



香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

COMP 5212

Machine Learning

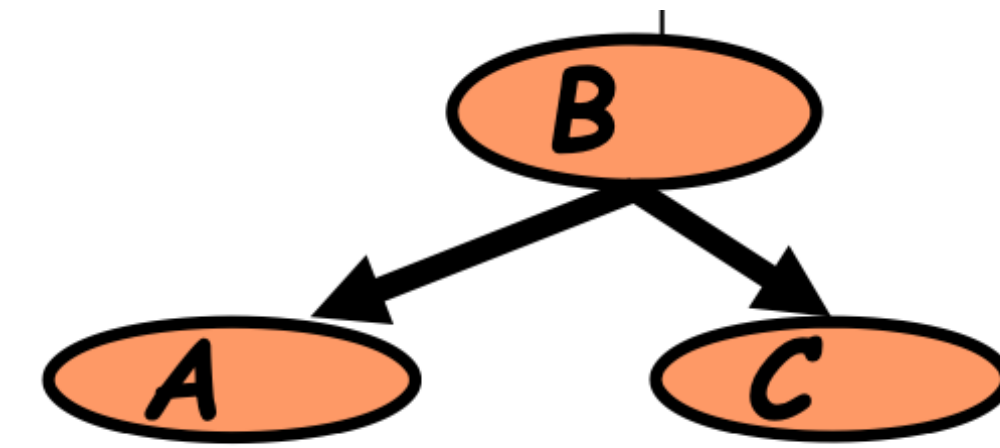
Lecture 13

Variational AutoEncoder

Junxian He
Mar 26, 2026

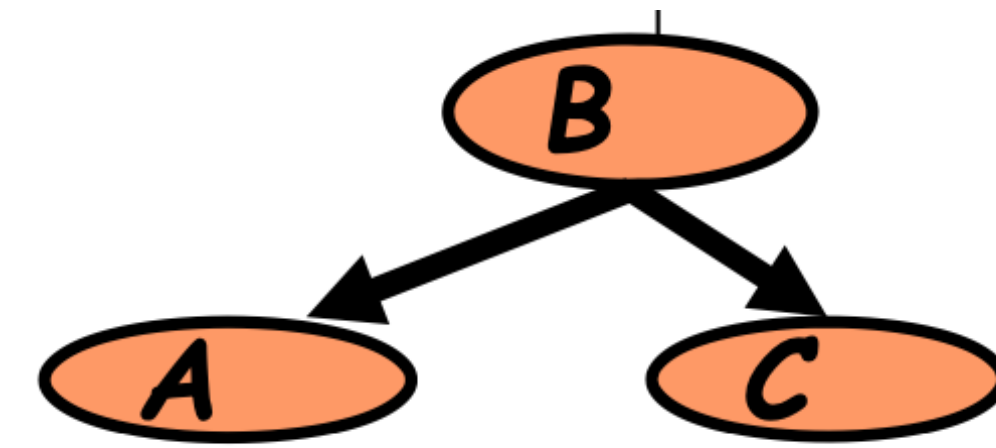
Recap: Local Structures & Independence

Recap: Local Structures & Independence



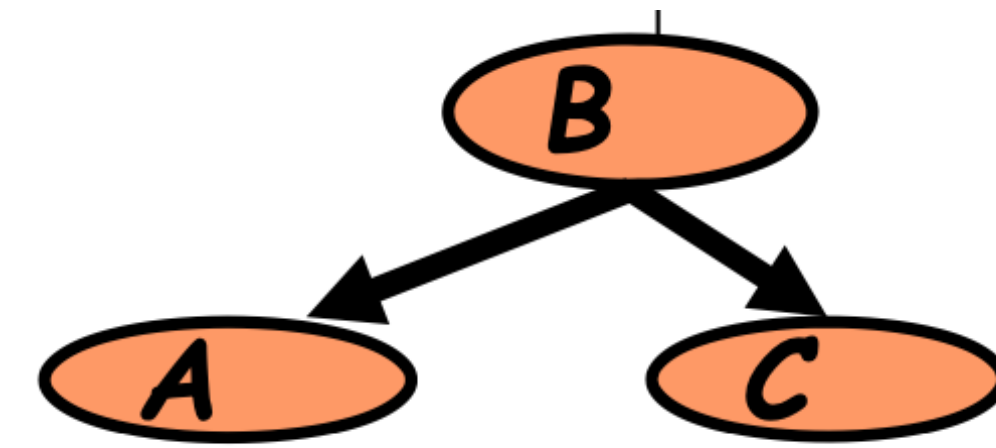
Recap: Local Structures & Independence

- Common parent
 - Fixing B decouples A and C
 - "given the level of gene B, the levels of A and C are independent"



Recap: Local Structures & Independence

- Common parent
 - Fixing B decouples A and C
"given the level of gene B, the levels of A and C are independent"

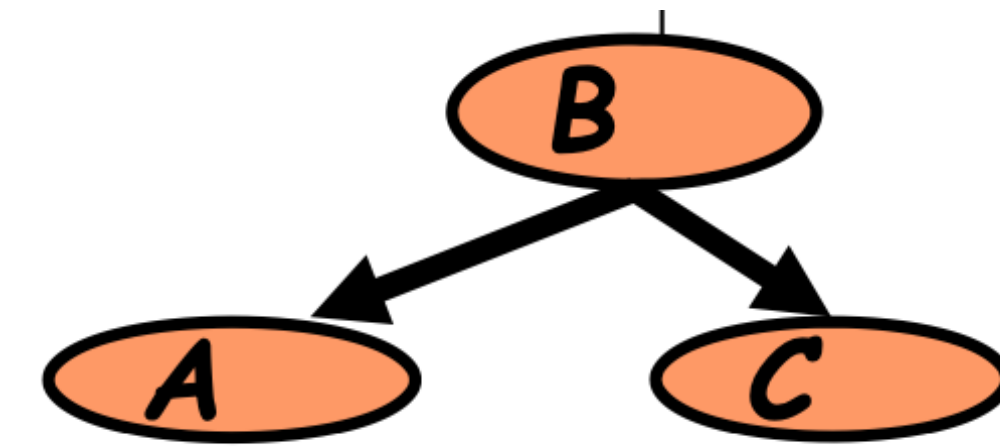


Recap: Local Structures & Independence

- Common parent

- Fixing B decouples A and C

"given the level of gene B, the levels of A and C are independent"



- Cascade

- Knowing B decouples A and C

"given the level of gene B, the level gene A provides no extra prediction value for the level of gene C"

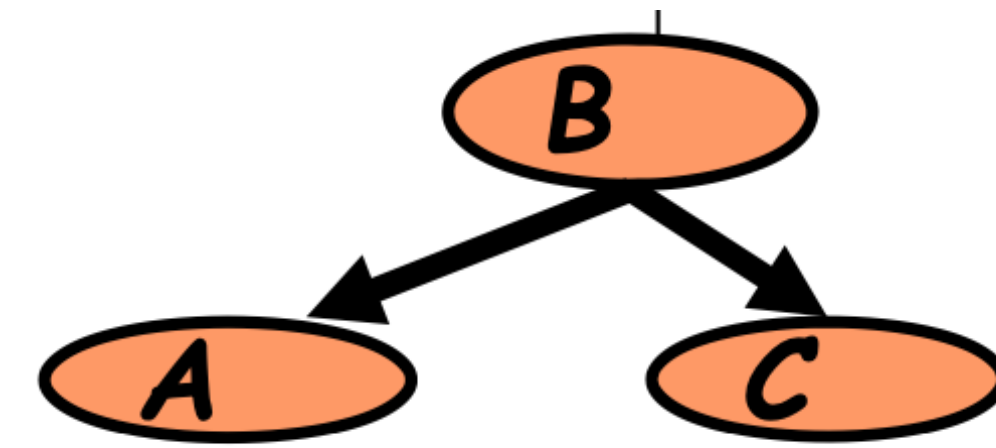


Recap: Local Structures & Independence

- Common parent

- Fixing B decouples A and C

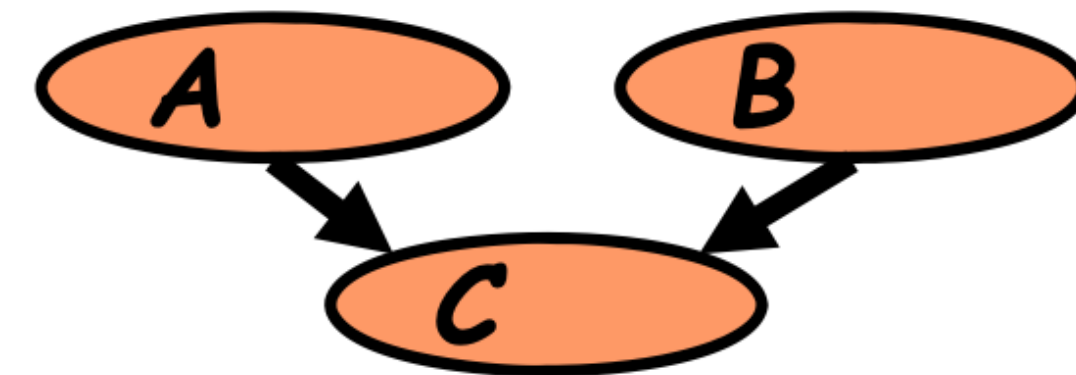
"given the level of gene B, the levels of A and C are independent"



- Cascade

- Knowing B decouples A and C

"given the level of gene B, the level gene A provides no extra prediction value for the level of gene C"

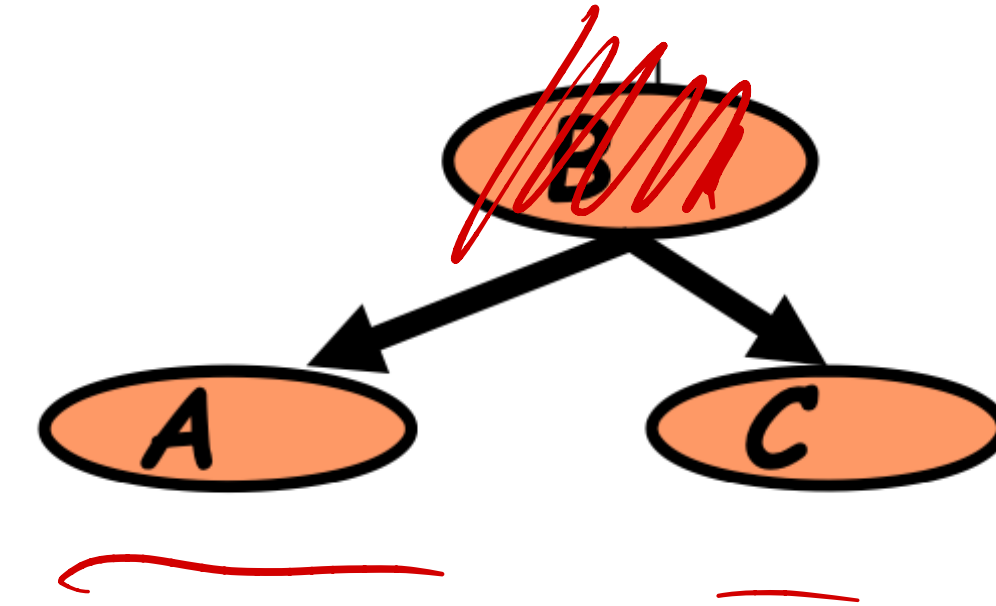


Recap: Local Structures & Independence

- Common parent

- Fixing B decouples A and C

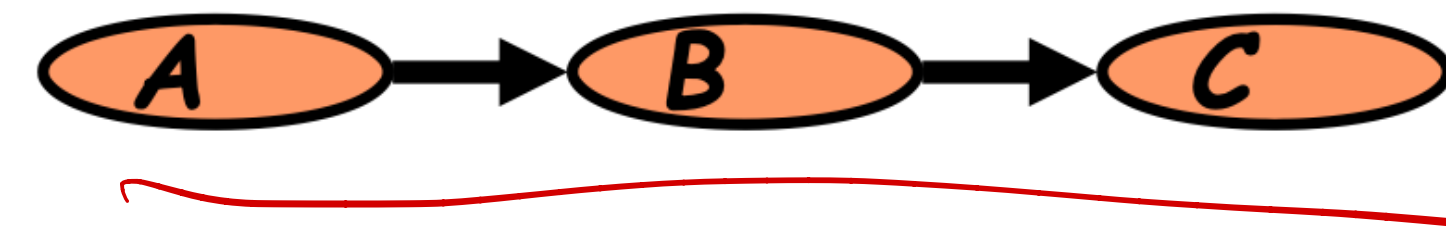
"given the level of gene B, the levels of A and C are independent"



- Cascade

- Knowing B decouples A and C

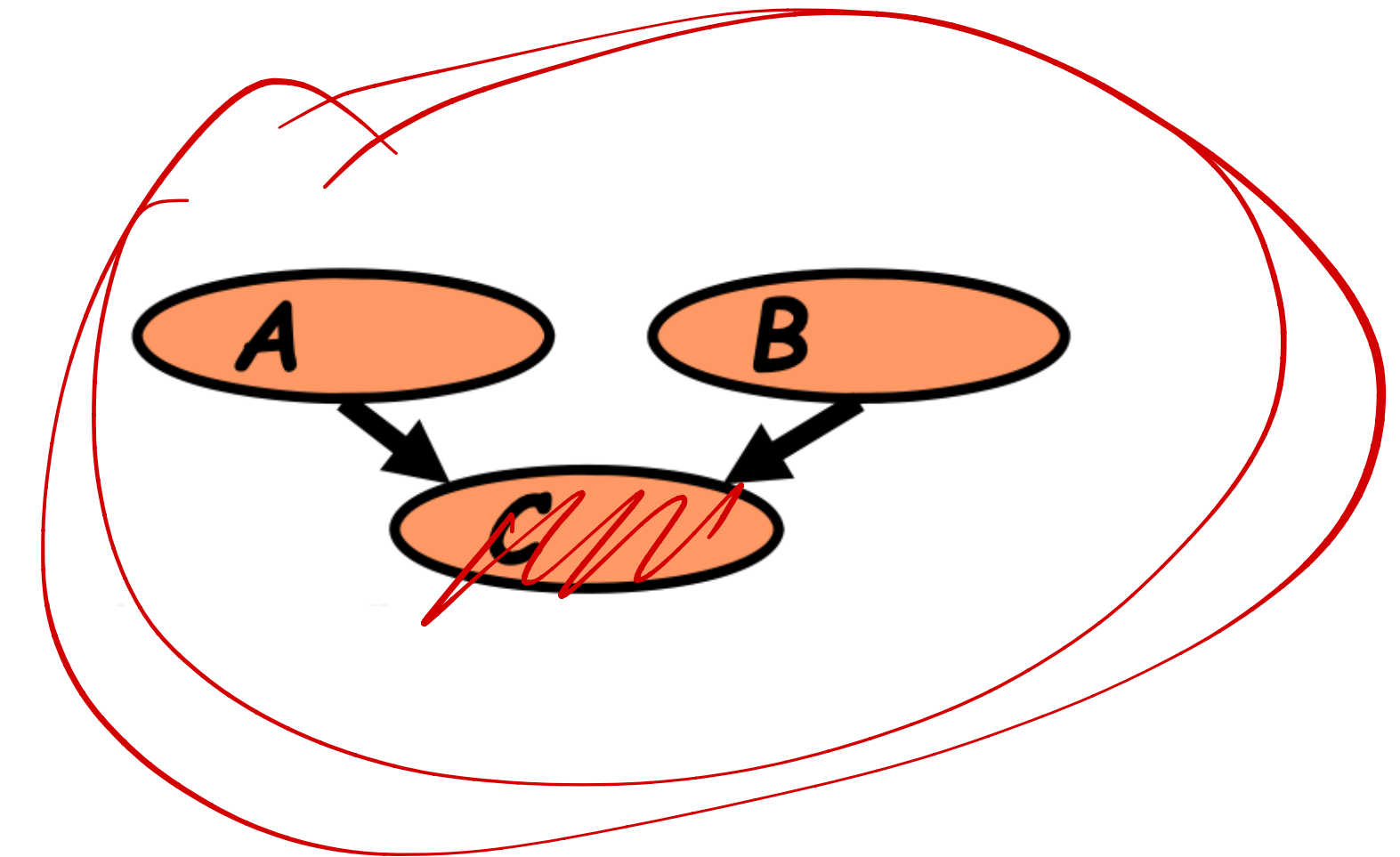
"given the level of gene B, the level gene A provides no extra prediction value for the level of gene C"



- V-structure

- Knowing C couples A and B because A can "explain away" B w.r.t. C

"If A correlates to C, then chance for B to also correlate to B will decrease"

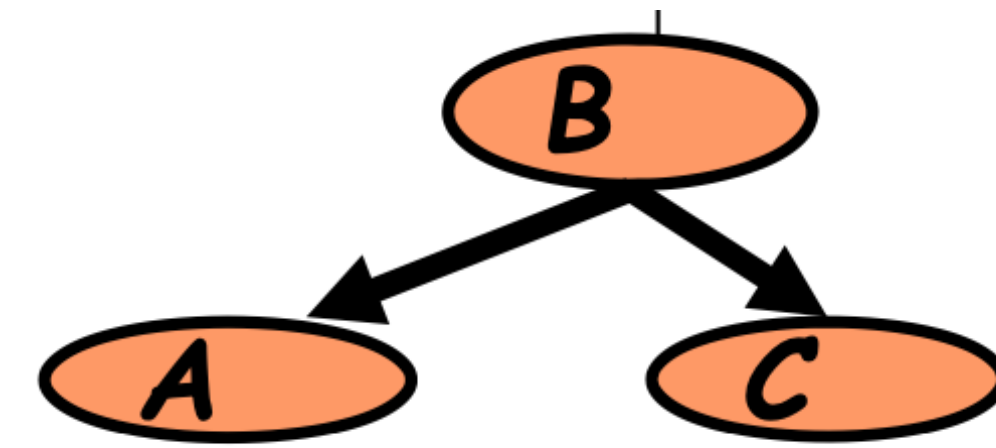


Recap: Local Structures & Independence

- Common parent

- Fixing B decouples A and C

"given the level of gene B, the levels of A and C are independent"



- Cascade

- Knowing B decouples A and C

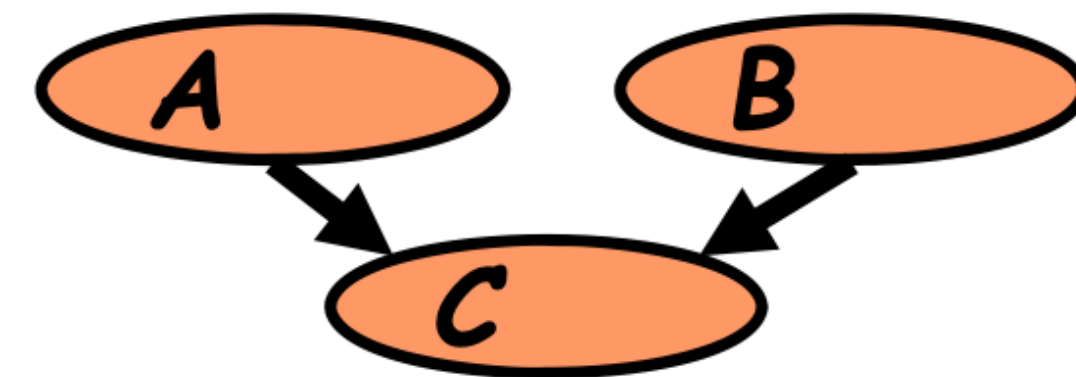
"given the level of gene B, the level gene A provides no extra prediction value for the level of gene C"



- V-structure

- Knowing C couples A and B because A can "explain away" B w.r.t. C

"If A correlates to C, then chance for B to also correlate to B will decrease"



The language is compact, the concepts are rich!

Recap: Global Markov Properties of DAGs

Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

Recap: Global Markov Properties of DAGs

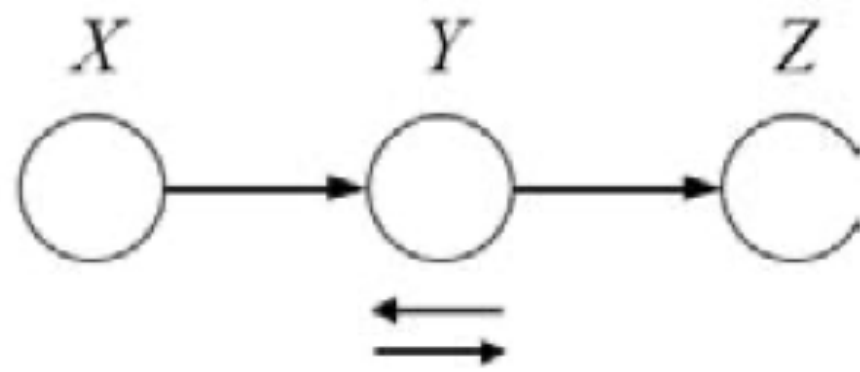
How to determine two variables are conditionally independent given another variable?

X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "*Bayes-ball*" algorithm illustrated below (and plus some boundary conditions):

Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

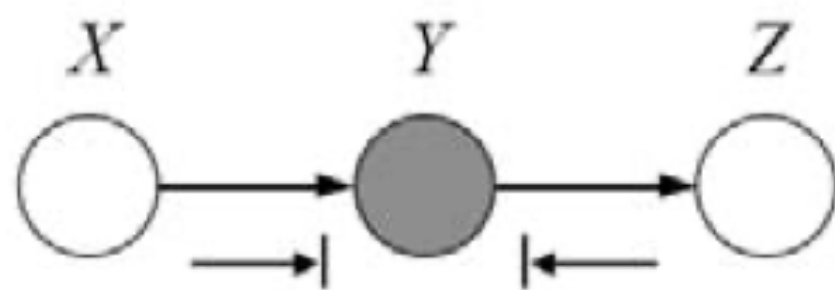
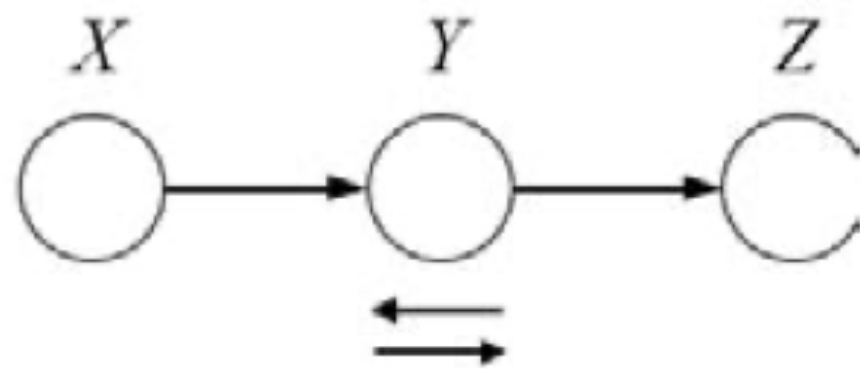
X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

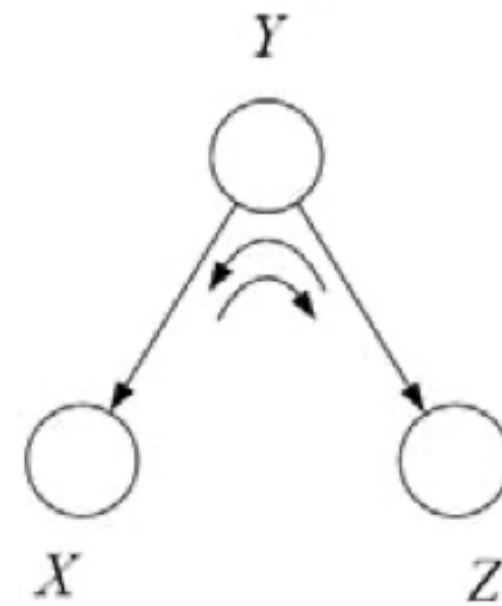
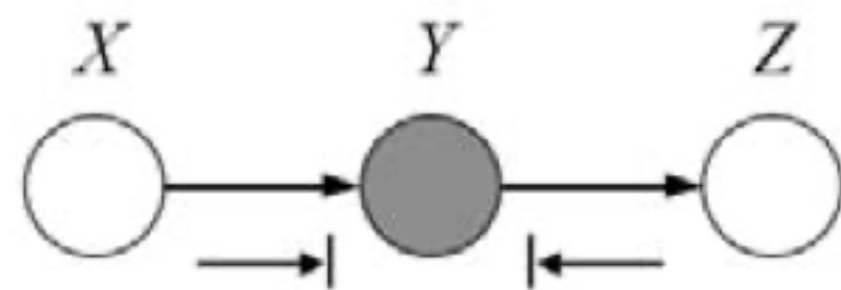
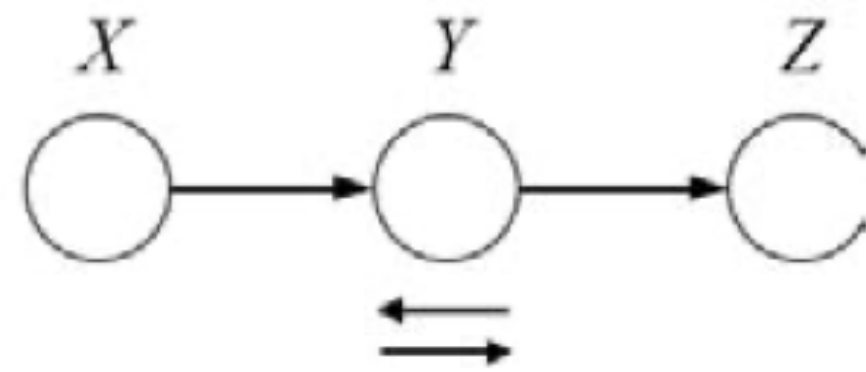
X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

