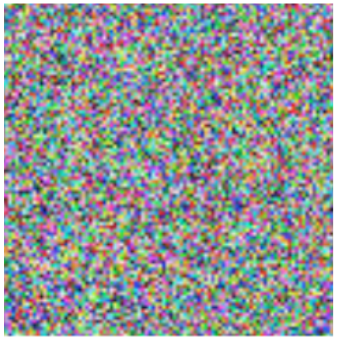
Directed Generative Nets and Statistical Mechanics

F. Noé[1]

IPAM 2019

Generative Neural Networks

Noise ~ N(0,1)



Generative Model

Directed Generative Nets

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

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

and perform a 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)

Directed Generative Nets

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

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

and perform a 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)

Directed Generative Nets

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

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

and perform a 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)

Directed Generative Nets

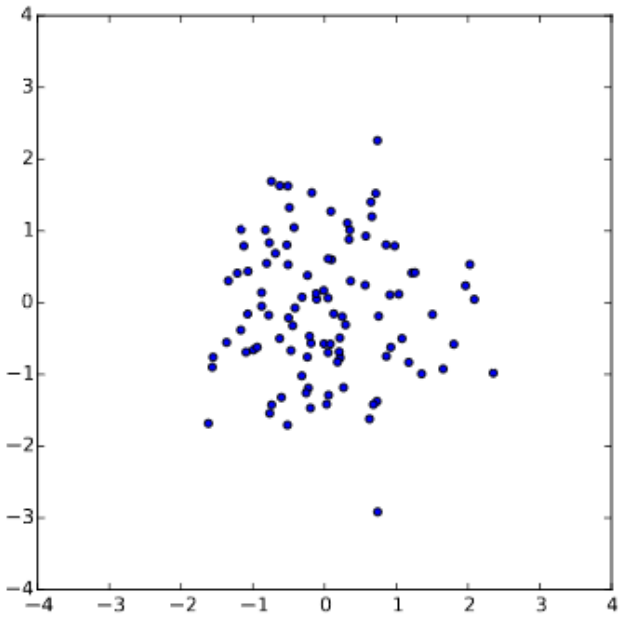
- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

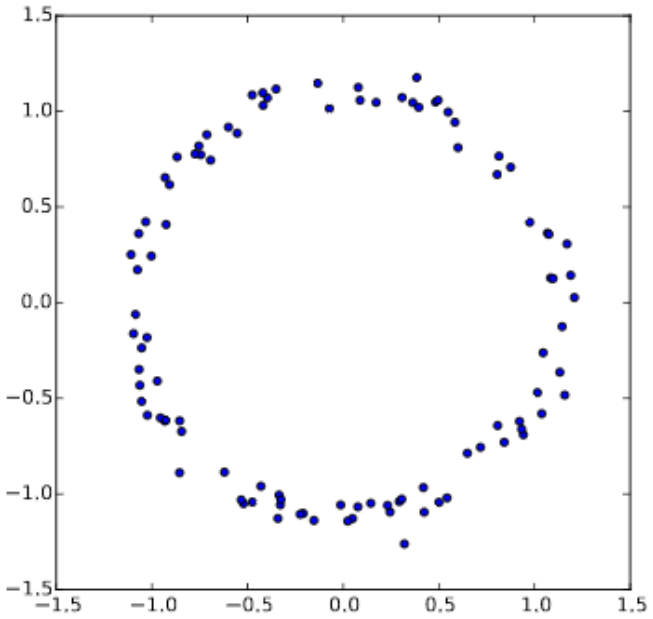
$$\mathbf{z} \sim p(\mathbf{z})$$

and perform a linear transformation to a desired distribution:

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

- **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.





Directed Generative Nets

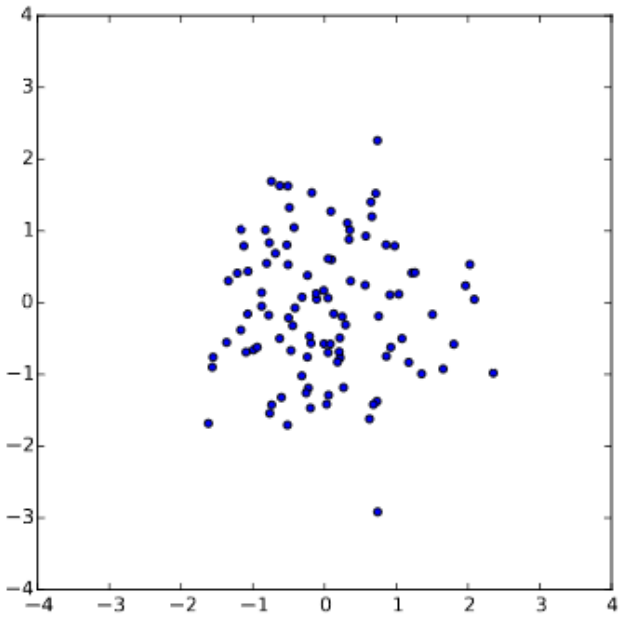
- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

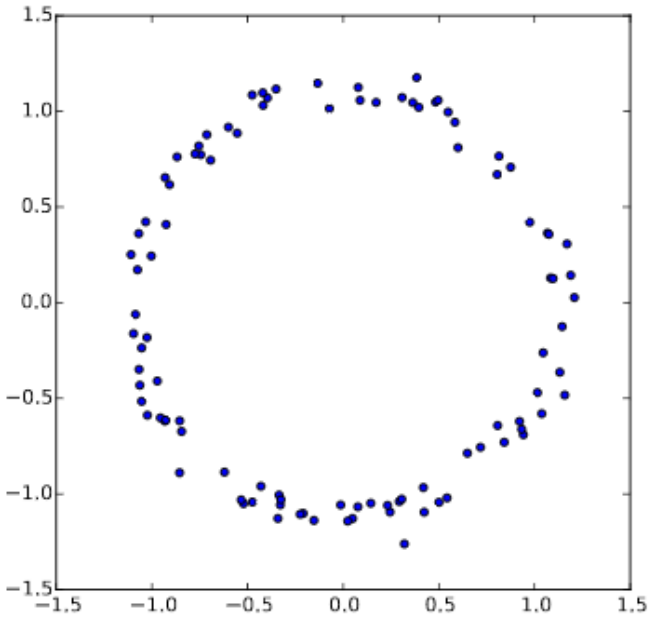
$$\mathbf{z} \sim p(\mathbf{z})$$

and perform a linear transformation to a desired distribution:

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

- **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.





Directed Generative 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:

Likelihood model	Network
None	Generative Adversarial Net (GAN)
Approximate	Variational Autoencoder (VAE)
Exact	Flow-based models Boltzmann Generation

Directed Generative 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:

Likelihood model	Network
None	Generative Adversarial Net (GAN)
Approximate	Variational Autoencoder (VAE)
Exact	Flow-based models Boltzmann Generation

Directed Generative 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:

Likelihood model	Network
None	Generative Adversarial Net (GAN)
Approximate	Variational Autoencoder (VAE)
Exact	Flow-based models Boltzmann Generation

- **Idea**: Game in which the Generator network competes against a Discriminator network, i.e. these two networks are adversaries.
- **Generator network** G: directly produces samples

$$\mathbf{x} = G(\mathbf{z}; \theta_G)$$

- **Discriminator network** D:
  - Attempts to distinguish between samples drawn from the training data and samples drawn from the generator.
  - Emits a probability value that $\mathbf{x}$ is a true sample and not a fake:

$$p_{\text{true}}(\mathbf{x}) = D(\mathbf{x}; \theta_D)$$

- Simplest formulation: **Zero Sum Game**
  - Discriminator receives payoff $v(\theta_G, \theta_D)$
  - Generator receives payoff $-v(\theta_G, \theta_D)$

