CS/DS 552: Class 4

Jacob Whitehill

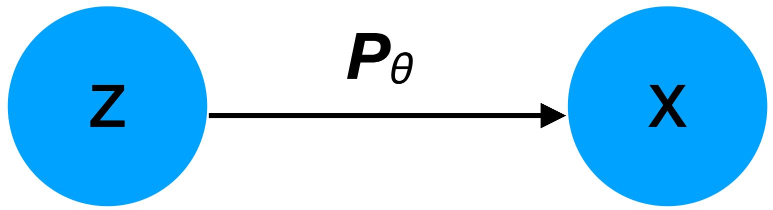
Variational auto-encoders (VAEs)

VAE architecture

- Fundamentally, a VAE is an LVM, where we posit that each **x** is “generated” by a latent code **z**:

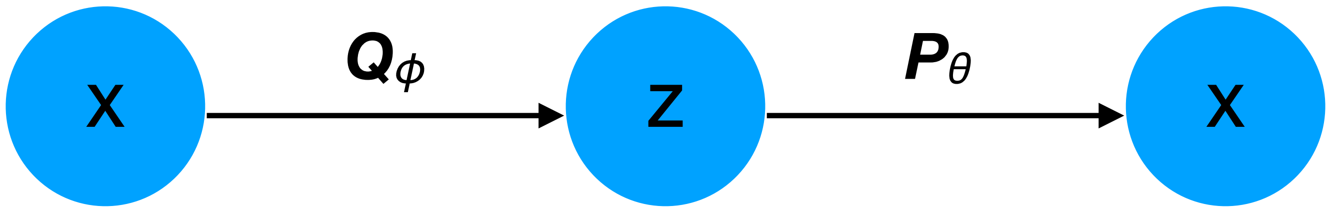
   1. Sample $\mathbf{z} \sim P(\mathbf{z}) \in \mathbb{R}^d$ using an easy-to-sample *P*(**z**).

   2. Compute $\mathbf{x} = P_\theta(\mathbf{x} \mid \mathbf{z}) \in \mathbb{R}^m$, where *g* is some “decoder” function with parameters *θ*.



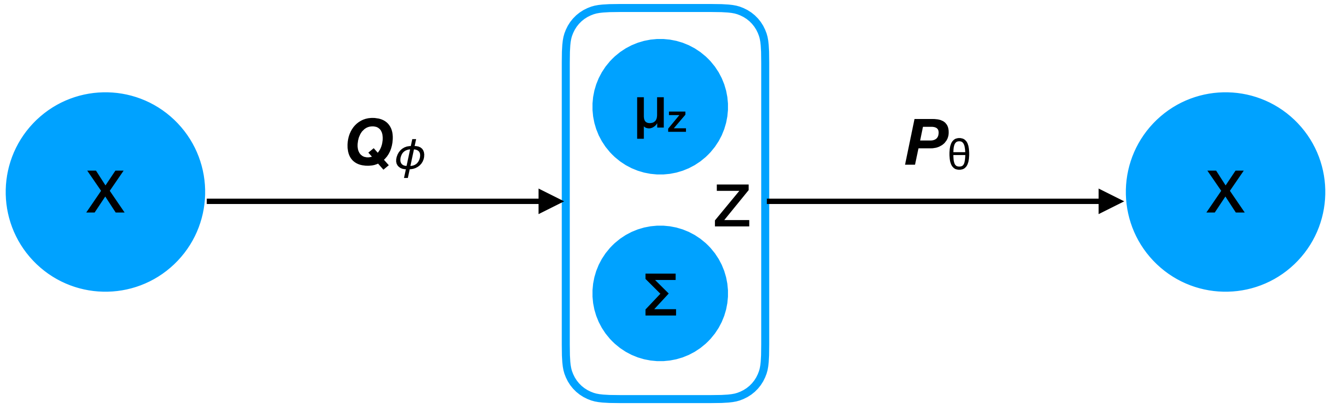
VAE architecture

- Architecturally, however, and *because of how it is trained*, a VAE consists of an encoder NN Q_ϕ and decoder NN P_θ.
  - $Q_\phi(\mathbf{z} \mid \mathbf{x})$ outputs a probability distribution over **Z** given **X**.
  - $P_\theta(\mathbf{x} \mid \mathbf{z})$ outputs a probability distribution over **X** given **Z**.



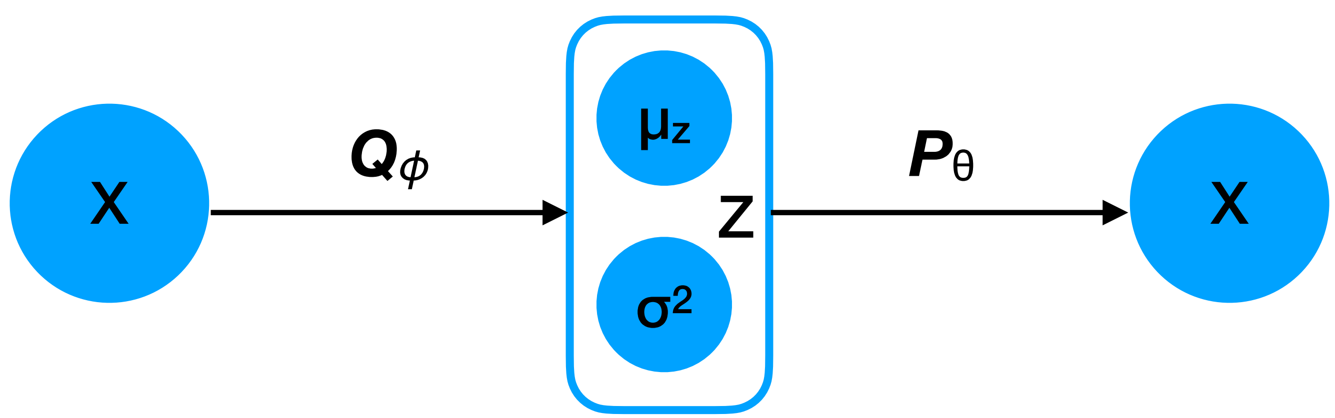
VAE architecture: encoder

- Most commonly, Q_ϕ is constrained to output a Gaussian distribution over latent codes **z** given input **x** .
- How do we force Q_ϕ to output a Gaussian distribution?
  - Given **x**, Q_ϕ needs to output:
    - Mean $\boldsymbol{\mu}_\mathbf{z}$
    - Covariance matrix $\boldsymbol{\Sigma}$



VAE architecture: encoder

- As a simplification, Q_ϕ can output a diagonal covariance matrix parameterized by just a vector [σ_1^2, …, σ_d^2].

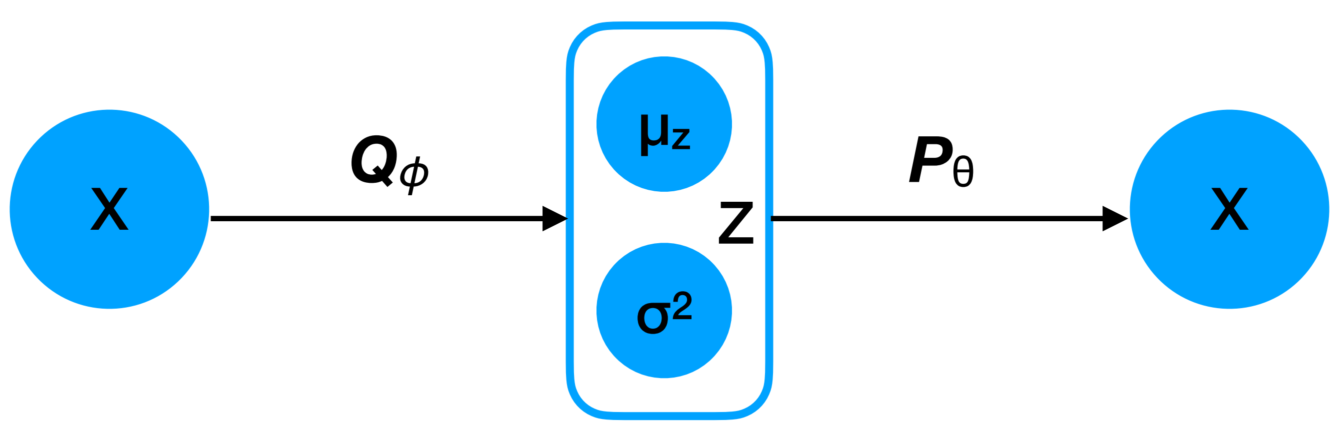


VAE architecture: encoder

- As a simplification, Q_ϕ can output a diagonal covariance matrix parameterized by just a vector [$\sigma_1^2, \ldots, \sigma_d^2$].

- All in all, Q_ϕ outputs $2d$ entries, where the first d specify the mean and the second d specify the covariance.

- We must force positivity of [$\sigma_1^2, \ldots, \sigma_d^2$], e.g., by exponentiating the output of Q_ϕ.

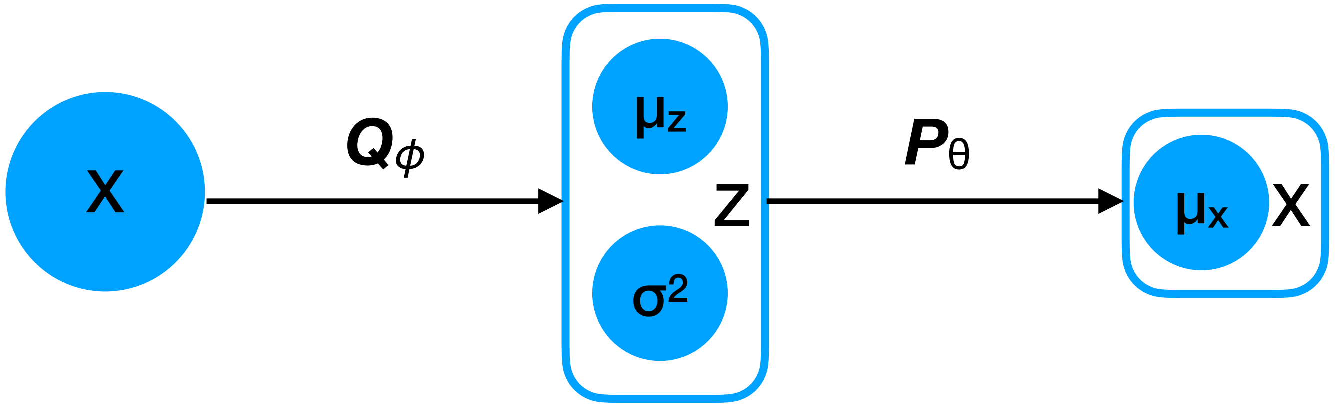


VAE architecture: decoder

- P_θ is usually one of:

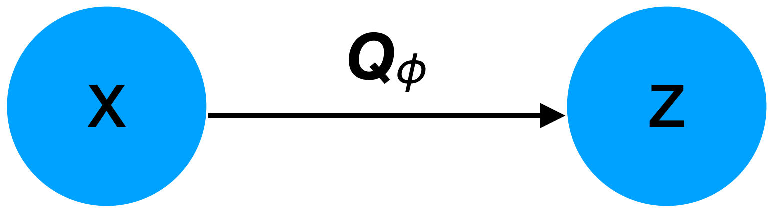
1. Gaussian $\mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I})$ with mean $\boldsymbol{\mu}_{\mathbf{x}}$ and $\mathbf{I}$-covariance
    - Useful for predicting data from $\mathbb{R}^m$.
    - NN: no activation function.

