CS11-711 Advanced NLP

Models w/ Latent Random Variables

Daniel Fried

Carnegie Mellon University
Language Technologies Institute

Site
https://cmu-anlp.github.io/

With Slides from Graham Neubig, Chunting Zhou, and Paul Liang

Discriminative vs. Generative Models

- **Discriminative model:** calculate the probability of output given input P(Y|X)

- **Generative model:** calculate the probability of a variable P(X), or multiple variables P(X,Y)

- Which of the following models are discriminative vs. generative?
  - BERT-based POS tagger with independent predictions
  - Encoder-decoder based POS tagger
  - Language model

Types of Variables

- Observed vs. Latent:
  - **Observed:** something that we can see from our data, e.g. X or Y
  - **Latent:** a variable that we assume exists, but we aren't given the value
- Deterministic vs. Random:
  - **Deterministic:** variables that are calculated directly according to some deterministic function
  - **Random (stochastic):** variables that obey a probability distribution, and may take any of several (or infinite) values

Quiz: What Types of Variables?

- In the an attentional sequence-to-sequence model using MLE/teacher forcing, are the following variables observed or latent? deterministic or random?
  - The input word ids **f**
  - The encoder hidden states **h**
  - The attention values **a**
  - The output word ids **e**

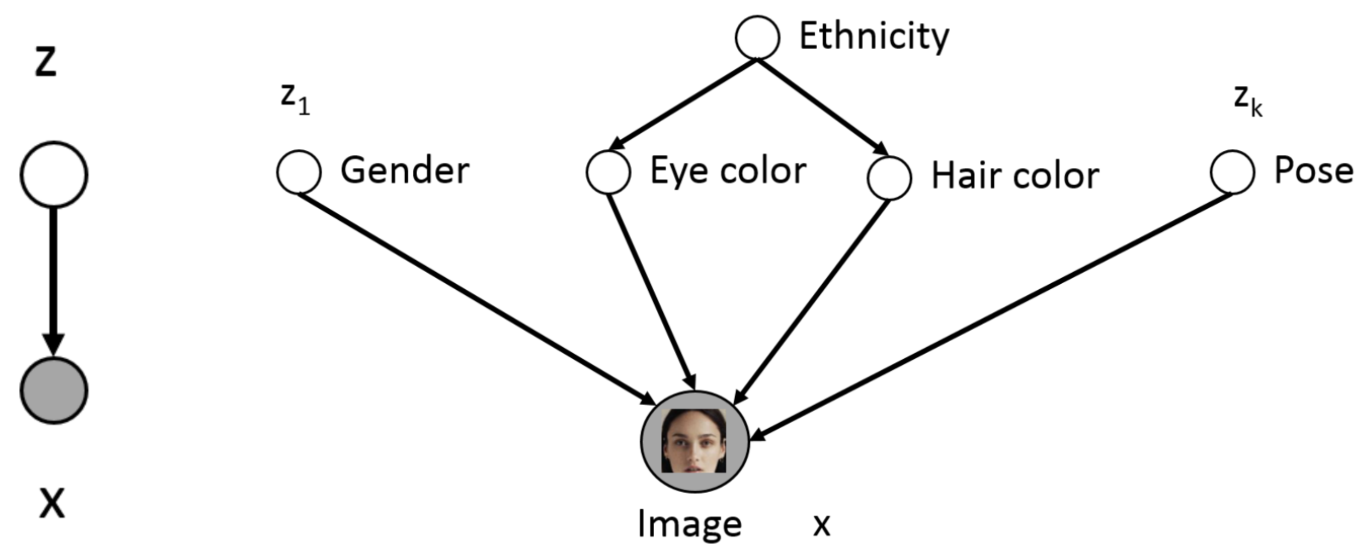
Latent Variable Models

- A latent variable model (LVM) is a probability distribution over two sets of variables x, z :

$$p(x, z; \theta)$$

where the x variables are observed at learning time in a dataset and z are latent variables.

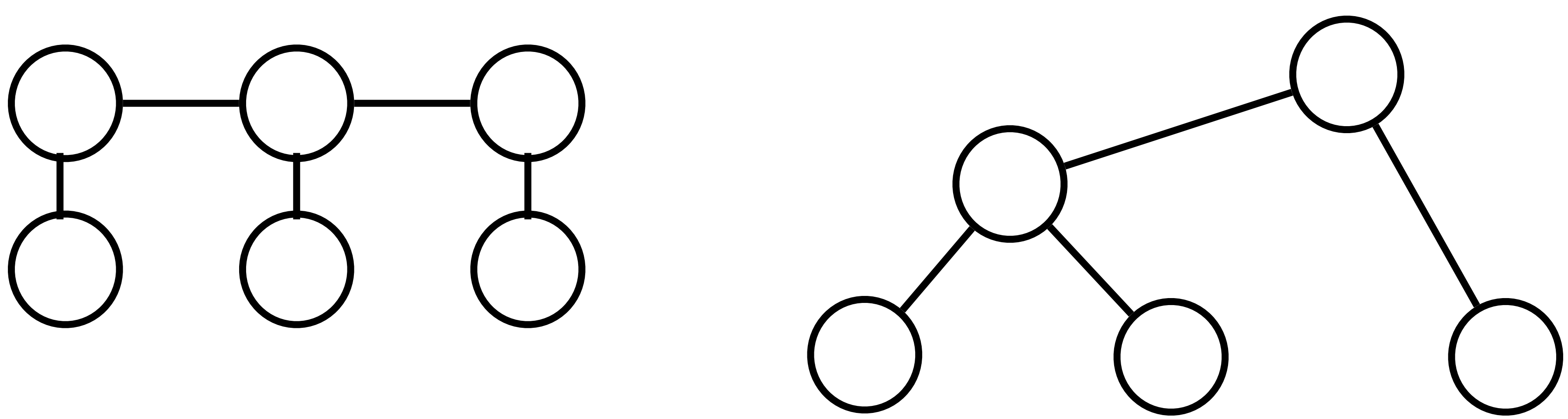
Latent Variable Models



- Only shaded variables x are observed in the data
- Latent variables z are unobserved
    - We want z to represent useful features e.g. for hair color, pose, etc.
    - But very difficult to specify these by hand and they're unobserved
    - Let's learn them instead

What Types of Latent Variables?

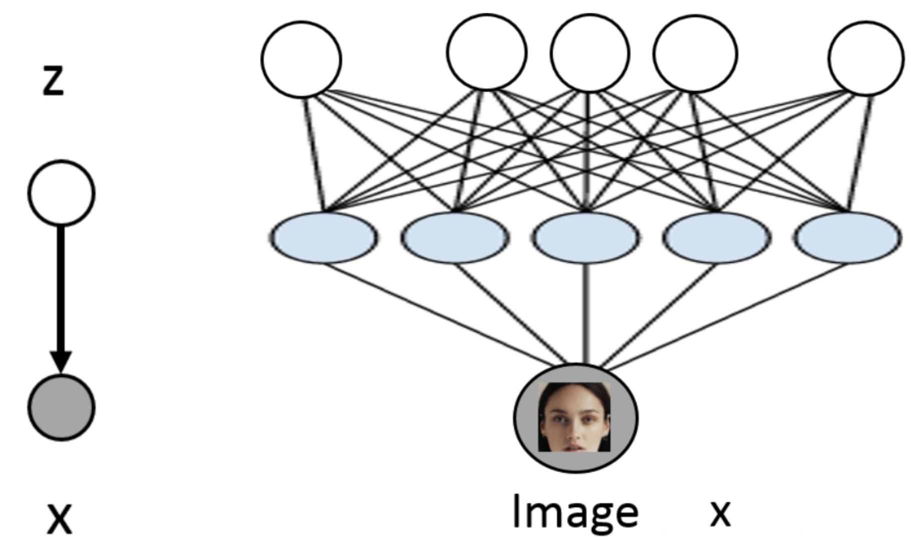
- Latent continuous vector (e.g. variational auto-encoder)
- Latent discrete vector (e.g. topic model)
- Latent structure (e.g. HMM or tree-structured model)



Variational Auto-encoders

(Kingma and Welling 2014)

