Deep Generative Models

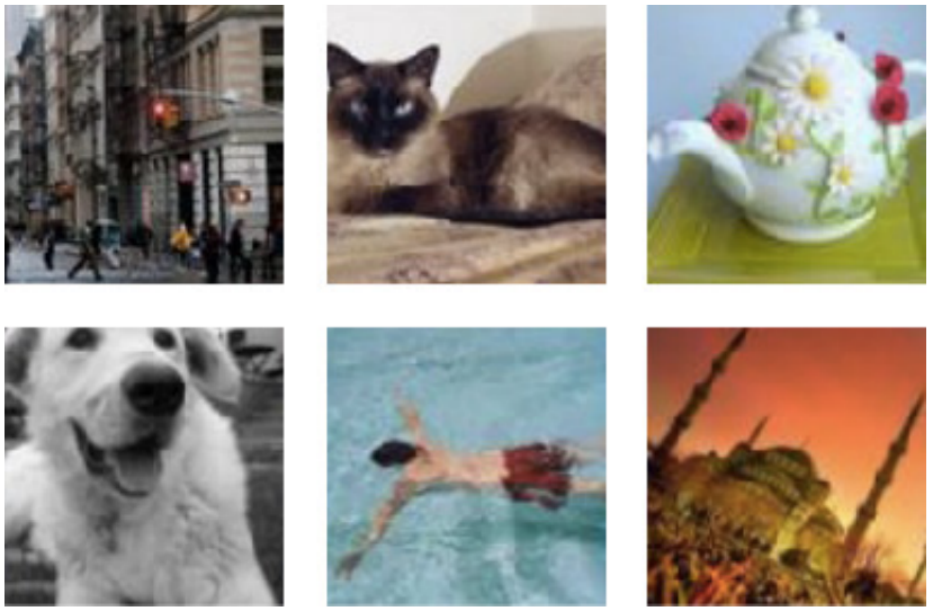
Advanced Statistical Inference

Simone Rossi

Generative models

Given training data from an unknown distribution $p_{\text{data}}(\boldsymbol{x})$, we want to learn a model $p_{\text{model}}(\boldsymbol{x})$ that approximates the true distribution.

Train from $\boldsymbol{x} \sim p_{\text{data}}(\boldsymbol{x})$.

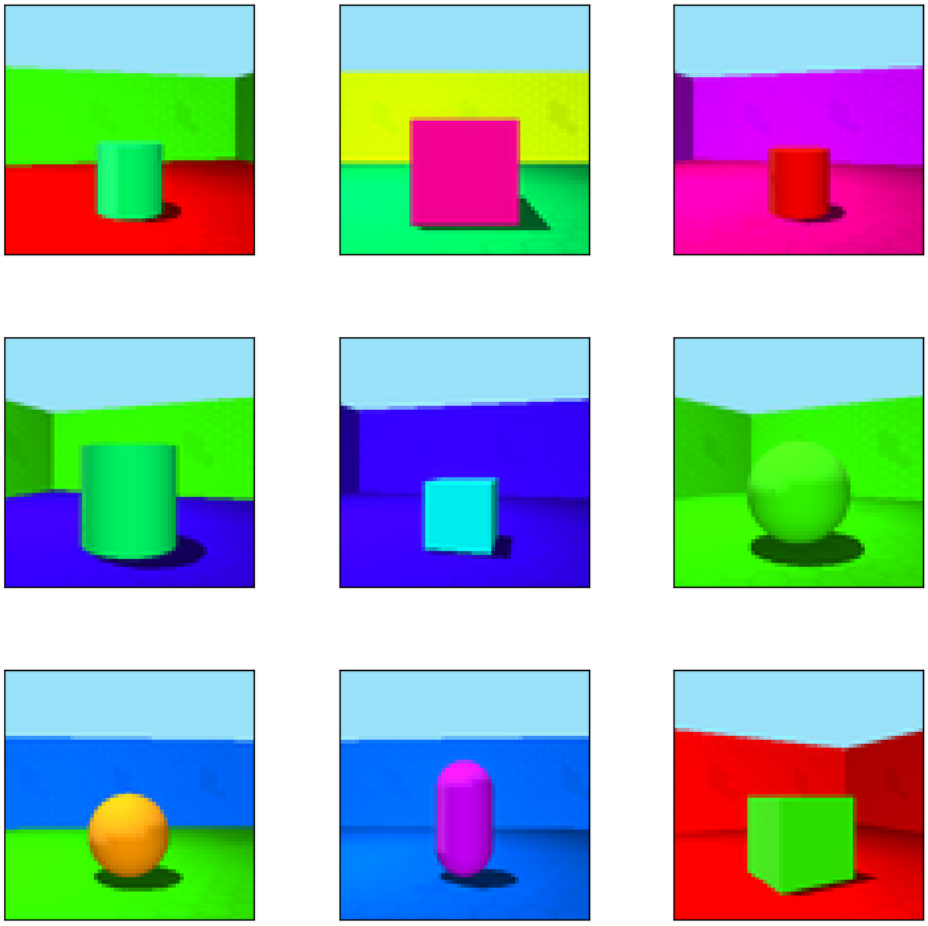


Generate from $\boldsymbol{x} \sim p_{\text{model}}(\boldsymbol{x})$.

