CS/DS 552: Class 3

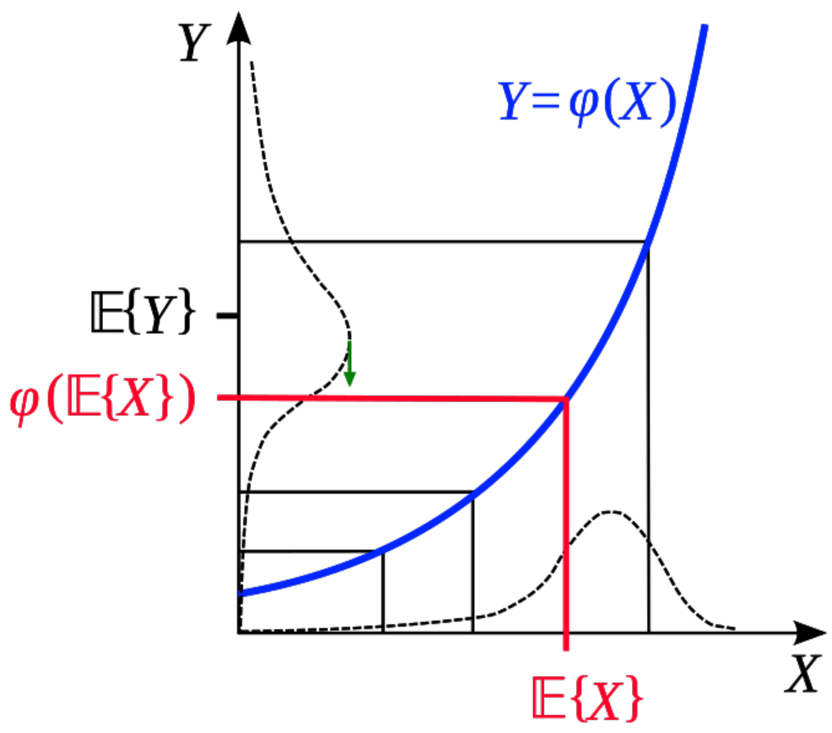
Jacob Whitehill

Probability theory: more review

Jensen's inequality

- When dealing with expectations of convex functions *f*, there is the handy **Jensen's inequality**:
  - $\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$
- Special case that is relevant to VAEs:
  - $\log \mathbb{E}_x[f(x)] \geq \mathbb{E}_x[\log f(x)]$

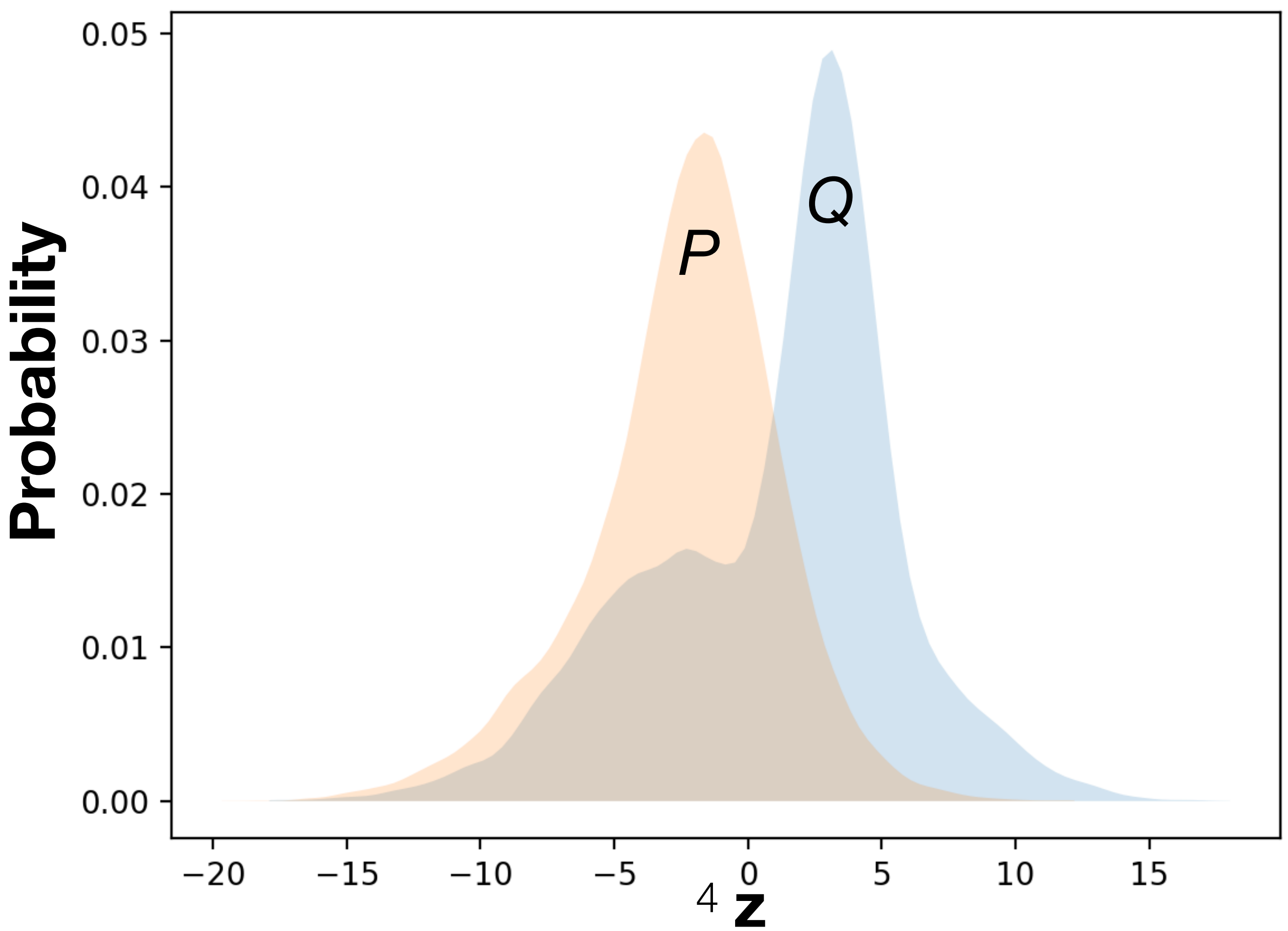
  (note the sign flips since log is *concave*)



https://en.wikipedia.org/wiki/Jensen%27s_inequality

Kullback-Leibler Divergence

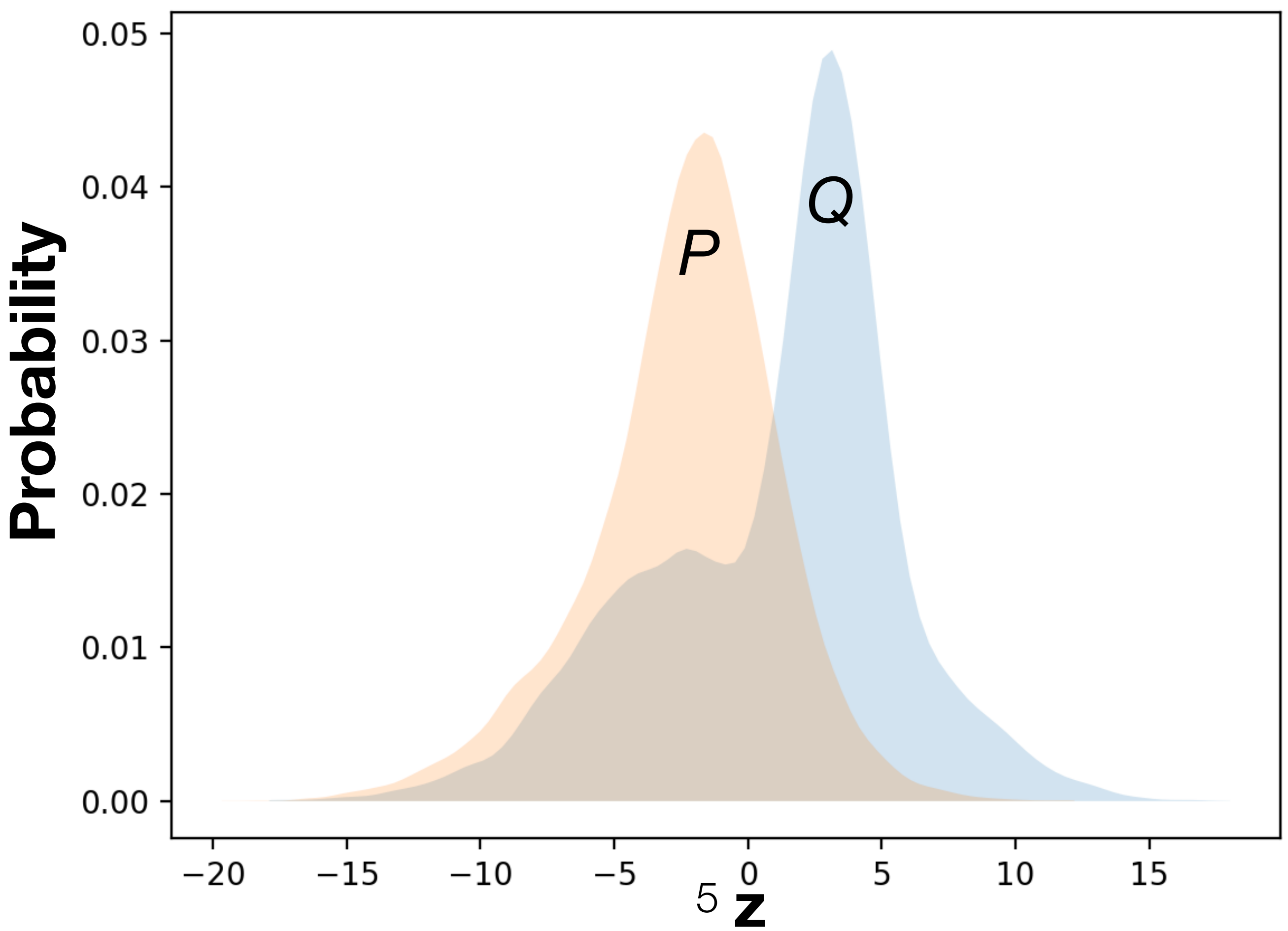
- Consider two probability distributions $P(\mathbf{z})$, $Q(\mathbf{z})$.
- How can we quantify the distance between them?



Kullback-Leibler Divergence

- The Kullback-Leibler (KL) divergence quantifies the distance of Q from P as the log ratio of probabilities at each $\mathbf{z}$ weighted by the probability of $\mathbf{z}$ according to P.

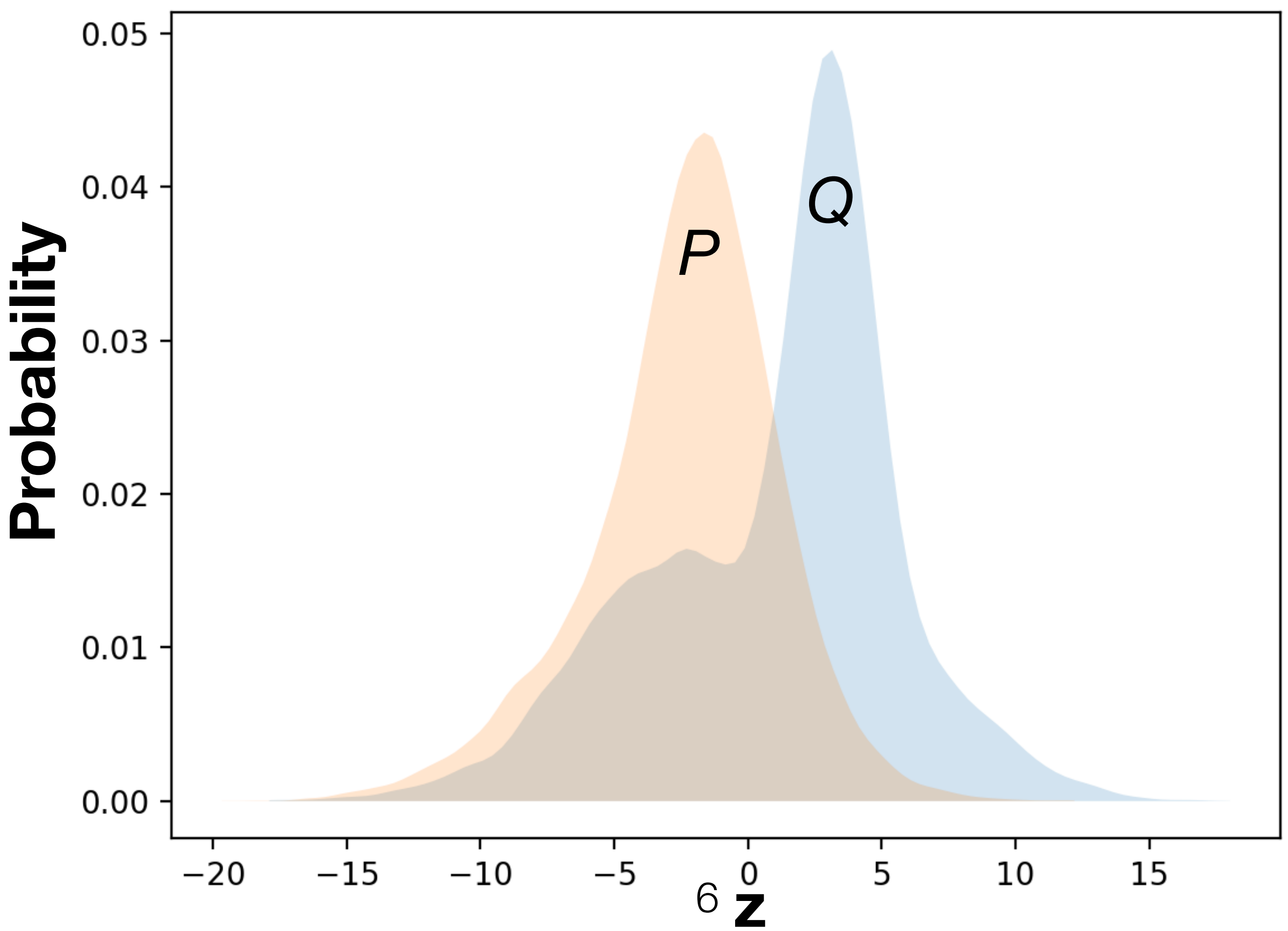
$$D_{\mathrm{KL}}(P(\mathbf{z}) \parallel Q(\mathbf{z})) = \int_{\mathbf{z}} P(\mathbf{z}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z})} d\mathbf{z}$$