2. Element-wise Bernoulli with mean $\boldsymbol{\mu}_{\mathbf{x}}$.
    - Useful for predicting data from $[0,1]^m$.
    - NN: logistic activation function.



VAE architecture

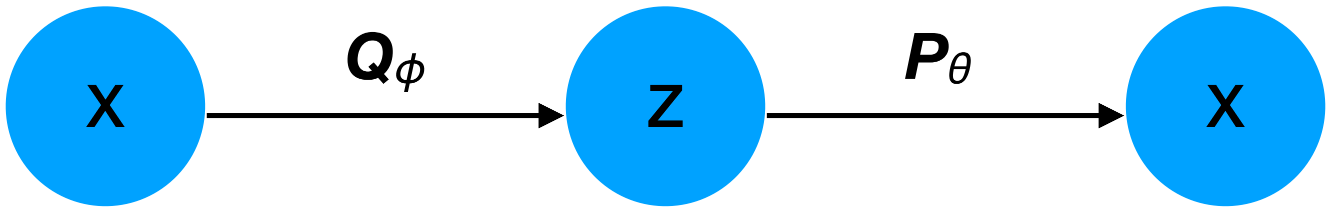
- With Q_ϕ, we can estimate from input $\mathbf{x} \in \mathbb{R}^m$ the latent code $\mathbf{z} \in \mathbb{R}^d$ that generated it.



VAE architecture

- With Q_ϕ, we can estimate from input $\mathbf{x} \in \mathbb{R}^m$ the latent code $\mathbf{z} \in \mathbb{R}^d$ that generated it.

- With P_θ, we can use input code $\mathbf{z} \in \mathbb{R}^d$ to generate $\mathbf{x} \in \mathbb{R}^m$.



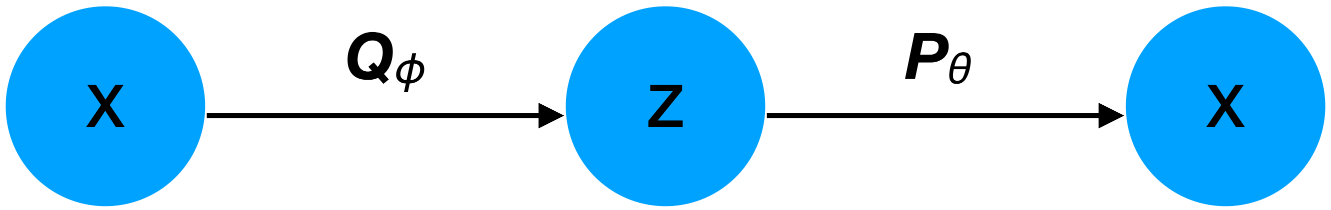
VAE architecture

- With Q_ϕ, we can estimate from input $\mathbf{x} \in \mathbb{R}^m$ the latent code $\mathbf{z} \in \mathbb{R}^d$ that generated it.

- With P_θ, we can use input code $\mathbf{z} \in \mathbb{R}^d$ to generate $\mathbf{x} \in \mathbb{R}^m$.

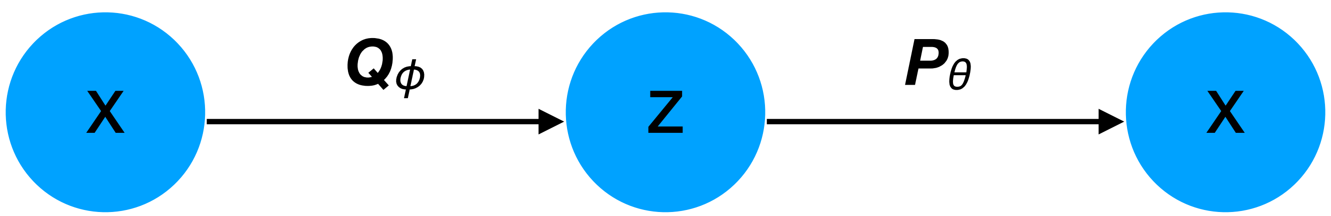
- $P(\mathbf{x} \mid \mathbf{z})$ represents how likely the VAE believes a particular example $\mathbf{x}$ is to be generated from latent code $\mathbf{z}$.

- For good reconstruction quality, we want $P(\mathbf{x} \mid \mathbf{z}) = P_\theta(Q_\phi(\mathbf{x}))$ to be *high*.



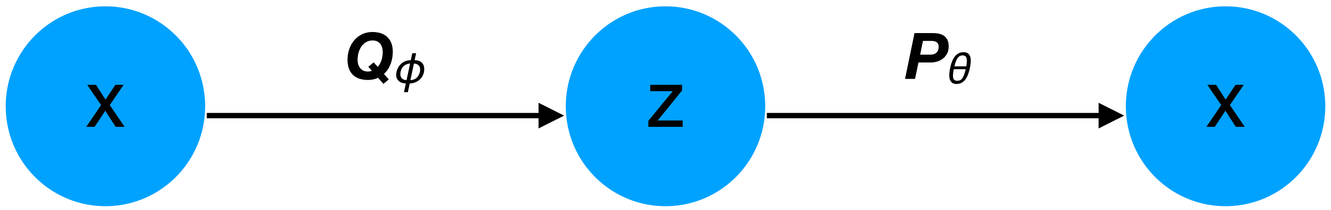
NN implementations

- VAEs can be applied to many types of data (images, tabular data, time series, …) given an appropriate architecture, e.g.:
    - FCNN encoder & decoder for tabular data.
    - Conv. encoder & de-conv. decoder for images.
    - RNN encoder & decoder for time series.



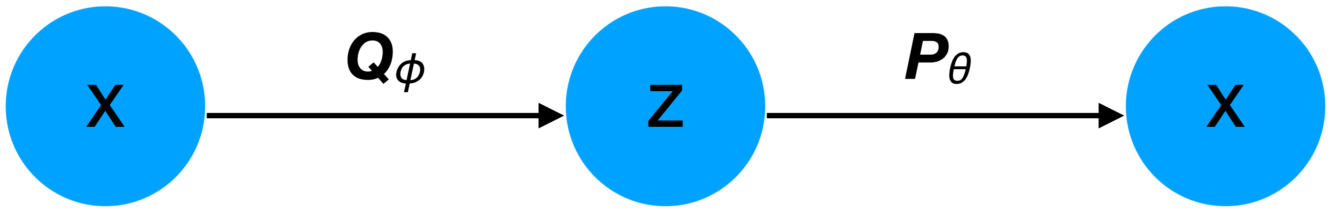
Likelihood models & loss functions

- The VAE decoder is usually one of:
  - Gaussian $P(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I})$, where $\mu_{\mathbf{x}} \in \mathbb{R}^m$.
  - Bernoulli $P(\mathbf{x} \mid \mathbf{z}) = \text{Ber}(\mu_{\mathbf{x}})$, where $\mu_{\mathbf{x}} \in [0,1]^m$.



Likelihood models & loss functions

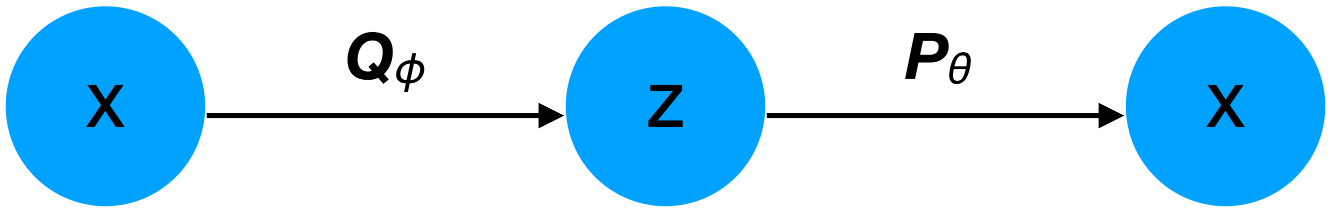
- This requires an activation at the end of the decoder:
  - Gaussian: identity/nothing
  - Bernoulli: logistic sigmoid



Likelihood models & loss functions

- *Maximizing* the likelihood = *Minimizing* a loss function:
  - Gaussian: MSE: $\frac{1}{2}\|x - \mu_{\mathbf{x}}\|^2$
  - Bernoulli: BCE: $-y \log \hat{y} - (1 - y)\log(1 - \hat{y})$

    Binary cross-entropy

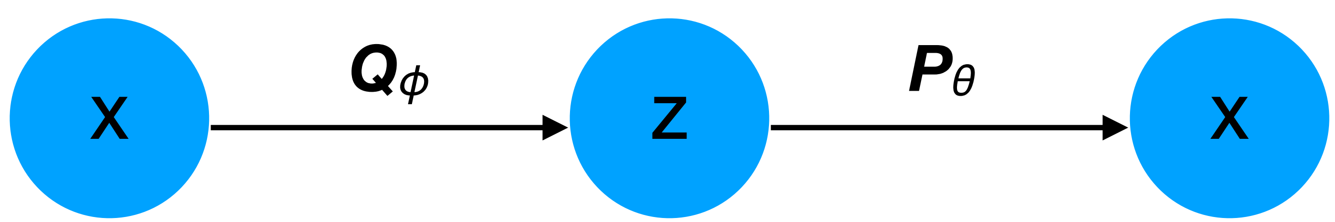


VAE architecture

- The parameters ϕ and θ are trained using maximum-likelihood estimation (MLE).

- We aim to maximize the likelihood of our observed training data, given *P*'s parameters θ, i.e.: $P_\theta(\{\mathbf{x}^{(i)}\}_{i=1}^{n})$

- Using a MLE approximation technique, we will also optimize Q's parameters ϕ along the way.



Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.

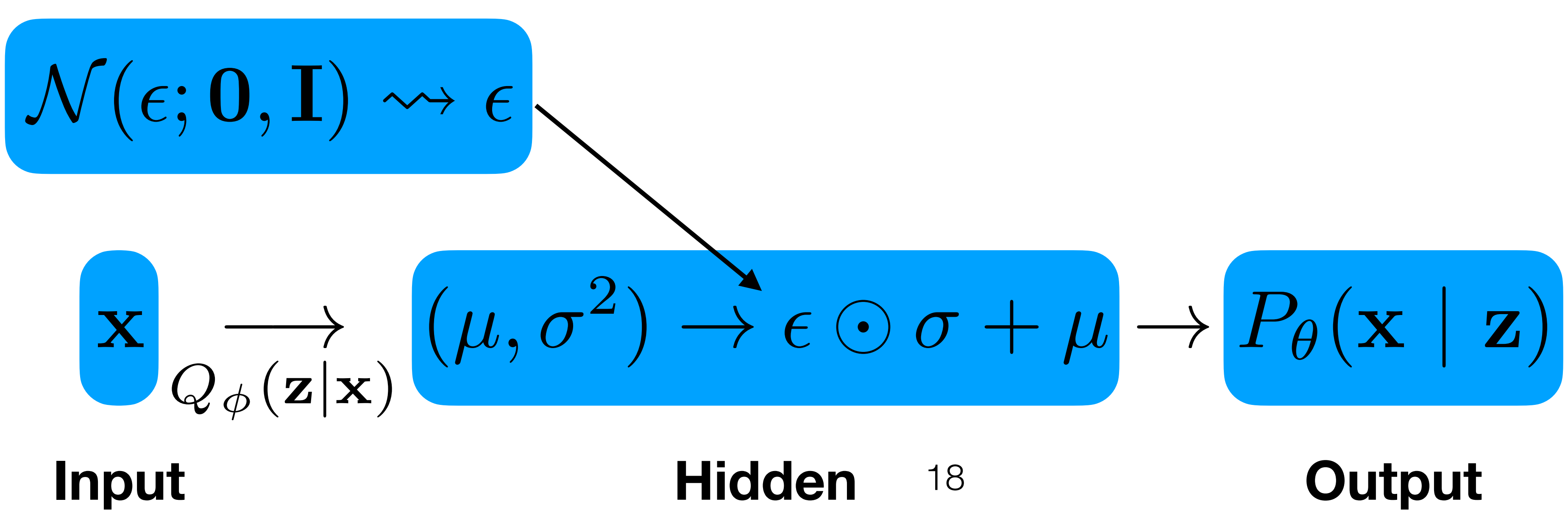
$$\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}) \rightsquigarrow \epsilon$$

$$\mathbf{x} \xrightarrow[Q_\phi(\mathbf{z}|\mathbf{x})]{} (\mu, \sigma^2) \to \epsilon \odot \sigma + \mu \to P_\theta(\mathbf{x} \mid \mathbf{z})$$

Input **Hidden** **Output**

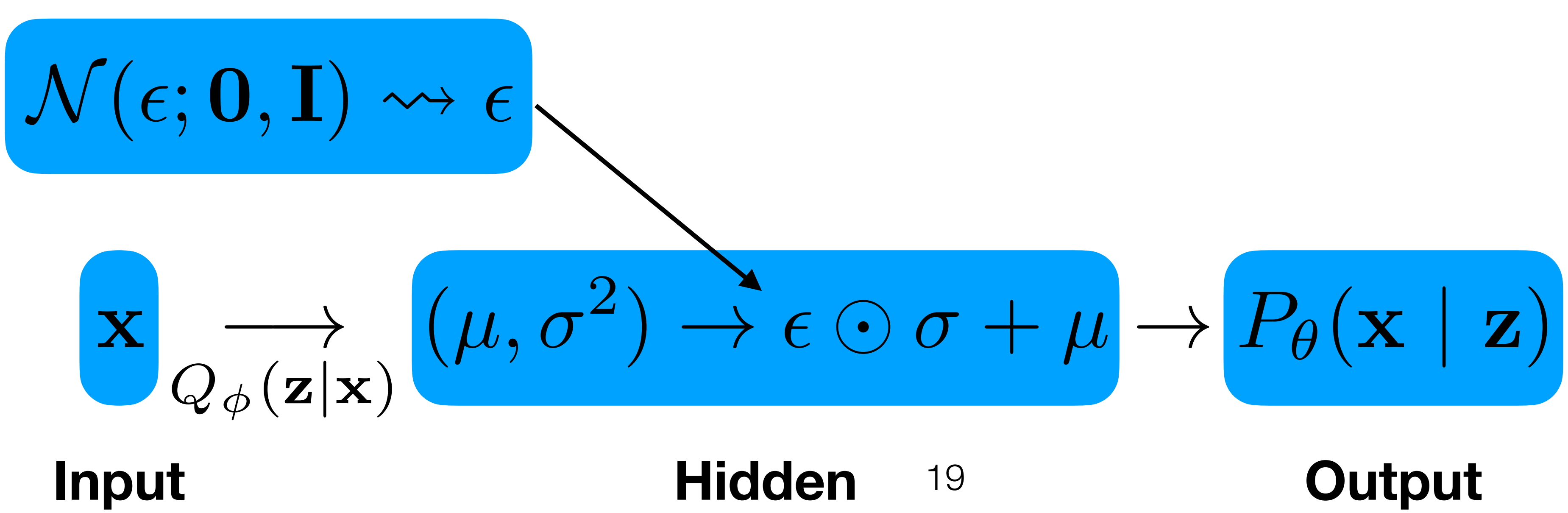
Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
  - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$



Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
  - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$
  - Sample $\tilde{n}$ noise vectors: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$



Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
  - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$
  - Sample $\tilde{n}$ noise vectors: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$
  - Compute: $(\mu^{(i)}, {\sigma^{(i)}}^2) = Q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}),\ \forall i$

$$\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}) \rightsquigarrow \epsilon$$

$$\mathbf{x} \xrightarrow[Q_\phi(\mathbf{z}|\mathbf{x})]{} (\mu, \sigma^2) \to \epsilon \odot \sigma + \mu \to P_\theta(\mathbf{x} \mid \mathbf{z})$$

Input **Hidden** **Output**

Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
  - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$
  - Sample $\tilde{n}$ noise vectors: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$
  - Compute: $(\mu^{(i)}, {\sigma^{(i)}}^2) = Q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}), \ \forall i$
  - Compute: $\mathbf{z}^{(i)} = \epsilon^{(i)} \odot \sigma^{(i)} + \mu^{(i)}$

$$\boxed{\mathbf{x}} \xrightarrow[Q_\phi(\mathbf{z}|\mathbf{x})]{} \boxed{(\mu, \sigma^2) \to \epsilon \odot \sigma + \mu} \to \boxed{P_\theta(\mathbf{x} \mid \mathbf{z})}$$

Input **Hidden** **Output**

Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
  - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$
  - Sample $\tilde{n}$ noise vectors: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$
  - Compute: $(\mu^{(i)}, {\sigma^{(i)}}^2) = Q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}), \ \forall i$
  - Compute: $\mathbf{z}^{(i)} = \epsilon^{(i)} \odot \sigma^{(i)} + \mu^{(i)}$
  - Maximize:
$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

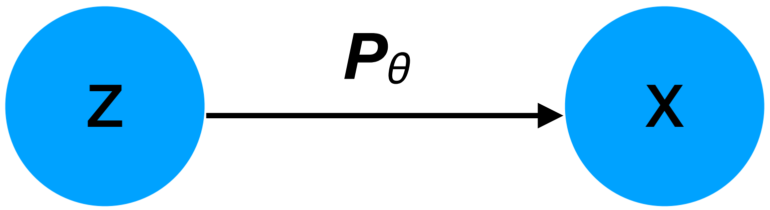
Training procedure

- Define networks Q_ϕ and decoder P_θ.
- Initialize parameters ϕ and θ.
- For each mini-batch:
    - Select $\tilde{n}$ examples: $\{\mathbf{x}^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^m$
    - Sample $\tilde{n}$ noise vectors: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$
    - Compute: $(\mu^{(i)}, {\sigma^{(i)}}^2) = Q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}), \ \forall i$
    - Compute: $\mathbf{z}^{(i)} = \epsilon^{(i)} \odot \sigma^{(i)} + \mu^{(i)}$
    - Maximize:
    $$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$
    - Update ϕ and θ using back-propagation.

VAE: MLE Derivation

VAE: MLE derivation

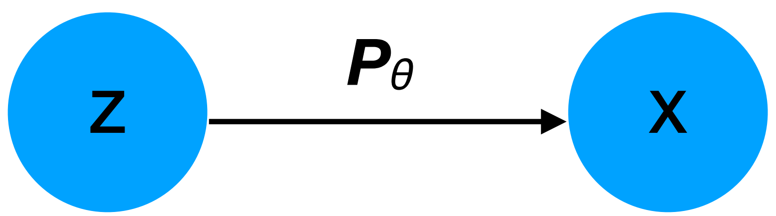
$$\log P(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$



VAE: MLE derivation

$$\log P(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

$$= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}$$ **Definition of conditional probability**

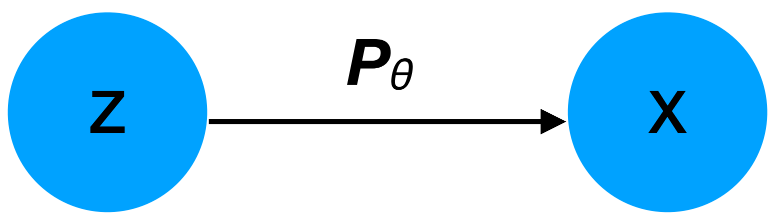


VAE: MLE derivation

$$\log P(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

$$= \log \boxed{\int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}}$$

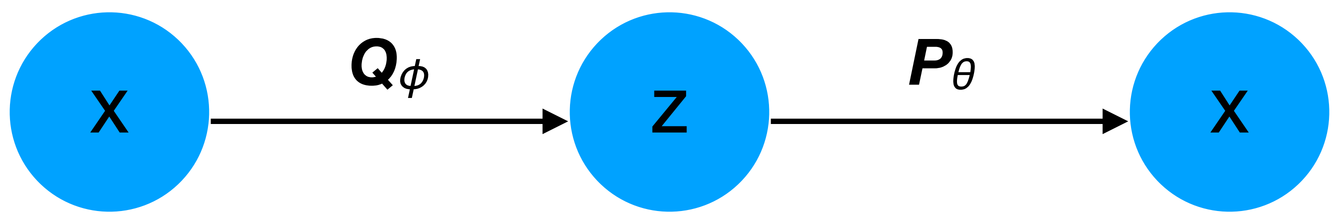
- We are in trouble!
  - This integral is the product of $P(\mathbf{z})$ (e.g., Gaussian) and $P(\mathbf{x} \mid \mathbf{z})$ (i.e., the decoder NN). We cannot resolve it!
  - We cannot integrate numerically (intractable)!
- We cannot even evaluate $P(\mathbf{x})$, let alone optimize it!



