Variational Autoencoders

Marina Kisley

THE UNIVERSITY OF ARIZONA

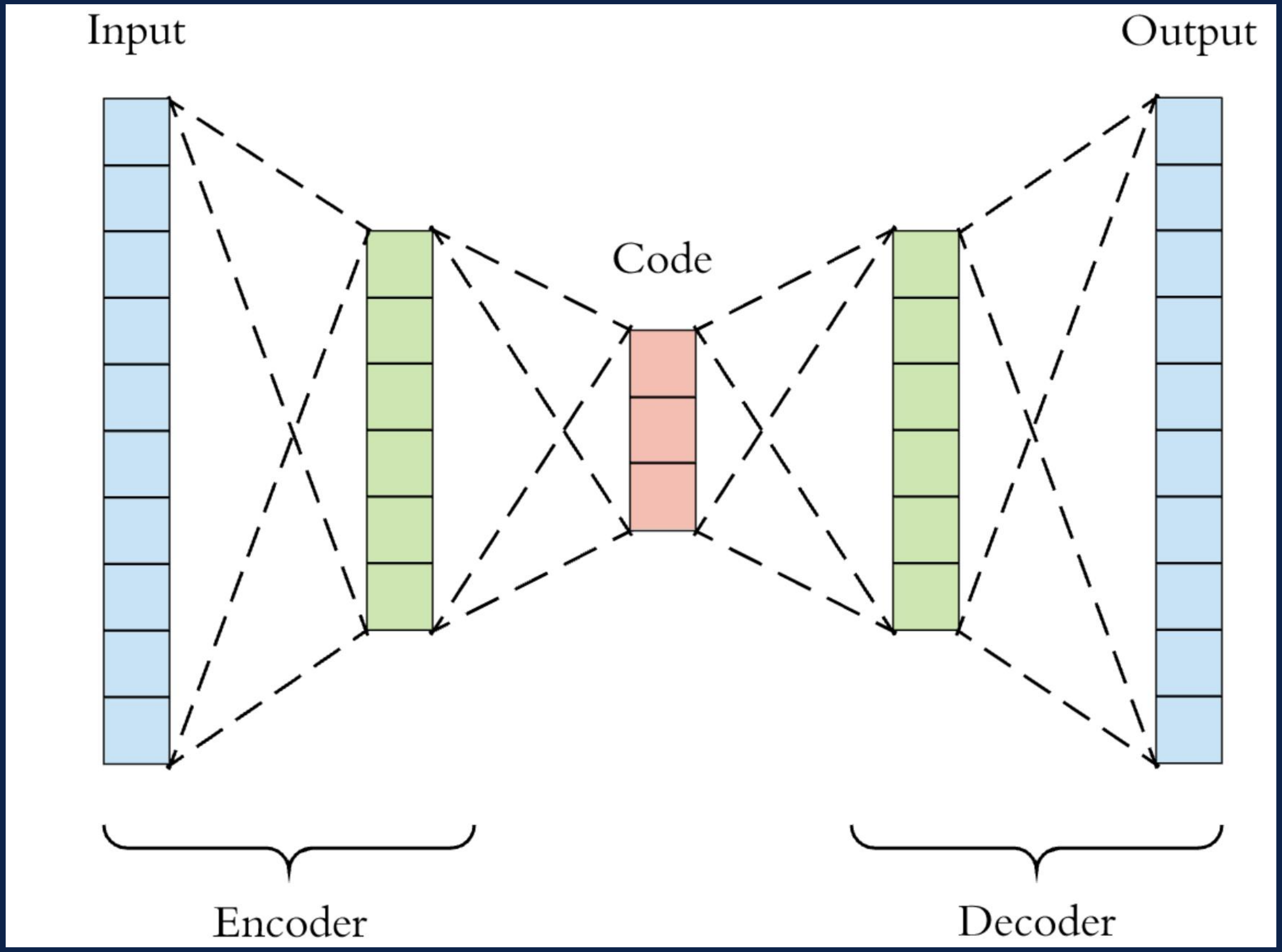
Image: LSST Project/NSF/AURA

Reading Overview

- Covering:
  - Tutorial on Variational Autoencoders by Carl Doersch, Sections 1 and 2
- Original papers:
  - *Auto-encoding Variational Bayes* Kingma, D. P. and Welling, M. ICLR, 2014
  - *Stochastic Backpropagation and Approximate Inference in Deep Generative Models* Rezende, et al. ICML, 2014

