CS/DS 541: Class 17

Jacob Whitehill

Expectation

- Recall that the expected value of a function *f* w.r.t. a probability distribution $P(\mathbf{z})$ is defined as:

$$\mathbb{E}_P[f(\mathbf{z})] = \int_{\mathbf{z}} f(\mathbf{z})P(\mathbf{z})d\mathbf{z}$$

Expectation

- Note that the probability distribution might be conditioned on a tertiary variable, e.g.:

$$\int_{\mathbf{z}} Q(\mathbf{z} \mid \mathbf{x}) f(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{Q(\mathbf{z} \mid \mathbf{x})}[f(\mathbf{z})]$$

Sampling from a Gaussian

- Suppose $z \sim P(z) = \mathcal{N}(z; \mu, \sigma^2)$
- To sample *z*, we can **either**:
  - Sample from *P*(*z*) directly (Python: `scipy.random.normal(loc=mu, scale=sigma)`).
  - Sample from a standard normal, multiply by σ, and add μ

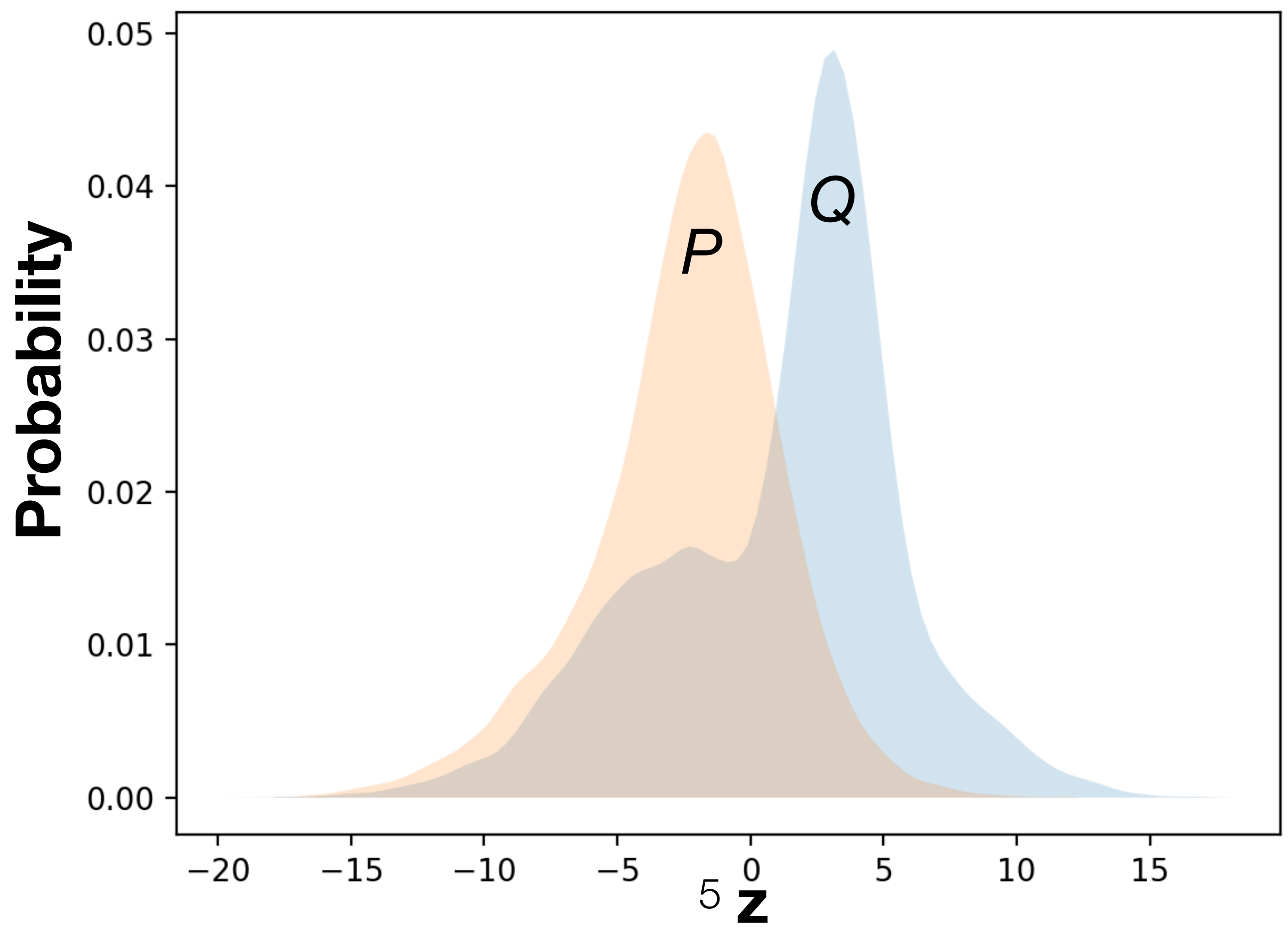
$$z' \sim \mathcal{N}(z'; 0, 1)$$
$$z = \sigma z' + \mu$$

(Python: `sigma*scipy.random.normal(0,1) + mu`).

Kullback-Leibler Divergence

- The Kullback-Leibler (KL) divergence quantifies the distance of *Q* from *P* as the log difference in probabilities at each **z** weighted by the probability of **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

- We can also write the KL divergence as:

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

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).

Jensen’s inequality

- For convex *f*, Jensen’s inequality implies:

$$tf(x_1) + (1-t)f(x_2) \geq f(tx_1 + (1-t)x_2)$$

$$\frac{\sum_{i=1}^{n} a_i f(x_i)}{\sum_{i=1}^{n} a_i} \geq f\left(\frac{\sum_{i=1}^{n} a_i x_i}{\sum_{i=1}^{n} a_i}\right)$$

$$\sum_i \frac{1}{n} f(x_i) \geq f\left(\sum_i \frac{1}{n} x_i\right)$$

$$\frac{1}{n}\sum_i f(x_i) \geq f\left(\frac{1}{n}\sum_i x_i\right)$$

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}[x])$$

$$\int_x f(x)P(x)dx \geq f\left(\int_x xP(x)dx\right)$$

Note: this is not a derivation! It is just a list of generalizations of the inequality.

Jensen’s inequality

- Consider *f*(*x*)=log(*x*).
- *f* is concave (opposite of convex) because its second derivative is negative everywhere in its domain.
- Therefore, when we apply Jensen’s inequality we reverse the sign, i.e.:

$$\int_x f(x)P(x)dx \geq f\left(\int_x xP(x)dx\right) \implies$$

$$\log \int_x xP(x)dx \geq \int_x \log(x)P(x)dx$$

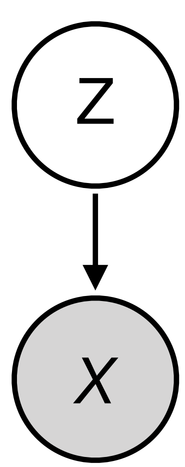
Here, we can “pull” the log into the integral.