Latent Variable Models

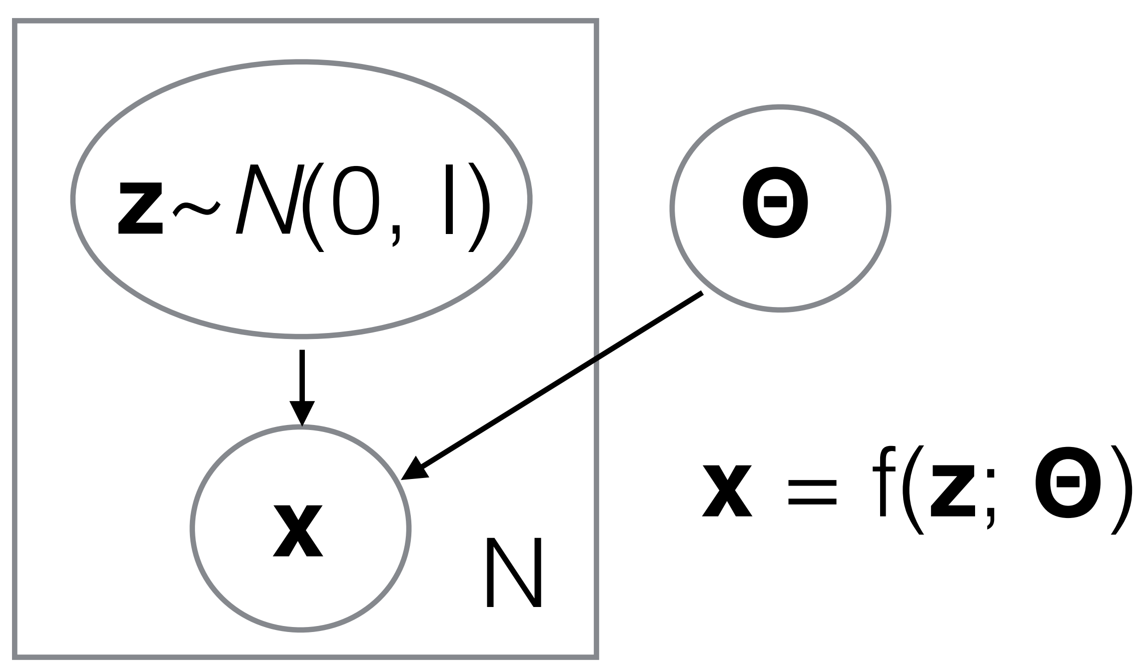


- Put a prior on z $\mathbf{z} \sim \mathcal{N}(0, I)$

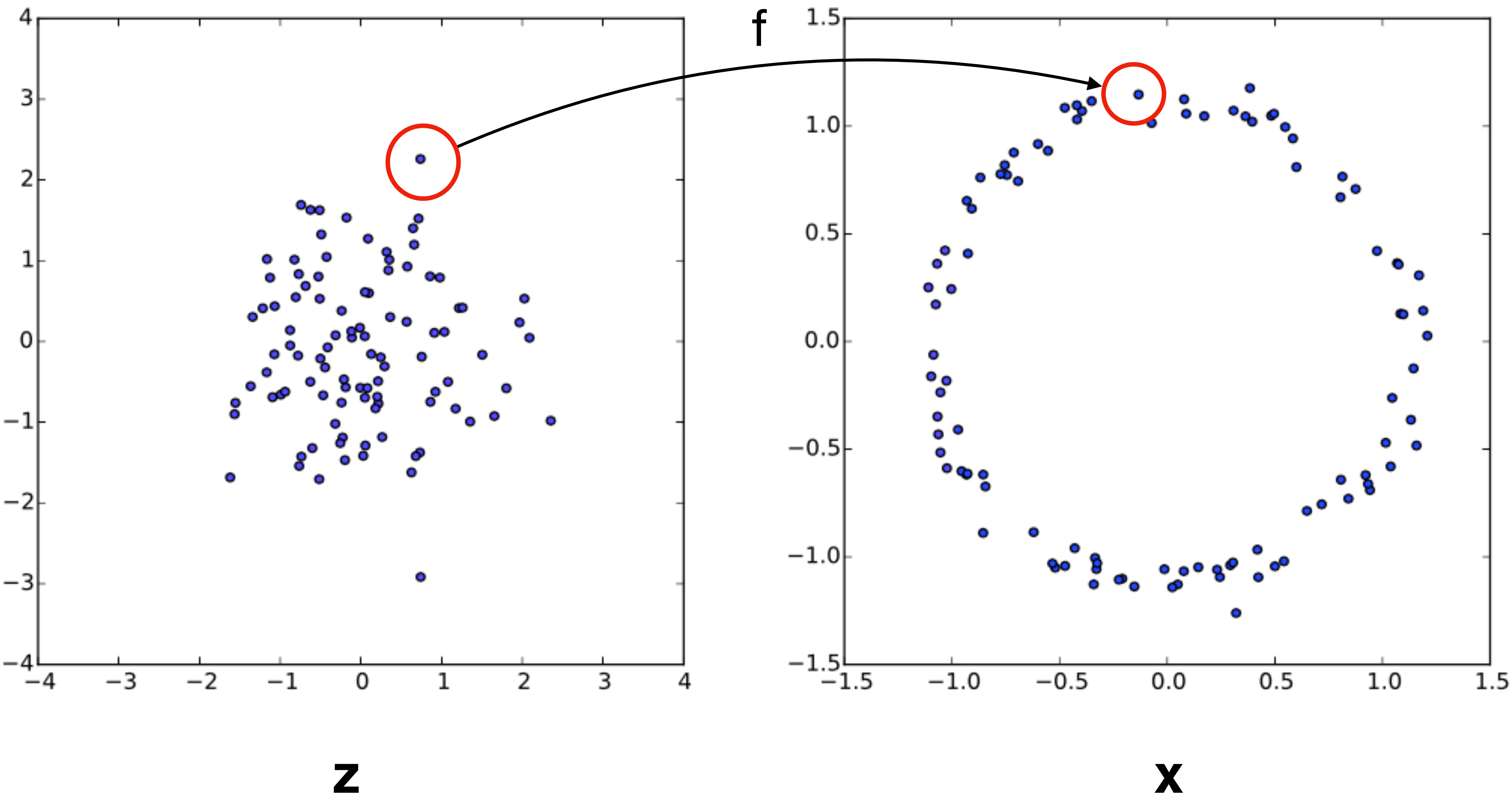
  $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}\left(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z})\right)$ where μ_θ,Σ_θ are neural networks
- Hope that after training, z will correspond to meaningful latent factors of variation
- Given a random z, a new x can be generated => control; if z is interpretable

A Latent Variable Model

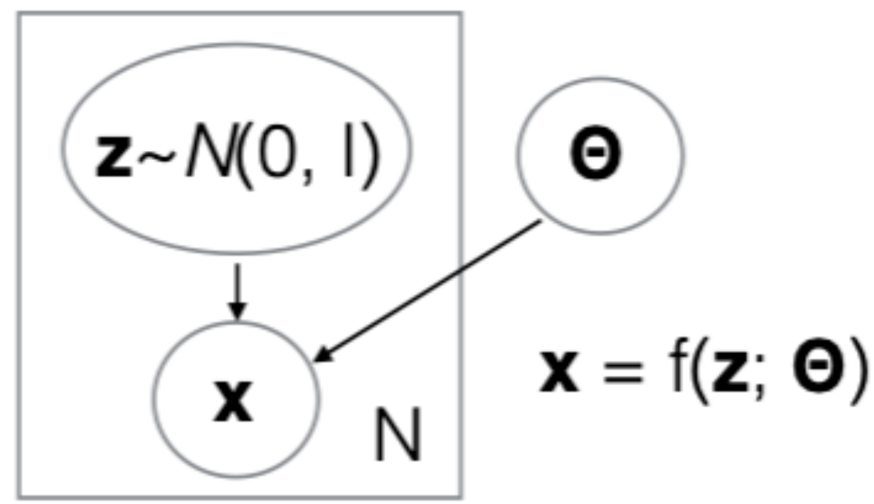
- We observed output **x** (assume a continuous vector for now)
- We have a latent variable **z** generated from a Gaussian
- We have a function f, parameterized by **Θ** that maps from **z** to **x**, where this function is usually a neural net



An Example (Goersch 2016)



A probabilistic perspective on Variational Auto-Encoder



- For each datapoint i :
  - Draw latent variables $z_i \sim p(z)$ (prior)
  - Draw data point $x_i \sim p_\theta(x|z)$
- Joint probability distribution over data and latent variables:

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

What is Our Loss Function?

- We would like to maximize the corpus log likelihood

$$\log P(\mathcal{X}) = \sum_{\boldsymbol{x} \in \mathcal{X}} \log P(\boldsymbol{x}; \theta)$$

- For a single example, the marginal likelihood is

$$P(\boldsymbol{x}; \theta) = \int P(\boldsymbol{x} \mid \boldsymbol{z}; \theta) P(\boldsymbol{z}) d\boldsymbol{z}$$

- We can approximate this by sampling **z**s then summing

$$P(\boldsymbol{x}; \theta) \approx \frac{1}{N} \sum_{\boldsymbol{z} \in S(\boldsymbol{x})} P(\boldsymbol{x} | \boldsymbol{z}; \theta) \quad \text{where} \quad S(\boldsymbol{x}) := \{\boldsymbol{z}'; \boldsymbol{z}' \sim P(\boldsymbol{z})\}$$

$$N := |S(\mathbf{x})|$$

Variational Inference

Two tasks of interest:

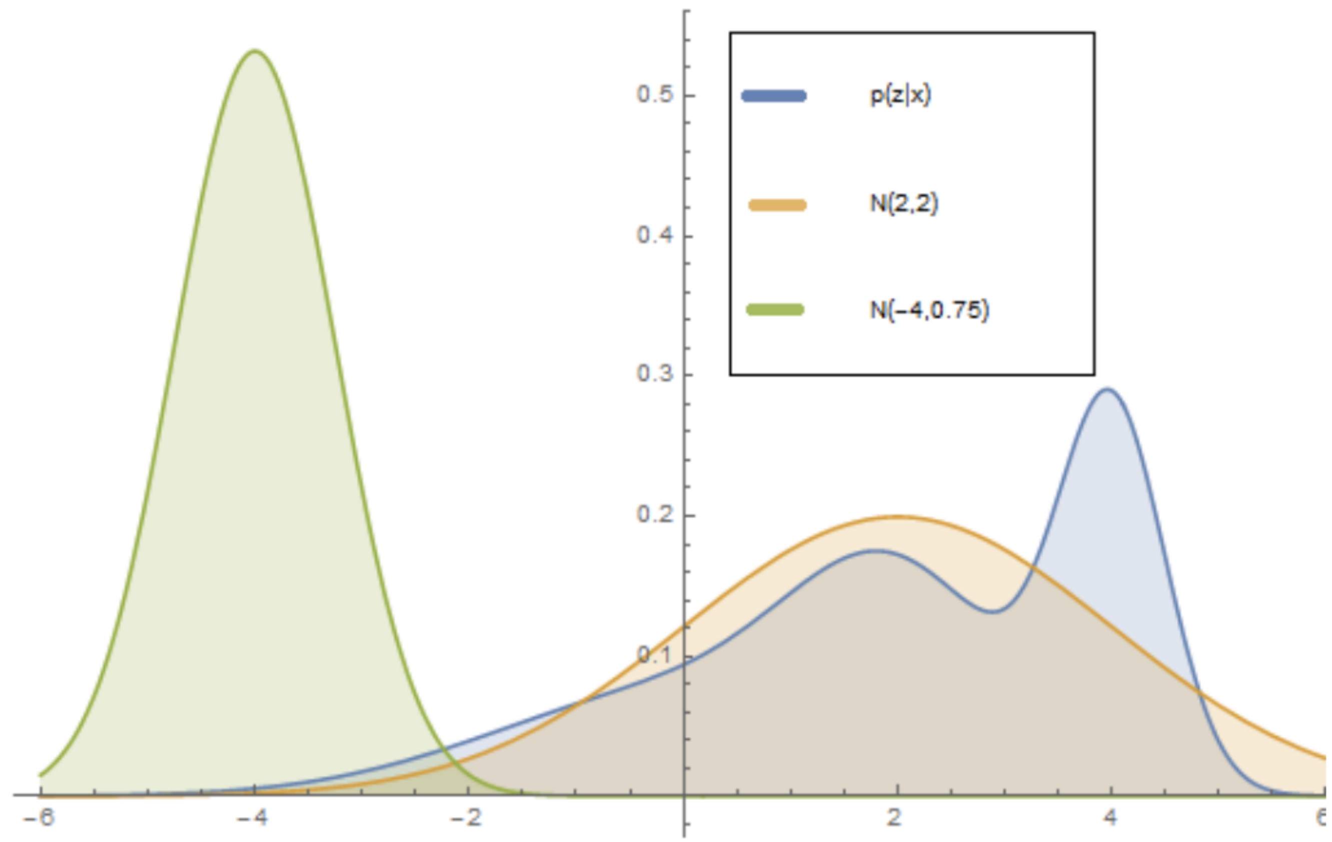
- Learn the parameters θ of $p_\theta(x|z)$
- Inference over z with the *posterior* distribution: $p_\theta(z|x)$ given input x, what are its latent factors?

$$p_\theta(z|x) = \frac{p_\theta(x|z)p(z)}{p(x)}$$

$$p(x) = \int p(z)p_\theta(x|z)dz$$ <- intractable

- Variational inference approximates the posterior with another model $q_\phi(z|x)$ that's easier to compute