- **Idea**: Game in which the Generator network competes against a Discriminator network, i.e. these two networks are adversaries.
- **Generator network** G: directly produces samples

$$\mathbf{x} = G(\mathbf{z}; \theta_G)$$

- **Discriminator network** D:
  - Attempts to distinguish between samples drawn from the training data and samples drawn from the generator.
  - Emits a probability value that $\mathbf{x}$ is a true sample and not a fake:

$$p_{\text{true}}(\mathbf{x}) = D(\mathbf{x}; \theta_D)$$

- Simplest formulation: **Zero Sum Game**
  - Discriminator receives payoff $v(\theta_G, \theta_D)$
  - Generator receives payoff $-v(\theta_G, \theta_D)$

- **Idea**: Game in which the Generator network competes against a Discriminator network, i.e. these two networks are adversaries.
- **Generator network** G: directly produces samples

$$\mathbf{x} = G(\mathbf{z}; \theta_G)$$

- **Discriminator network** D:
  - Attempts to distinguish between samples drawn from the training data and samples drawn from the generator.
  - Emits a probability value that $\mathbf{x}$ is a true sample and not a fake:

$$p_{\text{true}}(\mathbf{x}) = D(\mathbf{x}; \theta_D)$$

- Simplest formulation: **Zero Sum Game**
  - Discriminator receives payoff $v(\theta_G, \theta_D)$
  - Generator receives payoff $-v(\theta_G, \theta_D)$

- **Idea**: Game in which the Generator network competes against a Discriminator network, i.e. these two networks are adversaries.
- **Generator network** G: directly produces samples

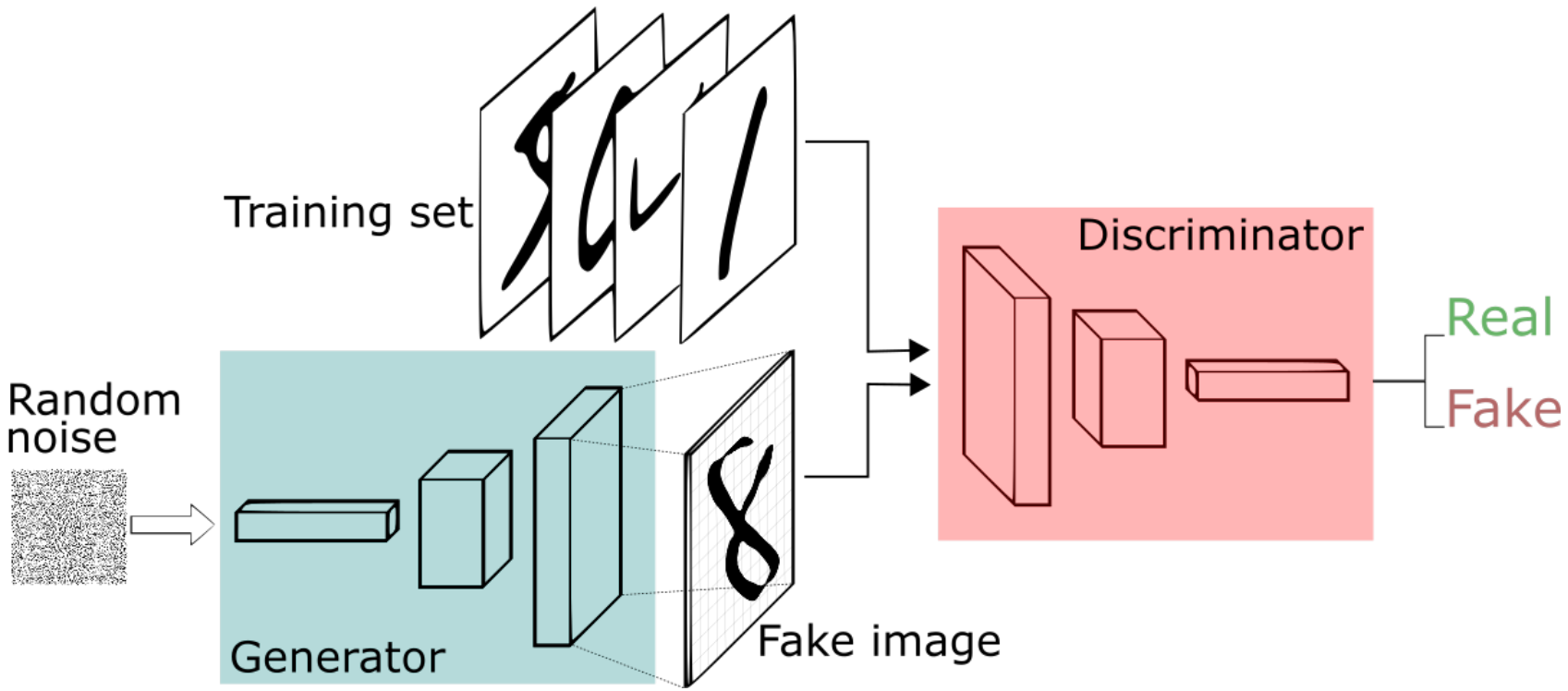
$$\mathbf{x} = G(\mathbf{z}; \theta_G)$$

- **Discriminator network** D:
  - Attempts to distinguish between samples drawn from the training data and samples drawn from the generator.
  - Emits a probability value that $\mathbf{x}$ is a true sample and not a fake:

$$p_{\text{true}}(\mathbf{x}) = D(\mathbf{x}; \theta_D)$$

- Simplest formulation: **Zero Sum Game**
  - Discriminator receives payoff $v(\theta_G, \theta_D)$
  - Generator receives payoff $-v(\theta_G, \theta_D)$

- Discrimiator randomly receives either generated (fake) or training (real) sample as input.
- Generator tries to fake a sample and trick Discriminator into believing it, Discriminator tries to reveal the truth.



- During learning, **each player attempts to maximize its own payoff**, so that at convergence:

$$\hat{\theta}_G = \arg\min_{\theta_G} \max_{\theta_D} v(\theta_G, \theta_D).$$

- **Default choice**

$$v(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}; \theta_D) + \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}(\theta_G)} \log\left(1 - D(\mathbf{x}; \theta_D)\right)$$

  - Discriminator gets reward for correctly classifying samples as real or fake.
  - Generator gets reward when fooling the classifier into believing its samples are real.

- **At convergence**:
  - The Generator's samples are indistinguishable from real data
  - Discriminator outputs 0.5 everywhere. The discriminator may then be discarded.

- During learning, **each player attempts to maximize its own payoff**, so that at convergence:

$$\hat{\theta}_G = \arg\min_{\theta_G} \max_{\theta_D} v(\theta_G, \theta_D).$$

- **Default choice**

$$v(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}; \theta_D) + \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}(\theta_G)} \log \left(1 - D(\mathbf{x}; \theta_D)\right)$$

  - Discriminator gets reward for correctly classifying samples as real or fake.
  - Generator gets reward when fooling the classifier into believing its samples are real.

- **At convergence**:
  - The Generator's samples are indistinguishable from real data
  - Discriminator outputs 0.5 everywhere. The discriminator may then be discarded.

- During learning, **each player attempts to maximize its own payoff**, so that at convergence:

$$\hat{\theta}_G = \arg\min_{\theta_G} \max_{\theta_D} v(\theta_G, \theta_D).$$

- **Default choice**

$$v(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \log D(\mathbf{x}; \theta_D) + \mathbb{E}_{\mathbf{x} \sim p_{\text{model}}(\theta_G)} \log \left(1 - D(\mathbf{x}; \theta_D)\right)$$

  - Discriminator gets reward for correctly classifying samples as real or fake.
  - Generator gets reward when fooling the classifier into believing its samples are real.
- **At convergence**:
  - The Generator's samples are indistinguishable from real data
  - Discriminator outputs 0.5 everywhere. The discriminator may then be discarded.