Variational Auto-encoders (VAE)

Latent variable models

- Recall how we constructed a shallow latent variable model (LVM) that was purely linear:

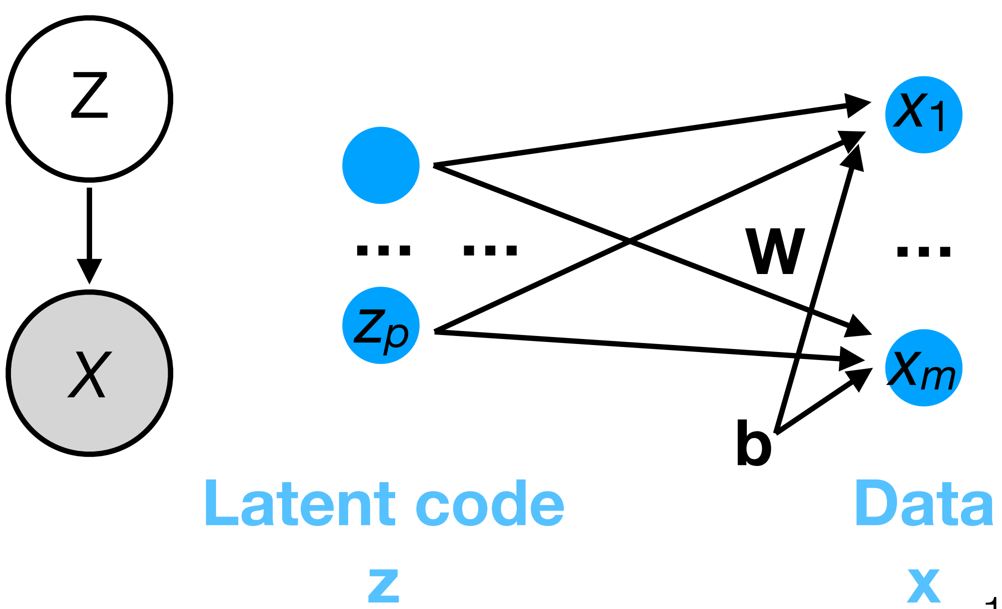
- We assumed that every data point $\mathbf{x} \in \mathbb{R}^m$ is generated from a low-dimensional latent variable (or **code**) $\mathbf{z} \in \mathbb{R}^p$, where $p < m$.



Latent variable models

- In particular, we assumed that each **x** is approximately linear in **h**, i.e.:

$$\mathbf{x} \approx \mathbf{W}\mathbf{z} + \mathbf{b}$$



Latent variable models

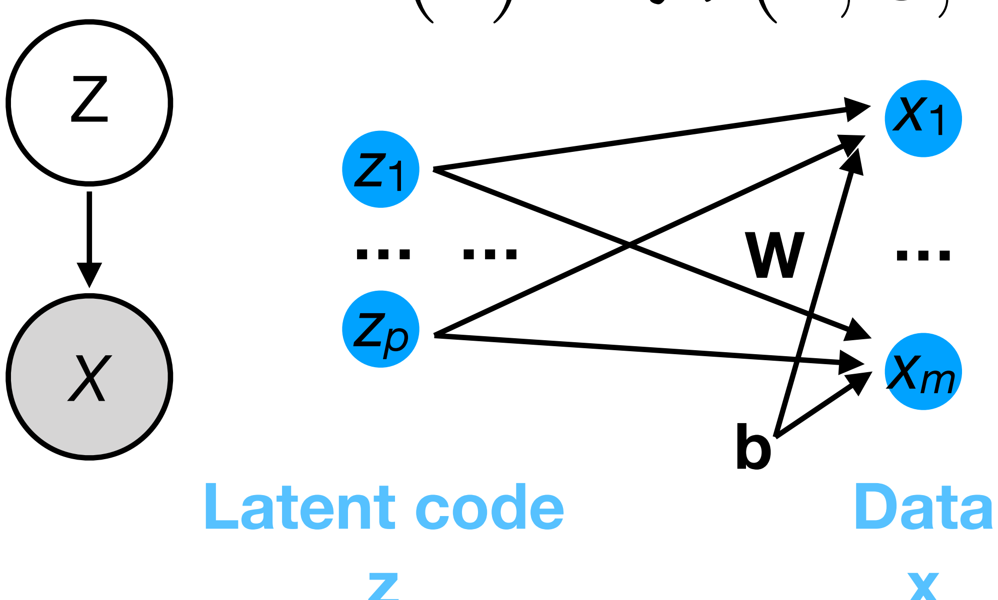
- In particular, we assumed that each **x** is approximately linear in **h**, i.e.:

$$\mathbf{x} \approx \mathbf{W}\mathbf{z} + \mathbf{b}$$

- We represented this probabilistically as:

$$P_\theta(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z} + \mathbf{b}, \sigma^2 \mathbf{I}) \quad \text{where } \theta = (\mathbf{W}, \mathbf{b}, \sigma^2$$

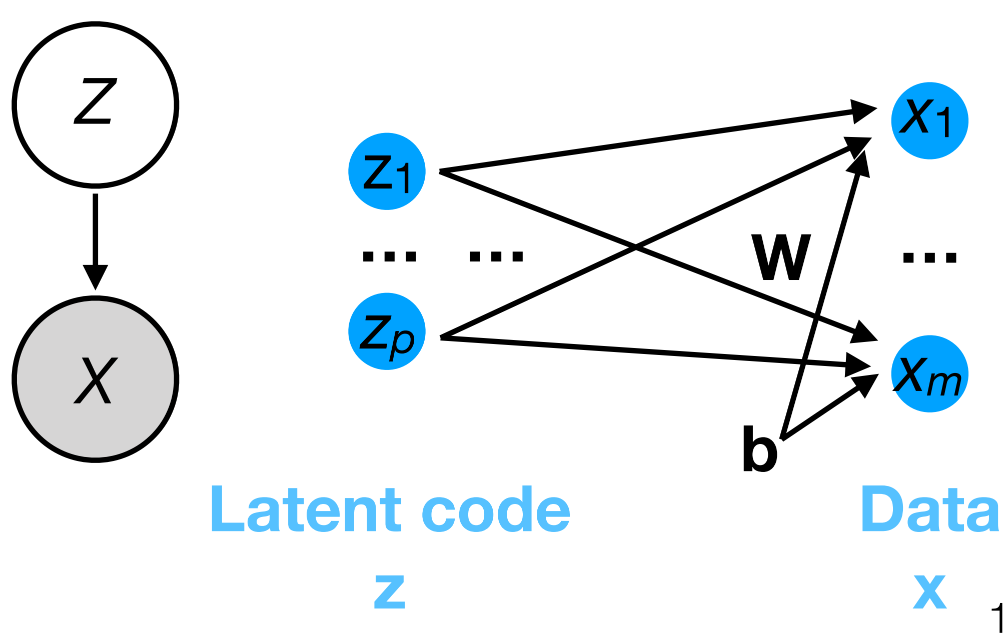
$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$



Latent variable models

- Given a training dataset $\mathcal{D}$, we can use MLE to maximize the log-likelihood of each training example $\mathbf{x}$ — i.e., $\log P(\mathbf{x})$ — w.r.t. the model parameters θ.

- For this linear 2-layer model, there is a closed-form solution.

