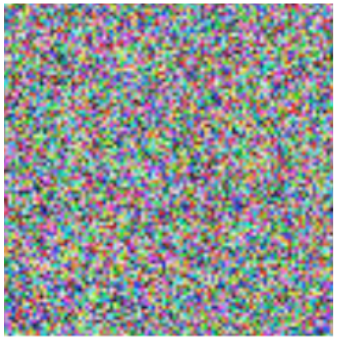
Directed Generative Nets

F. Noé[1]

Deep Learning Classes, FU Berlin 2018

Generative Neural Networks

Noise ~ N(0,1)



Generative Model

Differentiable Generator Nets

- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

$$\mathbf{z} \sim p(\mathbf{z})$$

and perform a linear transformation to a desired distribution:

$$\mathbf{x} = G(\mathbf{z}, \theta) \sim p(\mathbf{x}).$$

- Architecture of G defines family of possible $p(\mathbf{x})$ distributions. Parameters θ select distribution from that family
- **Simple Example**: standard procedure for drawing samples from a normal distribution with mean μ and covariance $\mathbf{\Sigma}$:
  1. Sample: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  2. Transform: $\mathbf{x} = G(\mathbf{z}) = \mu + \mathbf{L}\mathbf{z}$, where $\mathbf{\Sigma} = \mathbf{L}^\top \mathbf{L}$ (Cholesky decomposition).
- **Simple Example**: sampling a univariate distribution $p(x)$ with cumulative distribution function $q(x) = \int_{-\infty}^{x} p(v)\mathrm{d}v$:
  1. Sample $z \sim \mathrm{Uniform}(0, 1)$
  2. Transform $x = G(z) = q^{-1}(z)$.

Generative Neural Networks

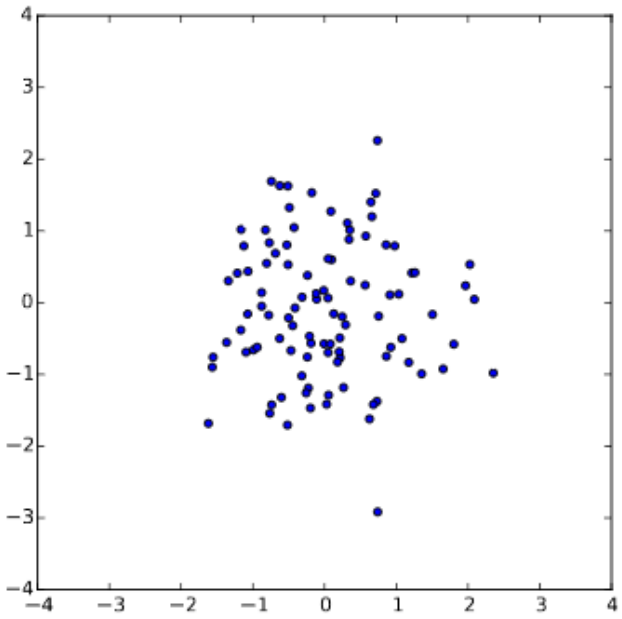
- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

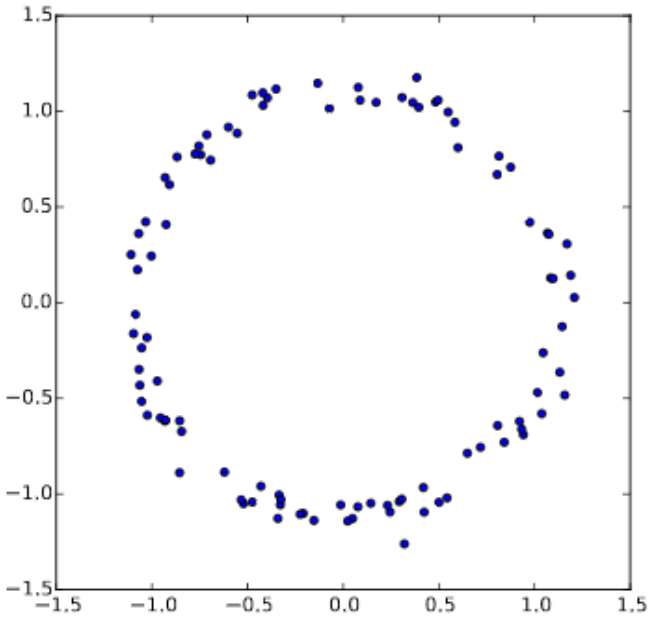
$$\mathbf{z} \sim p(\mathbf{z})$$

and perform a linear transformation to a desired distribution:

$$\mathbf{x} = G(\mathbf{z}, \theta) \sim p(\mathbf{x}).$$

- **Example**:
  - **Left**: Samples from normal distribution, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
  - **Right**: Samples mapped through $G(\mathbf{z}) = \frac{\mathbf{z}}{10} + \frac{\mathbf{z}}{\|\mathbf{z}\|}$ to form a ring.





Differentiable Generator Nets

- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

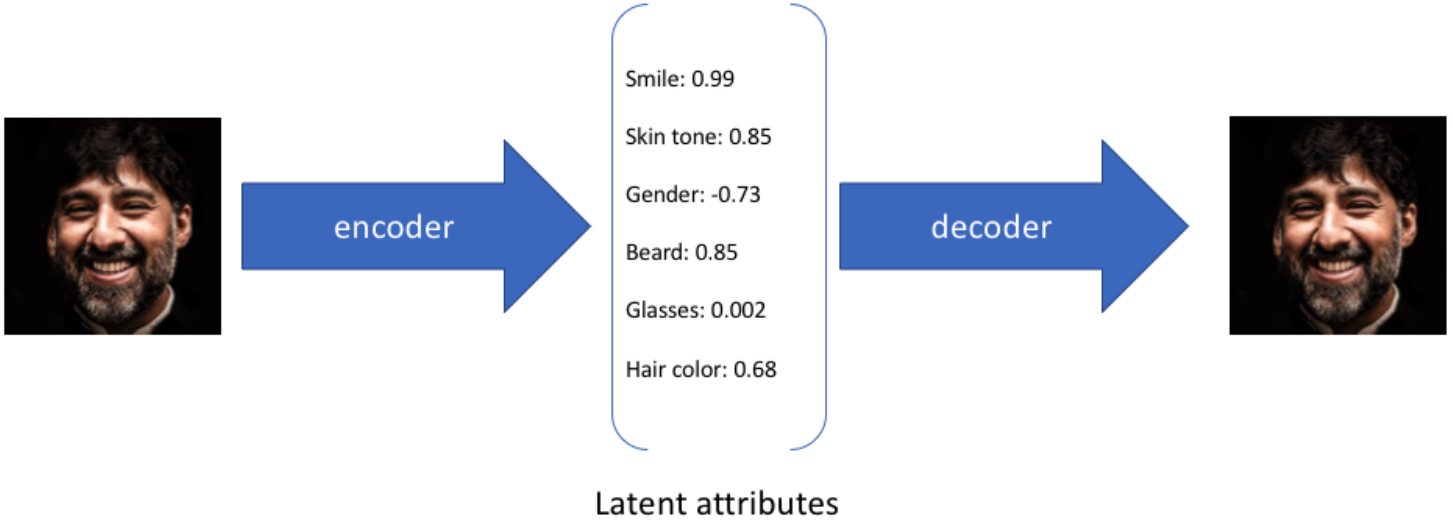
$$\mathbf{z} \sim p(\mathbf{z})$$

and perform a linear transformation to a desired distribution:

$$\mathbf{x} = G(\mathbf{z}, \theta) \sim p(\mathbf{x}).$$

- **Complex Distribution**:
  - G feedforward neural network
  - *train* parameters θ to sample from correct distribution.
- Well-known neural network architectures:
  - **Variational Autoencoders** (inference net + generator net)
  - **Generative Adversarial Networks** (generator network + discriminator network)

Reminder: Autoencoders

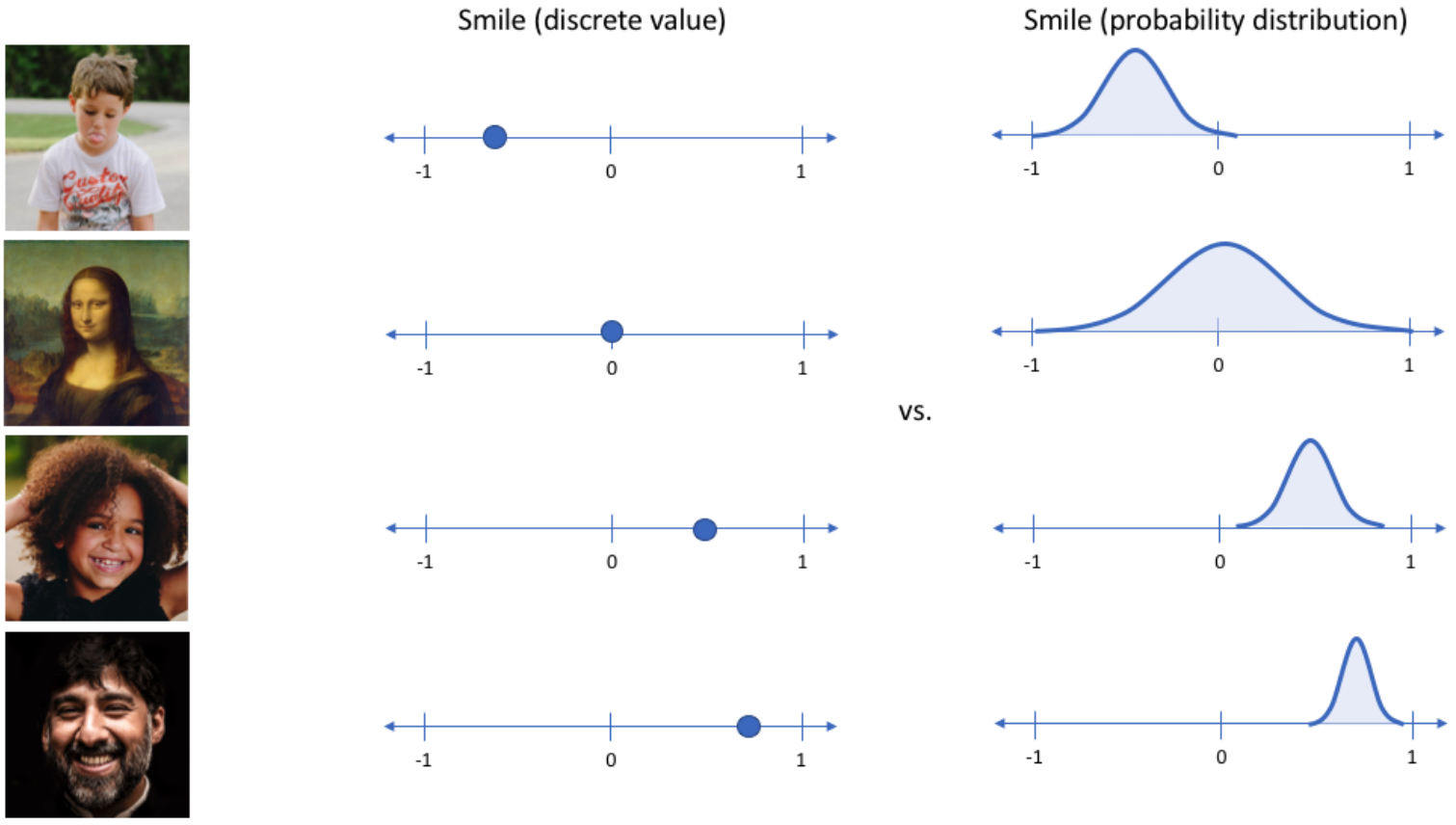


Latent variables encode "essential" information about data points $\mathbf{x}$.