Kullback-Leibler Divergence

- The Kullback-Leibler (KL) divergence quantifies the distance of Q from P as the log ratio of probabilities at each $\mathbf{z}$ weighted by the probability of $\mathbf{z}$ according to P.

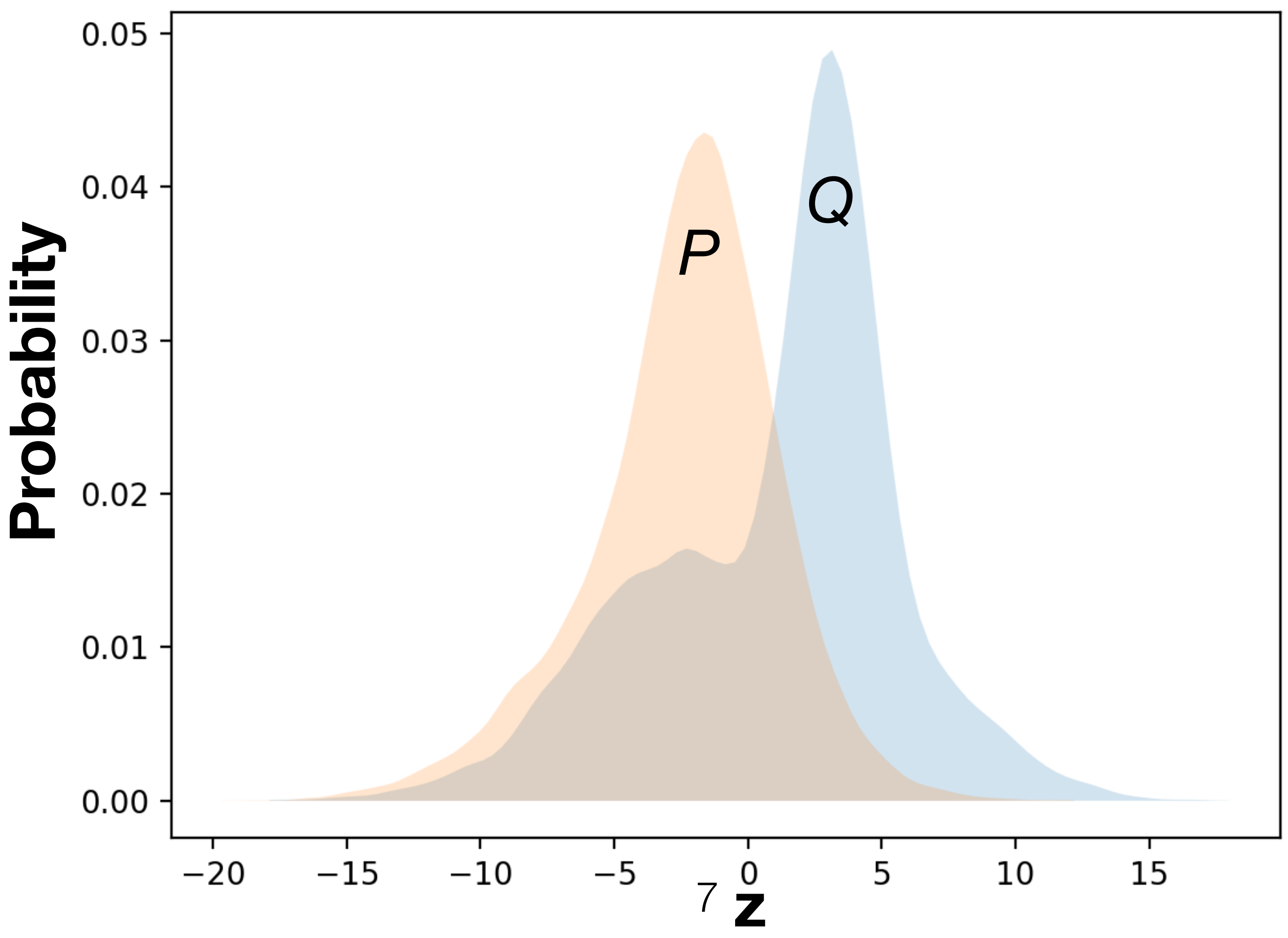
$$D_{\mathrm{KL}}(P(\mathbf{z}) \parallel Q(\mathbf{z})) = \int_{\mathbf{z}} P(\mathbf{z}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z})} d\mathbf{z}$$



Kullback-Leibler Divergence

- Note that the KL divergence is always non-negative.

$$D_{\mathrm{KL}}(P(\mathbf{z}) \parallel Q(\mathbf{z})) = \int_{\mathbf{z}} P(\mathbf{z}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z})} d\mathbf{z} \quad \geqq 0$$

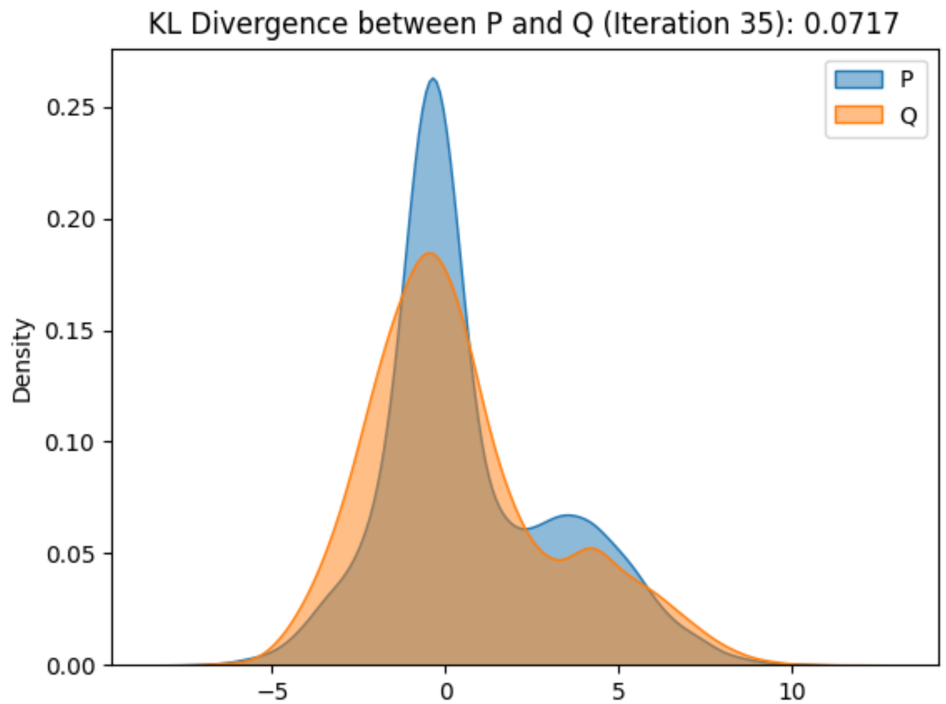


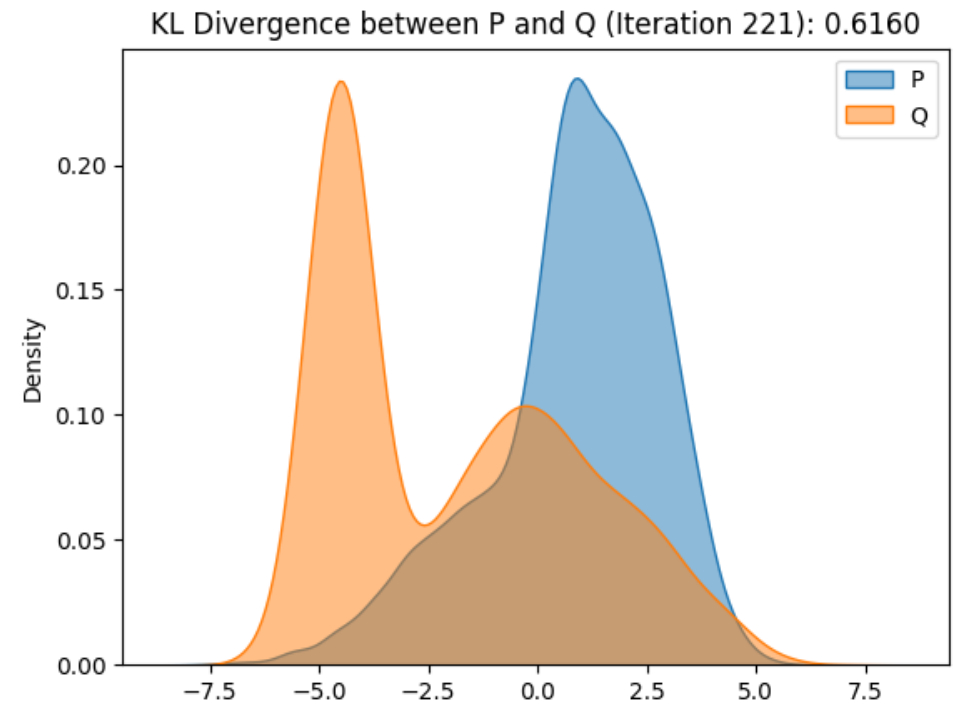
Kullback-Leibler Divergence

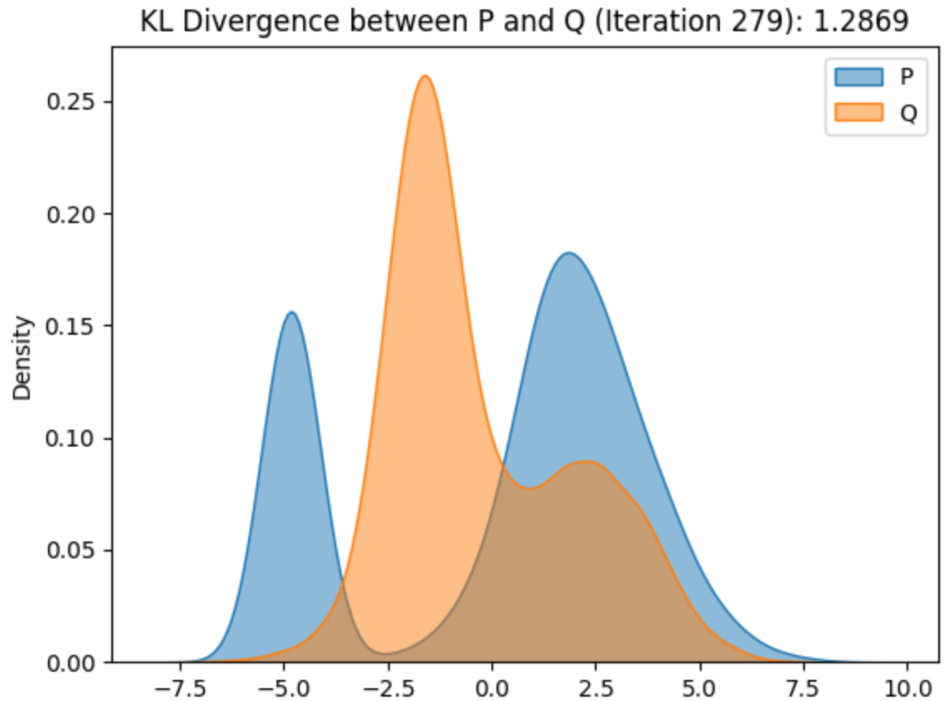
- We can also write the KL divergence as:

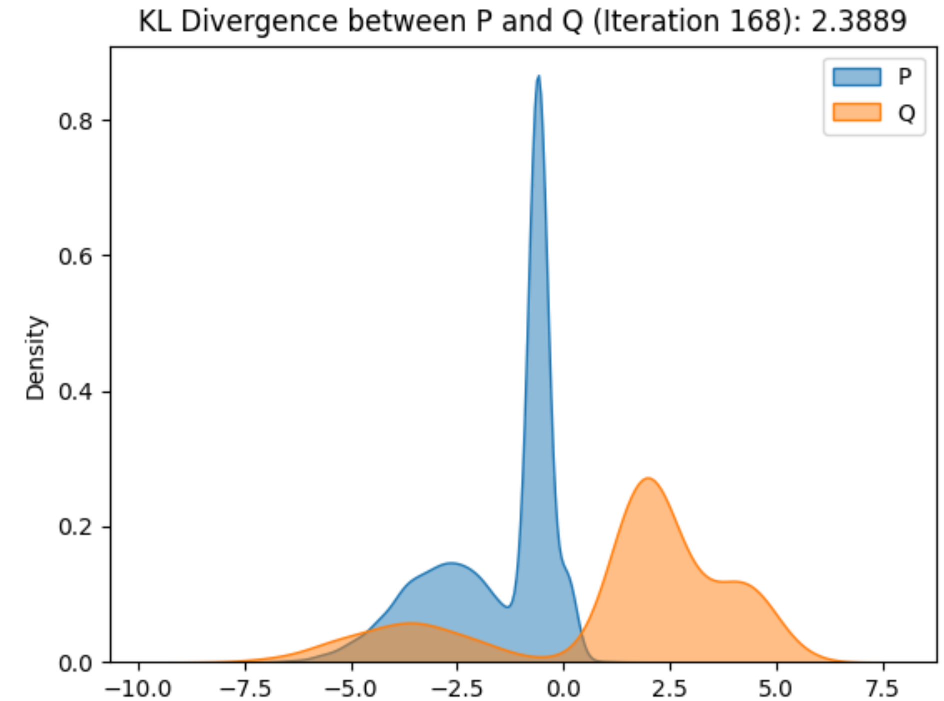
$$
\begin{aligned}
D_{\mathrm{KL}}(P(\mathbf{z}) \parallel Q(\mathbf{z})) &= \int_{\mathbf{z}} P(\mathbf{z}) \log \frac{P(\mathbf{z})}{Q(\mathbf{z})} d\mathbf{z} \\
&= -\int_{\mathbf{z}} P(\mathbf{z}) \log \frac{Q(\mathbf{z})}{P(\mathbf{z})} d\mathbf{z}
\end{aligned}
$$

Kullback-Leibler Divergence: Examples









KL-divergence for Gaussian distributions

- For the special case of two Gaussian distributions

  $$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) \text{ and } Q(\mathbf{z}) = \mathcal{N}\left(\mathbf{z}; \mu, \mathrm{diag}[\sigma_1^2, \ldots, \sigma_m^2]\right)$$

  there is a closed formula for the KL-divergence:

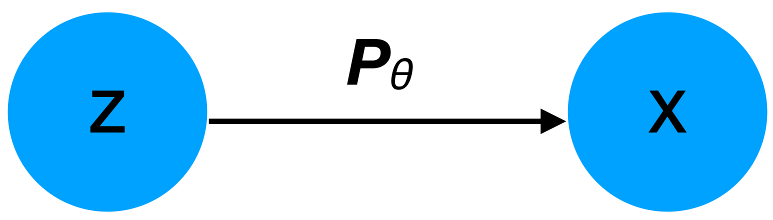
  $$D_{\mathrm{KL}}(Q(\mathbf{z}) \parallel P(\mathbf{z})) = -\frac{1}{2}\sum_{j=1}^{m}(1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)$$

- Importantly, this function is differentiable in *μ* and *σ* (this will become useful later).

Variational auto-encoders (VAEs)

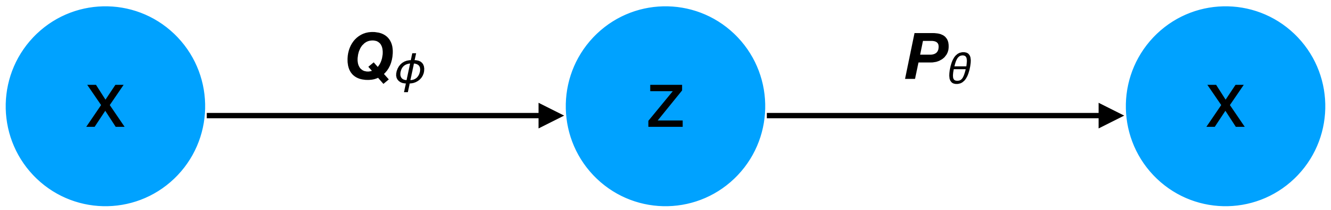
VAE architecture

- 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*.



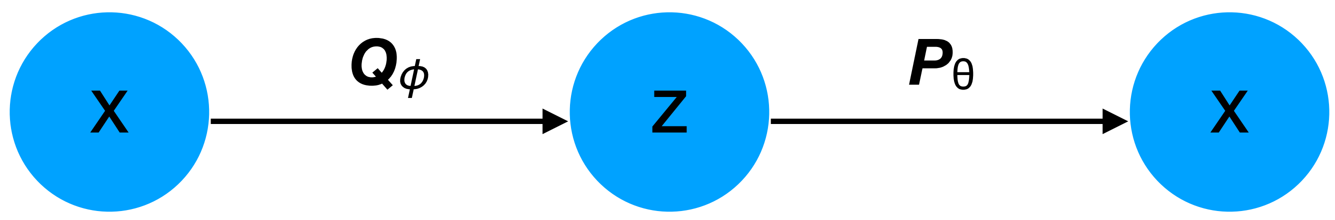
VAE architecture

- 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

- 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?

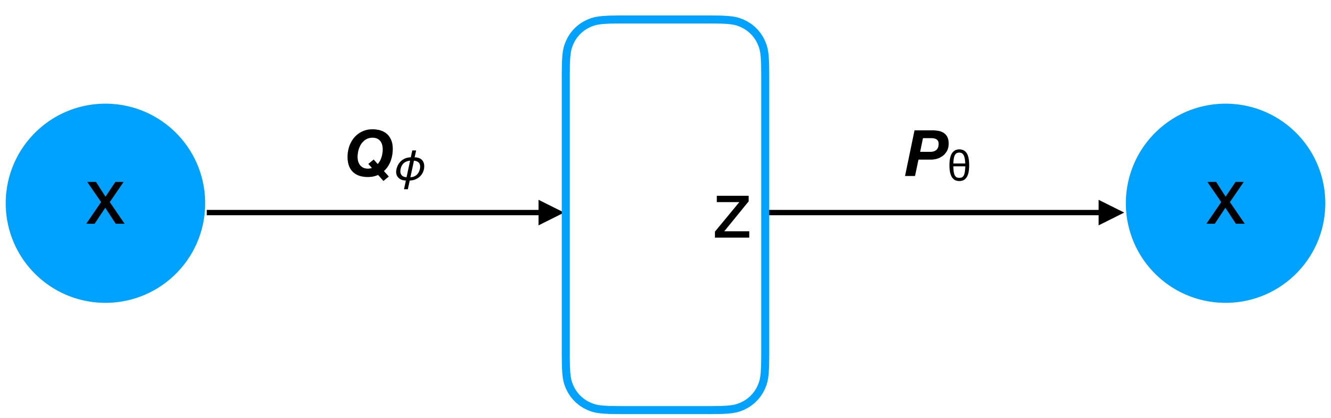


VAE architecture

- 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:



VAE architecture

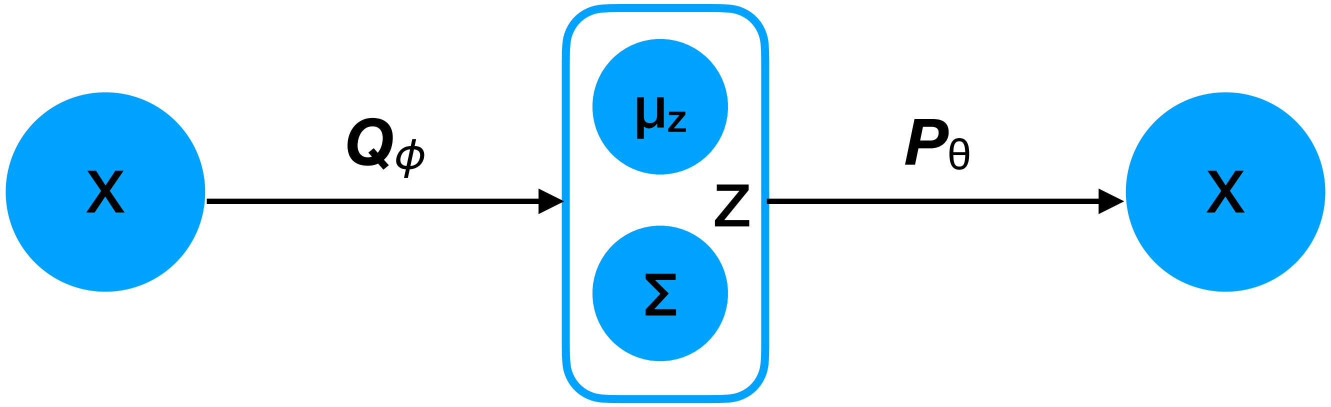
- Most commonly, Q_ϕ is constrained to output a Gaussian distribution over latent codes **z** given input **x** .

- How do we force Q_ϕ to output a Gaussian distribution?

  - Given **x**, Q_ϕ needs to output:

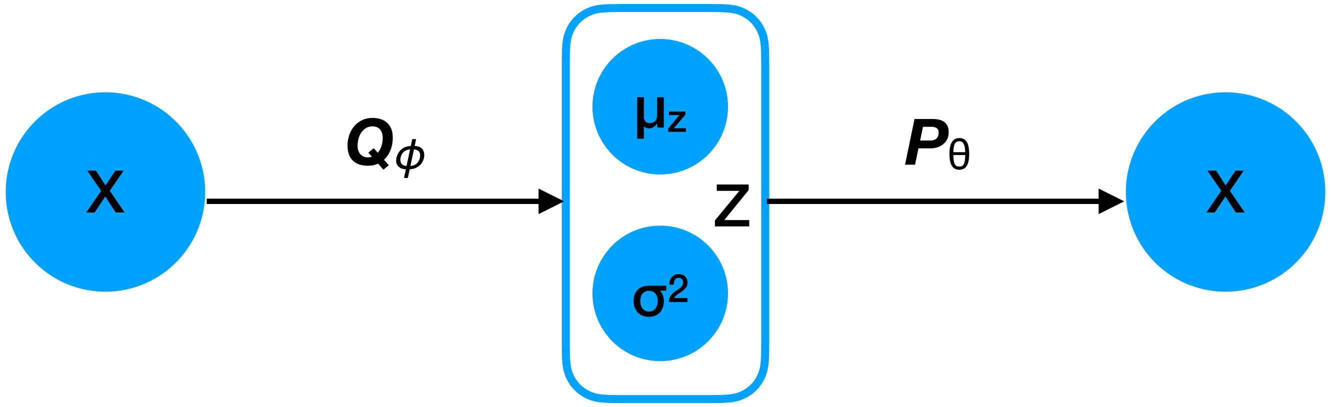
    - Mean $\boldsymbol{\mu}_\mathbf{z}$

    - Covariance matrix $\boldsymbol{\Sigma}$



VAE architecture

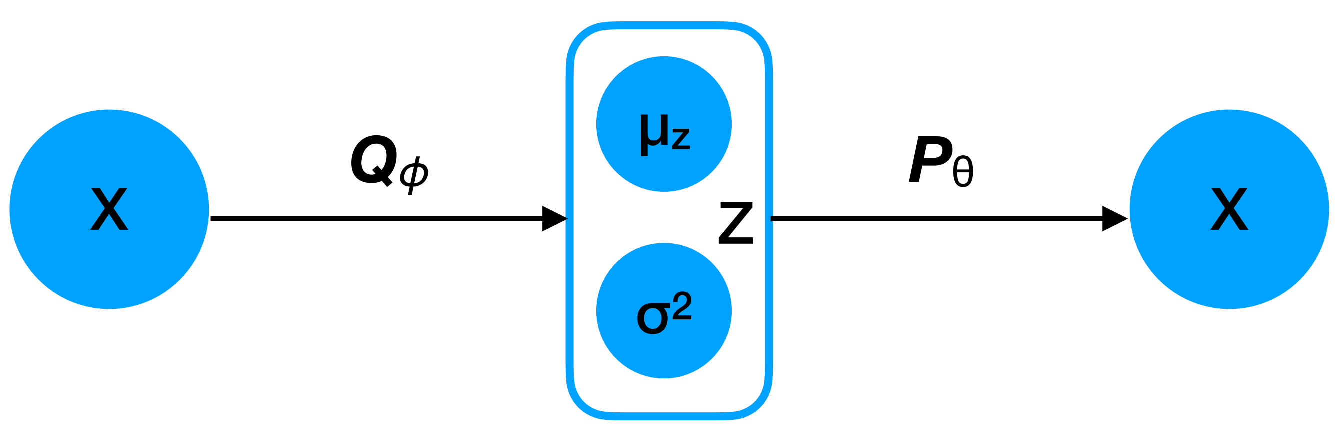
- As a simplification, Q_ϕ can output a diagonal covariance matrix parameterized by just a vector [$\sigma_1^2, \ldots, \sigma_d^2$].