Latent variable models

Our objective is to learn the data distribution $p(\boldsymbol{x})$, but we suppose that each data point $\boldsymbol{x}$ is associated with a **latent variable** $\boldsymbol{z}$.

For example, image we want to learn the distribution of images of objects, like the one below:



Each image is a huge vector of pixels $x \in \mathbb{R}^{H \times W \times C}$, where H is the height, W is the width and C is the number of channels (in this case, $64 \times 64 \times 3$).

Each image can be described by a set of 6 latent variables: floor, wall and object color, shape, orientation and scale.

Latent variable models

Latent variables are unobserved variables, we cannot measure them directly, but we can infer them from the observed data.

In math terms, we model our data distribution as a **marginal** of the joint distribution of the observed data and the latent variables:

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \quad \text{or} \quad p(\boldsymbol{x}) = \sum_i p(\boldsymbol{x}, \boldsymbol{z}_i)$$

where $p(\boldsymbol{x}, \boldsymbol{z})$ is the **joint distribution** of the observed data and the latent variables.

Linear latent variable models

Gaussian mixture model (GMM)

Conditional distribution of the data given the latent variable:

$$p(\boldsymbol{x} \mid \boldsymbol{z} = k) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

where $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are the mean and covariance of the k-th Gaussian component.

Prior distribution of the latent variable:

$$p(\boldsymbol{z}) = \text{Categorical}(\boldsymbol{z} \mid \boldsymbol{\pi})$$

where π_k are the mixing coefficients, $\boldsymbol{\mu}_k$ is the mean of the k-th Gaussian component, and $\boldsymbol{\Sigma}_k$ is the covariance of the k-th Gaussian component.

Probabilistic PCA (PPCA)

Conditional distribution of the data given the latent variable:

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{W}\boldsymbol{z} + \boldsymbol{b}, \sigma^2 \boldsymbol{I})$$

where $\boldsymbol{W}$ is a linear transformation matrix, $\boldsymbol{b}$ is a bias vector, and $\sigma^2 \boldsymbol{I}$ is the noise covariance.

Prior distribution of the latent variable:

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

Variational autoencoders (VAEs)

Variational autoencoders (VAEs)

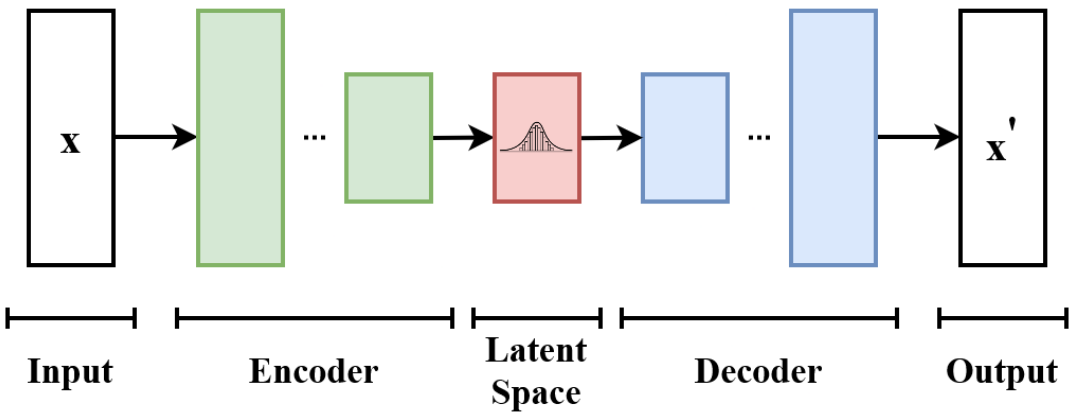
Idea:

1. We want to learn a generative model $p_{\text{model}}(x)$ that approximates the true data distribution $p_{\text{data}}(x)$.
2. We introduce a latent variable z to explain the observed data x.
3. We model the relationship between the data and the latent variable with a more complex function.
4. We use a variational inference approach to learn the model parameters.