Variational Inference



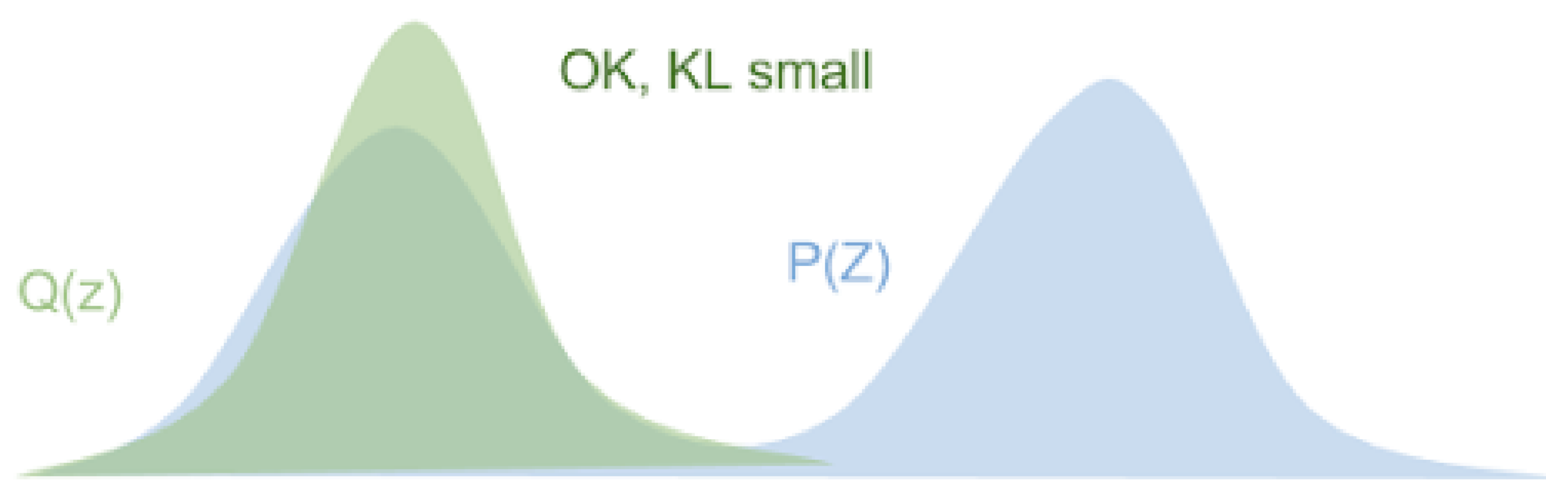
- Variational inference: optimize variational parameters so that $q(\mathbf{z};\phi)$ is as close as possible to $p(\mathbf{z}|\mathbf{x};\theta)$ while being simple to compute
- E.g. in figure, posterior (in blue) is better approximated by orange Gaussian than green

Variational Inference

- The KL divergence for variational inference is:

$$\mathbf{D}_{KL}(q(z) \| p(z|x)) = \int q(z) \log \frac{q(z)}{p(z|x)} dz$$

- Intuitively, there are three cases
    a. If q is low then we don't care (because of the expectation).
    b. If q is high and p is high then we are happy.
    c. If q is high and p is low then we pay a price.
- Note that p must be > 0 wherever q > 0



Variational Inference

- Variational inference approximates the true posterior $p_\theta(z|x)$ with an "inference" model $q_\phi(z|x)$

$$\text{minimize}: \ \mathrm{KL}(q_\phi(z|x)||p_\theta(z|x))$$

- Variational Lower Bound (ELBO)

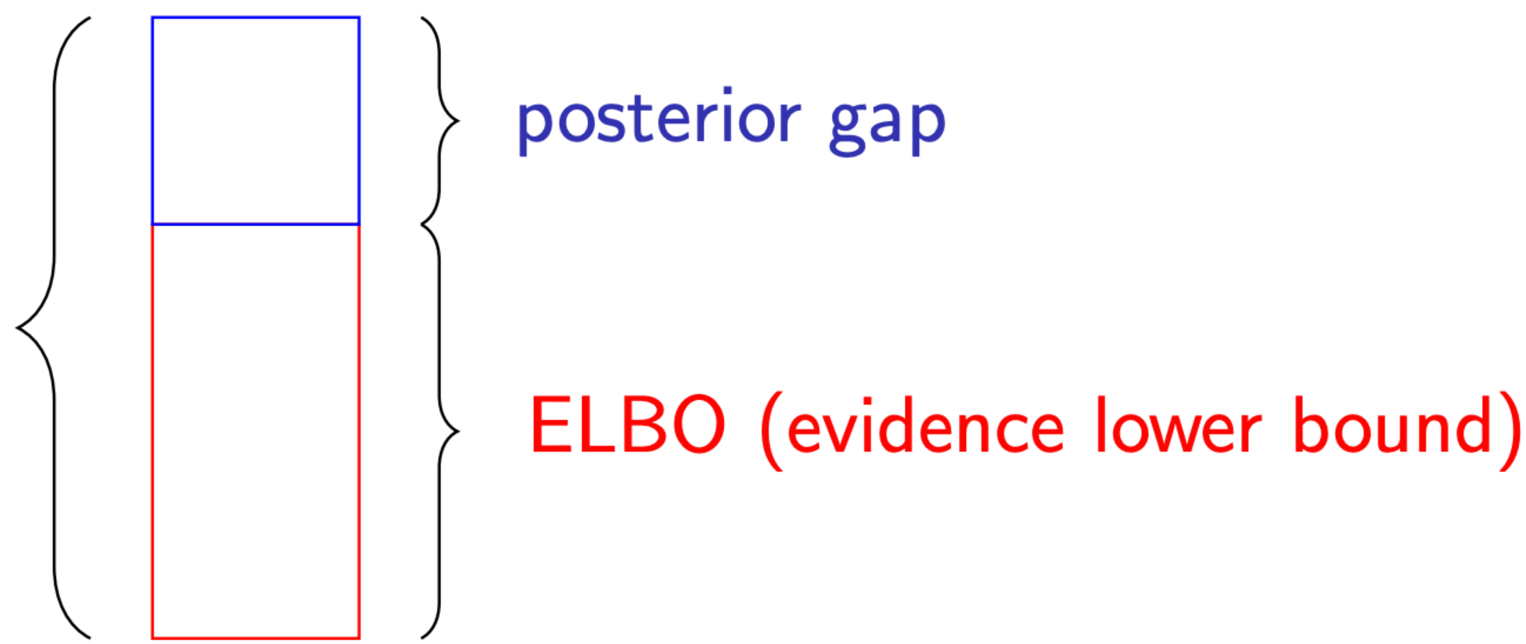
$$\log p(x) = \mathrm{ELBO} + \underbrace{\mathrm{KL}(q_\phi(z|x)||p_\theta(z|x))}_{\text{"posterior gap"}}$$

$$\mathrm{KL}(q||p) \geq 0 => \log p(x) \geq \mathrm{ELBO}$$

Variational Inference

For any[1] distribution $q(z \mid x; \lambda)$ over z,

$$\log \prod_{\mathbf{x} \in \mathcal{D}} p(\mathbf{x}; \theta) = L(\theta) = \mathbb{E}_q \left[\log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)} \right] + \mathrm{KL}[q(z \mid x; \lambda) \,\|\, p(z \mid x; \theta)]$$



Since KL is always non-negative, $L(\theta) \geq \mathrm{ELBO}(\theta, \lambda)$.

[1] Technical condition: $\mathrm{supp}(q(z)) \subset \mathrm{supp}(p(z \mid x; \theta))$

Slides from Kim, Wiseman, and Rush, https://nlp.seas.harvard.edu/latent-nlp-tutorial.html

Variational Inference

$$
\begin{aligned}
\log p(x;\,\theta) &= \mathbb{E}_q \log p(x) \quad \textit{(Expectation over } z\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x, z)}{p(z \,|\, x)} \quad \textit{(Mult/div by } p(z|x)\textit{, combine numerator)} \\
&= \mathbb{E}_q \log \left(\frac{p(x, z)}{q(z \,|\, x)} \frac{q(z \,|\, x)}{p(z \,|\, x)} \right) \quad \textit{(Mult/div by } q(z|x)\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x, z)}{q(z \,|\, x)} + \mathbb{E}_q \log \frac{q(z \,|\, x)}{p(z \,|\, x)} \quad \textit{(Split Log)} \\
&= \underbrace{\mathbb{E}_q \log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)}}_{\text{Evidence Lower Bound (ELBO)}} + \underbrace{\mathrm{KL}[q(z \,|\, x;\, \lambda) \,\|\, p(z \,|\, x;\, \theta)]}_{\text{posterior gap}}
\end{aligned}
$$

We'll choose q, and parameters θ, λ, to maximize this

Typically uncomputable, but hopefully small if we chose q well

q: like an encoder network: data -> latent

Huge number of algorithms from choices of q and decompositions of ELBO

We'll ignore this term from now on

Variational Autoencoders

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathrm{ELBO}$$

$$\mathrm{ELBO} = \mathbb{E}_{z \sim q_\phi(z|x)} \log \frac{p_\theta(x, z)}{q_\phi(z \mid x)}$$

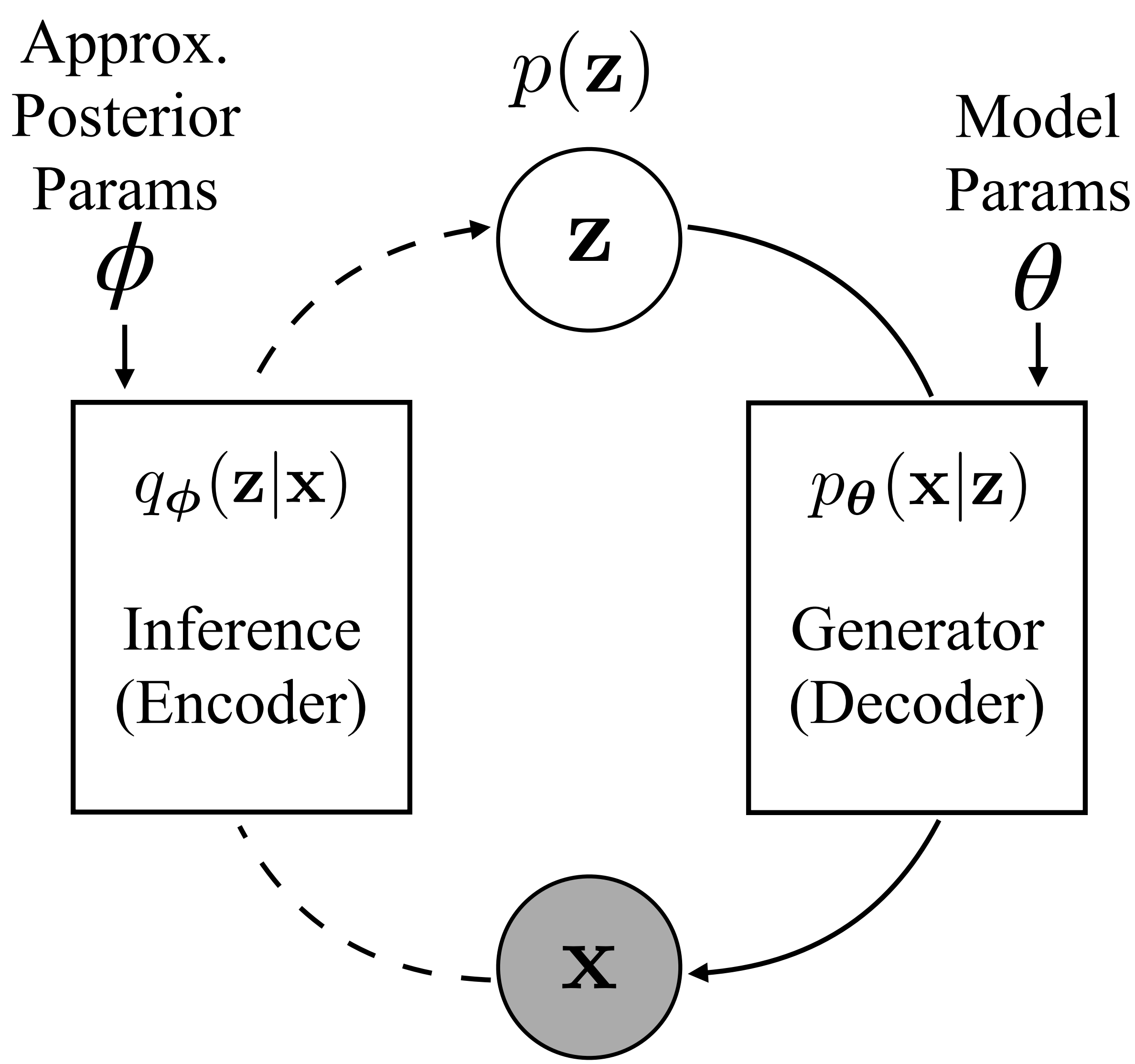
$$= \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))}_{\text{KL Regularizer}}$$