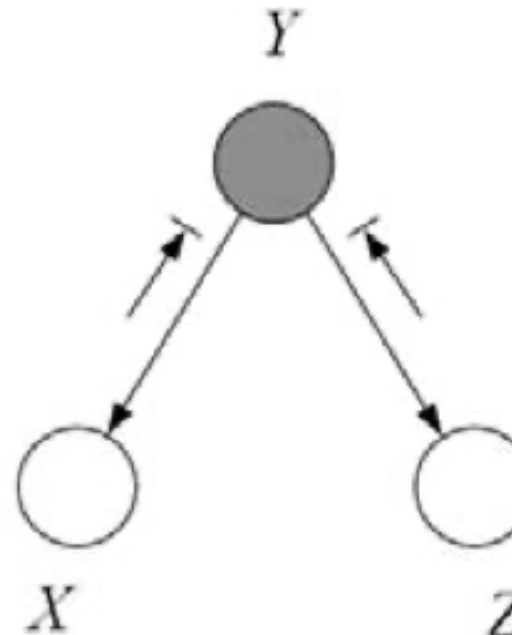
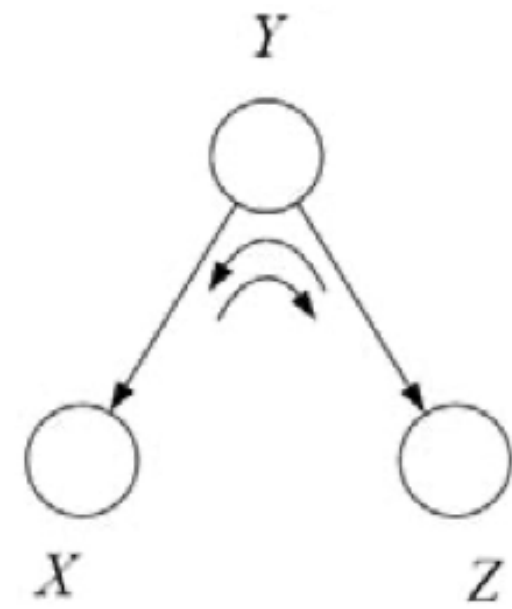
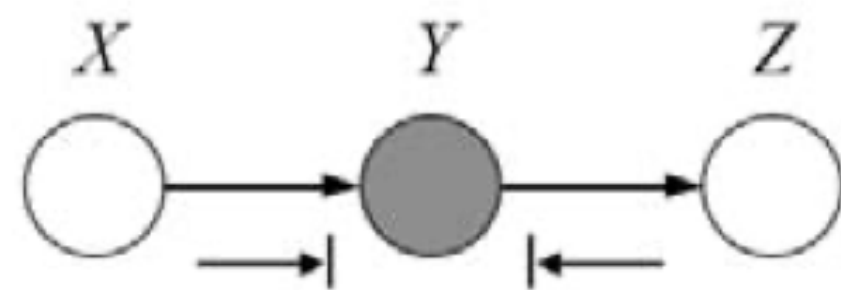
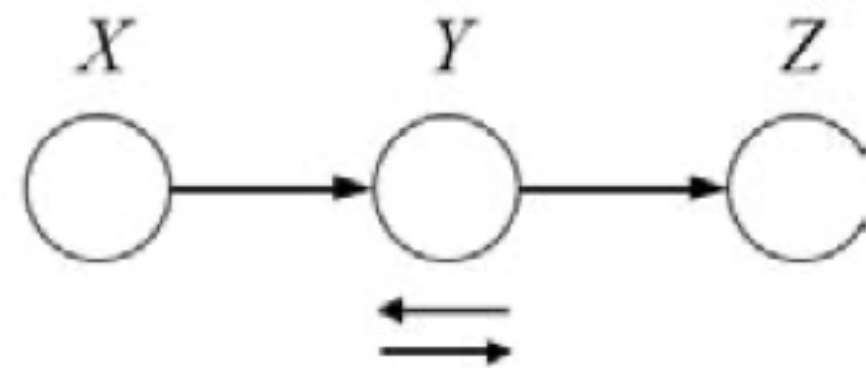
X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

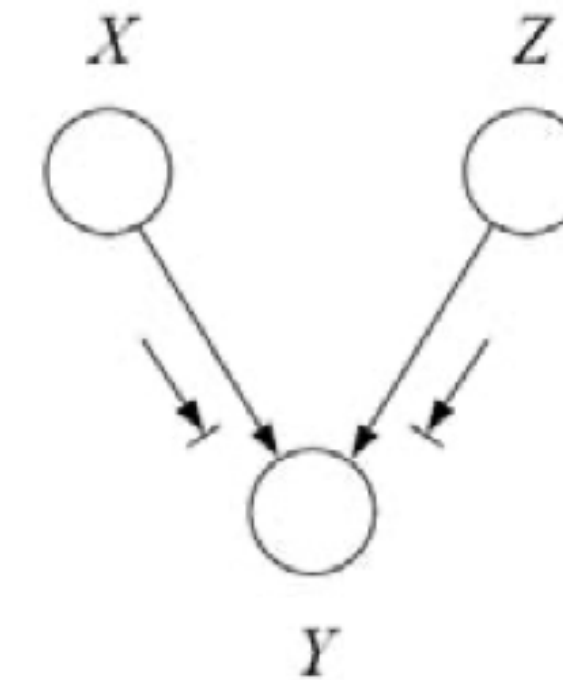
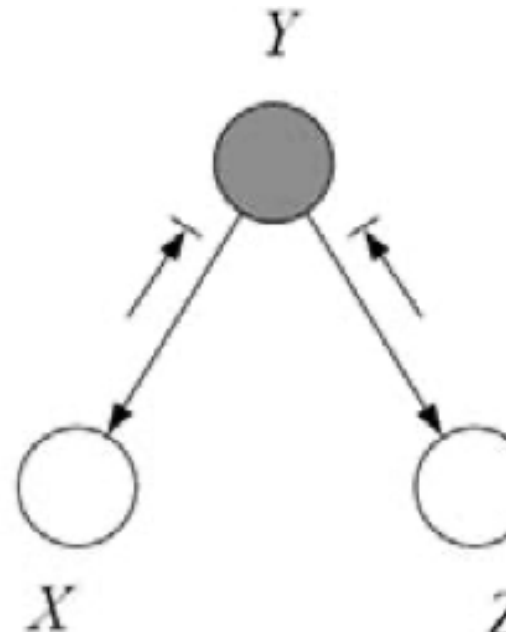
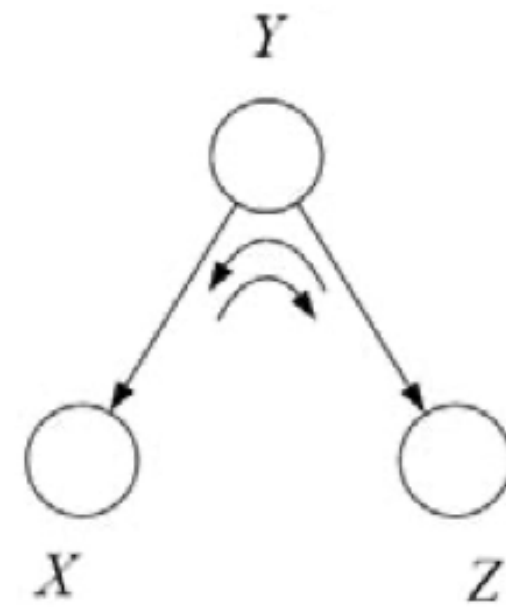
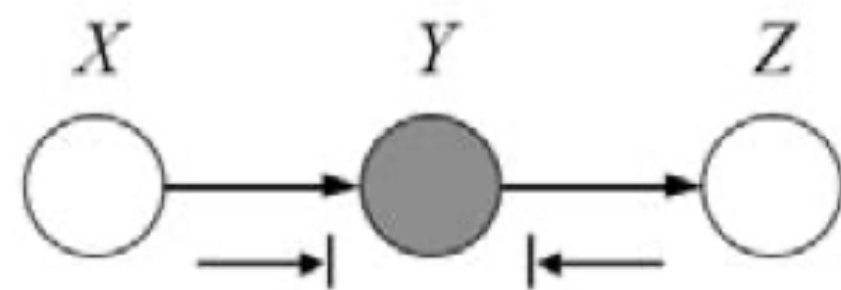
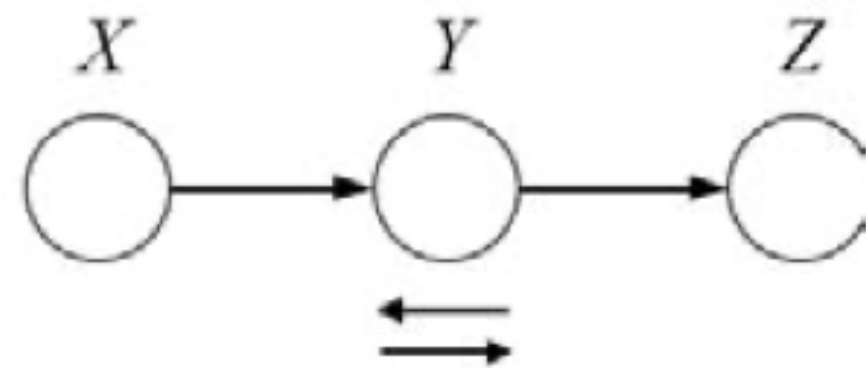
X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



Recap: Global Markov Properties of DAGs

How to determine two variables are conditionally independent given another variable?

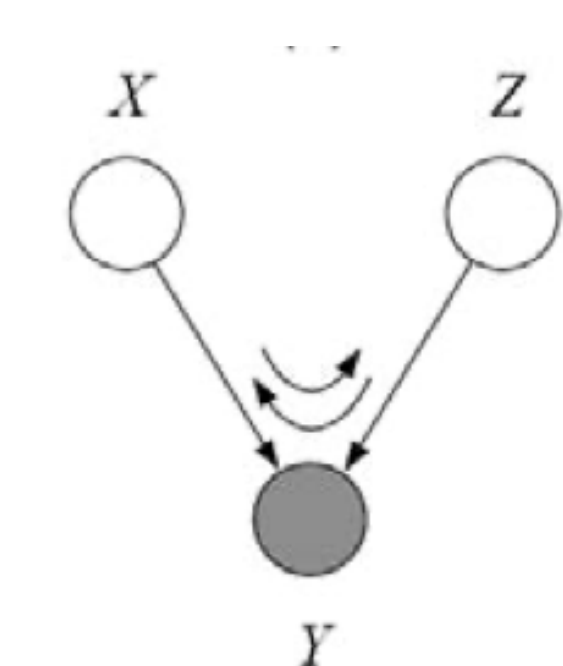
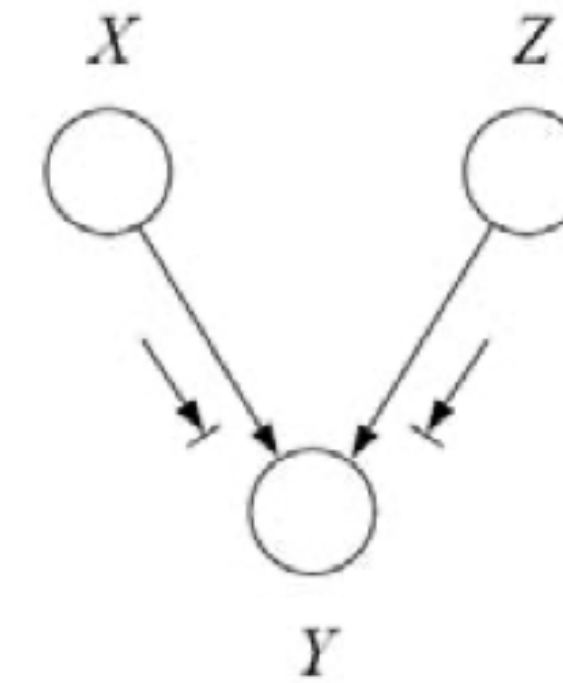
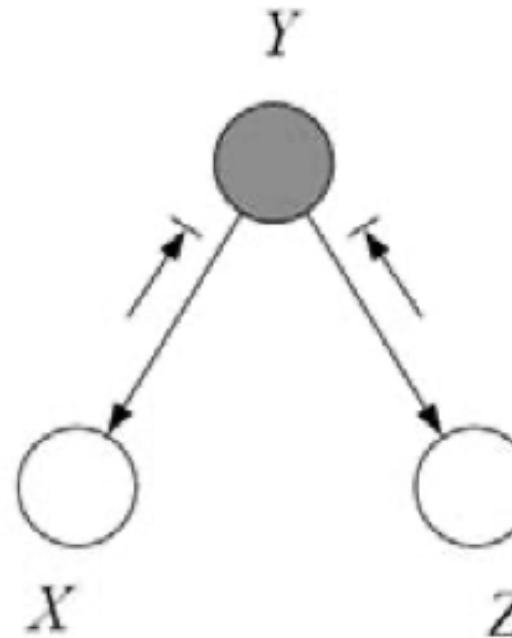
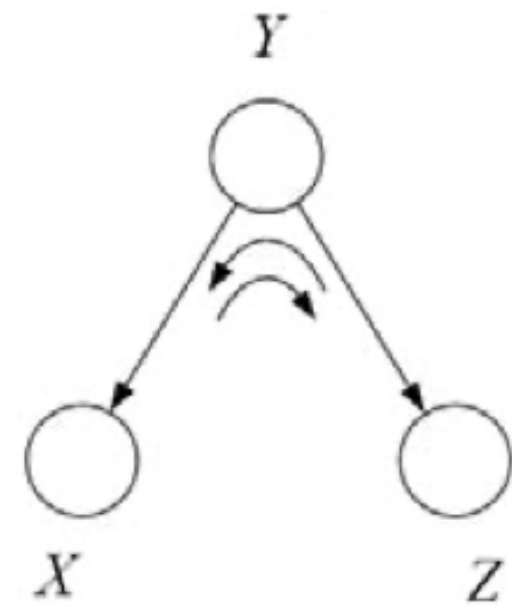
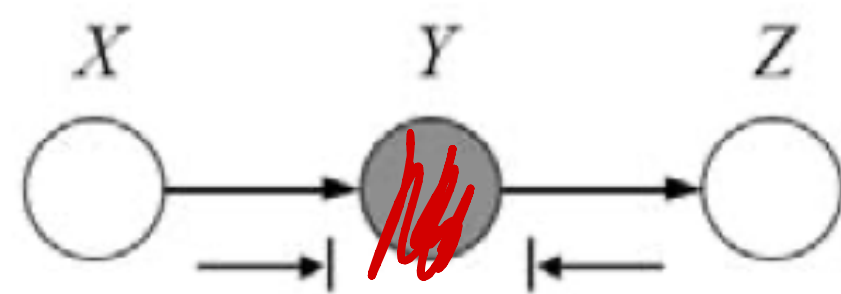
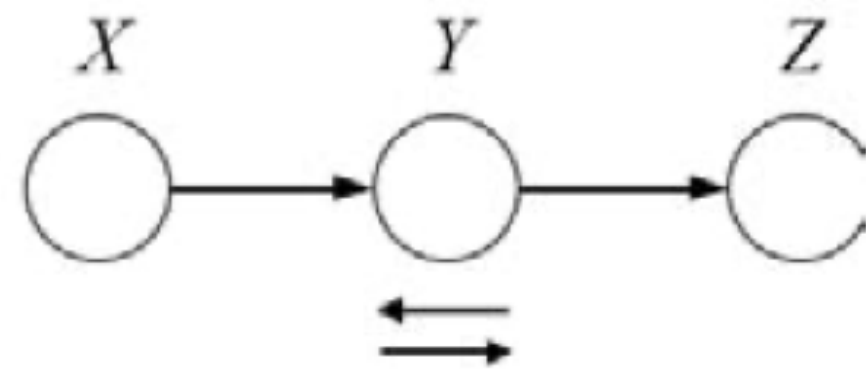
X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



Recap: Global Markov Properties of DAGs

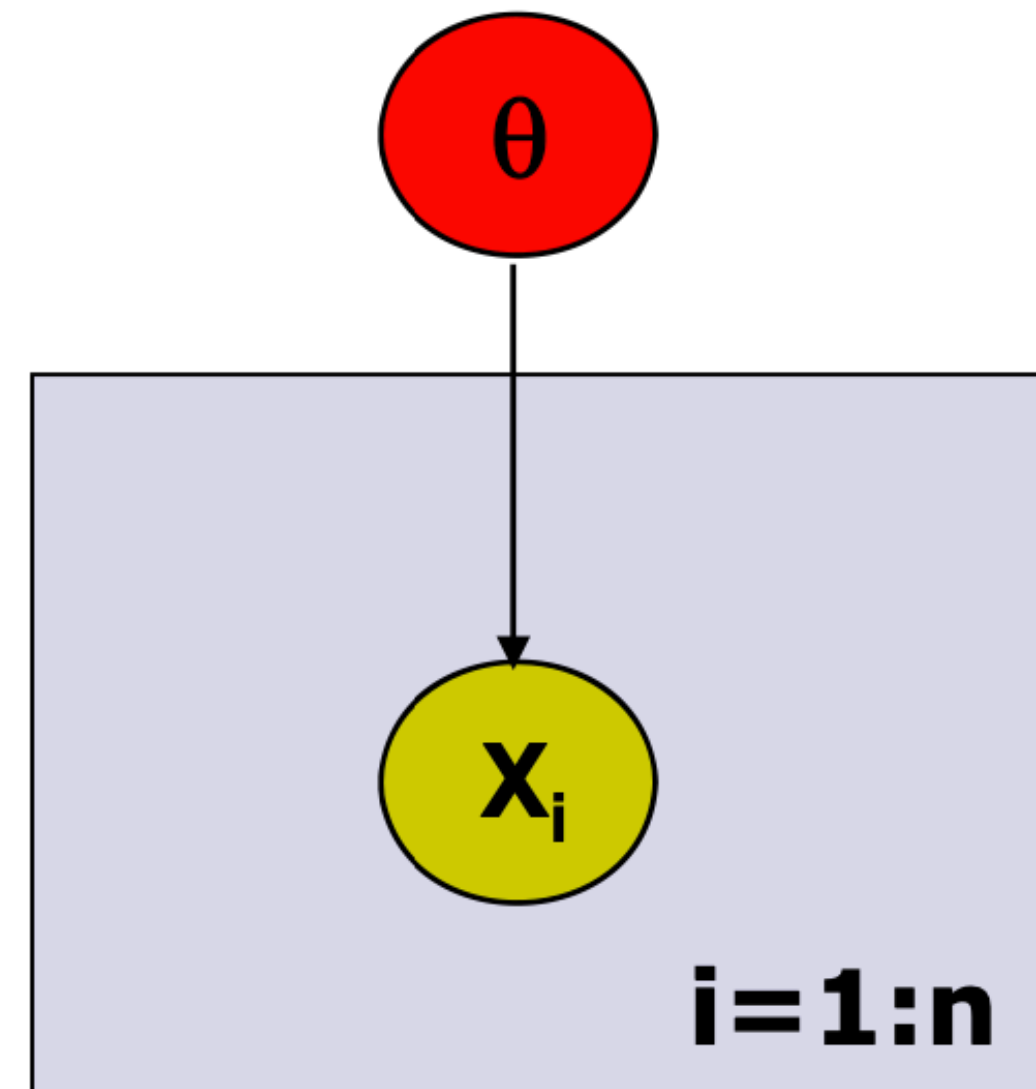
How to determine two variables are conditionally independent given another variable?

X is **d-separated** (directed-separated) from Z given Y if we can't send a ball from any node in X to any node in Z using the "**Bayes-ball**" algorithm illustrated below (and plus some boundary conditions):



“Plate” Notation

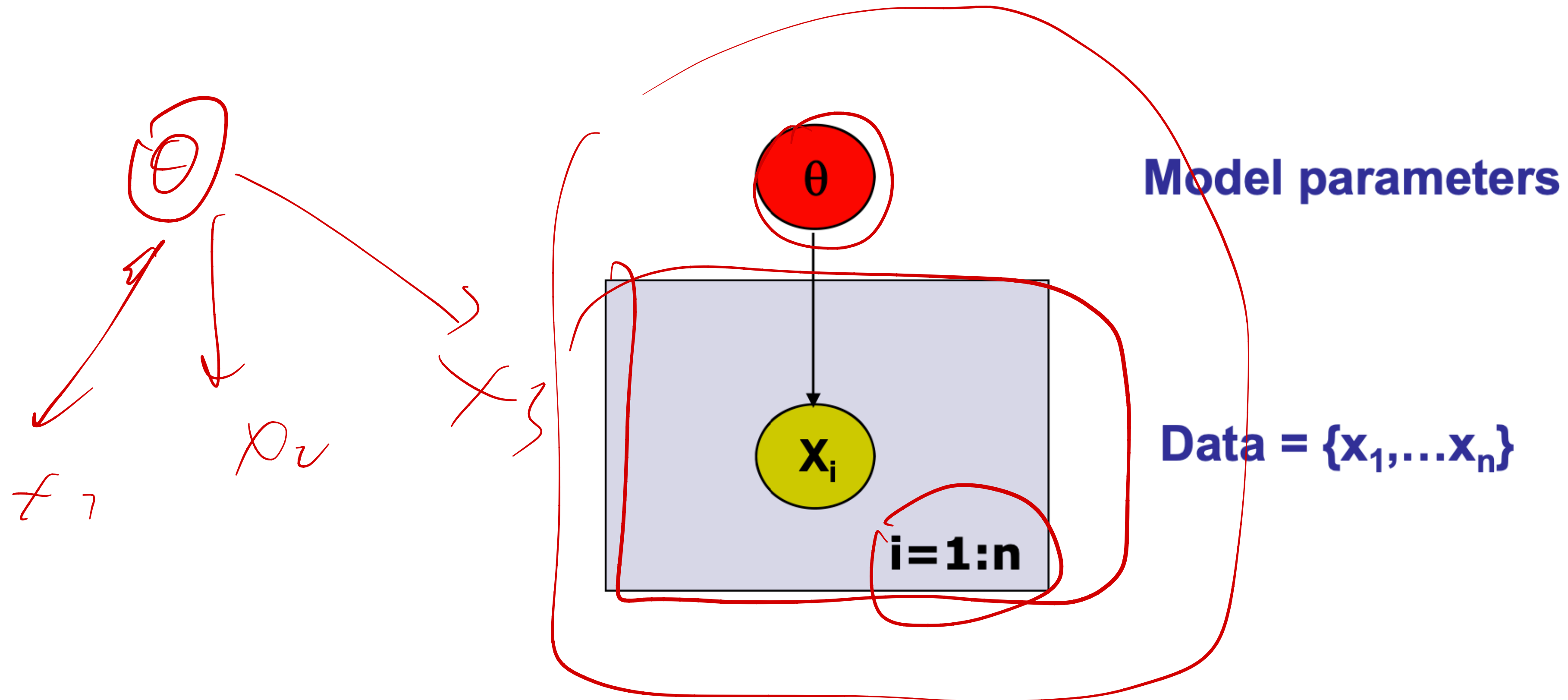
“Plate” Notation



Model parameters

Data = $\{x_1, \dots, x_n\}$

“Plate” Notation



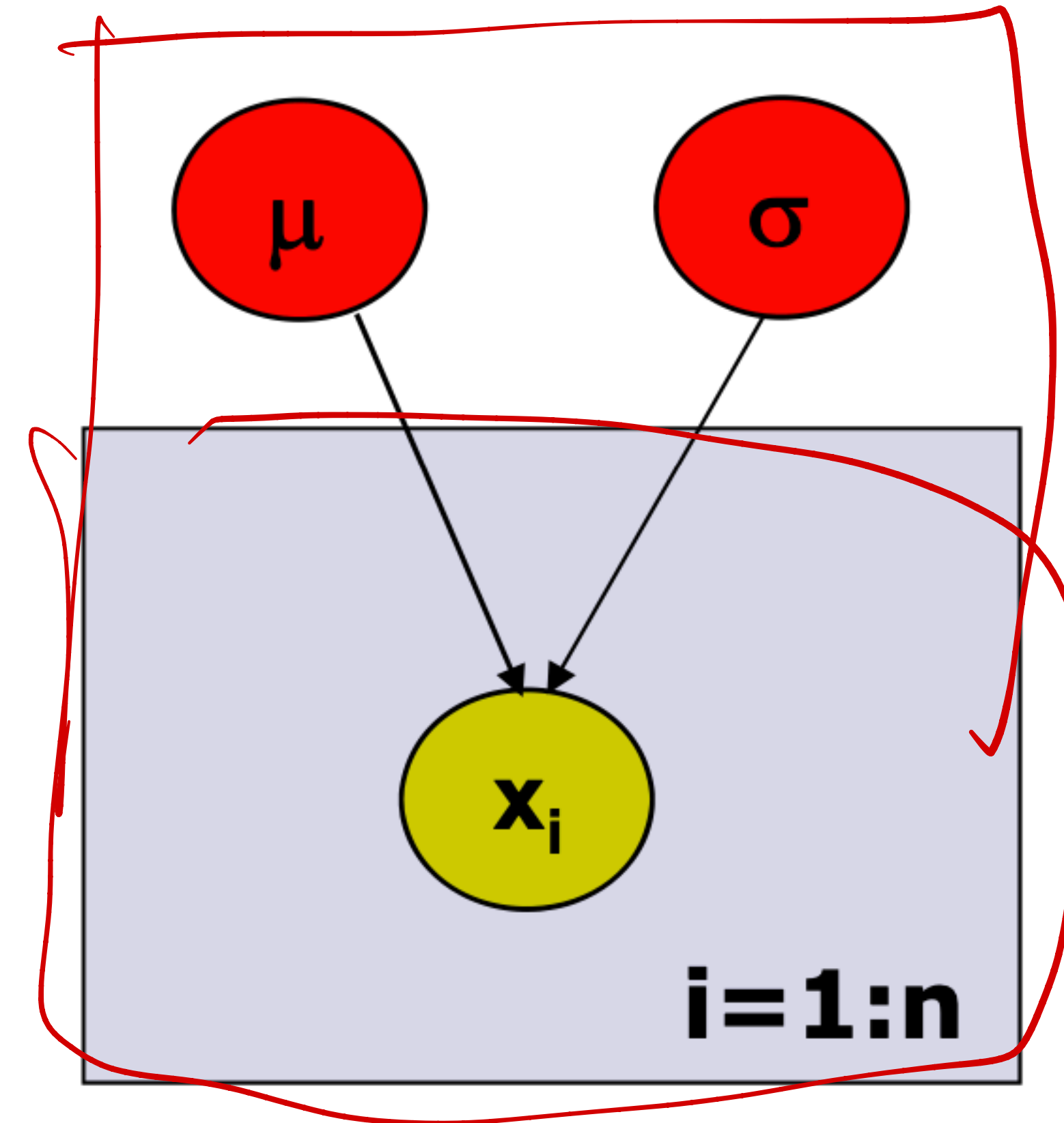
**variables within a plate are replicated
in a conditionally independent manner**

Example: Gaussian Model

Generative model:

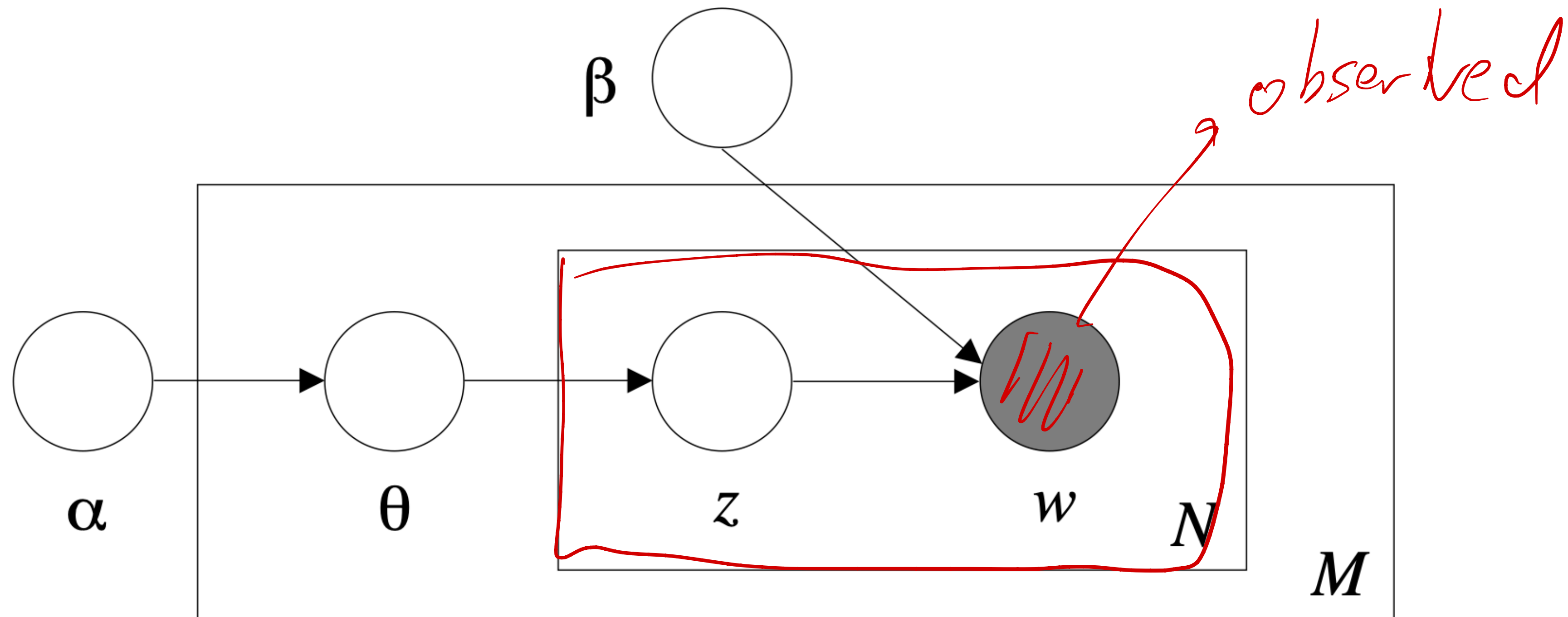
$$\begin{aligned} p(x_1, \dots, x_n \mid \mu, \sigma) &= \prod p(x_i \mid \mu, \sigma) \\ &= p(\text{data} \mid \text{parameters}) \\ &= p(D \mid \theta) \end{aligned}$$

where $\theta = \{\mu, \sigma\}$



$x \sim N(\mu, \sigma)$

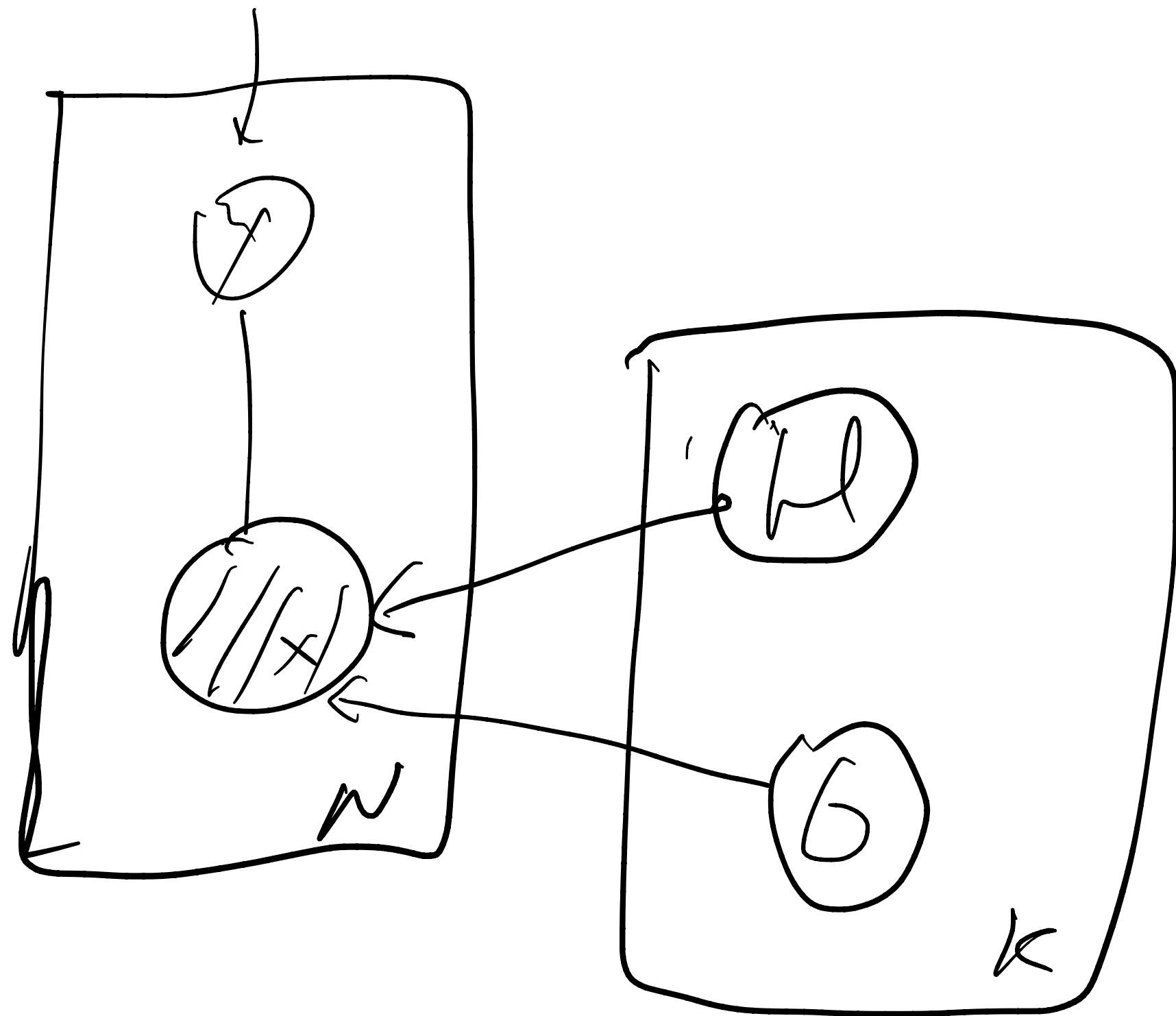
Observed Variable and Latent Variable Notations



Gaussian Mixture Model / Gaussian Discriminative Analysis in PGMs

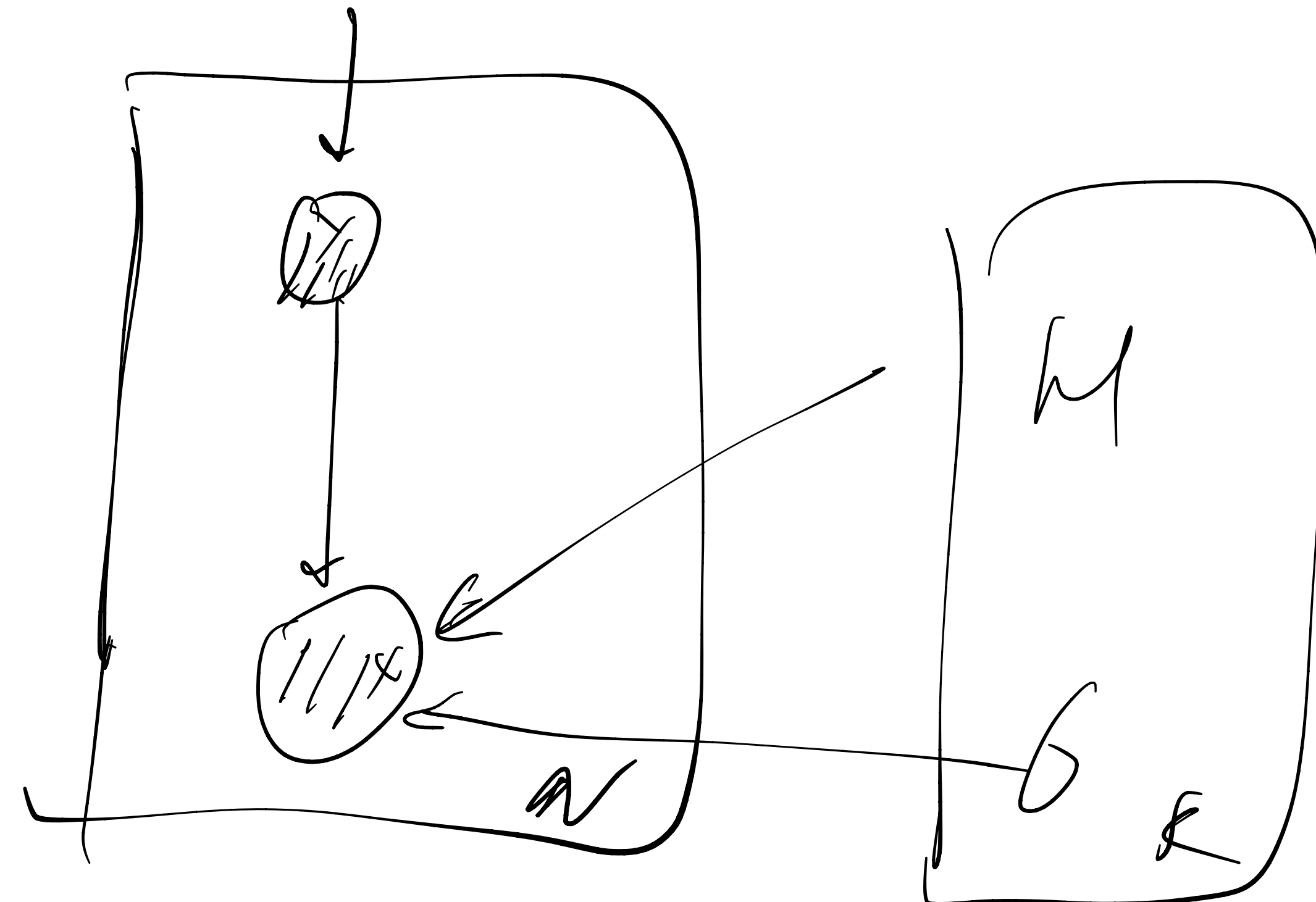
GMM

$P(x, y)$



GDA

$P(x, y)$



Inference and Learning

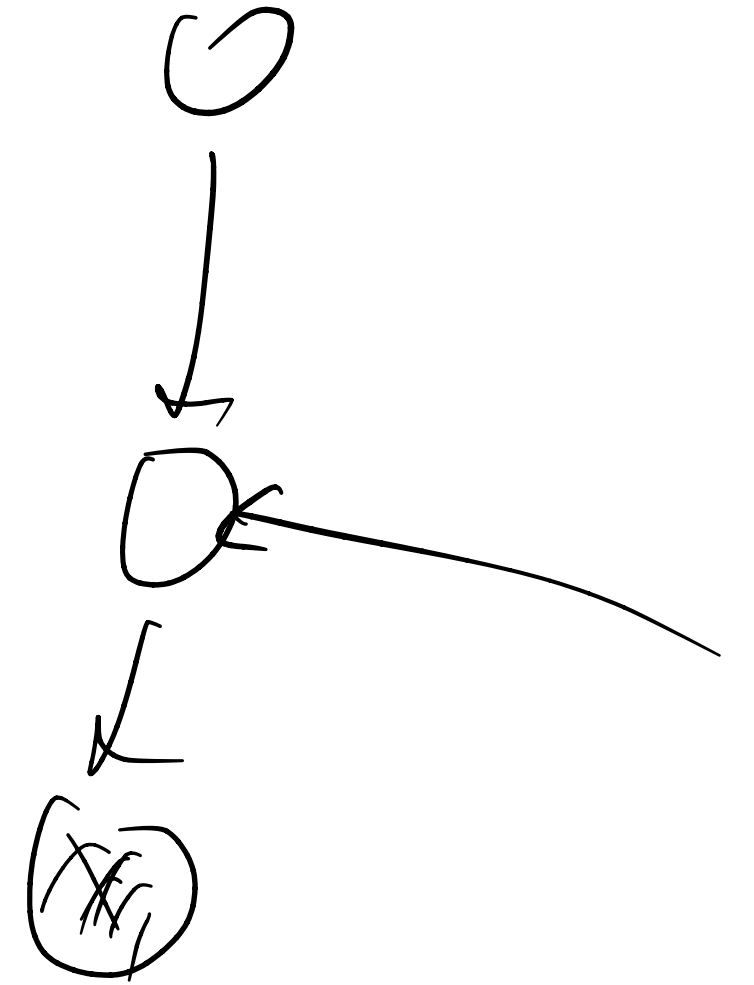
Inference and Learning

- Task 1: How do we answer **queries** about P ?
 - We use **inference** as a name for the process of computing answers to such queries

Inference and Learning

Query a node (random variable) in the graph

- Task 1: How do we answer **queries** about P ?
 - We use **inference** as a name for the process of computing answers to such queries



Inference and Learning

Query a node (random variable) in the graph

- Task 1: How do we answer **queries** about P ?
 - We use **inference** as a name for the process of computing answers to such queries
- Task 2: How do we estimate a **plausible model** M from data D ?
 - i. We use **learning** as a name for the process of obtaining point estimate of M .

Examples

Examples

- **Prediction:** what is the probability of an outcome given the starting condition

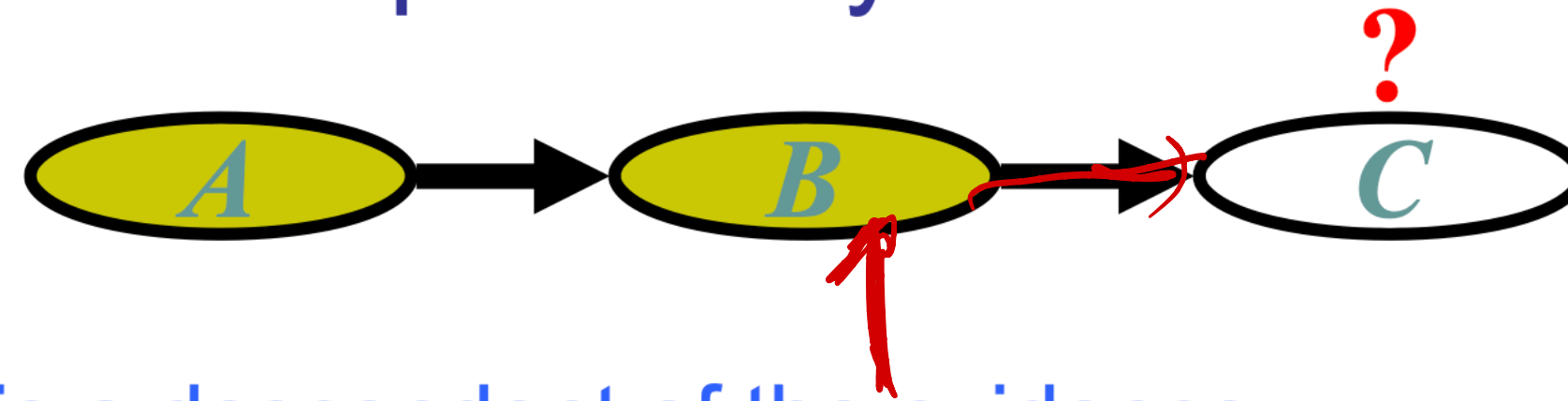


- the query node is a descendent of the evidence

$$P(C|A, B)$$

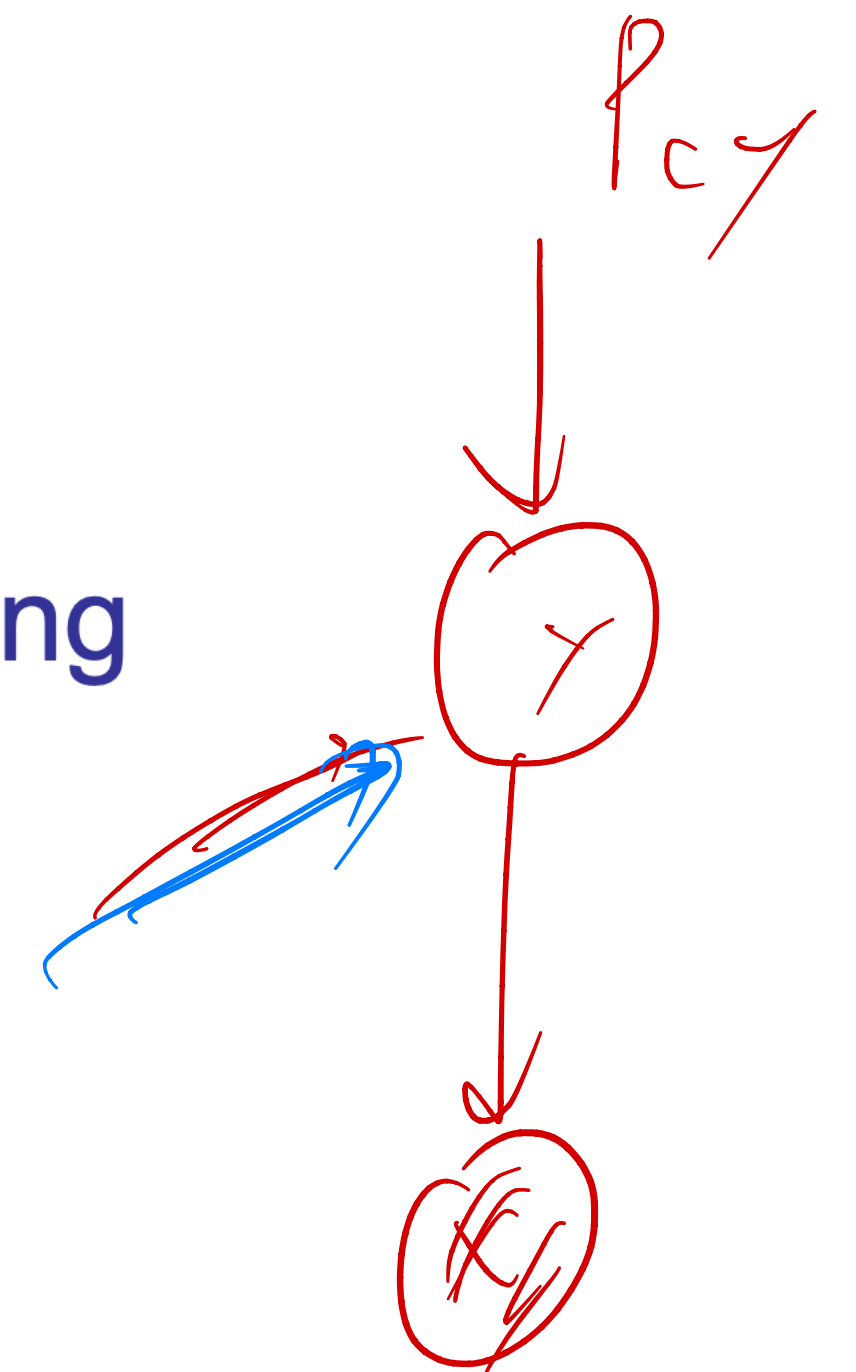
Examples

- **Prediction:** what is the probability of an outcome given the starting condition

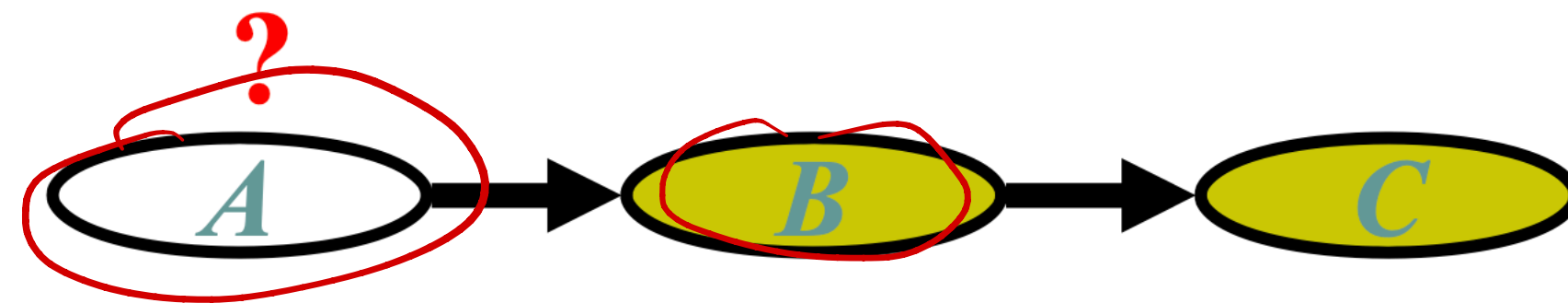


- the query node is a descendent of the evidence

$$P_C | X$$



- **Diagnosis:** what is the probability of disease/fault given symptoms



- the query node an ancestor of the evidence

$$P_A | B, C$$

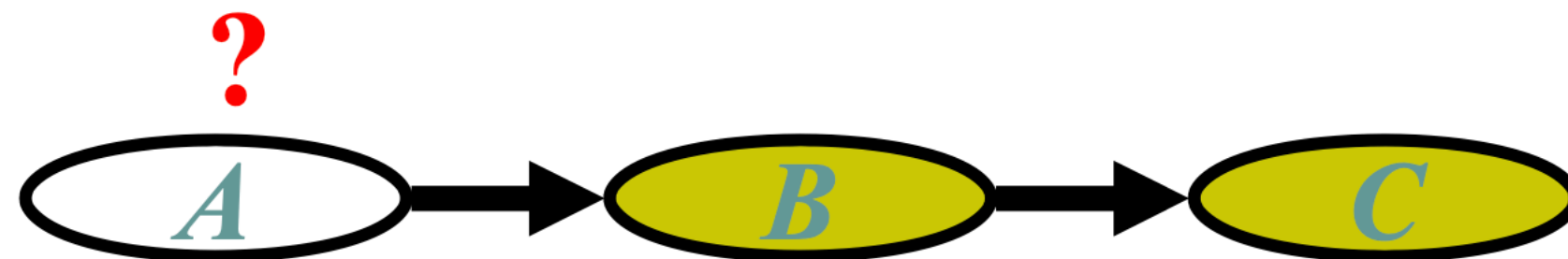
Examples

- **Prediction:** what is the probability of an outcome given the starting condition

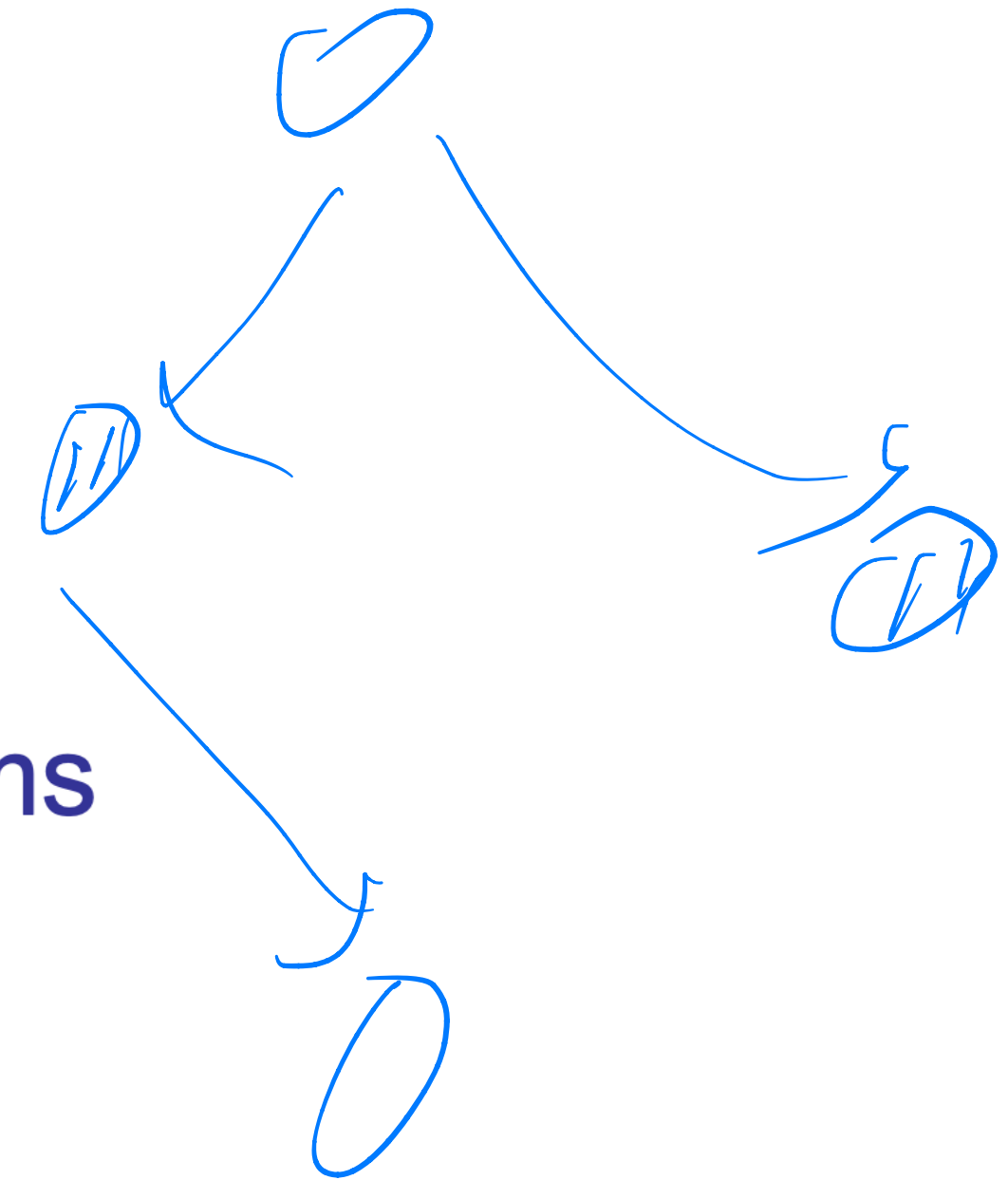


- the query node is a descendent of the evidence

- **Diagnosis:** what is the probability of disease/fault given symptoms



- the query node an ancestor of the evidence



In practice, the observed variable is often the data that is on the leaf nodes

How to Learn the Parameters

How to Learn the Parameters

When θ is the parameter and does not have prior \rightarrow MLE

$$\tilde{E}, \underline{q = P(z|x)}$$

$$\underline{M}: \text{ELBO}$$

$$= \underline{E_q \log \frac{P(x,z)}{q(z|x)}}$$

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

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

intractable

$$P(x; \theta) = \int_z p(x, z; \theta)$$

EM

How to do MLE on Latent Variable Models?

