Deep generative modeling: Latent Variable Models

Jakub M. Tomczak
Deep Learning

VRIJE UNIVERSITEIT AMSTERDAM

INTRODUCTION

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

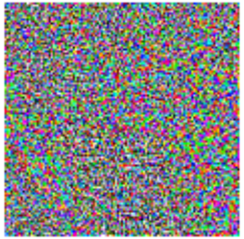
p(**panda**|x)=0.99

...

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

+



=

p(**panda**|x)=0.99

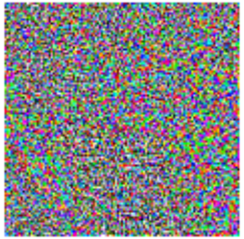
...

noise

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

+



=

p(**panda**|x)=0.99
...

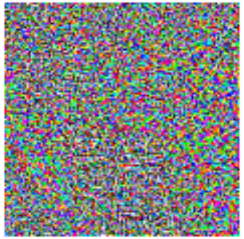
noise

p(**panda**|x)=0.01
...
p(**dog**|x)=0.9

IS GENERATIVE MODELING IMPORTANT?

We learn a neural network to classify images:

+



=

p(**panda**|x)=0.99
...

noise

p(**panda**|x)=0.01
…
p(**dog**|x)=0.9

There is no semantic understanding of images.

IS GENERATIVE MODELING IMPORTANT?

This simple example shows that:

- A discriminative model is (probably) **not enough**.
- We need a notion of **uncertainty**.
- We need to **understand** the reality.

IS GENERATIVE MODELING IMPORTANT?

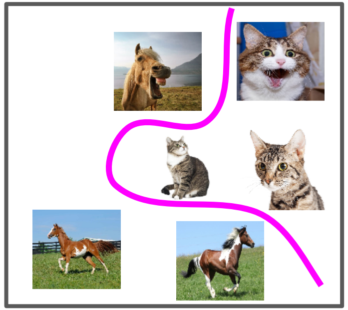
This simple example shows that:

- A discriminative model is (probably) **not enough**.
- We need a notion of **uncertainty**.
- We need to **understand** the reality.

A possible solution is **generative modeling**.

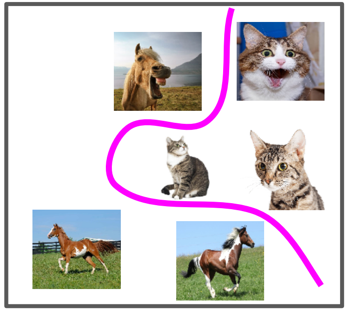
IS GENERATIVE MODELING IMPORTANT?

IS GENERATIVE MODELING IMPORTANT?

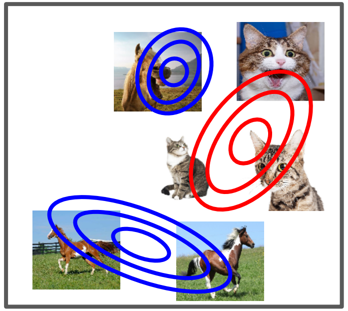


$$p_\theta(y|x)$$

IS GENERATIVE MODELING IMPORTANT?

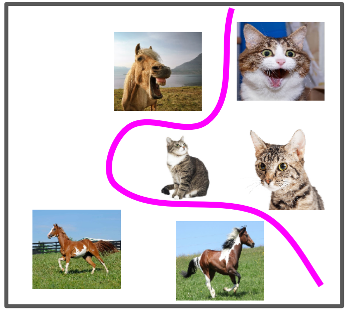


$$p_\theta(y|x)$$

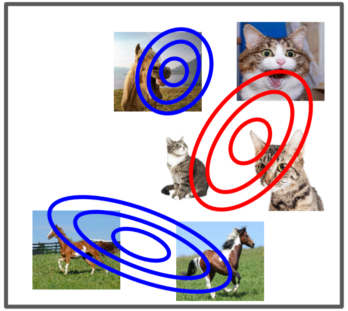


$$p_\theta(x, y) = p_\theta(y|x)\ p_\theta(x)$$

IS GENERATIVE MODELING IMPORTANT?



$$p_\theta(y|x)$$

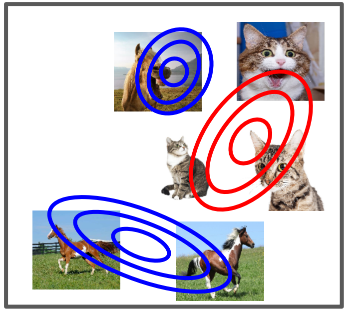


$$p_\theta(x, y) = p_\theta(y|x)\ p_\theta(x)$$

IS GENERATIVE MODELING IMPORTANT?

$$p_\theta(y|x)$$

High probability
of a **horse.**
=
Highly probable decision!



$$p_\theta(x, y) = p_\theta(y|x)\ p_\theta(x)$$

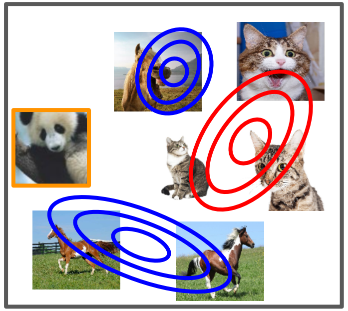
IS GENERATIVE MODELING IMPORTANT?

$$p_\theta(y|x)$$

High probability of a **horse.**

=

Highly probable decision!



$$p_\theta(x, y) = p_\theta(y|x)\ p_\theta(x)$$

High probability of a **horse.**

x

Low probability of the **object**

=

Uncertain decision!

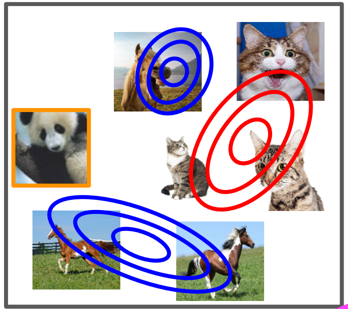
IS GENERATIVE MODELING IMPORTANT?

$$p_\theta(y|x)$$

High probability of a **horse.**

=

Highly probable decision!



$$p_\theta(x, y) = p_\theta(y|x)\, p_\theta(x)$$

!

High probability of a **horse.**

x

Low probability of the **object**

=

Uncertain decision!

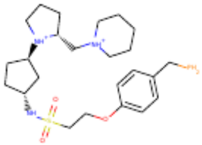
WHERE DO WE USE DEEP GENERATIVE MODELING?

" i want to talk to you . "
"i want to be with you . "
"i do n't want to be with you . "
i do n't want to be with you .
she did n't want to be with him .

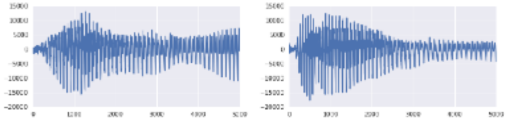
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Text analysis

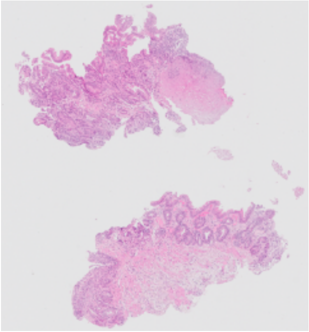
Image analysis



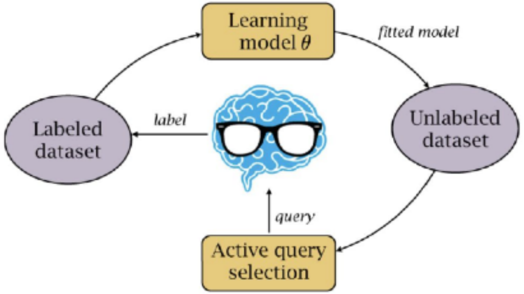
Graph analysis



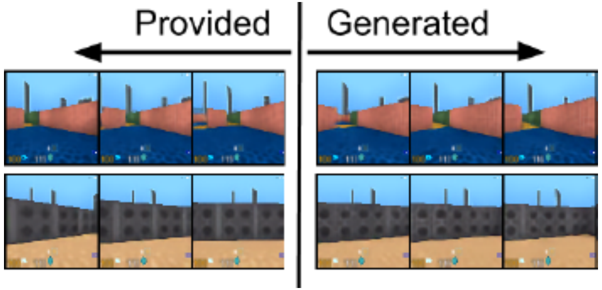
Audio analysis



Medical data



Active Learning



Reinforcement Learning

and more...

HOW TO FORMULATE DEEP GENERATIVE MODELS?

Generative model

Autoregressive (*e.g.*, PixelCNN)

Flow-based (e.g., RealNVP, GLOW)

Latent variable models

Implicit models (*e.g.*, GANs)

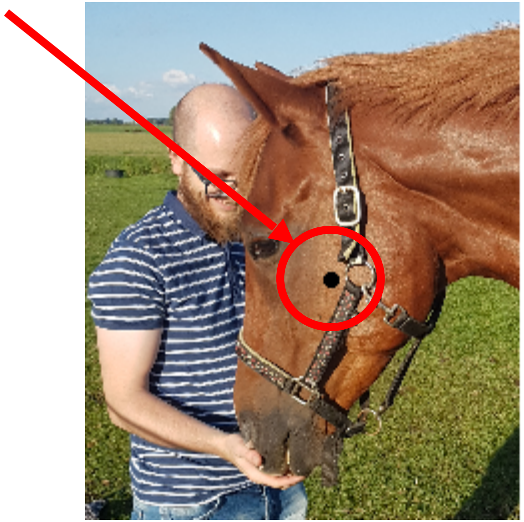
Prescribed models (*e.g.*, VAE)

HOW TO FORMULATE DEEP GENERATIVE MODELS?

	Training	Likelihood	Sampling	Compression
Autoregressive models (e.g., PixelCNN)	Stable	Exact	Slow	No
Flow-based models (e.g., RealNVP)	Stable	Exact	Fast/Slow	No
Implicit models (e.g., GANs)	Unstable	No	Fast	No
Prescribed models (e.g., VAEs)	Stable	Approximate	Fast	Yes

Modeling in high-dimensional spaces is difficult.

Modeling in high-dimensional spaces is difficult.



Modeling in high-dimensional spaces is difficult.

Modeling **all dependencies** among pixels:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c=1}^{C} \psi_c(\mathbf{x}_c)$$

Modeling in high-dimensional spaces is difficult.

Modeling **all dependencies** among pixels:

$$p(\mathbf{x}) = \frac{1}{Z}\prod_{c=1}^{C}\psi_c(\mathbf{x}_c)$$

problematic

Modeling in high-dimensional spaces is difficult.