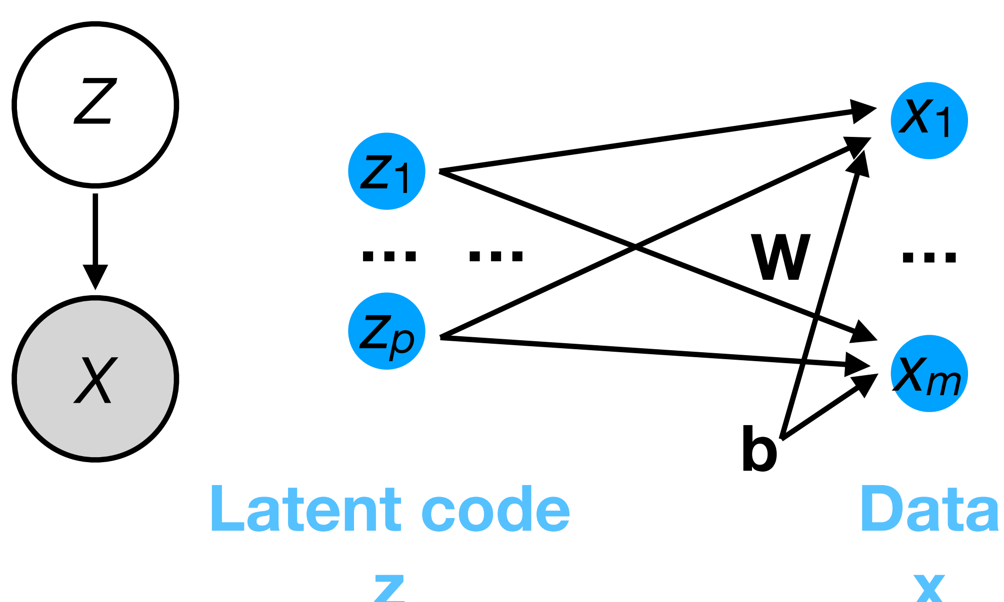


Latent variable models

- This LVM allows us to **generate** novel examples:

  1. Sample from the prior distribution over **z**.

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$



Latent variable models

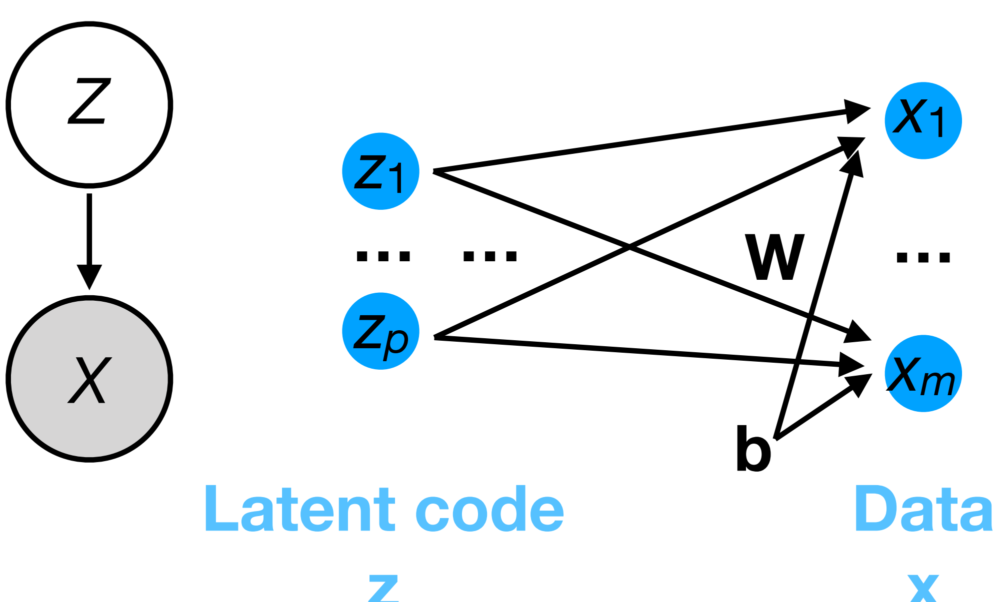
- This LVM allows us to **generate** novel examples:

1. Sample from the prior distribution over **z**.

$$P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$$

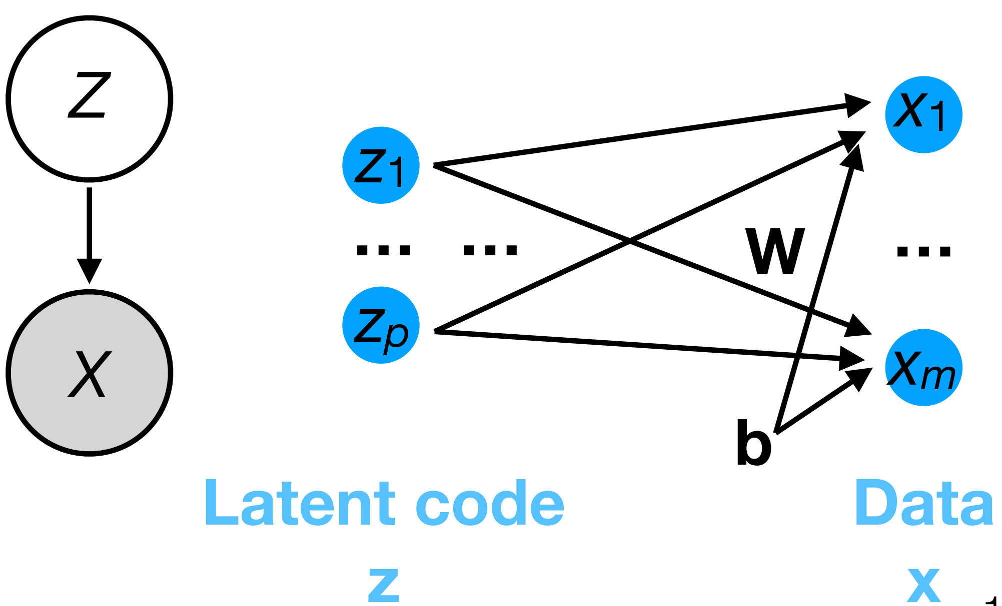
2. Sample from the conditional distribution of **x** | **z**.

$$P_\theta(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; \mathbf{W}\mathbf{z} + \mathbf{b}, \sigma^2\mathbf{I})$$



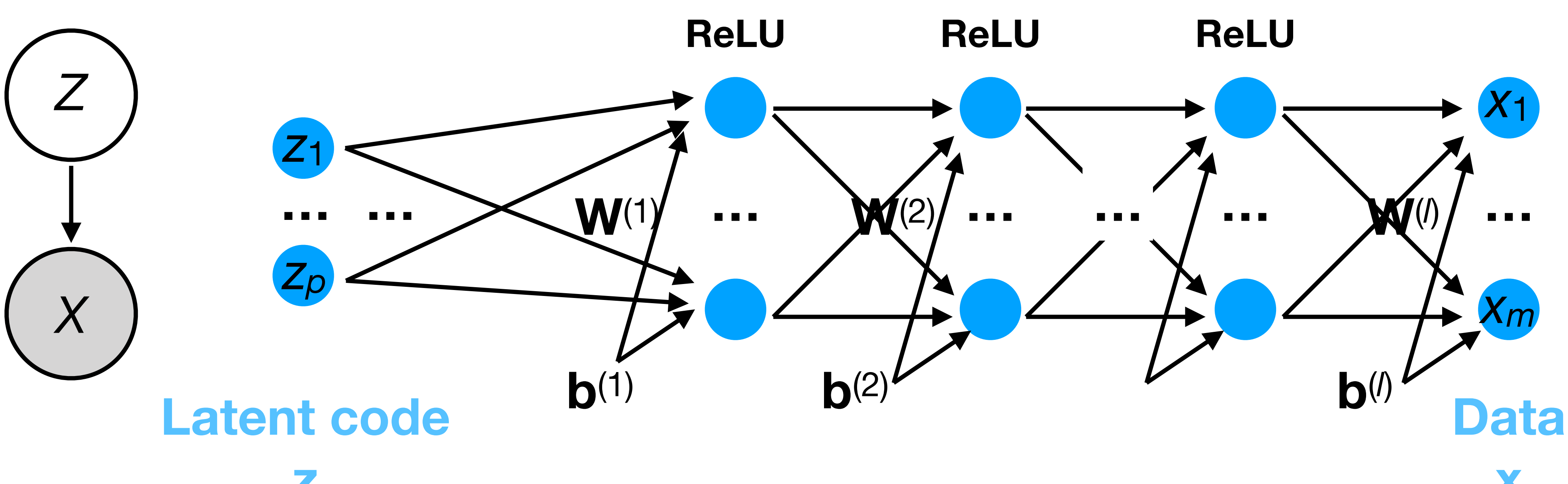
Latent variable models

- While easy to optimize, this shallow LVM is weak since it is strictly linear.
- The generated data are often not representative of the true distribution $P(\mathbf{x})$, e.g.:



Latent variable models

- With DL, we can construct and train deeper and more powerful LVMs that are non-linear.
- The generated data can be much more realistic, e.g.:

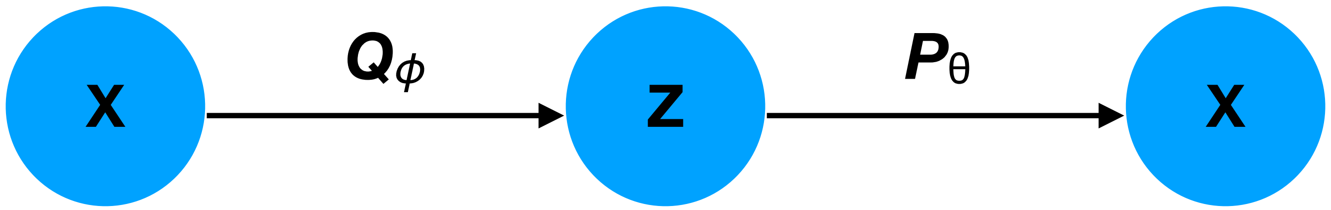


Variational auto-encoder

- The variational auto-encoder (VAE; Kingma & Welling 2014) is a deep probabilistic LVM.
- It is one of the two chief DL techniques to generate data (the other is generative adversarial networks).

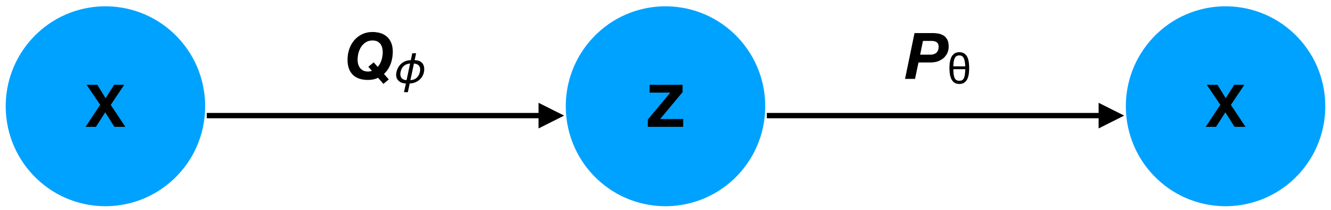
VAE architecture

- The VAE consists of an encoder Q_ϕ and decoder P_θ.
  - $Q(\mathbf{z} \mid \mathbf{x})$ outputs a probability distribution over **Z** given **X**.
  - $P(\mathbf{x} \mid \mathbf{z})$ outputs a probability distribution over **X** given **Z**.



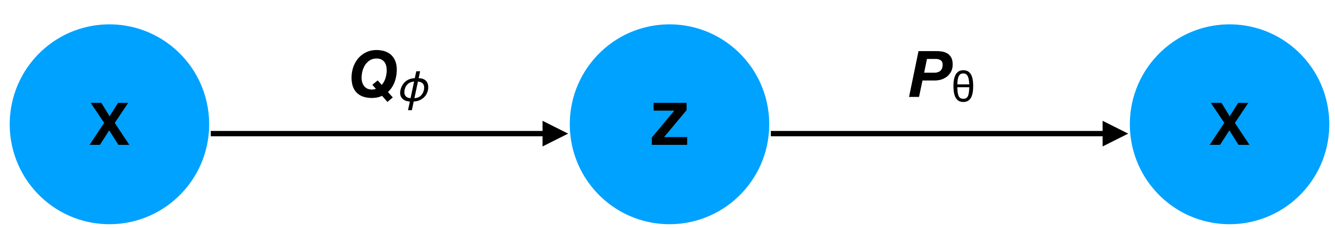
VAE architecture

- The VAE consists of an encoder Q_ϕ and decoder P_θ.
  - $Q(\mathbf{z} \mid \mathbf{x})$ outputs a probability distribution over **Z** given **X**.
  - $P(\mathbf{x} \mid \mathbf{z})$ outputs a probability distribution over **X** given **Z**.
- We fix the probability distribution of the hidden state to be $P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$; this makes it easy to generate new data.



VAE architecture

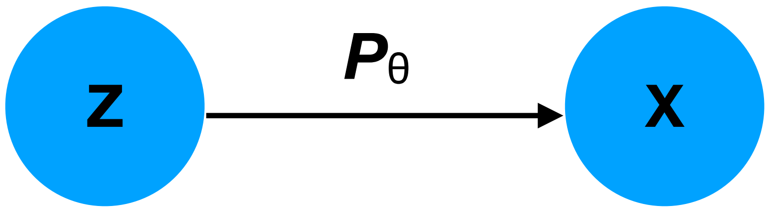
- Once trained, the VAE can be used for two purposes:
  - **Density estimation**: estimate how likely a given **x** is to occur, e.g., for anomaly detection. Requires only *Q*.
  - **Generation**: create novel data. Requires only *P*.



VAE architecture

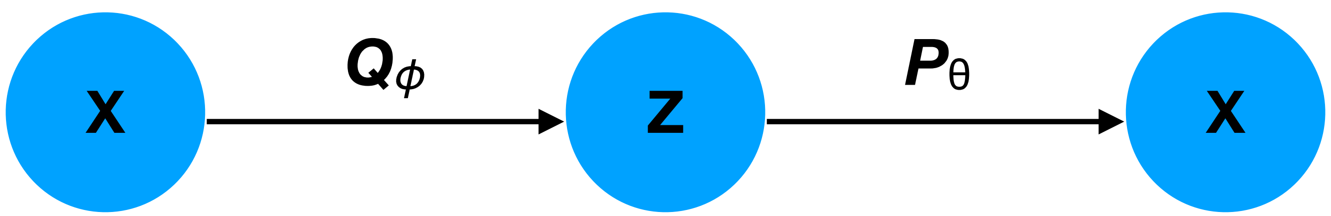
- Here is how we can generate data:

  1. Sample $\mathbf{z} \sim P(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$
  2. Compute $P_\theta(\mathbf{x} \mid \mathbf{z})$
  3. Sample $\mathbf{x} \sim P_\theta(\mathbf{x} \mid \mathbf{z})$



VAE architecture

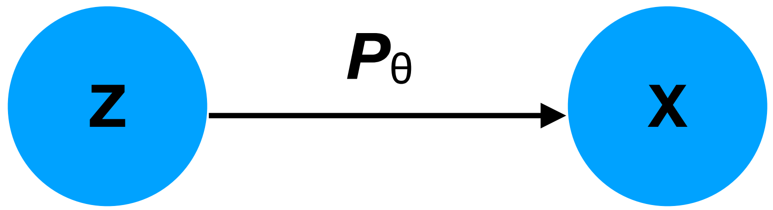
- The parameters ϕ and θ are trained using maximum-likelihood estimation (MLE).
- We aim to maximize the likelihood of our observed training data, given *P*'s parameters θ, i.e.: $P_\theta(\{\mathbf{x}^{(i)}\}_{i=1}^n)$
- Using a variational approximation technique, we will also optimize Q's parameters ϕ along the way.