Auto-encoders

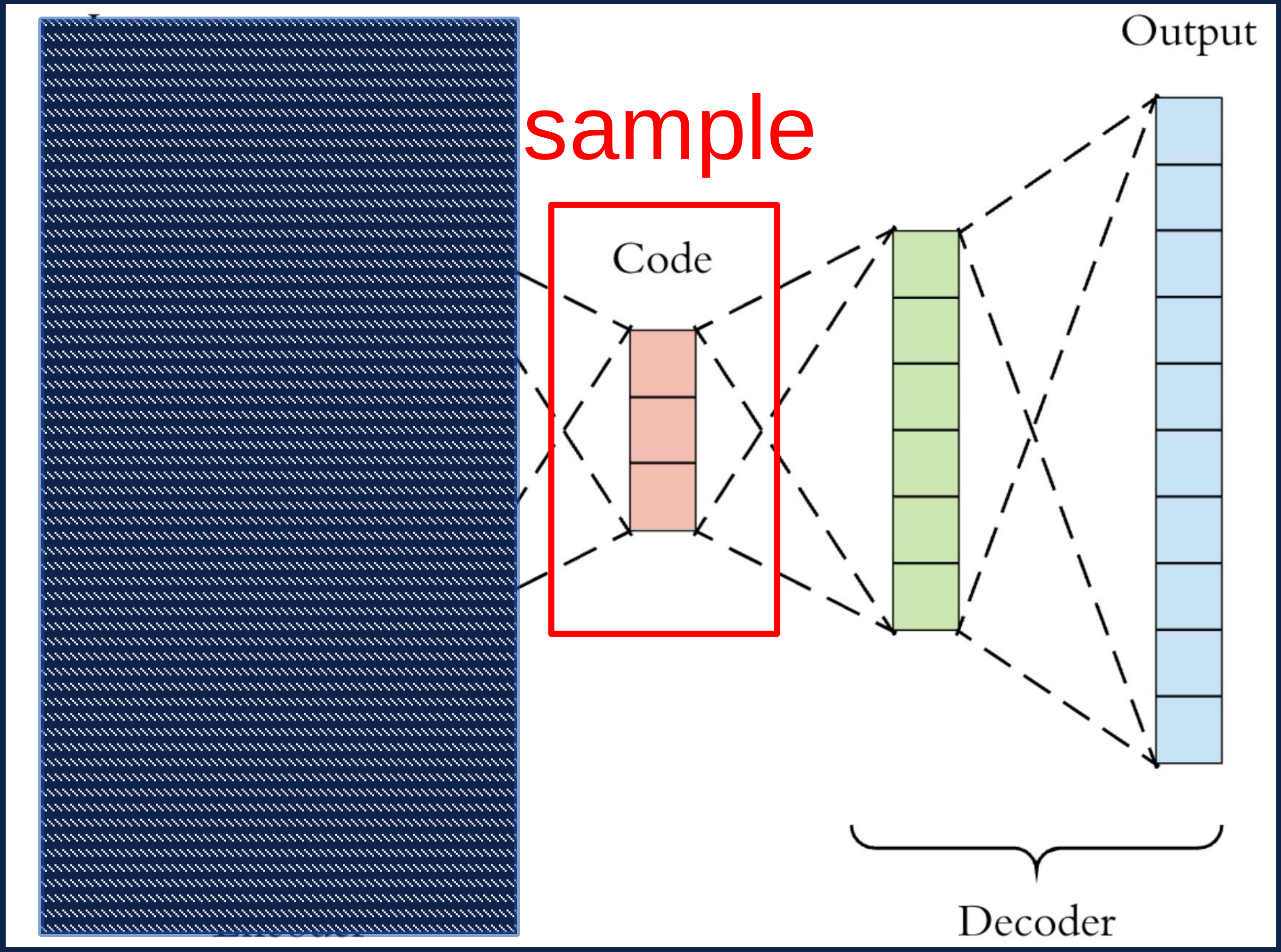
Infer latent state & reconstruct data from it



https://towardsdatascience.com/generating-images-with-autoencoders-77fd3a8dd368

Variational autoencoders (VAEs)

- Generate new data like training data, X



Generative models

- generate output as similar as possible to P(X)
- examples:
  - generate images
  - generate a mass of similar looking objects (trees in video games)
  - generating text
  - …

Other generative models vs VAEs

Existing approaches

- computationally expensive
- impose structure on data

VAEs

- fast training via backpropagation
- weak assumptions about structure of data

VAE Objective

Generate output as similar as possible to P(X) by **maximizing *Equation 1*:**

$$P(X) = \int P(X|z;\theta)P(z)dz$$

- X : is training/observed data
- z : vector of latent variables
- $P(X|z;\theta)$: likelihood of producing training samples; often Gaussian, $P(X|z,\theta) = \mathcal{N}(X|f(z;\theta),\sigma^2 * I)$
  - $f(z;\theta)$: (mean of normal) family of deterministic functions that generate data X using z and parameters θ
  - $\sigma^2 * I$: (variance of normal) identity matrix * scalar

VAE Steps

1. Train network
    1. $\boxed{\text{Define latent variables } z}$
    2. Find computable formula for P(X)
    3. Optimize computable formula for P(X) using stochastic gradient descent (and back-propagation)
2. Generate new samples
    1. Generate new samples from P(z)

1. Define latent variables

Example

MNIST handwritten digits latent variables

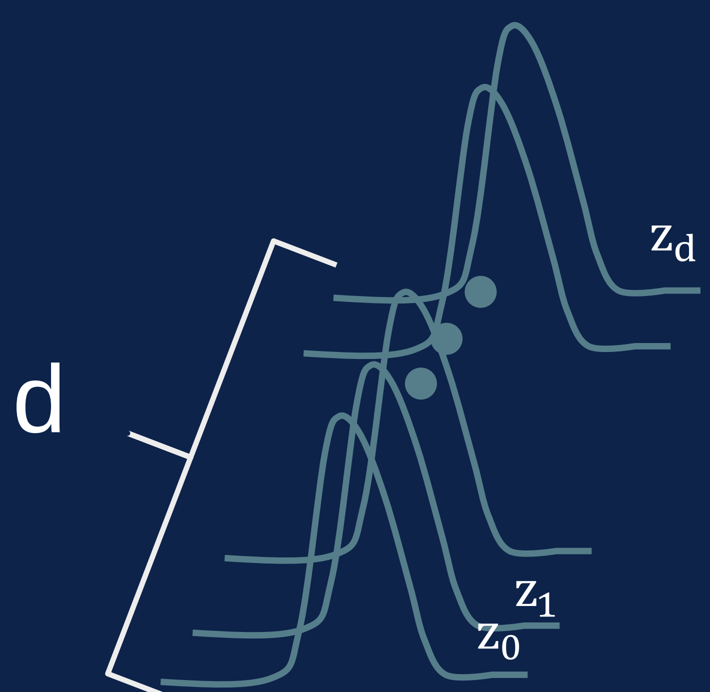
- slant
- size
- stroke thickness
- …
- so many!