- **Pros**:
  - Learning process does not require approximations such as variational inference.
  - When $\max_{\theta_D} v(\theta_G, \theta_D)$ is convex in θ_D, the procedure is guaranteed to converge.
- **Cons**:
  - Learning in GANs can be difficult in practice when G and D are represented by neural networks and $\max_{\theta_D} v(\theta_G, \theta_D)$ is not convex.
  - Optimization seeks a saddle point. Simultaneous gradient descent on two players' costs is not guaranteed to reach an equilibrium.

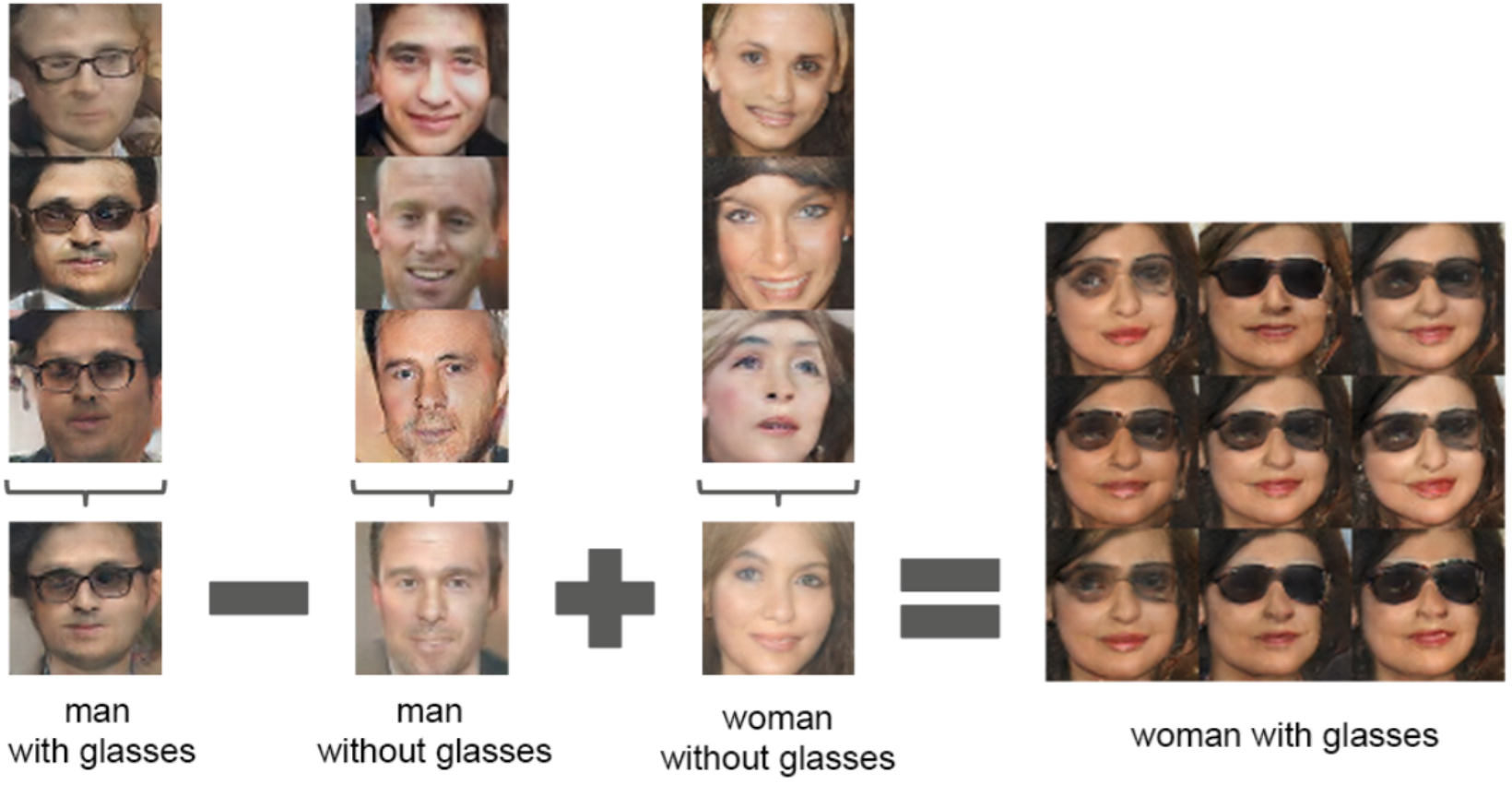
- **Pros**:
  - Learning process does not require approximations such as variational inference.
  - When $\max_{\theta_D} v(\theta_G, \theta_D)$ is convex in θ_D, the procedure is guaranteed to converge.
- **Cons**:
  - Learning in GANs can be difficult in practice when G and D are represented by neural networks and $\max_{\theta_D} v(\theta_G, \theta_D)$ is not convex.
  - Optimization seeks a saddle point. Simultaneous gradient descent on two players' costs is not guaranteed to reach an equilibrium.

- **Pros**:
  - Learning process does not require approximations such as variational inference.
  - When $\max_{\theta_D} v(\theta_G, \theta_D)$ is convex in θ_D, the procedure is guaranteed to converge.
- **Cons**:
  - Learning in GANs can be difficult in practice when G and D are represented by neural networks and $\max_{\theta_D} v(\theta_G, \theta_D)$ is not convex.
  - Optimization seeks a saddle point. Simultaneous gradient descent on two players' costs is not guaranteed to reach an equilibrium.

- **Pros**:
  - Learning process does not require approximations such as variational inference.
  - When $\max_{\theta_D} v(\theta_G, \theta_D)$ is convex in θ_D, the procedure is guaranteed to converge.
- **Cons**:
  - Learning in GANs can be difficult in practice when G and D are represented by neural networks and $\max_{\theta_D} v(\theta_G, \theta_D)$ is not convex.
  - Optimization seeks a saddle point. Simultaneous gradient descent on two players' costs is not guaranteed to reach an equilibrium.

Generative Adversarial Network

GAN (Goodfellow et al., 2014)