VAE: MLE derivation

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

Law of total probability

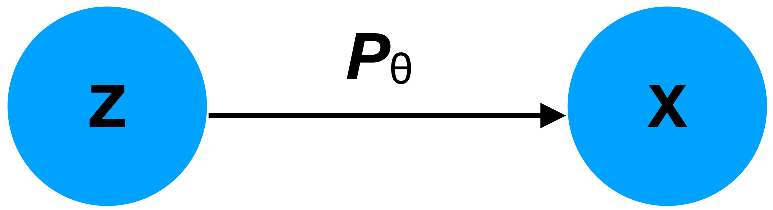


VAE: MLE derivation

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

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

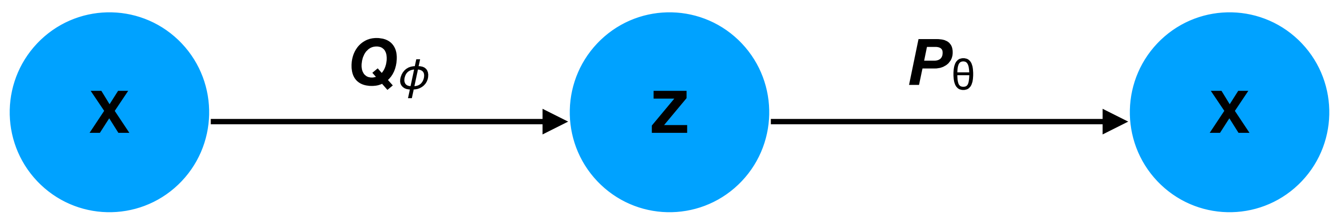
Definition of conditional probability



VAE: MLE derivation

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

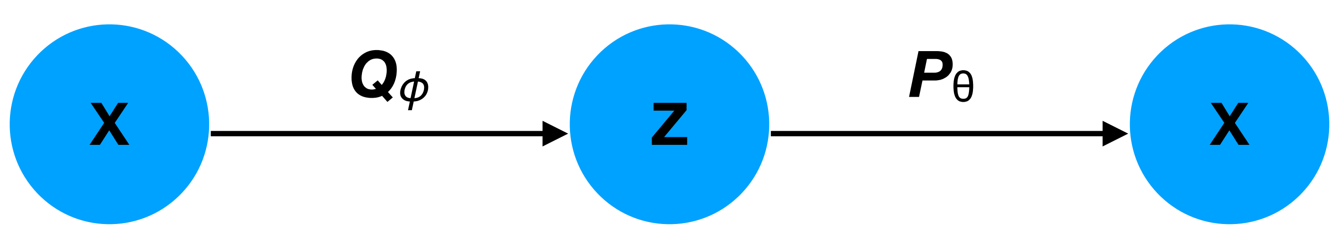
This holds for any non-zero *Q*.



VAE: MLE derivation

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

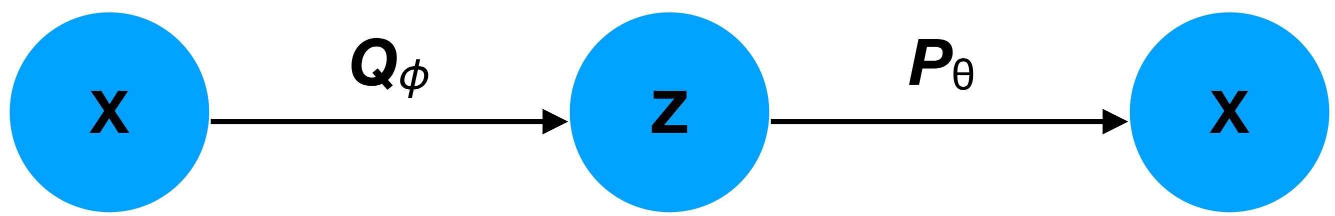
Note also that we can interpret this function as the expectation w.r.t. Q(z | x).



VAE: MLE derivation

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

Jensen's inequality



VAE: MLE derivation

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

Split the logarithm.

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.

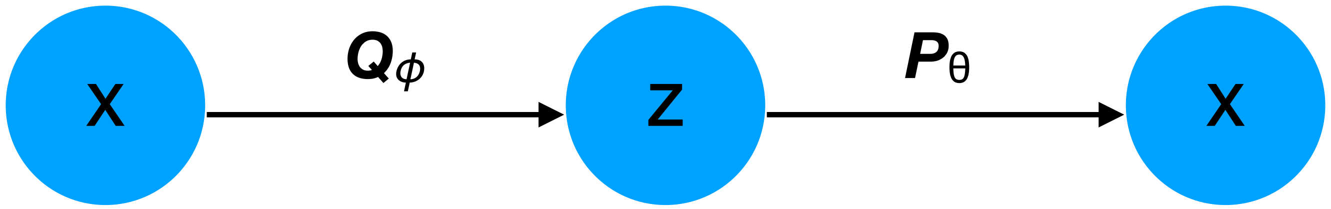
VAE: MLE derivation

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

VAE: MLE derivation

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

    and
  - Maximize the reconstruction probability.



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

Maximizing the lower bound

- How do we optimize this 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 previous lecture).

Maximizing the lower bound

- How do we optimize this 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 second term is more problematic — we can try to 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 this 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:

sampling

$$\mathbf{x} \to Q_\phi(\mathbf{z} \mid \mathbf{x}) \rightsquigarrow \mathbf{z} \to P_\theta(\mathbf{x} \mid \mathbf{z})$$

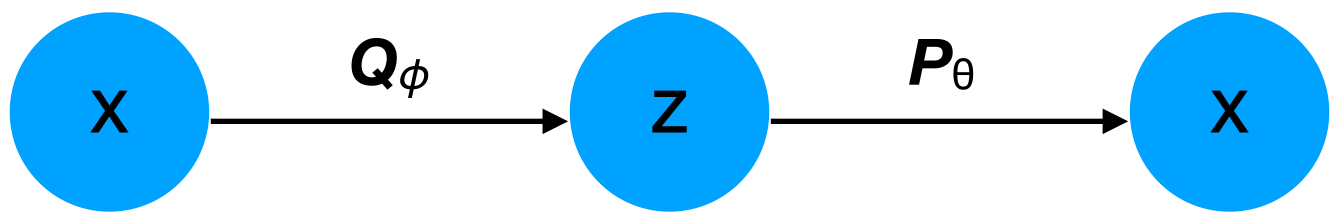
Input **Hidden** **Output**

Training details

VAE architecture

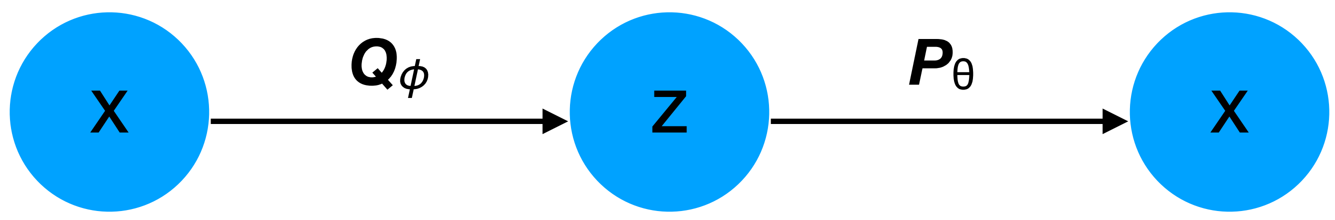
- To generate examples $\mathbf{x} \in [0,1]^m$ (like in Homework 6), $P_\theta(\mathbf{x} \mid \mathbf{z})$ can end with *m* logistic sigmoid functions.