VAE: MLE derivation

$$\log P(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$
$$= \log \boxed{\int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}}$$

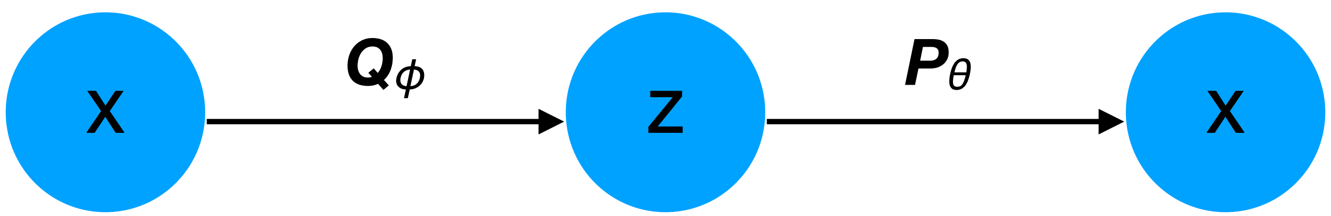
- By introducing auxiliary parameters ϕ through an encoder network Q, we actually make the optimization easier…



VAE: MLE derivation

$$
\begin{aligned}
\log P(\mathbf{x}) &= \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z}
\end{aligned}
$$

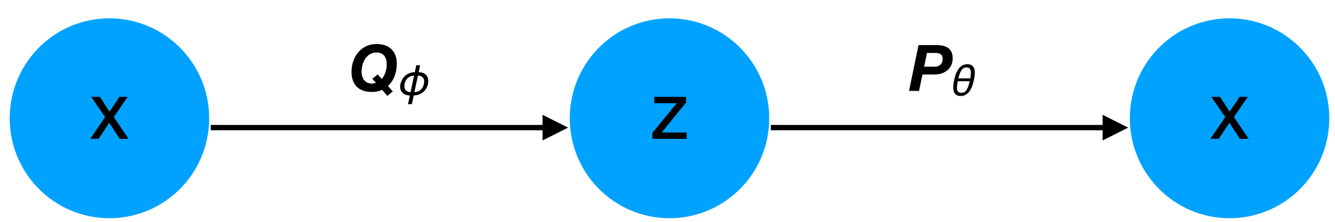
This holds for any non-zero *Q*.



VAE: MLE derivation

$$\log P(\mathbf{x}) = \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$
$$= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z}$$
$$= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z}$$

Note also that we can interpret this function as the expectation w.r.t. Q(z | x).



VAE: MLE derivation

$$
\begin{aligned}
\log P(\mathbf{x}) &= \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\
&\geq \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \left(P(\mathbf{x} \mid \mathbf{z}) \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} \right) d\mathbf{z} \qquad \text{Jensen's inequality}
\end{aligned}
$$

- This is called the **Evidence Lower Bound (ELBO)**.

VAE: MLE derivation

$$\begin{aligned}
\log P(\mathbf{x}) &= \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\
&\geq \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \left(P(\mathbf{x} \mid \mathbf{z}) \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} \right) d\mathbf{z} \qquad \text{Jensen's inequality}
\end{aligned}$$

- It turns out (see Prince, sec. 17.4.1) that, if $Q(\mathbf{z} \mid \mathbf{x}) = P(\mathbf{z})$, then this inequality is actually an equality.
- Hence, this lower-bound can actually be made "tight" if Q is powerful enough to approximate $P(\mathbf{z})$.

VAE: MLE derivation

$$
\begin{aligned}
\log P(\mathbf{x}) &= \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\
&\geq \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \left(P(\mathbf{x} \mid \mathbf{z}) \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} \right) d\mathbf{z} \\
&= \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} + \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log P(\mathbf{x} \mid \mathbf{z}) d\mathbf{z}
\end{aligned}
$$

VAE: MLE derivation

$$
\begin{aligned}
\log P(\mathbf{x}) &= \log \int_{\mathbf{z}} P(\mathbf{x}, \mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z}) d\mathbf{z} \\
&= \log \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \frac{P(\mathbf{x} \mid \mathbf{z}) P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} \\
&\geq \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \left(P(\mathbf{x} \mid \mathbf{z}) \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} \right) d\mathbf{z} \\
&= \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z} \mid \mathbf{x})} d\mathbf{z} + \int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) \log P(\mathbf{x} \mid \mathbf{z}) d\mathbf{z} \\
&= -D_{\mathrm{KL}}(Q(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_Q[\log P(\mathbf{x} \mid \mathbf{z})]
\end{aligned}
$$

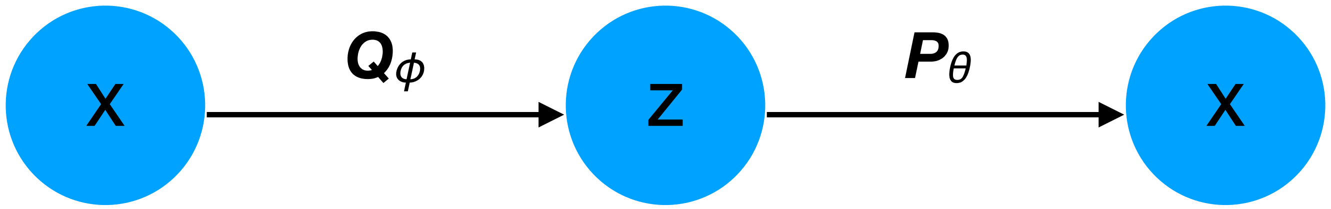
Definitions of KL-divergence and expectation.

ELBO

- To maximize the ELBO, we need to:
  - Minimize KL-divergence of hidden state w.r.t. standard normal distribution.

    and

  - Maximize the expected reconstruction log-likelihood.



$$= -D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})]$$

ELBO

- The first term in the ELBO:

$$= -D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})]$$

has closed-form differentiable solutions when $P(\mathbf{z})$ and $Q_\phi(\mathbf{z} \mid \mathbf{x})$ are Gaussian (see earlier slide).

ELBO

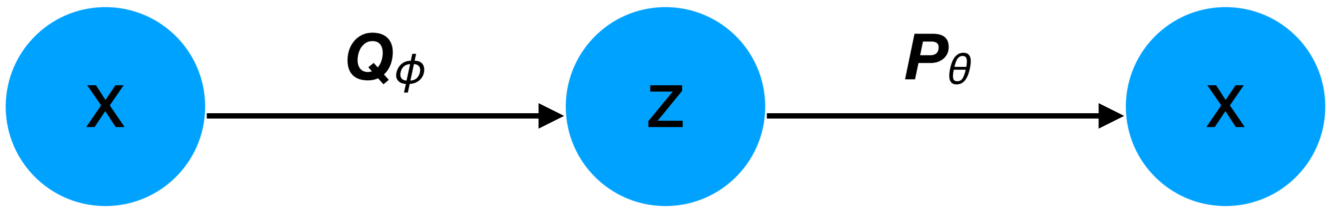
- The second term in the ELBO:

$$= -D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})]$$

can be approximated by sampling:

$$\mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})] \approx \frac{1}{K} \sum_{k=1}^{K} \log P_\theta(\mathbf{x} \mid \mathbf{z}^{(k)})$$

$$\text{where } \mathbf{z}^{(k)} \sim Q_\phi(\mathbf{z} \mid \mathbf{x})$$

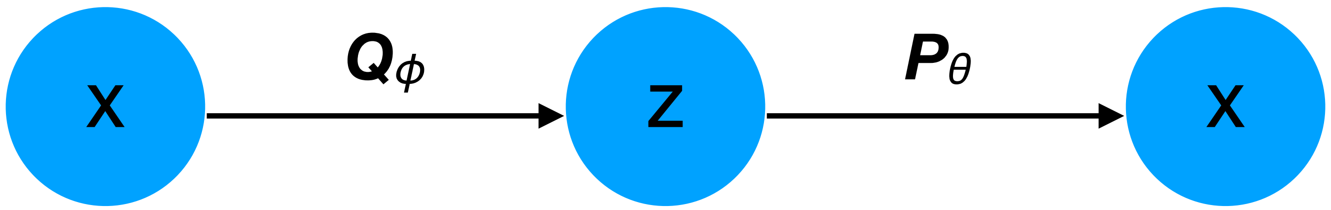


ELBO

- In particular:
    1. For input **x**, compute *Q*(**z** | **x**).
    2. Sample *K different* $\mathbf{z}^{(k)}$ from this distribution.
    3. For each $\mathbf{z}^{(k)}$, compute $P(\mathbf{x} \mid \mathbf{z}^{(k)})$, i.e., reconstruction probability of **x** using **z**.

$$\mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})] \approx \frac{1}{K} \sum_{k=1}^{K} \log P_\theta(\mathbf{x} \mid \mathbf{z}^{(k)})$$

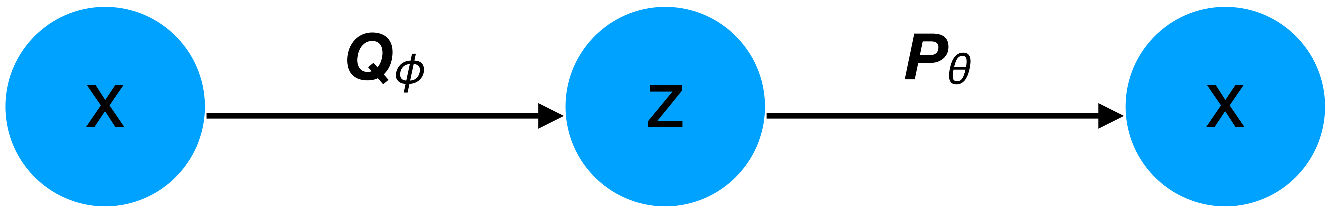
$$\text{where } \mathbf{z}^{(k)} \sim Q_\phi(\mathbf{z} \mid \mathbf{x})$$



ELBO

- In practice, we usually set *K*=1 for simplicity.