Modeling **all dependencies** among pixels:

$$p(\mathbf{x}) = \frac{1}{Z} \prod_{c=1}^{C} \psi_c(\mathbf{x}_c)$$

problematic

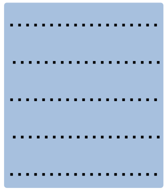
A possible **solution**: **Latent Variable Models**!

GENERATIVE MODELING WITH LATENT VARIABLES

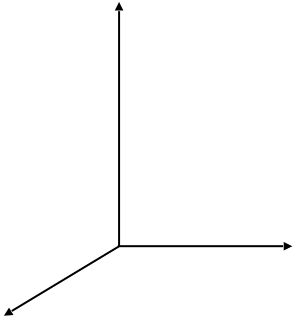
Generative process:

1. $\mathbf{z} \sim p_{\lambda}(\mathbf{z})$
2. $\mathbf{x} \sim p_{\theta}(\mathbf{x}|\mathbf{z})$

Z: 2D



X: 3D

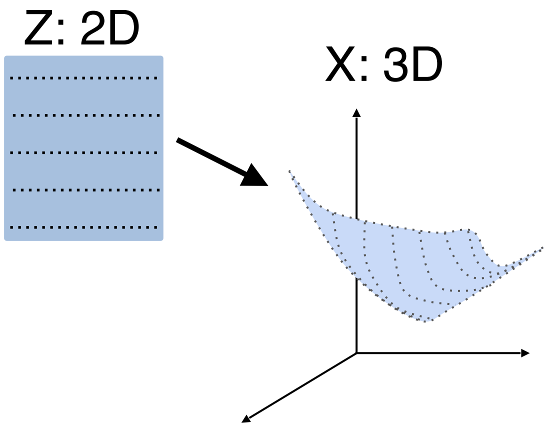


GENERATIVE MODELING WITH LATENT VARIABLES

Generative process:

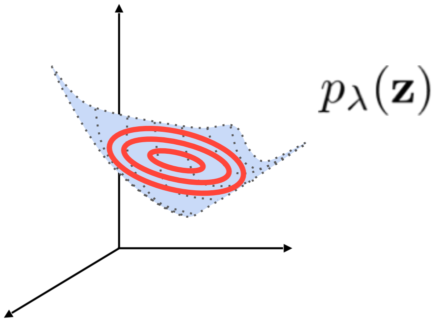
$$1.\ \mathbf{z} \sim p_\lambda(\mathbf{z})$$

$$2.\ \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$



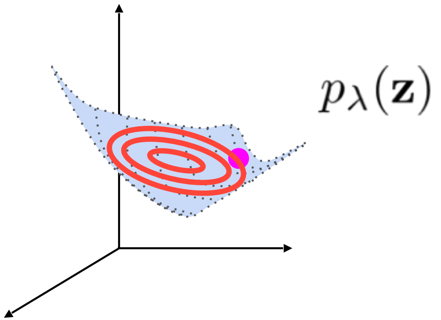
Generative process:

1. $\mathbf{z} \sim p_\lambda(\mathbf{z})$
2. $\mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$



Generative process:

1. $\mathbf{z} \sim p_{\lambda}(\mathbf{z})$
2. $\mathbf{x} \sim p_{\theta}(\mathbf{x}|\mathbf{z})$

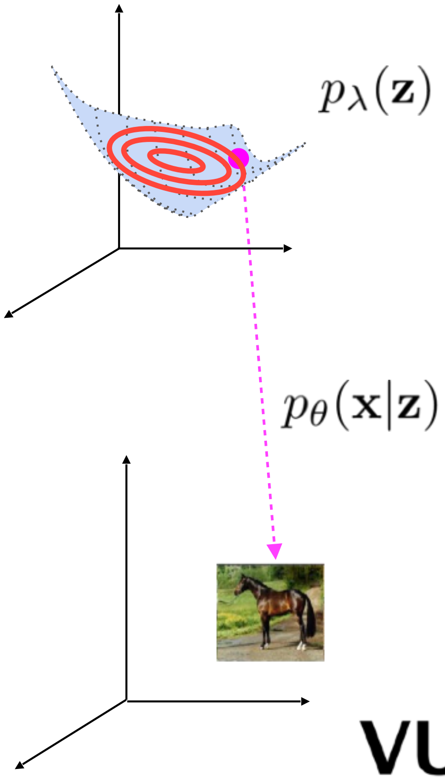


GENERATIVE MODELING WITH LATENT VARIABLES

Generative process:

$$1.\ \mathbf{z} \sim p_{\lambda}(\mathbf{z})$$

$$2.\ \mathbf{x} \sim p_{\theta}(\mathbf{x}|\mathbf{z})$$

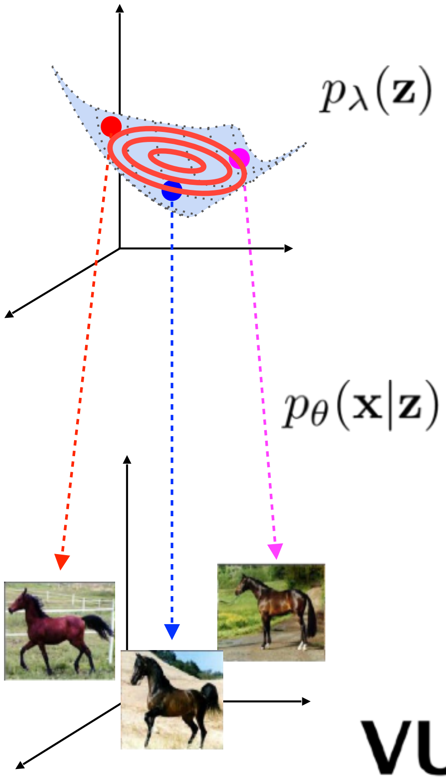


GENERATIVE MODELING WITH LATENT VARIABLES

Generative process:

$$1.\ \mathbf{z} \sim p_\lambda(\mathbf{z})$$

$$2.\ \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$

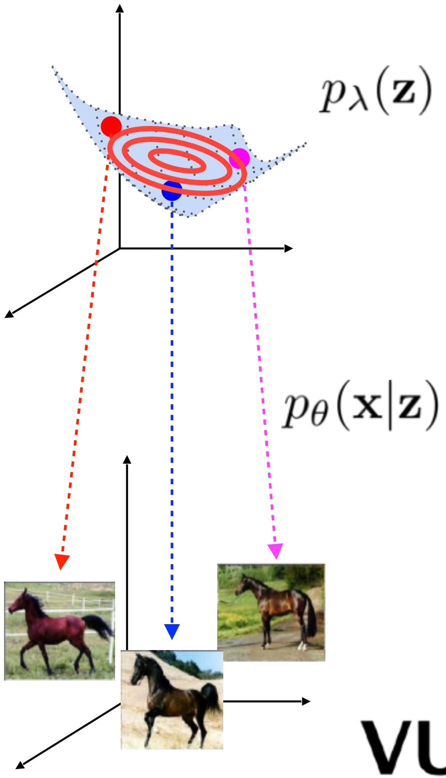


Generative process:

$$1.\ \mathbf{z} \sim p_\lambda(\mathbf{z})$$
$$2.\ \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$

The log-likelihood function:

$$\log p_\vartheta(\mathbf{x}) = \log \int p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z}$$



GENERATIVE MODELING WITH LATENT VARIABLES

Generative process:

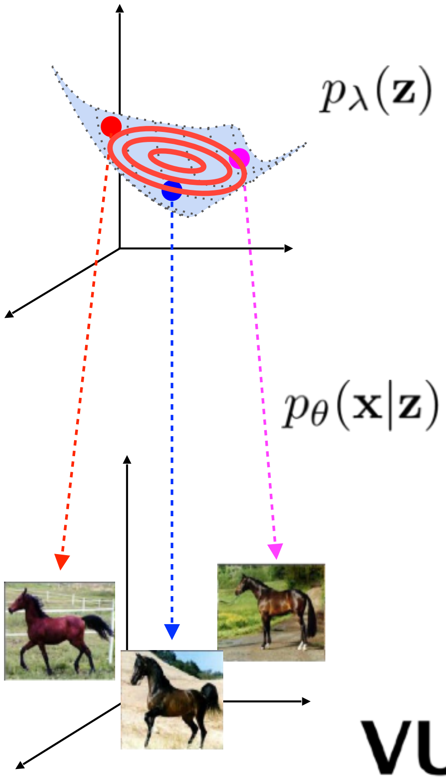
$$1.\ \mathbf{z} \sim p_\lambda(\mathbf{z})$$

$$2.\ \mathbf{x} \sim p_\theta(\mathbf{x}|\mathbf{z})$$

The log-likelihood function:

$$\log p_\vartheta(\mathbf{x}) = \log \underbrace{\int}_{} p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z}$$

How to train such model efficiently?



Let us assume: $p(\mathbf{z}) = \mathcal{N}(0,\mathbf{I})$.

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$.

And a linear transformation ($\mathbf{W} \in \mathbb{R}^{D \times M}$):

$$\mathbf{x} = \mathbf{W}\mathbf{z} + \mu + \varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$$

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0,\mathbf{I})$.

And a linear transformation ($\mathbf{W} \in \mathbb{R}^{D\times M}$):

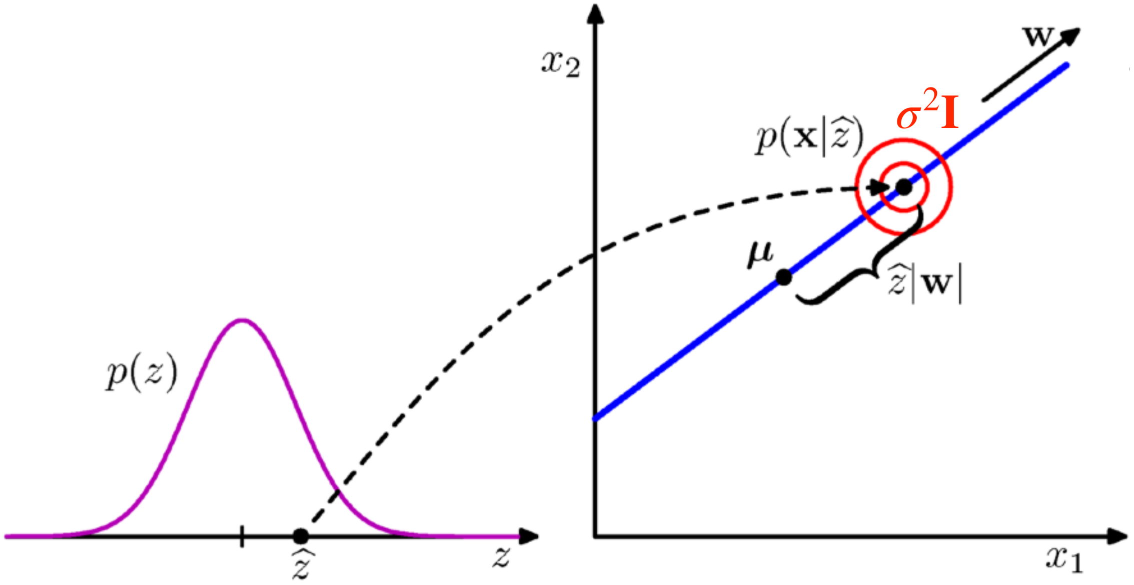
$$\mathbf{x} = \mathbf{W}\mathbf{z} + \mu + \varepsilon, \text{ where } \varepsilon \sim \mathcal{N}(0,\sigma^2\mathbf{I})$$

that results in the following conditional distribution:

$$p(\mathbf{x}\,|\,\mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$$

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$.