How to do MLE on Latent Variable Models?

Expectation Maximization!

How to do MLE on Latent Variable Models?

Expectation Maximization!

The E-step computes the posterior distribution $p(z | x)$

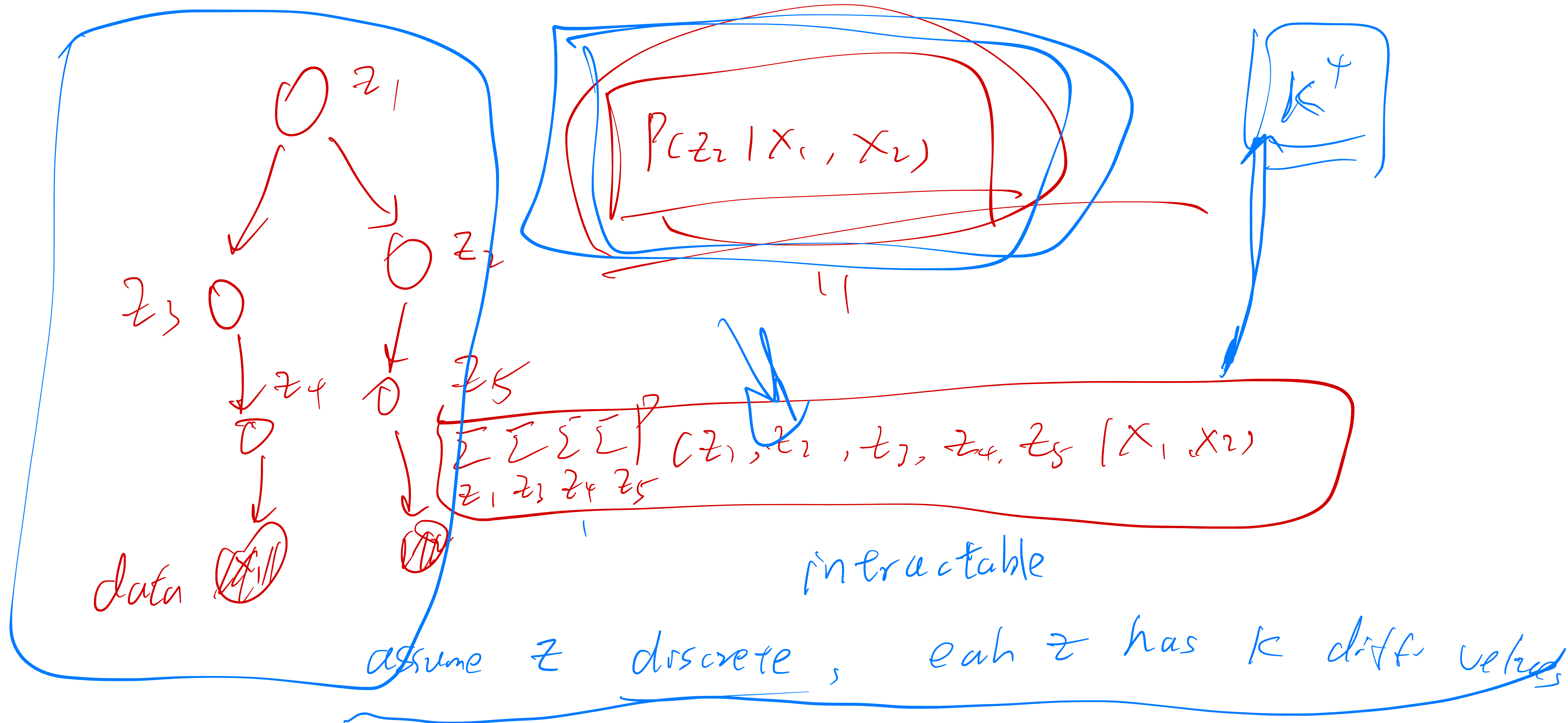
How to do MLE on Latent Variable Models?

Expectation Maximization!

The E-step computes the posterior distribution $p(z|x)$

This process is referred to as inference

Approaches to Inference



Approaches to Inference

- Exact inference algorithms

- The elimination algorithm
- Belief propagation
- The junction tree algorithms (but will not cover in detail here)

Approaches to Inference

- Exact inference algorithms
 - The elimination algorithm
 - Belief propagation
 - The junction tree algorithms (but will not cover in detail here)
- Approximate inference techniques
 - Variational algorithms
 - Stochastic simulation / sampling methods
 - Markov chain Monte Carlo methods

VAE

Approaches to Inference

- Exact inference algorithms

- The elimination algorithm
- Belief propagation
- The junction tree algorithms (but will not cover in detail here)

- Approximate inference techniques

- Variational algorithms
- Stochastic simulation / sampling methods
- Markov chain Monte Carlo methods

Variational Autoencoders

Elimination Algorithm/ Marginalization

$$P(h) = \sum_g \sum_f \sum_e \sum_d \sum_c \sum_b \sum_a P(a, b, c, d, e, f, g, h)$$

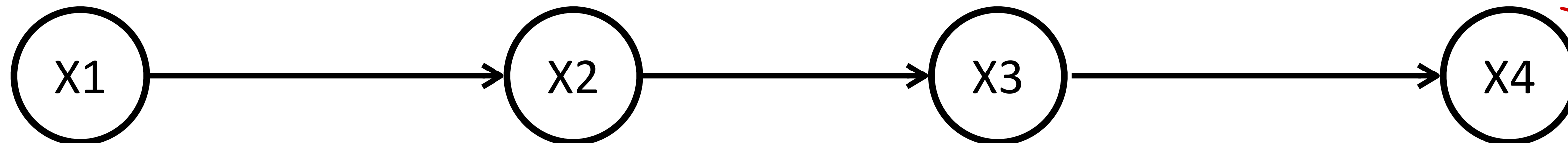
a naïve summation needs to
enumerate over an exponential
number of terms

Elimination Algorithm/ Marginalization

$$P(h) = \sum_g \sum_f \sum_e \sum_d \sum_c \sum_b \sum_a P(a, b, c, d, e, f, g, h)$$

a naïve summation needs to
enumerate over an exponential
number of terms

$$P(x_2) = \sum_{x_1} \sum_{x_3} \sum_{x_4} P(x_1, x_2, x_3, x_4)$$

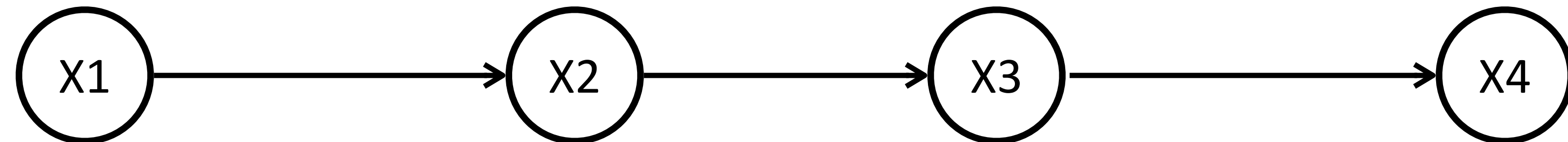


Elimination Algorithm/ Marginalization

$$P(h) = \sum_g \sum_f \sum_e \sum_d \sum_c \sum_b \sum_a P(a, b, c, d, e, f, g, h)$$

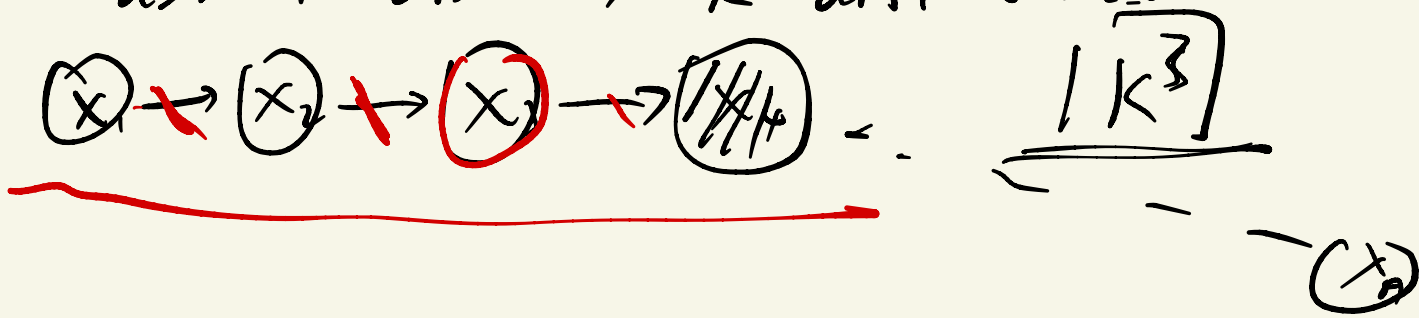


a naïve summation needs to
enumerate over an exponential
number of terms



What if the random variables follow this chain structure?

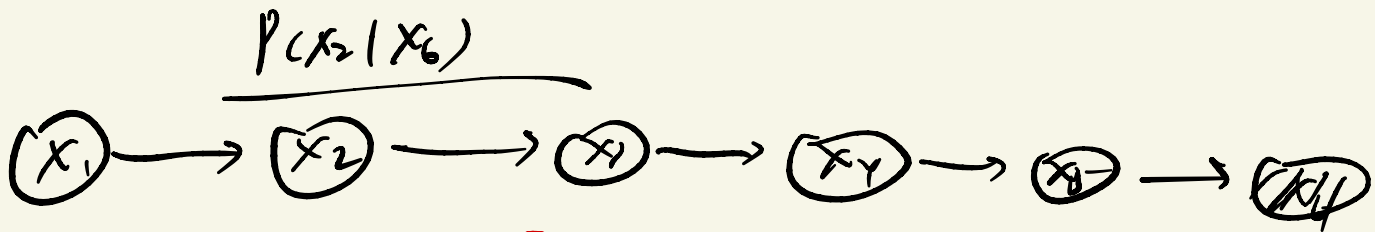
assump discrete, k diff values



$$\underline{P(x_1, \dots, x_4) = P(x_1) P(x_2|x_1) P(x_3|x_2) P(x_4|x_3)} \quad \underline{\underline{k^4}}$$

$$\sum_{x_1} \sum_{x_3} \sum_{x_4} P(x_1) P(x_2|x_1) P(x_3|x_2) P(x_4|x_3) \quad x_4$$

$$P(x_2|x_4) = \sum_{x_1, x_3} P$$



$$P(X_2 | X_6) = \sum_{x_1} \sum_{x_3} \sum_{x_4} \sum_{x_5} P(x_1, x_2, \dots, x_5 | X_6)$$

$$P(x_1, x_2, \dots, x_5 | X_6) = \frac{P(x_1) P(x_2 | x_1) \dots P(x_5 | x_4) P(X_6 | x_5)}{P(X_6)}$$

$$P(X_2 | X_6) = \sum_{x_1} \sum_{x_3} \sum_{x_4} P(x_1) P(x_2 | x_1) P(x_3 | x_2) P(x_4 | x_3) P(x_5 | x_4) \sum_{x_5} \frac{P(x_5 | x_4) P(X_6 | x_5)}{P(X_6)}$$

for each value
 x_4, X_6

$$f(x_4, X_6)$$

no x_5

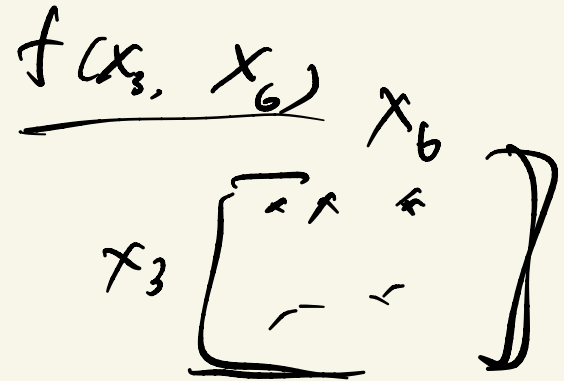
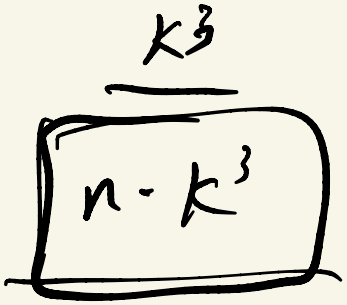
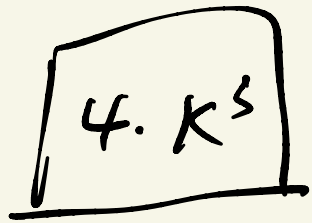
$$\sum_{x_1, x_6} \sum_{x_4} P(x_1) P(x_2|x_1) P(x_3|x_2) P(x_4|x_3) f(x_4, x_6)$$

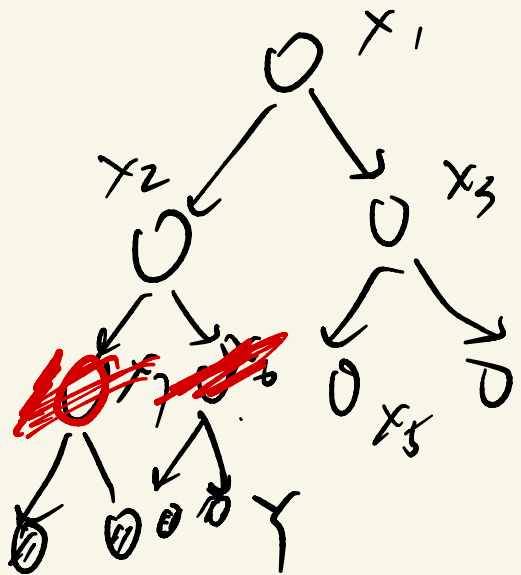
$$= \sum_{x_1, x_3} P(x_1) P(x_2|x_1) P(x_3|x_2)$$

$$\sum_{x_4} P(x_4|x_3) f(x_4, x_6)$$

x_1, \dots, x_n

4.





$$P(z|x)$$

$$P(x_6 | y)$$

$$\underline{P(x_1)} \quad \underline{P(x_2|x_1)} \quad \underline{P(x_3|x_1)} \quad P(x_7|x_2) \quad - \quad -$$

Auto-Encoding Variational Bayes

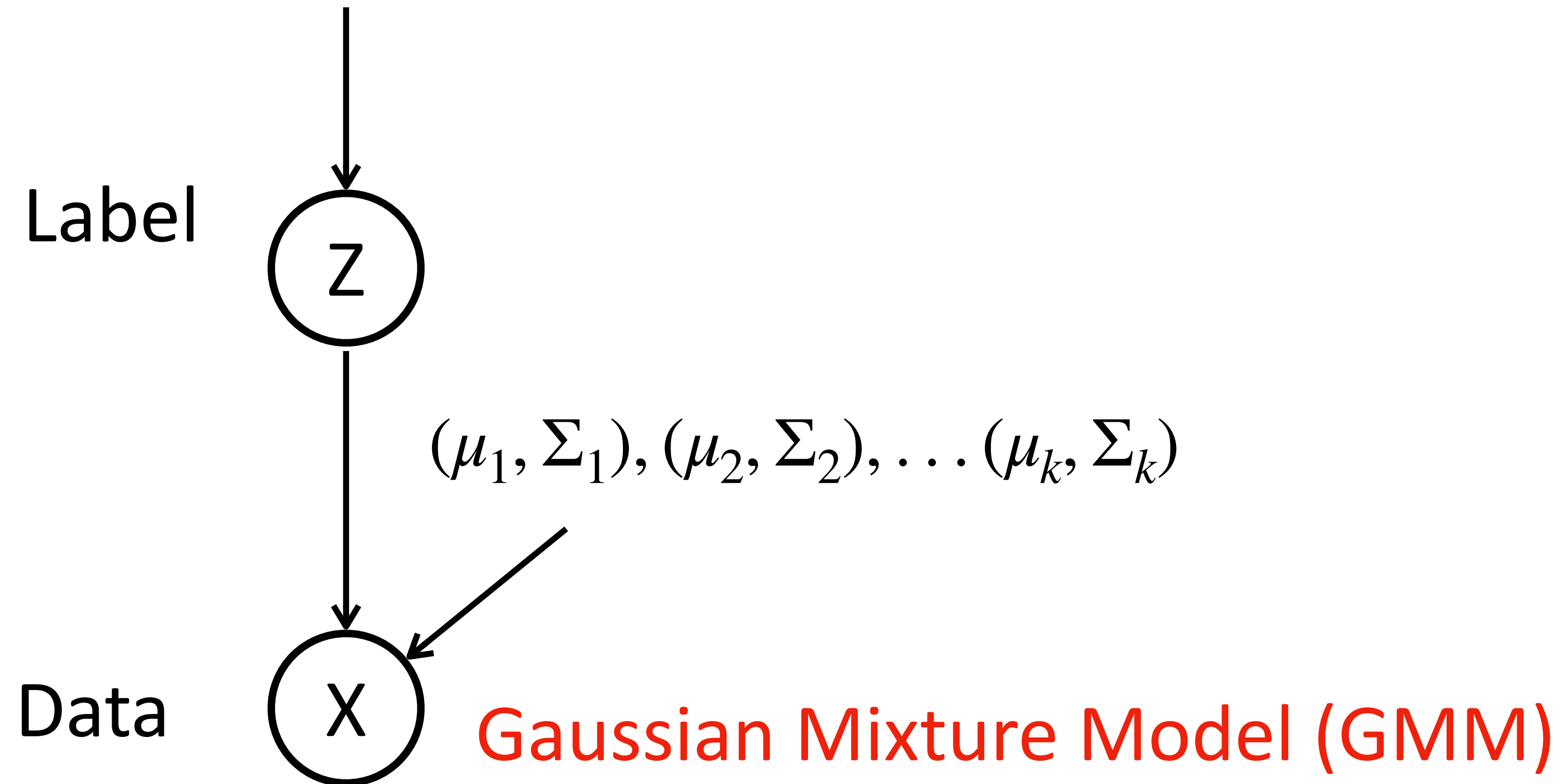
Diederik P. Kingma
Machine Learning Group
Universiteit van Amsterdam
dpkingma@gmail.com

Max Welling
Machine Learning Group
Universiteit van Amsterdam
welling.max@gmail.com

Variational Autoencoders

VAE is a Generative Model

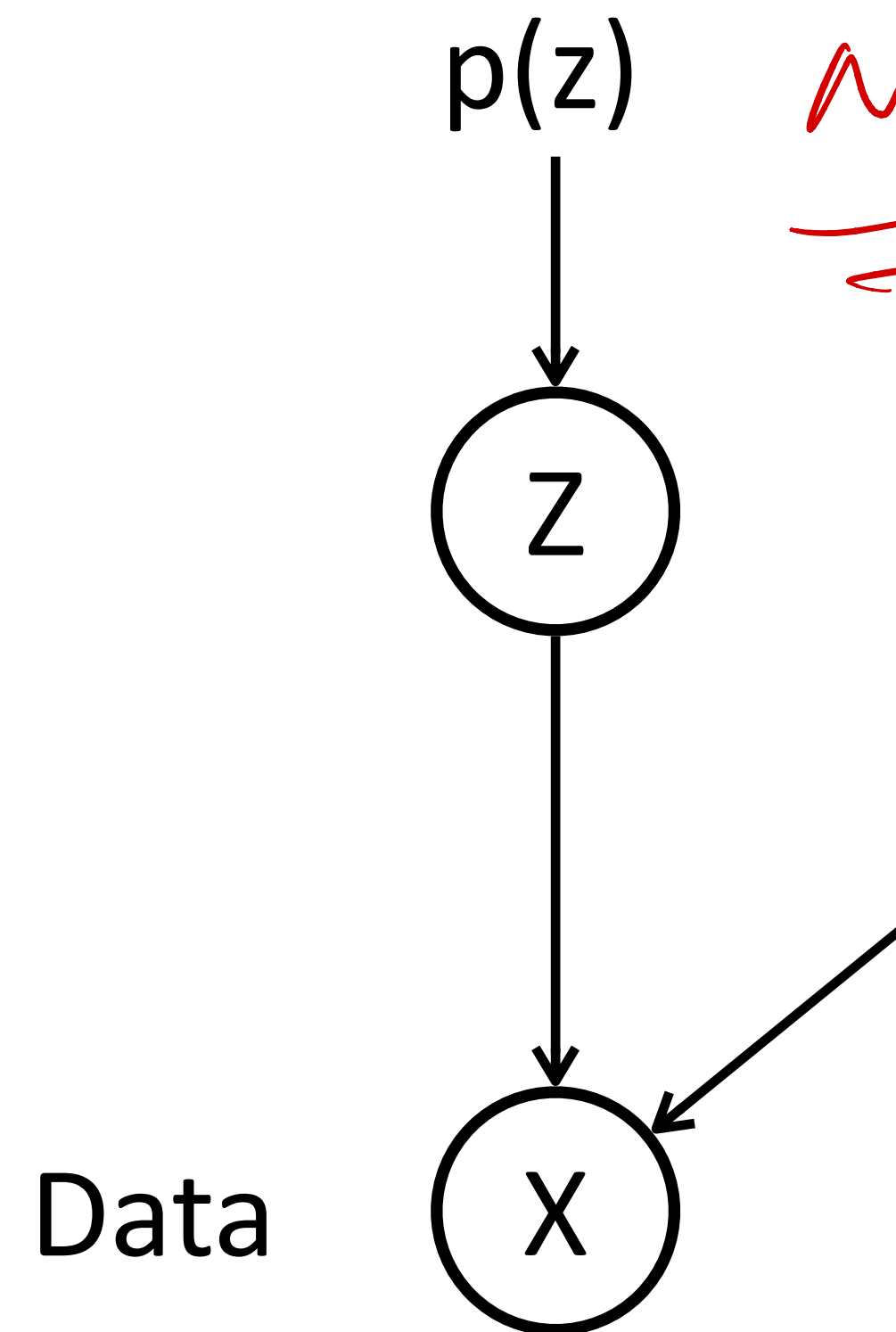
$p(z)$: multinomial, k
classes (e.g. uniform)



The VAE Model

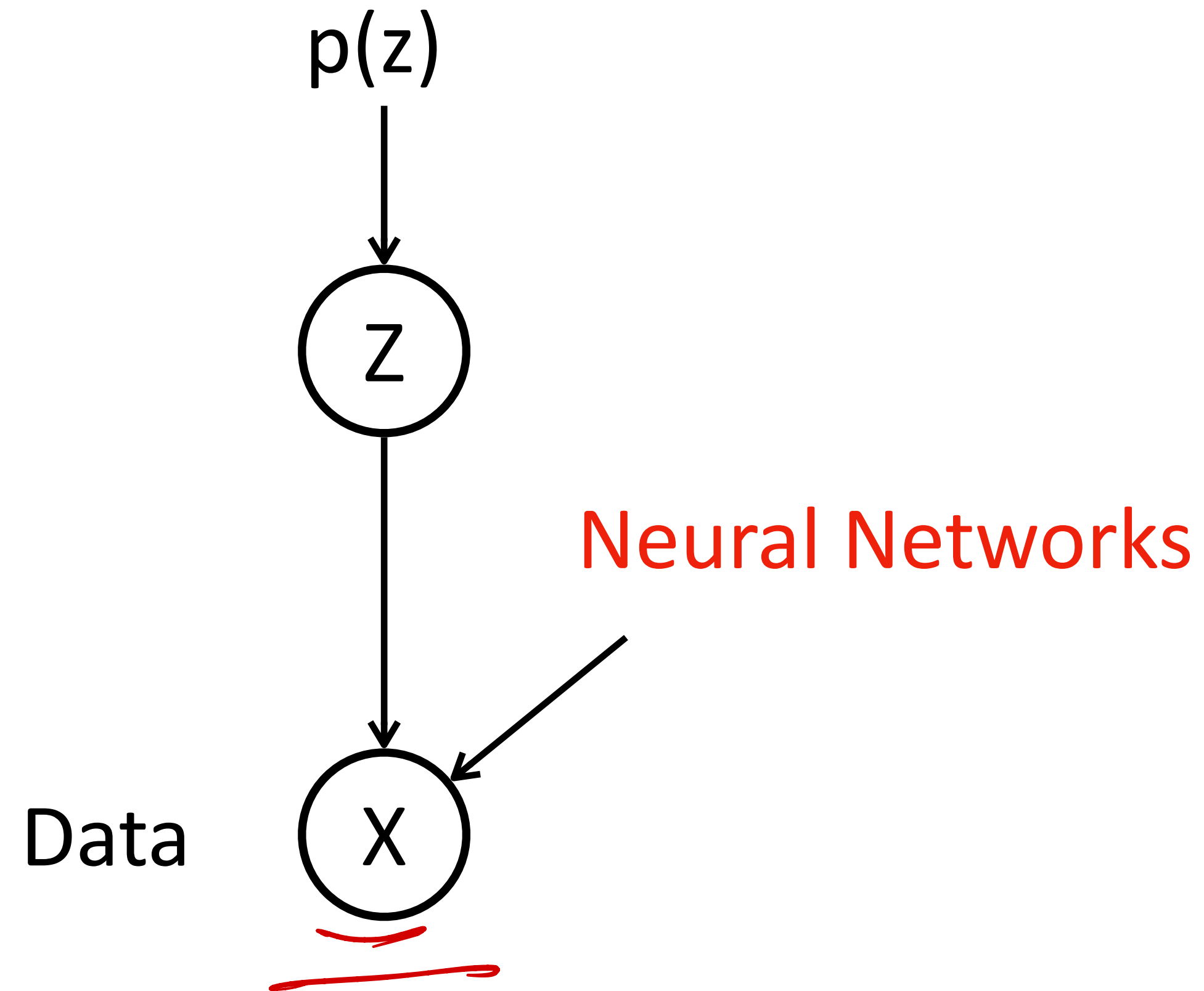
$p(z)$ is a normal distribution in most cases