Reconstruct the input from a “hint”, z

Prior prevents hint from being too informative

Variational Autoencoders

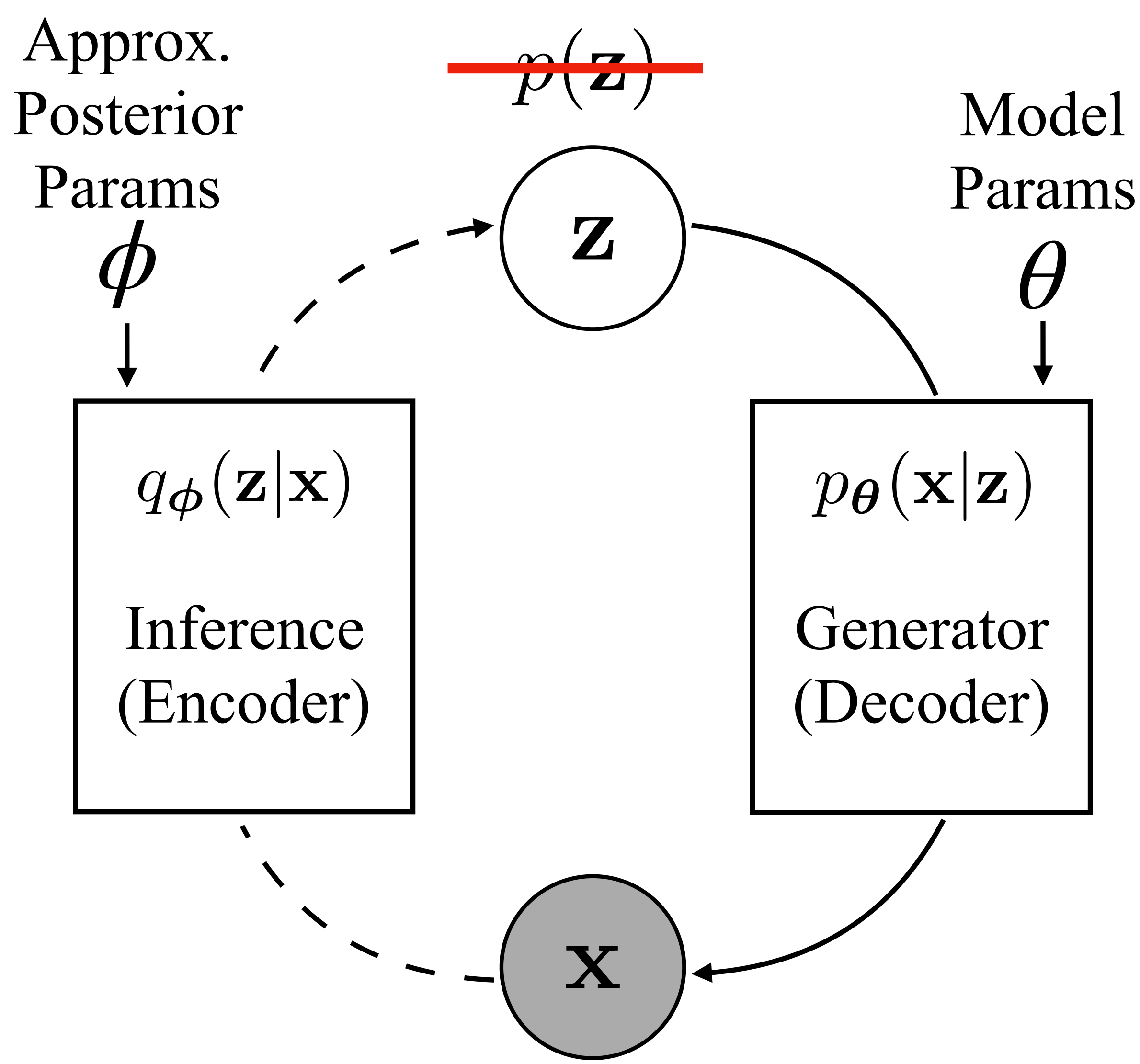
$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) >= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Regularizer}}$$



Regularized Autoencoder

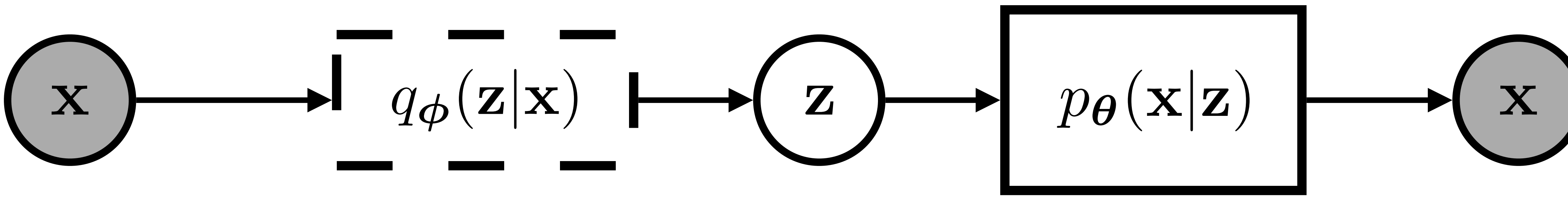
Why prior ?

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) >= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} \cancel{- \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Regularizer}}}$$



VAE vs. AE

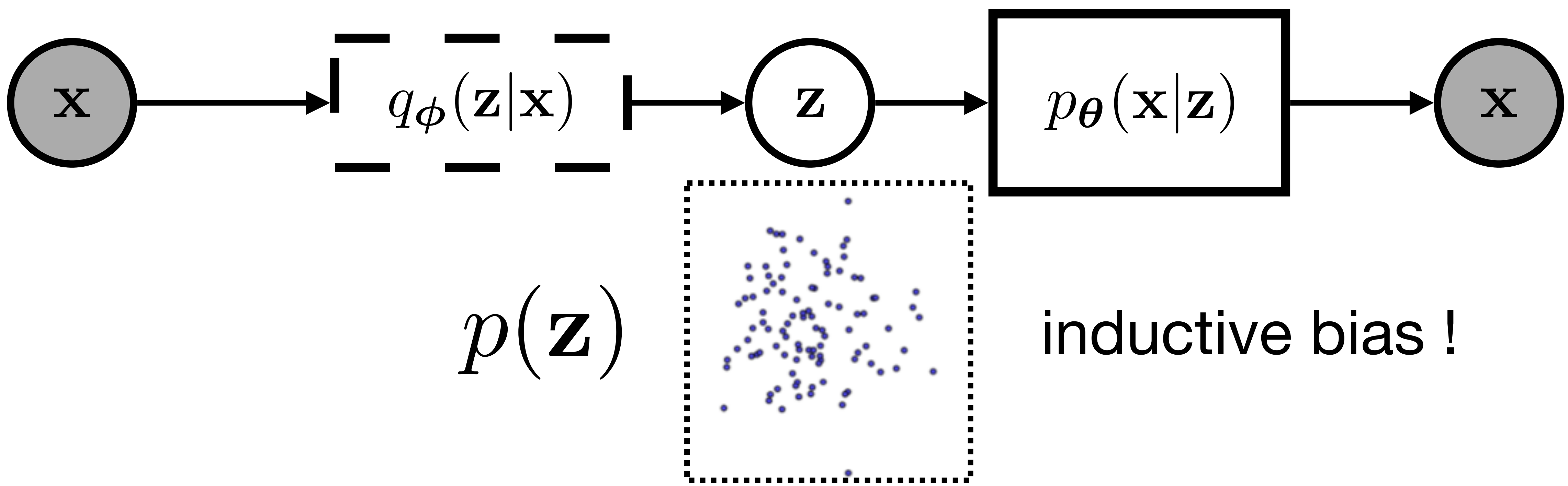
VAE



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

VAE vs. AE

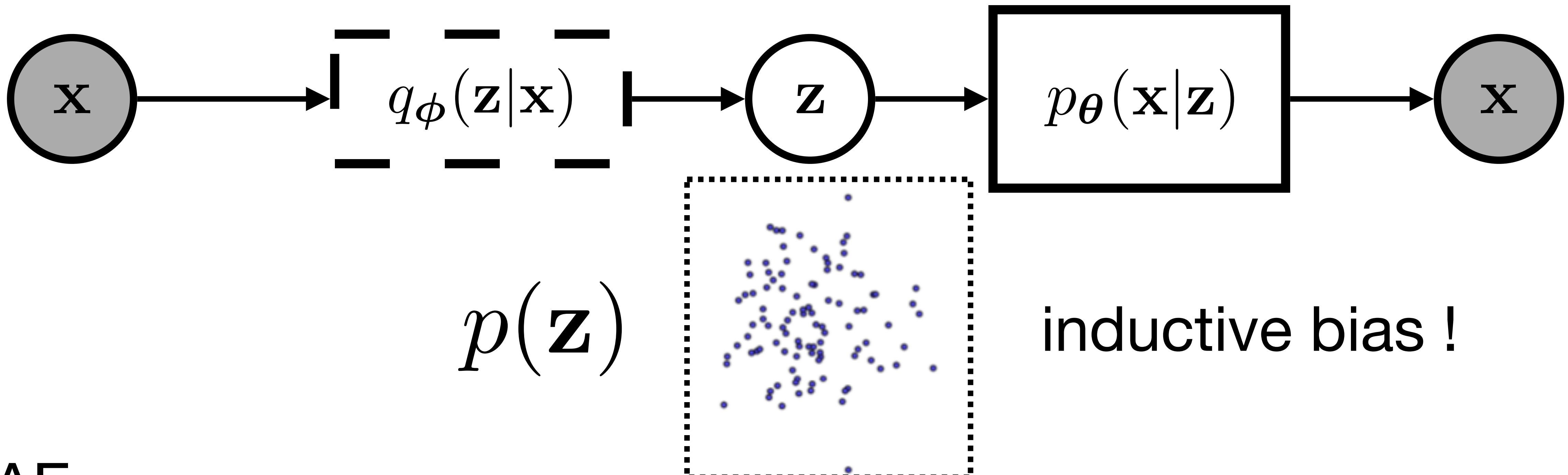
VAE



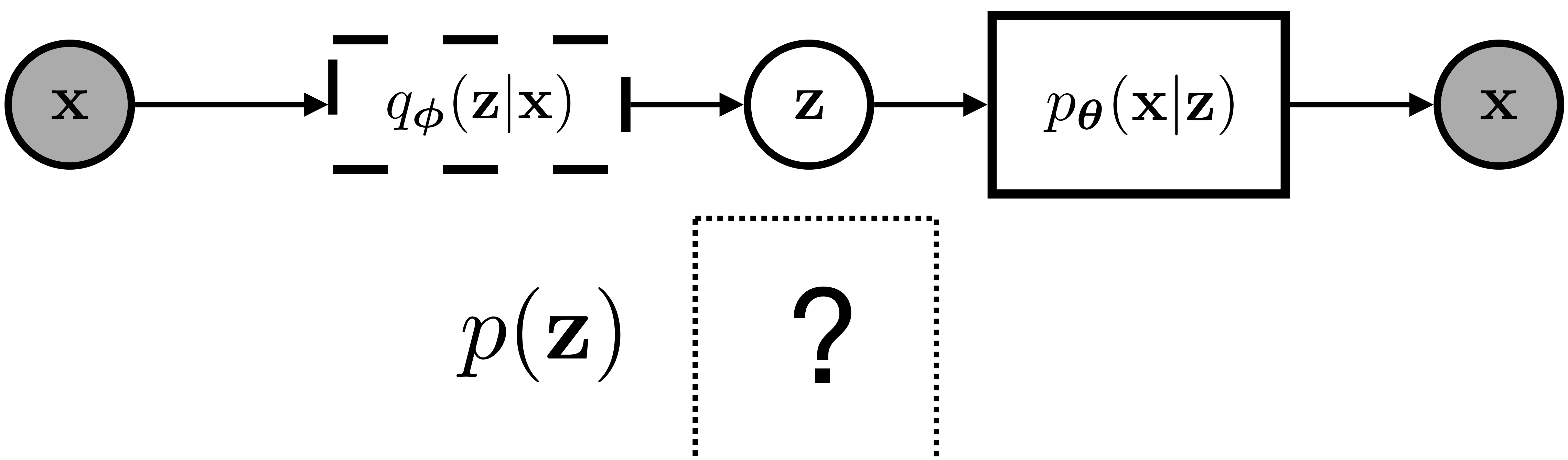
inductive bias !