And the following conditional distribution: $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2 \mathbf{I})$



Generative process

Now, the question is how to calculate the log-likelihood:

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}$$

Now, the question is how to calculate the log-likelihood:

$$p(\mathbf{x}) = \int \underbrace{p(\mathbf{x} \mid \mathbf{z})}_{\text{Gaussian}} \underbrace{p(\mathbf{z})}_{\text{Gaussian}} \mathrm{d}\mathbf{z}$$

Now, the question is how to calculate the log-likelihood:

$$p(\mathbf{x}) = \int \underbrace{p(\mathbf{x}\,|\,\mathbf{z})}_{\text{Gaussian}}\underbrace{p(\mathbf{z})}_{\text{Gaussian}}\mathrm{d}\mathbf{z}$$

$$= \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^{\top} + \sigma^2\mathbf{I})$$

Marginal and Conditional Gaussians

Given a marginal Gaussian distribution for $\mathbf{x}$ and a conditional Gaussian distribution for $\mathbf{y}$ given $\mathbf{x}$ in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \quad (2.113)$$

$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}) \quad (2.114)$$

the marginal distribution of $\mathbf{y}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}) \quad (2.115)$$

$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\Sigma}\{\mathbf{A}^{\mathrm{T}}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\}, \boldsymbol{\Sigma}) \quad (2.116)$$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^{\mathrm{T}}\mathbf{L}\mathbf{A})^{-1}. \quad (2.117)$$

Bishop, “Pattern Recognition and Machine Learning”

Now, the question is how to calculate the log-likelihood:

$$p(\mathbf{x}) = \int \underbrace{p(\mathbf{x}\,|\,\mathbf{z})}_{\text{Gaussian}} \underbrace{p(\mathbf{z})}_{\text{Gaussian}} \mathrm{d}\mathbf{z}$$

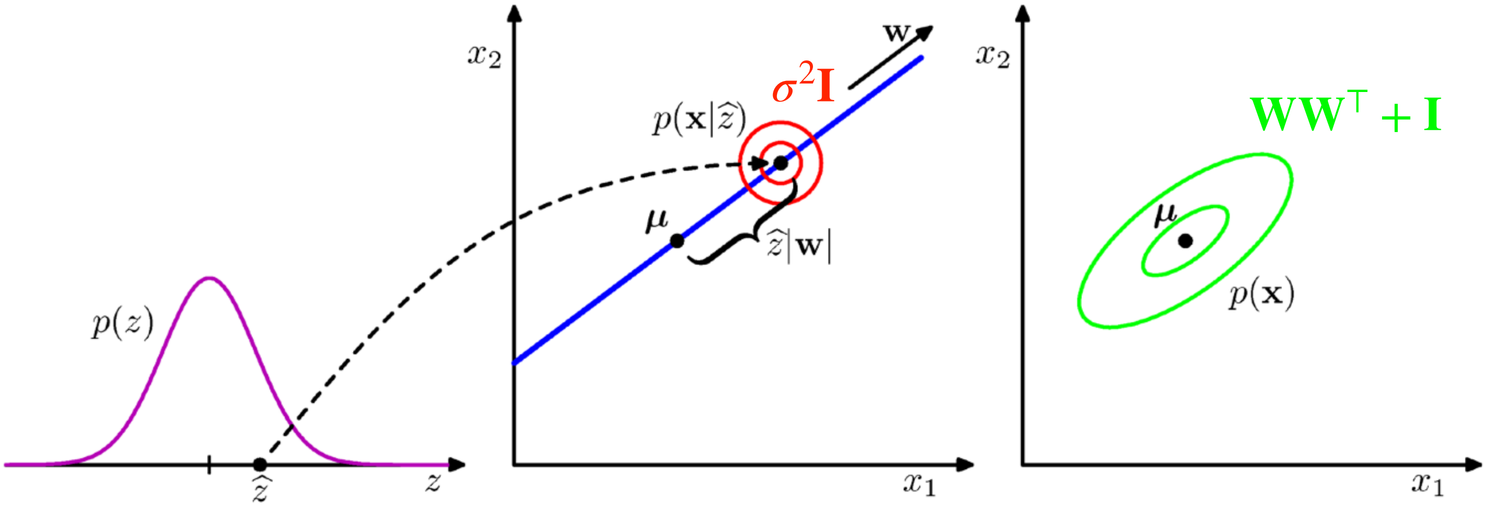
$$= \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^\top + \sigma^2 \mathbf{I})$$

The integral is tractable, and it is again Gaussian!

Now, the question is how to calculate the log-likelihood:

$$p(\mathbf{x}) = \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})\mathrm{d}\mathbf{z}$$

$$= \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I})$$



Since the model is linear, and all distributions are Gaussians, we can also calculate the posterior over $\mathbf{z}$:

$$p(\mathbf{z} \,|\, \mathbf{x}) = \mathcal{N}\left(\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \mu), \sigma^{-2}\mathbf{M}\right)$$

where:

$$\mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}$$

Since the model is linear, and all distributions are Gaussians, we can also calculate the posterior over $\mathbf{z}$:

$$p(\mathbf{z}\,|\,\mathbf{x}) = \mathcal{N}\left(\mathbf{M}^{-1}\mathbf{W}^{\top}(\mathbf{x} - \mu), \sigma^{-2}\mathbf{M}\right)$$

where:

$$\mathbf{M} = \mathbf{W}^{\top}\mathbf{W} + \sigma^2\mathbf{I}$$

Marginal and Conditional Gaussians

Given a marginal Gaussian distribution for $\mathbf{x}$ and a conditional Gaussian distribution for $\mathbf{y}$ given $\mathbf{x}$ in the form

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) \tag{2.113}$$
$$p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\mathbf{x} + \mathbf{b}, \mathbf{L}^{-1}) \tag{2.114}$$

the marginal distribution of $\mathbf{y}$ and the conditional distribution of $\mathbf{x}$ given $\mathbf{y}$ are given by

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \mathbf{A}\boldsymbol{\Lambda}^{-1}\mathbf{A}^{\mathrm{T}}) \tag{2.115}$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\Sigma}\{\mathbf{A}^{\mathrm{T}}\mathbf{L}(\mathbf{y} - \mathbf{b}) + \boldsymbol{\Lambda}\boldsymbol{\mu}\}, \boldsymbol{\Sigma}) \tag{2.116}$$

where

$$\boldsymbol{\Sigma} = (\boldsymbol{\Lambda} + \mathbf{A}^{\mathrm{T}}\mathbf{L}\mathbf{A})^{-1}. \tag{2.117}$$

The final model is the following ($\mathbf{W} \in \mathbb{R}^{D\times M}$):

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

$$p(\mathbf{x}\,|\,\mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$$

$$p(\mathbf{z}\,|\,\mathbf{x}) = \mathcal{N}\left(\mathbf{M}^{-1}\mathbf{W}^\top(\mathbf{x} - \mu), \sigma^{-2}\mathbf{M}\right)$$

where $\mathbf{M} = \mathbf{W}^\top\mathbf{W} + \sigma^2\mathbf{I}$.

and the marginal distribution:

$$p(\mathbf{x}) = \mathcal{N}(\mu, \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I})$$

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

The logarithm of the likelihood function:

$$\ln p\left(\mathbf{X} \,|\, \mu, \mathbf{W}, \sigma^2\right) = \sum_{n=1}^{N} \ln p\left(\mathbf{x}_n \,|\, \mathbf{W}, \mu, \sigma^2\right)$$

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

The logarithm of the likelihood function:

$$\ln p\left(\mathbf{X} \mid \mu, \mathbf{W}, \sigma^2\right) = \sum_{n=1}^{N} \ln p\left(\mathbf{x}_n \mid \mathbf{W}, \mu, \sigma^2\right)$$

REMEMBER: Everything is Gaussian!

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

The logarithm of the likelihood function:

$$\ln p\left(\mathbf{X} \,|\, \mu, \mathbf{W}, \sigma^2\right) = \sum_{n=1}^{N} \ln p\left(\mathbf{x}_n \,|\, \mathbf{W}, \mu, \sigma^2\right)$$

$$= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{x}_n - \mu\right)^\top \mathbf{C}^{-1}\left(\mathbf{x}_n - \mu\right)$$

where

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}$$

$$\mathbf{C}^{-1} = \sigma^{-1}\mathbf{I} - \sigma^{-2}\mathbf{W}\mathbf{M}^{-1}\mathbf{W}^\top$$

The logarithm of the likelihood function:

$$\ln p\left(\mathbf{X} \,|\, \mu, \mathbf{W}, \sigma^2\right) = \sum_{n=1}^{N} \ln p\left(\mathbf{x}_n \,|\, \mathbf{W}, \mu, \sigma^2\right)$$

$$= -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\mathbf{C}| - \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{x}_n - \mu\right)^\top \mathbf{C}^{-1}\left(\mathbf{x}_n - \mu\right)$$

where

$$\mathbf{C} = \mathbf{W}\mathbf{W}^\top + \sigma^2\mathbf{I}$$

$$\underbrace{\mathbf{C}^{-1}} = \sigma^{-1}\mathbf{I} - \sigma^{-2}\mathbf{W}\underbrace{\mathbf{M}^{-1}}\mathbf{W}^\top$$

Inverting C (*D*x*D*) reduces to inverting M (*M*x*M*).