Variational Autoencoders (VAEs)

Variational Autoencoder (VAE) – Kingma, 2013; Rezende et al., 2014

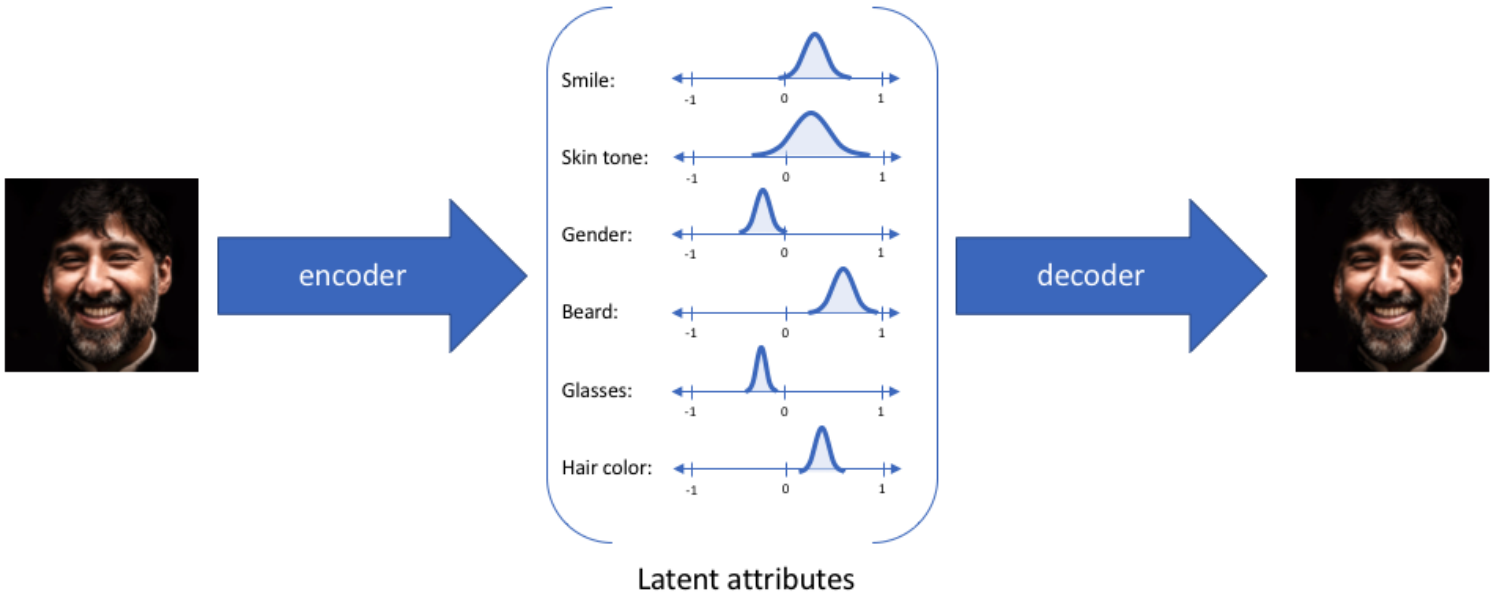
- Instead of a single value for each attribute, represent each latent attribute as a range of possible values.
- E.g., what single value would you assign for the smile attribute if you feed in a photo of the Mona Lisa?
- VAE: describe latent attributes in probabilistic terms.



Variational Autoencoders (VAEs)

Variational Autoencoder (VAE) – Kingma, 2013; Rezende et al., 2014

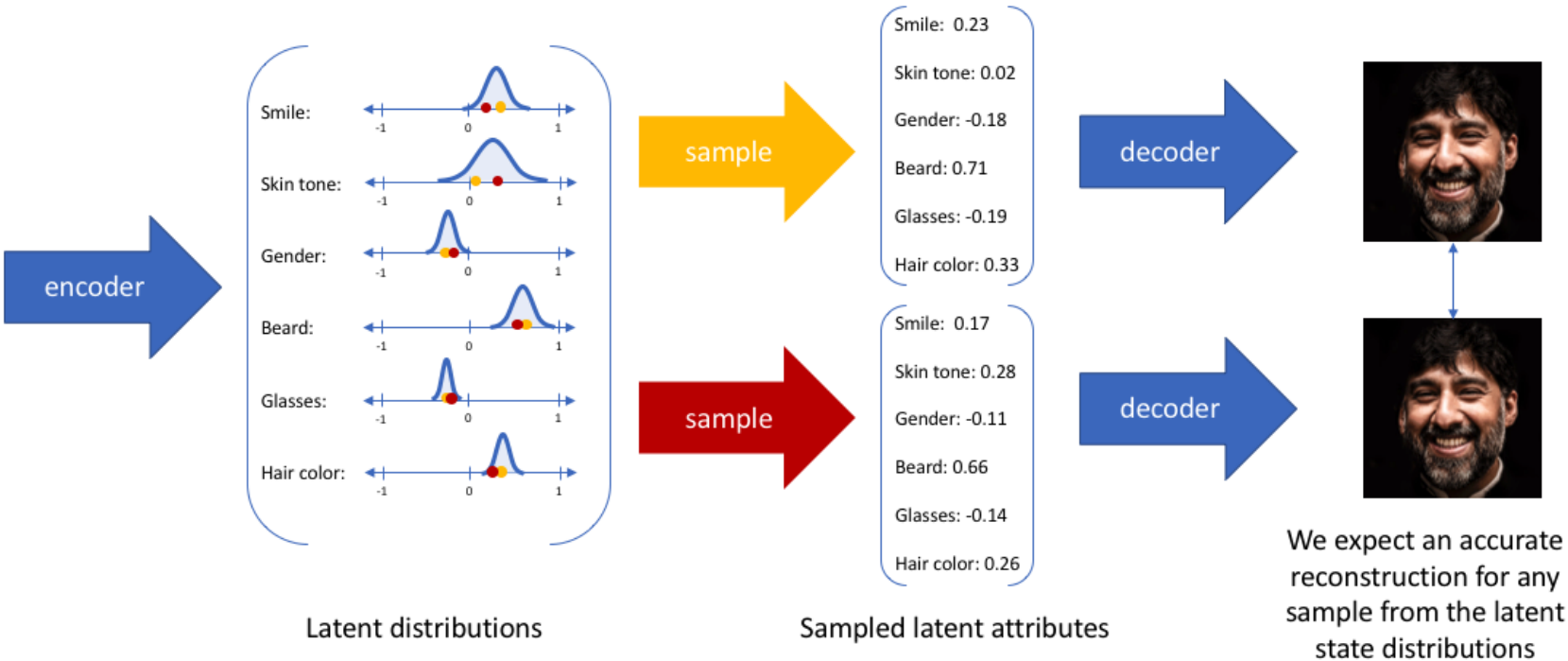
- Directed model that uses learned approximate inference and can be trained purely with gradient-based methods.
- Represent each latent attribute for a given input as a probability distribution.
- When decoding from the latent state, randomly sample from latent state distribution to generate a vector as input for decoder model.



Variational Autoencoders (VAEs)

Variational Autoencoder (VAE) – Kingma, 2013; Rezende et al., 2014

- **Encoder**: define probability distribution of latent variables
- **Sample**: latent variables **z** given the encoding of input **x**
- **Decode**: **z** so as to reconstruct corresponding input **x**.
- Enforces a continuous, smooth latent space representation.
- Values which are nearby to one another in latent space should correspond to similar reconstructions.



Variational Inference

- Variables $\mathbf{x}$ are visible but $\mathbf{z}$ are hidden $\rightarrow$ we need to infer the characteristics of $\mathbf{z}$ from $\mathbf{x}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- But computing $p(\mathbf{x})$ is extremely difficult:

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

- Approaches:
  - Markov-Chain Monte Carlo (no bias, but high variance)
  - Variational inference (bias, no variance)
- **Variational inference idea**: approximate $p(\mathbf{z} \mid \mathbf{x})$ by a tractable distribution $q(\mathbf{z} \mid \mathbf{x}; \theta)$ by optimizing parameters θ and then perform inference with q.

Kullback-Leibler Divergence

- Kullback-Leibler divergence (KL-divergence or relative entropy) between two distributions $p(\mathbf{x})$ and $q(\mathbf{x})$ measures the dissimilarity between the two distributions:

$$D_{\mathrm{KL}}(q \parallel p) = \int_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \mathrm{d}\mathbf{x}$$

- **Interpretation**: Expectation w.r.t. q of the logarithmic difference between the two distributions p and q.
- **Properties**:
  - **Nonnegativity**: $D_{\mathrm{KL}}(q \parallel p) \geq 0$ with equality if and only if $p \equiv q$ (in the sense of probability distributions)
  - $D_{\mathrm{KL}}(q \parallel p) \neq D_{\mathrm{KL}}(p \parallel q)$ – the KL-divergence is not symmetric in its arguments.

- **Goal**: Ensure that tractable distribution $q(\mathbf{z}|\mathbf{x})$ is similar to intractable distribution $p(\mathbf{z}|\mathbf{x})$.
- **Means**: minimize KL divergence

$$D_{\mathrm{KL}}(q \parallel p) = D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z} \mid \mathbf{x})) = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z} \mid \mathbf{x})} \mathrm{d}\mathbf{z}$$

- Direct computation is not possible because:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})}$$

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

Is intractable. $\rightarrow$ We cannot directly compute $D_{\mathrm{KL}}(q \parallel p)$. Can we find another way to minimize $D_{\mathrm{KL}}(q \parallel p)$ without knowing its value?