VAE vs. AE

VAE

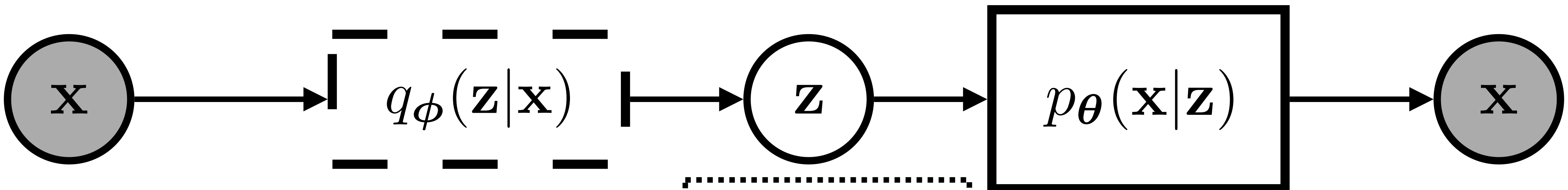


AE

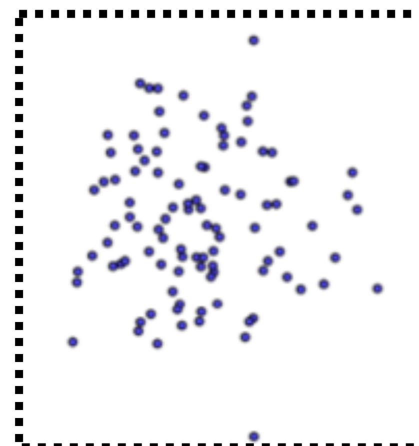


VAE vs. AE

VAE

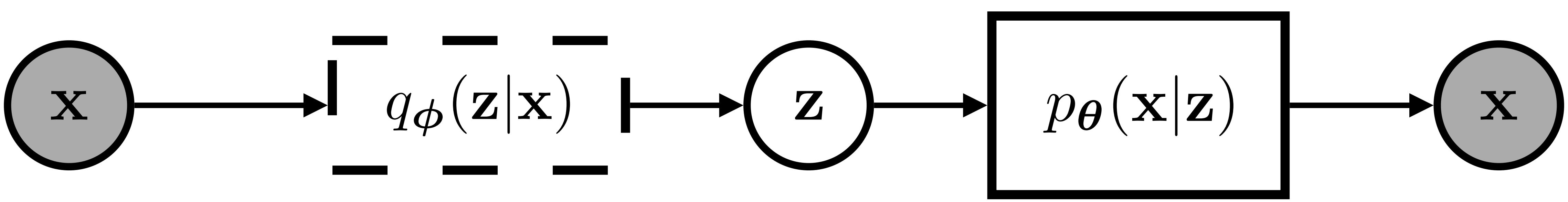


$$p(\mathbf{z})$$

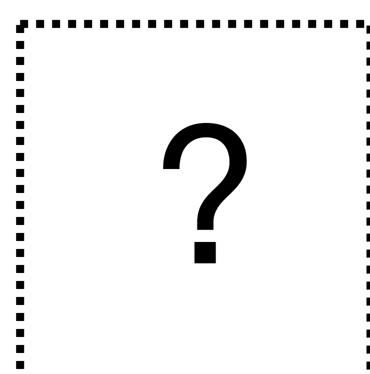


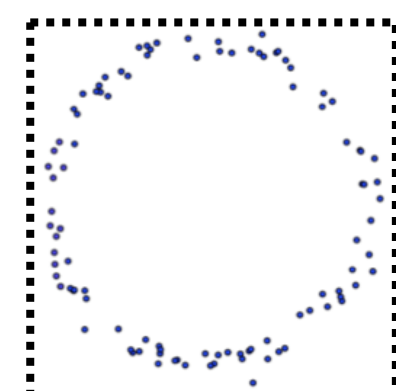
inductive bias !

AE



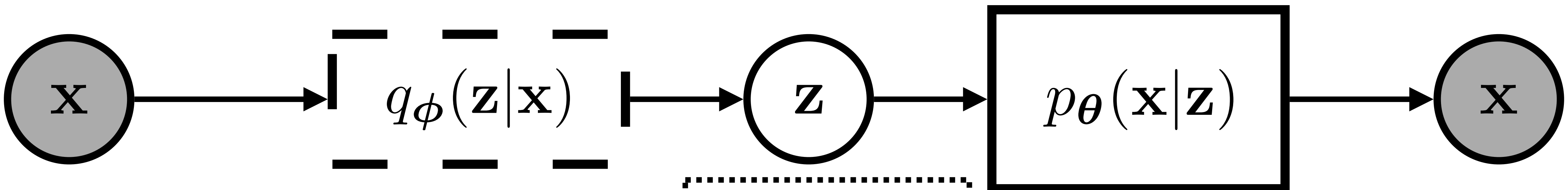
$$p(\mathbf{z})$$



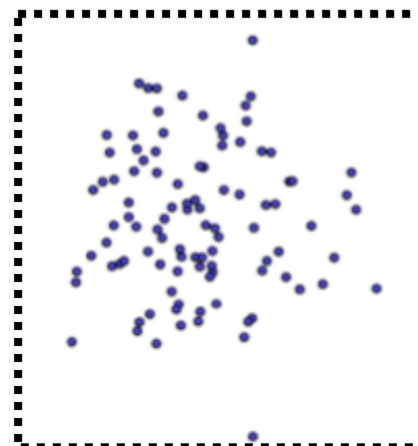


VAE vs. AE

VAE

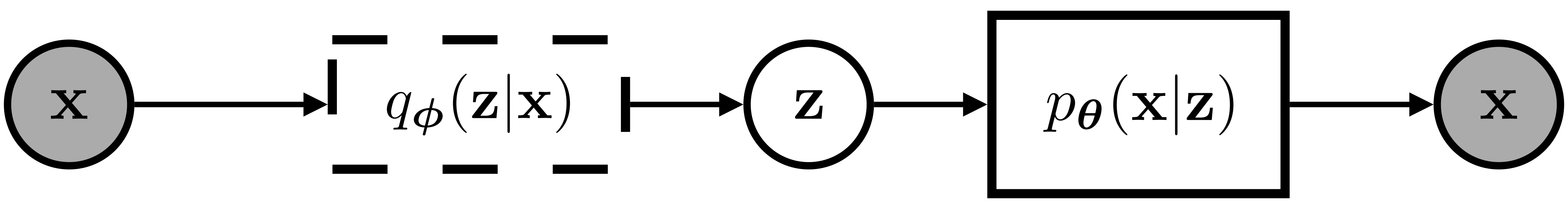


$p(\mathbf{z})$



inductive bias !

AE

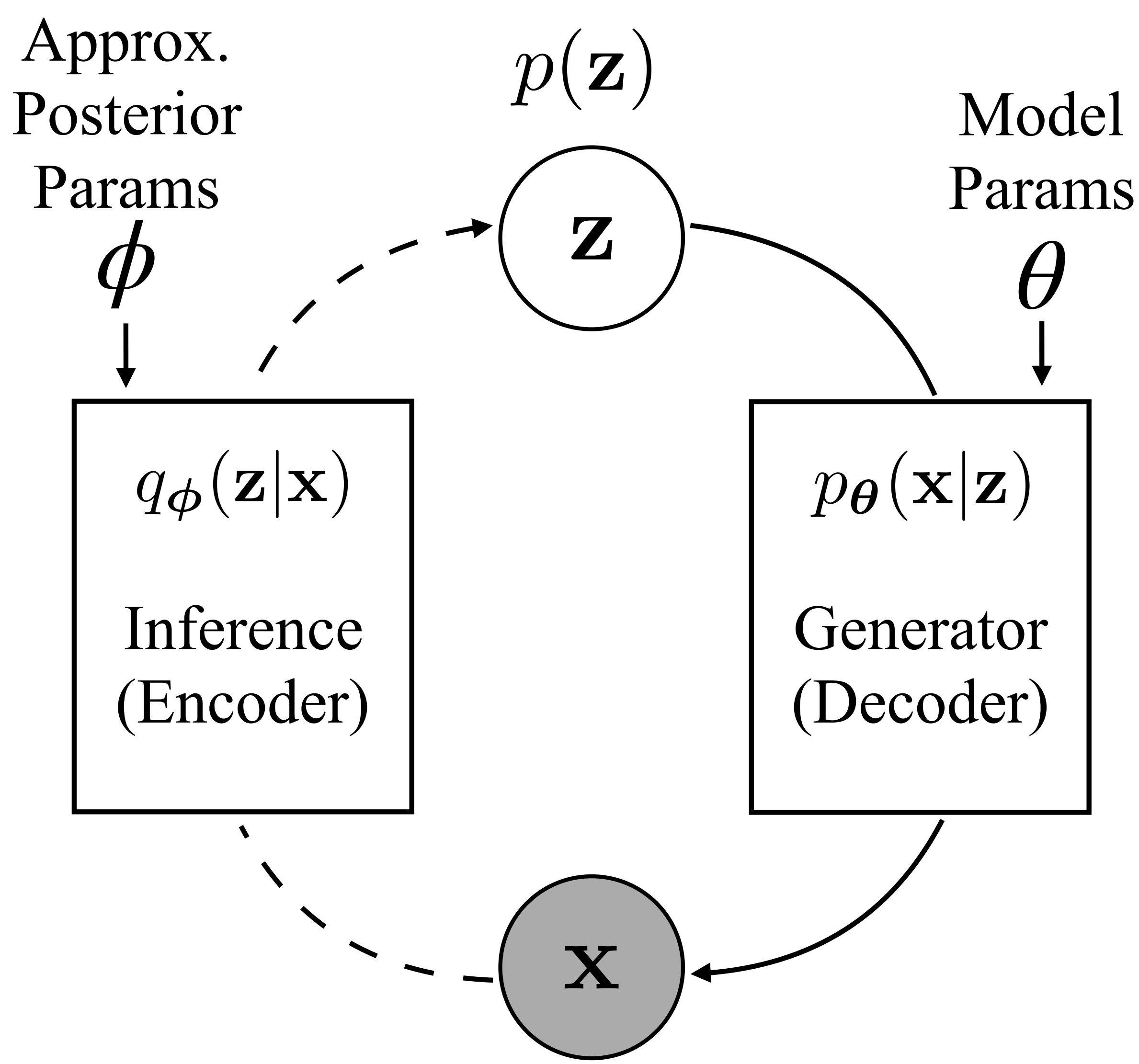


AE is not a generative model:

(1) Can't sample new data from AE
(2) Can't compute or bound the log likelihood of data x

Learning VAEs

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) >= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Regularizer}}$$



We need to compute the gradients

$$\underbrace{\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{Decoder: easy}} \text{ and } \underbrace{\nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{Encoder: hard}}$$

Learning VAEs

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) >= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Regularizer}}$$

$$\underbrace{\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{Decoder:}} \text{ and } \nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)$$

Gradient for decoder depends only on the reconstruction loss.