Plug & Play Generative Networks:
Nguyen et al, 2016

Generative Face Completion

Li et al, 2017

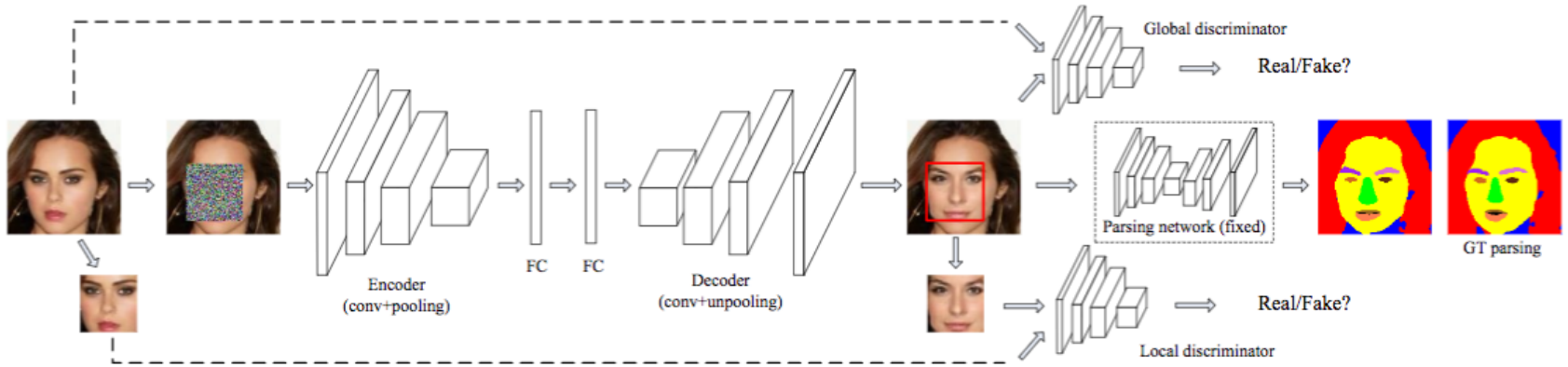


Image-to-Image Translation

Isola et al, 2017

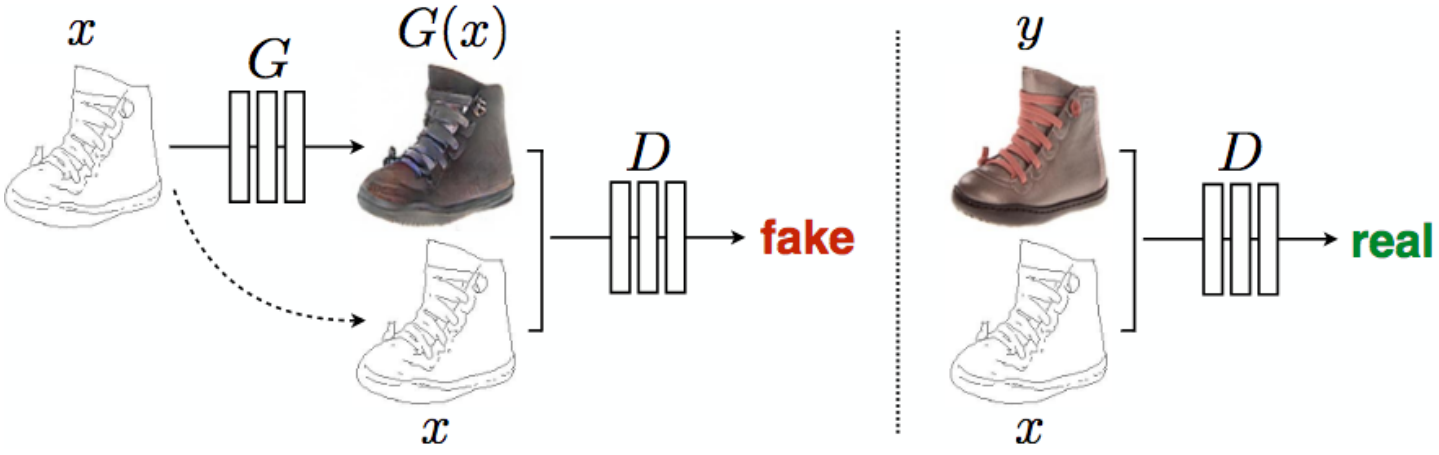
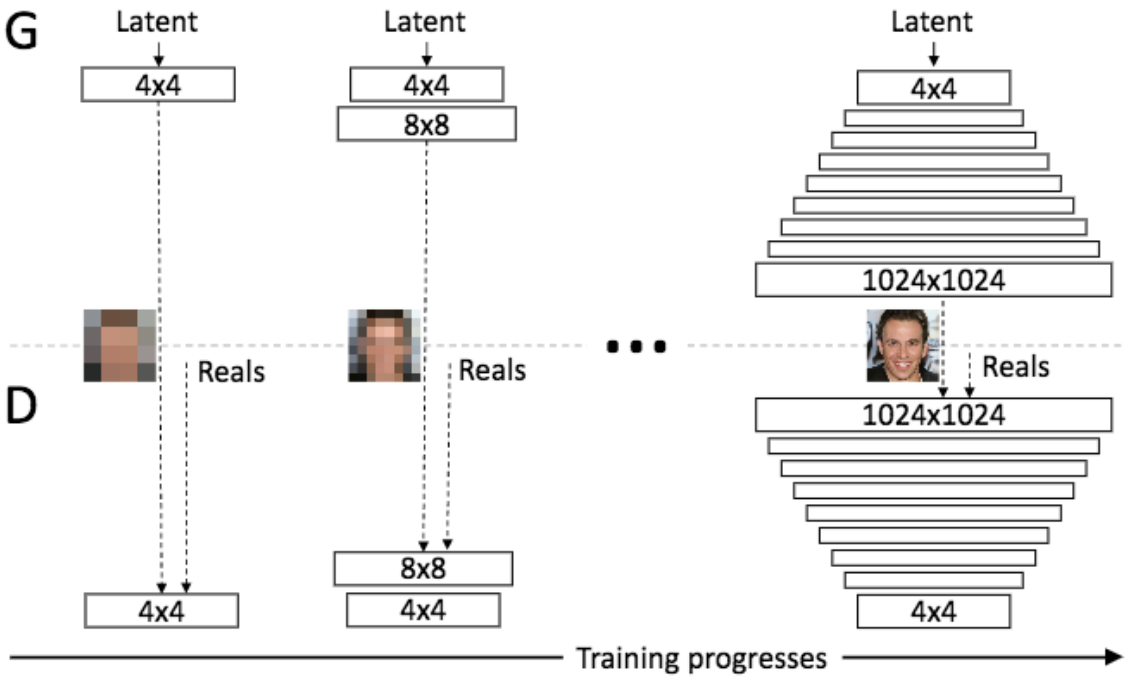