- Using $p(\mathbf{z} \mid \mathbf{x}) = p(\mathbf{x}, \mathbf{z})/p(\mathbf{x})$:

$$D_{\mathrm{KL}}(q \parallel p) = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z} \mid \mathbf{x})} \mathrm{d}\mathbf{z}$$
$$= \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log \left(\frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} p(\mathbf{x}) \right) \mathrm{d}\mathbf{z}$$

- Using Properties of the log and splitting the integral:

$$D_{\mathrm{KL}}(q \parallel p) = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} + \log p(\mathbf{x}) \right] \mathrm{d}\mathbf{z}$$
$$= \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} \right] \mathrm{d}\mathbf{z} + \log p(\mathbf{x}) \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \mathrm{d}\mathbf{z}$$

- Exploiting that q is a probability distribution $\int_{\mathbf{z}} q(\mathbf{z}) = 1$:

$$D_{\mathrm{KL}}(q \parallel p) = \underbrace{\int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} \right] \mathrm{d}\mathbf{z}}_{L} + \log p(\mathbf{x})$$

- KL divergence

$$D_{\mathrm{KL}}(q \parallel p) = L + \log p(\mathbf{x})$$

- **Variational free energy** (upper bound to $-\log p(\mathbf{x})$):

$$L = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} \right] \mathrm{d}\mathbf{z}$$

- $-L$: Variational **Evidence Lower BOund** (ELBO):

$$-L = \log p(\mathbf{x}) - D_{\mathrm{KL}}(q \parallel p) \leq \log p(\mathbf{x})$$

- Since $p(\mathbf{x})$ is a constant in $\mathbf{z}$ we can minimize $D_{\mathrm{KL}}(q \parallel p)$ by minimizing the variational free energy or maximizing the ELBO:

$$\arg\min D_{\mathrm{KL}}(q \parallel p) = \arg\min L = \arg\max -L$$

- Using $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})$:

$$\begin{aligned} L &= \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x}, \mathbf{z})} \right] \mathrm{d}\mathbf{z} \\ &= \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \left[\log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})} \right] \mathrm{d}\mathbf{z} \end{aligned}$$

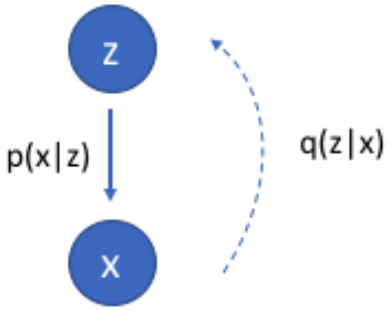
- Using properties of the log:

$$L = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} \mathrm{d}\mathbf{z} - \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log p(\mathbf{x} \mid \mathbf{z}) \mathrm{d}\mathbf{z}$$

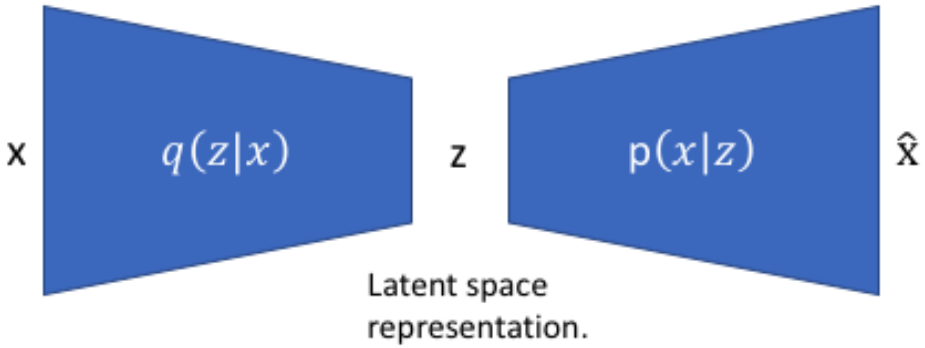
- Interpreting these terms:

$$L = D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) - \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z} \mid \mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z})$$

- We assume that q has a tractable form (e.g. factorizes)



We'd like to use our observations to understand the hidden variable.



Latent space representation.

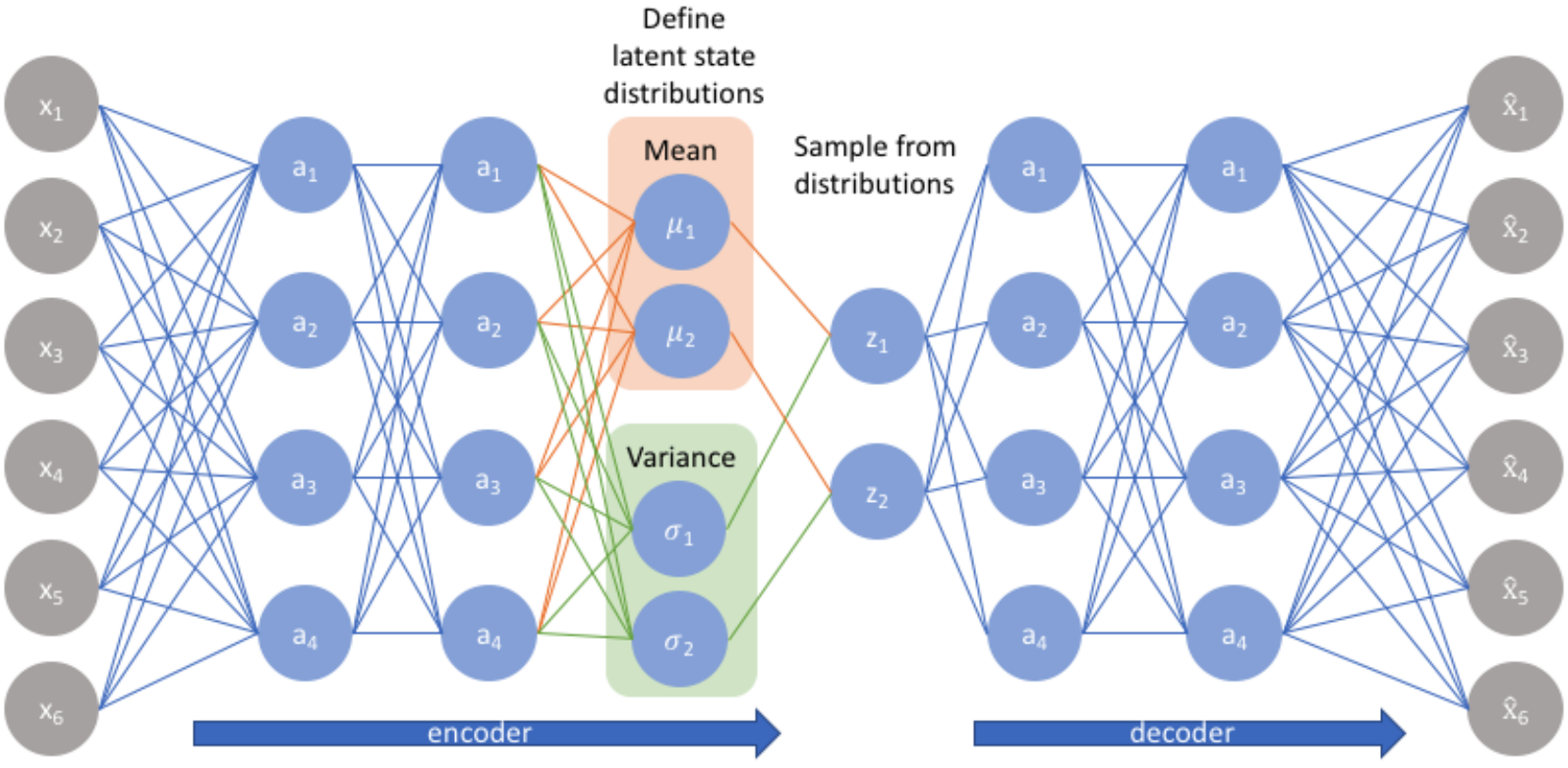
Neural network mapping x to z.

Neural network mapping z to x.

$$\min \left\{ -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}) + D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) \right\}$$

- **Encoder** $q(\mathbf{z} \mid \mathbf{x})$ (inference network, recognition model):
  - Maps to latent space
  - Models approximate posterior distribution q.
  - $\mathscr{D}_{\mathrm{KL}}\left[q(\mathbf{z} \mid \mathbf{x}) \parallel p_{\mathrm{model}}(\mathbf{z})\right]$ tries to make $q(\mathbf{z} \mid \mathbf{x})$ and $p_{\mathrm{model}}(\mathbf{z})$ similar.
- **Decoder** $p(\mathbf{x} \mid \mathbf{z})$.
  - Decodes $\mathbf{z} \rightarrow \hat{\mathbf{x}}$ with the aim to reconstruct the input $\mathbf{x}$.
  - $\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|x)} \log p_{\mathrm{model}}(\mathbf{x} \mid \mathbf{z})$ reconstruction log-likelihood

Gaussian VAE
Structure



- **Ansatz**: isotropic Gaussian generative model:

$$q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu(\mathbf{x}), \mathrm{diag}(\sigma^2(\mathbf{x}))) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{d}\sigma_i} \exp\left[-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i - \mu_i}{\sigma_i}\right)^2\right]$$

  and standard normal latent variables:

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

- We can compute $D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z}))$ explicitly:

$$D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) = \int_{\mathbf{z}} q(\mathbf{z} \mid \mathbf{x}) \log \frac{q(\mathbf{z} \mid \mathbf{x})}{p(\mathbf{z})} \mathrm{d}\mathbf{z} = \frac{1}{2}\sum_{i=1}^{d}\left(1 + \log \sigma_i^2(\mathbf{x}) - \mu_i^2(\mathbf{x}) - \sigma_i^2(\mathbf{x})\right)$$

- This loss can be easily computed.