Can be computed by sampling **z**s from q, and standard backprop.

Learning VAEs

$$\log p_{\boldsymbol{\theta}}(\mathbf{x}) \quad >= \quad \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{KL Regularizer}}$$

$$\nabla_\theta \mathcal{L}(\mathbf{x};\theta,\phi) \text{ and } \underbrace{\nabla_\phi \mathcal{L}(\mathbf{x};\theta,\phi)}_{\text{Encoder:}}$$

Gradient for encoder, q, depends on both terms.

And the encoder is used to compute the expectation in the reconstruction loss!

Problem!
Sampling Breaks Backprop

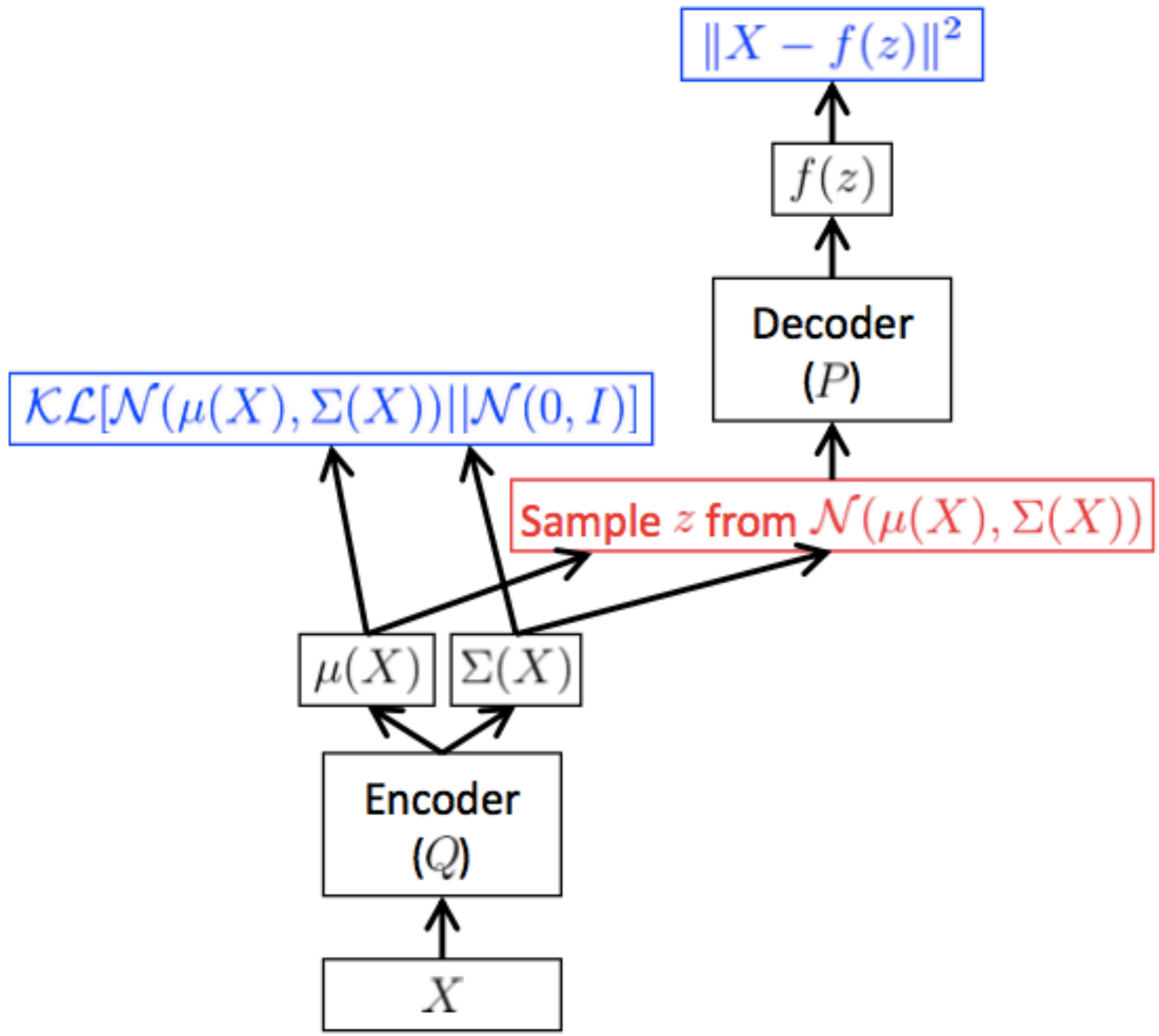


Figure Credit: Doersch (2016)

Solution: Re-parameterization Trick

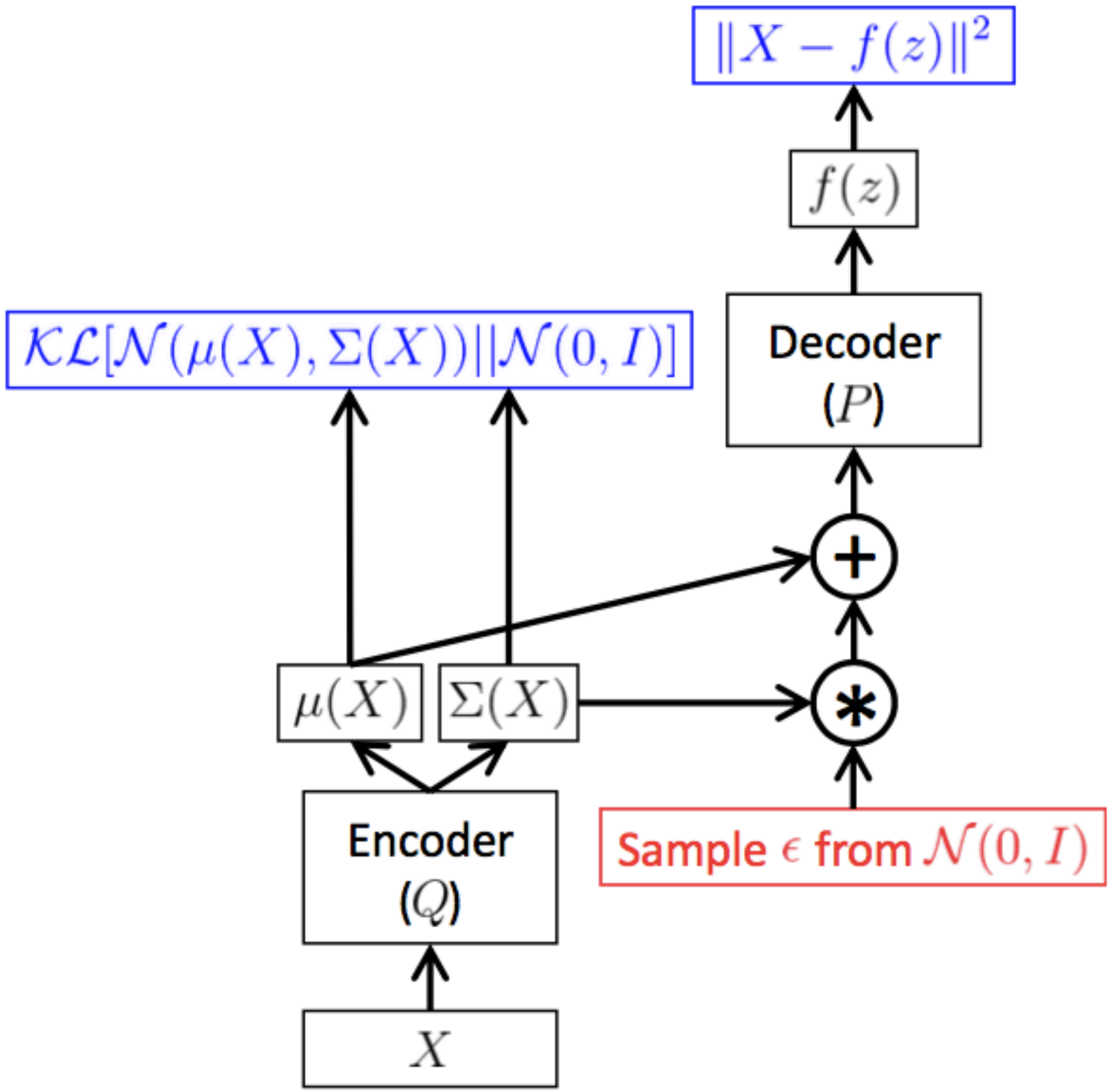


Figure Credit: Doersch (2016)

An Example: Generating Sentences w/ Variational Autoencoders

Generating from Language Models

- **Remember:** using autoregressive sampling, we can generate from a normal language model

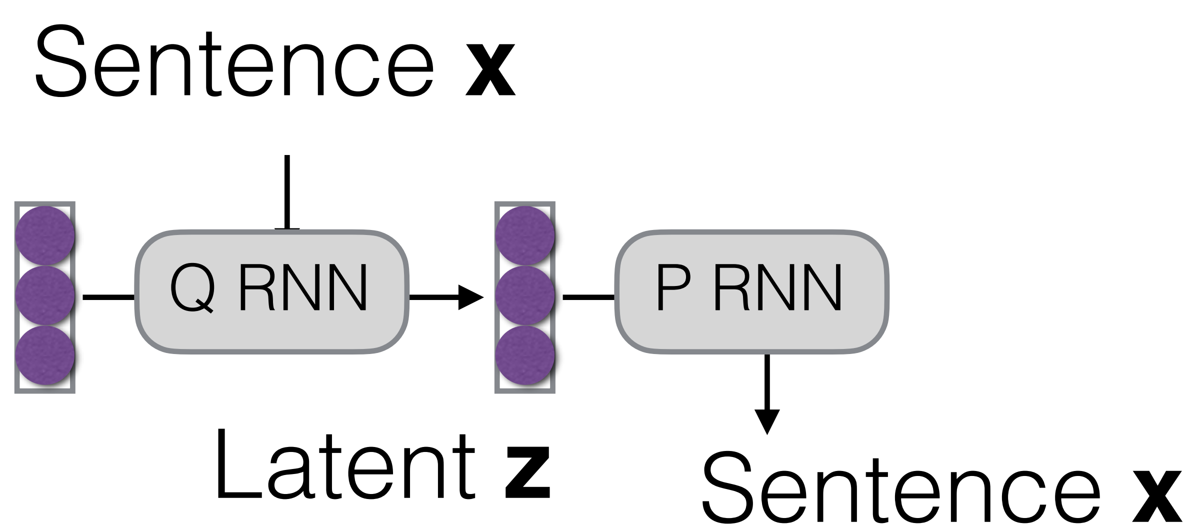
```
while x_{j-1} != "</s>":
  x_j ~ P(x_j | x_1, …, x_{j-1})
```

- We can also generate conditioned on something $P(\mathbf{y}|\mathbf{x})$ (e.g. translation, image captioning)

```
while y_{j-1} != "</s>":
  y_j ~ P(y_j | X, y_1, …, y_{j-1})
```

Generating Sentences from a Continuous Space (Bowman et al. 2015)

- The VAE-based approach is conditional language model that conditions on a latent variable **z**
- Like an encoder-decoder, but latent representation is latent variable, input and output are identical



Motivation for Latent Variables

- Allows for a **consistent latent space** of sentences?
  - e.g. interpolation between two sentences

Standard encoder-decoder

i went to the store to buy some groceries .
i store to buy some groceries .
i were to buy any groceries .
horses are to buy any groceries .
horses are to buy any animal .
horses the favorite any animal .
horses the favorite favorite animal .
horses are my favorite animal .