$$\mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})] \approx \log P_\theta(\mathbf{x} \mid \mathbf{z})$$
$$\text{where } \mathbf{z} \sim Q_\phi(\mathbf{z} \mid \mathbf{x})$$

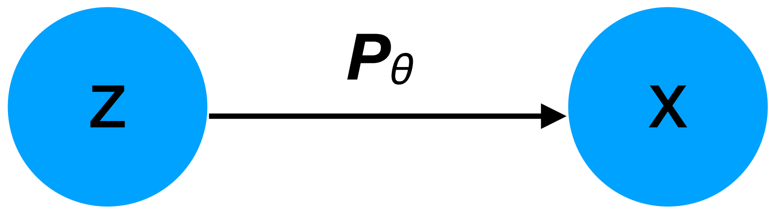


Optimization with auxiliary variables

- Conceptually, we have changed from an optimization:

$$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

to:



Optimization with auxiliary variables

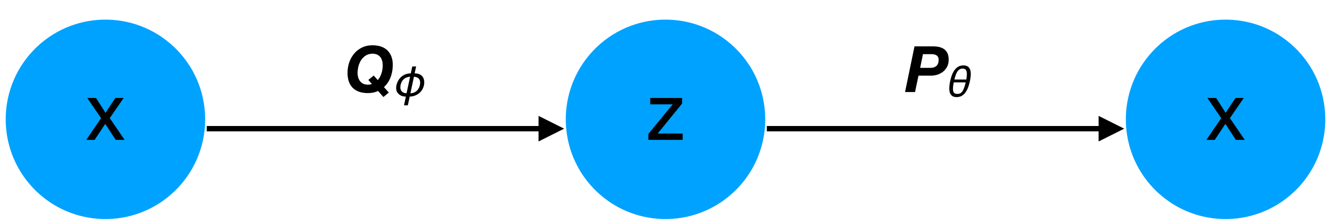
- Conceptually, we have changed from an optimization:

$$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

to:

$$\arg\max_{\theta,\phi} g(\theta, \phi)$$ We can both estimate and (using SGD) optimize *g*!

where we can just discard ϕ afterwards.



Optimization with auxiliary variables

- Conceptually, we have changed from an optimization:

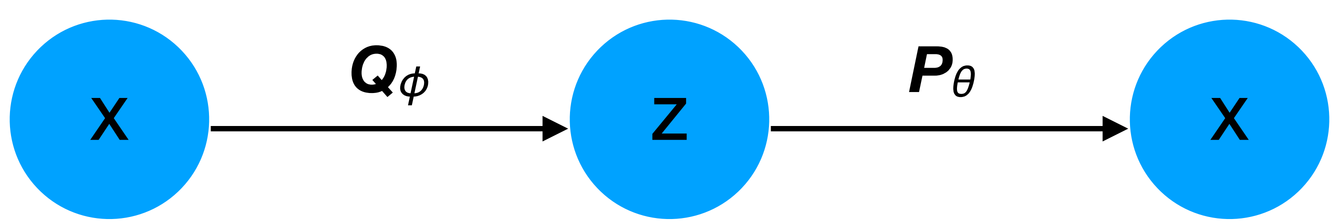
$$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

to:

$$\arg\max_{\theta,\phi} g(\theta, \phi)$$ We can both estimate and (using SGD) optimize *g*!

where we can just discard ϕ afterwards.

- Under what conditions can this actually work?



Optimization with auxiliary variables

- Conceptually, we have changed from an optimization:

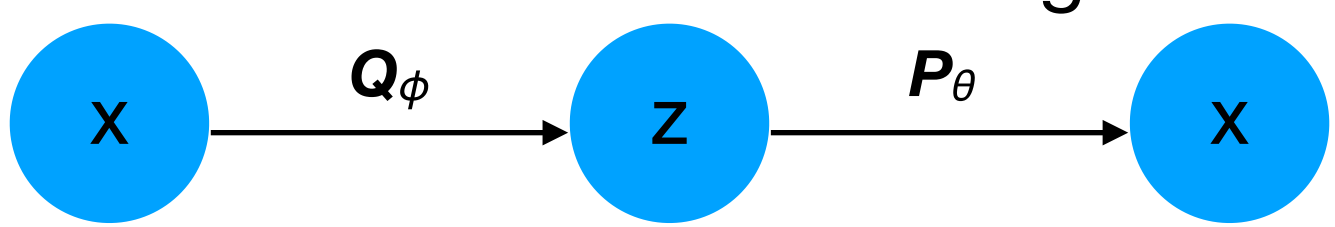
  $$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

  to:

  $$\arg\max_{\theta,\phi} g(\theta, \phi)$$ We can both estimate and (using SGD) optimize *g*!

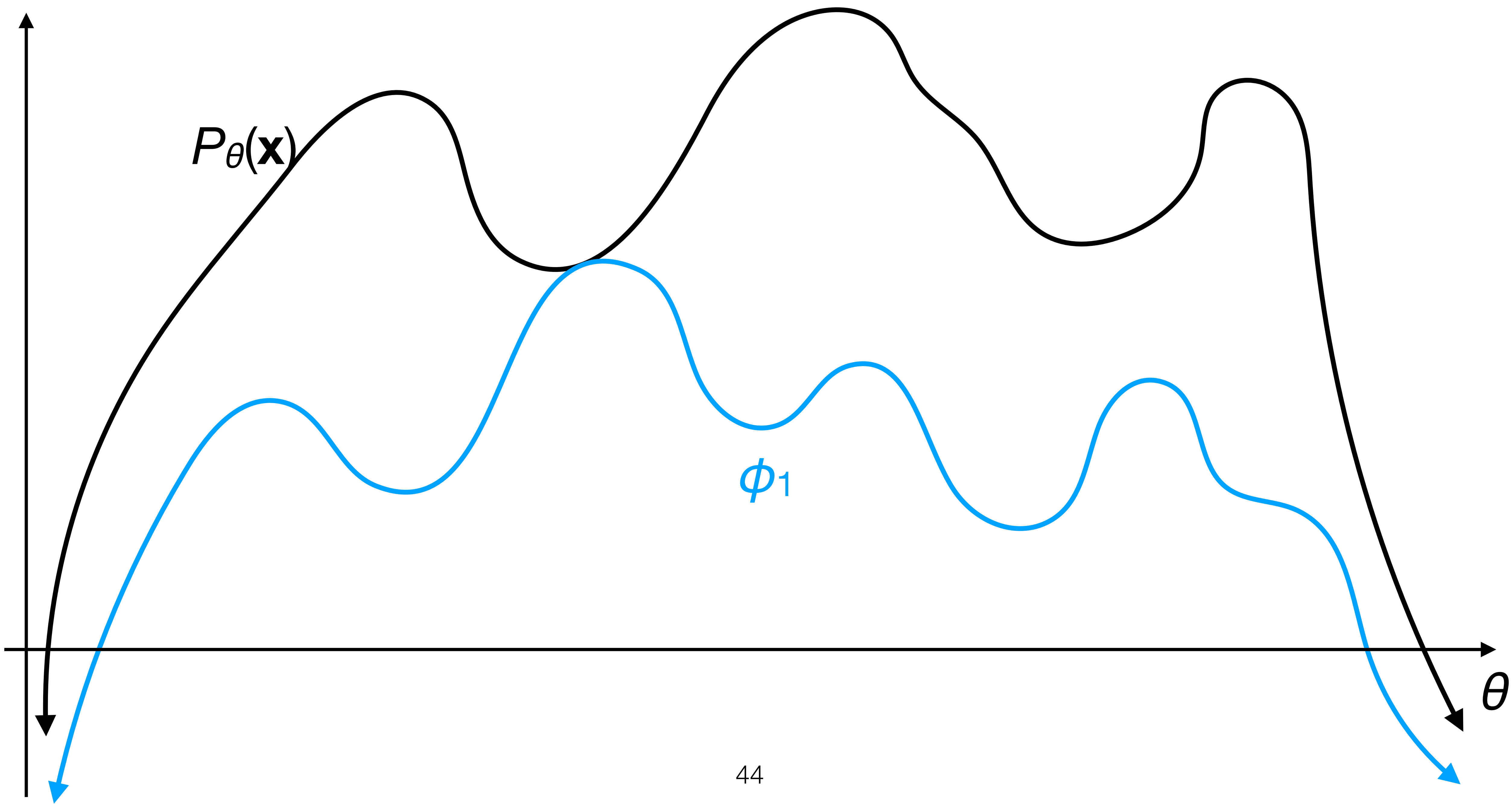
  where we can just discard ϕ afterwards.

- Under what conditions can this actually work?
  1. *g* is a lower bound for *f*.
  2. This lower bound can be made "tight" to *f*.