VAE architecture

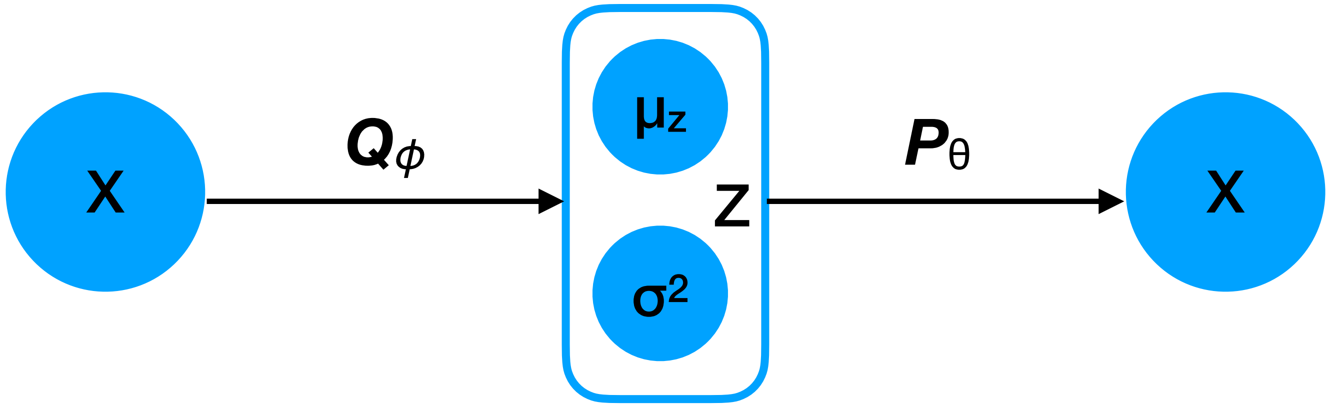
- As a simplification, Q_ϕ can output a diagonal covariance matrix parameterized by just a vector [$\sigma_1^2, \ldots, \sigma_d^2$].

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



VAE architecture

- 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.

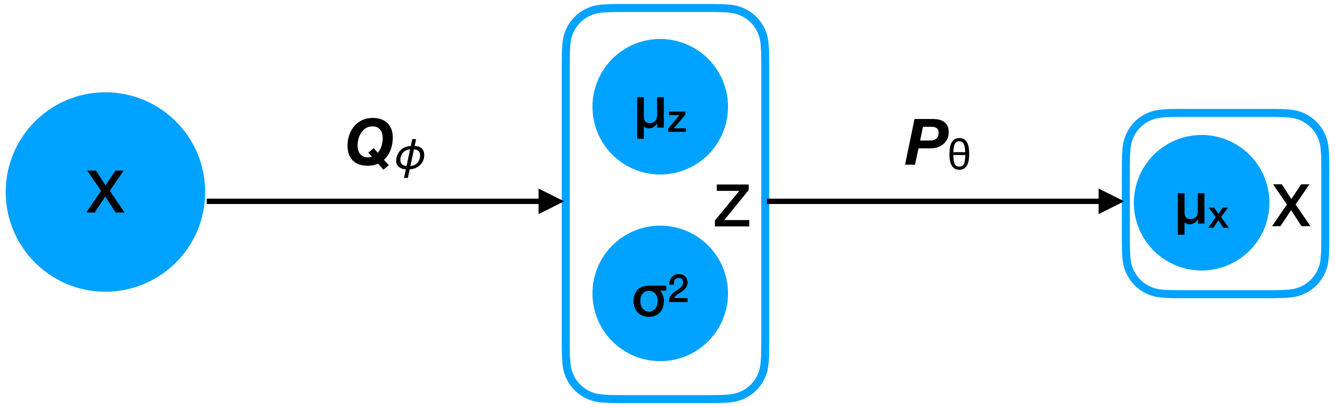


VAE architecture

- P_θ is usually one of:

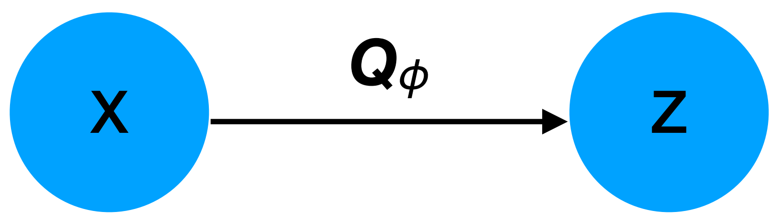
1. Gaussian $\mathcal{N}(\mu_{\mathbf{x}}, \mathbf{I})$ with mean $\boldsymbol{\mu}_{\mathbf{x}}$ and **I**-covariance
   - Useful for predicting data from $\mathbb{R}^m$.
   - NN: no activation function.

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



VAE architecture

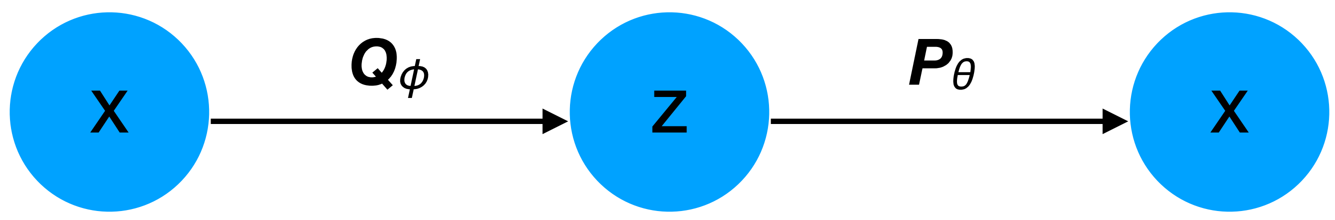
- With Q_ϕ, we can estimate from input $\mathbf{x} \in \mathbb{R}^m$ the latent code $\mathbf{z} \in \mathbb{R}^d$ that generated it.



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 $\mathbf{x} \in \mathbb{R}^m$.



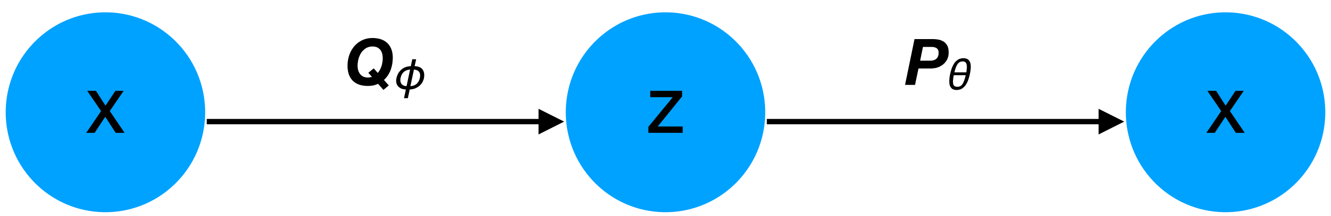
VAE architecture

- 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 $\mathbf{x} \in \mathbb{R}^m$.

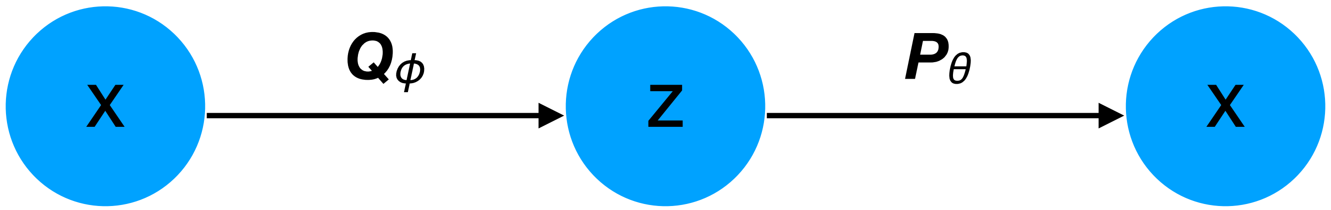
- $P(\mathbf{x} \mid \mathbf{z})$ represents how likely the VAE believes a particular example $\mathbf{x}$ is to be generated from latent code $\mathbf{z}$.

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



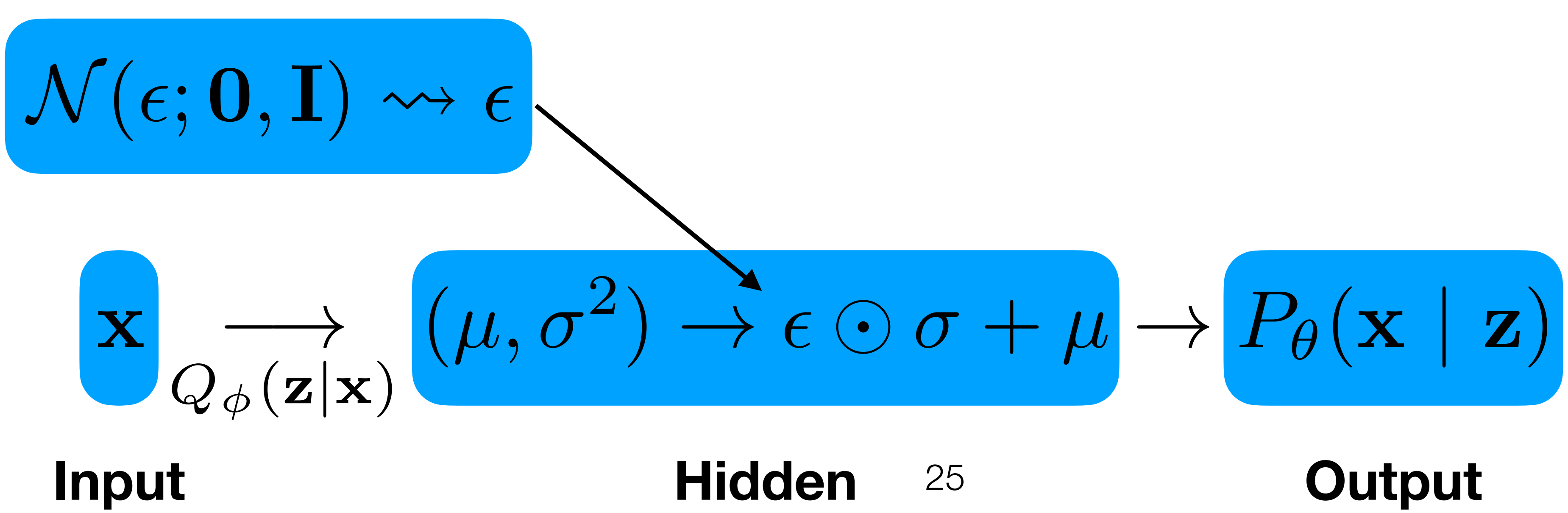
VAE architecture

- 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.