VAE

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

- **More robust to noise?** VAE can be viewed as standard model + regularization.

Difficulties in Training

- Of the two components in the VAE objective, the KL divergence term is much easier to learn!

$$\underbrace{\mathbb{E}_{\boldsymbol{z}\sim Q(\boldsymbol{z}|\boldsymbol{x})}[\log P(\boldsymbol{x} \mid \boldsymbol{z})]}_{\text{Requires good generative model}} - \underbrace{\mathcal{KL}[Q(\boldsymbol{z} \mid \boldsymbol{x})||P(\boldsymbol{z})]}_{\text{Just need to set the mean/variance of Q to be same as P}}$$

- Results in the model learning to rely solely on decoder and ignore latent variable ($P(x|z) = P(x)$) -> **Posterior Collapse**

Solution 1: KL Divergence Annealing

- Basic idea: Multiply KL term by a constant λ starting at zero, then gradually increase to 1
- Result: model can learn to use **z** before getting penalized

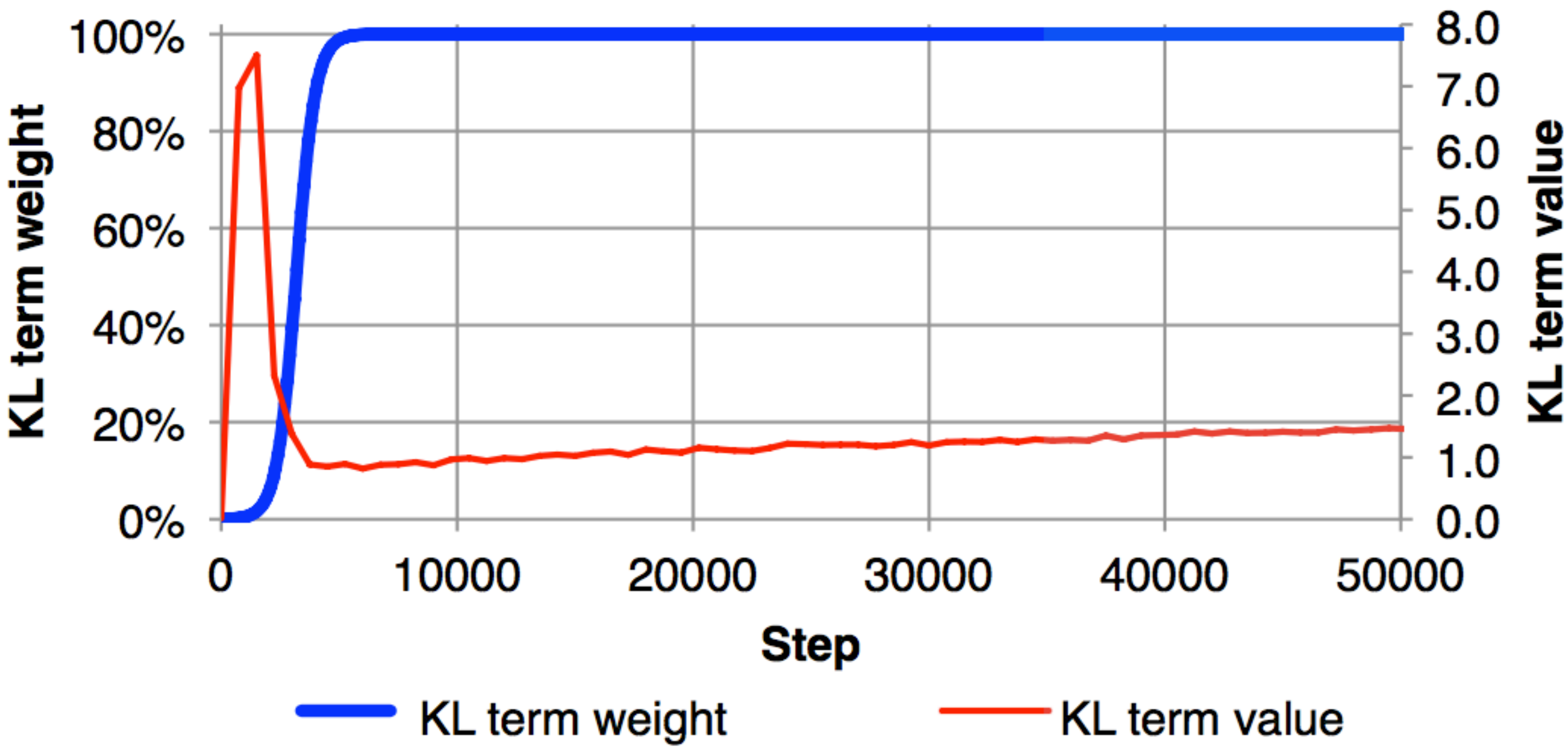


Figure Credit: Bowman et al. (2017)

Solution 2: Free bits / KL thresholding

- Free bits replaces the KL term in ELBO with a hinge loss that maxes each component of the original KL with a constant:

$$\sum_i \max[\lambda, D_{\mathrm{KL}}(q_\phi(z_i|x)||p(z_i))]$$

- λ :Target rate

Solution 3: Aggressive Inference Network Learning

- Freeze the generation network p during some training of the inference network q

$$\max_{\boldsymbol{\theta},\boldsymbol{\phi}} \quad \underbrace{\log p_{\boldsymbol{\theta}}(\mathbf{x})}_{\text{marginal log data likelihood}} \quad - \quad \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))}_{\text{agreement between approximate and model posteriors}}$$

$$\downarrow$$

$$\max_{\boldsymbol{\theta}} \max_{\boldsymbol{\phi}} \quad \underbrace{\log p_{\boldsymbol{\theta}}(\mathbf{x})}_{\text{marginal log data likelihood}} \quad - \quad \underbrace{D_{\mathrm{KL}}(q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})|p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x}))}_{\text{agreement between approximate and model posteriors}}$$

(He et al. 2019)

Handling Discrete Latent Variables

Discrete Latent Variables?

- Many variables are better treated as discrete
  - Part-of-speech of a word
  - Class of a question
  - Writer traits (left-handed or right-handed, etc.)
- How do we handle these?

Method 1: Enumeration

- For discrete variables, our integral is a sum

$$P(\boldsymbol{x};\theta) = \sum_{\boldsymbol{z}} P(\boldsymbol{x} \mid \boldsymbol{z};\theta)P(\boldsymbol{z})$$

- If the number of possible configurations for **z** is small, we can just sum over all of them

Method 2: Sampling

- Randomly sample a subset of configurations of **z** and optimize with respect to this subset
- Various flavors:
  - Minimum risk training
  - Maximize ELBO loss
- Score function gradient estimator - Policy Gradient Method
  - Like RL, but with p(x | z) as the reward!
  - Unbiased estimator but high variance - need to control variance

Method 3: Reparameterization

(Maddison et al. 2017, Jang et al. 2017)

- Reparameterization also possible for discrete variables!

Original Categorical Sampling Method:

$$\hat{\boldsymbol{z}} = \text{cat-sample}(P(\boldsymbol{z} \mid \boldsymbol{x}))$$

Reparameterized Method

$$\hat{\boldsymbol{z}} = \text{argmax}(\log P(\boldsymbol{z} \mid \boldsymbol{x}) + \text{Gumbel(0,1)})$$

where the Gumbel distribution is

$$\text{Gumbel}(0,1) = -\log(-\log(\text{Uniform(0,1)}))$$

- Backprop is still not possible, due to argmax

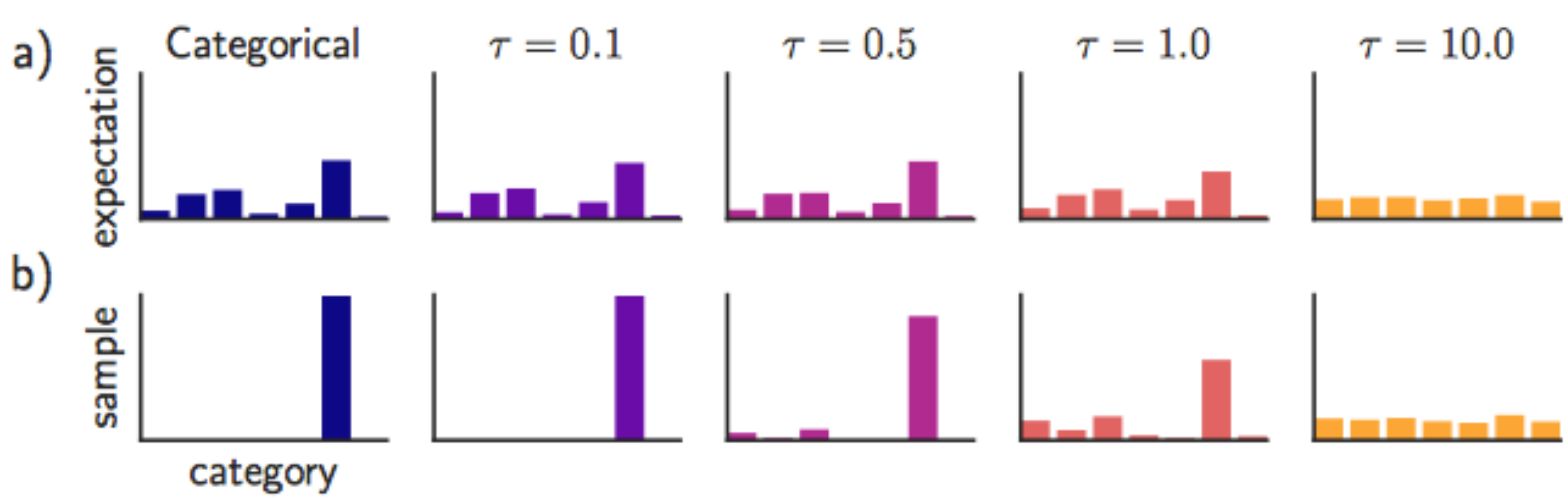
Gumbel-Softmax

- A way to soften the decision and allow for continuous gradients

- Instead of argmax, take softmax with temperature τ

$$\hat{\boldsymbol{z}} = \mathrm{softmax}((\log P(\boldsymbol{z} \mid \boldsymbol{x}) + \mathrm{Gumbel}(0{,}1))^{1/\tau})$$