Computing the Reconstruction Loss
Reparametrization Trick

- Let us write the loss explicitly with parameters:

$$L = -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}; \theta_{\text{dec}}) + D_{\text{KL}}(q(\mathbf{z} \mid \mathbf{x}, \theta_{\text{enc}}) \parallel p(\mathbf{z}))$$

- Computing reconstruction loss involves a sampling of hidden variables $\mathbf{z}$.
- In stochastic gradient descent, it is natural to replace expectation by a single sample for each $\mathbf{x}$:

$$\log p(\mathbf{x} \mid \mathbf{z}; \theta_{\text{dec}}) \sim \mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}; \theta_{\text{dec}})$$

- However, the process of sampling a pdf itself is not differentiable. In order to compute derivatives, we use the reparametrization trick:
  1. Randomly sample ε from a unit Gaussian

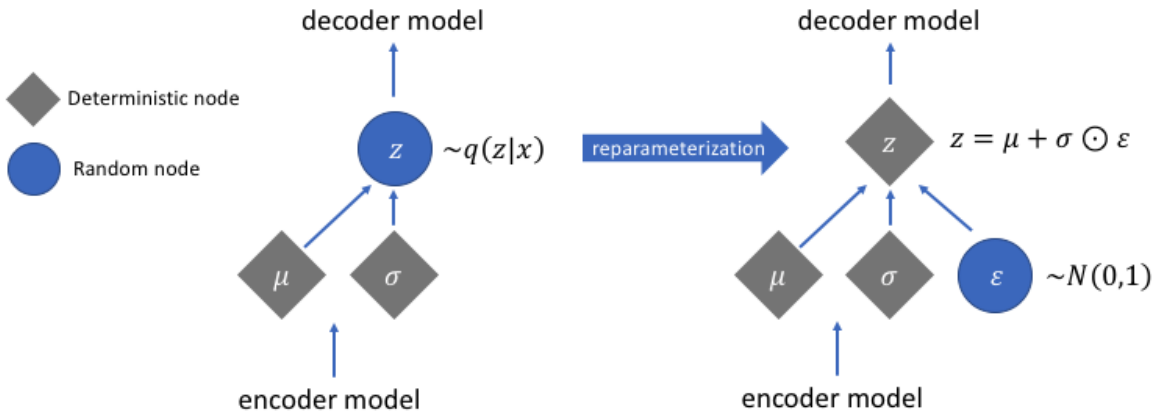
$$\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

  2. Shift ε by mean and scale it by variance of the latent distribution:

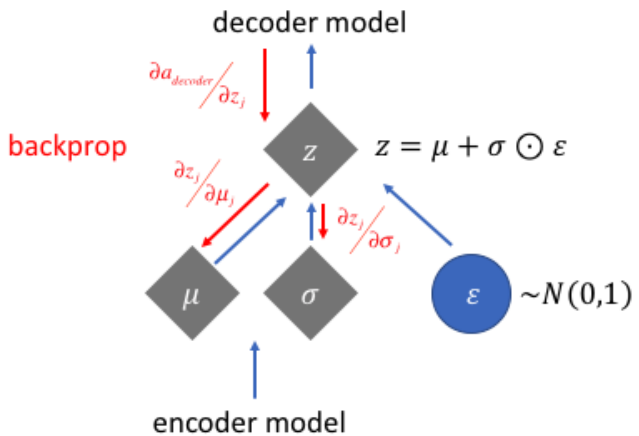
$$\mathbf{z} = \mu(\mathbf{x}) + \sigma(\mathbf{x}) \odot \varepsilon$$

Computing the Reconstruction Loss

Reparametrization Trick



Now we can optimize the parameters of the distribution while still maintaining the ability to randomly sample from that distribution.



Note: To avoid negative values for σ, we can learn $\log \sigma$ and take the

Computing the Reconstruction Loss

Evaluations Reconstruction Loss by Sample

- For each sample pair $\mathbf{x}$, $\mathbf{z}$, evaluate:

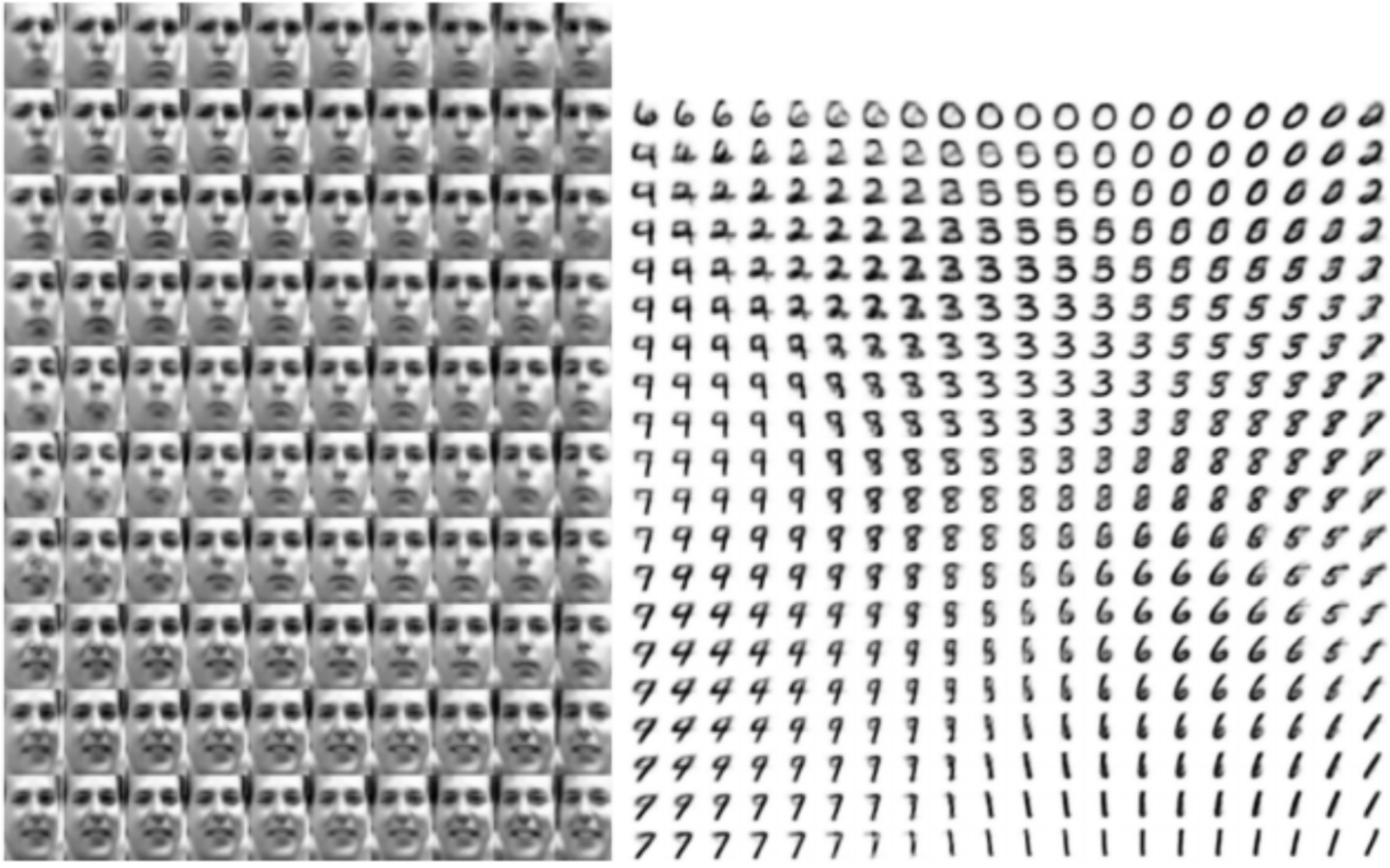
$$\log p(\mathbf{x} \mid \mathbf{z}; \theta)$$

- **Example: Binary MNIST**
  - Use binary images $x_i \in \{0, 1\}$,
  - Use logistic (sigmoid) output layer in decoder to model $\hat{x}_i(\mathbf{x}, \theta) = (p(x_i) = 1)$.
  - Compute log-likelihood (see last lecture, logistic regression)

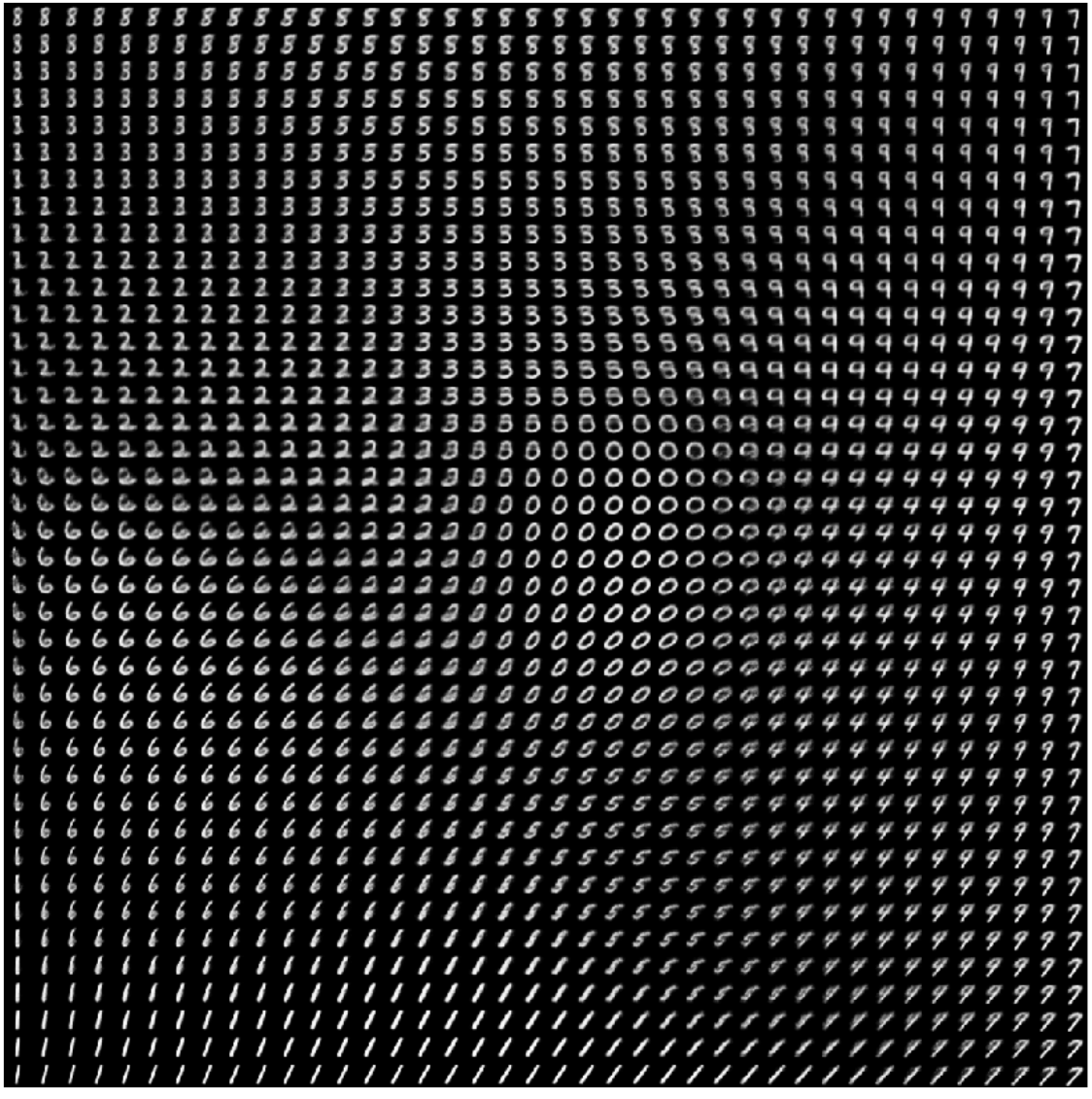
$$L(\theta) = \sum_{i=1}^{N} x_i \log \hat{x}_i(\mathbf{x}, \theta) + (1 - x_i) \log [1 - \hat{x}_i(\mathbf{x}, \theta)]$$

- In practice often other reconstruction losses are used, e.g. $\|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2$.
- There is a disconnect between the mathematical theory and common implementations that are often based on trying to do something similar as suggested by the intuitive interpretation of mathematics!