Avoid defining the latent structure

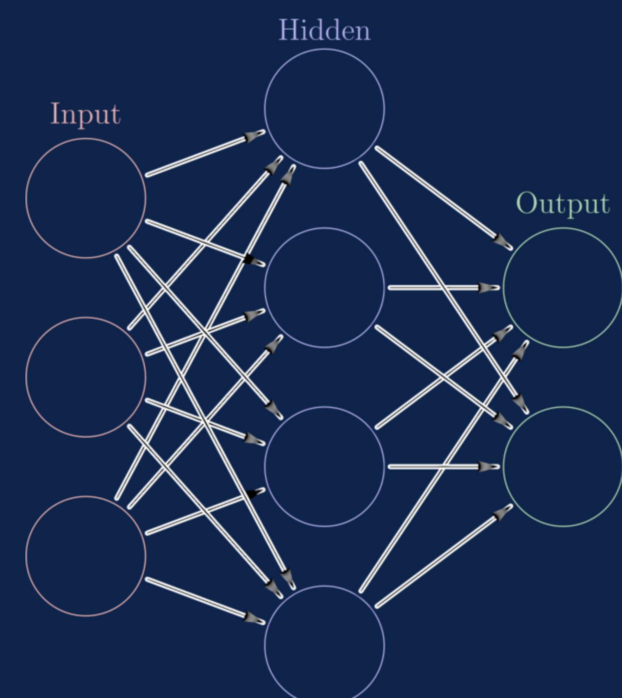
1. Define latent variables

“any distribution in d dimensions can be generated by taking a set of d variables that are normally distributed and mapping them through a sufficiently complicated function”

d dimensions of Z



neural net $f(z;\theta)$



https://en.wikipedia.org/wiki/Artificial_neural_network

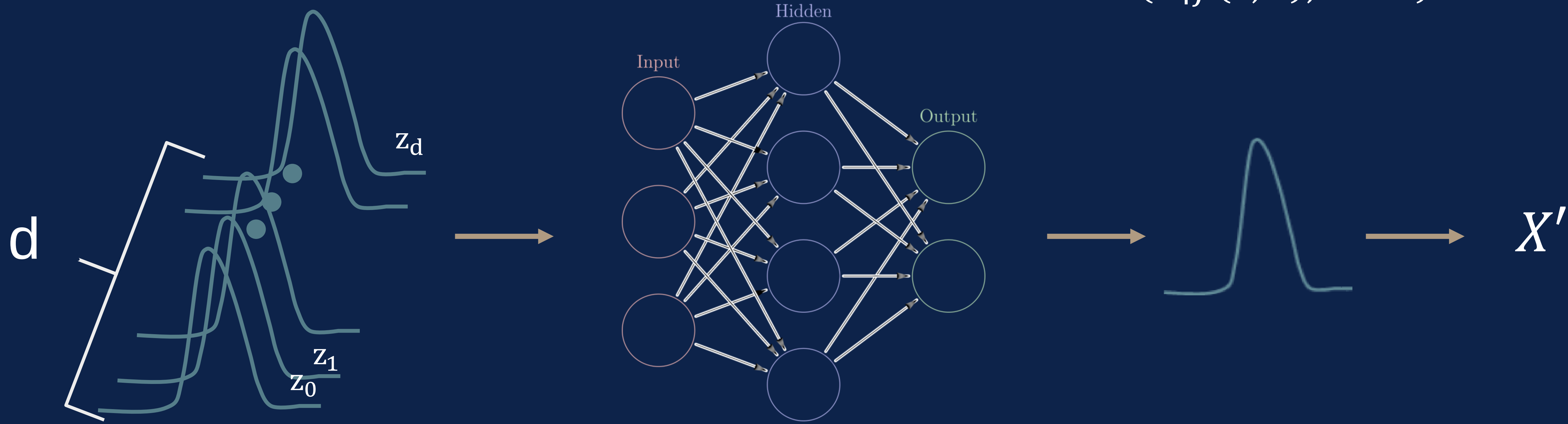
1. Define latent variables

- $\mathrm{P}(\mathrm{z}) = \mathcal{N}(0, I)$
- $f(z;\theta)$: neural net; mean of likelihood
- $P(X|z;\theta)$: likelihood

d dimensions of Z,
$\mathrm{z} \sim \mathcal{N}(0, I)$

neural net $f(z;\theta)$

d-dimensional normal
$P(X|z;\theta) =$
$\mathcal{N}(X|f(z;\theta), \sigma^2 * I)$



https://en.wikipedia.org/wiki/Artificial_neural_network

VAE Steps

1. Train network
   1. Define latent variables z
   2. Find computable formula for P(X)
   3. Optimize computable formula for P(X) using stochastic gradient descent (and back-propagation)
2. Generate new samples
   1. Generate new samples from P(z)