Variational autoencoders (VAEs)

Disclaimer:

- Variational autoencoders (VAEs) are often misrepresented as a simple autoencoder with a probabilistic twist.
- This is **wrong** (sort of)! It hides the real nature of the encoder and decoder!



Neural networks as flexible likelihood models

Consider a maximum likelihood training objective for a generative model with latent variables:

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

Objective: increase the flexibility of $p(\boldsymbol{x} \mid \boldsymbol{z})$ to approximate the true data distribution $p_{\text{data}}(\boldsymbol{x})$.

. . .

Solution: define a noisy observation model

$$p(\boldsymbol{x} \mid \boldsymbol{z}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z}), \sigma^2 \boldsymbol{I})$$

where $\boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z})$ is a function of the latent variable $\boldsymbol{z}$ and the model parameters $\boldsymbol{\theta}$.

. . .

Note: $p(\boldsymbol{x} \mid \boldsymbol{z})$ is *not* a deterministic function, it is a probabilistic model that outputs a distribution over the data given the latent variable. If it were deterministic, then $p(\boldsymbol{x}) = 0$ almost everywhere.

. . .

Notation: we will use $p(\boldsymbol{x}; \boldsymbol{\theta})$ and $p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta})$ to remember that these two distributions depend on the model parameters $\boldsymbol{\theta}$.

Neural networks as flexible likelihood models

$p(\boldsymbol{x}; \boldsymbol{\theta}) = \int p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z}$ is well-defined

. . .

- Can we compute it?
  - **No**, $\boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z})$ is very complex, so there is no hope of finding a closed-form solution.

. . .

- Can we maximize it w.r.t. $\boldsymbol{\theta}$?
  - **No**, we cannot compute the integral, so we cannot maximize the likelihood directly.

. . .

Solution: try to maximize a lower bound on the likelihood $\log p(\boldsymbol{x}; \boldsymbol{\theta})$

$$\log \mathbb{E}[X] \geq \mathbb{E}[\log X]$$

Variational inference to the rescue

We can use Jensen's inequality to obtain a lower bound on the log-likelihood.

Suppose we have a distribution $q(\boldsymbol{z})$ (we will see later how to choose it):

$$\begin{aligned}
\log p(\boldsymbol{x}; \boldsymbol{\theta}) &= \log \int p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) p(\boldsymbol{z}) \, \mathrm{d}\boldsymbol{z} \\
&= \log \int q(\boldsymbol{z}) \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{z} \\
&\geq \int q(\boldsymbol{z}) \log \left(\frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) \right) \mathrm{d}\boldsymbol{z} \qquad \text{(Jensen's inequality)} \\
&= \mathbb{E}_{q(\boldsymbol{z})} \frac{p(\boldsymbol{z})}{q(\boldsymbol{z})} + \mathbb{E}_{q(\boldsymbol{z})} \log p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta})
\end{aligned}$$

Variational inference to the rescue

Remember the **Variational Inference** lecture?

We can rewrite the lower bound as:

$$\log p(\boldsymbol{x}; \boldsymbol{\theta}) \geq \mathbb{E}_{q(\boldsymbol{z})} \log p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) - \mathrm{KL}\left(q(\boldsymbol{z}) \| p(\boldsymbol{z})\right)$$

. . .

1. The first term is the expected log-likelihood of the data given the latent variable. Since we assume a Gaussian model:

$$\begin{aligned}
\log p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) &= \log \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z}), \sigma^2 \boldsymbol{I}) \\
&= \log \left(\frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(-\frac{1}{2\sigma^2} \|\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z})\|^2 \right) \right) \\
&= -\frac{1}{2\sigma^2} \|\boldsymbol{x} - \boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z})\|^2 + \text{const}
\end{aligned}$$

. . .

Meaning that we want to minimize the expected reconstruction error between the data $\boldsymbol{x}$ and the model output $\boldsymbol{f}(\boldsymbol{\theta}, \boldsymbol{z})$, when $\boldsymbol{z}$ is sampled from $q(\boldsymbol{z})$.

Variational inference to the rescue

2. The second term is the KL divergence between the variational distribution $q(\boldsymbol{z})$ and the prior distribution $p(\boldsymbol{z})$:

$$\mathrm{KL}\left(q(\boldsymbol{z}) \| p(\boldsymbol{z})\right) = \int q(\boldsymbol{z}) \log \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \, \mathrm{d}\boldsymbol{z}$$

   This term acts as a regularizer, encouraging the variational distribution $q(\boldsymbol{z})$ to be close to the prior distribution $p(\boldsymbol{z})$.