Variational Autoencoders



Variational Autoencoders

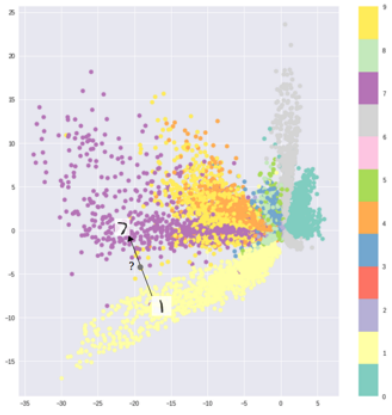


Variational Autoencoders

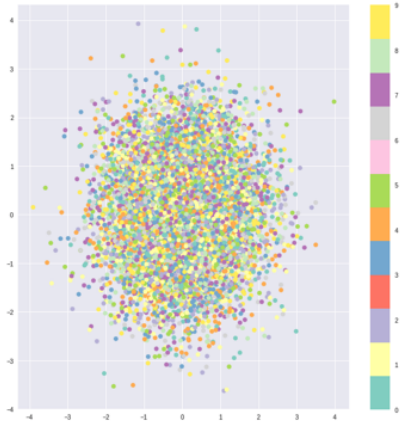
MNIST VAE / Variational Autoencoder

$$\min \left\{ -\mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}) + D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) \right\}$$

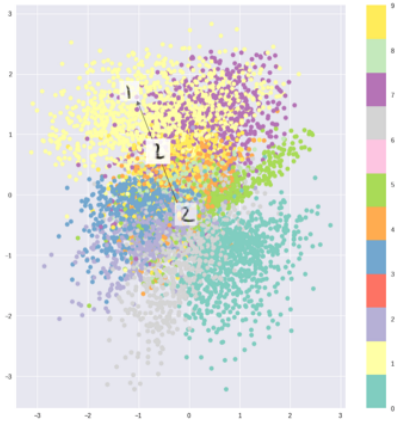
Only reconstruction loss



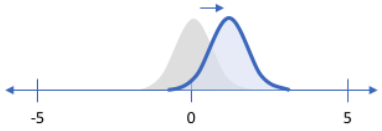
Only KL divergence



Combination



Penalizing reconstruction loss encourages the distribution to describe the input



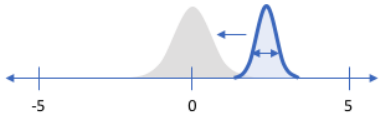
Our distribution deviates from the prior to describe some characteristic of the data

Without regularization, our network can "cheat" by learning narrow distributions

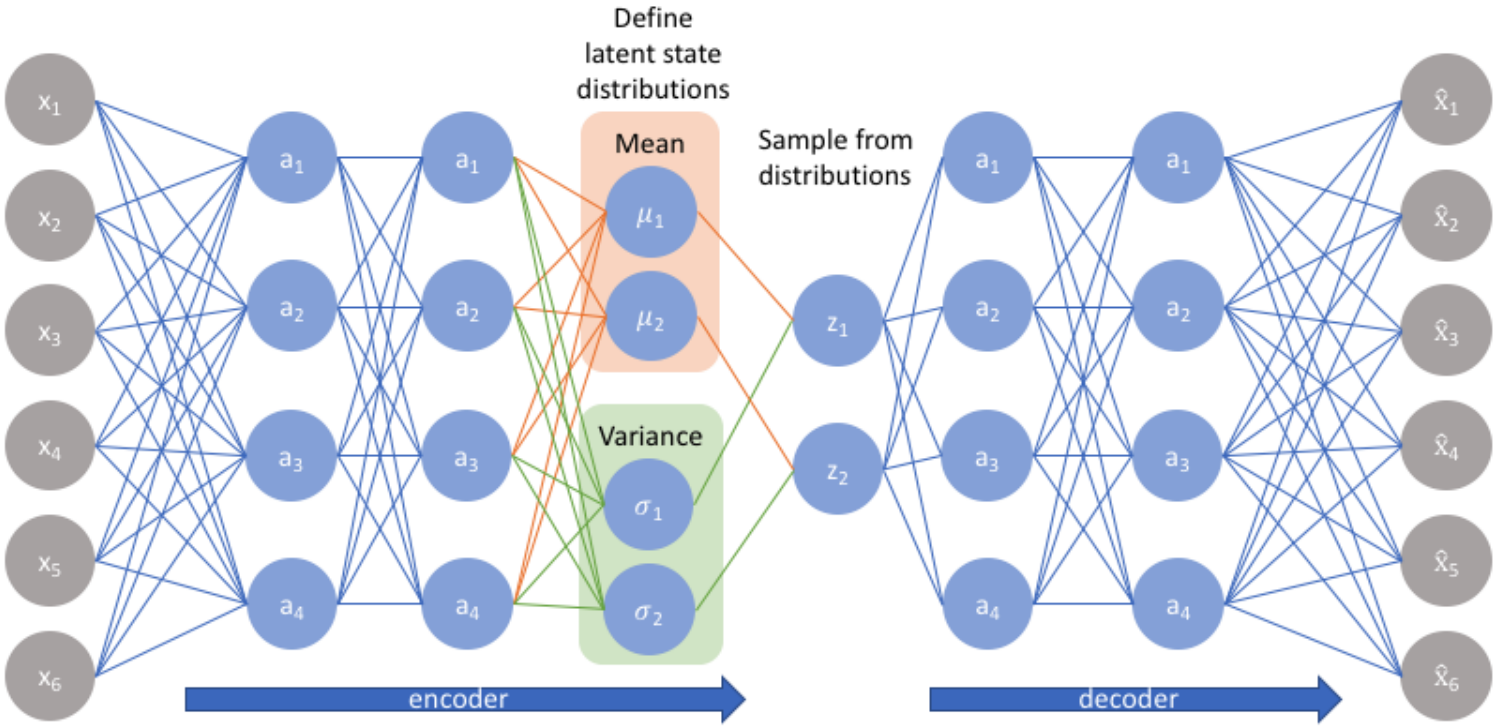
With a small enough variance, this distribution is effectively only representing a single value

Penalizing KL divergence acts as a regularizing force

Attract distribution to have zero mean

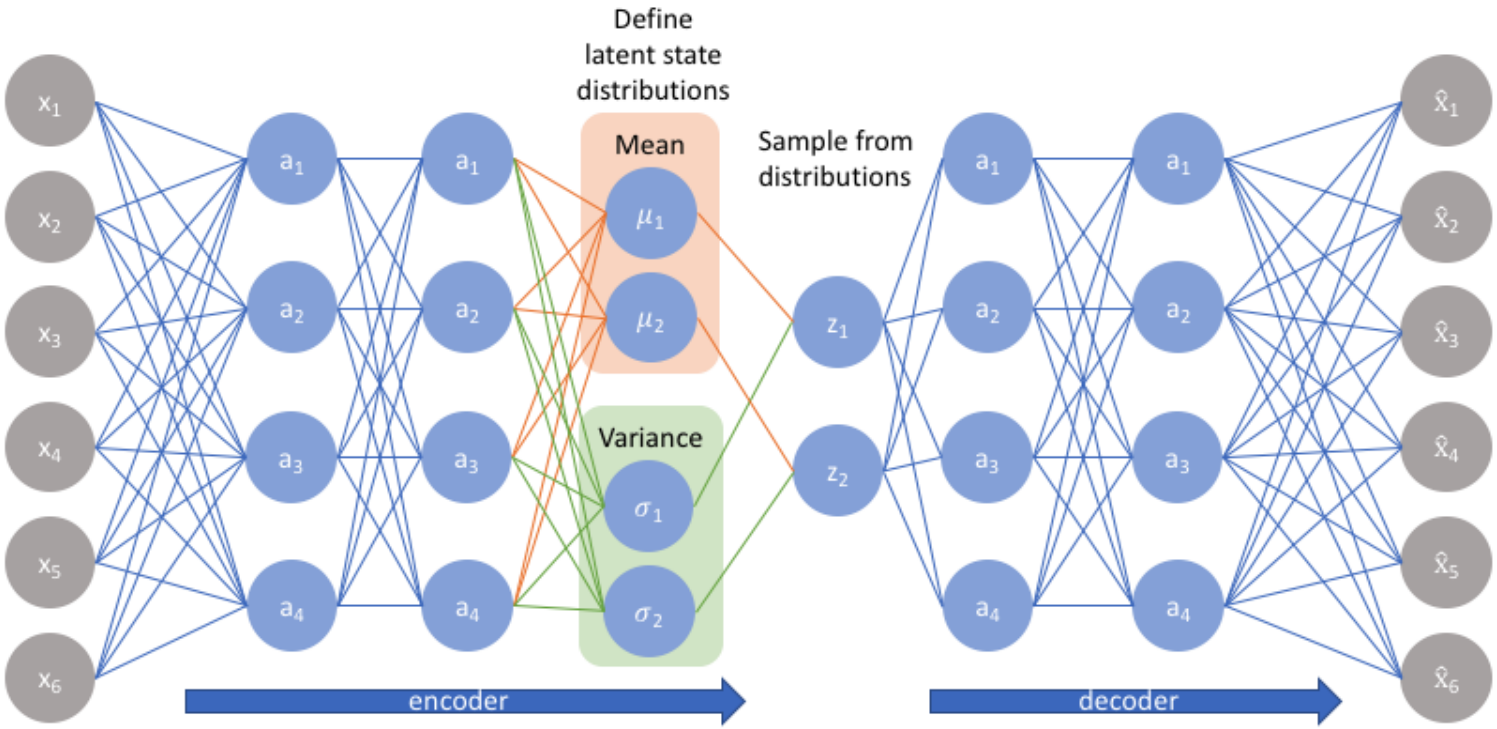


Ensure sufficient variance to yield a smooth latent space



VAE advantages:

- Structure is elegant, theoretically pleasing, and simple to implement.
- Excellent results, among the state of the art approaches to generative modeling.
- Very robust $\rightarrow$ key advantage over Boltzmann machines, which require extremely careful model design to maintain tractability.
- Work very well with a diverse family of differentiable operators.