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

It is possible to calculate the solution analytically:

$$\mathbf{W}_{\mathrm{ML}} = \mathbf{U}_M \left(\mathbf{L}_M - \sigma^2 \mathbf{I}\right)^{1/2} \mathbf{R}$$

$$\sigma^2_{\mathrm{ML}} = \frac{1}{D-M} \sum_{i=M+1}^{D} \lambda_i$$

where:

$\mathbf{U}_M$ - is a $D \times M$ matrix whose columns are eigenvectors of $\mathbf{S}$

$\mathbf{L}_M$ - is a $M \times M$ diagonal matrix whose elements are eigenvalues of $\mathbf{S}$

$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{x}_n - \overline{\mathbf{x}}\right) \left(\mathbf{x}_n - \overline{\mathbf{x}}\right)^{\mathrm{T}}$ - is the sample covariance matrix

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

It is possible to calculate the solution analytically:

$$\mathbf{W}_{\mathrm{ML}} = \mathbf{U}_M \left(\mathbf{L}_M - \sigma^2 \mathbf{I}\right)^{1/2} \underbrace{\mathbf{R}}_{\text{any orthogonal matrix}}$$

$$\sigma^2_{\mathrm{ML}} = \underbrace{\frac{1}{D-M} \sum_{i=M+1}^{D} \lambda_i}_{\text{The average variance of discarded dimensions.}}$$

where:

$\mathbf{U}_M$ - is a $D \times M$ matrix whose columns are eigenvectors of $\mathbf{S}$

$\mathbf{L}_M$ - is a $M \times M$ diagonal matrix whose elements are eigenvalues of $\mathbf{S}$

$\mathbf{S} = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{x}_n - \overline{\mathbf{x}}\right) \left(\mathbf{x}_n - \overline{\mathbf{x}}\right)^{\mathrm{T}}$ - is the sample covariance matrix

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

- If we use the **eigendecomposition** of the sample covariance matrix, then we can simply take $\mathbf{R} = \mathbf{I}$.

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

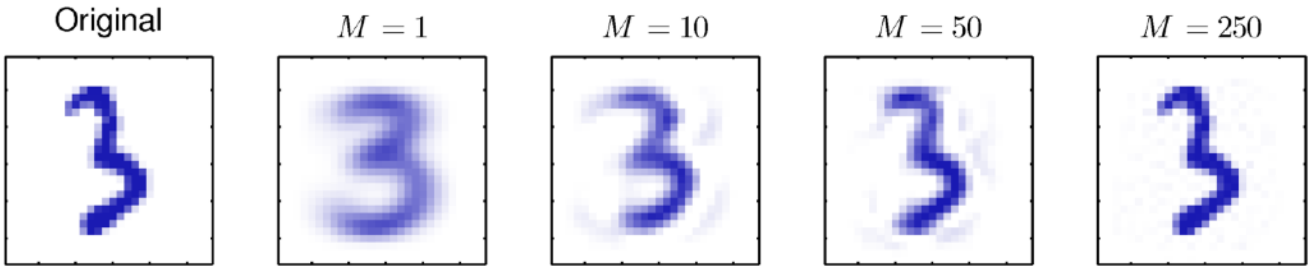
- If we use the **eigendecomposition** of the sample covariance matrix, then we can simply take $\mathbf{R} = \mathbf{I}$.
- In practice, the complexity of the eigendecomposition is $O(D^3)$, and the complexity of calculating the covariance matrix is $O(ND^2)$.

LEARNING PPCA: THE LIKELIHOOD FUNCTION (IID DATA)

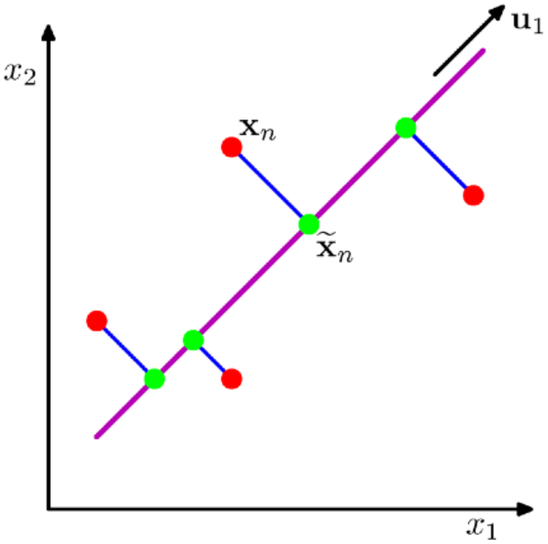
- If we use the **eigendecomposition** of the sample covariance matrix, then we can simply take $\mathbf{R} = \mathbf{I}$.
- In practice, the complexity of the eigendecomposition is $O(D^3)$, and the complexity of calculating the covariance matrix is $O(ND^2)$.
- If we have large problems (i.e., $D > 1000$, $N > 1000$), then we can use:
  - **Expectation-Maximization** (EM)
  - **Gradient-based optimization** (e.g., SGD).
- For numerical algorithms, $\mathbf{R}$ could be arbitrary, so **no unique solution**.

PROBABILISTIC PCA

Reconstruction:



Projection of 2D data onto 1D space:



Advantages

✓ Exact likelihood.

✓ Analytical solution.

✓ Posterior over **z** is analytical.

✓ Allows compression.

✓ Allows to generate.

Disadvantages

- Linear transformation drastically limits the applicability.
- For more complex data, pPCA requires M close to D to work well.
- No analytical solution for binary data.

GENERATIVE MODELS

	Training	Likelihood	Sampling	Compression
Autoregressive models (e.g., PixelCNN)	Stable	Exact	Slow	No
Flow-based models (e.g., RealNVP)	Stable	Exact	Fast/Slow	No
Implicit models (e.g., GANs)	Unstable	No	Fast	No
Prescribed models (e.g., VAEs)	Stable	Approximate	Fast	Yes

Generative process:

$$1.\ \mathbf{z} \sim p_{\lambda}(\mathbf{z})$$

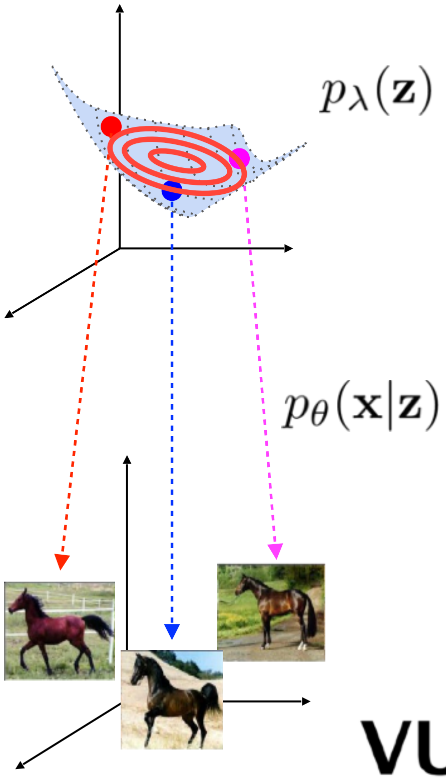
$$2.\ \mathbf{x} \sim p_{\theta}(\mathbf{x}|\mathbf{z})$$

The log-likelihood function:

$$\log p_{\vartheta}(\mathbf{x}) = \log \int p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z}) \mathrm{d}\mathbf{z}$$

How to train such model efficiently?

Now we consider non-linear transformations.



LATENT-VARIABLE MODELS WITH NON-LINEAR TRANSFORMATIONS

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0,\mathbf{I})$.

For the pPCA: $p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$

LATENT-VARIABLE MODELS WITH NON-LINEAR TRANSFORMATIONS

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0,\mathbf{I})$.

For the pPCA: $p(\mathbf{x}\,|\,\mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$

Now, we consider: $p(\mathbf{x}\,|\,\mathbf{z}) = \mathcal{N}(f(\mathbf{z};\mathbf{W}), \sigma^2\mathbf{I})$.

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$.

For the pPCA: $p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$

Now, we consider: $p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(f(\mathbf{z}; \mathbf{W}), \sigma^2\mathbf{I})$.

Since f could be any non-linear transformation, Prof. Bishop cannot provide us any tricks to solve the integral:

$$p(\mathbf{x}) = \int p(\mathbf{x} | \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}$$

This is an infinite mixture of Gaussians.

Let us assume: $p(\mathbf{z}) = \mathcal{N}(0,\mathbf{I})$.

For the pPCA: $p(\mathbf{x} \,|\, \mathbf{z}) = \mathcal{N}(\mathbf{W}\mathbf{z} + \mu, \sigma^2\mathbf{I})$

Now, we consider: $p(\mathbf{x} \,|\, \mathbf{z}) = \mathcal{N}(f(\mathbf{z}; \mathbf{W}), \sigma^2\mathbf{I})$.

Since f could be any non-linear transformation, Prof. Bishop cannot provide us any tricks to solve the integral:

$$p(\mathbf{x}) = \int p(\mathbf{x} \,|\, \mathbf{z}) p(\mathbf{z}) \mathrm{d}\mathbf{z}$$

This is an infinite mixture of Gaussians.

BUT we can use variational inference!
(Chapter 10 in Bishop's book 😉)

VARIATIONAL INFERENCE FOR LATENT VARIABLE MODELS

$$\begin{aligned}
\log p_\vartheta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] - \mathrm{KL}\Big(q_\phi(\mathbf{z}|\mathbf{x})||p_\lambda(\mathbf{z})\Big)
\end{aligned}$$

Variational posterior

$$
\begin{aligned}
\log p_{\vartheta}(\mathbf{x}) &= \log \int p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})} p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}\Big[\log p_{\theta}(\mathbf{x}|\mathbf{z})\Big] - \mathrm{KL}\Big(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\lambda}(\mathbf{z})\Big)
\end{aligned}
$$

$$
\begin{aligned}
\log p_{\vartheta}(\mathbf{x}) &= \log \int p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})} p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}\Big[\log p_{\theta}(\mathbf{x}|\mathbf{z})\Big] - \mathrm{KL}\Big(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\lambda}(\mathbf{z})\Big)
\end{aligned}
$$

Variational posterior

We can learn a separate *q* for each **x**, but it would be too complicated. Therefore, we use **amortization**.

$$
\begin{aligned}
\log p_\vartheta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] - \mathrm{KL}\Big(q_\phi(\mathbf{z}|\mathbf{x}) || p_\lambda(\mathbf{z})\Big)
\end{aligned}
$$