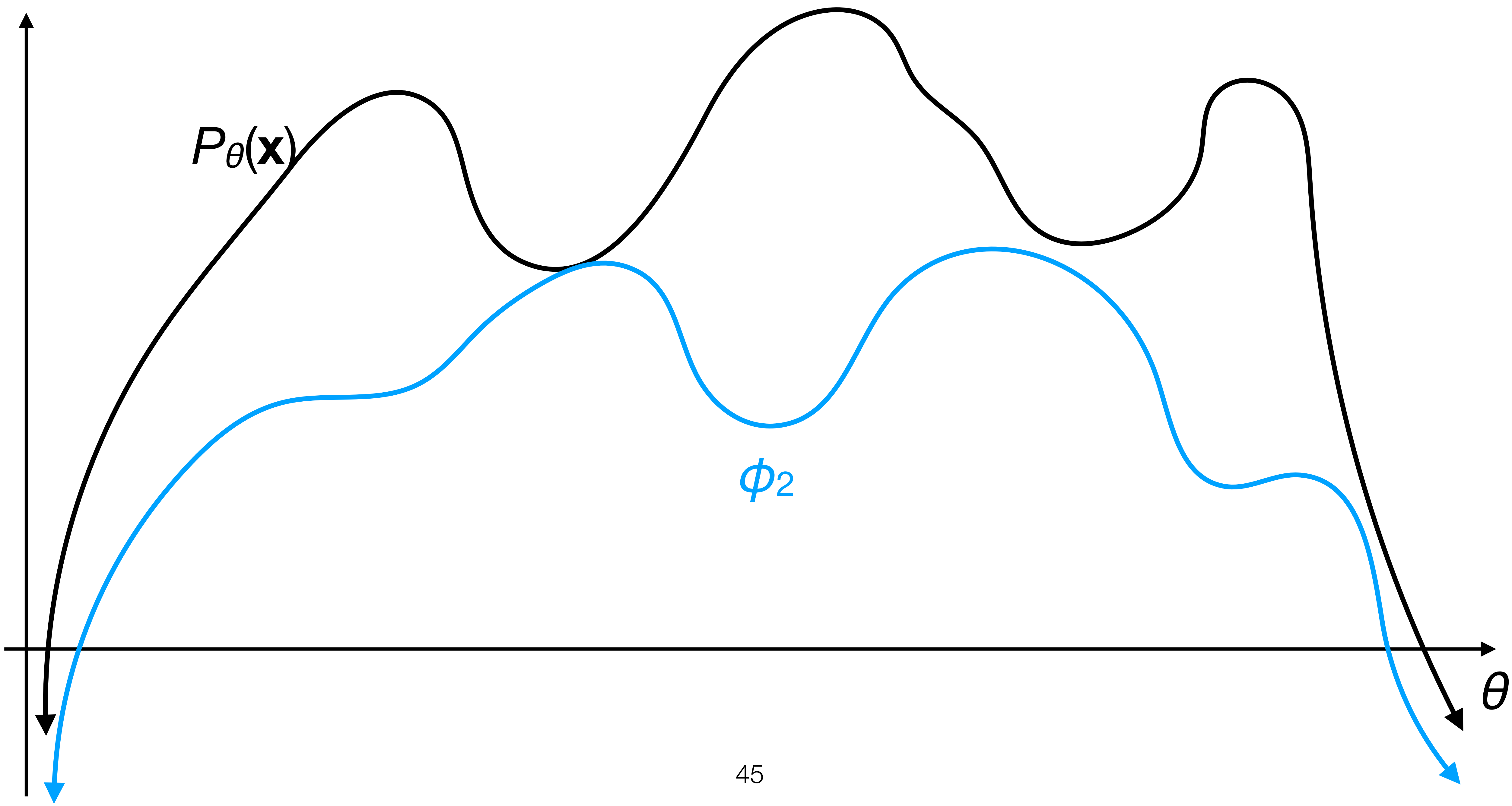
Maximizing the lower bound

- We can approximately maximize $P_\theta(\mathbf{x})$ w.r.t. θ by maximizing the lower bound w.r.t. θ and ϕ.



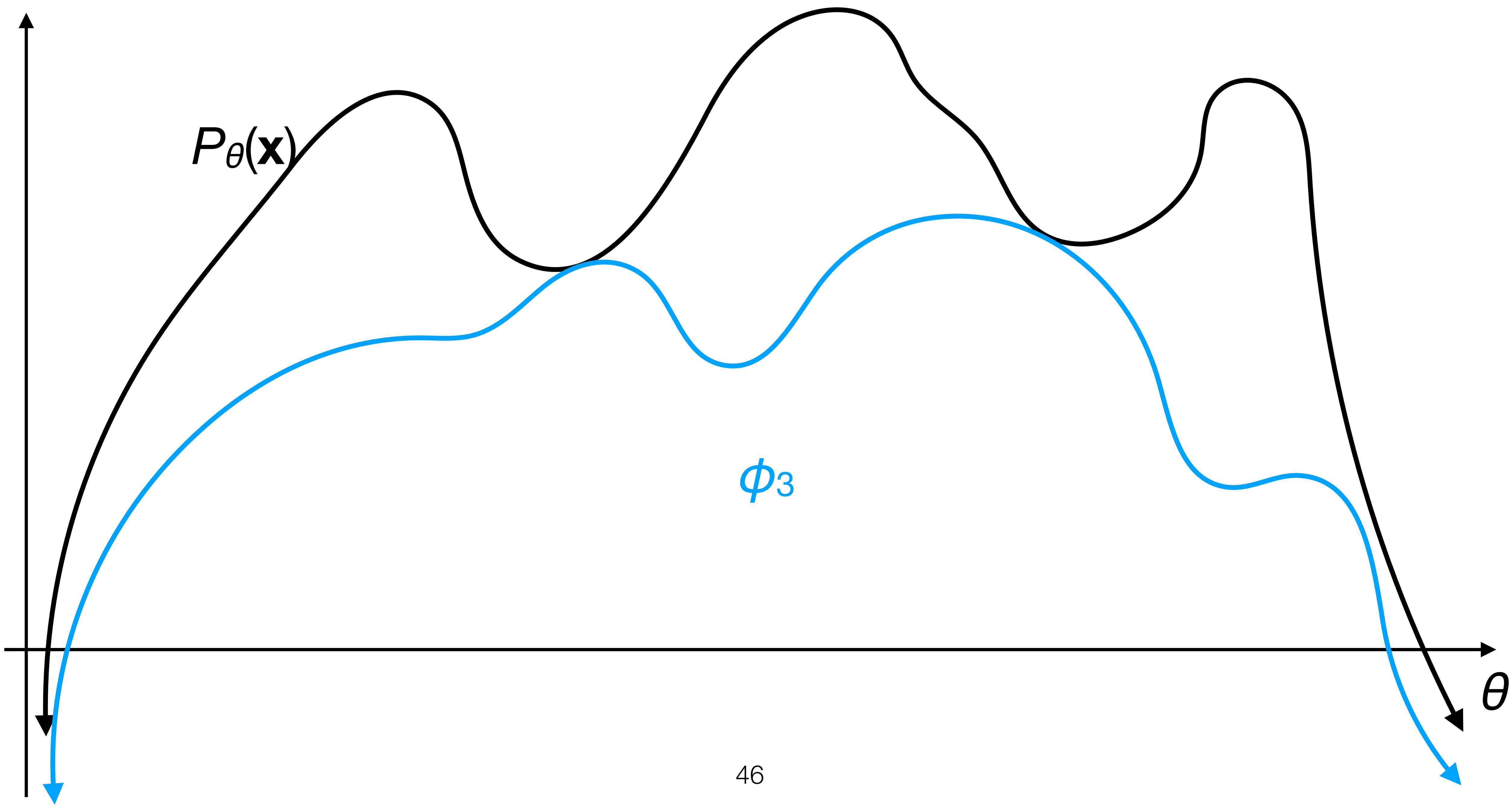
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\boldsymbol{x})$).



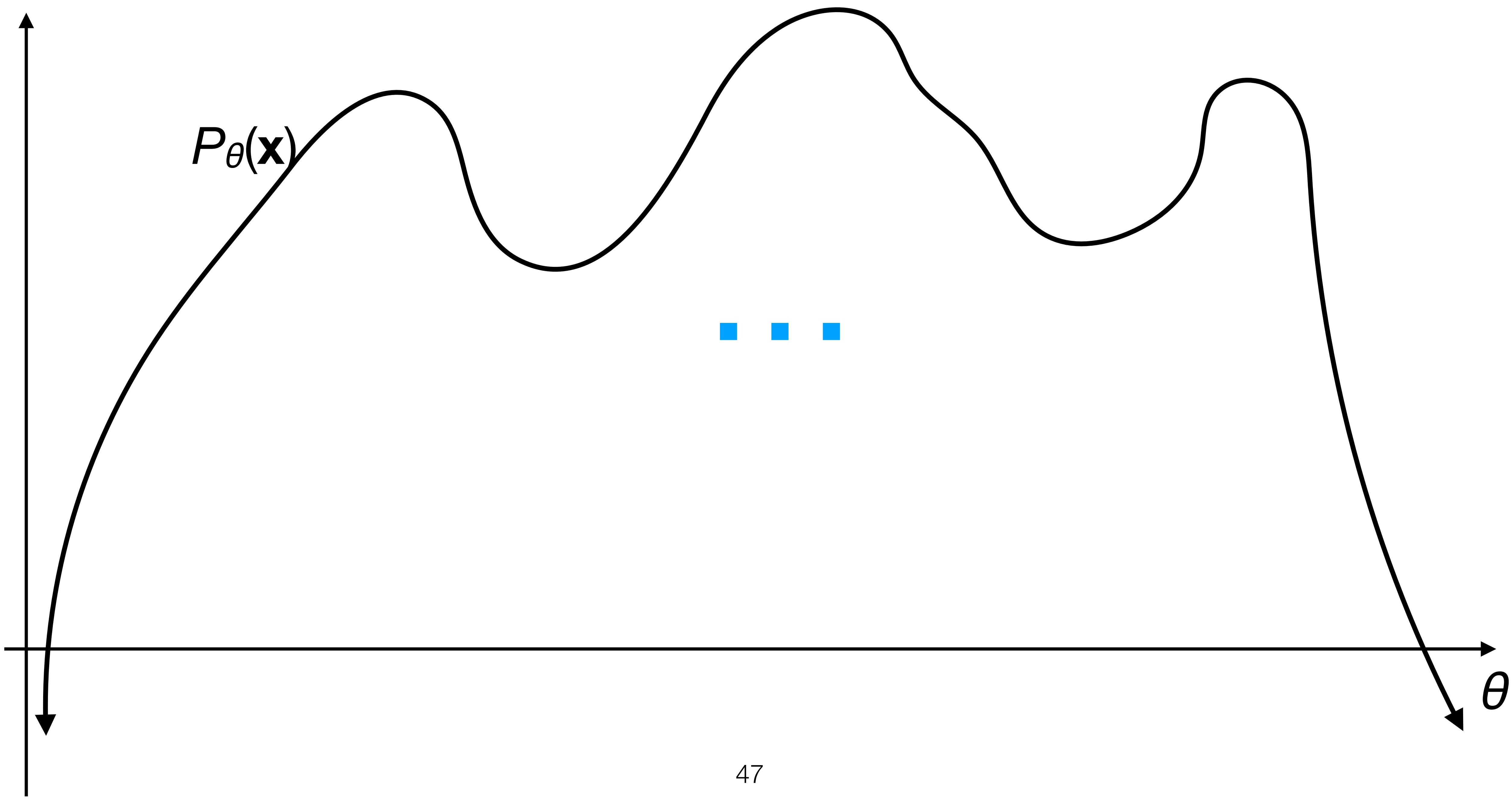
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\boldsymbol{x})$).



