{x}_n$ (Deep LGM)

$$\boldsymbol{z}_n \rightarrow \boldsymbol{h}_n^{(L)} \rightarrow \boldsymbol{h}_n^{(L-1)} \ldots \boldsymbol{h}_n^{(2)} \rightarrow \boldsymbol{h}_n^{(1)} \rightarrow \boldsymbol{x}_n$$

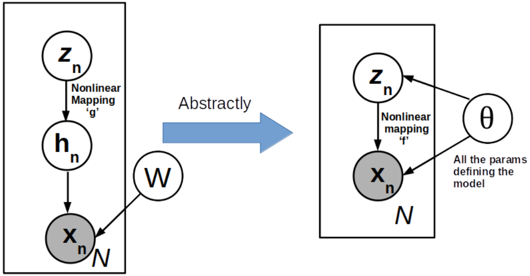
- L denotes the number of "layers" in the model

Nonlinear Latent Gaussian Models

- Assume that a Gaussian drawn $\boldsymbol{z}_n$ is pushed through an NN to generate $\boldsymbol{x}_n$ (Deep LGM)

$$\boldsymbol{z}_n \rightarrow \boldsymbol{h}_n^{(L)} \rightarrow \boldsymbol{h}_n^{(L-1)} \ldots \boldsymbol{h}_n^{(2)} \rightarrow \boldsymbol{h}_n^{(1)} \rightarrow \boldsymbol{x}_n$$

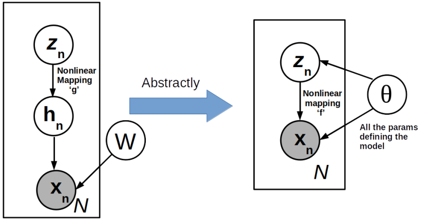
- L denotes the number of "layers" in the model. For $L = 1$, the model looks like this



$$\begin{aligned}
\boldsymbol{z}_n &\sim \mathcal{N}(\boldsymbol{z}_n|0, \mathbf{I}_K) \\
\boldsymbol{h}_n &= g(\boldsymbol{z}_n) \quad \text{(deterministic nonlinear mapping)} \\
\boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|\mathbf{W}\boldsymbol{h}_n) \\
\boldsymbol{x}_n &\sim \text{ExpFam}(\boldsymbol{x}_n|f(\boldsymbol{z}_n)) \quad \text{(Equivalent to the above)}
\end{aligned}$$

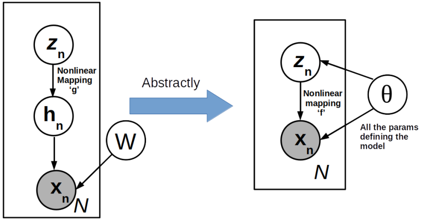
Inference for Deep Latent Gaussian Models

- Assume a prior $p_\theta(\boldsymbol{z})$ and likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ for this DLGM
- Let's assume we are doing only point estimation for θ (but full posterior for $\boldsymbol{z}$)



Inference for Deep Latent Gaussian Models

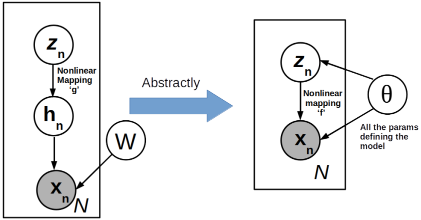
- Assume a prior $p_\theta(\boldsymbol{z})$ and likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ for this DLGM
- Let's assume we are doing only point estimation for θ (but full posterior for $\boldsymbol{z}$)



- Assume the prior $p_\theta(\boldsymbol{z})$ to be Gaussian $\mathcal{N}(\boldsymbol{z}|0, \mathbf{I}_K)$

Inference for Deep Latent Gaussian Models

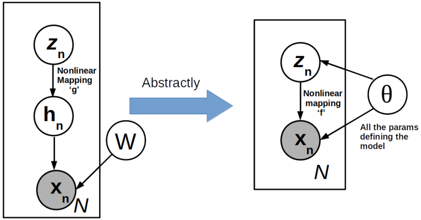
- Assume a prior $p_\theta(\boldsymbol{z})$ and likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ for this DLGM
- Let's assume we are doing only point estimation for θ (but full posterior for $\boldsymbol{z}$)