- It can then be trained with the log-likelihood objective summed over all *m* outputs.



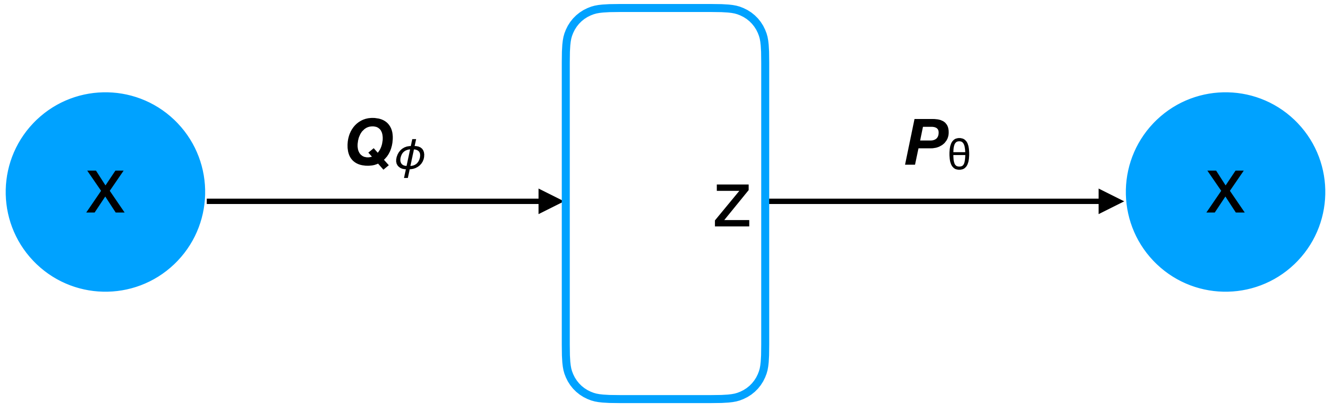
VAE architecture

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



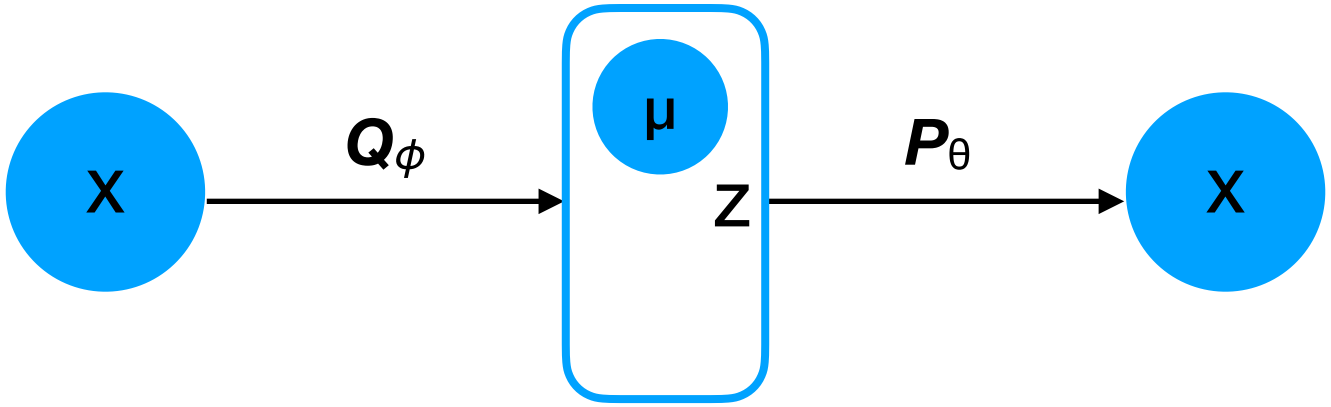
VAE architecture

- How do we force Q_ϕ to output a Gaussian distribution?
  - Given **x**, Q_ϕ needs to output:



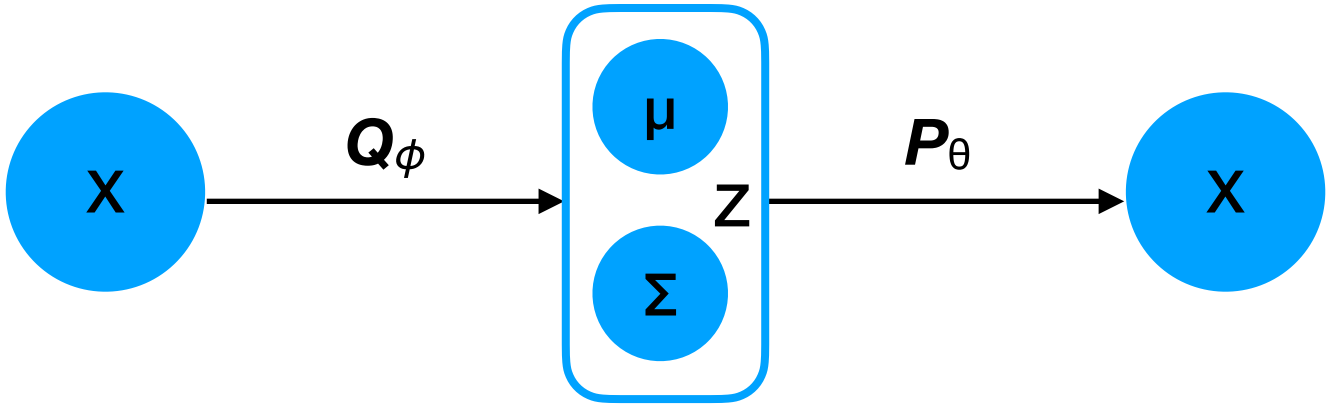
VAE architecture

- How do we force Q_ϕ to output a Gaussian distribution?
  - Given **x**, Q_ϕ needs to output:
    - Mean **μ**