Image-to-Image Translation

Isola et al, 2017

Progressive growing of GANs

Karras et al, 2018



Progressive growing of GANs

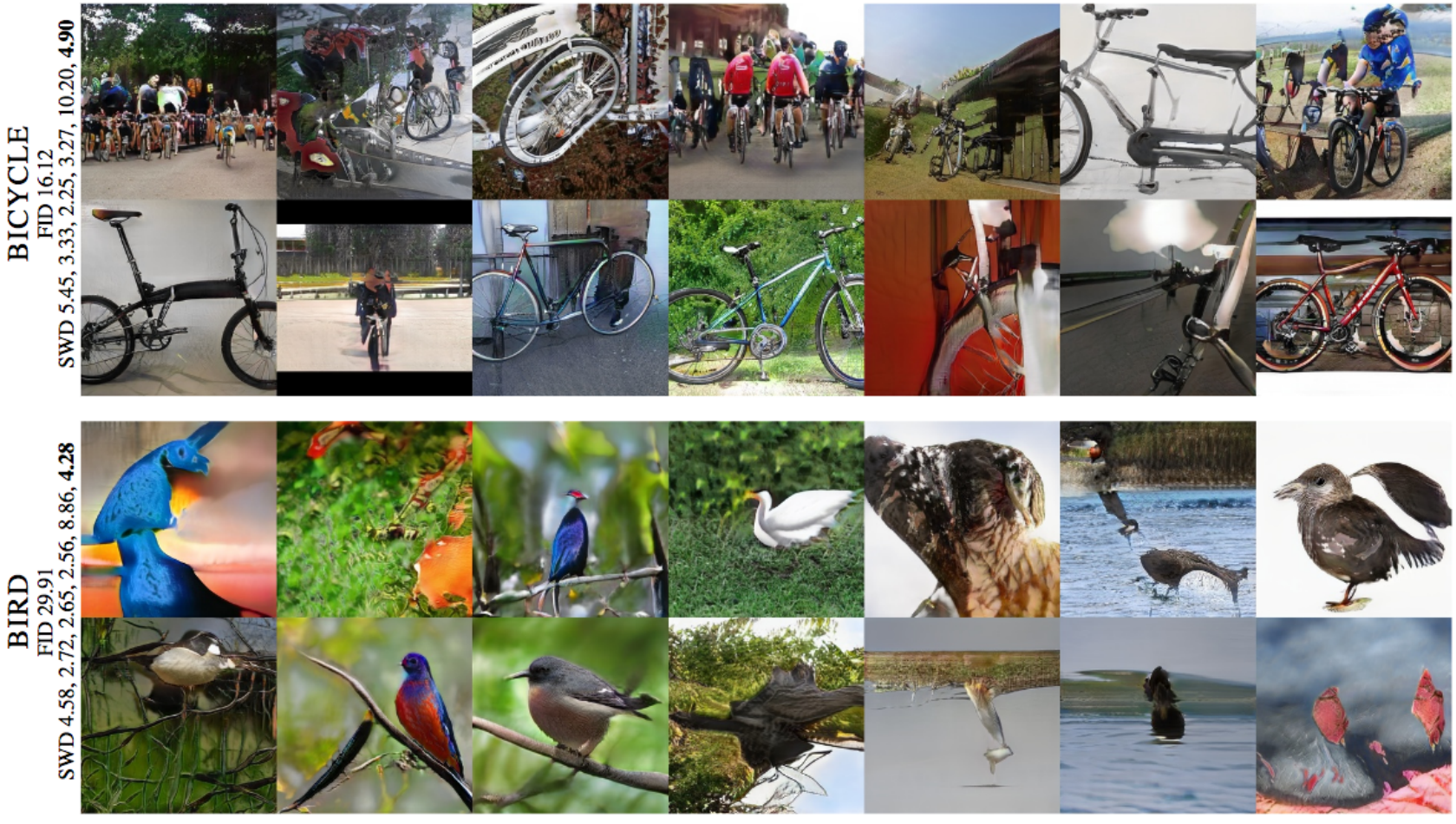
Karras et al, 2018

Progressive growing of GANs

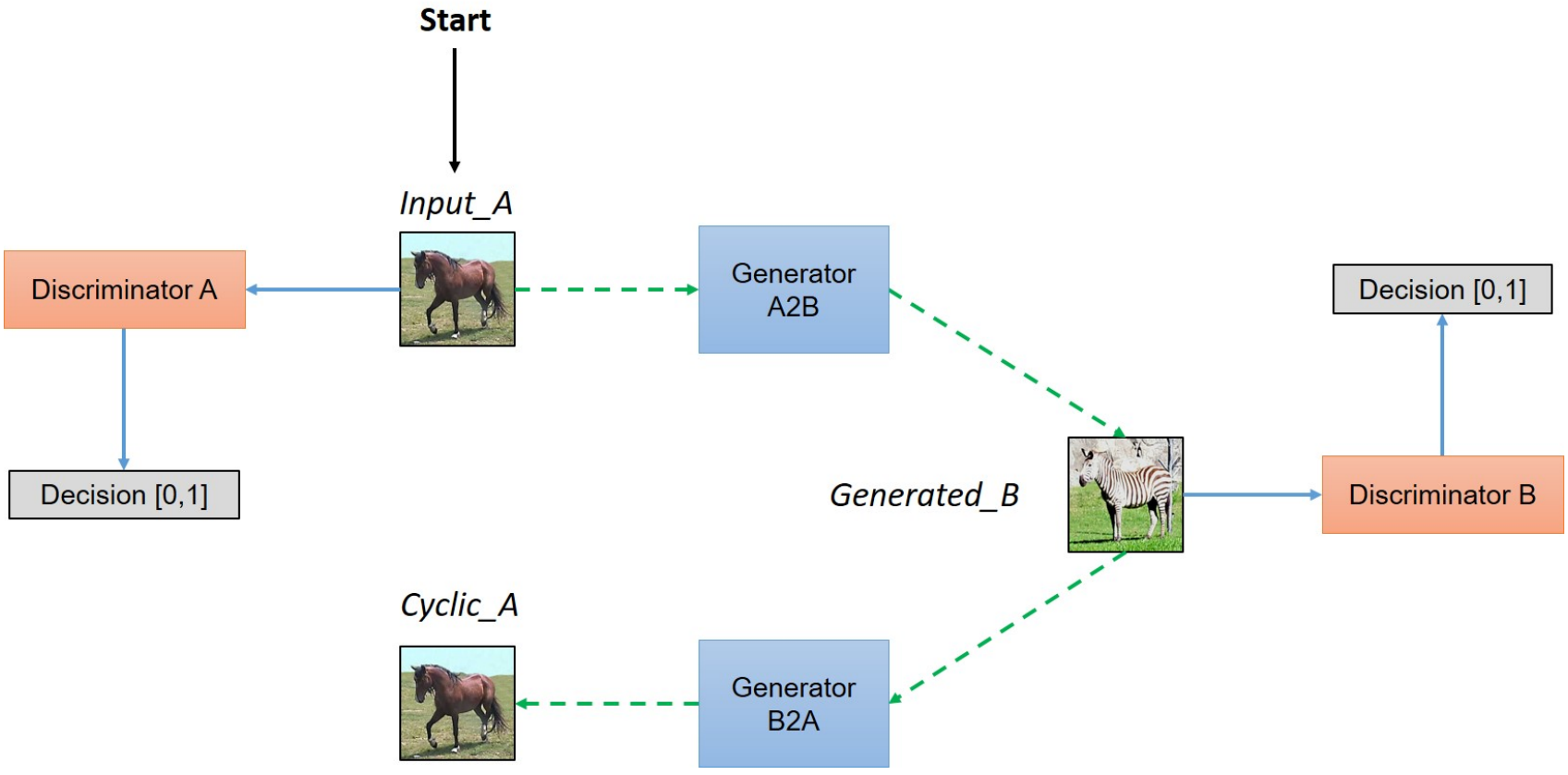
Karras et al, 2018

Progressive growing of GANs