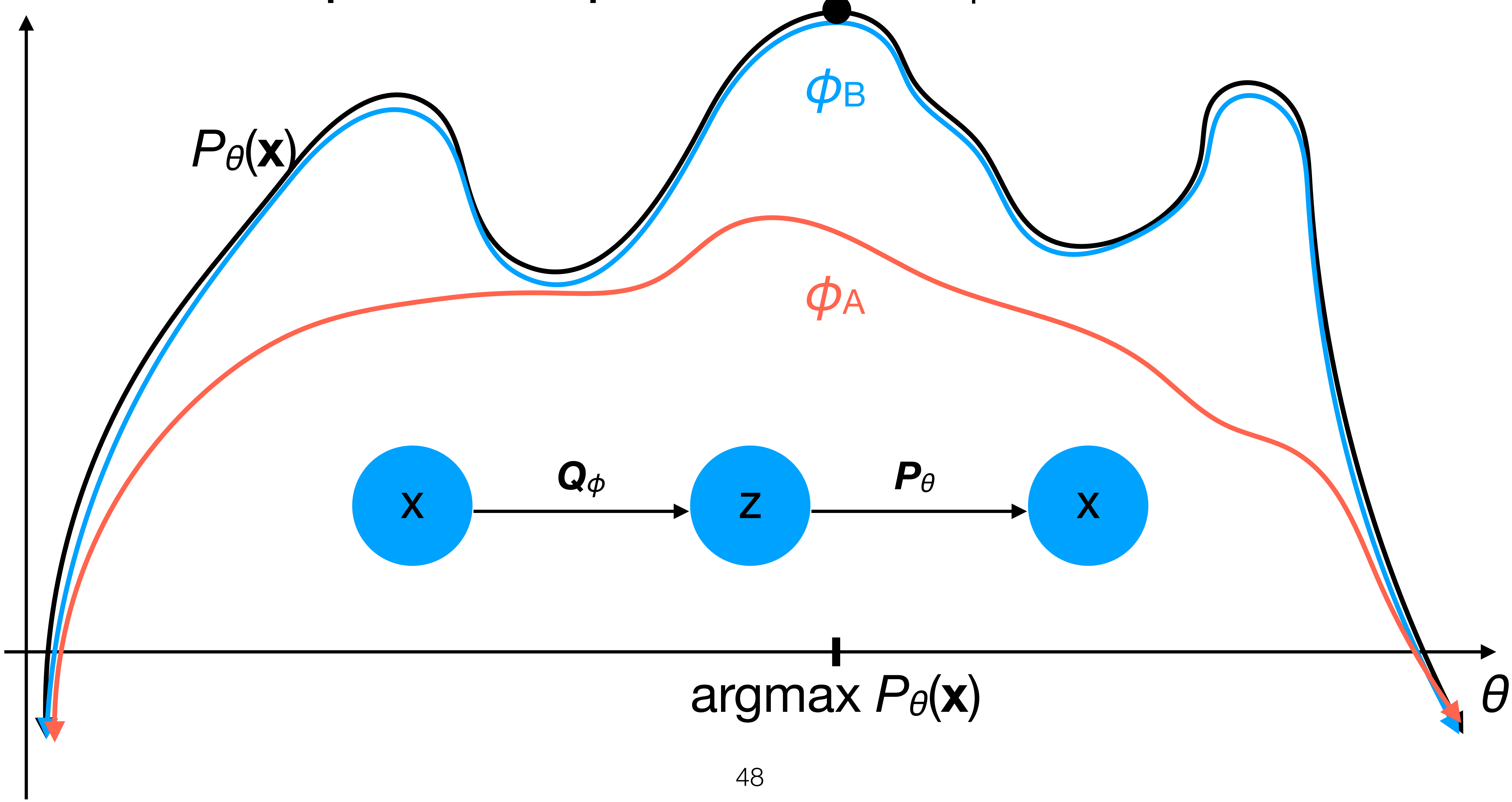
Maximizing the lower bound

- By iteratively updating Q w.r.t. ϕ, we can increase the upper bound (though it will never exceed $P_\theta(\boldsymbol{x})$).



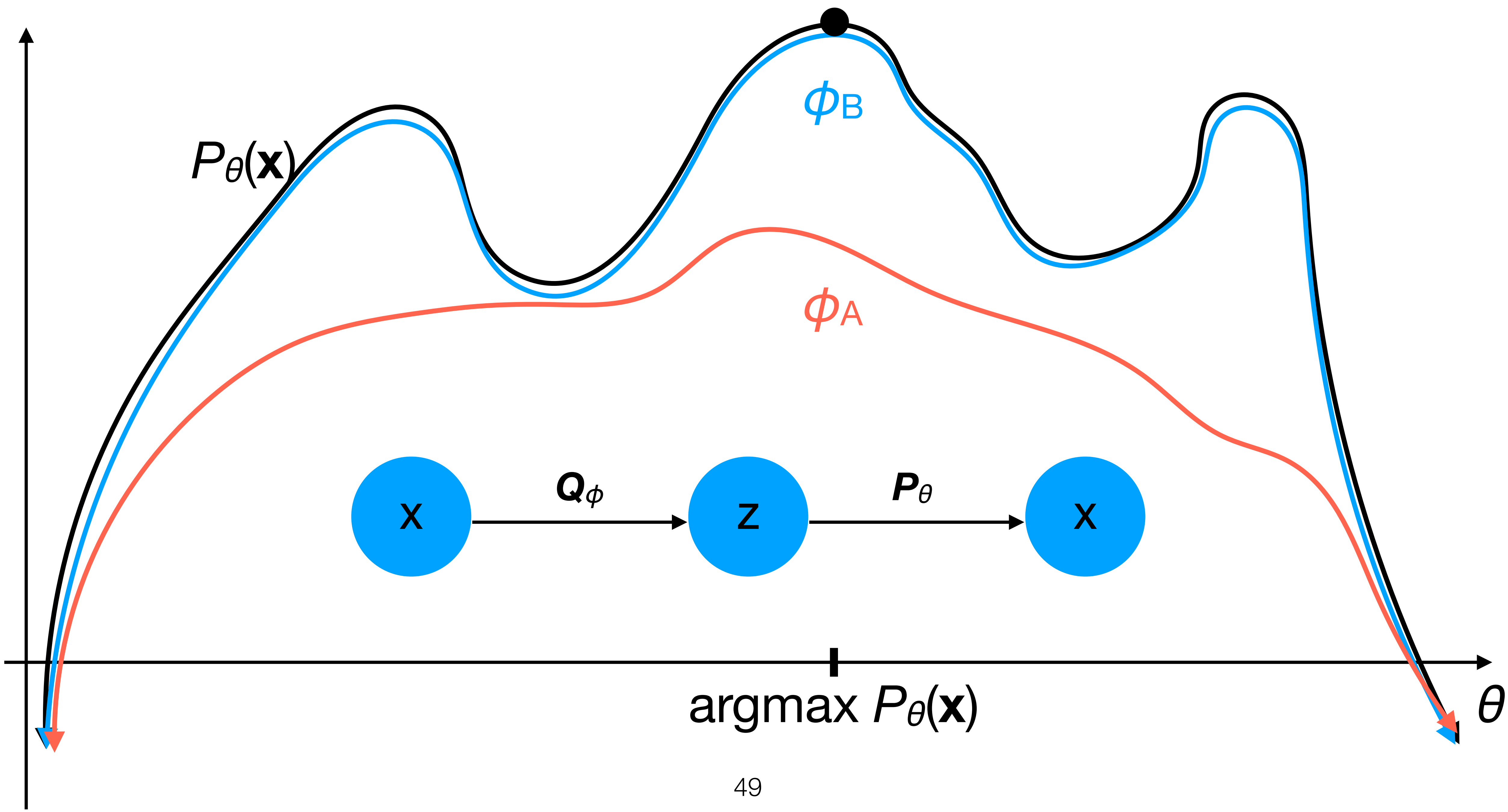
Solution

- **However**: if the approximation is "too tight", then $Q(\mathbf{z} \mid \mathbf{x}) \approx P(\mathbf{z})$ — i.e., Q "ignores" $\mathbf{x}$ altogether and adheres "too much" to $P(\mathbf{z})$.
- P thus receives no specific information about $\mathbf{x}$ to reconstruct it with — despite the tightness of the bound, SGD has no ability to find the "good" values for θ.
- This is called **posterior collapse** and is a common problem with VAEs.



Solution

- ϕ_A is a looser lower bound, but it will likely yield a better estimate (via SGD) of $\text{argmax}_\theta\, P_\theta(\mathbf{x})$ — which is what we really care about.



Avoiding Posterior Collapse

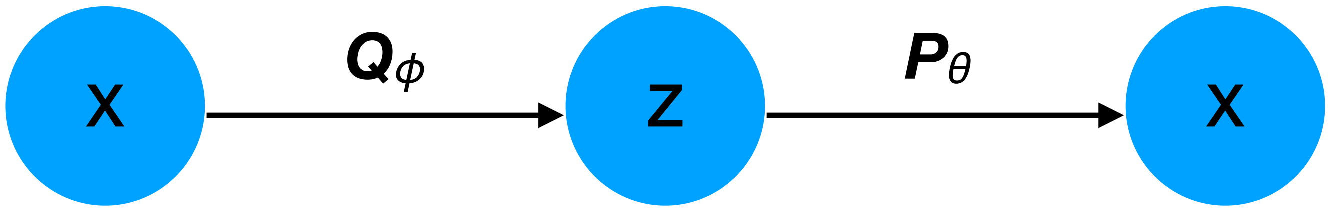
- It is common practice when training VAEs to weight (with hyperparameter *β*) how much we care about the DL term versus the reconstruction term:
  - Modified ELBO:

$$-\beta D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \log P_\theta(\mathbf{x} \mid \mathbf{z})$$

Blurry images

- VAEs are known to produce blurry images. Why?

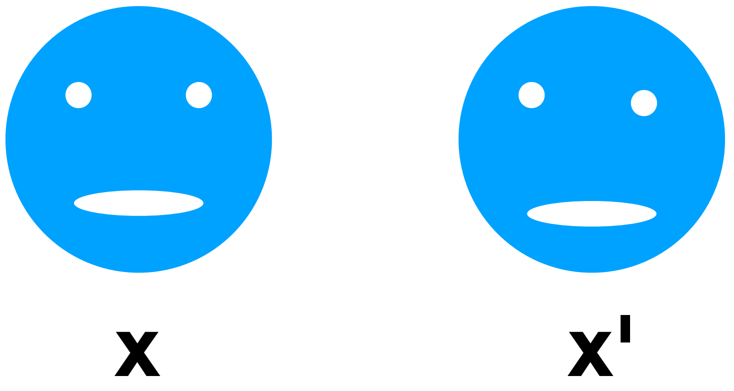
- Consider a decoder likelihood model $P(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I})$ – i.e., based on a latent **z**, the mean $\mu_{\mathbf{x}}$ of a Gaussian is chosen as the expected generated image.



Blurry images

- Now, suppose two images **x**, **x**' are similar to each other and have identical latent codes **z**.

    - Why? Because in practice, it's difficult for latent **z** to perfectly encode all features of an image **x**.