Jensen's inequality

$\log \mathbb{E}_q[\ldots] \geq \mathbb{E}_q[\log \ldots]$

VARIATIONAL INFERENCE FOR LATENT VARIABLE MODELS

$$
\begin{aligned}
\log p_\vartheta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x}|\mathbf{z})\Big] - \mathrm{KL}\Big(q_\phi(\mathbf{z}|\mathbf{x})||p_\lambda(\mathbf{z})\Big)}_{\text{Evidence Lower BOund (ELBO)}}
\end{aligned}
$$

$$
\begin{aligned}
\log p_{\vartheta}(\mathbf{x}) &= \log \int p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&= \log \int \frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})} p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z} \\
&\geq \int q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z} \\
&= \underbrace{\mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}\Big[\log p_{\theta}(\mathbf{x}|\mathbf{z})\Big]}_{\text{Reconstruction error (RE)}} - \underbrace{\mathrm{KL}\Big(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\lambda}(\mathbf{z})\Big)}_{\text{Regularization (KL)}}
\end{aligned}
$$

VARIATIONAL INFERENCE FOR LATENT VARIABLE MODELS

$$
\begin{aligned}
\log p_\vartheta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z}) d\mathbf{z} \\
&= \log \int \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z}) d\mathbf{z} \\
&\geq \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z}) \right] - \mathrm{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}) || p_\lambda(\mathbf{z}) \right)
\end{aligned}
$$

decoder

encoder

marginal
(prior)

$$\log p_{\vartheta}(\mathbf{x}) = \log \int p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z}$$

$$= \log \int \frac{q_{\phi}(\mathbf{z}|\mathbf{x})}{q_{\phi}(\mathbf{z}|\mathbf{x})} p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})\mathrm{d}\mathbf{z}$$

$$\geq \int q_{\phi}(\mathbf{z}|\mathbf{x}) \log \frac{p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})}{q_{\phi}(\mathbf{z}|\mathbf{x})} \mathrm{d}\mathbf{z}$$

$$= \mathbb{E}_{\mathbf{z}\sim q_{\phi}(\mathbf{z}|\mathbf{x})}\left[\log p_{\theta}(\mathbf{x}|\mathbf{z})\right] - \mathrm{KL}\left(q_{\phi}(\mathbf{z}|\mathbf{x})||p_{\lambda}(\mathbf{z})\right)$$

decoder

encoder

marginal
(prior)

= Variational Auto-Encoder

VARIATIONAL INFERENCE FOR LATENT VARIABLE MODELS (A DIFFERENT DERIVATION)

$$
\begin{aligned}
\ln p(\mathbf{x}) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x})\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{z}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}|\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z}|\mathbf{x})}\frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}|\mathbf{z}) - \ln \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})} + \ln \frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}|\mathbf{z})\right] - KL\left[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z})\right] + KL\left[q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}|\mathbf{x})\right]
\end{aligned}
$$

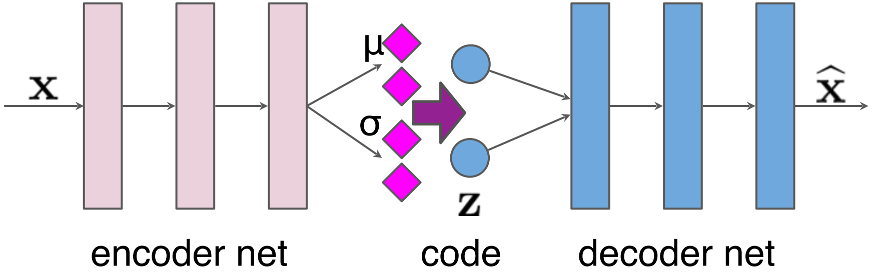
$$
\begin{aligned}
\ln p(\mathbf{x}) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x})\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{z}\,|\,\mathbf{x})p(\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}\,|\,\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}\,|\,\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}\frac{q(\mathbf{z}\,|\,\mathbf{x})}{q(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{x})}\frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z}) - \ln \frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z})} + \ln \frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z})\right] - KL\left[q(\mathbf{z}\,|\,\mathbf{x})\|p(\mathbf{z})\right]}_{\textbf{ELBO}} + \underbrace{KL\left[q(\mathbf{z}\,|\,\mathbf{x})\|p(\mathbf{z}\,|\,\mathbf{x})\right]}_{\geq 0}
\end{aligned}
$$

$$
\begin{aligned}
\ln p(\mathbf{x}) &= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x})\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{z}\,|\,\mathbf{x})p(\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}\,|\,\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln \frac{p(\mathbf{x}\,|\,\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}\,|\,\mathbf{x})}\frac{q(\mathbf{z}\,|\,\mathbf{x})}{q(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z})\frac{p(\mathbf{z})}{q(\mathbf{z}\,|\,\mathbf{x})}\frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z}) - \ln \frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z})} + \ln \frac{q(\mathbf{z}\,|\,\mathbf{x})}{p(\mathbf{z}\,|\,\mathbf{x})}\right] \\
&= \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}\left[\ln p(\mathbf{x}\,|\,\mathbf{z})\right] - KL\left[q(\mathbf{z}\,|\,\mathbf{x})\|p(\mathbf{z})\right]}_{\text{ELBO}} + KL\left[q(\mathbf{z}\,|\,\mathbf{x})\|p(\mathbf{z}\,|\,\mathbf{x})\right]
\end{aligned}
$$

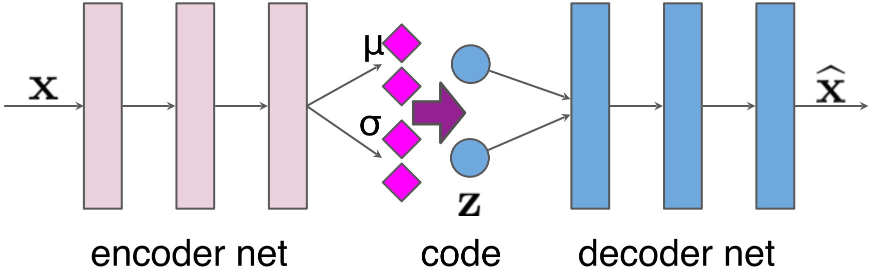
≥ 0

If variational posterior is poorly chosen, then the lower bound is very loose.

Variational posterior (**encoder**) and the likelihood function (**decoder**) are parameterized by neural networks.



Variational posterior (**encoder**) and the likelihood function (**decoder**) are parameterized by neural networks.



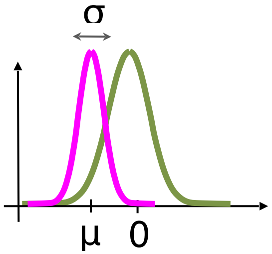
Reparameterization trick:

move the stochasticity to independent random variables

$$z = f(\theta, \varepsilon), \quad \varepsilon \sim p(\varepsilon)$$

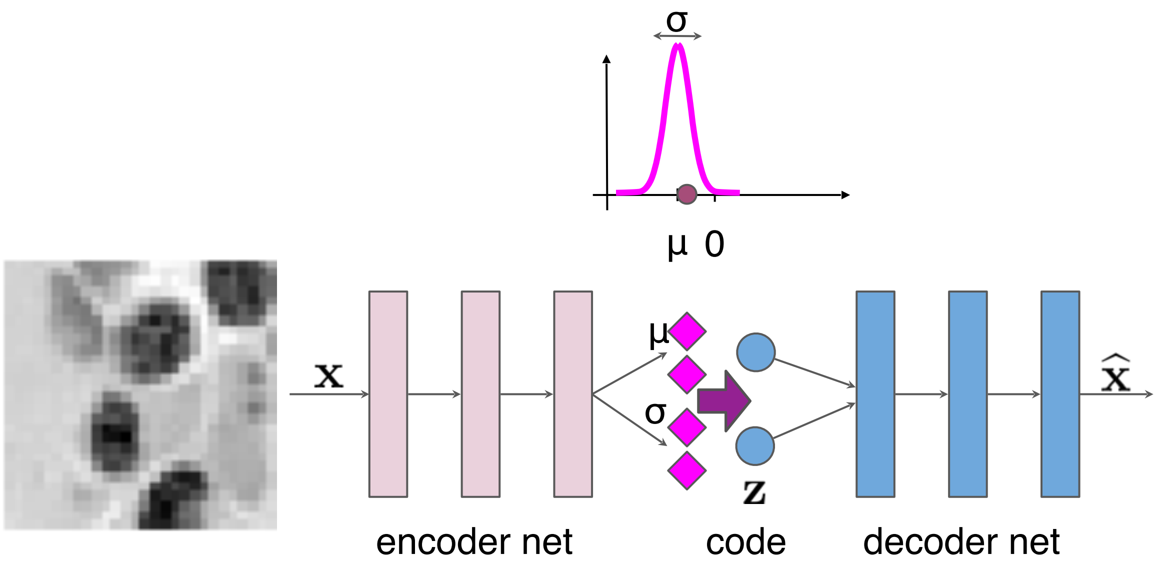
e.g.

$$z = \mu + \sigma \cdot \varepsilon, \quad \varepsilon \sim \mathcal{N}(0,1)$$



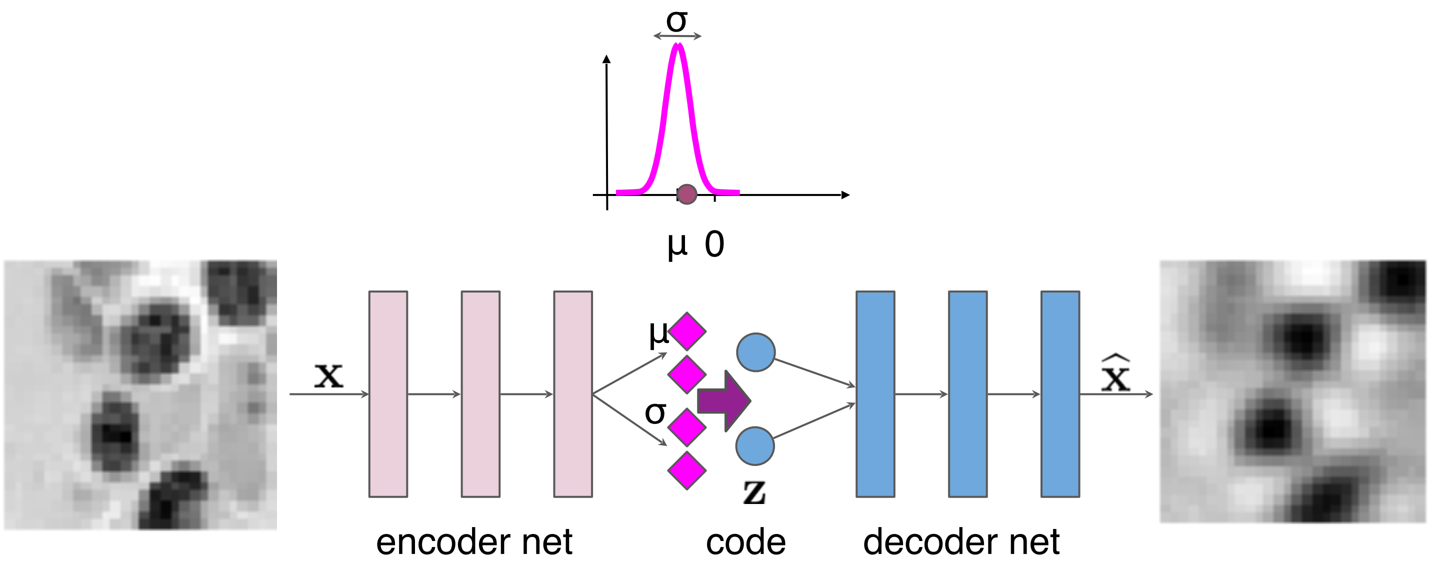
VAE copies input to output through a **bottleneck**.