VAE Objective

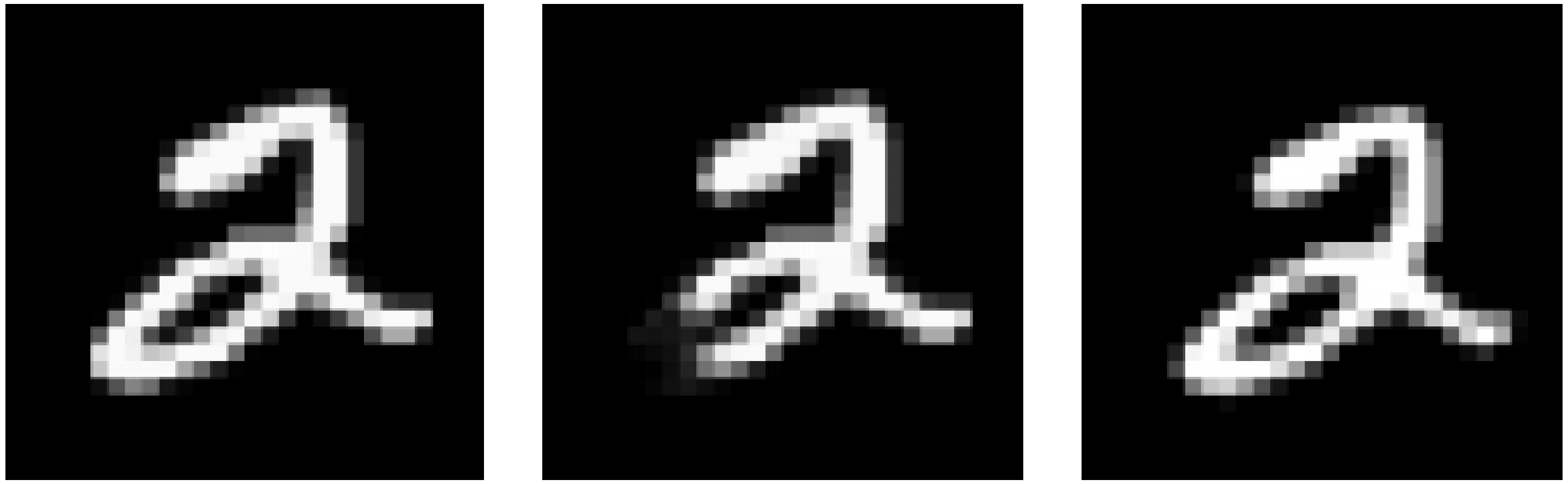
Generate output as similar as possible to P(X) by **maximizing *Equation 1*:**

$$P(X) = \int P(X|z;\theta)P(z)dz$$

- X : is training/observed data
- z : vector of latent variables
- $P(X|z;\theta)$: likelihood of producing training samples; often Gaussian, $P(X|z,\theta) = \mathcal{N}(X|f(z;\theta),\sigma^2 * I)$
- $P(z) = \mathcal{N}(\mathrm{z}|0,\mathrm{I})$

VAE Objective: Sampling

- Why not sample?
  - sampling in high-dimensional space requires *many* samples



(a) (b) (c)

target bad model good model

$$distance^2 \quad 0.0387 \quad < \quad distance^2 \quad 0.2693$$

VAE solution to sampling

- Make space of Z sampling ‘smaller’ by using *new function*
- *new function* $Q(z|X)$ gives distribution over z values that are likely to produce X
  - ideally, $\mathrm{Q}(z|X)$ space will be **smaller** than $P(z)$
- usually normal
- $Q(z|X) = \mathcal{N}(z|\mu(X;\theta), \Sigma(X;\theta))$
  - μ, Σ deterministic functions with parameters θ that can be learned from data; usually implemented via neural networks

How does $P(X)$ relate to $\mathbb{E}_{z\sim Q}P(X|z)$?

KL-divergence definition

$$\mathcal{D}\big(q(x) \parallel p(x)\big) = \mathbb{E}_{q(x)}\left[\log\left(\frac{q(x)}{p(x)}\right)\right]$$

KL-divergence between $P(z|X)$ and Q(z)

$$\mathcal{D}\left(Q(Z) \parallel P(z|X)\right) = \mathbb{E}_{\mathrm{z}\sim Q(z)}\left[\log\left(\frac{Q(z)}{P(z|X)}\right)\right]$$

$$\mathcal{D}\left(Q(Z) \parallel P(z|X)\right) = \mathbb{E}_{\mathrm{z}\sim Q(z)}\left[\log\big(Q(z)\big) - \log\big(P(z|X)\big)\right]$$

Equation 2

How does $P(X)$ relate to $\mathbb{E}_{z\sim Q}P(X|z)$?

$$\mathcal{D}\left(Q(Z) \parallel P(z|X)\right) = \mathbb{E}_{z\sim Q(z)}\left[\log\left(Q(z)\right) - \log\left(P(z|X)\right)\right]$$

Equation 2

Apply Bayes rule: $P(z|X) = \frac{P(X|z)p(z)}{P(X)}$

$$\log\left(P(X)\right) - \mathcal{D}[Q(z)||P(z|X)] = E_{z\sim Q}\left[\log\left(P(X|z)\right)\right] - \mathcal{D}(Q(z)||P(z))$$

Equation 4

$Q(z)$ does not depend on X though? Replace $Q(z)$ with $Q(z|X)$

How does $P(X)$ relate to $\mathbb{E}_{z\sim Q}P(X|z)$?