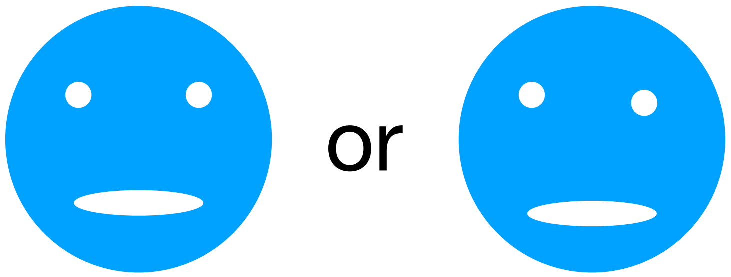


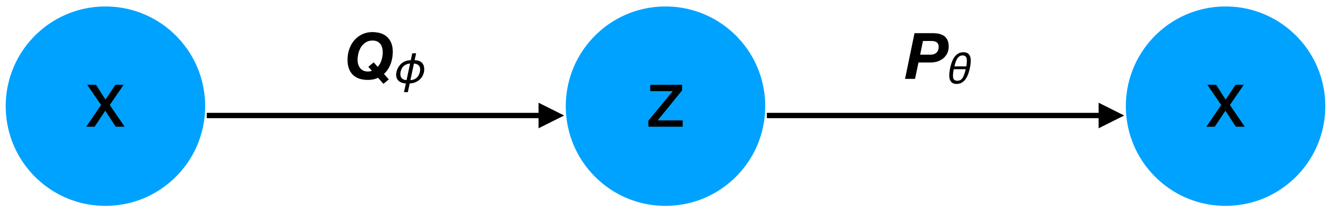
$$x \xrightarrow{Q_\phi} z \xrightarrow{P_\theta} x$$

Blurry images

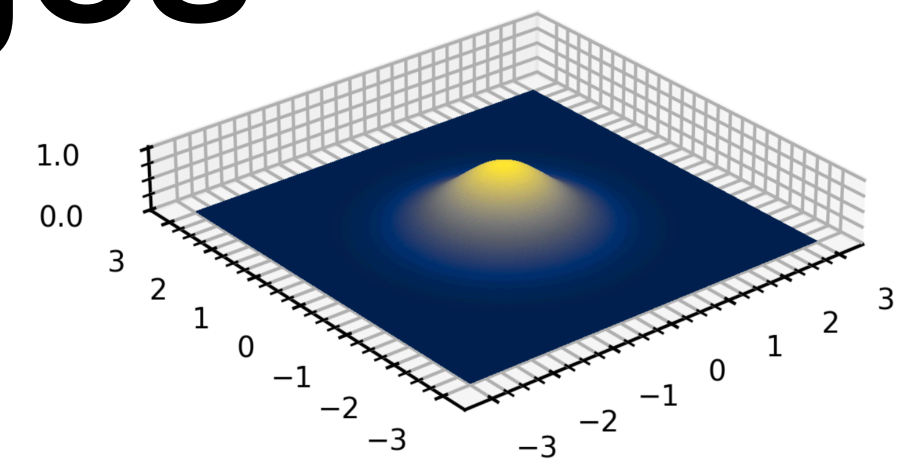
- Ideally, when sampling from $P(\mathbf{x} \mid \mathbf{z})$, we would obtain either of the two images, and they would both be “sharp” images.

$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z})$$





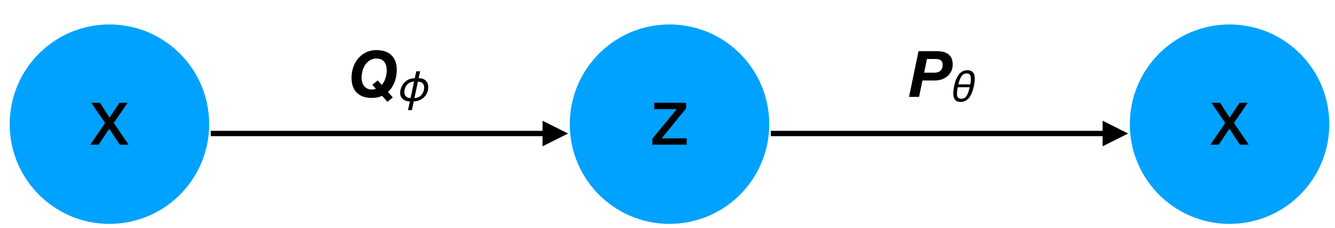
Blurry images



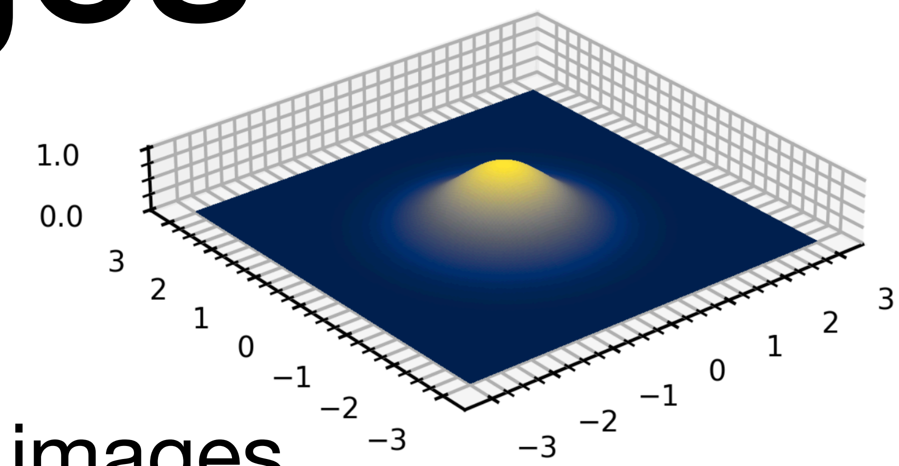
- However, a Gaussian model

  $$\mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I}) \propto \exp\left(-\frac{1}{2}|\mathbf{x} - \mu_{\mathbf{x}}|^2\right)$$

  cannot express this — it is a unimodal distribution where likelihood shrinks quickly as **x** deviates from the mean $\boldsymbol{\mu_x}$.

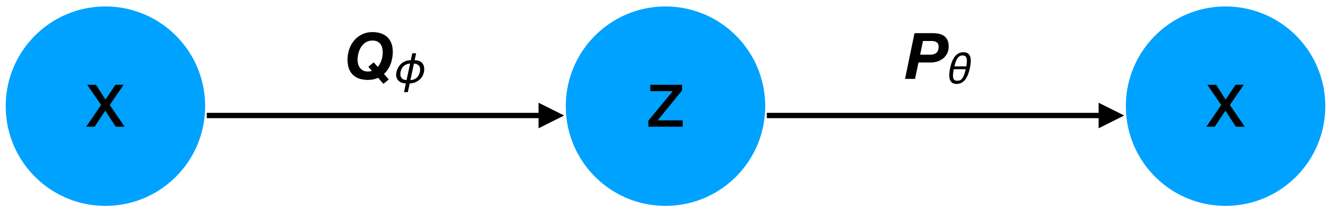


Blurry images

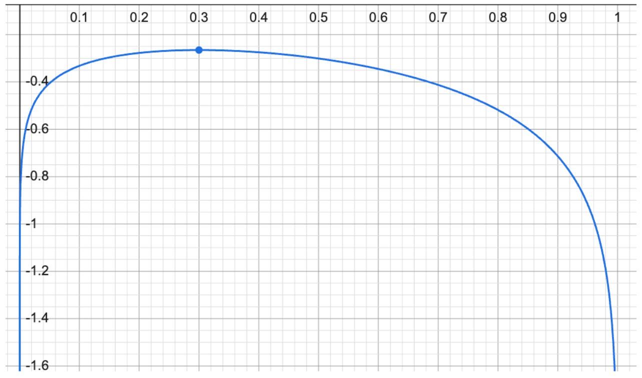


- For this reason, the decoder will maximize the likelihood over {**x**, **x**'} by picking $\boldsymbol{\mu}_x$ to be the *average* of the two images (see homework 1, part 4), creating blurry predictions.

$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z})$$

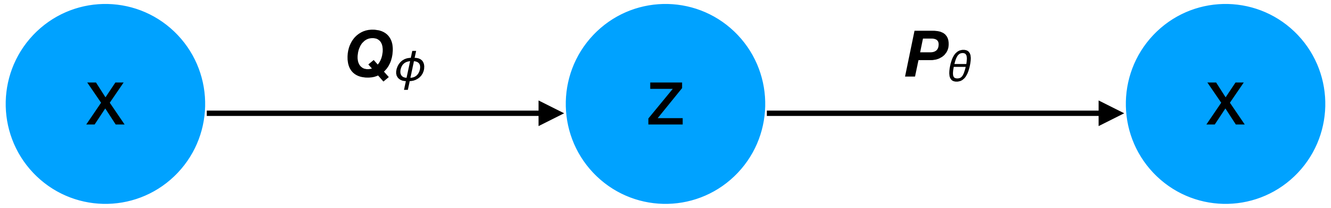


Blurry images



- The same problem exists for other likelihood models and reconstruction losses as well, e.g., binary cross-entropy.

$$\mathbf{x} \sim P(\mathbf{x} \mid \mathbf{z})$$



Group exercise

- Please do **not** use ChatGPT (or another AI tool) on this exercise.
- Download `vae_exercise.py` from Canvas->Files.
- Answer the following questions:
    1. Where/what are all the bugs in the code?
    2. Which lines (which might not be contiguous!) of code implement the expectation $\mathbb{E}_{Q(\mathbf{z} \mid \mathbf{x})}[P(\mathbf{x} \mid \mathbf{z})]$, and how many samples is the expectation calculated over?
    3. Which lines (which might not be contiguous!) of code implement the KL divergence? Where are the inputs to the KL term calculated, and how do they relate to the formula in the slides?
    4. What is the dimension *d* of the code **z**?
    5. How many NN linear layers are traversed in the encoder to process each **x** to produce the mean of $P(\mathbf{z} \mid \mathbf{x})$?
    6. In terms of the VAE definition from the slides, what probability distribution does `VAE.reparameterize()` sample from?

Group exercise

Prior dist. over latent code **z** $P(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$

Output of encoder Q: $(\mu^{(i)}, {\sigma^{(i)}}^2) = Q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}), \ \forall i$

ELBO: $-D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})]$

KL divergence between two diagonal Gaussians:

$$D_{\mathrm{KL}}(Q(\mathbf{z}) \parallel P(\mathbf{z})) = -\frac{1}{2}\sum_{j=1}^{m}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$