$\mathcal{N}(0, 1)$



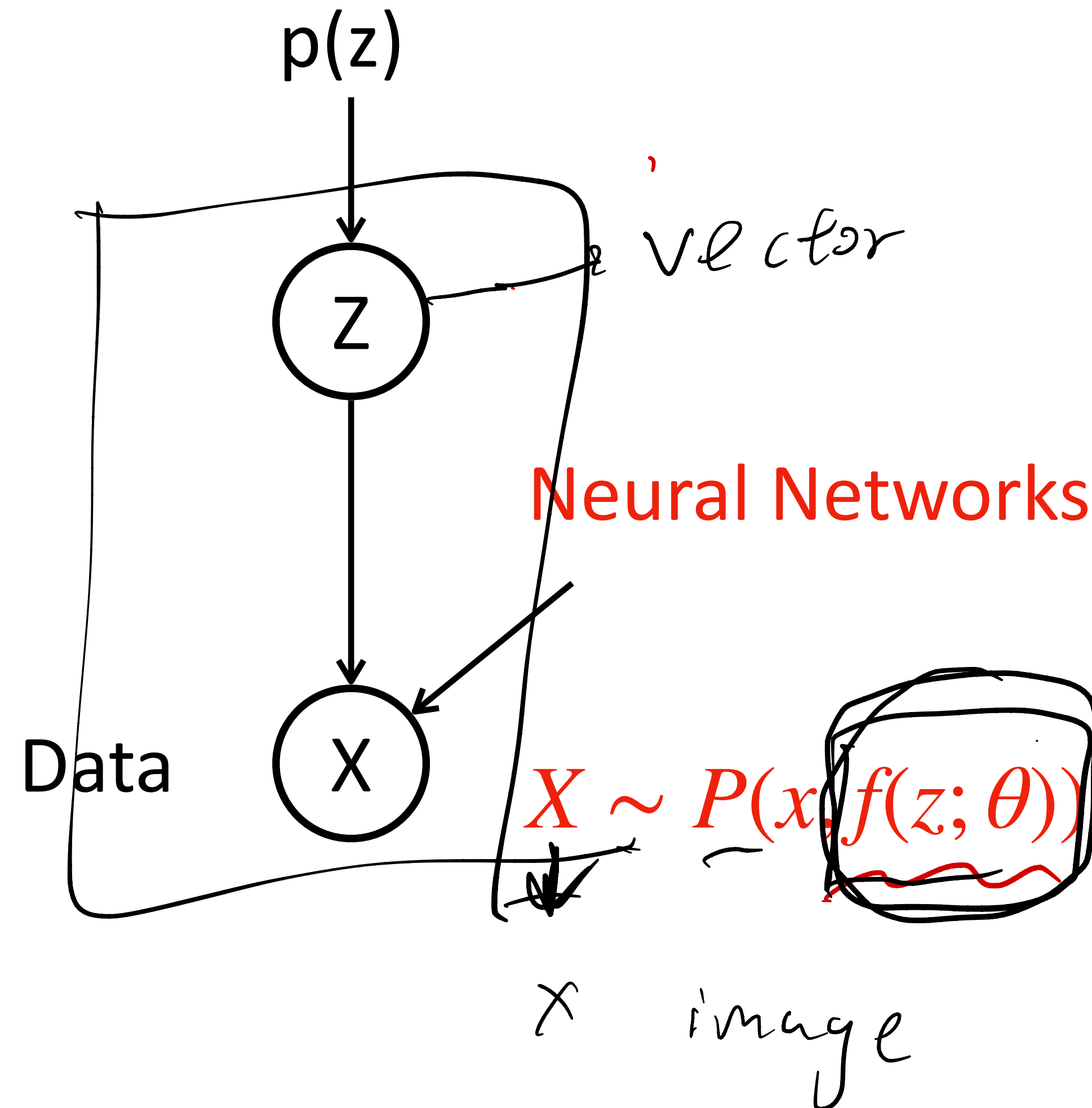
The VAE Model

$p(z)$ is a normal distribution in most cases



The VAE Model

$p(z)$ is a normal distribution in most cases

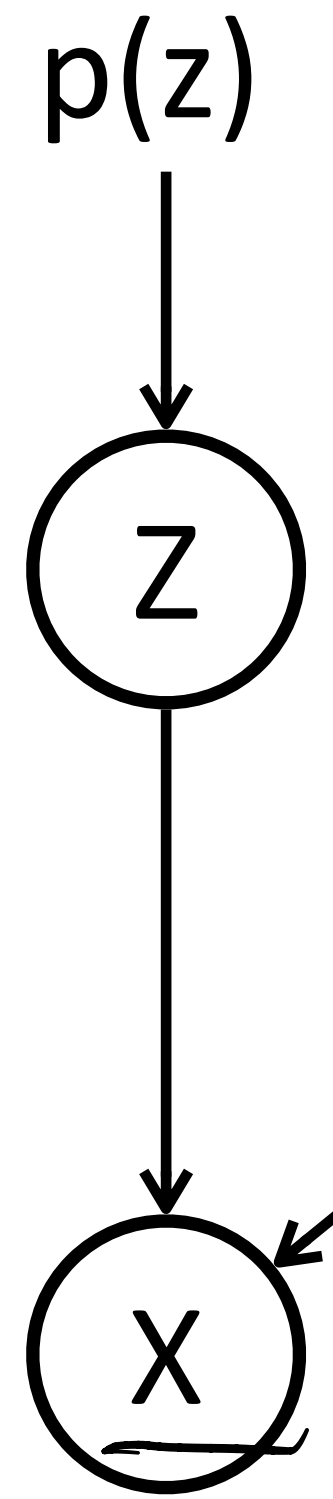


The VAE Model

$p(z)$ is a normal distribution in most cases

x discrete
word ~ categorical
[0.1, 0.2, 0.5] proba

$f(z; \theta)$ Data



Neural Networks

$$x \sim P(x, f(z; \theta))$$

f is a neural network taking Z as input

0 0 1 0 0 1 0 0 prob

Softmax

$\uparrow M$

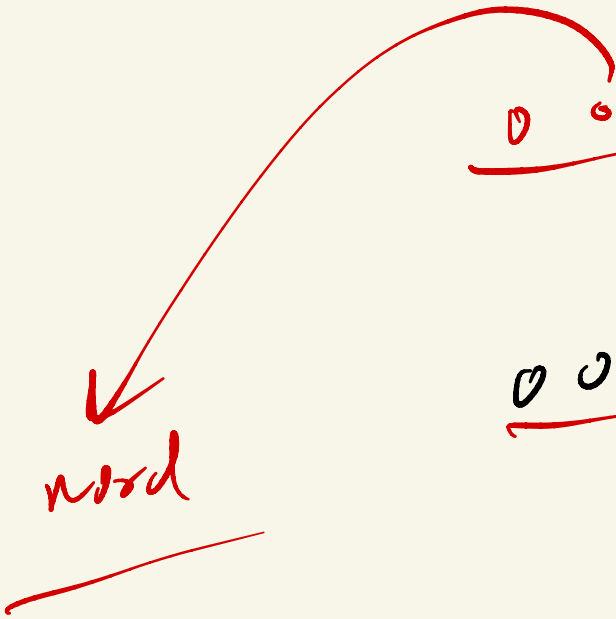
0 0 0 0 0 0 0 0 vector

$\uparrow f(z)$

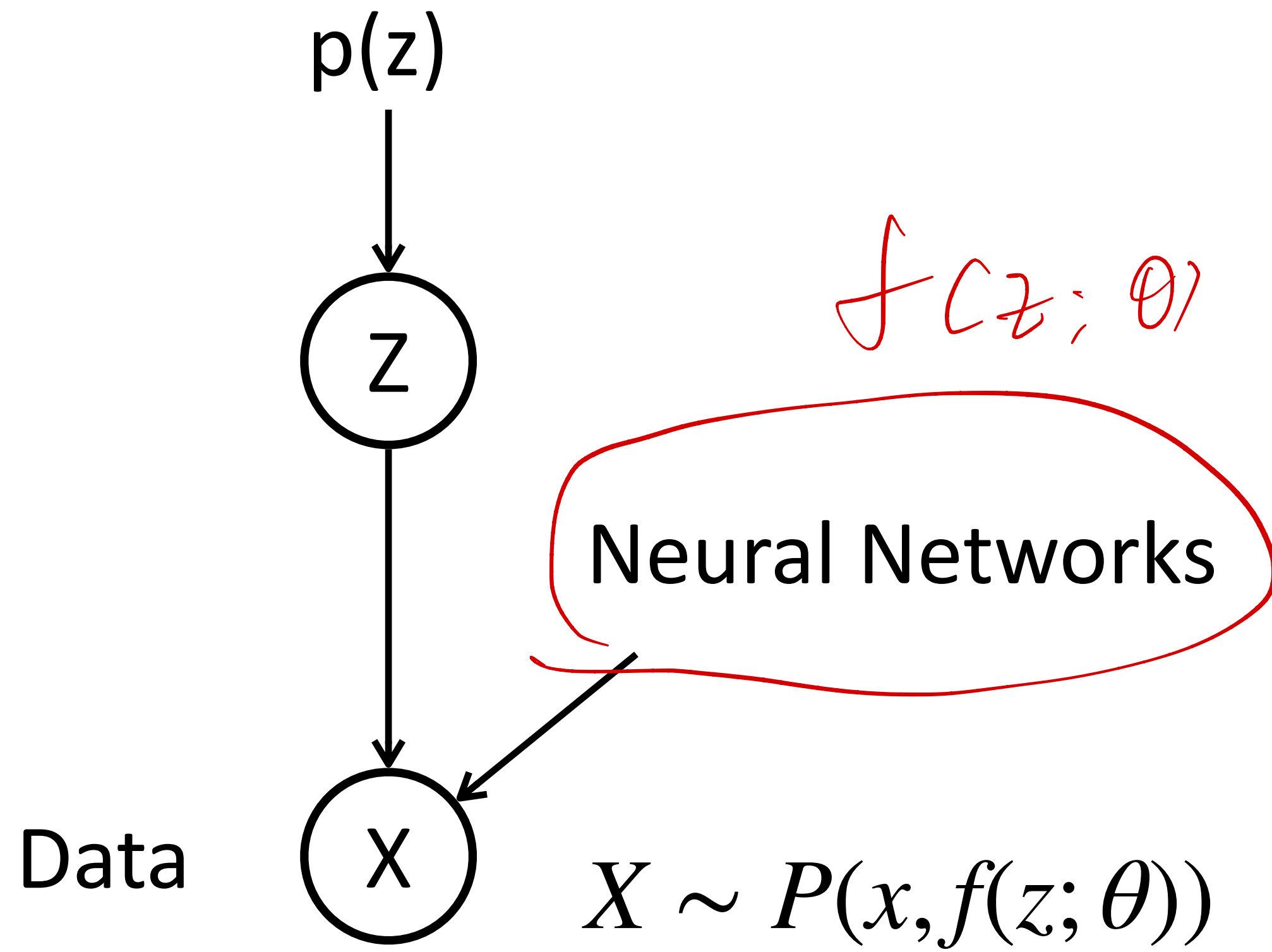
0 0 0 0 0 0
 z

$\uparrow N(0,1)$

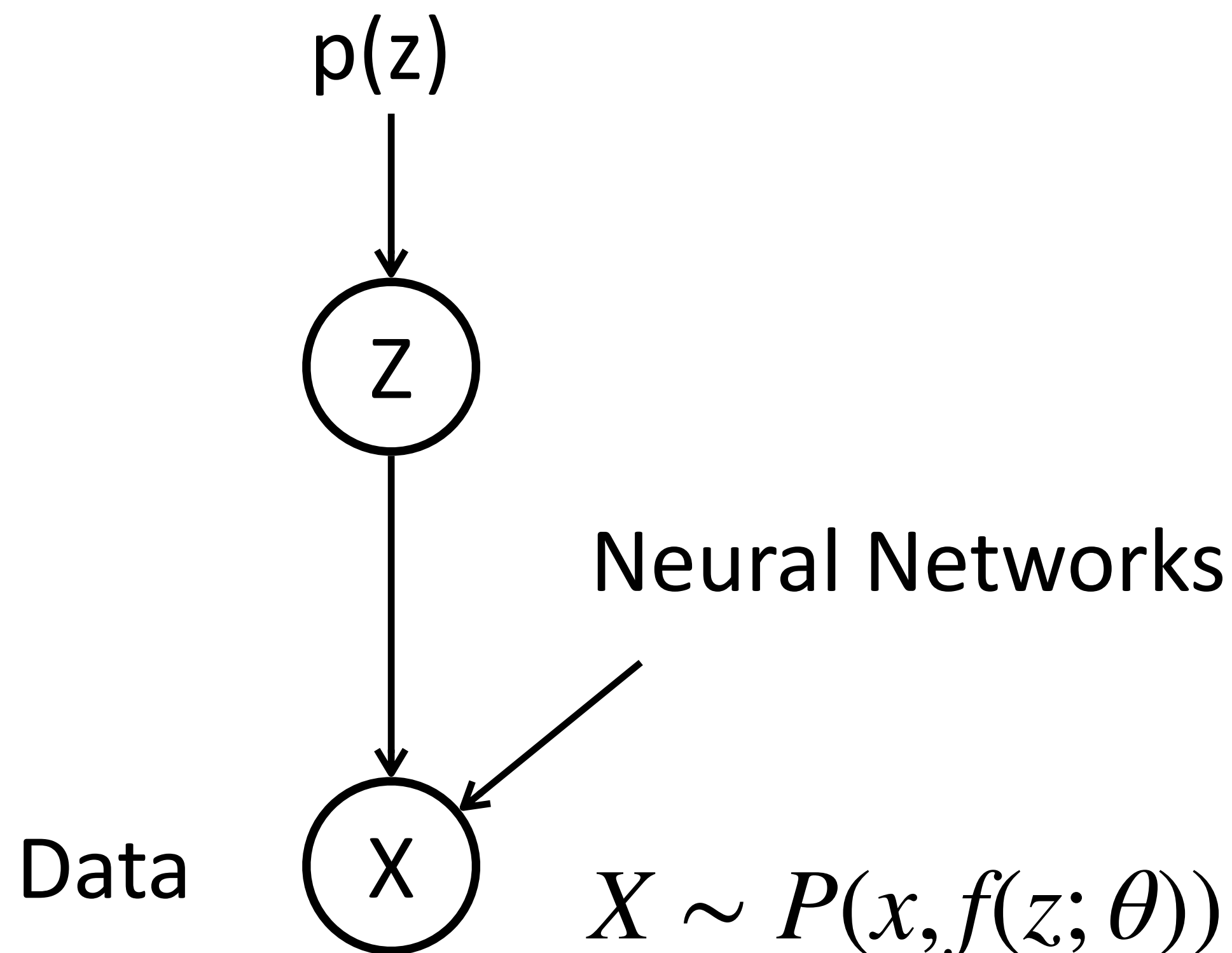
word



Training



Training



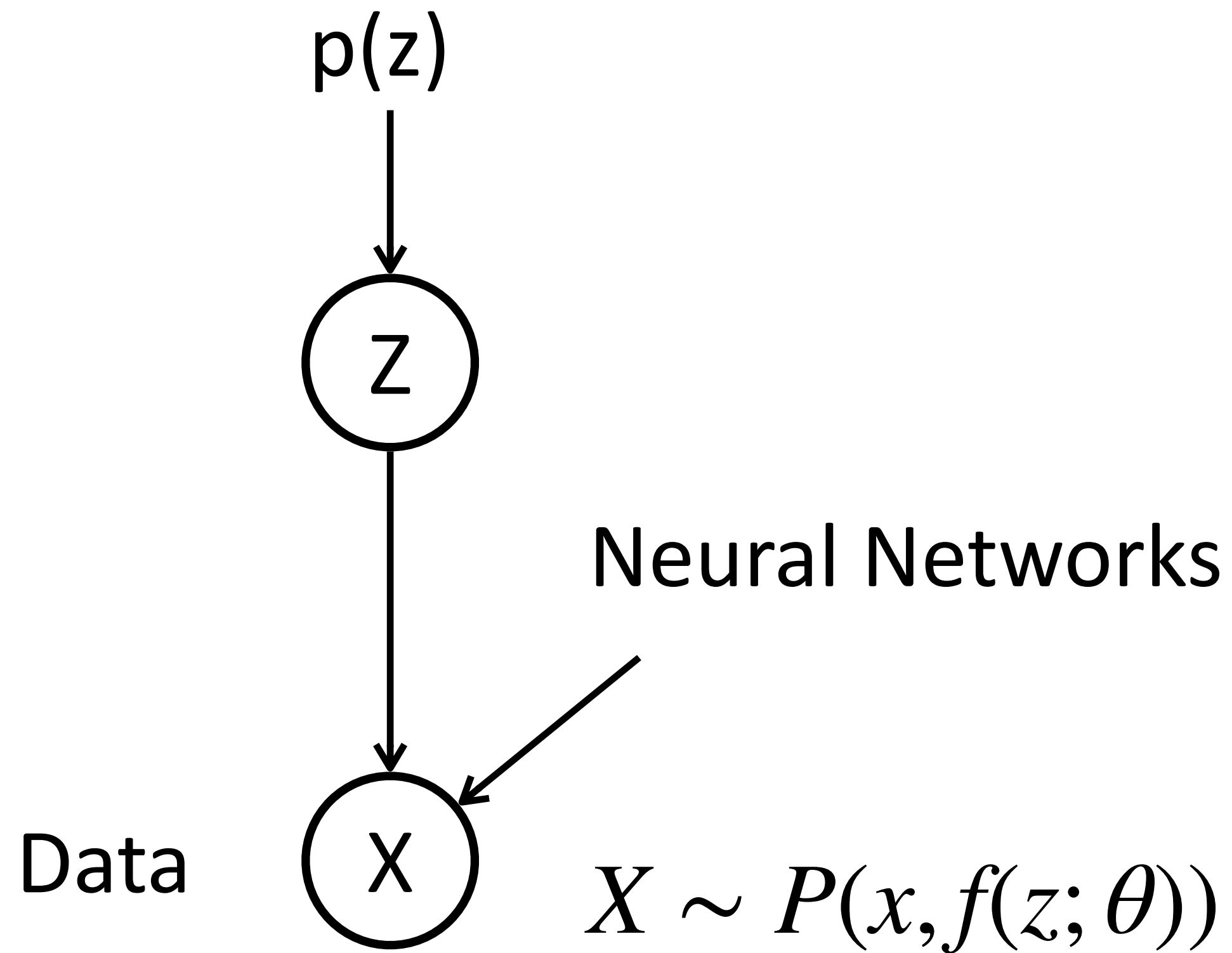
How to train the model? Can we do MLE?

$$\underline{\underline{P(x)}}$$

$$P(x) = \int_z P(x, z) = \int_z P(z) P(x|z)$$

$$= \int_z P(z) N(x|f(z), \sigma^2)$$

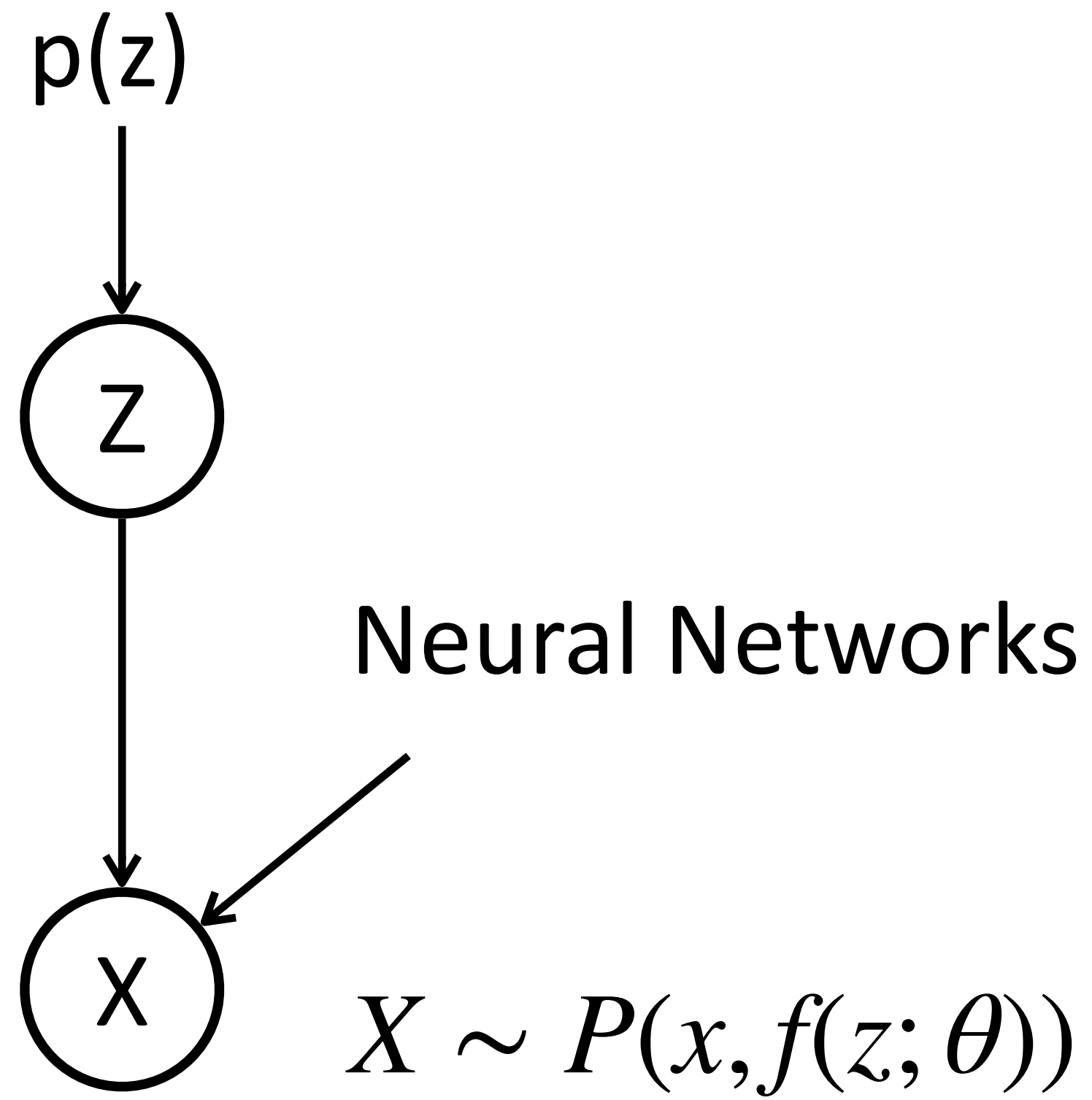
Training



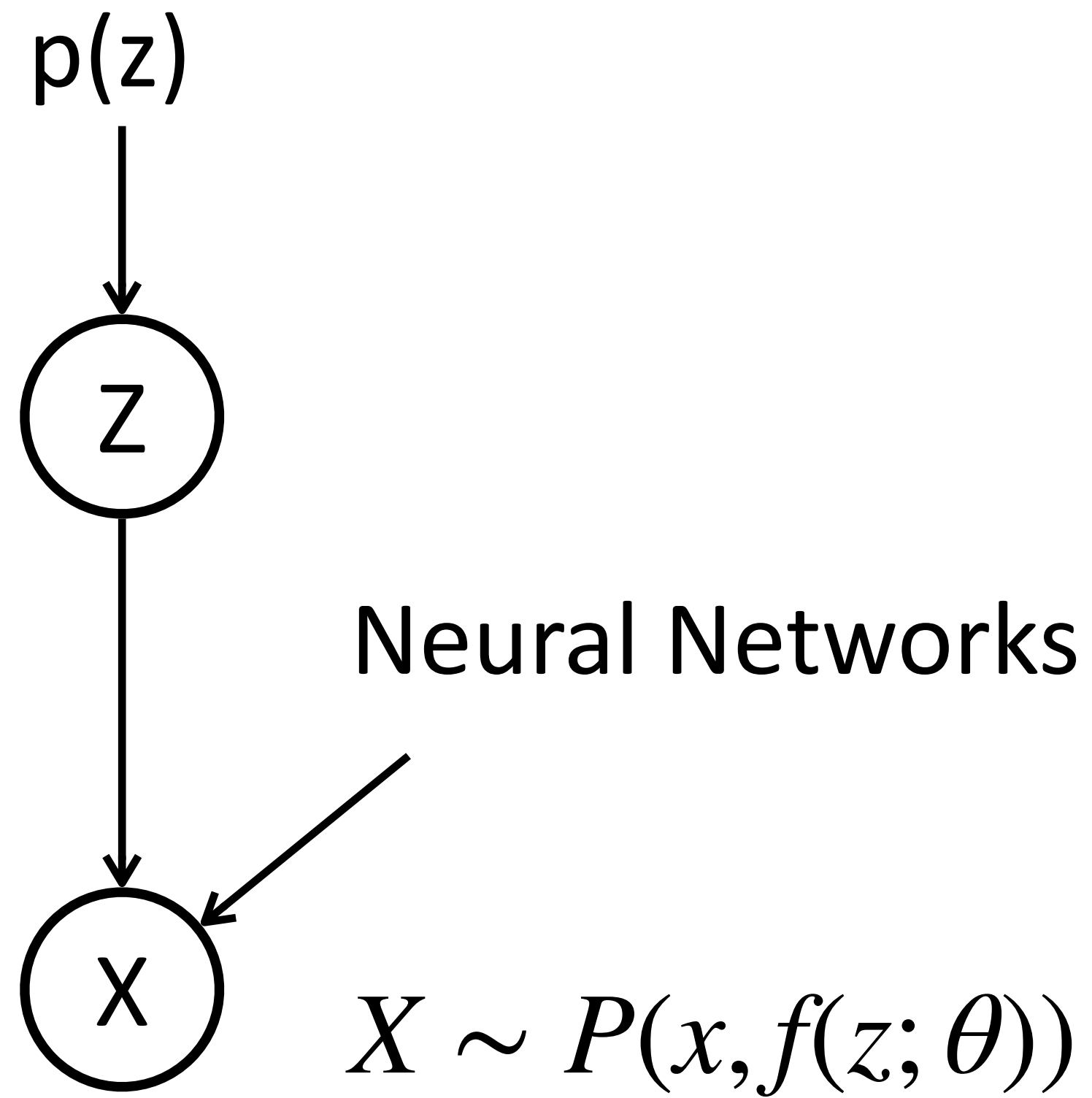
How to train the model? Can we do MLE?

Intractable $P(X)$, EM algorithm?