VAE learns a **code** of the data.



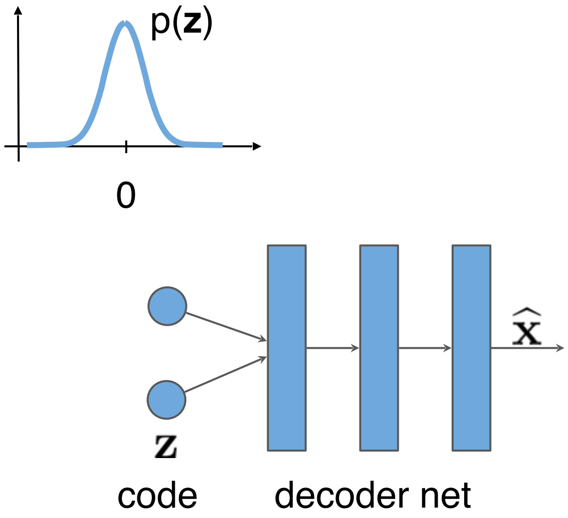
VAE copies input to output through a **bottleneck**.

VAE learns a **code** of the data.



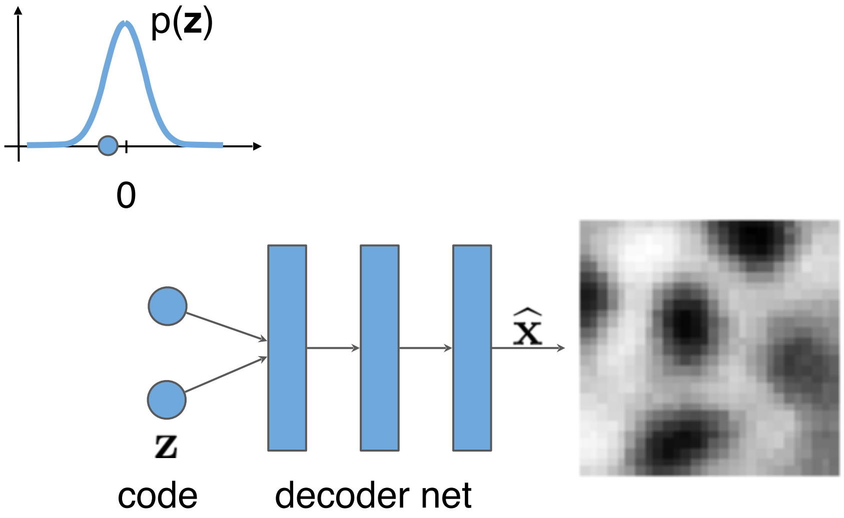
VAE has a **marginal** on the latent code.

VAE can **generate** new data.



VAE has a **marginal** on the latent code.

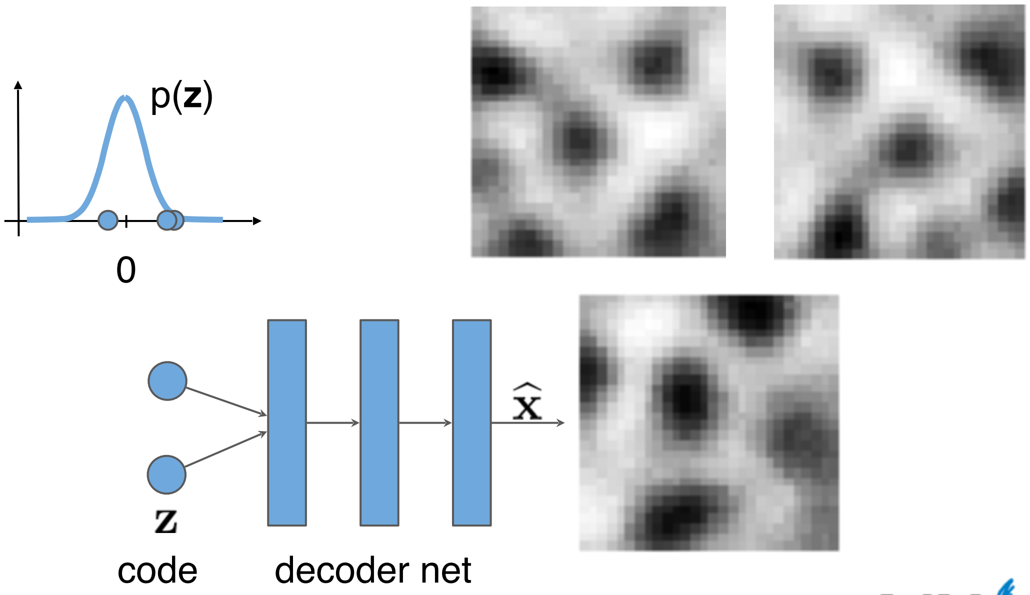
VAE can **generate** new data.



VARIATIONAL AUTO-ENCODERS

VAE has a **marginal** on the latent code.

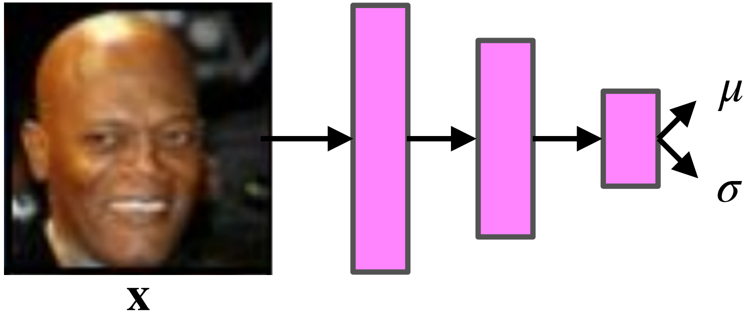
VAE can **generate** new data.



VANILLA VAE

$\mathbf{x}$

VANILLA VAE

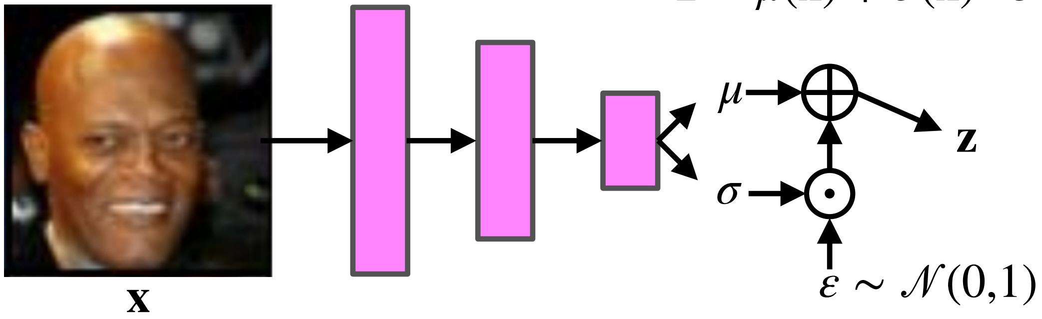


Example architecture for the encoder:

$\mathbf{x}$ -> Linear(D, 300) -> ReLU -> Linear(300, 2M) -> split to 2 vectors

VANILLA VAE

$$\mathbf{z} = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \cdot \varepsilon$$



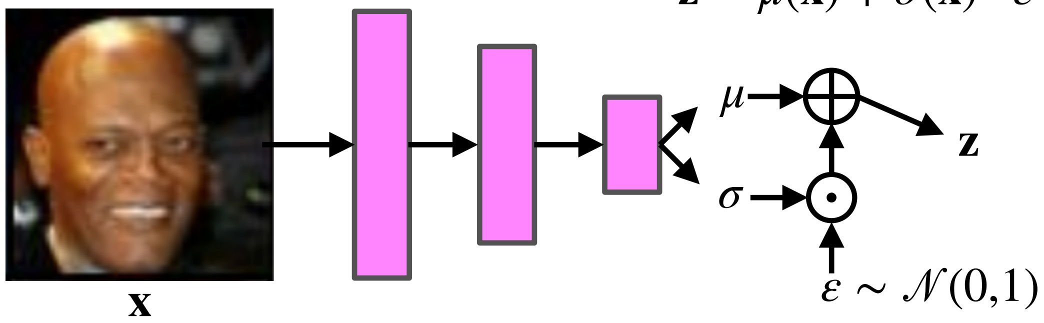
Reparameterization trick!

Example architecture for the encoder:

x -> Linear(D, 300) -> ReLU -> Linear(300, 2*M*) -> split to 2 vectors

VANILLA VAE

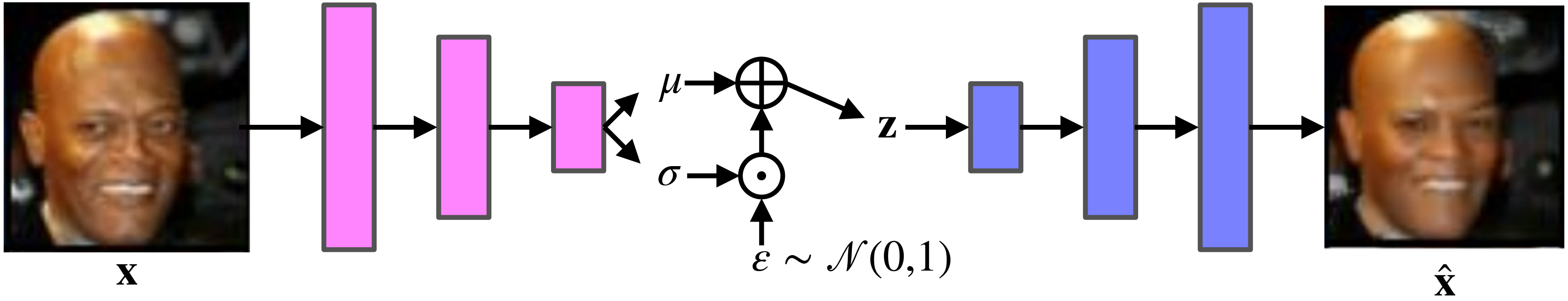
$$\mathbf{z} = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \cdot \varepsilon$$



Example architecture for the encoder:

x -> Linear(D, 300) -> ReLU -> Linear(300, 2*M*) -> split to 2 vectors

No non-linearity here!
We model means and log-std
for Gaussian.