- Assume the prior $p_\theta(\boldsymbol{z})$ to be Gaussian $\mathcal{N}(\boldsymbol{z}|0, \mathbf{I}_K)$
- Assume the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ to be of the form $\mathcal{N}(\boldsymbol{x}|f(\boldsymbol{z}), \sigma^2\mathbf{I})$ with f being an NN

Inference for Deep Latent Gaussian Models

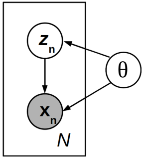
- Assume a prior $p_\theta(\boldsymbol{z})$ and likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ for this DLGM
- Let's assume we are doing only point estimation for θ (but full posterior for $\boldsymbol{z}$)



- Assume the prior $p_\theta(\boldsymbol{z})$ to be Gaussian $\mathcal{N}(\boldsymbol{z}|0, \mathbf{I}_K)$
- Assume the likelihood $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ to be of the form $\mathcal{N}(\boldsymbol{x}|f(\boldsymbol{z}), \sigma^2\mathbf{I})$ with f being an NN
- Given N observations $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the goal will be to infer $\mathbf{Z} = \{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ. In general, this will be an intractable problem (just like other Bayesian models).

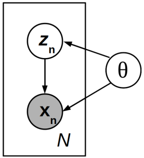
Inference for Deep Latent Gaussian Models

- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



Inference for Deep Latent Gaussian Models

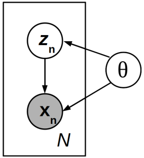
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow

Inference for Deep Latent Gaussian Models

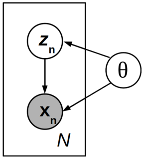
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate

Inference for Deep Latent Gaussian Models

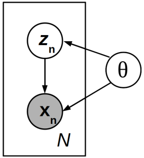
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities

Inference for Deep Latent Gaussian Models

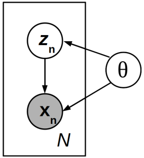
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used

Inference for Deep Latent Gaussian Models

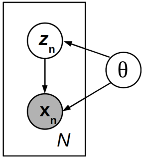
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$

Inference for Deep Latent Gaussian Models

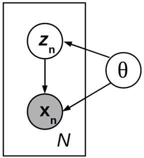
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$

Inference for Deep Latent Gaussian Models

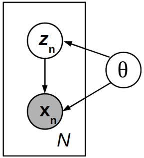
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$
  - Note: SVI can help (but only so much..)

Inference for Deep Latent Gaussian Models

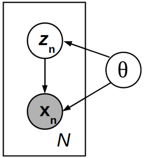
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$
  - Note: SVI can help (but only so much..)
  - Also, the process of inferring $q_{\phi_n}(\boldsymbol{z}_n)$ in VB (and also MCMC) is iterative.

Inference for Deep Latent Gaussian Models

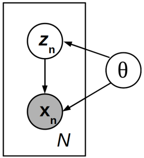
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$
  - Note: SVI can help (but only so much..)
  - Also, the process of inferring $q_{\phi_n}(\boldsymbol{z}_n)$ in VB (and also MCMC) is iterative. Can't get a fast prediction of $\boldsymbol{z}_n$.

Inference for Deep Latent Gaussian Models

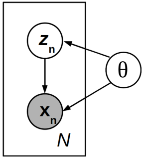
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$
  - Note: SVI can help (but only so much..)
  - Also, the process of inferring $q_{\phi_n}(\boldsymbol{z}_n)$ in VB (and also MCMC) is iterative. Can't get a fast prediction of $\boldsymbol{z}_n$. Ideally, we would like to predict $\boldsymbol{z}_n$ fast for any given $\boldsymbol{x}_n$

Inference for Deep Latent Gaussian Models

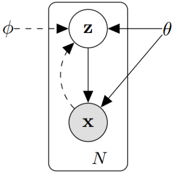
- The goal: Given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, infer $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ and θ