Equation 5

$$\log\big(P(X)\big) - \mathcal{D}[Q(z|\mathrm{X})||P(z|X)] = \boxed{E_{z\sim Q}\big[\log\big(P(X|z)\big)\big] - \mathcal{D}(Q(z|\mathrm{X})||P(z))}$$

maximize probability of data

minimize divergence (but $P(z|X)$ cannot be computed analytically)

Perform stochastic gradient descent on right hand side (RHS)

($Q(z|\mathrm{X})$ is tractable)

cannot compute

can optimize with stochastic gradient descent

VAE Steps

1. Train network
   1. Define latent variables z
   2. Find computable formula for P(X)
   3. Optimize computable formula for P(X) using stochastic gradient descent (and back-propagation)
2. Generate new samples
   1. Generate new samples from P(z)

Performing stochastic gradient descent on

$$E_{z\sim Q}\left[\log\left(P(X|z)\right)\right] - \mathcal{D}[Q(z|\mathrm{X})||P(z)]$$

$$\mathcal{D}[Q(z|\mathrm{X})||P(z)]$$

- $Q(z|X) = \mathcal{N}(z|\mu(X;\theta), \Sigma(X;\theta))$
  - μ, Σ deterministic functions with parameters θ that can be learned from data; usually implemented via neural networks
- $P(z) = \mathcal{N}(\mathrm{z}|0, \mathrm{I})$
- KL-divergence between 2 multivariate Gaussian distributions -> closed form:

$$\mathcal{D}[Q(z|\mathrm{X})||P(z)] = \mathcal{D}[\mathcal{N}\left(\mu(\mathrm{X}), \Sigma(\mathrm{X})\right)||\mathcal{N}(0, \mathcal{I})]$$

Performing stochastic gradient descent on

$$E_{z\sim Q}\left[\log\left(P(X|z)\right)\right]-\mathcal{D}[Q(z|\mathrm{X})||P(z)]$$

$$E_{z\sim Q}\left[\log\left(P(X|z)\right)\right]$$

- take 1 sample of z, $z\sim\mathrm{Q}$ and use it to approximate $P(X|z)$

Performing stochastic gradient descent on

$$E_{z\sim Q}\left[\log\left(P(X|z)\right)\right] - \mathcal{D}[Q(z|\mathrm{X})||P(z)]$$

Optimize

$$E_{X\sim D}[E_{z\sim Q(\mathrm{z}|X)}\left[\log\left(P(X|z)\right)\right] - \mathcal{D}[Q(z|\mathrm{X})||P(z)]]$$

Equation 8

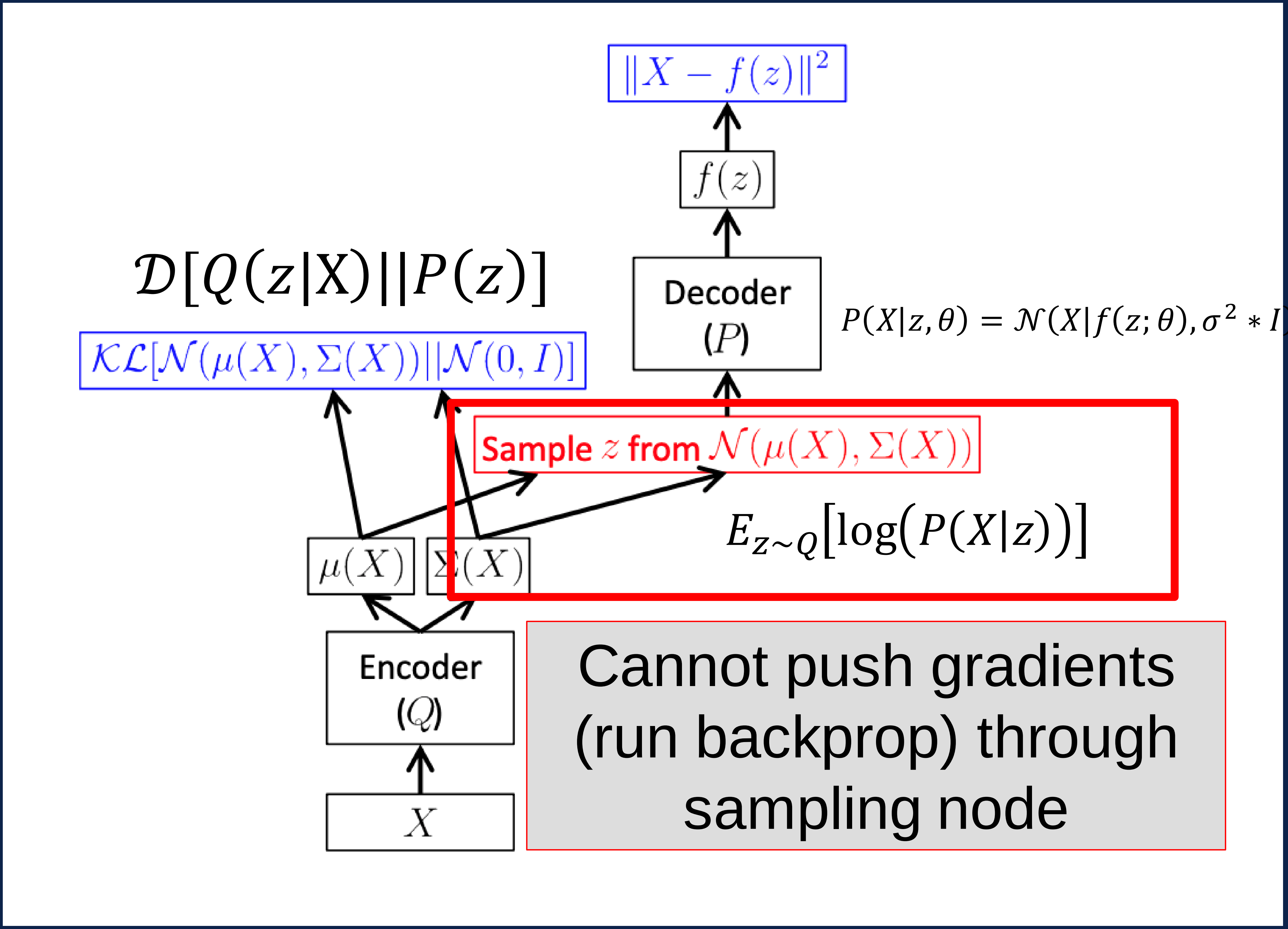
For each sample of X (from training data D), use single sample of Z from $Q(\mathrm{z}|X)$ to compute gradient of