Example architecture for the encoder:

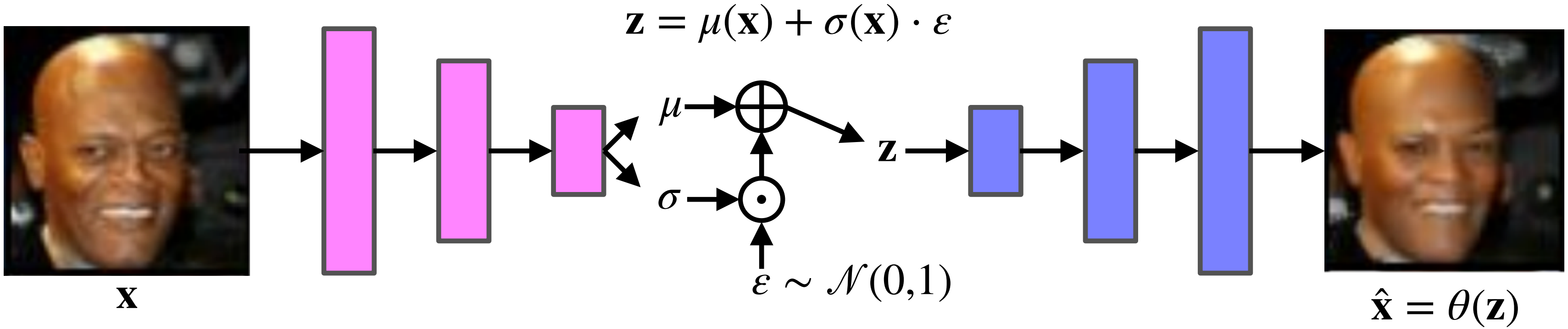
x -> Linear(D, 300) -> ReLU -> Linear(300, 2*M*) -> split to 2 vectors

Example architecture for the decoder:

z -> Linear(*M*, 300) -> ReLU -> Linear(300, *D*) -> means

No non-linearity here!
We model means only.

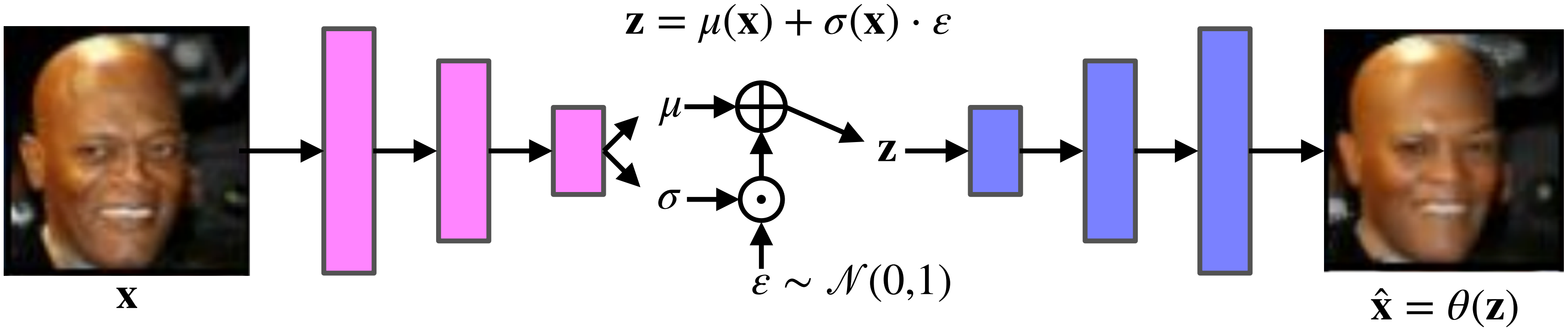
VANILLA VAE



We approximate expected values using a single sample:

$$ELBO = \underbrace{\ln \mathcal{N}(\mathbf{x} \mid \theta(\mathbf{z}), 1)}_{p_\theta(\mathbf{x}|\mathbf{z})} - \left[\underbrace{\ln \mathcal{N}(\mathbf{z} \mid \mu(\mathbf{x}), \sigma^2(\mathbf{x}))}_{q_\phi(\mathbf{z}|\mathbf{x})} - \underbrace{\ln \mathcal{N}(\mathbf{z} \mid 0, 1)}_{p_\lambda(\mathbf{z})} \right]$$

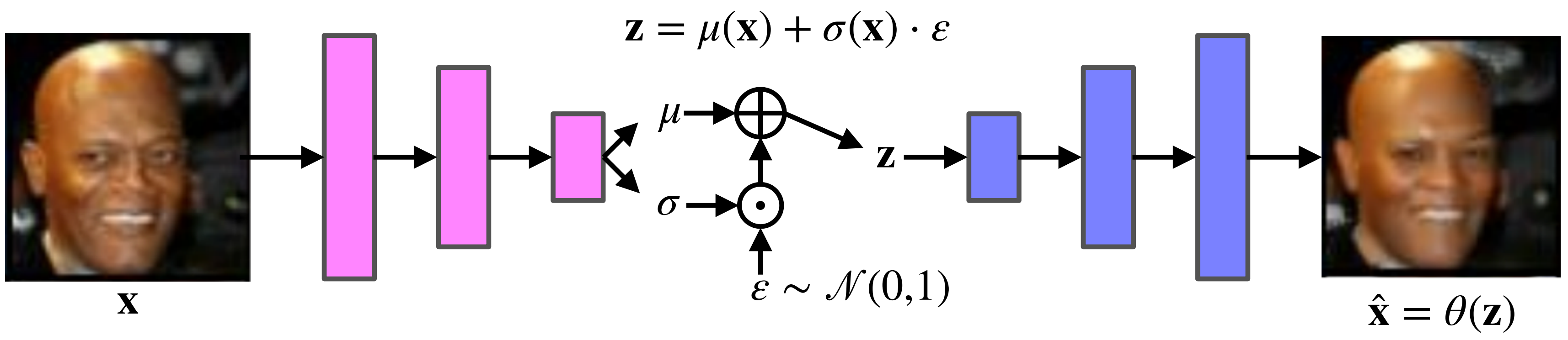
VANILLA VAE



We approximate expected values using a single sample:

$$ELBO = \underbrace{\ln \mathcal{N}(\mathbf{x} | \theta(\mathbf{z}), 1)}_{\text{RE}} - \underbrace{\left[\ln \mathcal{N}(\mathbf{z} | \mu(\mathbf{x}), \sigma^2(\mathbf{x})) - \ln \mathcal{N}(\mathbf{z} | 0, 1)\right]}_{\text{KL}}$$

VANILLA VAE



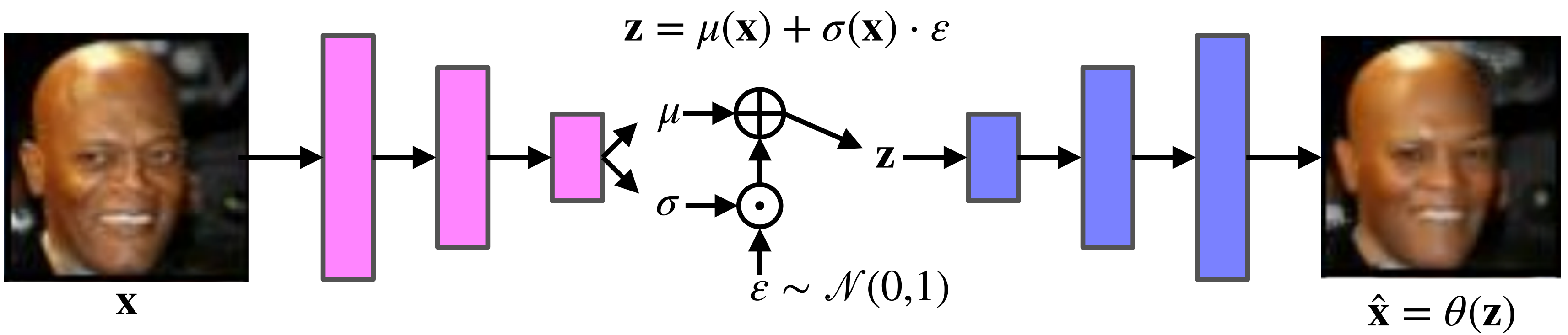
We approximate expected values using a single sample:

We assume a Gaussian variational posterior.

$$ELBO = \underbrace{\ln \mathcal{N}(\mathbf{x}|\theta(\mathbf{z}),1)}_{\text{RE}} - \underbrace{\left[\ln \mathcal{N}(\mathbf{z}|\mu(\mathbf{x}),\sigma^2(\mathbf{x})) - \ln \mathcal{N}(\mathbf{z}|0,1)\right]}_{\text{KL}}$$

We assume a standard Gaussian prior.

VANILLA VAE



We approximate expected values using a single sample:

$$ELBO = \underbrace{\ln \mathcal{N}(\mathbf{x} \,|\, \theta(\mathbf{z}), 1)}_{\text{RE}} - \underbrace{\left[\ln \mathcal{N}(\mathbf{z} \,|\, \mu(\mathbf{x}), \sigma^2(\mathbf{x})) - \ln \mathcal{N}(\mathbf{z} \,|\, 0, 1) \right]}_{\text{KL}}$$

REMEMBER! We cannot pick an arbitrary distribution. We must choose a distribution that is appropriate for our data.
Real-valued -> e.g., Gaussian
Binary -> Bernoulli

VANILLA VAE

```
import torch.nn as nn

class VAE(nn.Module):
    def __init__(self, D, M):
        super(VAE, self).__init__()
        self.D = D
        self.M = M

        self.enc1 = nn.Linear(in_features=self.D, out_features=300)
        self.enc2 = nn.Linear(in_features=300, out_features=self.M*2)

        self.dec1 = nn.Linear(in_features=self.M, out_features=300)
        self.dec2 = nn.Linear(in_features=300, out_features=self.D)

    def reparameterize(self, mu, log_std):
        std = torch.exp(log_std)
        eps = torch.randn_like(std)
        z = mu + (eps * std)
        return z
```

VANILLA VAE

```
def forward(self, x):
    # encoder
    x = nn.functional.relu(self.enc1(x))
    x = self.enc2(x).view(-1, 2, self.M)

    # get mean and log-std
    mu = x[:, 0, :]
    log_std = x[:, 1, :]

    # reparameterization
    z = self.reparameterize(mu, log_std)

    # decoder
    x_hat = nn.functional.relu(self.dec1(z))
    x_hat = self.dec2(x)
    return x_hat, mu, log_std
```

VANILLA VAE

```
def elbo(self, x, x_hat, z, mu, log_std):
    # reconstruction error
    RE = nn.loss.mse(x, x_hat)

    # kl-regularization
    # We assume here that log_normal is implemented
    KL = log_normal(z, mu, log_std) - log_normal(z, 0, 1)

    # REMEMBER! We maximize ELBO, but optimizers minimize.
    # Therefore, we need to take the negative sign!
    return -(RE - KL)
```

COMMON ISSUES WITH VAES

$$q_\phi(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})$$

Weak **decoders** $\rightarrow$ bad generations/reconstructions

Weak **encoders** $\rightarrow$ bad latent representation, *posterior collapse* (variational posterior = prior).