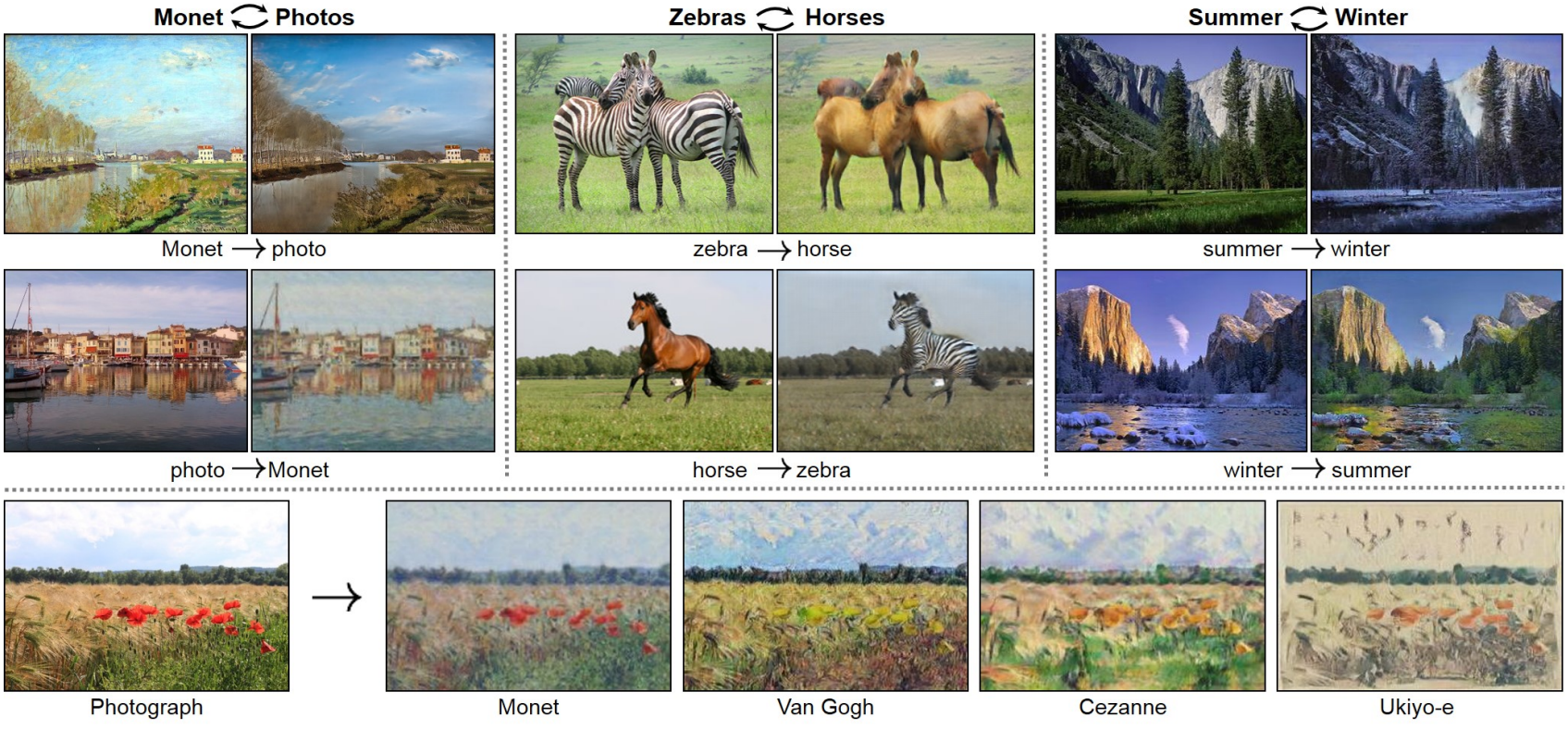
Karras et al, 2018



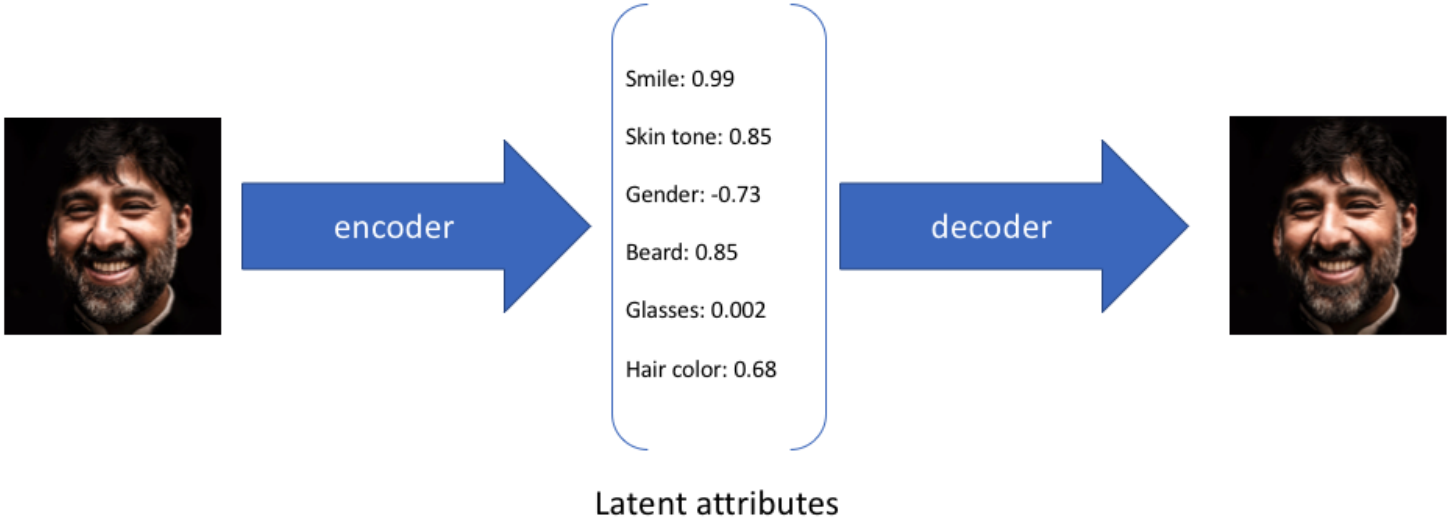
Cycle GANs
Zhu et al, ICCV 2017



Cycle GANs
Zhu et al, ICCV 2017



Reminder: Autoencoders

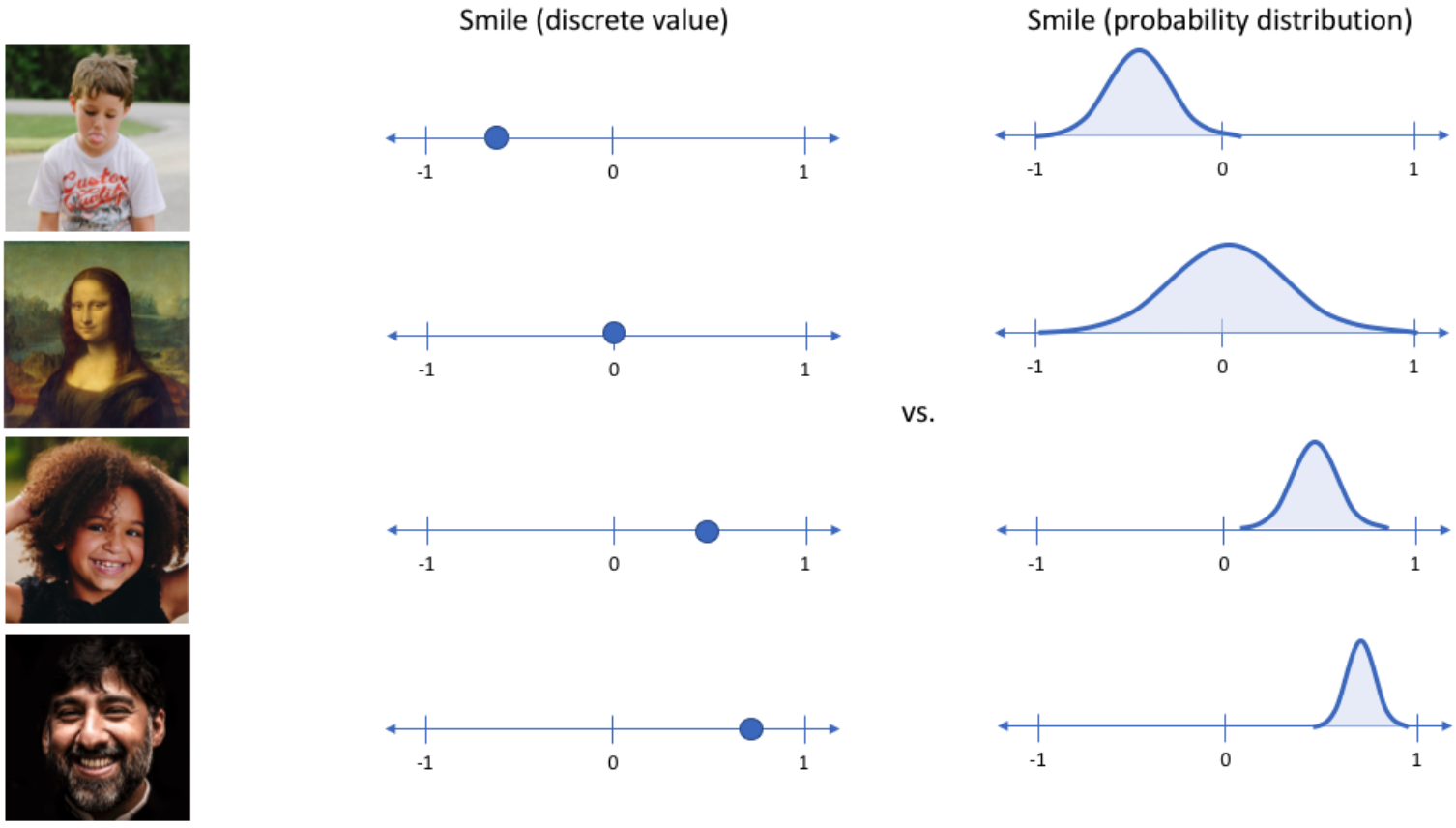


Latent variables encode "essential" information about data points $\mathbf{x}$.