Training procedure

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

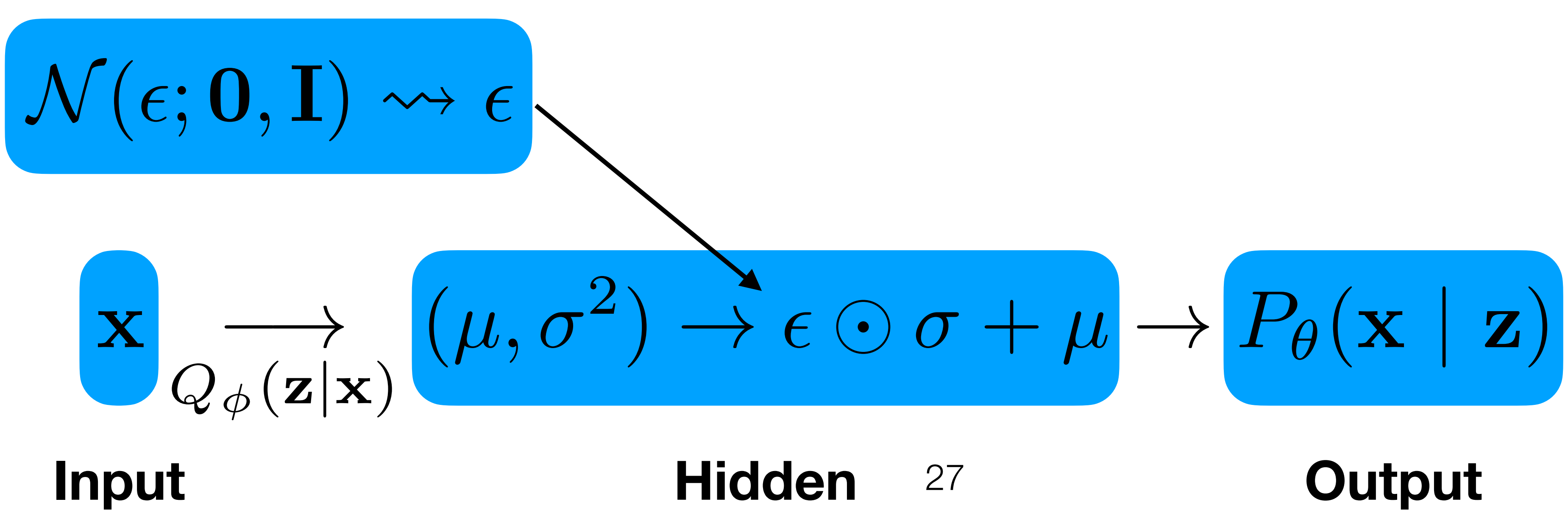
$$\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}) \rightsquigarrow \epsilon$$

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

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: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^d$



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: $\mathbf{z}^{(i)} = \epsilon^{(i)} \odot \sigma^{(i)} + \mu^{(i)}$
  - Compute likelihood:
    $$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

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: $\mathbf{z}^{(i)} = \epsilon^{(i)} \odot \sigma^{(i)} + \mu^{(i)}$
  - Compute likelihood:
    $$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$
  - Update ϕ and θ using back-propagation.

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

  - The reconstruction probability is computed w.r.t. each dimension of $\mathbf{x}^{(i)} \in \mathbb{R}^m$ and then summed, e.g.:
    $\mathbf{x}^{(i)}$ = `[0.1,  0.8,  0.7,  0.23, 0.5, …]` // Ground-truth
    $\hat{\mathbf{x}}^{(i)}$ = `[0.08, 0.83, 0.58, 0.21, 0.42, …]` // E[$\mathbf{x}^{(i)}$ | $\mathbf{z}^{(i)}$]

$$\boxed{\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})} - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

    - The reconstruction probability is computed w.r.t. each dimension of $\mathbf{x}^{(i)} \in \mathbb{R}^m$ and then summed, e.g.:

$\mathbf{x}^{(i)}$ = `[0.1,  0.8,  0.7,  0.23, 0.5, …]` // Ground-truth
$\hat{\mathbf{x}}^{(i)}$ = `[0.08, 0.83, 0.58, 0.21, 0.42, …]` // E[$\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}$]

Log-like.

$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

  - The reconstruction probability is computed w.r.t. each dimension of $\mathbf{x}^{(i)} \in \mathbb{R}^m$ and then summed, e.g.:

$\mathbf{x}^{(i)}$ = `[0.1,  0.8,  0.7,  0.23, 0.5, …]` // Ground-truth
$\hat{\mathbf{x}}^{(i)}$ = `[0.08, 0.83, 0.58, 0.21, 0.42, …]` // E[$\mathbf{x}^{(i)}$ | $\mathbf{z}^{(i)}$]

Log-like.

$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

  - The reconstruction probability is computed w.r.t. each dimension of $\mathbf{x}^{(i)} \in \mathbb{R}^m$ and then summed, e.g.:

x$^{(i)}$ = `[0.1,  0.8,  0.7,  0.23, 0.5, …]` // Ground-truth
$\hat{\mathbf{x}}^{(i)}$ = `[0.08, 0.83, 0.58, 0.21, 0.42, …]` // E[**x**$^{(i)}$ | **z**$^{(i)}$]

Log-like.

$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

  - The reconstruction probability is computed w.r.t. each dimension of $\mathbf{x}^{(i)} \in \mathbb{R}^m$ and then summed, e.g.:
    $\mathbf{x}^{(i)}$ = `[0.1,  0.8,  0.7,  0.23, 0.5, …]` // Ground-truth
    $\hat{\mathbf{x}}^{(i)}$ = `[0.08, 0.83, 0.58, 0.21, 0.42, …]` // E[$\mathbf{x}^{(i)}$ | $\mathbf{z}^{(i)}$]
    ...

$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

Optimization objective

- Note that, for the VAE, we want to maximize a likelihood instead of minimizing a loss.

- The likelihood consists of two components:

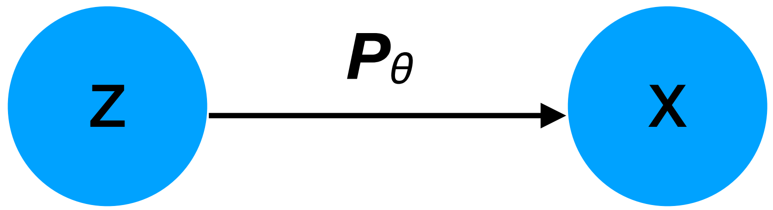
  - The KL-divergence is differentiable w.r.t. μ and σ, which in turn are differentiable w.r.t. ϕ.

$$\log P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - D_{\mathrm{KL}}(Q_\phi(\mathbf{z}^{(i)};\ \mathbf{x}^{(i)}) \parallel \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}))$$

VAE: MLE Derivation

VAE: MLE derivation

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

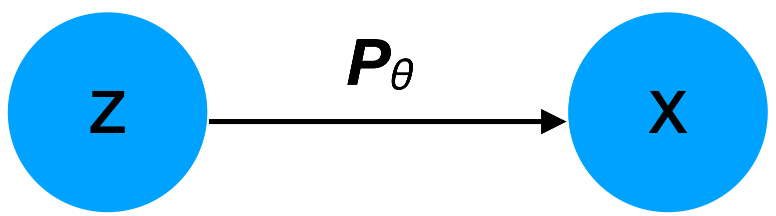


VAE: MLE derivation

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

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

Definition of conditional probability

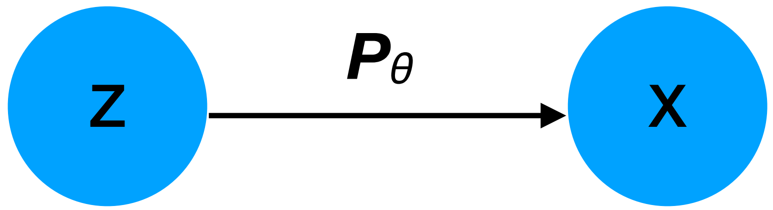


VAE: MLE derivation

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

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

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

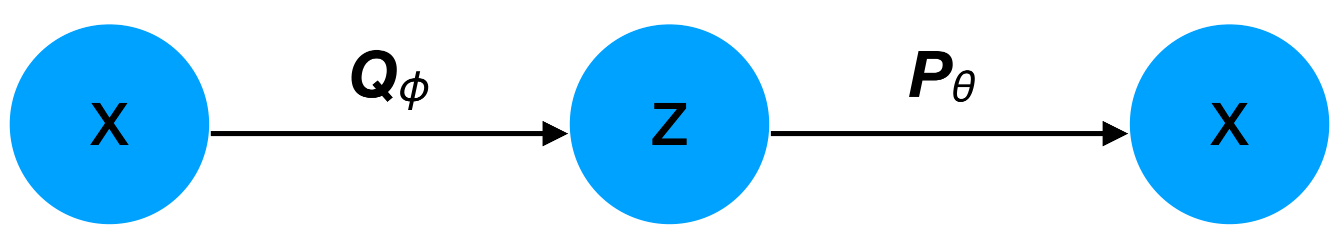


VAE: MLE derivation

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

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

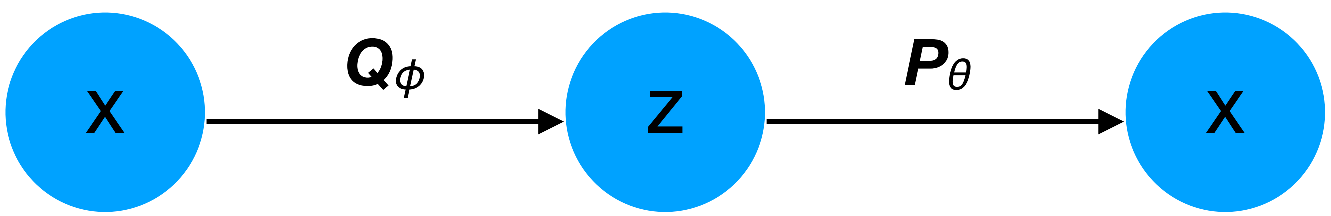
- By introducing auxiliary parameters ϕ through an encoder network Q, we actually make the optimization easier…



VAE: MLE derivation

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

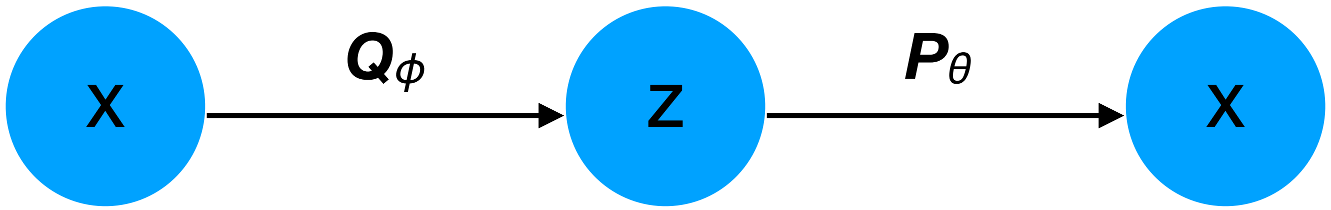
This holds for any non-zero Q.



VAE: MLE derivation

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

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



VAE: MLE derivation

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

- 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

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

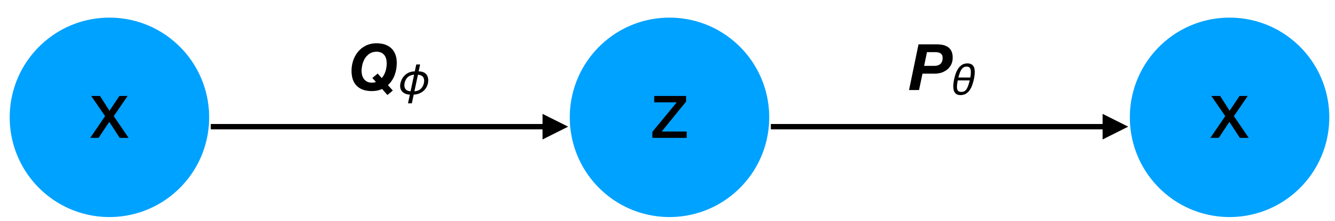
Definitions of KL-divergence and expectation.

VAE: MLE derivation

- In other words: to maximize *P*(**x**), we want to:
  - Minimize KL-divergence of hidden state w.r.t. standard normal distribution.

    and

  - Maximize the reconstruction probability.