Optimization objective

$$\log p(\boldsymbol{x}; \boldsymbol{\theta}) \geq \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{z})} \log p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) - \mathrm{KL}\left(q(\boldsymbol{z}) \| p(\boldsymbol{z})\right)$$

We would like to maximize the lower bound on the log-likelihood, but we want the bound to be tight

. . .

Remember that the KL divergence is always non-negative, so we can rewrite the objective as:

$$\log p(\boldsymbol{x}; \boldsymbol{\theta}) - \mathcal{L}_{\mathrm{ELBO}}(\boldsymbol{\theta}) = \mathrm{KL}\left(q(\boldsymbol{z}) \| p(\boldsymbol{z} \mid \boldsymbol{x})\right)$$

This means that the closer the variational distribution $q(\boldsymbol{z})$ is to the true posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$, the tighter the bound will be.

Approximate posterior inference

- To fit q, we need to define a **variational distribution** $q(\boldsymbol{z})$ that approximates the true posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$.
- The variational distribution is often chosen to be Gaussian, i.e., $q(\boldsymbol{z}; \boldsymbol{\nu}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I})$, where $\boldsymbol{\nu} = \{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ are the variational parameters.
- Gaussian is good because it is easy to sample from, it has a closed-form KL divergence, and it is possible to use the reparameterization trick to compute gradients of the ELBO w.r.t. the variational parameters $\boldsymbol{\nu}$.

. . .

> **!** Important
>
> The **reparameterization trick** allows us to sample from the variational distribution $q(\boldsymbol{z}; \boldsymbol{\nu})$ in a way that is differentiable w.r.t. the variational parameters $\boldsymbol{\nu}$.
>
> $$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$$

Training a VAE

1. We need to maximize the likelihood $\log p(\boldsymbol{x}; \boldsymbol{\theta})$. But we cannot do it directly.
2. We derive a lower bound on the likelihood, the **evidence lower bound (ELBO)** $\mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta})$, which we can maximize instead. But we need to define a variational distribution $q(\boldsymbol{z})$ that approximates the true posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$.
3. We parametrize the variational distribution $q(\boldsymbol{z}; \boldsymbol{\nu})$ with a Gaussian distribution, where $\boldsymbol{\nu} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$ are the variational parameters.

. . .

At the end, we are left with the following optimization problem:

$$\max_{\boldsymbol{\theta}, \boldsymbol{\nu}} \mathcal{L}_{\text{ELBO}}(\boldsymbol{\theta}, \boldsymbol{\nu}) = \mathbb{E}_{q(\boldsymbol{z}; \boldsymbol{\nu})} \log p(\boldsymbol{x} \mid \boldsymbol{z}; \boldsymbol{\theta}) - \text{KL}\left(q(\boldsymbol{z}; \boldsymbol{\nu}) \| p(\boldsymbol{z})\right)$$

Note:

- The ELBO is a function of both the likelihood parameters $\boldsymbol{\theta}$ and the variational parameters $\boldsymbol{\nu}$.
- The likelihood are **shared** across all data points, while the variational parameters are **per-sample**.

Training a VAE via joint optimization is hard

In classic VI (e.g., Bayesian logistic regression):

- ELBO depends **only** on $q(\boldsymbol{\theta})$
- Optimization is stable — just improve the approximation to the true posterior

In our case, for variational autoencoders:

- We optimize **both**:
  - $\boldsymbol{\theta}$: parameters of the generative model

- $\boldsymbol{\nu}$: variational parameters

- But these two are **coupled**:
  - ELBO depends on **both** sets of parameters
  - Changing $\boldsymbol{\theta}$ shifts the optimal $\boldsymbol{\nu}$, changing $\boldsymbol{\nu}$ affects how we learn $\boldsymbol{\theta}$.
  - No close form solution for $\boldsymbol{\nu}$ given $\boldsymbol{\theta}$ exists (no EM-like solution).