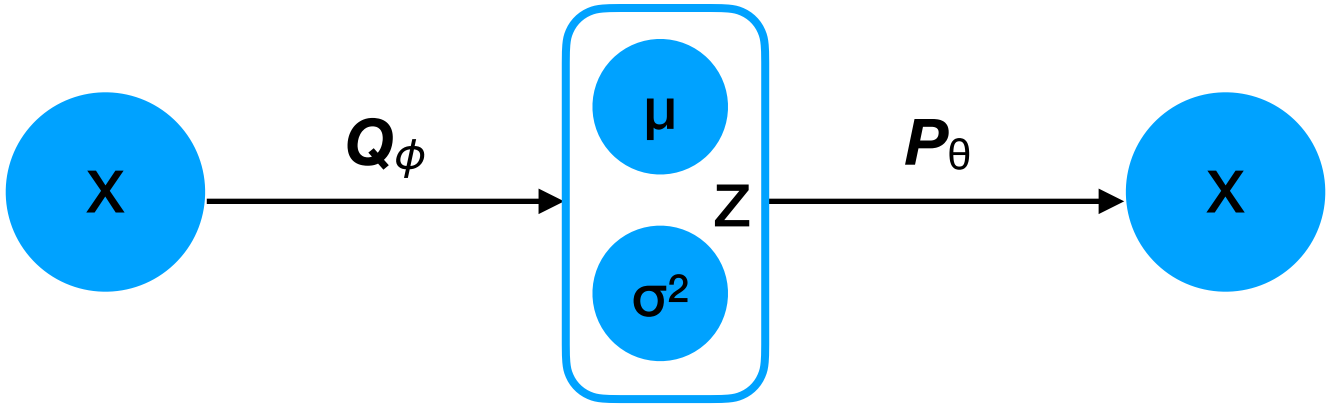
VAE architecture

- How do we force Q_ϕ to output a Gaussian distribution?
  - Given **x**, Q_ϕ needs to output:
    - Mean $\boldsymbol{\mu}$
    - Covariance matrix $\boldsymbol{\Sigma}$



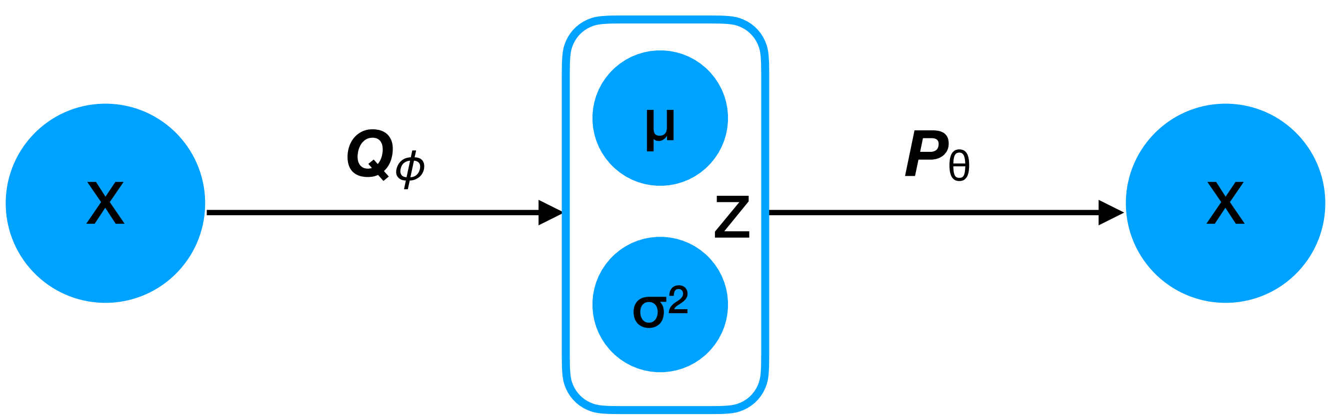
VAE architecture

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



VAE architecture

- As a simplification, Q_ϕ can output a diagonal covariance matrix parameterized by just a vector [σ_1^2, …, σ_p^2].
- All in all, Q_ϕ outputs $2p$ entries, where the first p specify the mean and the second p specify the covariance.

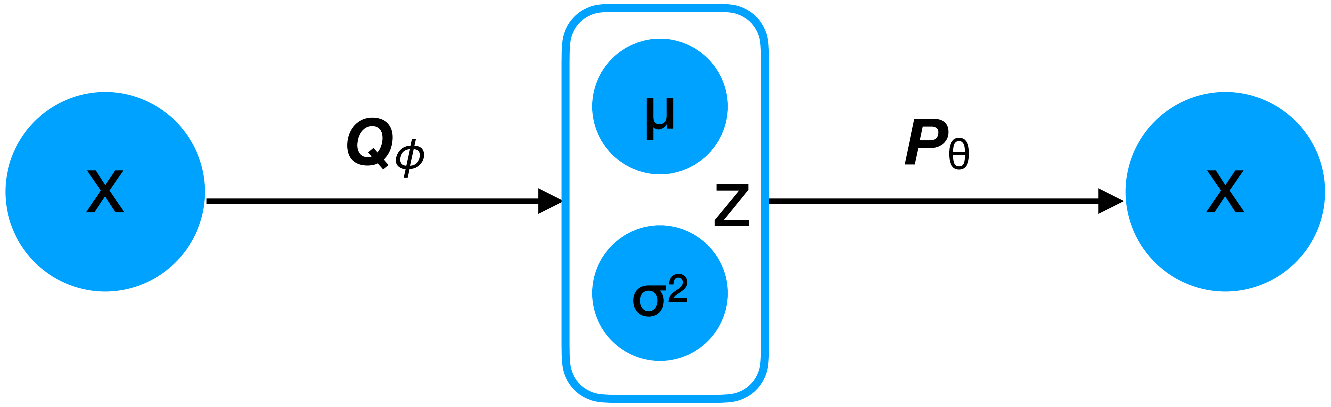


VAE architecture

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

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

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



Maximizing the lower bound

- How do we optimize this 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:

sampling

$$\boxed{\mathbf{x}} \to \boxed{Q_\phi(\mathbf{z} \mid \mathbf{x}) \rightsquigarrow \mathbf{z}} \to \boxed{P_\theta(\mathbf{x} \mid \mathbf{z})}$$