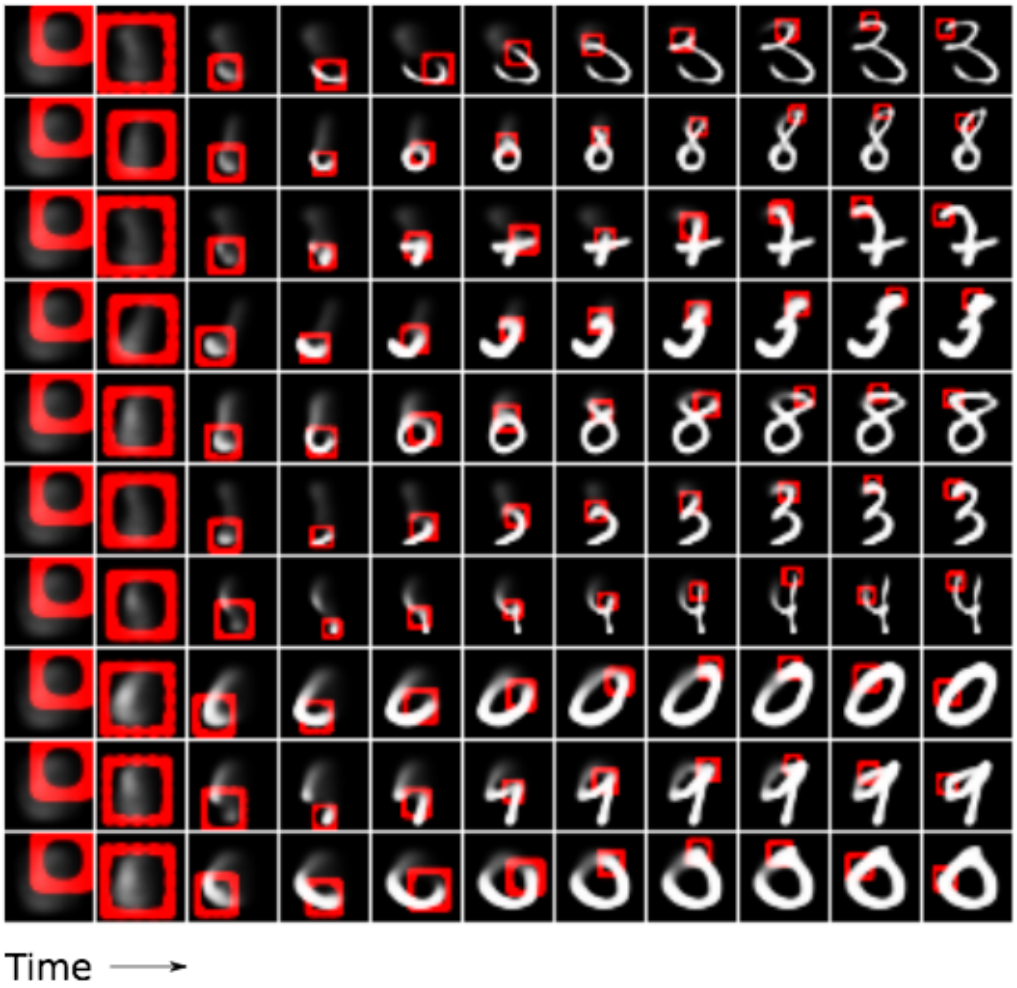


VAE disadvantages:

- Image VAE samples tend to be blurry.
- Maximizing a lower bound on the likelihood of such a distribution is similar to training a traditional autoencoder with mean squared error
- Tends to ignore small/local features of the input.
- Current VAEs tend to use only a small subset of the dimensions of $\mathbf{z}$.

Examples

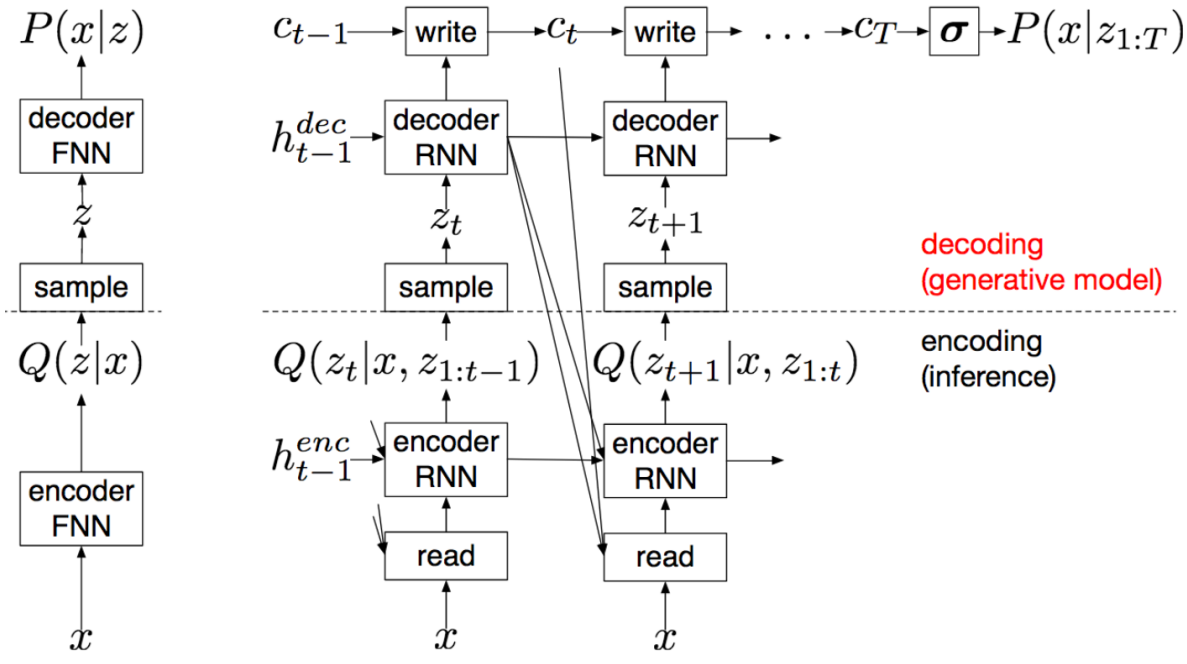
DRAW: A Recurrent Neural Network For Image Generation (Gregor et al, 2015)



Trained DRAW network generating MNIST digits. Each row shows successive stages in the generation of a single digit. Note how the lines composing the digits appear to be "drawn" by the network. The red rectangle delimits the area attended to by the network at each time-step, with the focal precision indicated by the width of the rectangle border.

Examples

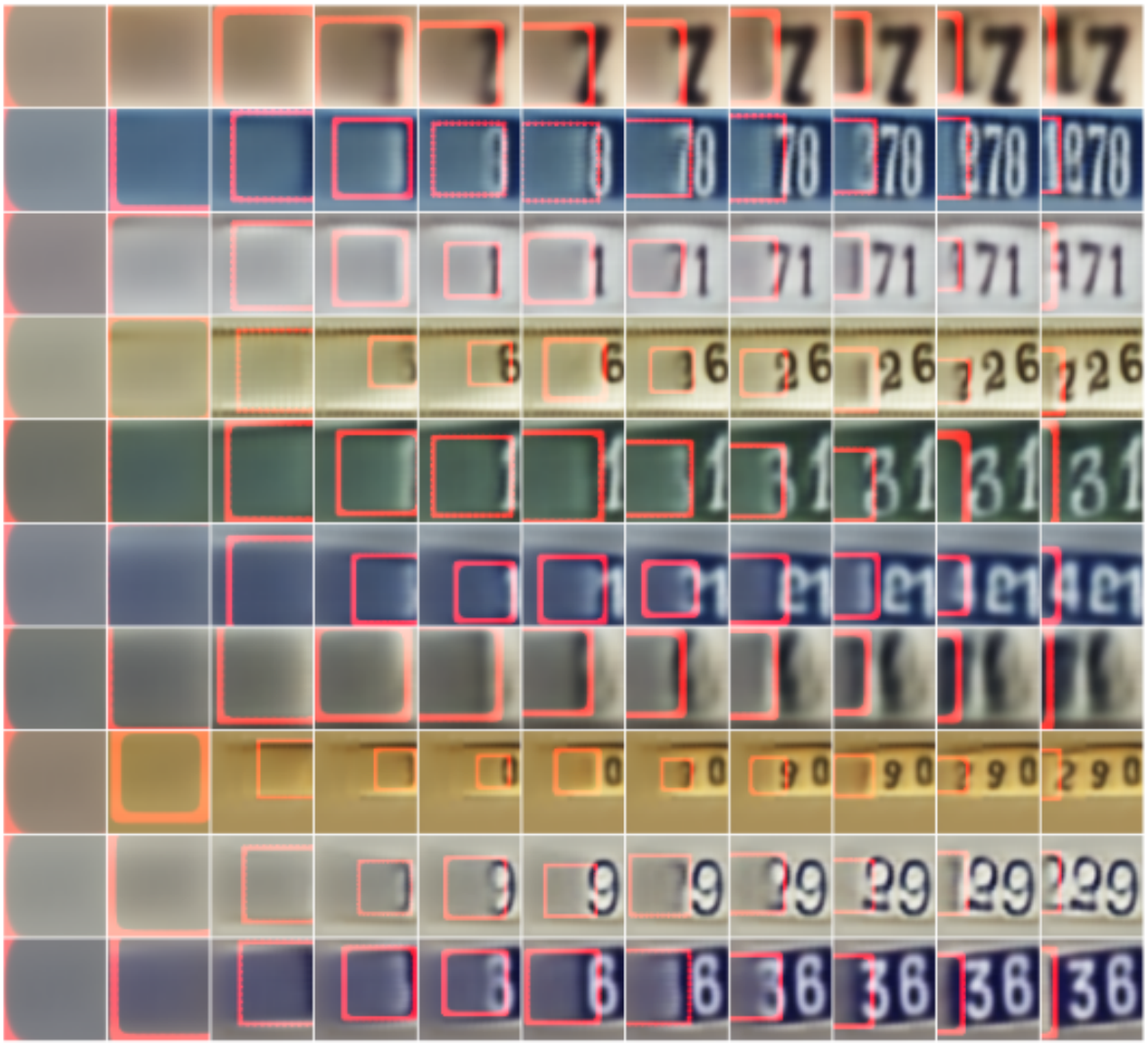
DRAW: A Recurrent Neural Network For Image Generation (Gregor et al, 2015)



Left: Conventional Variational Auto-Encoder. **Right: DRAW Network**. At each time-step a sample $\mathbf{z}_t$ from the prior $P(\mathbf{z}_{1:T})$ is passed to the recurrent decoder network, which then modifies part of the canvas matrix. The final canvas matrix c_T is used to compute $P(\mathbf{x}|\mathbf{z}_{1:T})$. During inference the input is read at every time- step and the result is passed to the encoder RNN. The RNNs at the previous time-step specify where to read. The output of the encoder RNN is used to compute the approximate posterior over the latent variables at that time-step.