Variational Autoencoders (VAEs)

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

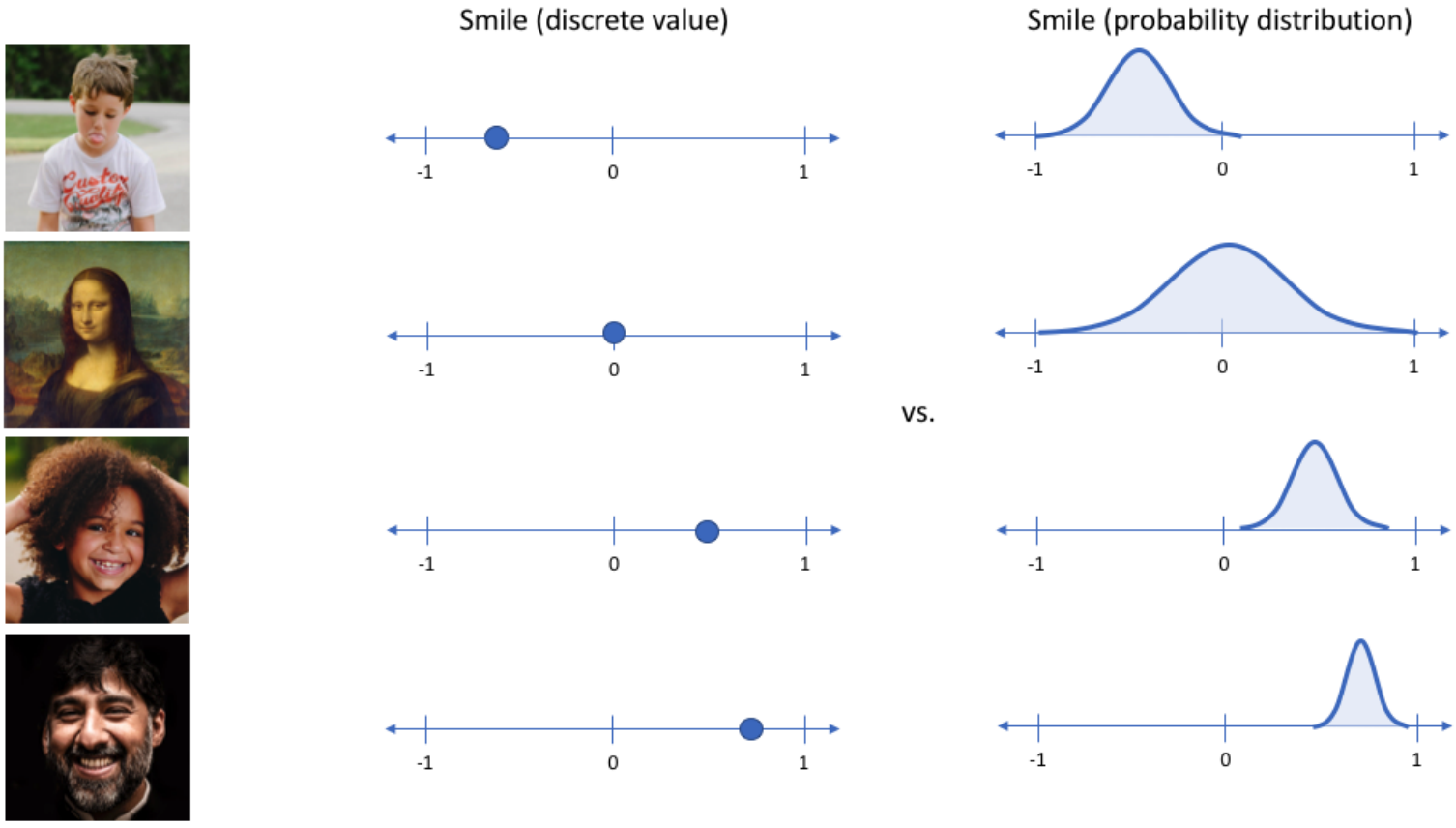
- Instead of a single value for each attribute, represent each latent attribute as a range of possible values.
- E.g., what single value would you assign for the smile attribute if you feed in a photo of the Mona Lisa?
- VAE: describe latent attributes in probabilistic terms.



Variational Autoencoders (VAEs)

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

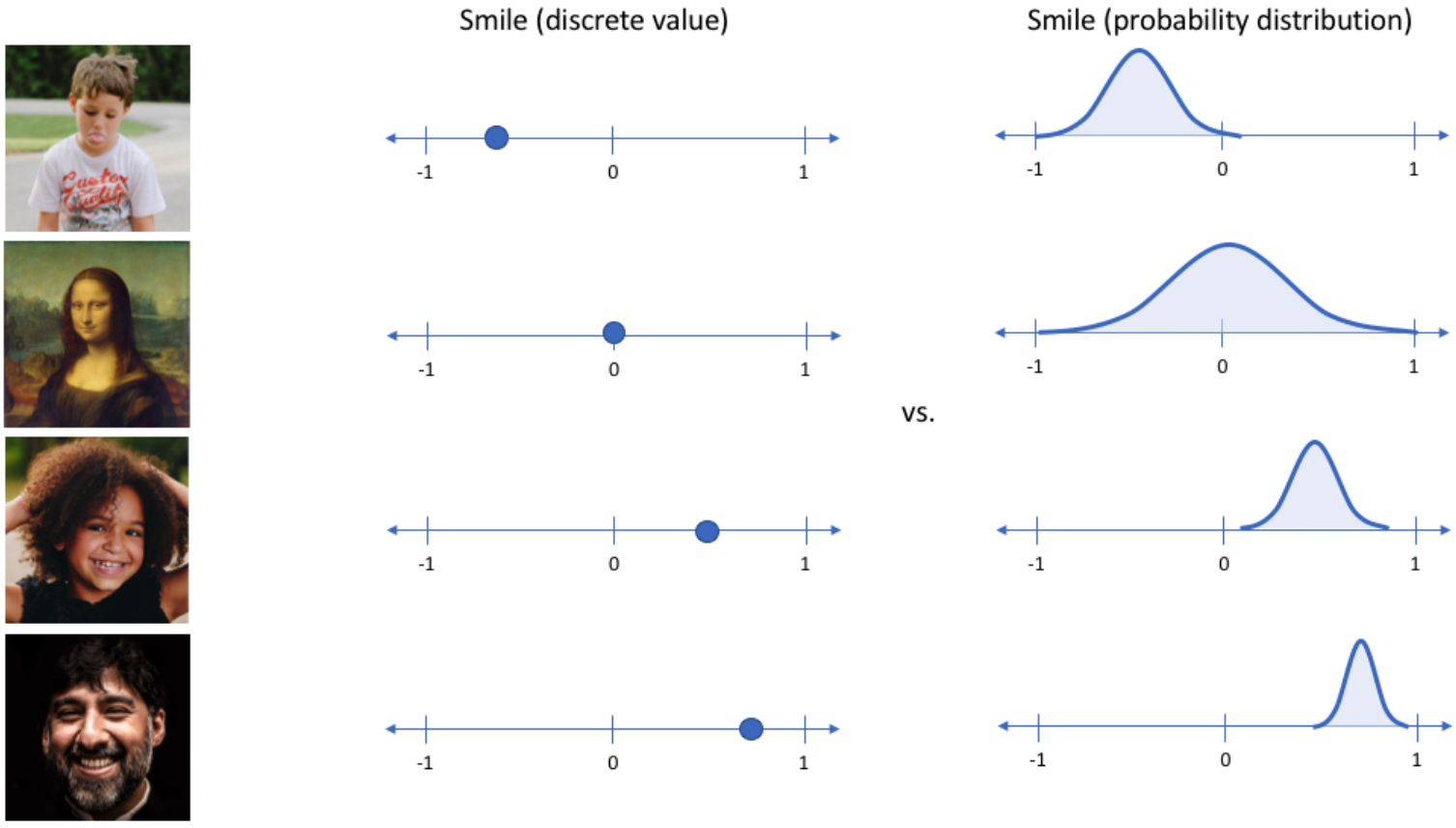
- Instead of a single value for each attribute, represent each latent attribute as a range of possible values.
- E.g., what single value would you assign for the smile attribute if you feed in a photo of the Mona Lisa?
- VAE: describe latent attributes in probabilistic terms.