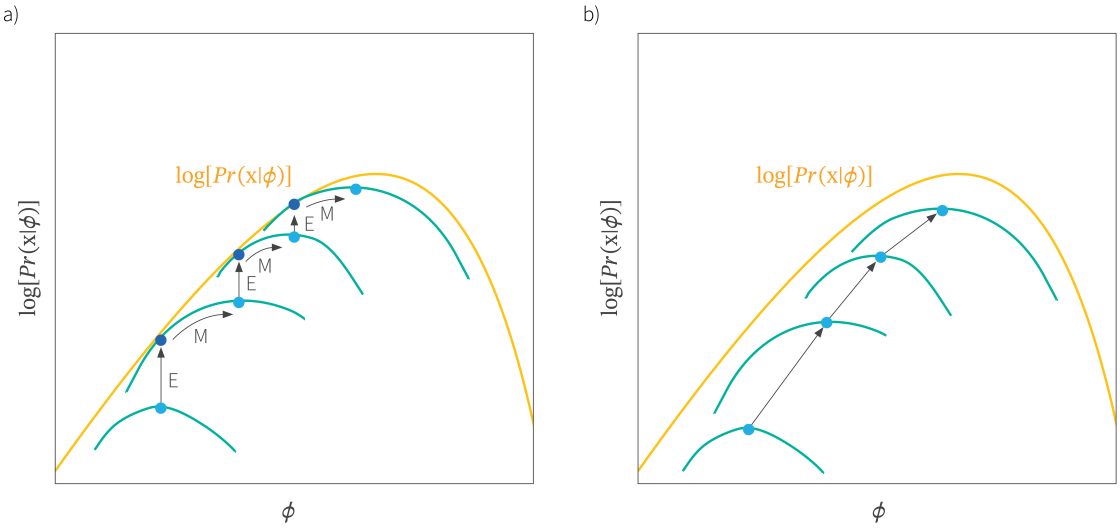
. . .

This may lead to unstable or inefficient training, especially when $\boldsymbol{f}(\cdot)$ is complex (e.g., deep neural networks).

Expectation-Maximization vs Variational Inference

The joint optimization of $\boldsymbol{\theta}$ and $\boldsymbol{\nu}$ in VAEs is similar to the **Expectation-Maximization (EM)** algorithm, but with some key differences:

1. In EM, the E-step computes the exact posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x}; \boldsymbol{\theta})$, while in VAEs, we approximate it with a variational distribution $q(\boldsymbol{z}; \boldsymbol{\nu})$.
2. In EM, the M-step maximizes the exact expected log-likelihood, while in VAEs, we maximize the ELBO, which is a lower bound on the log-likelihood.