Examples

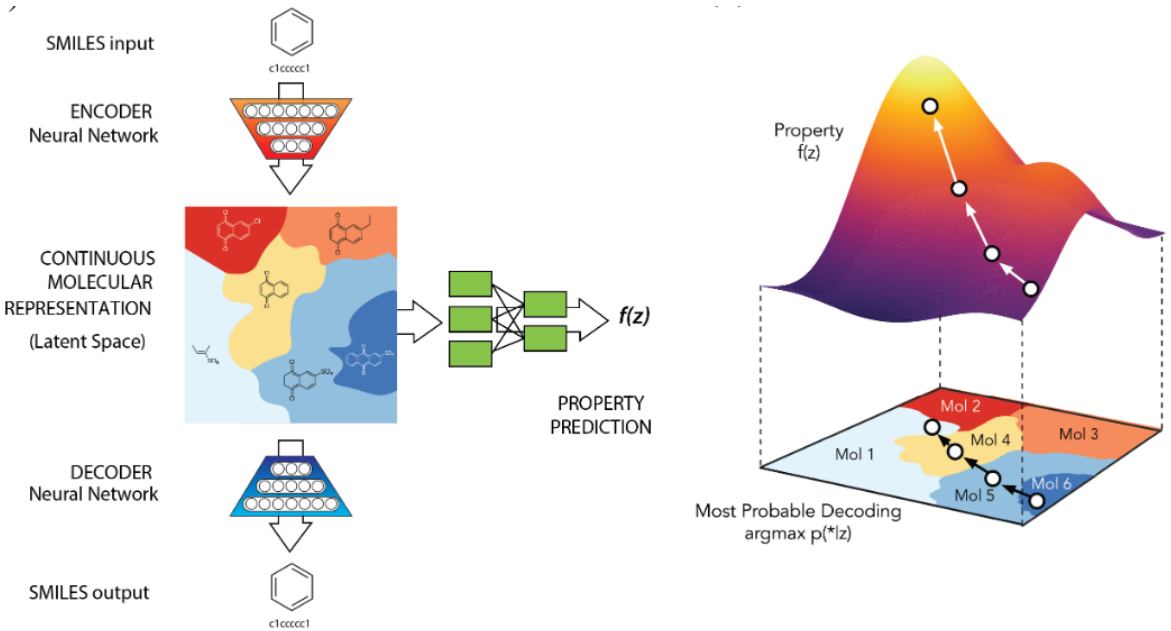
DRAW: A Recurrent Neural Network For Image Generation (Gregor et al, 2015)



SVHN Generation Sequences. The red rectangle indicates the attention patch. Notice how the network draws the digits one at a time, and how it moves and scales the writing patch to produce numbers with different slopes and sizes.

Examples

Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)

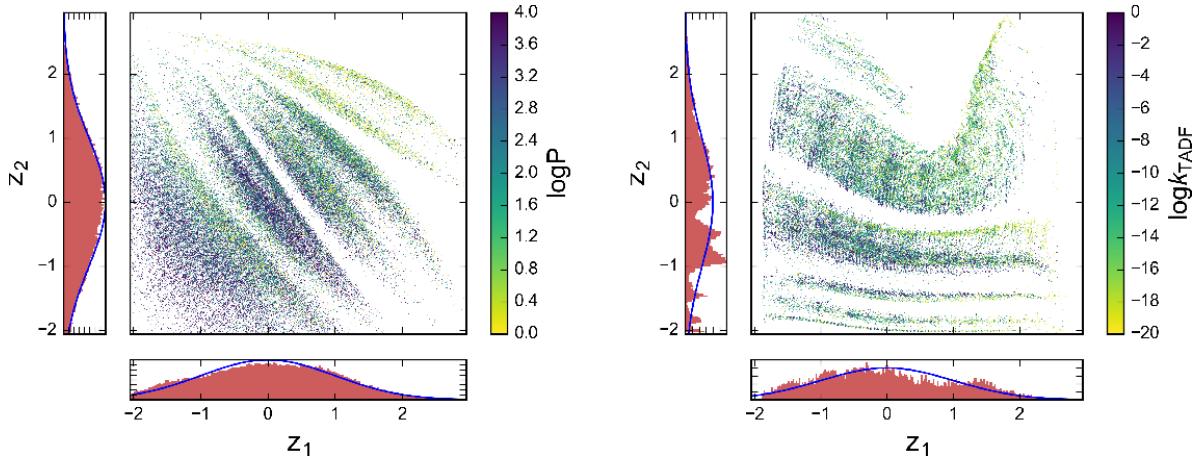


Interpolation. Starting from a discrete molecular representation, such as a SMILES string, the encoder network converts each molecule into a vector in the latent space, which is effectively a continuous molecular representation. Given a point in the latent space, the decoder network produces a corresponding SMILES string.
Architecture:

- Encode characters strings into vectors using recurrent neural networks (RNNs).
- Encoder: 1D convolutional layers, fully-connected layer
- Decoder: Three layers of gated recurrent unit (GRU) networks.
- The last layer of the RNN decoder defines a probability distribution over all possible characters at each position in the SMILES string (stochastic writeout)

Examples

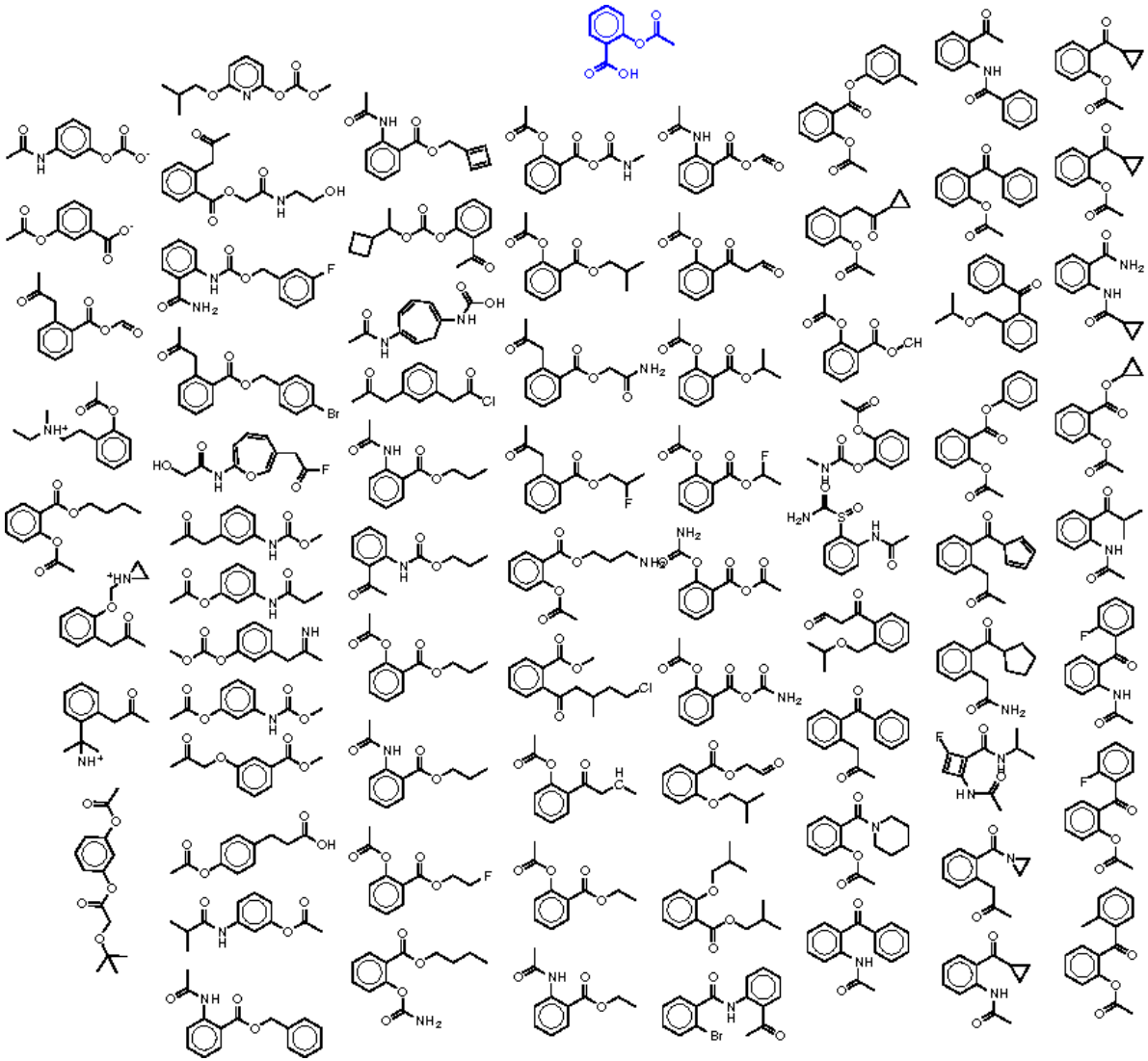
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



Interpolation. Projection of the molecular training sets onto learned two-dimensional latent spaces. The one-dimensional histograms show the distribution of the training data along each dimension, overlaid with the Gaussian prior imposed in the variational autoencoder. The points are colored along a chemical property that is relevant to their function, and will be the target of optimization experiments. **Left**: A natural library of drug-like molecules, colored by their predicted water-octanol partition coefficient. **Right**: A combinatorially-generated library of organic LED molecules, colored by their predicted delayed fluorescent emission rate (kTADF in μs^{-1}).