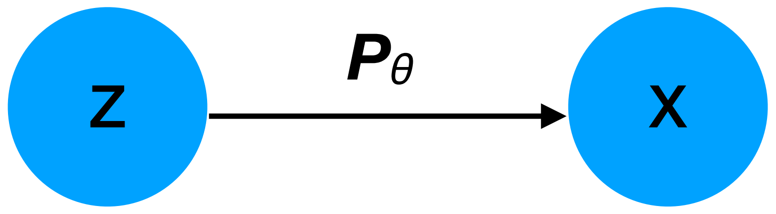
$$= -D_{\mathrm{KL}}(Q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel P(\mathbf{z})) + \mathbb{E}_{Q_\phi}[\log P_\theta(\mathbf{x} \mid \mathbf{z})]$$

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

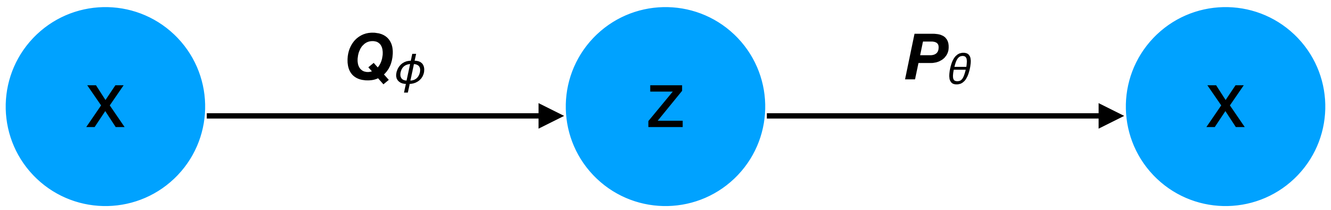
- Conceptually, we have changed from an optimization:

$$\arg\max_{\theta} f(\theta)$$ 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:

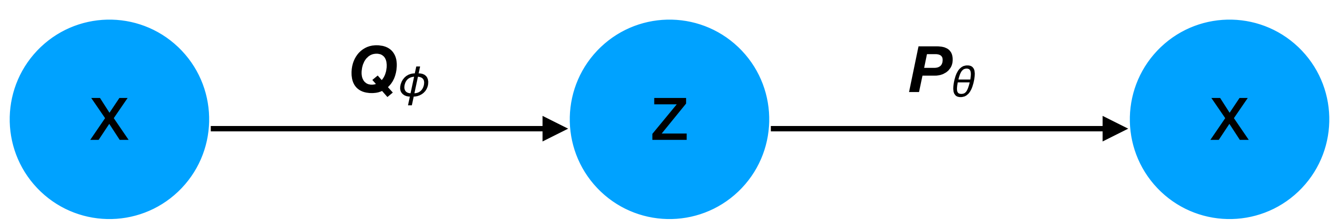
$$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

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:

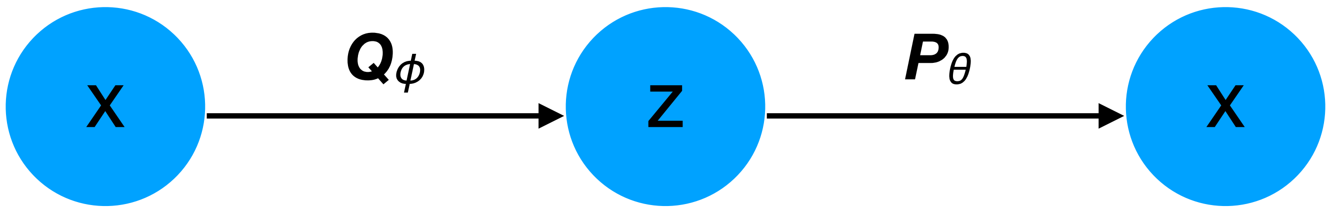
$$\arg\max_{\theta} f(\theta)$$ We cannot even evaluate *f*!

to:

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

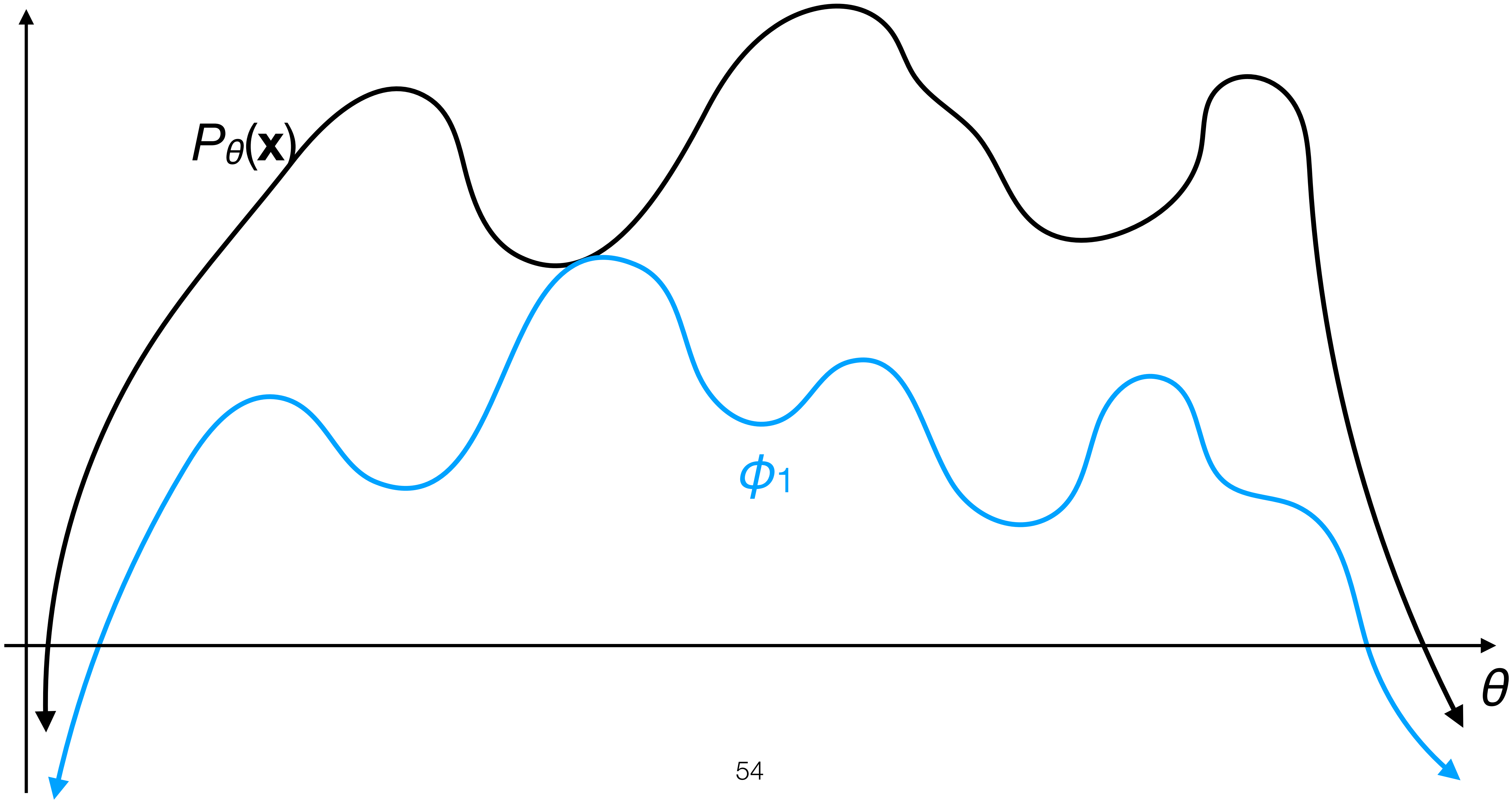
where we can just discard ϕ afterwards.

- 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*.