Weak **marginals** $\rightarrow$ bad generations

Variational **posteriors** $\rightarrow$ what family of distributions?

COMPONENTS OF VAES

$$q_\phi(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})$$

ResNets, DenseNets
DRAW
Autoregressive models
Normalizing flows

COMPONENTS OF VAES

$$q_\phi(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})$$

ResNets, DenseNets
Normalizing flows
Hyperspherical dist.
Hyperbolic-normal dist.
Group theory

ResNets, DenseNets
DRAW
Autoregressive models
Normalizing flows

COMPONENTS OF VAES

$$q_{\phi}(\mathbf{z}|\mathbf{x}) \propto p_{\theta}(\mathbf{x}|\mathbf{z})\ p_{\lambda}(\mathbf{z})$$

ResNets, DenseNets
Normalizing flows
Hyperspherical dist.
Hyperbolic-normal dist.
Group theory

ResNets, DenseNets
DRAW
Autoregressive models
Normalizing flows

Autoregressive models
Normalizing flows
VampPrior
Implicit prior

COMPONENTS OF VAES

$$q_\phi(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})$$

ResNets, DenseNets
Normalizing flows
Hyperspherical dist.
Hyperbolic-normal dist.
Group theory

ResNets, DenseNets
DRAW
Autoregressive models
Normalizing flows

Autoregressive models
Normalizing flows
VampPrior
Implicit prior

$$\mathrm{ELBO}(\mathbf{x}; \theta, \phi, \lambda)$$

Adversarial learning
MMD
Wasserstein AE

COMPONENTS OF VAES

$$q_\phi(\mathbf{z}|\mathbf{x}) \propto p_\theta(\mathbf{x}|\mathbf{z})\ p_\lambda(\mathbf{z})$$

ResNets, DenseNets
Normalizing flows
Hyperspherica
Hyperbolic-no
Group theory

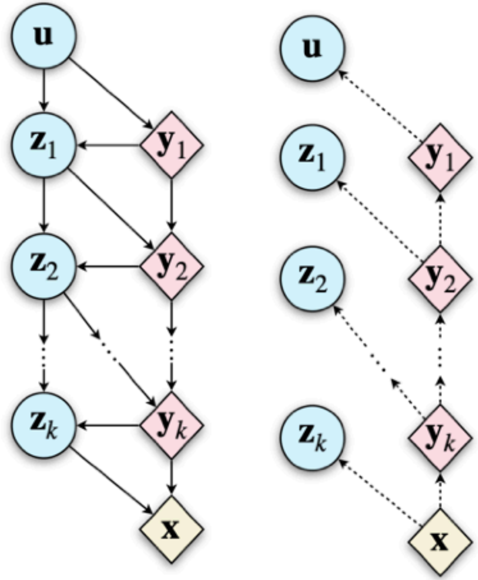
ResN
ssive models
Normalizing flows

gressive models
Normalizing flows
VampPrior
Implicit prior

Hierarchical VAEs!

$$\mathrm{ELBO}(\mathbf{x}; \theta, \phi, \lambda)$$

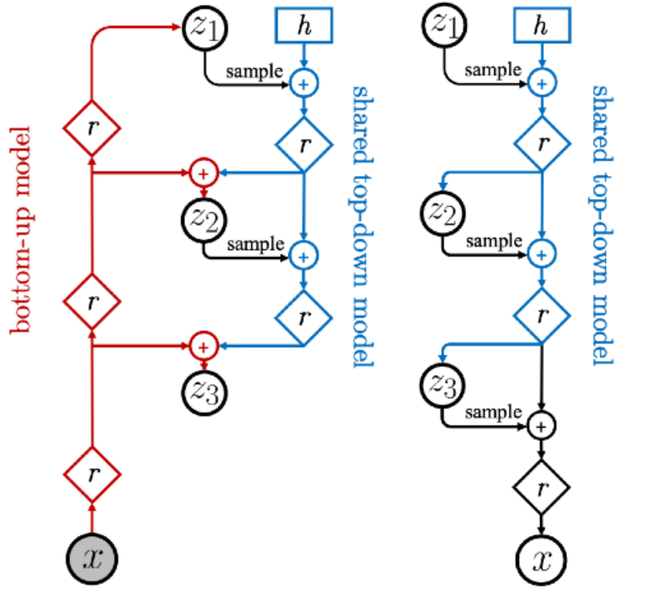
Adversarial learning
MMD
Wasserstein AE



i) Generative Model ii) Inference Model

Figure 4: Hierarchical selfVAE.

Gatopoulos, I., & Tomczak, J.M. (2020).
Self-supervised Variational Auto-Encoders

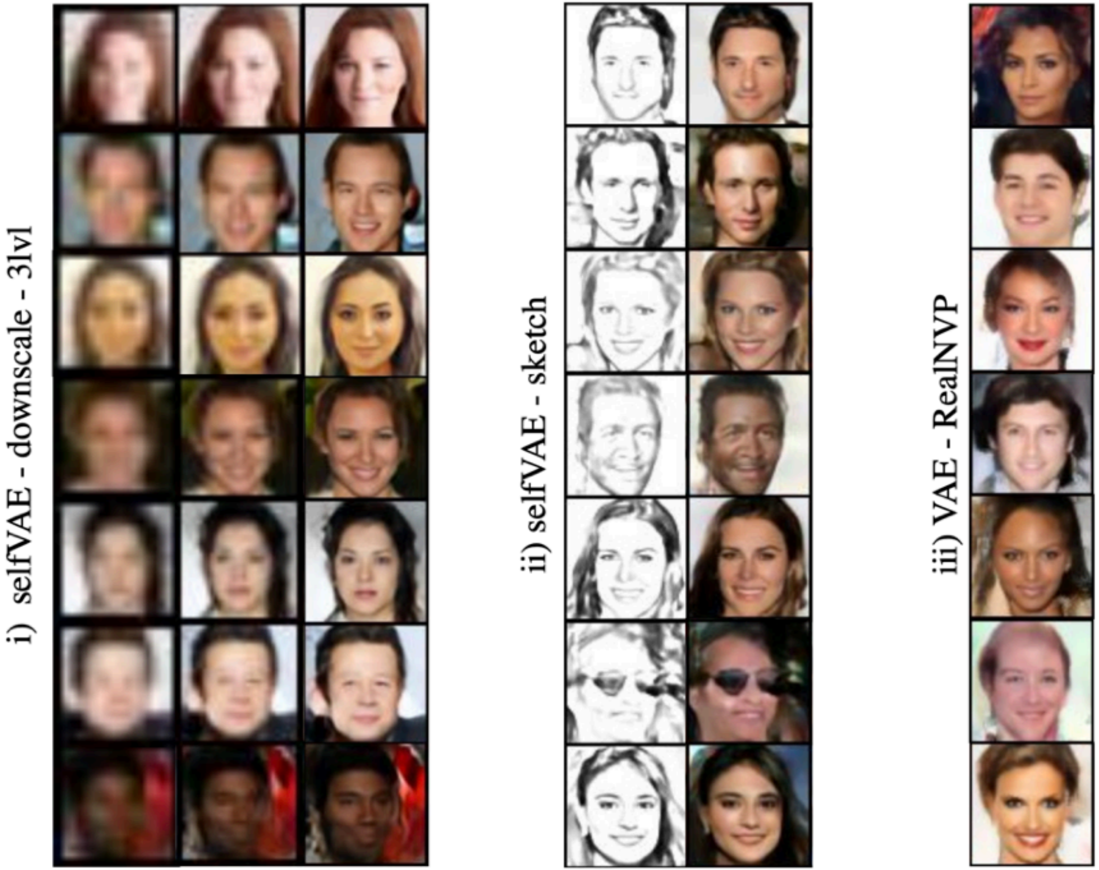


(a) Bidirectional Encoder (b) Generative Model

Figure 2: The neural networks implementing an encoder $q(\boldsymbol{z}|\boldsymbol{x})$ and generative model $p(\boldsymbol{x}, \boldsymbol{z})$ for a 3-group hierarchical VAE. r denotes residual neural networks, $\oplus$ denotes feature combination (e.g., concatenation), and $\boxed{h}$ is a trainable parameter.

Vahdat, A., & Kautz, J. (2020).
Nvae: A deep hierarchical variational autoencoder. *NeurIPS 2020*

HIERARCHICAL VAES



Gatopoulos, I., & Tomczak, J.M. (2020). Self-supervised Variational Auto-Encoders

HIERARCHICAL VAES

Figure 1: 256×256-pixel samples generated by NVAE, trained on CelebA HQ [28].

Vahdat, A., & Kautz, J. (2020). Nvae: A deep hierarchical variational autoencoder. *NeurIPS 2020*

VARIATIONAL AUTO-ENCODERS

Advantages

✓ Non-linear transformations.

✓ Stable training.

✓ Allows compression.

✓ Allows to generation.

✓ The likelihood could be approximated.

Disadvantages

- No analytical solutions.
- No exact likelihood.
- Potential mismatch between true posterior and variational posterior
- Blurry images

VARIATIONAL AUTO-ENCODERS

Advantages

- ✓ Non-linear transformations.
- ✓ Stable training.
- ✓ Allows compression.
- ✓ Allows to generation.
- ✓ The likelihood could be approximated.

Disadvantages

- No analytical solutions.
- No exact likelihood.
- Potential mismatch between true posterior and variational posterior
- ~~Blurry images~~

Thank you!

VU VRIJE UNIVERSITEIT AMSTERDAM