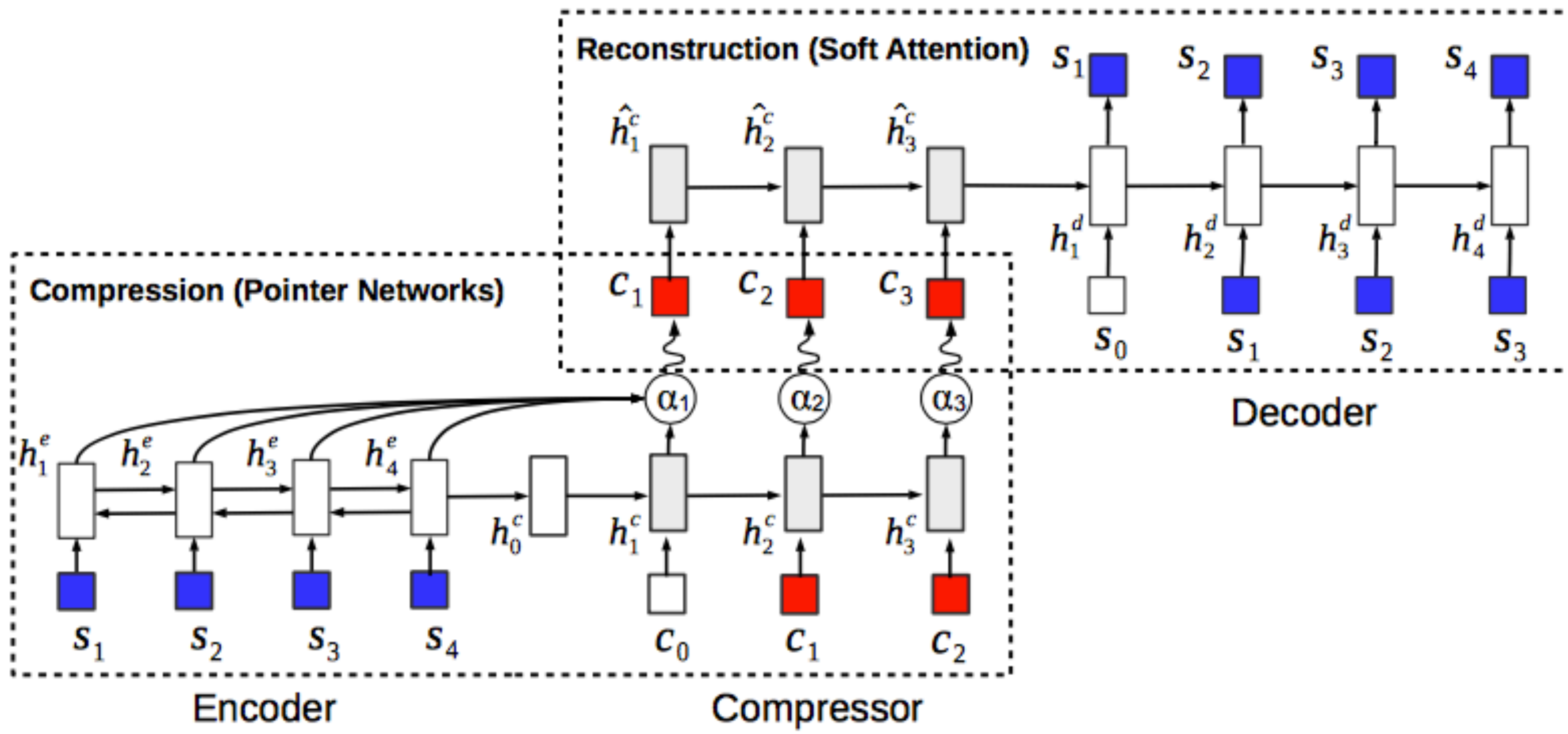
- As τ approaches 0, will approach max



Application Examples in NLP

Symbol Sequence Latent Variables (Miao and Blunsom 2016)

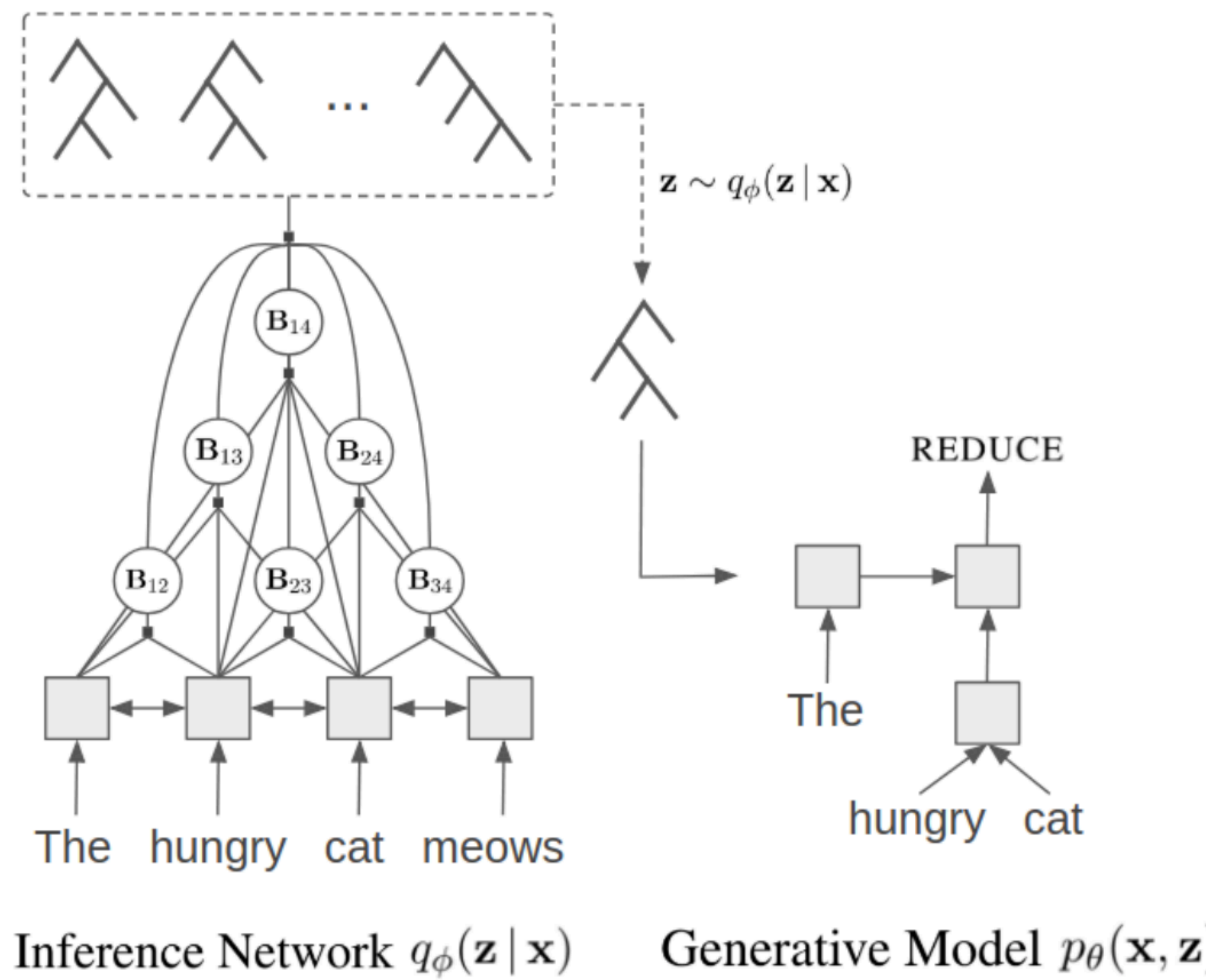
- Encoder-decoder with a sequence of latent symbols



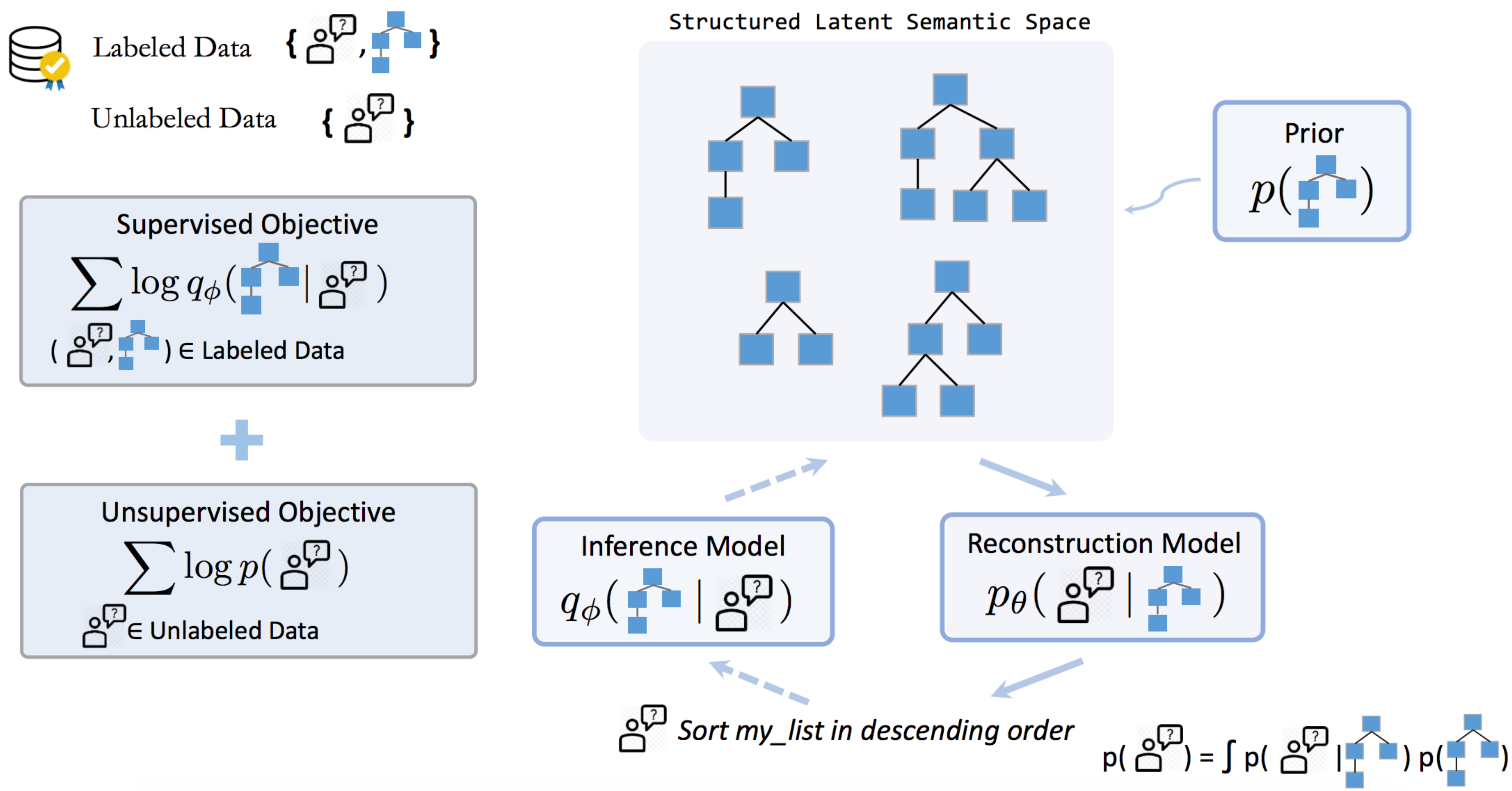
- Summarization in Miao and Blunsom (2016)
- Attempts to "discover" language (e.g. Havrylov and Titov 2017)
  - But things may not be so simple! (Kottur et al. 2017)

Unsupervised Recurrent Neural Network Grammars

(Kim et al., 2019)



STRUCTVAE: Tree-structured Latent Variable Models for Semi-supervised Semantic Parsing (Yin et al. 2018)



Questions?