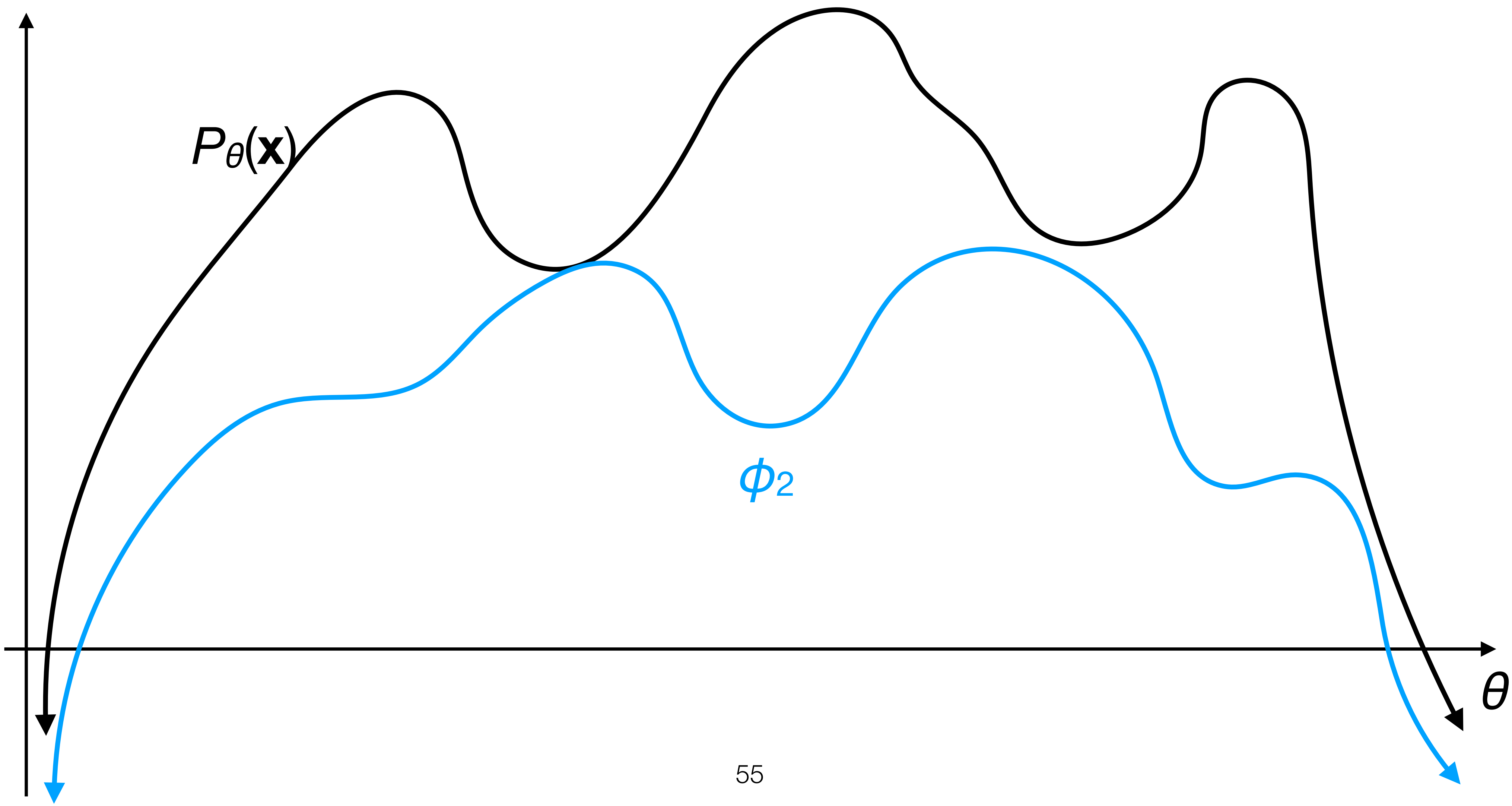
Maximizing the lower bound

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



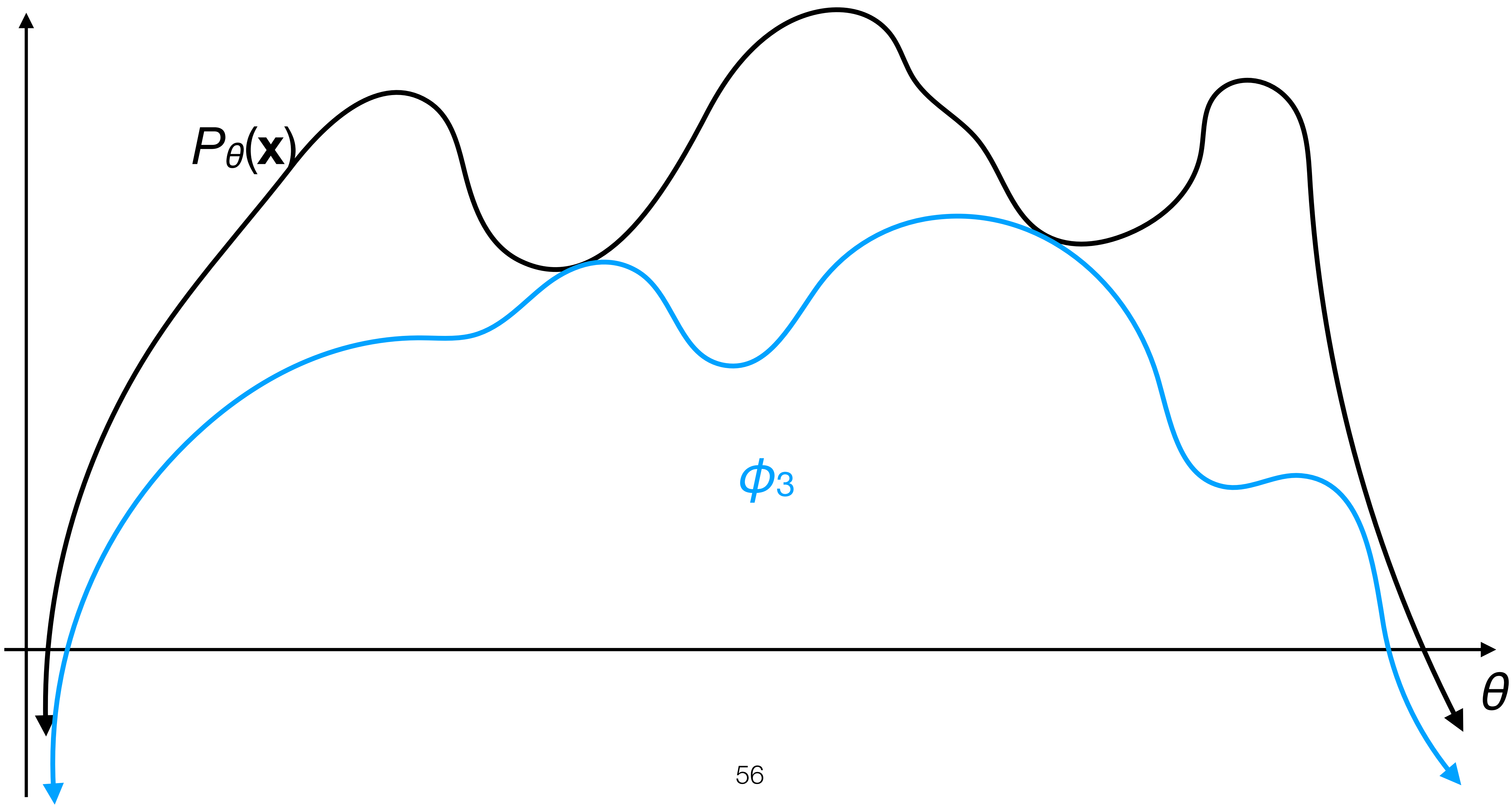
Maximizing the lower bound

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



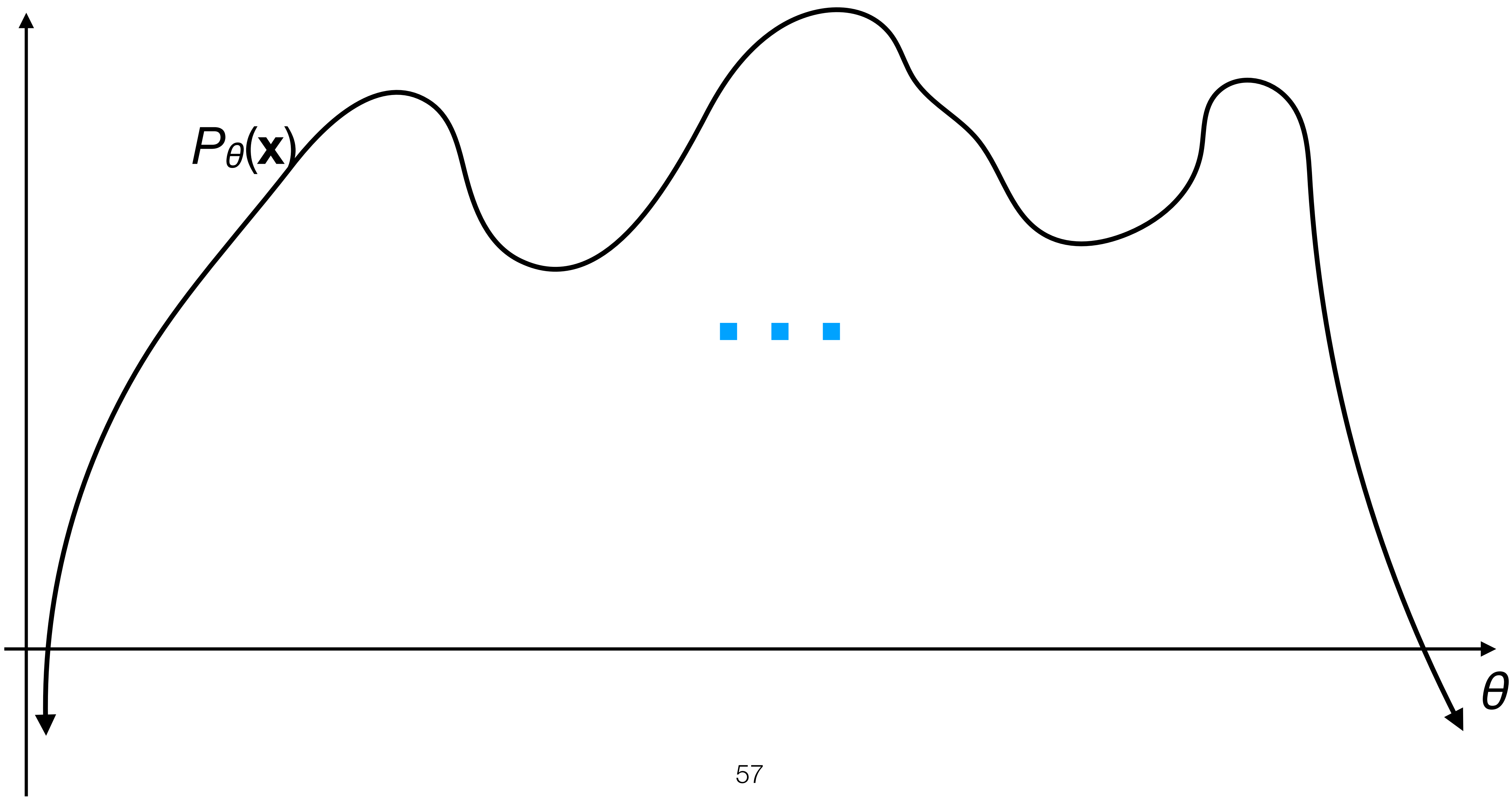
Maximizing the lower bound

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



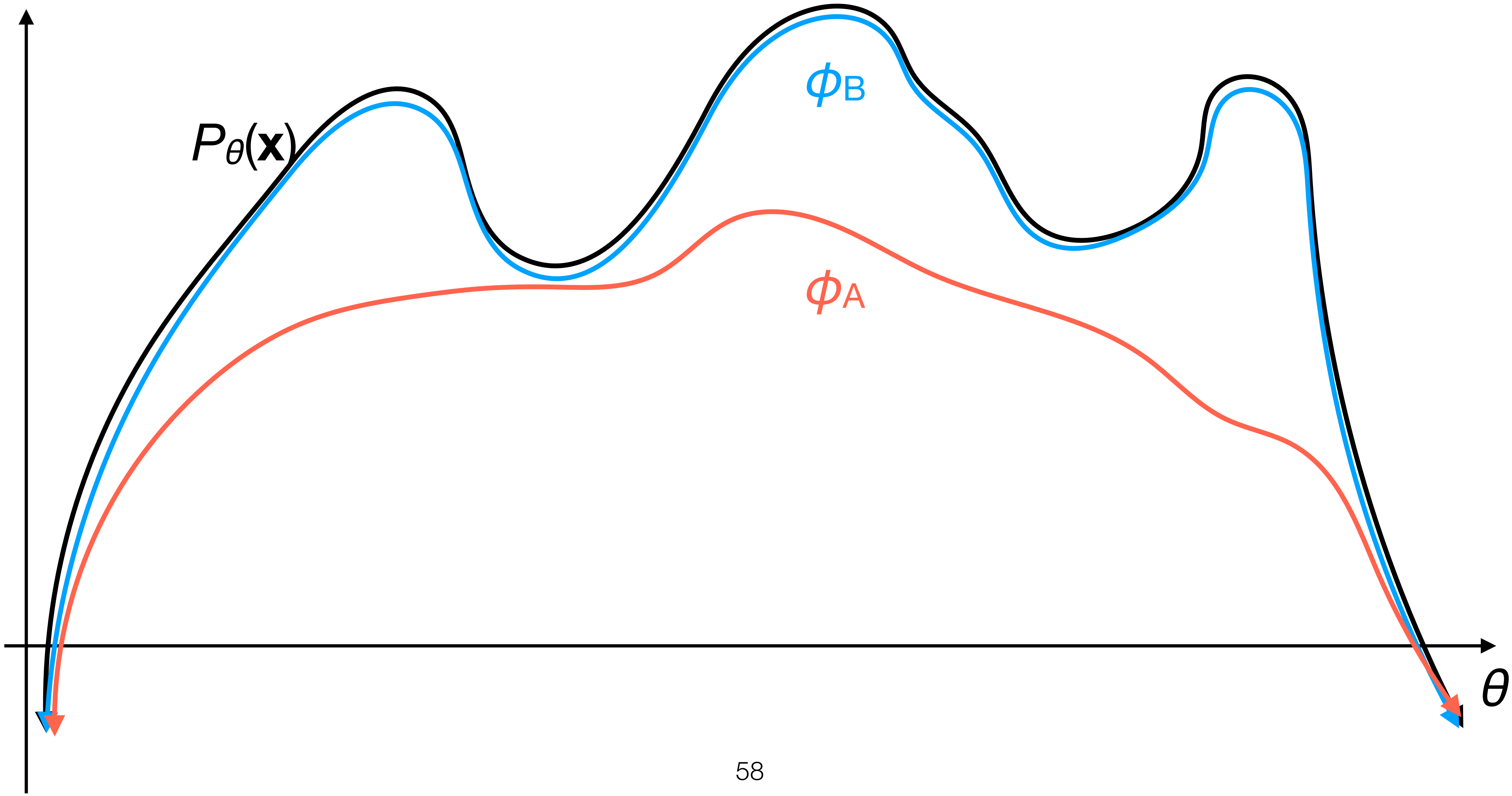
Maximizing the lower bound

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



Exercise

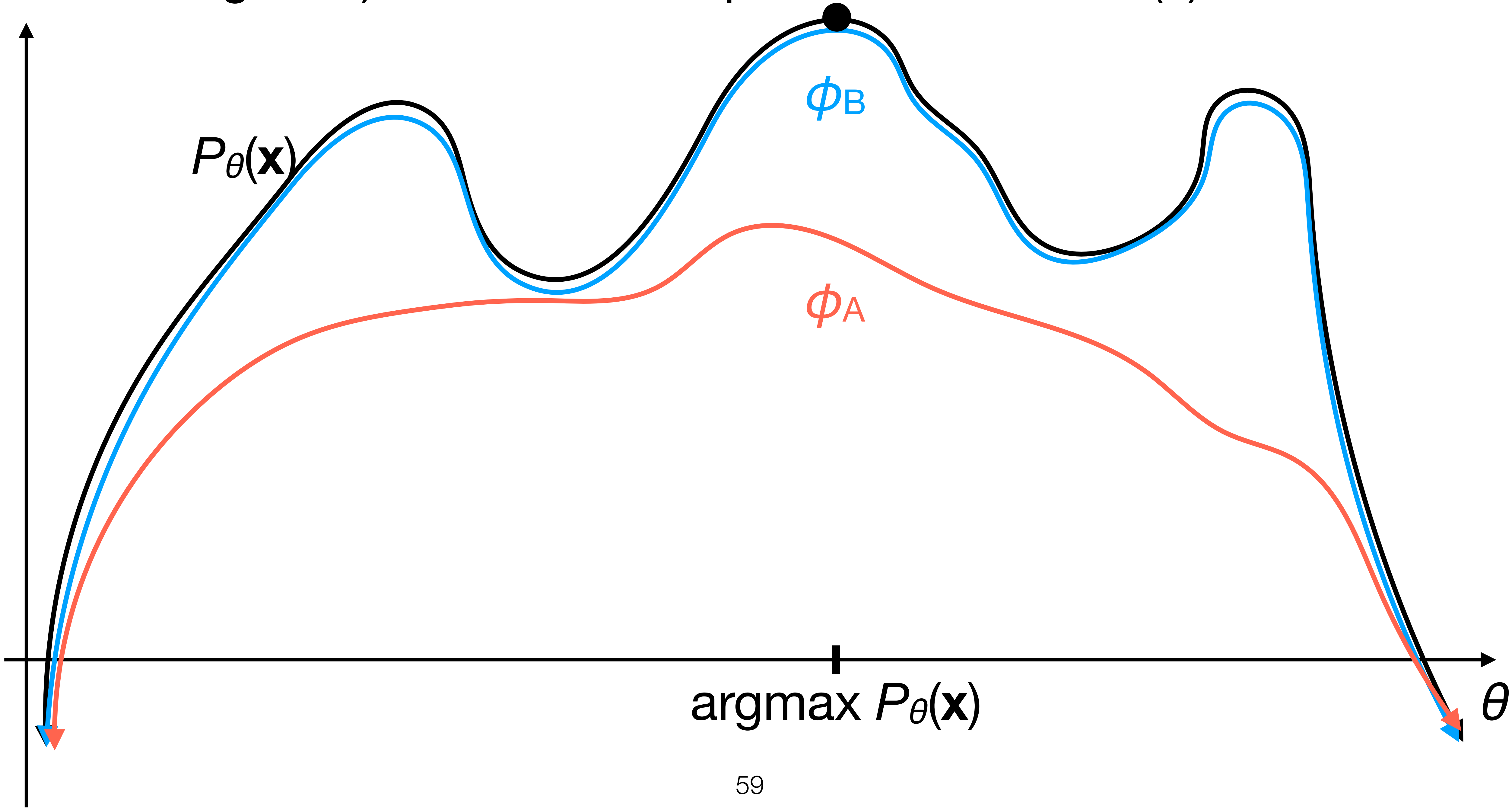
- Which curve (ϕ_A or ϕ_B) would you prefer, and why?