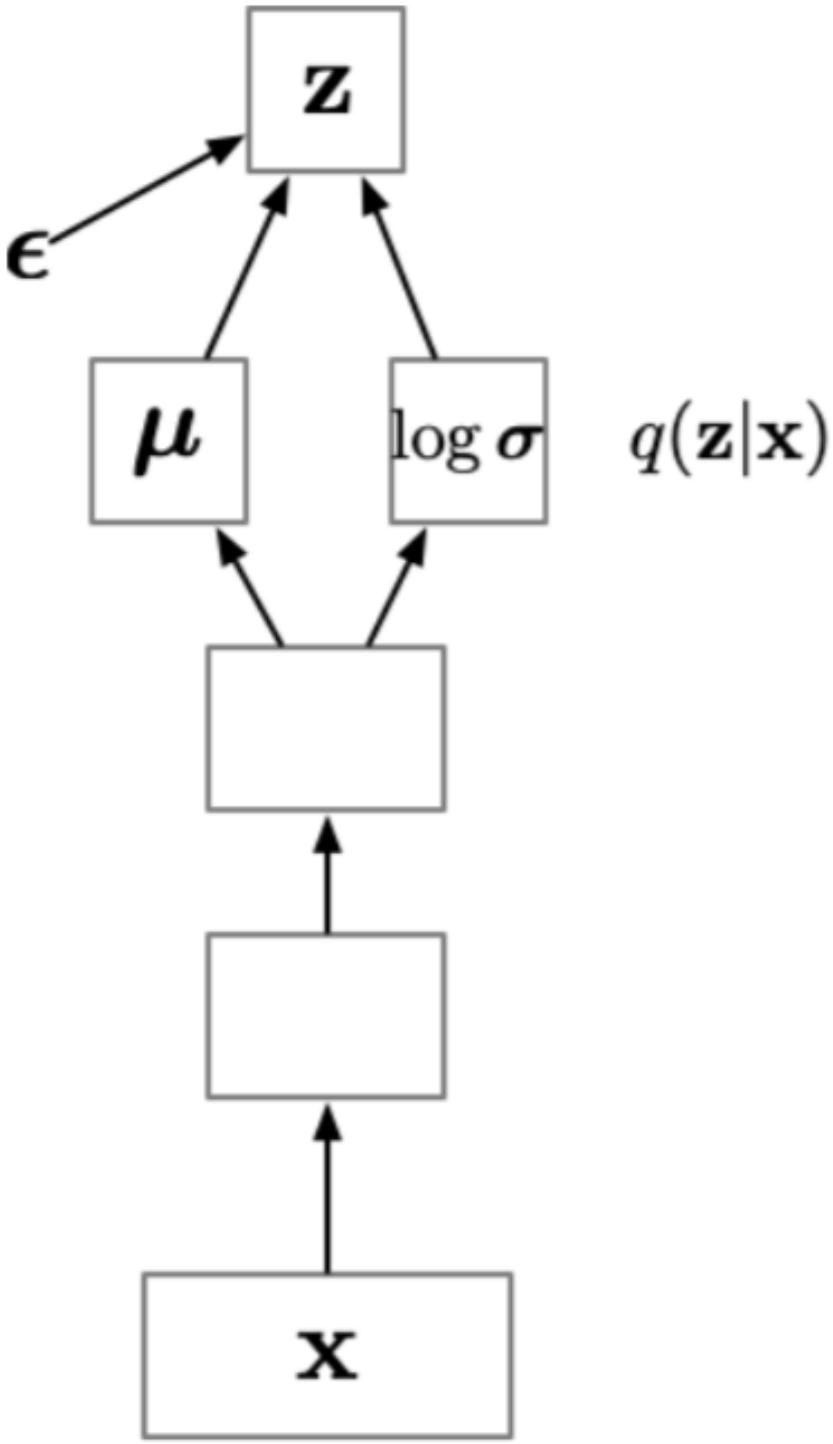


Amortization

- In the previous approach we learn a separate set of variational parameters $\boldsymbol{\nu}$ for each data point $\boldsymbol{x}$.
- This requires an expensive iterative optimization for each data point, which will take a long time to process large datasets.
- **Amortization**: instead of learning a separate set of variational parameters for each data point, we learn a function $\boldsymbol{g}(\boldsymbol{\phi}, \boldsymbol{x})$ that maps the data point $\boldsymbol{x}$ to the variational parameters $\boldsymbol{\nu}$.
- This function is often implemented as a neural network, which allows us to learn a complex mapping from the data to the variational parameters.

Amortization

- **Idea**: "amortize" the cost of computing the variational parameters $\boldsymbol{\nu}$ by learning a function that predicts $\{\boldsymbol{\mu}, \boldsymbol{\sigma}^2\}$ as a function of the data $\boldsymbol{x}$.
- The output of this function is generally $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}$ (the log is needed to ensure that the variance is positive).
- The reparametrization trick is still used to sample from the variational distribution, the only difference is that the variational parameters are not free variables anymore, but are predicted by a neural network $\boldsymbol{g}(\boldsymbol{\phi}, \boldsymbol{x})$.
- Sometimes, this is noted as $q(\boldsymbol{z} \mid \boldsymbol{x})$ to emphasize that the variational distribution depends on the data $\boldsymbol{x}$, even though it's not actually a conditional distribution.

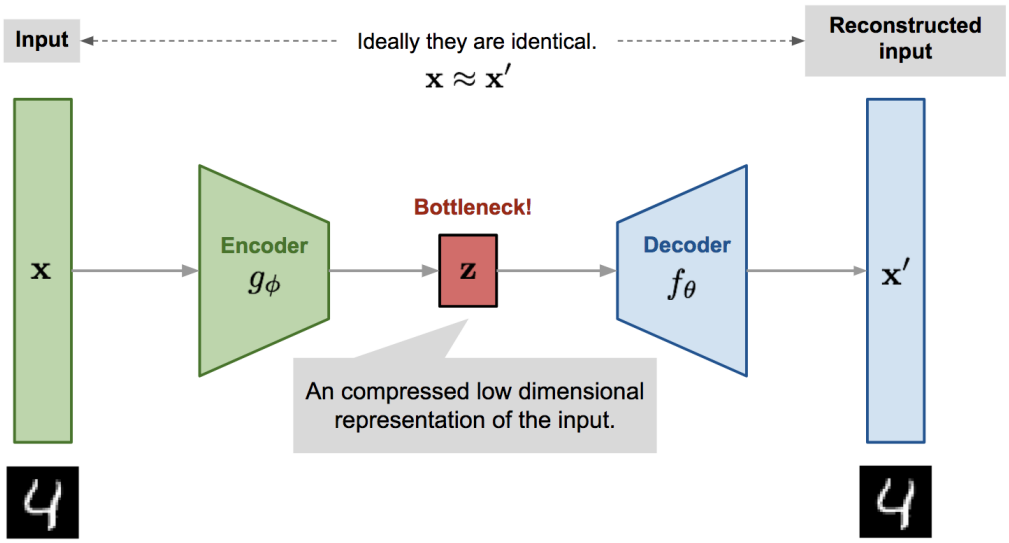

z
ϵ
μ
log σ
q(z|x)
x

Putting it all together

- We have a neural network $g(x; \phi)$ that predicts the variational parameters $\{\mu, \sigma^2\}$ from the data x.
- We can sample from the variational distribution $q(z \mid x; \phi)$ using the reparameterization trick
- We compute the likelihood $\log p(x \mid z; \theta)$ using using a neural network $f(z; \theta)$ that maps the latent variable z to the data space.
- We can now optimize the ELBO with respect to both the likelihood parameters θ and the amortization parameters ϕ:

$$\mathcal{L}_{\text{ELBO}}(\theta, \phi) = \mathbb{E}_{q(z|x;\phi)} \log p(x \mid z; \theta) - \text{KL}\left(q(z \mid x; \phi) \| p(z)\right)$$

Putting it all together



Generating samples from the VAE

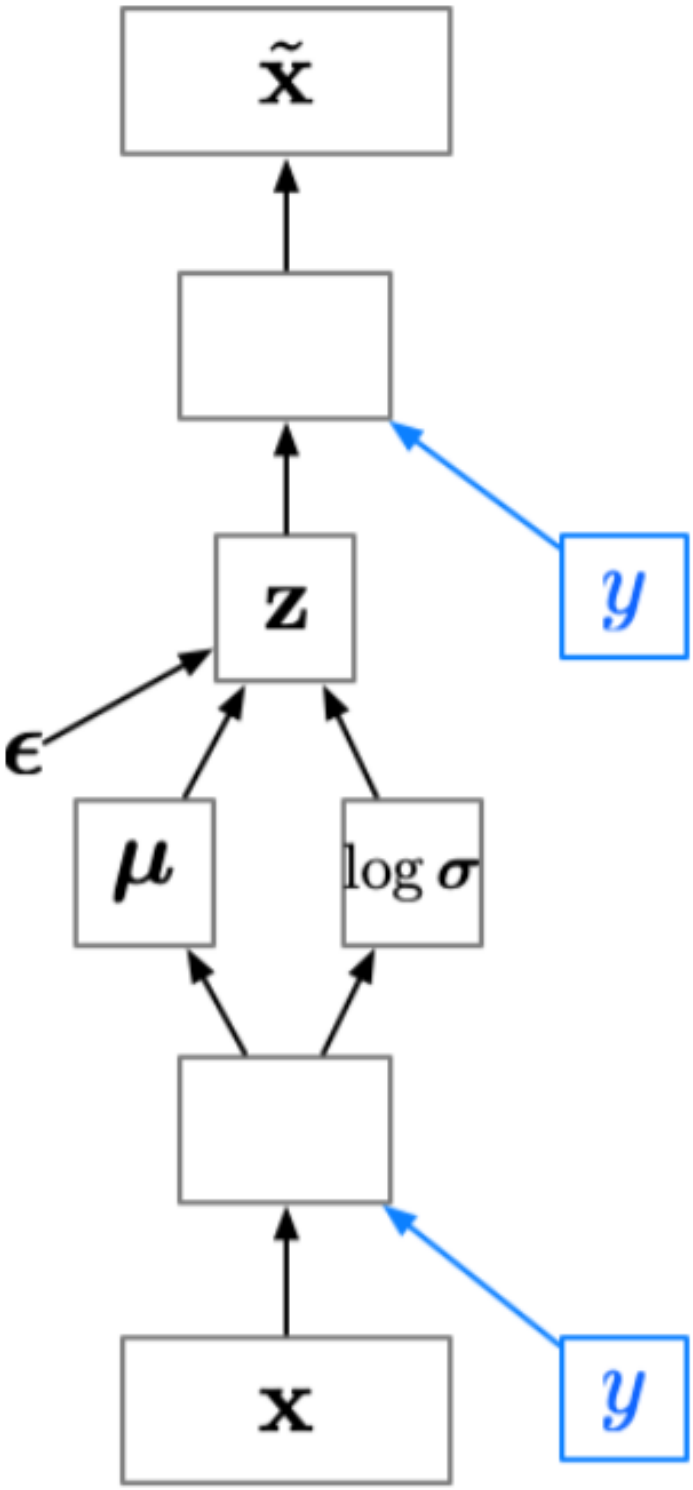
Follow the generation process:

1. Sample from the latent space $z \sim p(z)$, where $p(z)$ is the **prior** distribution.
2. Decode the latent variable z to generate a sample $x \sim p(x \mid z)$ using the decoder network.