- MCMC can be applied in practice but tends to be slow
- VB can be faster but difficult to work out the ELBO if $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ aren't conjugate
  - This usually is the case since $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is defined by a complex NN with nonlinearities
  - BBVI or other VB methods based on Monte-Carlo approximation used
- VB (and MCMC too) has another issue: Need to infer a $q_{\phi_n}(\boldsymbol{z}_n)$ for each $\boldsymbol{z}_n$
  - Very slow if N is large because we need to infer a large number of $q_{\phi_n}(\boldsymbol{z}_n)$, $n = 1, \ldots, N$
  - Note: SVI can help (but only so much..)
  - Also, the process of inferring $q_{\phi_n}(\boldsymbol{z}_n)$ in VB (and also MCMC) is iterative. Can't get a fast prediction of $\boldsymbol{z}_n$. Ideally, we would like to predict $\boldsymbol{z}_n$ fast for any given $\boldsymbol{x}_n$ (e.g., $\boldsymbol{z}_*$ for a new test input $\boldsymbol{x}_*$)

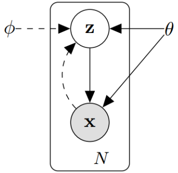
Variational Auto-encoder (VAE)

- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



Variational Auto-encoder (VAE)

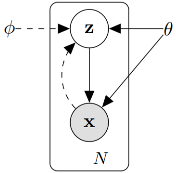
- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model

Variational Auto-encoder (VAE)

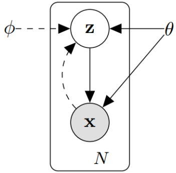
- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model
- The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ serves as a "decoder" model (given $\boldsymbol{z}$, decode $\boldsymbol{x}$)

Variational Auto-encoder (VAE)

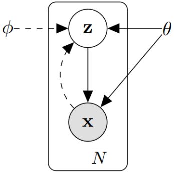
- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model
- The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ serves as a "decoder" model (given $\boldsymbol{z}$, decode $\boldsymbol{x}$)
- The goal is to learn both the encoder and decoder models (basically, their parameters ϕ and θ)

Variational Auto-encoder (VAE)

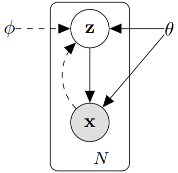
- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model
- The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ serves as a "decoder" model (given $\boldsymbol{z}$, decode $\boldsymbol{x}$)
- The goal is to learn both the encoder and decoder models (basically, their parameters ϕ and θ)
- Note: No need to do inference for $\boldsymbol{z}_n$. Can compute $\boldsymbol{z}_n$ directly by plugging in $\boldsymbol{x}_n$ in q_ϕ

Variational Auto-encoder (VAE)

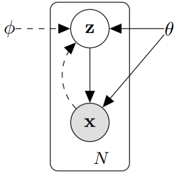
- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model
- The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ serves as a "decoder" model (given $\boldsymbol{z}$, decode $\boldsymbol{x}$)
- The goal is to learn both the encoder and decoder models (basically, their parameters ϕ and θ)
- Note: No need to do inference for $\boldsymbol{z}_n$. Can compute $\boldsymbol{z}_n$ directly by plugging in $\boldsymbol{x}_n$ in q_ϕ
    - This approach of inferring $\boldsymbol{z}_n$ also is called a "recognition model"

Variational Auto-encoder (VAE)

- A DLGM with an encoder-decoder mechanism (we can think of the latent $\boldsymbol{z}$ as an encoding of $\boldsymbol{x}$). Proposed almost simultaneously by Kingma & Welling (2013), and Rezende *et al* (2014)



- Idea: Instead of approximating each posterior $p(\boldsymbol{z}_n|\boldsymbol{x}_n)$ as $q_{\phi_n}(\boldsymbol{z}_n)$, approximate it by a single global function $q_\phi(\boldsymbol{z}_n|\boldsymbol{x}_n)$ defined by a neural net with parameters ϕ. Called the "encoder" model
- The generative model $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ serves as a "decoder" model (given $\boldsymbol{z}$, decode $\boldsymbol{x}$)
- The goal is to learn both the encoder and decoder models (basically, their parameters ϕ and θ)
- Note: No need to do inference for $\boldsymbol{z}_n$. Can compute $\boldsymbol{z}_n$ directly by plugging in $\boldsymbol{x}_n$ in q_ϕ
    - This approach of inferring $\boldsymbol{z}_n$ also is called a "recognition model"
- More details in the next class