$$\left[\log\left(P(X|z)\right)\right] - \mathcal{D}[Q(z|\mathrm{X})||P(z)]$$

Average gradient over many samples of X and z to converge on gradient of *Equation 8*

Performing stochastic gradient descent on

$$E_{z\sim Q}\left[\log\left(P(X|z)\right)\right] - \mathcal{D}[Q(z|X)||P(z)]$$



Performing stochastic gradient descent using ***reparameterization trick***

- Redefine sampled latent vector $z \sim Q(\mathrm{z}|X)$ as:
  - $\mu + \sigma * \epsilon$
    - $\mu + \sigma$ we are learning
    - $\epsilon \sim \mathcal{N}(0,1)$
- Now, μ and σ have gradients but ϵ will never change - it is a fixed stochastic node, and we do not need to run backprop on it.

Performing stochastic gradient descent using ***reparameterization trick***

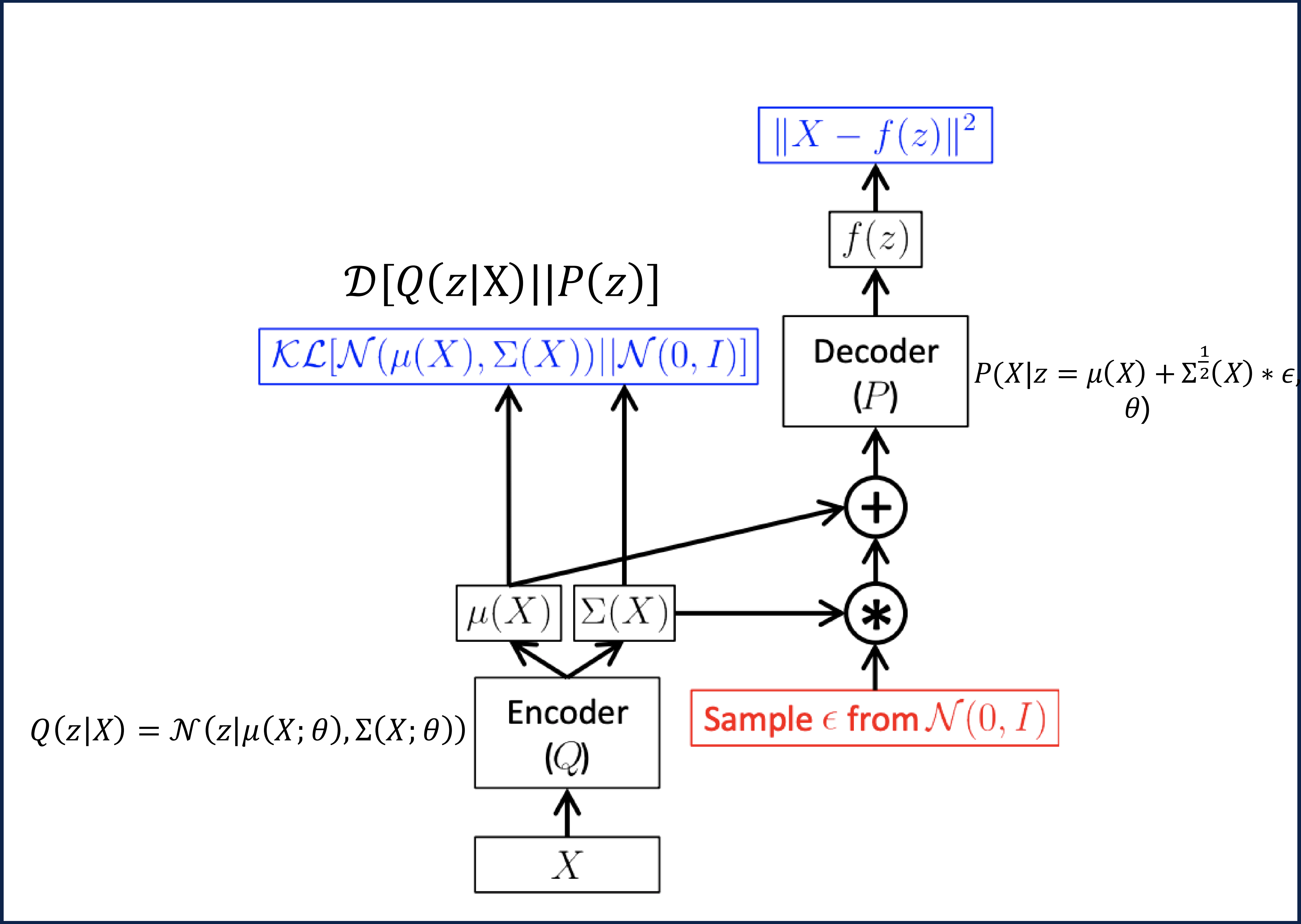
1. Get mean and covariance of $Q(z|X)$: $\mu(X)$ and $\sigma(X)$
2. "Sample" from $\mathcal{N}(\mu(X), \sigma(X))$ by:
   1. sampling $\epsilon \sim \mathcal{N}(0,1)$
   2. computing $z = \mu(X) + \Sigma^{\frac{1}{2}}(X) * \epsilon$