Examples

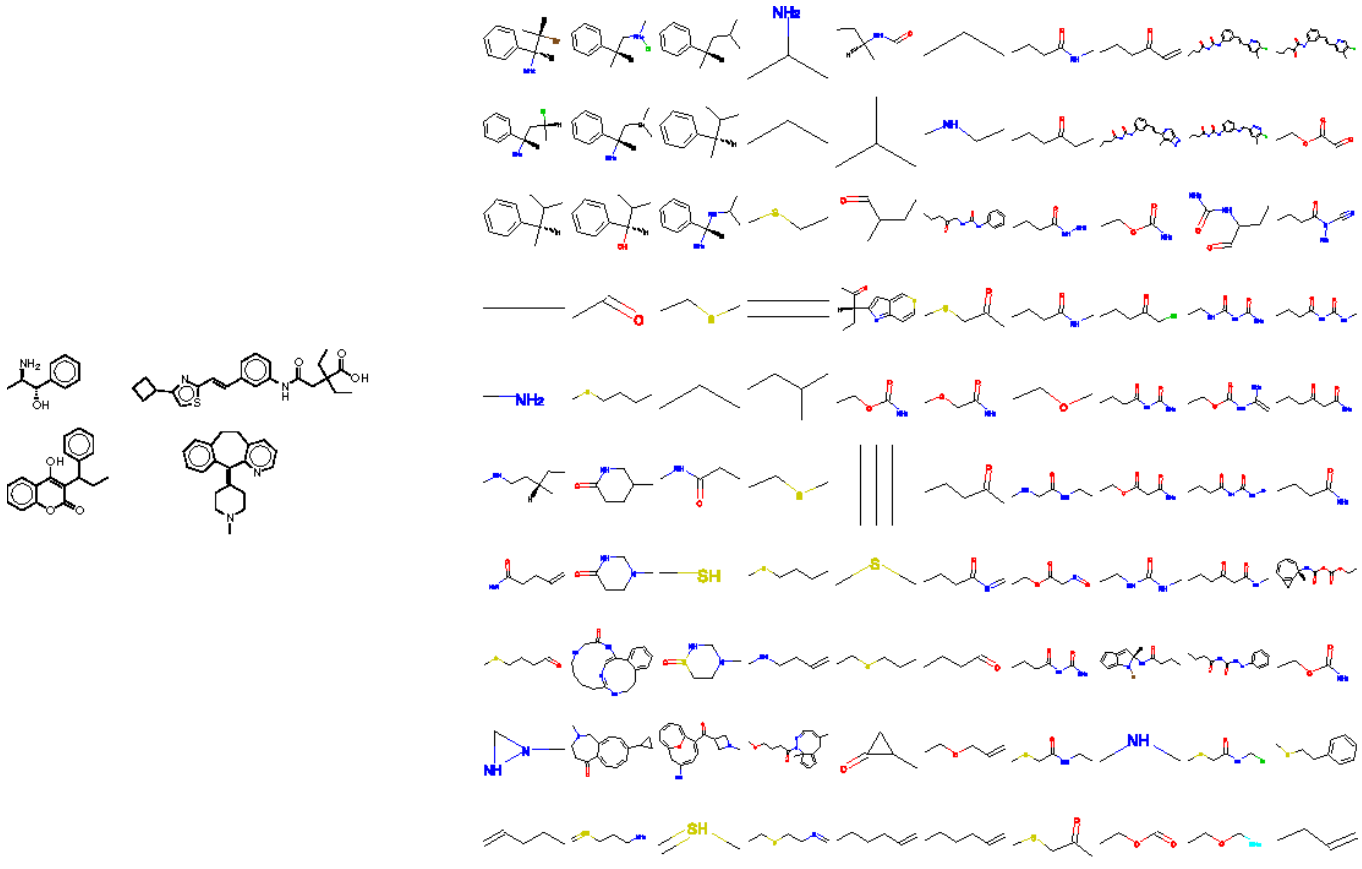
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



Interpolation. Molecules decoded from randomly-sampled points in the latent space of a variational autoencoder, near to a given molecule (aspirin [2-(acetyloxy)benzoic acid], highlighted in blue).

Examples

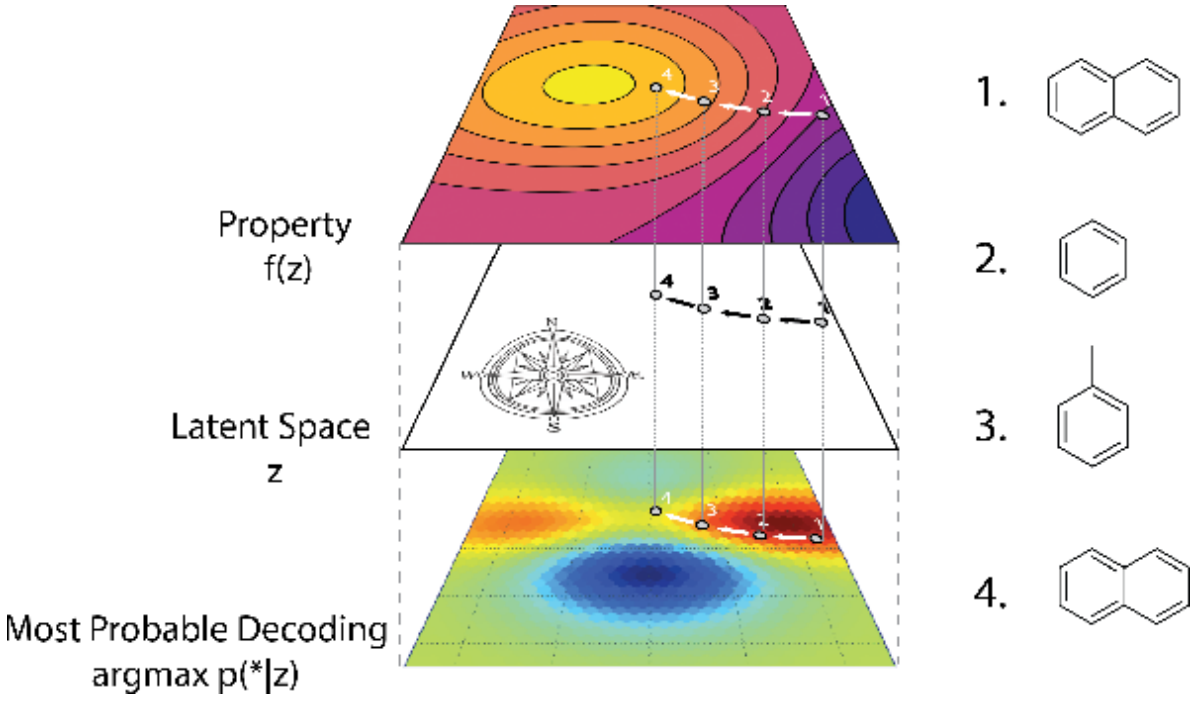
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



Interpolation. Two-dimensional interpolation between four random drugs. Left: Starting molecules, whose encodings defined the four corners of a place in the latent space. Right: Decodings of linearly-interpolated points between the latent representations of the four molecules to the right.

Examples

Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)

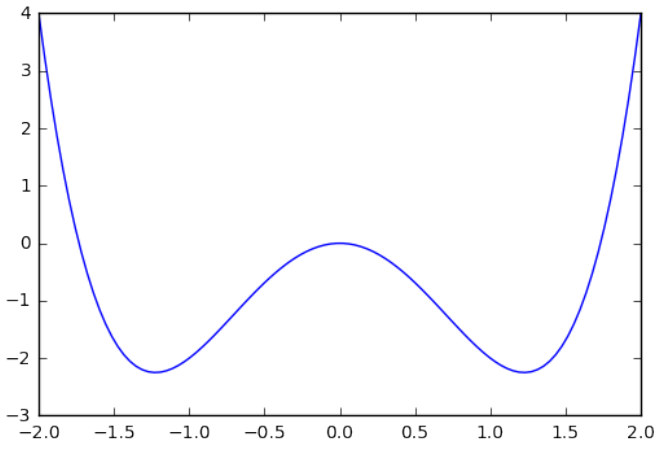


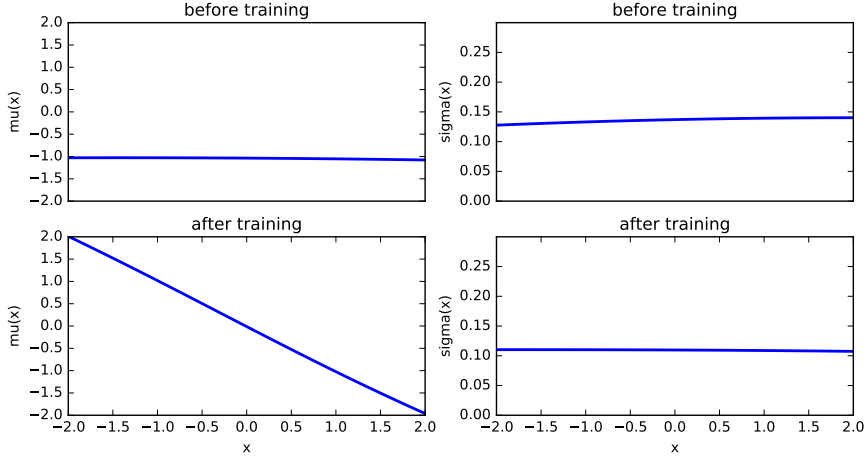
Interpolation. Gradient-based optimization in continuous latent space. After training a surrogate model $f(\mathbf{z})$ to predict the properties of molecules based on their latent representation $\mathbf{z}$, we can optimize $f(\mathbf{z})$ with respect to $\mathbf{z}$ to find new latent representation expected to have high values of desired properties. These new latent representations can then be decoded into SMILES strings, at which point their properties can be tested emprically.

Project (Noé group)
Adaptive Markov Chain Monte Carlo (MCMC)

- **Aim**: Given $\mu(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$ as input, we want to sample $\mathbf{x}$ efficiently using MCMC.
- **Difficulty**: Make efficient steps $\mathbf{x}_1 \to \mathbf{x}_2$ that have a high acceptance probability and cover a large distance $d(\mathbf{x}_1, \mathbf{x}_2)$.
- **Approach**:
  - Use neural network to learn complex proposal step $\mathbf{x}_1 \to \mathbf{x}_2$ in a way that allows us to compute $p_{\mathrm{acc}}(\mathbf{x}_1 \to \mathbf{x}_2)$
  - Maximize efficiency

$$S(\mathbf{x}_1 \to \mathbf{x}_2) = p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)\, d(\mathbf{x}_1, \mathbf{x}_2)$$





Project (Noé group)
Adaptive Markov Chain Monte Carlo (MCMC)

- **Aim**: Given $\mu(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$ as input, we want to sample $\mathbf{x}$ efficiently using MCMC.
- **Difficulty**: Make efficient steps $\mathbf{x}_1 \to \mathbf{x}_2$ that have a high acceptance probability and cover a large distance $d(\mathbf{x}_1, \mathbf{x}_2)$.
- **Approach**:
  - Use neural network to learn complex proposal step $\mathbf{x}_1 \to \mathbf{x}_2$ in a way that allows us to compute $p_{\mathrm{acc}}(\mathbf{x}_1 \to \mathbf{x}_2)$
  - Maximize efficiency

$$S(\mathbf{x}_1 \to \mathbf{x}_2) = p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)\, d(\mathbf{x}_1, \mathbf{x}_2)$$