> **!** Note
>
> When using amortized inference, the encoder network is not used during generation, since we are not conditioning on any observed data
> It does not make sense to compute $q(\boldsymbol{z} \mid \boldsymbol{x})$ when we don't have any $\boldsymbol{x}$

Conditional VAE (CVAE)

- Sometimes we want to generate samples conditioned on some additional information $p(\boldsymbol{x} \mid \boldsymbol{y})$, e.g., we want to generate images of a specific class.
- Since the latent code $\boldsymbol{z}$ no longer has to model the image category, it can focus on modeling the stylistic features.
- If we're lucky, this lets us disentangle style and content. (Note: disentanglement is still a dark art.)
- **Conditional VAE** can be achieve by adding an additional input to the encoder and decoder networks, which is the additional information $\boldsymbol{y}$:

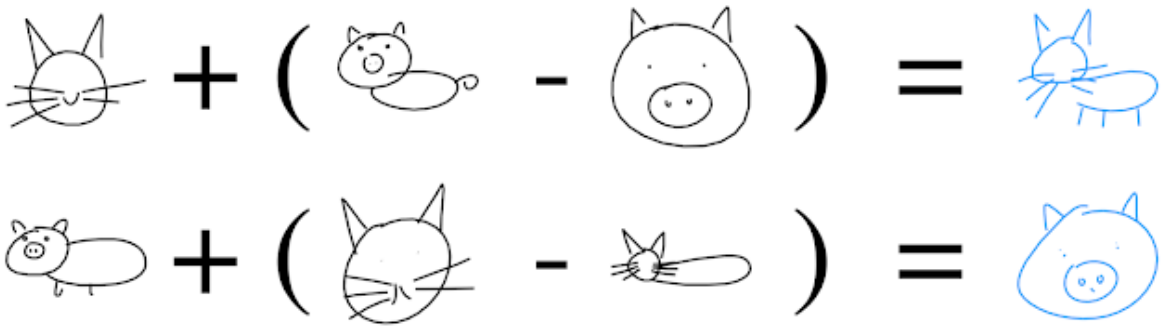

$\tilde{\mathbf{x}}$
$\mathbf{z}$
y
ϵ
μ
$\log \sigma$
$\mathbf{x}$
y

Latent space analysis

You can often get interesting results by interpolating between two vectors in the latent space:

. . .

You can also perform "arithmetic" in the latent space, e.g. by adding or subtracting vectors:

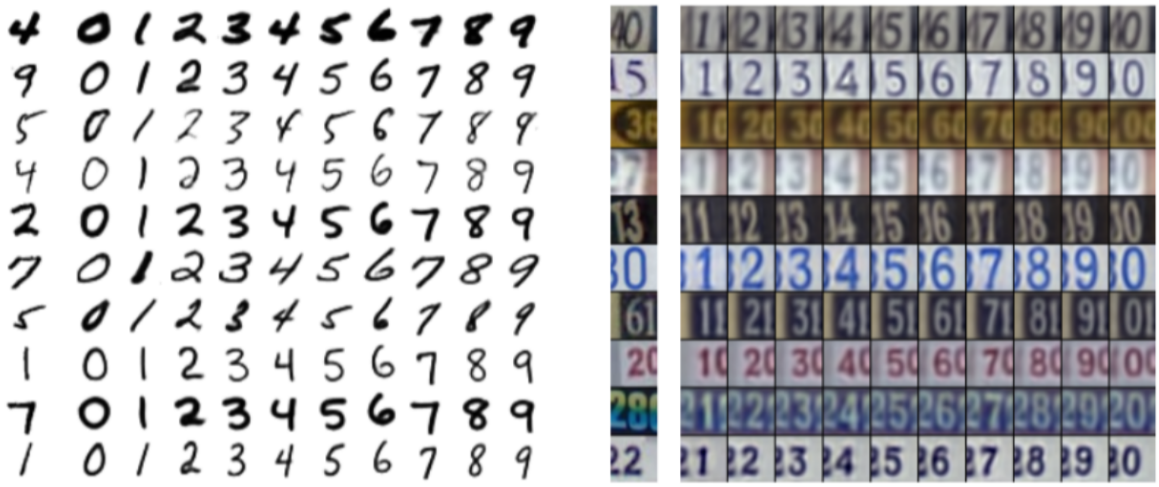


Latent space visualization

By varying two latent dimensions (i.e. dimensions of z) while holding y fixed, we can visualize the latent space.

Latent space interpolation

By varying the label y while holding z fixed, we can solve image analogies.



Problems with VAEs

- The samples may not be very good:
  - VAEs suffer from **mode collapse**, where the amortization network learns to predict the prior. This is due to the strong effect of the KL divergence term in the ELBO.
  - The spherical prior $p(z)$ is too restrictive, it can lead to a "blurry" output distribution (or very noisy samples, if we sample from the likelihood).