Gradient Equation:

$$E_{X\sim D}\left[E_{\epsilon\sim\mathcal{N}(0,I)}[\log P(X|z = \mu(X) + \Sigma^{1/2}(X) * \epsilon)] - \mathcal{D}\left[Q(z|X) \| P(z)\right]\right]$$

Equation 10

Performing stochastic gradient descent using ***reparameterization trick***



4 normal distributions in VAE

1. $P(X|z,\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I)$
   - Probability of each training example when sampled from an area of the latent space
2. $P(z) = \mathcal{N}(\mathrm{z}|0, \mathrm{I})$
   - Generating the latent space distribution, P(**z**) of dimension **d**, using **d** normal distributions
3. $Q(z|X) = \mathcal{N}(z|\mu(X;\theta), \Sigma(X;\theta))$
   - Enforcing P(**z**) towards Q(**z**) by setting Q(**z**) to the normal distribution
4. $\epsilon \sim \mathcal{N}(0,1)$
   - Generating points for our decoder P, so Q is differentiable and we can use back propagation (***reparameterization trick***)

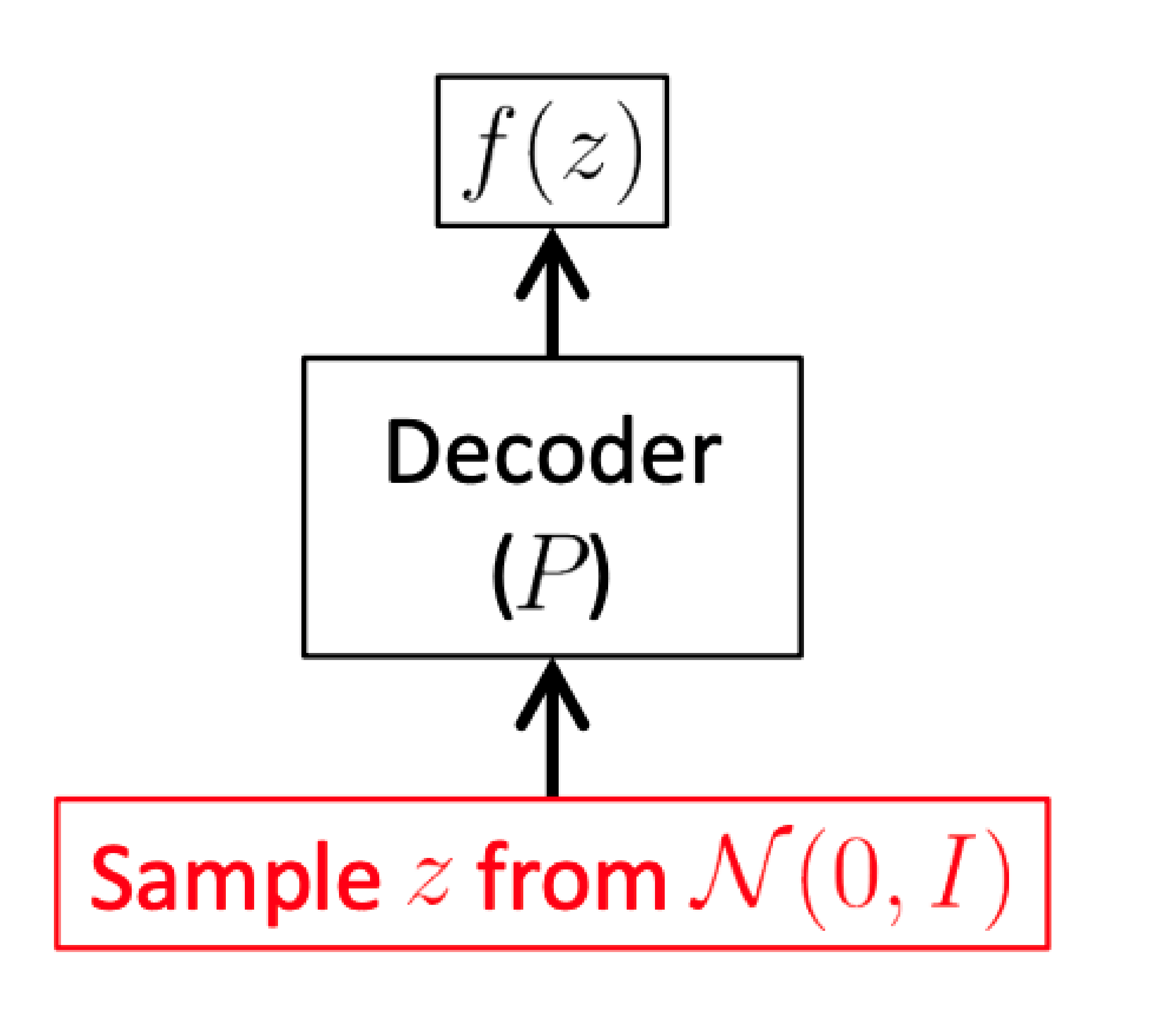
https://medium.com/datadriveninvestor/variational-autoencoder-vae-d1cf436e1e8f