Let's try EM



Let's try EM



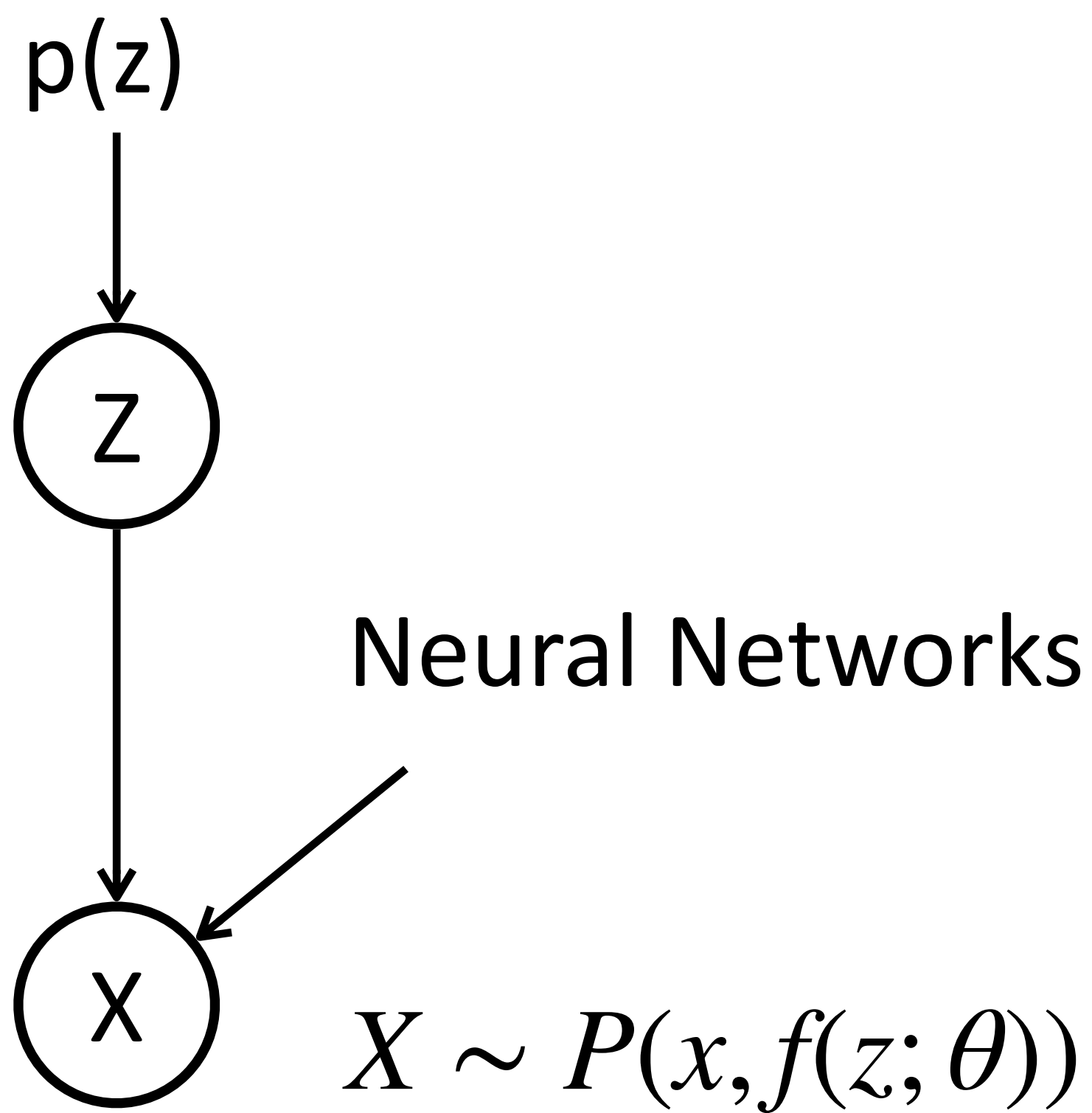
E-Step: compute $P(z|x)$

$$Q(z) = P(z|x) \propto P(z)P(x|z)$$

*known
N(0,1)*

N(z, \sigma^2)

Let's try EM



E-Step: compute $P(z|x)$

$$Q(z) = P(z|x) \propto P(z)P(x|z) \quad \text{This is ok?}$$

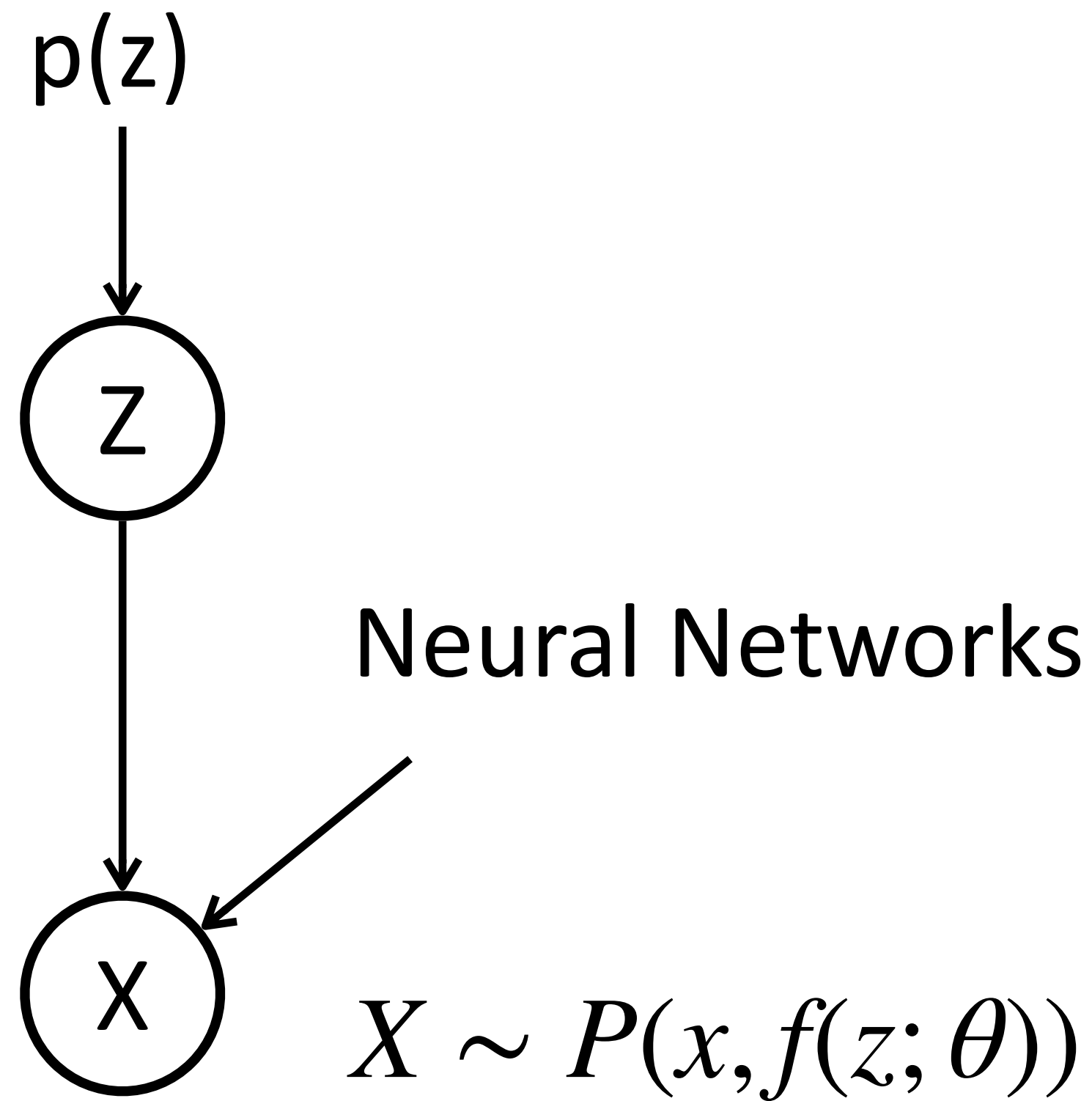
$N(0, 1)$

$N(\mu = f(z); \sigma^2)$

$$\frac{d}{dz} \log P(x|f(z))$$

$$f(z) = z^3 + z^2 + z + 1$$

Let's try EM



E-Step: compute $P(z|x)$

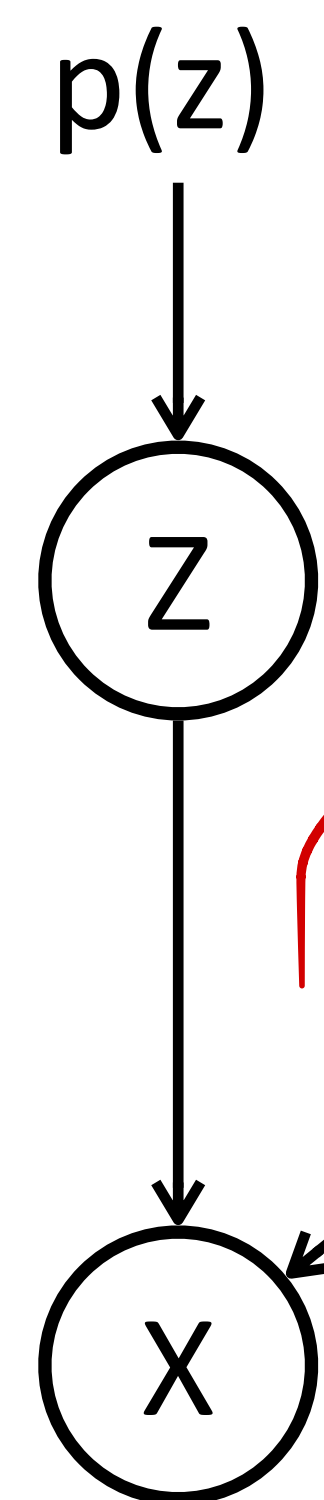
$$Q(z) = P(z|x) \propto P(z)P(x|z) \quad \text{This is ok?}$$

M-Step: the ELBO objective

$$\text{argmax}_{\theta} \sum_z Q(z) \log p(x, z; \theta) = \text{argmax}_{\theta} \mathbb{E}_{z \sim Q(z)} \log p(x, z; \theta)$$

Sampling $z \sim Q(z)$

Let's try EM



Neural Networks

$$X \sim P(x, f(z; \theta))$$

E-Step: compute $P(z|x)$

$$Q(z) = P(z|x) \propto P(z)P(x|z)$$

conjugate prior

$P(x|z)$

This is ok?

Exponential

M-Step: the ELBO objective

$$\operatorname{argmax}_{\theta} \sum_z Q(z) \log p(x, z; \theta) = \operatorname{argmax}_{\theta} \mathbb{E}_{z \sim Q(z)} \log p(x, z; \theta)$$

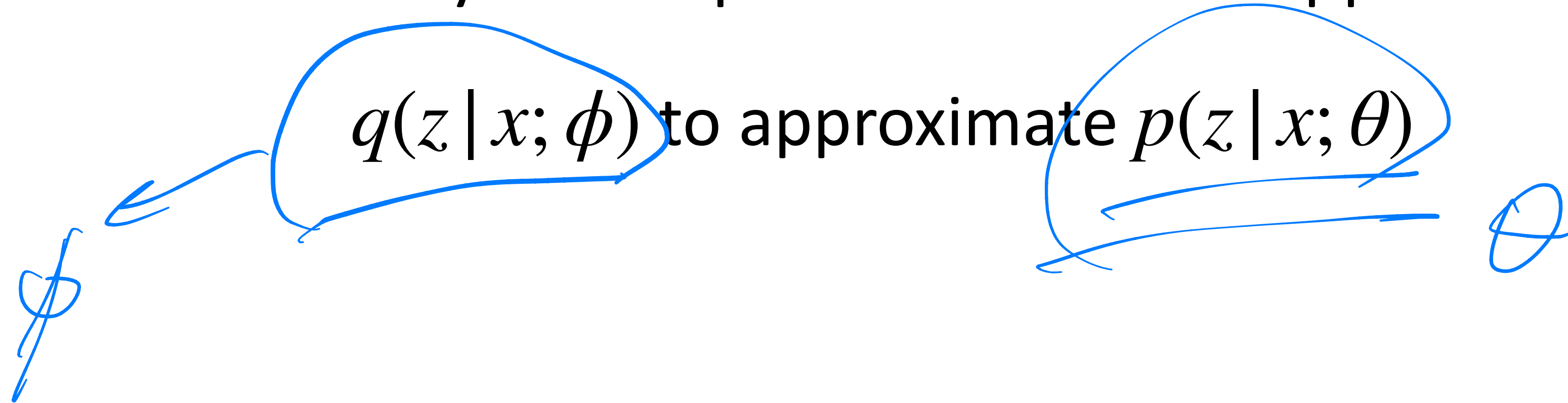
In most cases, we cannot do the sum, and cannot easily sample from $Q(z)$ either

Approximate Posterior

We need an easy-to-sample distribution to approximate $P(z|x)$

Approximate Posterior

We need an easy-to-sample distribution to approximate $P(z|x)$



Approximate Posterior

We need an easy-to-sample distribution to approximate $P(z|x)$

$q(z|x; \phi)$ to approximate $p(z|x; \theta)$

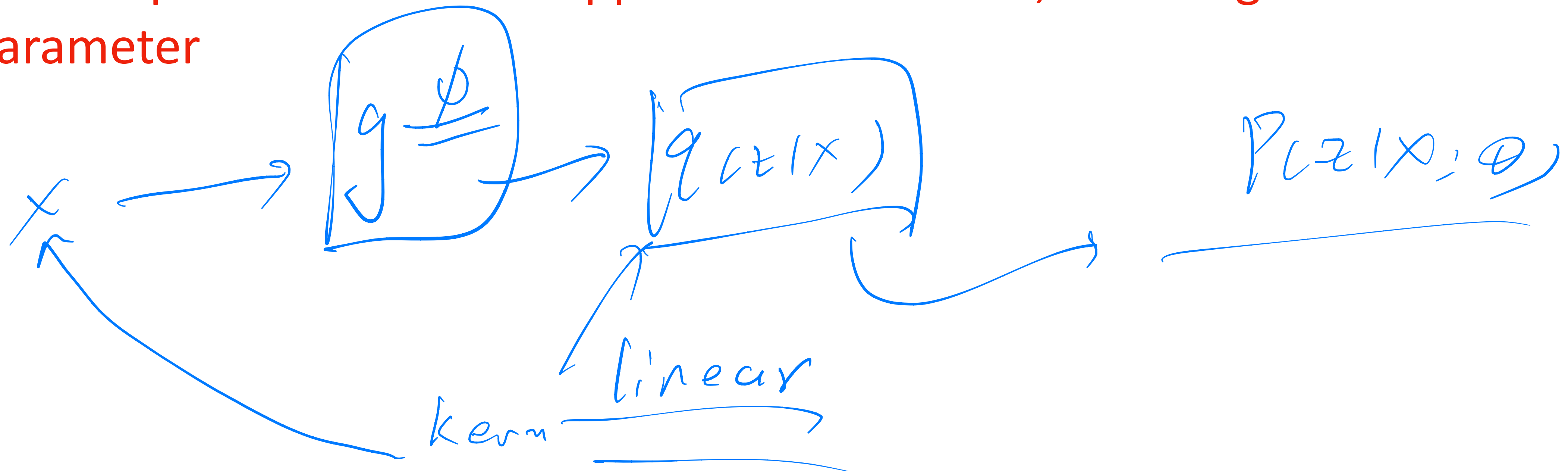
Why conditioned on x ?

Approximate Posterior

We need an easy-to-sample distribution to approximate $P(z|x)$

$q(z|x; \phi)$ to approximate $p(z|x; \theta)$ Why conditioned on x ?

ϕ is the parameter for the approximate function, θ is the generative model parameter



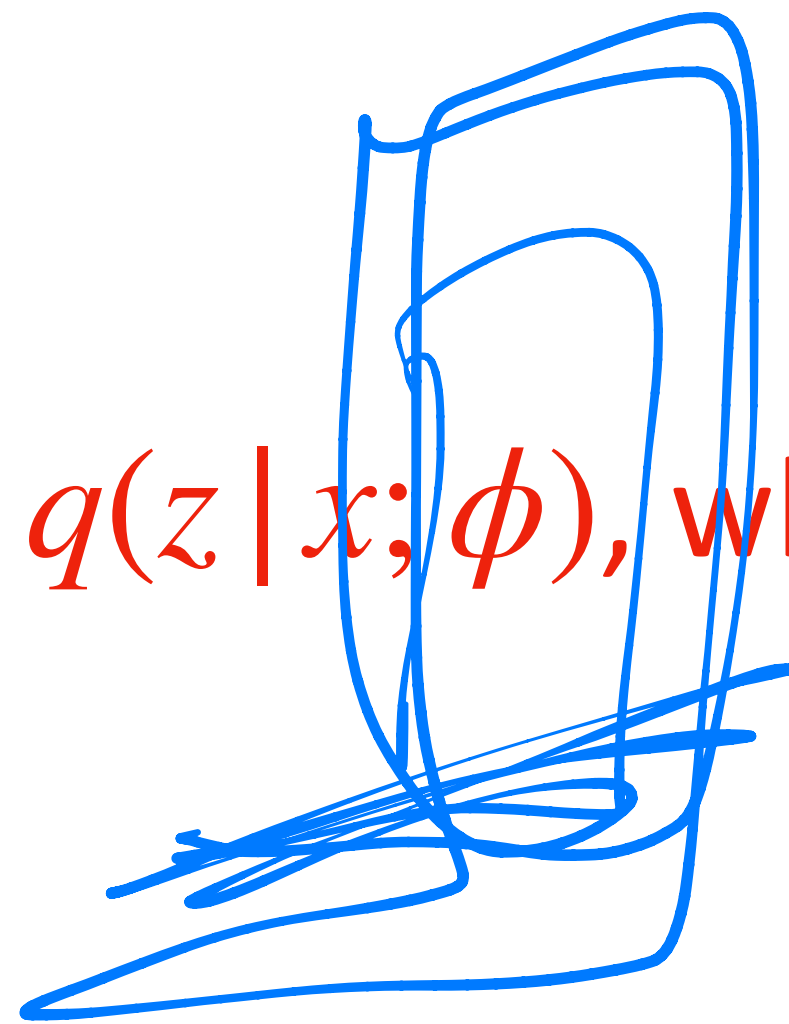
Approximate Posterior

We need an easy-to-sample distribution to approximate $P(z|x)$

$q(z|x; \phi)$ to approximate $p(z|x; \theta)$ Why conditioned on x ?

ϕ is the parameter for the approximate function, θ is the generative model parameter

How to train $q(z|x; \phi)$, what would be the loss to find ϕ ?



\bar{E} step: θ is fixed
KL

Recap: ELBO

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)}$$

What is $\text{argmax}_{Q(z)} \text{ELBO}(x; Q, \theta)$?

$p(z|x; \theta)$

Recap: ELBO

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)}$$

What is $\text{argmax}_{Q(z)} \text{ELBO}(x; Q, \theta)$?

ELBO is maximized when $Q(z)$ is equal to $p(z|x)$

Recap: ELBO

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)}$$

What is $\text{argmax}_{Q(z)} \text{ELBO}(x; Q, \theta)$?

$q(z|x; \phi) \rightarrow p(z|x)$

ELBO is maximized when $Q(z)$ is equal to $p(z|x)$

Therefore, we can approximate the true posterior by maximizing ELBO:

$$\text{argmax}_{\phi} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

$$ELBO = \log p(x) - KL(q(z) \parallel p(z|x))$$

not related to ϕ

$$ELBO = \log p(x) - KL(q(z|x; \phi) \parallel p(z|x))$$

$$\arg \max_{\phi} ELBO = \arg \min_{\phi} KL(q(z|x; \phi) \parallel p(z|x))$$

Recap: ELBO

$$\text{ELBO}(x; Q, \theta) = \sum_z Q(z) \log \frac{p(x, z; \theta)}{Q(z)}$$

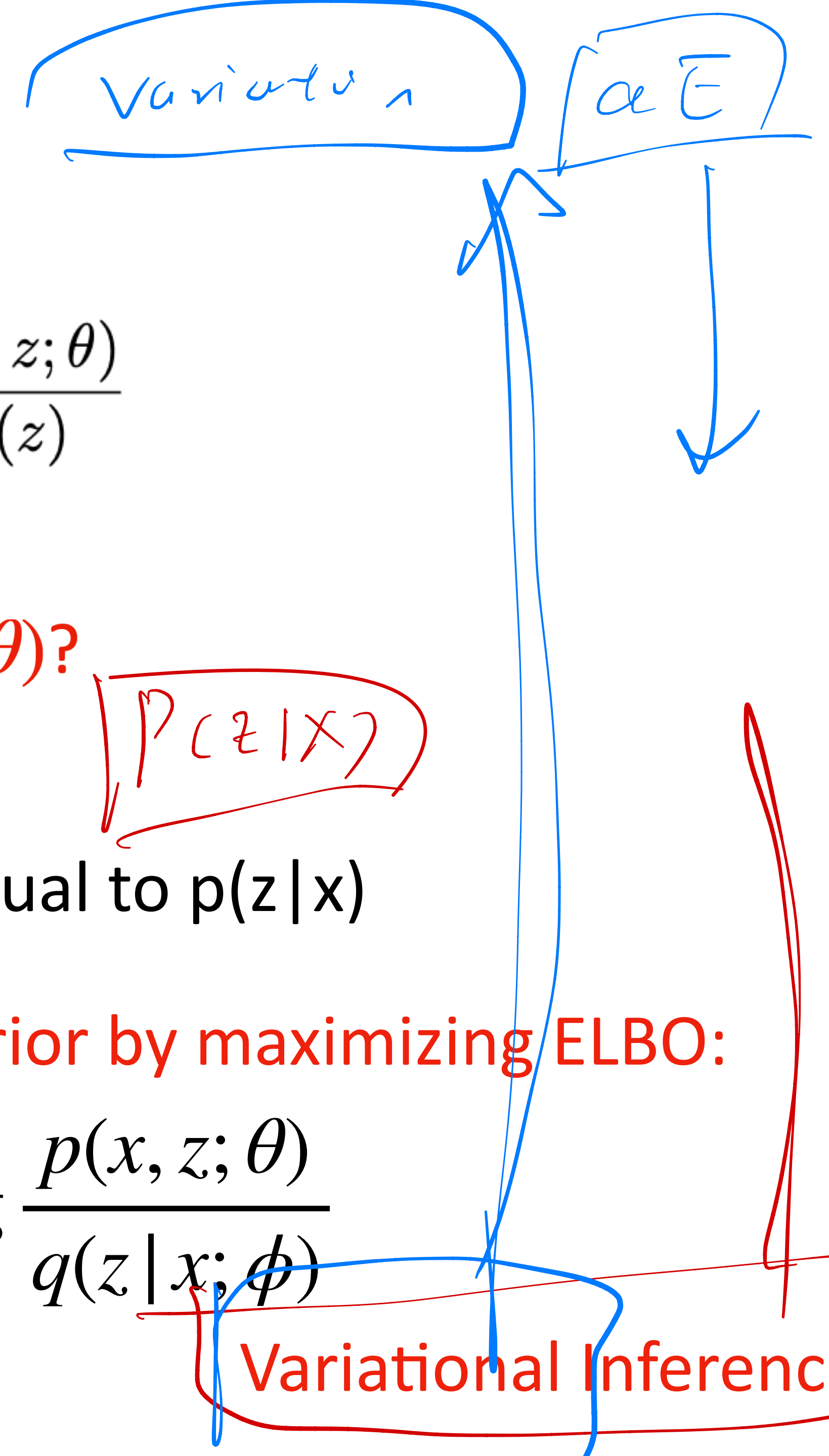
What is $\text{argmax}_{Q(z)} \text{ELBO}(x; Q, \theta)$?

ELBO is maximized when $Q(z)$ is equal to $p(z|x)$

Therefore, we can approximate the true posterior by maximizing ELBO:

$$\text{argmax}_{\phi} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

Variational Inference



Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Same objective, different parameters to optimize

Training VAEs

derivative

$\partial z(\phi)$

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

Same objective, different parameters to optimize

Because we use approximate rather than exact posterior, it is also called Variational EM

$$E_{z \sim q(z|x; \phi)} \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

$$z \sim q(z|x; \phi)$$

$$z^{(i)} \sim q(z|x; \phi)$$

A Common Choice for $q(z | x; \phi)$

A Common Choice for $q(z|x; \phi)$

$$q(z|x; \phi) = N(\mu, \sigma^2) \text{ easy to sample from}$$

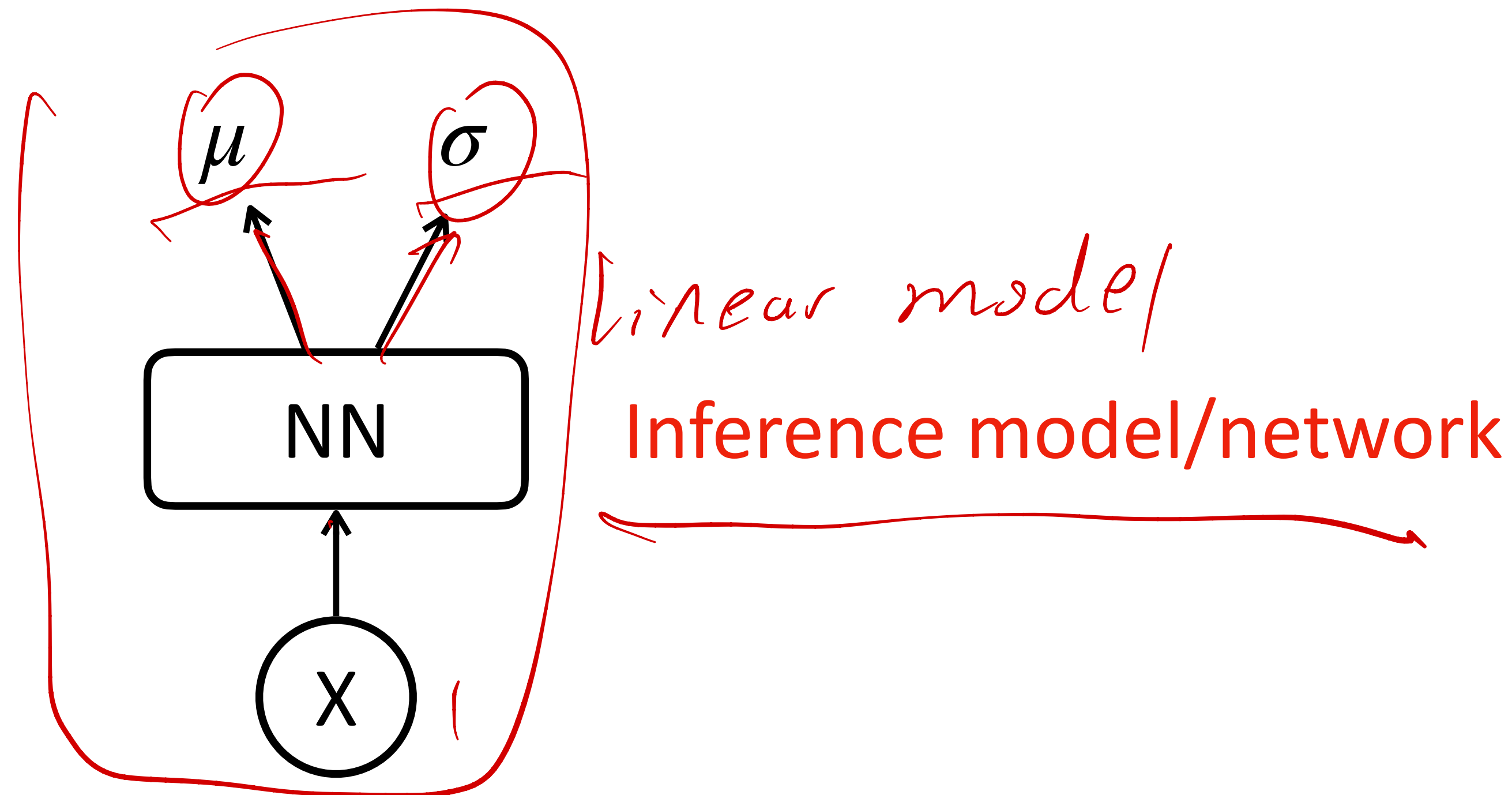
$$\mu, \sigma = g(x; \phi) \quad \text{NN}$$

A Common Choice for $q(z | x; \phi)$

$$q(z | x; \phi) = N(\mu, \sigma^2)$$

sample from

$$\mu, \sigma = g(x; \phi)$$



Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Same objective, different parameters to optimize

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Same objective, different parameters to optimize

Because we use approximate rather than exact posterior, it is also called Variational EM

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

We use MC sampling to approximate expectation and use gradient descent to optimize θ

Training VAEs

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

Can we do gradient descent over ϕ ?