Solution

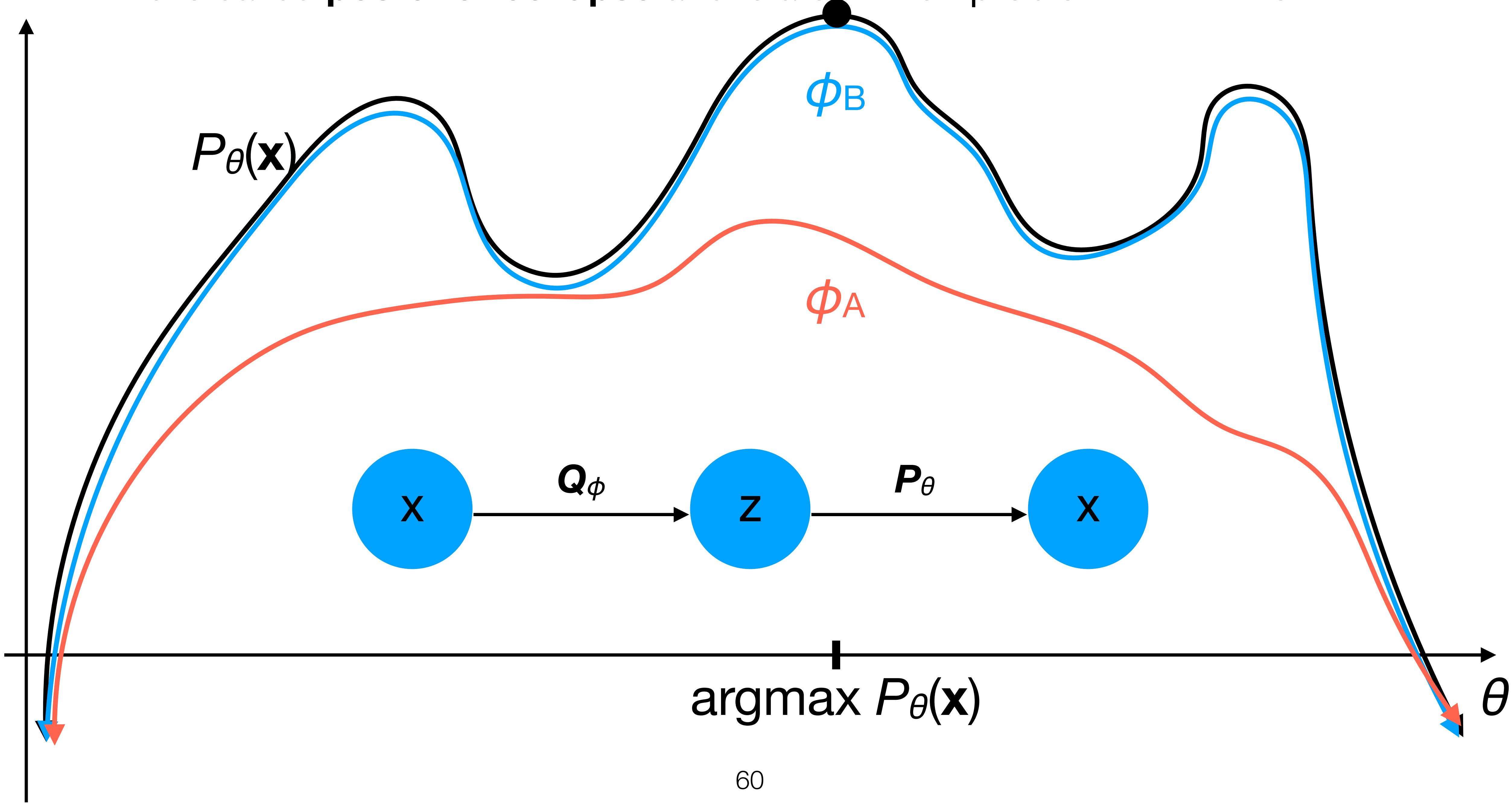
- ϕ_B gives a tighter approximation of $P_\theta(\mathbf{x})$.

- This corresponds to $Q(\mathbf{z} \mid \mathbf{x})$ being more similar (lower KL divergence) to our desired prior distribution $P(\mathbf{z})$.



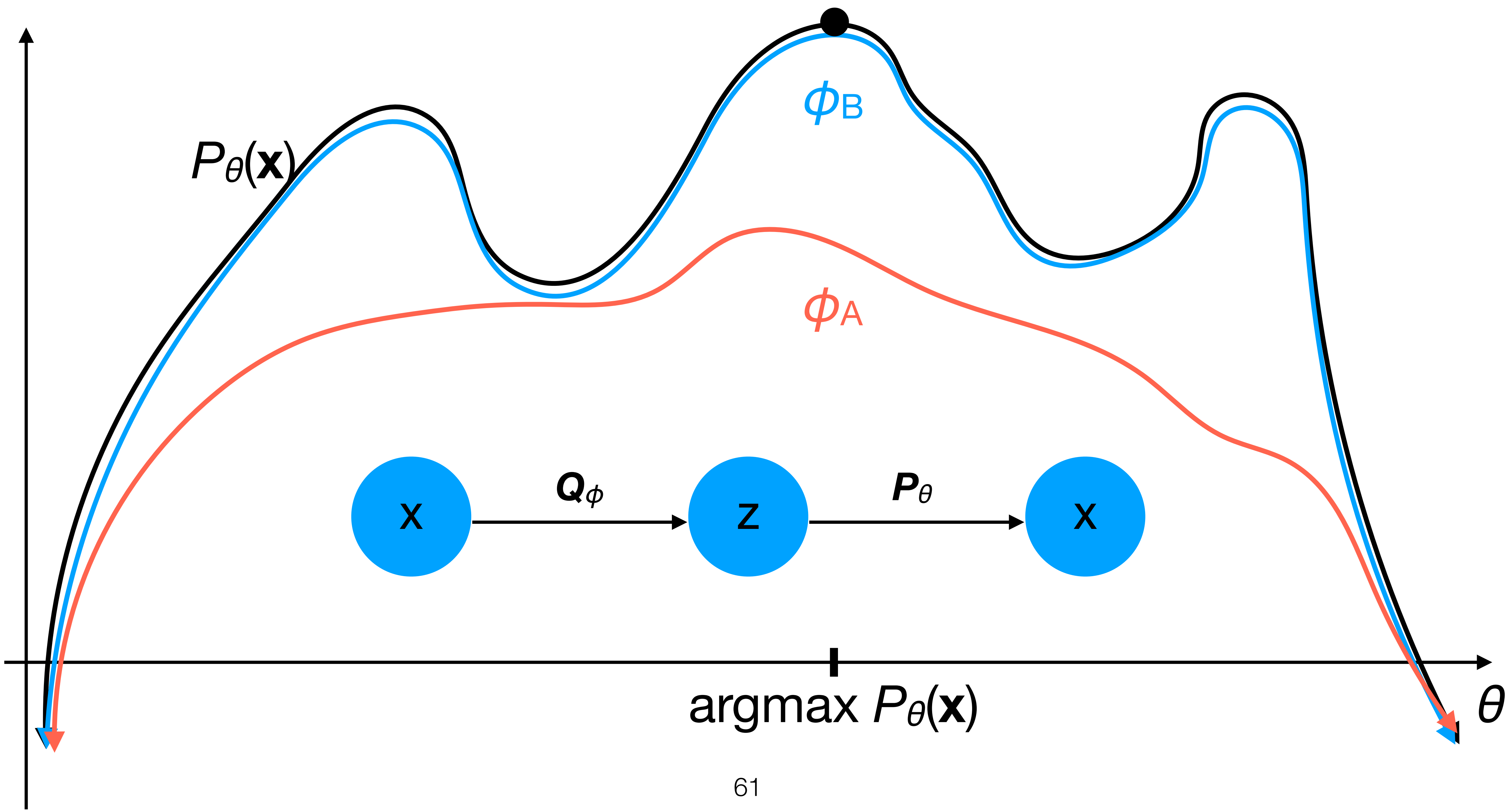
Solution

- **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.



Maximizing the lower bound

- How do we optimize the lower bound w.r.t. θ and ϕ?

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

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

Maximizing the lower bound

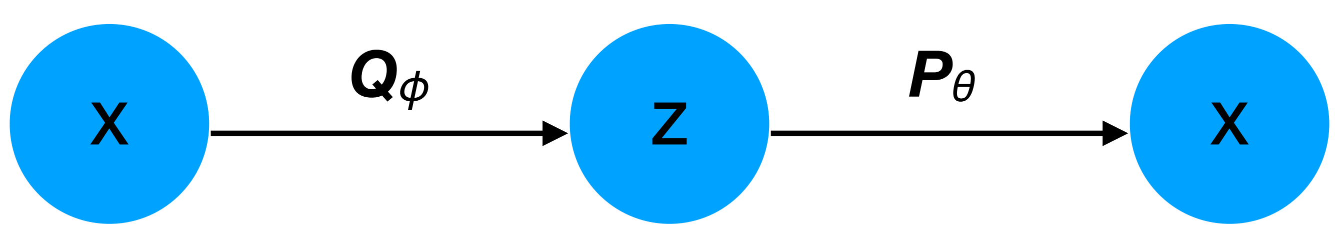
- How do we optimize the lower bound w.r.t. θ and ϕ?

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

- For the second term, we can estimate the expectation by sampling:

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

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



Maximizing the lower bound

- How do we optimize the lower bound w.r.t. θ and ϕ?

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

- But sampling a value from a probability distribution is a **non-differentiable operation** — we can no longer use back-propagation:

$$\underbrace{\mathbf{x}}_{\textbf{Input}} \rightarrow \underbrace{Q_\phi(\mathbf{z} \mid \mathbf{x}) \overset{\textbf{sampling}}{\rightsquigarrow} \mathbf{z}}_{\textbf{Hidden}} \rightarrow \underbrace{P_\theta(\mathbf{x} \mid \mathbf{z})}_{\textbf{Output}}$$

Reparameterization trick

- Suppose $\mathbf{z} \sim P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mu, \sigma^2 \mathbf{I})$
- To sample **z**, we can **either**:
  - Sample from *P*(**z**) directly; **or**
  - Sample from a standard normal, multiply element-wise by σ, and add μ:

$$\mathbf{z}' \sim \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$

$$\mathbf{z} = \mathbf{z}' \odot \sigma + \mu$$

Reparameterization trick

- In the context of the VAE:
    - Instead of sampling $\mathbf{z} \sim Q_\phi(\mathbf{z} \mid \mathbf{x})$ within the computational graph, which would break back-propagation…

sampling

$$\mathbf{x} \xrightarrow[Q_\phi(\mathbf{z}|\mathbf{x})]{} \mathcal{N}(\mathbf{z}; \mu, \sigma^2) \rightsquigarrow \mathbf{z} \rightarrow P_\theta(\mathbf{x} \mid \mathbf{z})$$

Input **Hidden** **Output**

Reparameterization trick

- In the context of the VAE:
  - …we instead sample from outside the graph, multiply the result element-wise by vector σ, and add **μ**.