VAE Steps

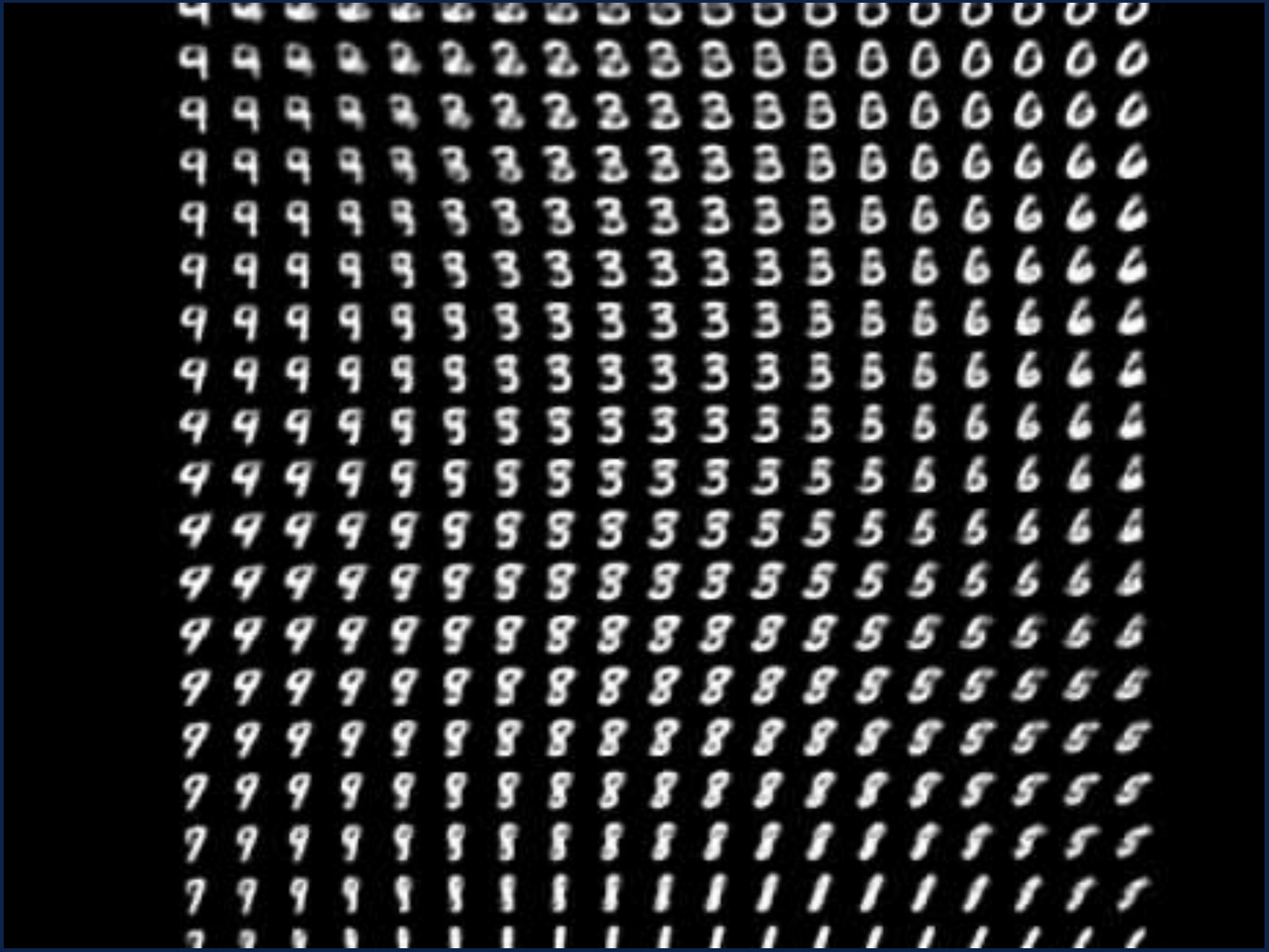
1. Train network
   1. Define latent variables z
   2. Find computable formula for P(X)
   3. Optimize computable formula for P(X) using stochastic gradient descent (and back-propagation)
2. Generate new samples
   1. Generate new samples from P(z)

Generating new samples

1. Sample $z \sim \mathcal{N}(0, I)$
2. Input sample into decoder



VAE MNIST Training



YouTube: Variational Autoencoder 2D latent space evolution on MNIST

Questions?