M-Step:

$$\operatorname{argmax}_{\theta} \sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)}$$

No

We use MC sampling to approximate expectation and use gradient descent to optimize θ

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

First, we cannot do sum, but we can sample z_i from $q(z | x; \phi)$, which depends on ϕ , how do we propagate gradients to ϕ ?

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

First, we cannot do sum, but we can sample z_i from $q(z | x; \phi)$, which depends on ϕ , how do we propagate gradients to ϕ ?

Try to express z as a deterministic function $z = g_{\phi}(\epsilon, x)$, where ϵ is an auxiliary random variable

$$x = f(\mu, \sigma^2, \epsilon)$$
$$x \sim \mathcal{N}(\mu, \sigma^2)$$

$$\epsilon \sim \mathcal{N}(0, 1)$$

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

First, we cannot do sum, but we can sample z_i from $q(z | x; \phi)$, which depends on ϕ , how do we propagate gradients to ϕ ?

Try to express z as a deterministic function $z = g_{\phi}(\epsilon, x)$, where ϵ is an auxiliary random variable

$$z \sim N(\mu, \sigma^2) \longrightarrow z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim N(0, 1)$$

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

First, we cannot do sum, but we can sample z_i from $q(z | x; \phi)$, which depends on ϕ , how do we propagate gradients to ϕ ?

Try to express z as a deterministic function $z = g_{\phi}(\epsilon, x)$, where ϵ is an auxiliary random variable

$$z \sim N(\mu, \sigma^2) \longrightarrow z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim N(0, 1)$$

Can you verify z in this equation is Gaussian?

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

For every gradient step (assuming batch size=1):

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

For every gradient step (assuming batch size=1):

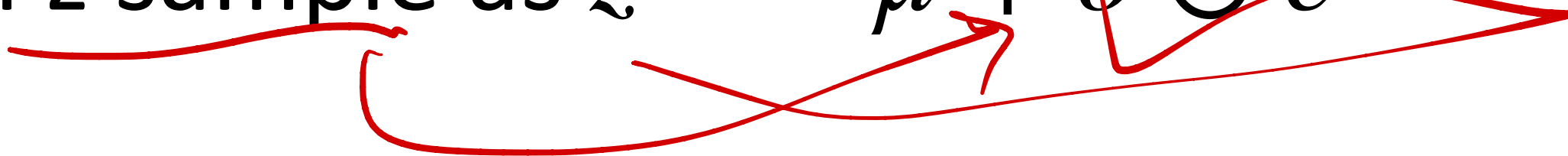
1. Randomly sample $\epsilon^{(i)} \sim N(0,1)$

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

For every gradient step (assuming batch size=1):

1. Randomly sample $\epsilon^{(i)} \sim N(0,1)$
 2. Obtain z sample as $z^{(i)} = \mu + \sigma \odot \epsilon^{(i)}$
- 

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

For every gradient step (assuming batch size=1):

1. Randomly sample $\epsilon^{(i)} \sim N(0,1)$
2. Obtain z sample as $z^{(i)} = \mu + \sigma \odot \epsilon^{(i)}$
3. Perform gradient descent w.r.t. $\log \frac{p(x, z^{(i)}; \theta)}{q(z^{(i)} | x; \phi)}$

Reparameterization Trick

E-Step:

$$\operatorname{argmax}_{\phi} \sum_z q(z | x; \phi) \log \frac{p(x, z; \theta)}{q(z | x; \phi)}$$

For every gradient step (assuming batch size=1):

1. Randomly sample $\epsilon^{(i)} \sim N(0,1)$
2. Obtain z sample as $z^{(i)} = \mu + \sigma \odot \epsilon^{(i)}$
3. Perform gradient descent w.r.t. $\log \frac{p(x, z^{(i)}; \theta)}{q(z^{(i)} | x; \phi)}$

We can now propagate gradients from z to ϕ

Reparameterization Trick

VAE is a class of models

What kind of $q(z | x; \phi)$ allows for such a reparameterization trick?

Reparameterization Trick

VAE is a class of models

$\mathcal{N}(\cdot)$

What kind of $q(z | x; \phi)$ allows for such a reparameterization trick?

1. Tractable inverse CDF. In this case, let $\epsilon \sim \mathcal{U}(\mathbf{0}, \mathbf{I})$, and let $g_\phi(\epsilon, \mathbf{x})$ be the inverse CDF of $q_\phi(\mathbf{z} | \mathbf{x})$. Examples: Exponential, Cauchy, Logistic, Rayleigh, Pareto, Weibull, Reciprocal, Gompertz, Gumbel and Erlang distributions.
2. Analogous to the Gaussian example, for any "location-scale" family of distributions we can choose the standard distribution (with location = 0, scale = 1) as the auxiliary variable ϵ , and let $g(\cdot) = \text{location} + \text{scale} \cdot \epsilon$. Examples: Laplace, Elliptical, Student's t, Logistic, Uniform, Triangular and Gaussian distributions.
3. Composition: It is often possible to express random variables as different transformations of auxiliary variables. Examples: Log-Normal (exponentiation of normally distributed variable), Gamma (a sum over exponentially distributed variables), Dirichlet (weighted sum of Gamma variates), Beta, Chi-Squared, and F distributions.

ELBO

$$\sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)} = \mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)]$$

ELBO

$$\mathbb{E}_{z \sim q(z|x)} \log \frac{p(x, z)}{p(z)} = [\quad]$$

$$\sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)} = \mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)]$$

ELBO is implemented with the following form:

max

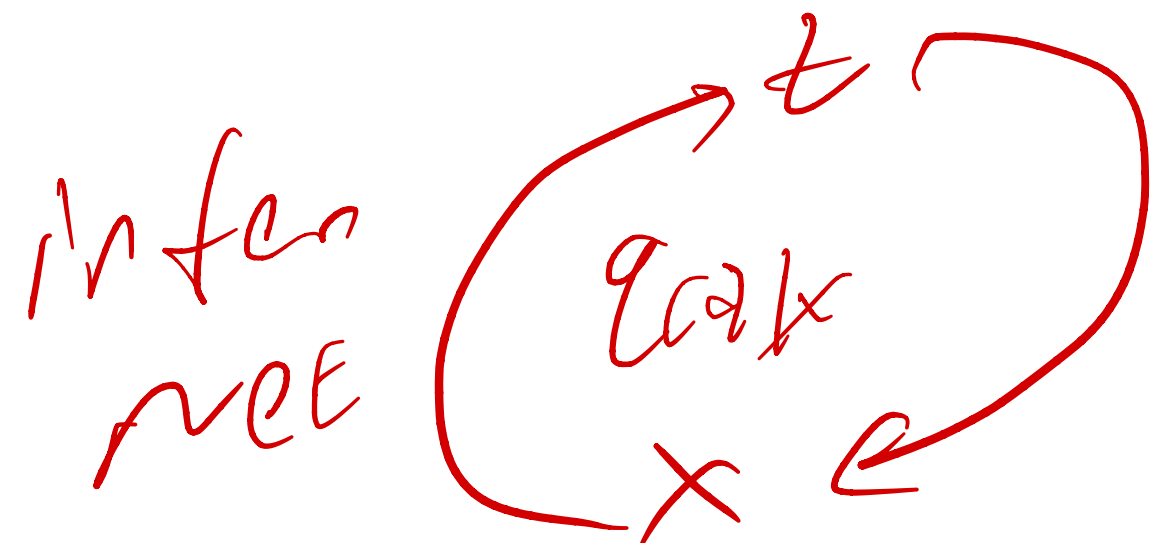
$$\mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(\mathbf{x}|z)]$$

Reconstruction Loss

$$D_{\text{KL}}(q_\phi(z|x) || p(z))$$

KL Regularizer

ELBO



close-form

ELBO

$$\sum_z q(z|x; \phi) \log \frac{p(x, z; \theta)}{q(z|x; \phi)} = \mathbb{E}_{z \sim q_\phi(z|x)} [\log p_\theta(x, z) - \log q_\phi(z|x)]$$

ELBO is implemented with the following form:

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

Autoencoder

ELBO

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

ELBO

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

Autoencoder Loss

ELBO

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

Autoencoder Loss

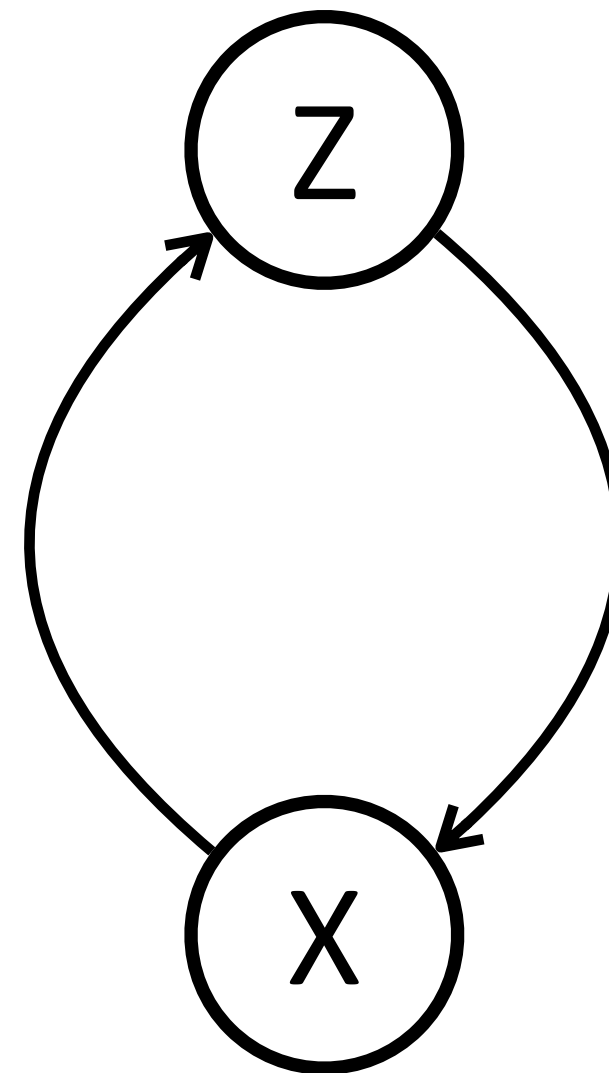
$q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ are both Gaussian,
there is a closed-form for this

ELBO

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

Autoencoder Loss

$q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ are both Gaussian,
there is a closed-form for this

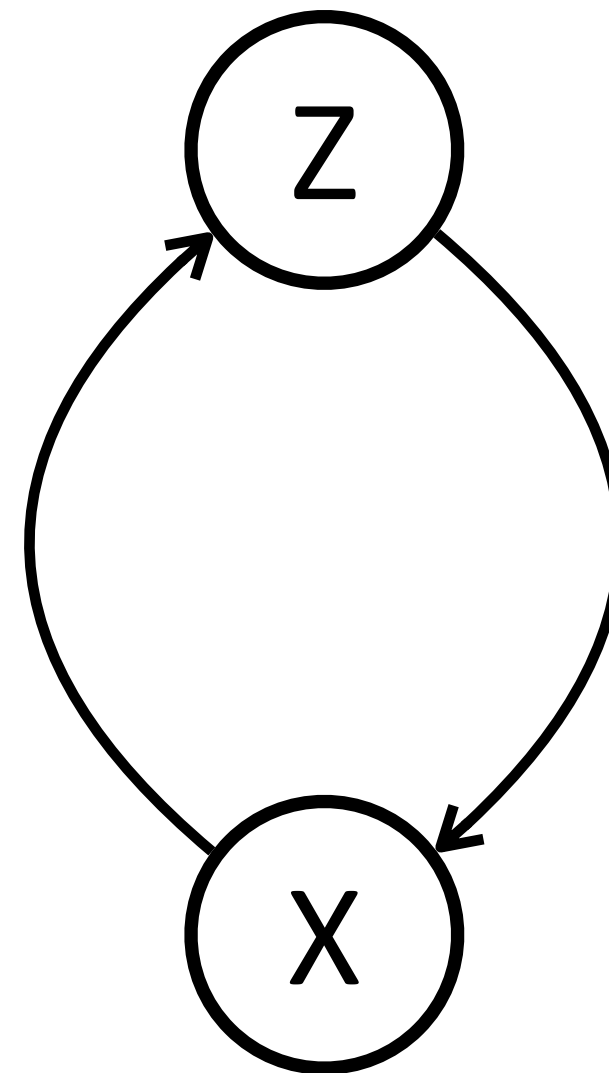


ELBO

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

Autoencoder Loss

$q(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$ are both Gaussian,
there is a closed-form for this



This is why it is called variational “autoencoder”

ELBO

$$D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

ELBO

$$D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).$$

$$\begin{aligned} \int q_{\theta}(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 + \sigma_j^2) \end{aligned}$$

J is the dimensionality of z

ELBO

$$D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).$$

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 + \sigma_j^2)\end{aligned}$$

J is the dimensionality of z

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log q_{\theta}(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (1 + \log \sigma_j^2)\end{aligned}$$

ELBO

$$D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$$

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log p(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I}) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (\mu_j^2 + \sigma_j^2)\end{aligned}$$

J is the dimensionality of z

$$\begin{aligned}\int q_{\theta}(\mathbf{z}) \log q_{\theta}(\mathbf{z}) d\mathbf{z} &= \int \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) \log \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2) d\mathbf{z} \\ &= -\frac{J}{2} \log(2\pi) - \frac{1}{2} \sum_{j=1}^J (1 + \log \sigma_j^2)\end{aligned}$$

$$\begin{aligned}-D_{\text{KL}}(q_{\phi}(\mathbf{z})||p_{\theta}(\mathbf{z})) &= \int q_{\theta}(\mathbf{z}) (\log p_{\theta}(\mathbf{z}) - \log q_{\theta}(\mathbf{z})) d\mathbf{z} \\ &= \frac{1}{2} \sum_{j=1}^J (1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2)\end{aligned}$$

Training VAEs

E-Step:

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

M-Step:

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

Training VAEs

E-Step:

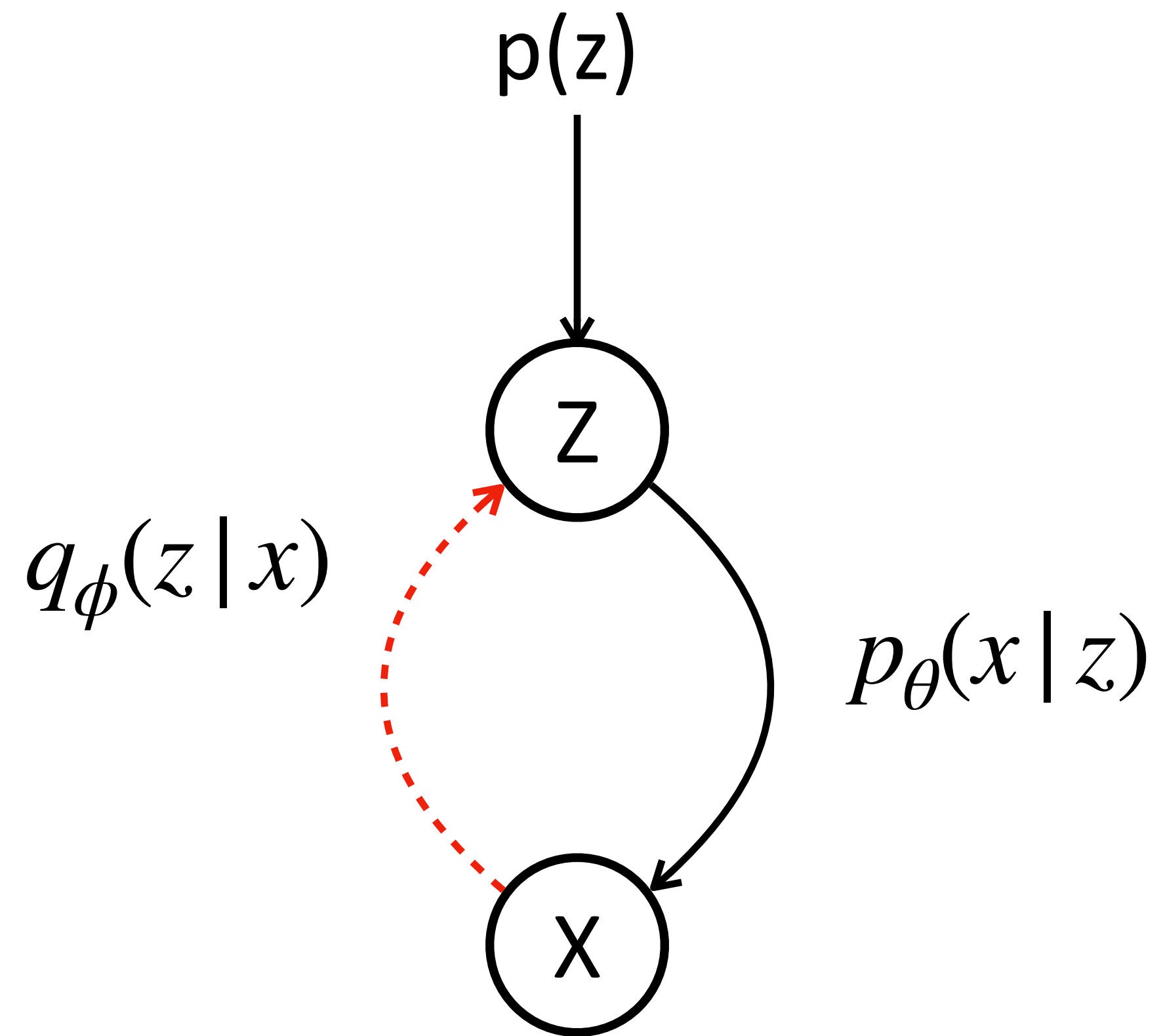
$$\operatorname{argmax}_{\phi} \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}))}_{\text{KL Regularizer}}$$

M-Step:

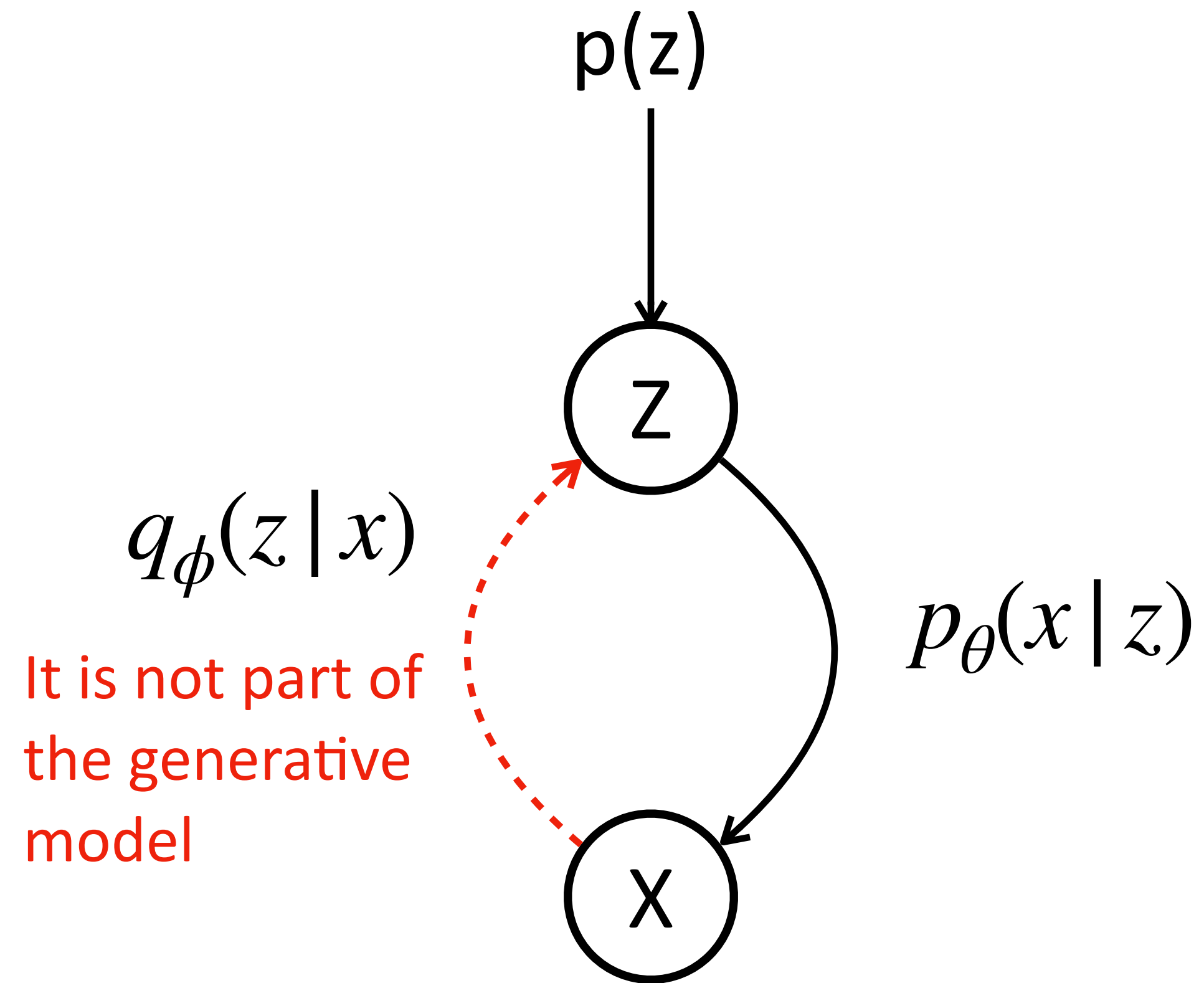
$$\operatorname{argmax}_{\theta} \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}))}_{\text{KL Regularizer}}$$

Intuitively we hope to approximate $p(\mathbf{z}|\mathbf{x})$ with $q(\mathbf{z}|\mathbf{x})$ accurately in the E-step, to approximate the true EM algorithm