Input **Hidden** **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…

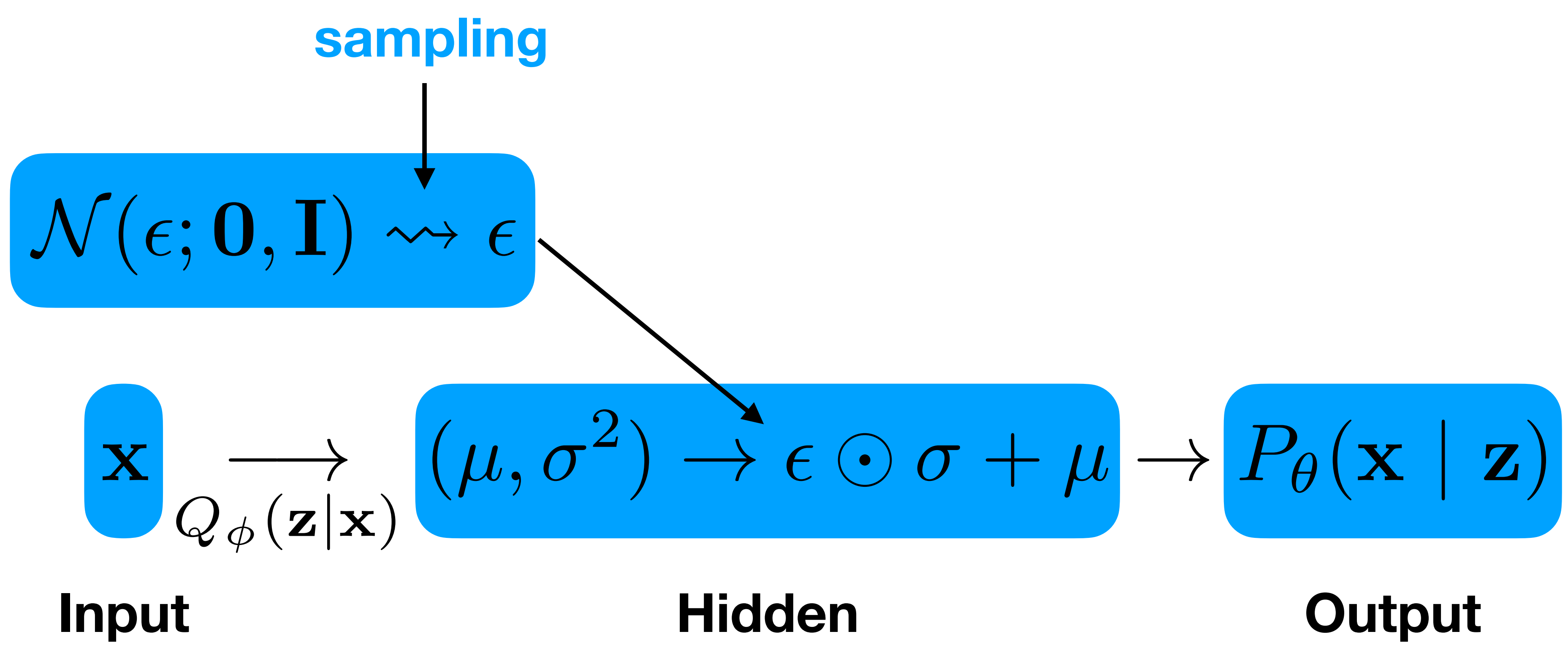
sampling

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

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**.



Training procedure

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

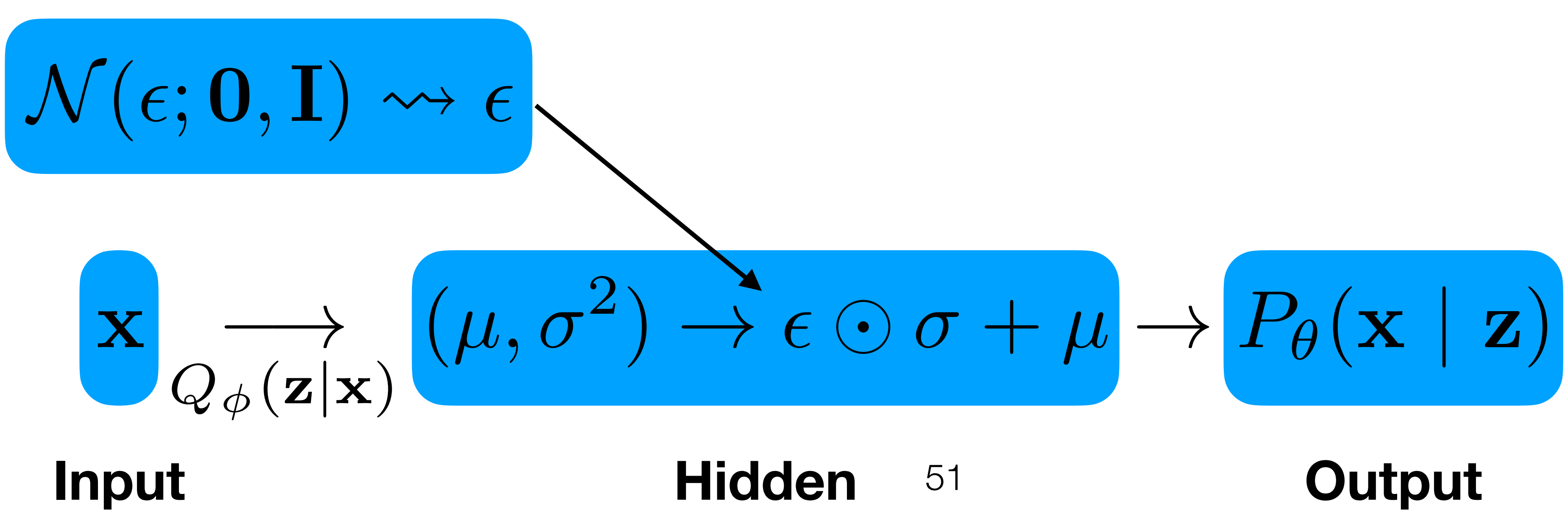
$$\mathcal{N}(\epsilon; \mathbf{0}, \mathbf{I}) \rightsquigarrow \epsilon$$

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

Input **Hidden** **Output**

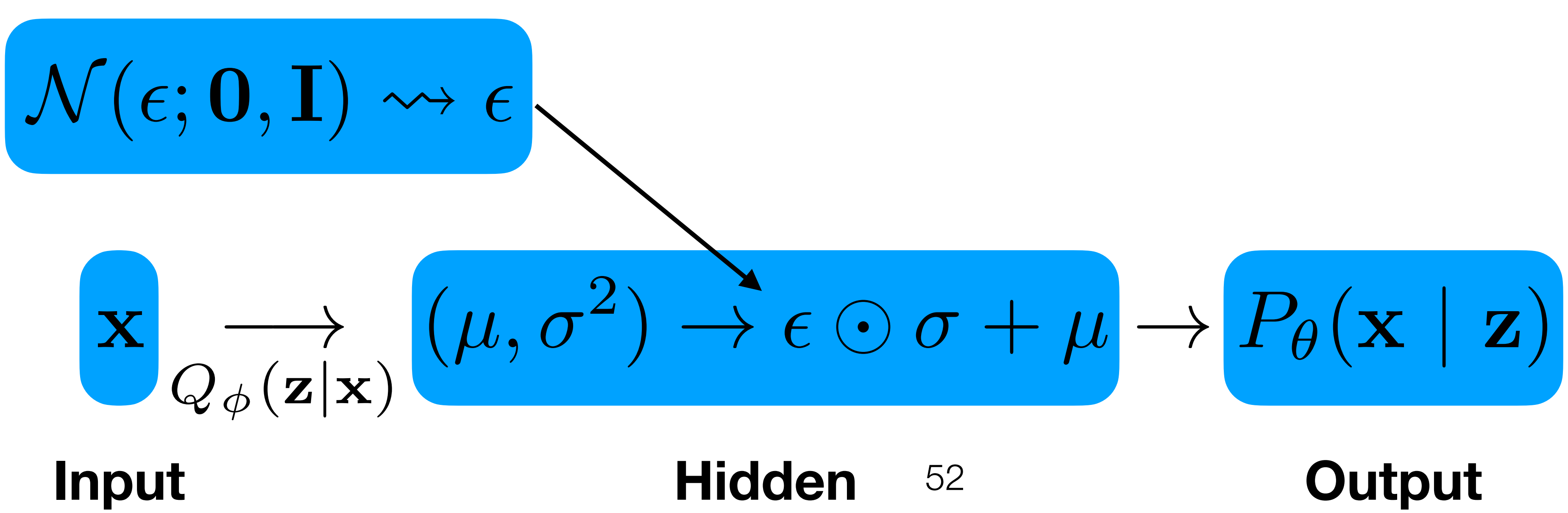
Training procedure

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



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

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

Note: this is an approximation of the expectation with just 1 sample.

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: $\{\epsilon^{(i)}\}_{i=1}^{\tilde{n}} \subset \mathbb{R}^p$
  - 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, …]` // $P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$

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

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.:

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, …]` // $P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$

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, …]` // $P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$

Log-like.

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

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, …]` // $P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$

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.:
    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, …]` // $P_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)})$
    …

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

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.:

  - In practice, you can use a binary cross-entropy loss in either TensorFlow or PyTorch.

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

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

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