Review VAE

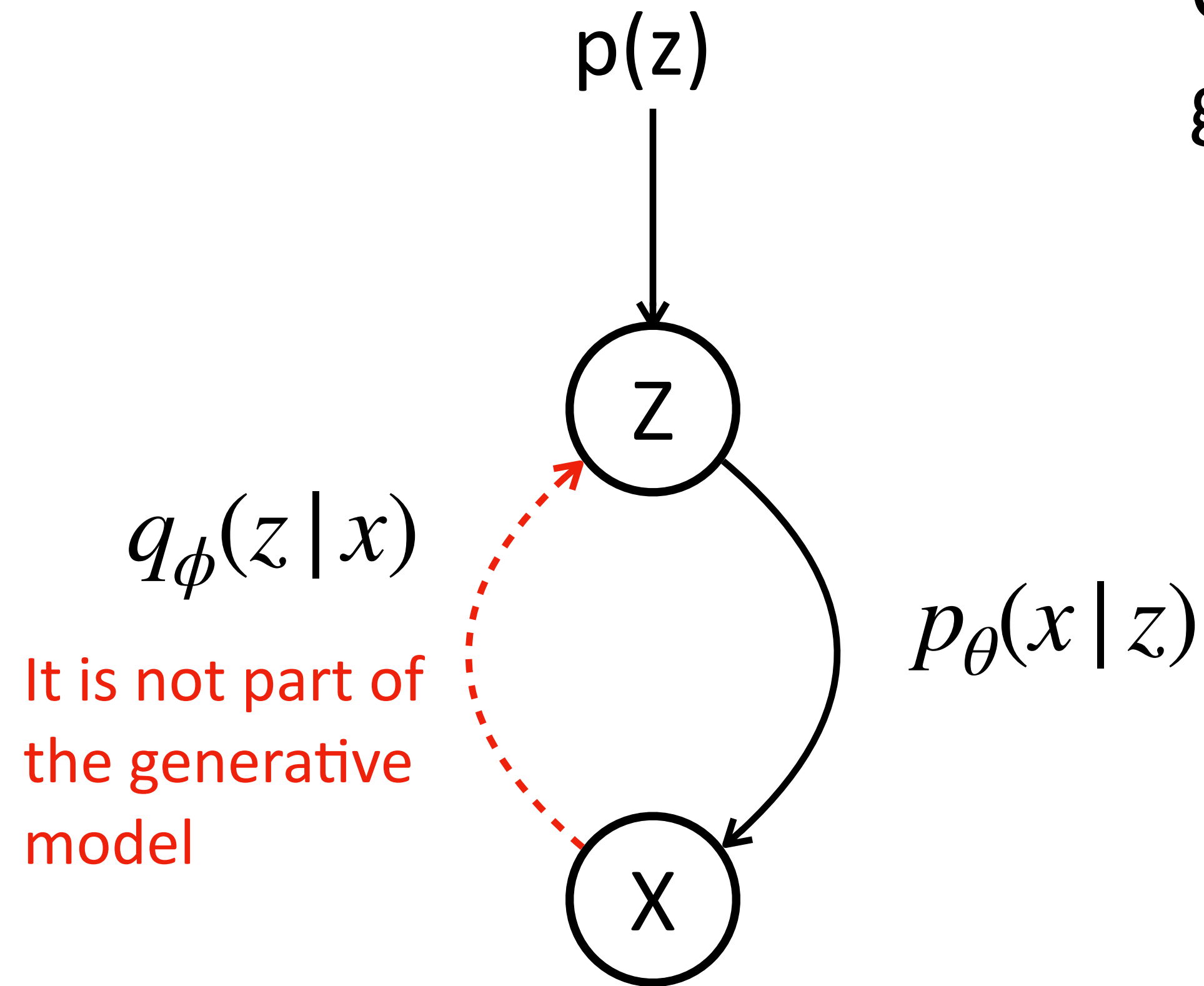


Review VAE



Review VAE

Only the right (black) part defines the generative model, and the distribution

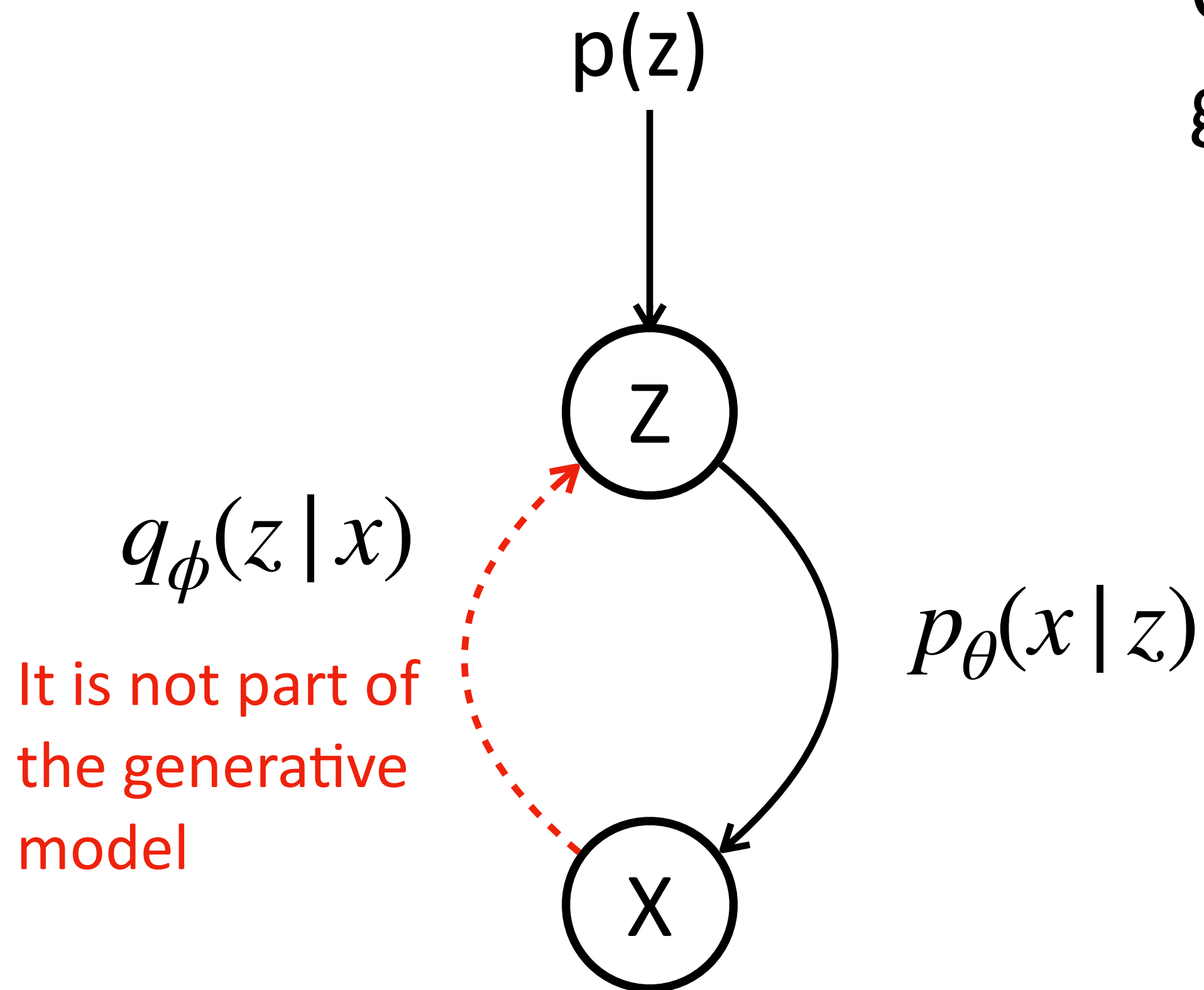


Review VAE

Only the right (black) part defines the generative model, and the distribution

$p_{\theta}(x | z)$: generative network/decoder

$q_{\phi}(z | x)$: inference network/encoder

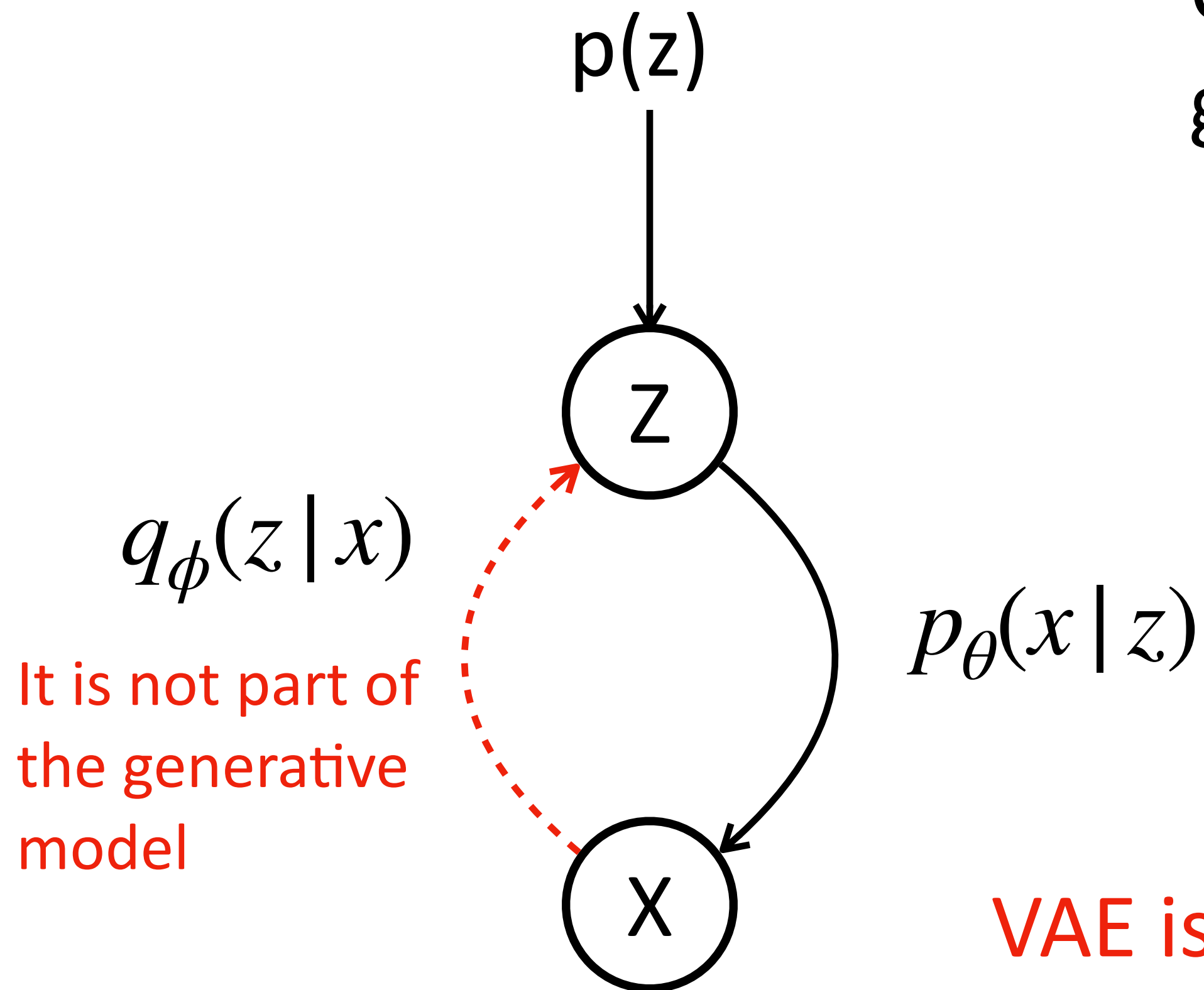


Review VAE

Only the right (black) part defines the generative model, and the distribution

$p_{\theta}(x | z)$: generative network/decoder

$q_{\phi}(z | x)$: inference network/encoder



VAE is a name to represent both the model $p(x)$ and the inference network that is used to train the model, but do not confuse them together

Training VAEs

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\theta, \phi \leftarrow$ Initialize parameters

repeat

$\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)

$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$ (Gradients of minibatch estimator (8))

$\theta, \phi \leftarrow$ Update parameters using gradients \mathbf{g} (e.g. SGD or Adagrad [DHS10])

until convergence of parameters (θ, ϕ)

return θ, ϕ

Training VAEs

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\theta, \phi \leftarrow$ Initialize parameters

repeat

$\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)

$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$ (Gradients of minibatch estimator (8))

$\theta, \phi \leftarrow$ Update parameters using gradients \mathbf{g} (e.g. SGD or Adagrad [DHS10])

until convergence of parameters (θ, ϕ)

return θ, ϕ

Training VAEs

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\theta, \phi \leftarrow$ Initialize parameters

repeat

$\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)

$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$ (Gradients of minibatch estimator (8))

$\theta, \phi \leftarrow$ Update parameters using gradients \mathbf{g} (e.g. SGD or Adagrad [DHS10])

until convergence of parameters (θ, ϕ)

return θ, ϕ

End-to-end, because the objectives are the same (ELBO)

Training VAEs

Algorithm 1 Minibatch version of the Auto-Encoding VB (AEVB) algorithm. Either of the two SGVB estimators in section 2.3 can be used. We use settings $M = 100$ and $L = 1$ in experiments.

$\theta, \phi \leftarrow$ Initialize parameters

repeat

$\mathbf{X}^M \leftarrow$ Random minibatch of M datapoints (drawn from full dataset)

$\epsilon \leftarrow$ Random samples from noise distribution $p(\epsilon)$

$\mathbf{g} \leftarrow \nabla_{\theta, \phi} \tilde{\mathcal{L}}^M(\theta, \phi; \mathbf{X}^M, \epsilon)$ (Gradients of minibatch estimator (8))

$\theta, \phi \leftarrow$ Update parameters using gradients \mathbf{g} (e.g. SGD or Adagrad [DHS10])

until convergence of parameters (θ, ϕ)

return θ, ϕ

End-to-end, because the objectives are the same (ELBO)

VAE training is optimizing ELBO with gradient descent

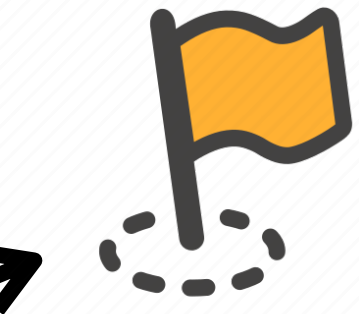
Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



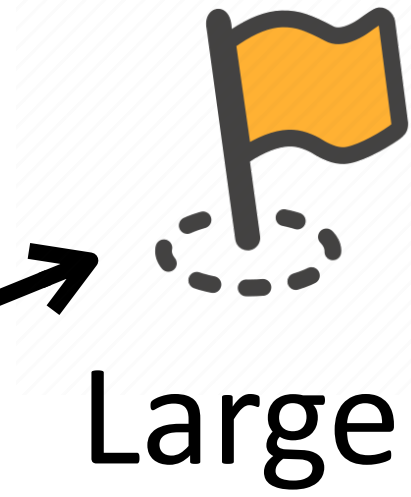
Larger

Recap: EM is Hill Climbing



$\log p(x; \theta)$

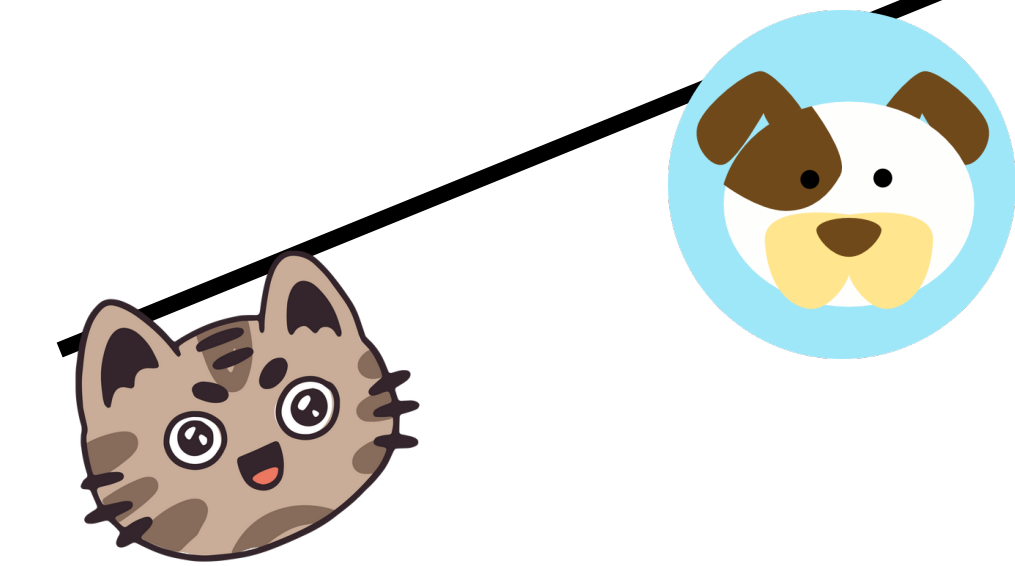
Only related to θ , no z



Larger



ELBO



Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



Larger

Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



E-step: $Q(z) = p(z | x; \theta)$, making ELBO tight



Larger

Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



Larger

E-step: $Q(z) = p(z | x; \theta)$, making ELBO tight
“dog” doesn’t change, because θ does not change

Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



Larger

E-step: $Q(z) = p(z | x; \theta)$, making ELBO tight
“dog” doesn’t change, because θ does not change

Recap: EM is Hill Climbing



$\log p(x; \theta)$

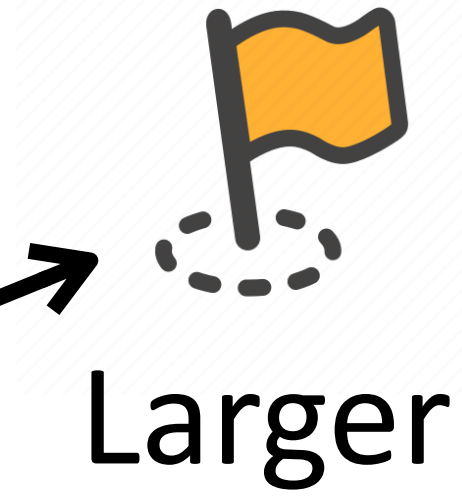


ELBO



M-step: $\max_{\theta} ELBO$

ELBO becomes larger, and it is not tight anymore because posterior changes



Larger

Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



M-step: $\max_{\theta} ELBO$



Larger

ELBO becomes larger, and it is not tight anymore because posterior changes

Recap: EM is Hill Climbing



$\log p(x; \theta)$



ELBO



M-step: $\max_{\theta} ELBO$



Larger

ELBO becomes larger, and it is not tight anymore because posterior changes

Is VAE training still Hill Climbing?

Is VAE training still Hill Climbing?

It is not, because $q(z|x)$ may not be accurate to approximate $p(z|x)$

Is VAE training still Hill Climbing?

It is not, because $q(z|x)$ may not be accurate to approximate $p(z|x)$

In VAE training, there is no guarantee that $\log p(x)$ is monotonically increasing

Is VAE training still Hill Climbing?

It is not, because $q(z|x)$ may not be accurate to approximate $p(z|x)$

In VAE training, there is no guarantee that $\log p(x)$ is monotonically increasing

E-Step:

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

According to EM, ϕ should be optimized to convergence to have a good approximation for $p(z|x)$ before conducting the M-step, but VAE does not

Is VAE training still Hill Climbing?

It is not, because $q(z|x)$ may not be accurate to approximate $p(z|x)$

In VAE training, there is no guarantee that $\log p(x)$ is monotonically increasing

It just works in many cases

E-Step:

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

According to EM, ϕ should be optimized to convergence to have a good approximation for $p(z|x)$ before conducting the M-step, but VAE does not

The Posterior Collapse Issue

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

The Posterior Collapse Issue

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

In practice, it is often found that after training, $q_{\phi}(z|x) = p(z)$ and z and x becomes independent (especially in applications of NLP)

The Posterior Collapse Issue

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

In practice, it is often found that after training, $q_{\phi}(z|x) = p(z)$ and z and x becomes independent (especially in applications of NLP)

Z does not affect x, the model degenerates to a generative model without latent variables

The Posterior Collapse Issue

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

In practice, it is often found that after training, $q_{\phi}(z|x) = p(z)$ and z and x becomes independent (especially in applications of NLP)

Z does not affect x, the model degenerates to a generative model without latent variables

Researchers commonly blame that the KL regularizer is too strong for this and use a weight $0 < \lambda < 1$ to control it:

The Posterior Collapse Issue

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

In practice, it is often found that after training, $q_{\phi}(z|x) = p(z)$ and z and x becomes independent (especially in applications of NLP)

Z does not affect x, the model degenerates to a generative model without latent variables

Researchers commonly blame that the KL regularizer is too strong for this and use a weight $0 < \lambda < 1$ to control it:

Reconstruction Loss - λ * KL regularizer

The Posterior Collapse Issue

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

In practice, it is often found that after training, $q_{\phi}(z|x) = p(z)$ and z and x becomes independent (especially in applications of NLP)

Z does not affect x , the model degenerates to a generative model without latent variables

Researchers commonly blame that the KL regularizer is too strong for this and use a weight $0 < \lambda < 1$ to control it:

Reconstruction Loss - λ * KL regularizer

This is not a lower-bound of $\log p(x)$ anymore and it breaks MLE, but what is wrong with MLE?

Is VAE training still Hill Climbing?

E-Step:

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

According to EM, ϕ should be optimized to convergence to have a good approximation for $p(\mathbf{z}|\mathbf{x})$ before conducting the M-step, but VAE does not

Is VAE training still Hill Climbing?

E-Step:

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

According to EM, ϕ should be optimized to convergence to have a good approximation for $p(\mathbf{z}|\mathbf{x})$ before conducting the M-step, but VAE does not

Can we make it closer to EM to have good guarantees?

VAE training that is Closer to EM

VAE training that is Closer to EM

At every iteration, perform multiple gradient updates of ϕ (E-step) before performing one step of θ (M-step)

VAE training that is Closer to EM

At every iteration, perform multiple gradient updates of ϕ (E-step) before performing one step of θ (M-step)

Published as a conference paper at ICLR 2019

LAGGING INFERENCE NETWORKS AND POSTERIOR COLLAPSE IN VARIATIONAL AUTOENCODERS

Junxian He, Daniel Spokoyny, Graham Neubig
Carnegie Mellon University
{junxianh, dspokoyn, gneubig}@cs.cmu.edu

Taylor Berg-Kirkpatrick
University of California San Diego
tberg@eng.ucsd.edu

AutoEncoders

AutoEncoders

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

AutoEncoders

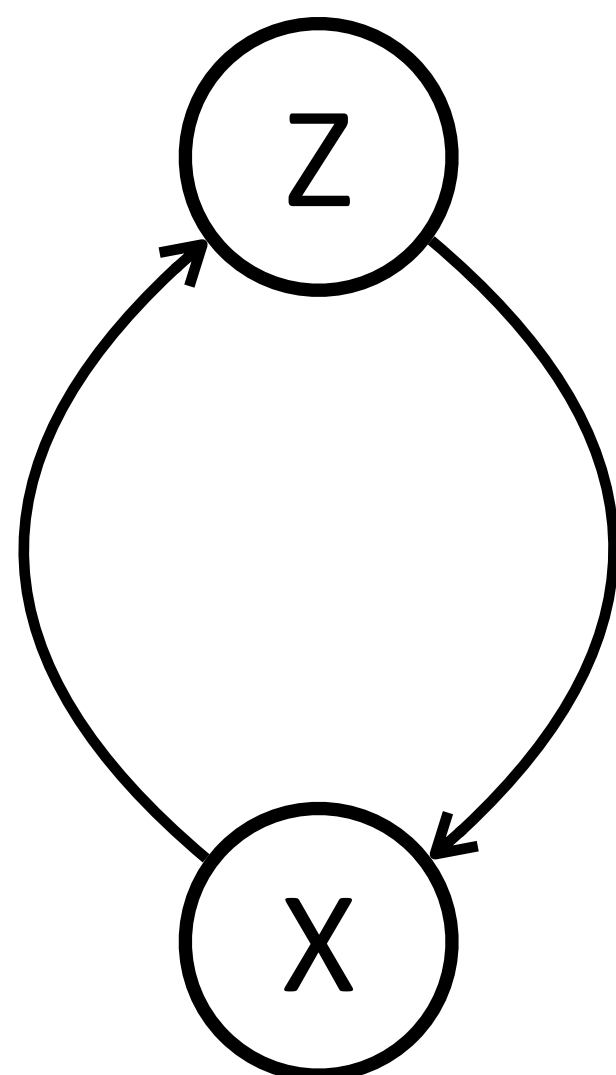
$$\text{VAE: } \underbrace{\mathbb{E}_{\mathbf{z} \sim q_{\phi}(\mathbf{z}|\mathbf{x})} [\log p_{\theta}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} - \underbrace{D_{\text{KL}}(q_{\phi}(\mathbf{z}|\mathbf{x}) || p(\mathbf{z}))}_{\text{KL Regularizer}}$$

$$\text{AE: } \log p_{\theta}(x | q(x))$$

AutoEncoders

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

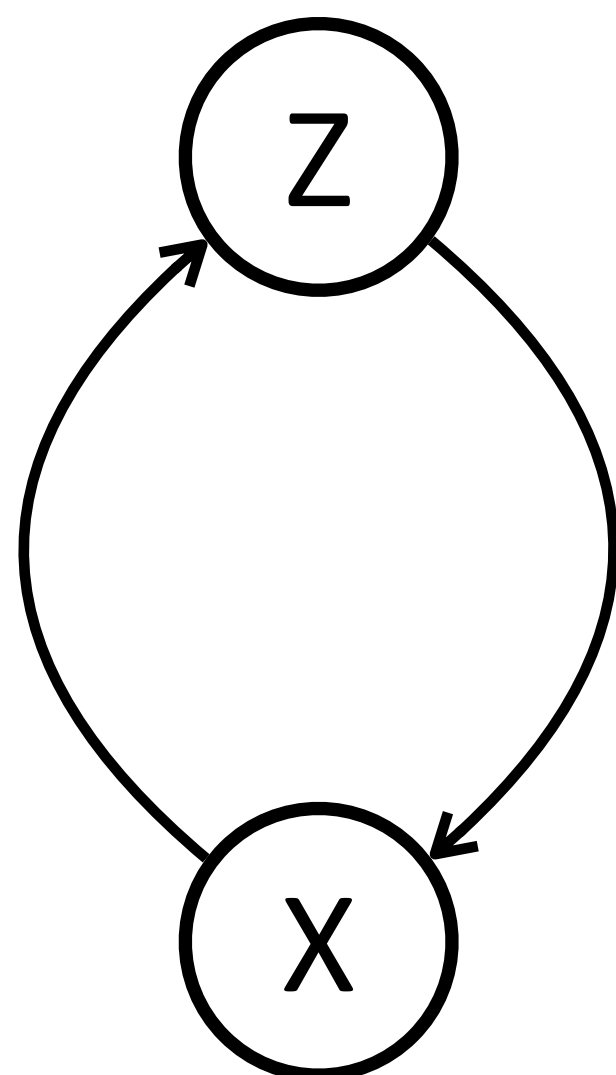
$$\text{AE: } \log p_{\theta}(x | q(x))$$



AutoEncoders

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

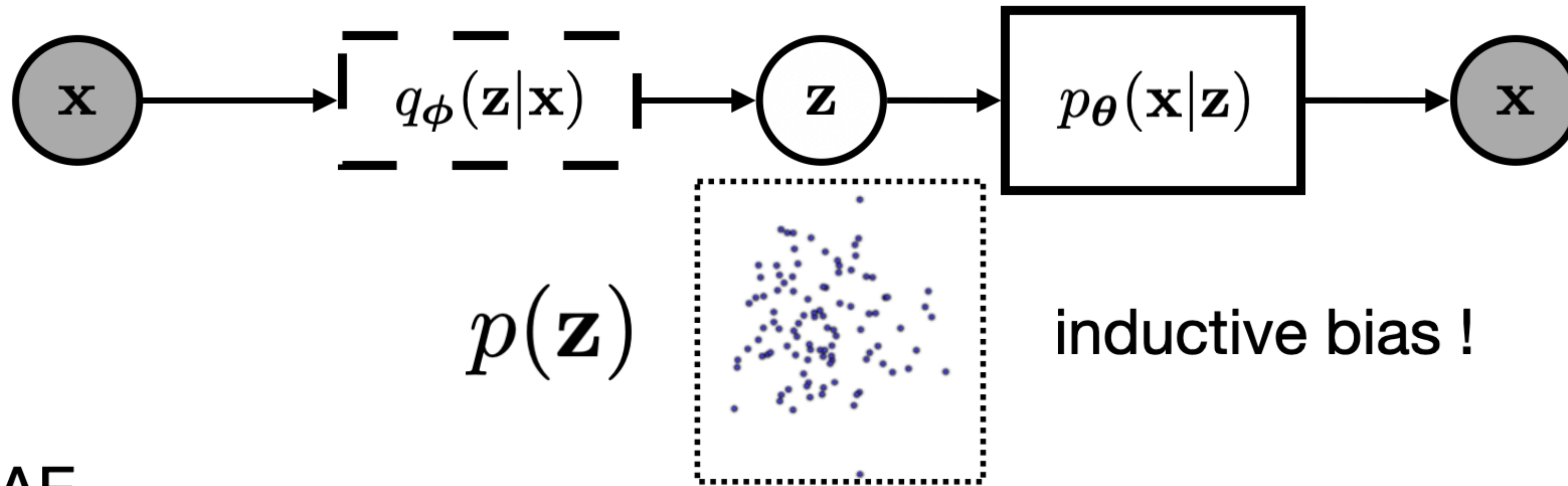
$$\text{AE: } \log p_{\theta}(x | q(x))$$



1. Can we generate X samples from an autoencoder?
2. Can we approximate $p(x)$ given x with an autoencoder?
3. What is the difference between the representation space from AE and VAE?

VAE v.s. AE

VAE



AE