Variational Autoencoders (VAEs)

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

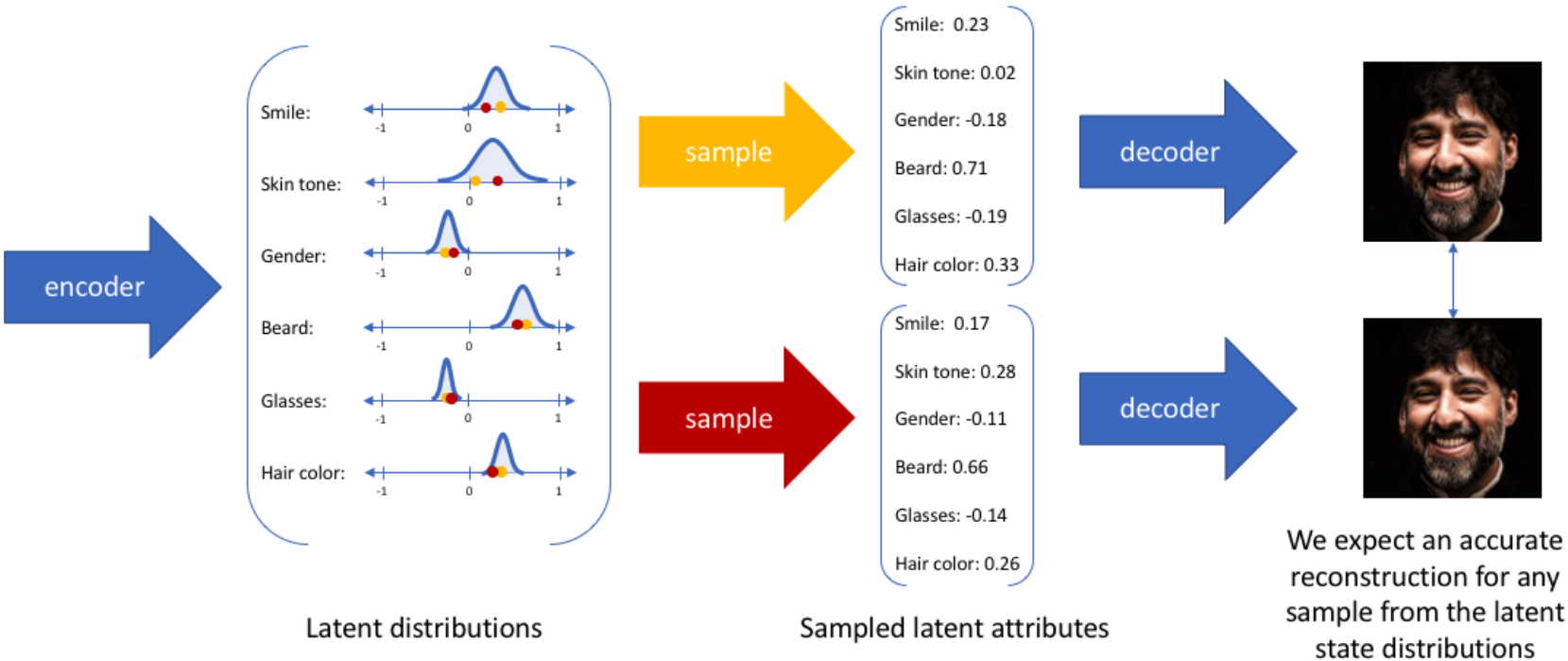
- Instead of a single value for each attribute, represent each latent attribute as a range of possible values.
- E.g., what single value would you assign for the smile attribute if you feed in a photo of the Mona Lisa?
- VAE: describe latent attributes in probabilistic terms.



Variational Autoencoders (VAEs)

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

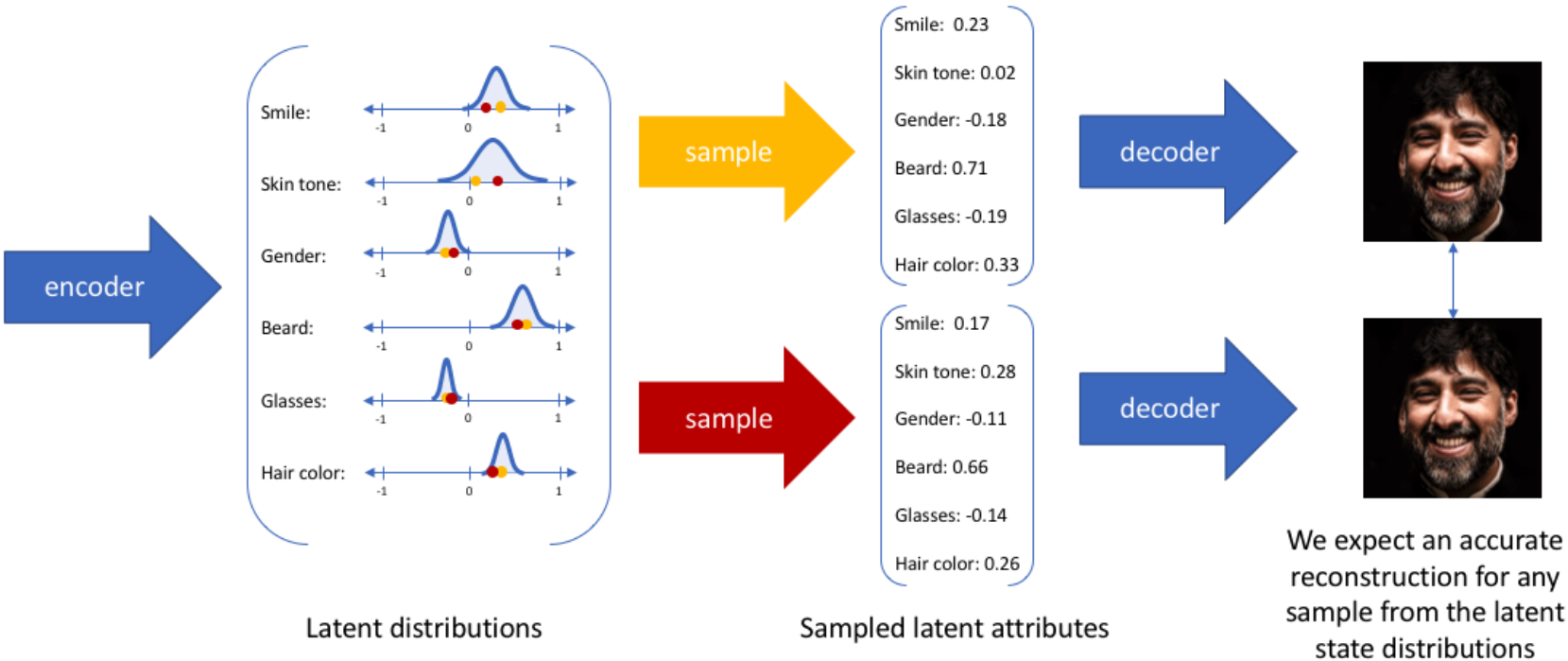
- **Encoder**: define probability distribution of latent variables
- **Sample**: latent variables **z** given the encoding of input **x**
- **Decode**: **z** so as to reconstruct corresponding input **x**.
- Enforces a continuous, smooth latent space representation.
- Values which are nearby to one another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



Variational Autoencoders (VAEs)

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

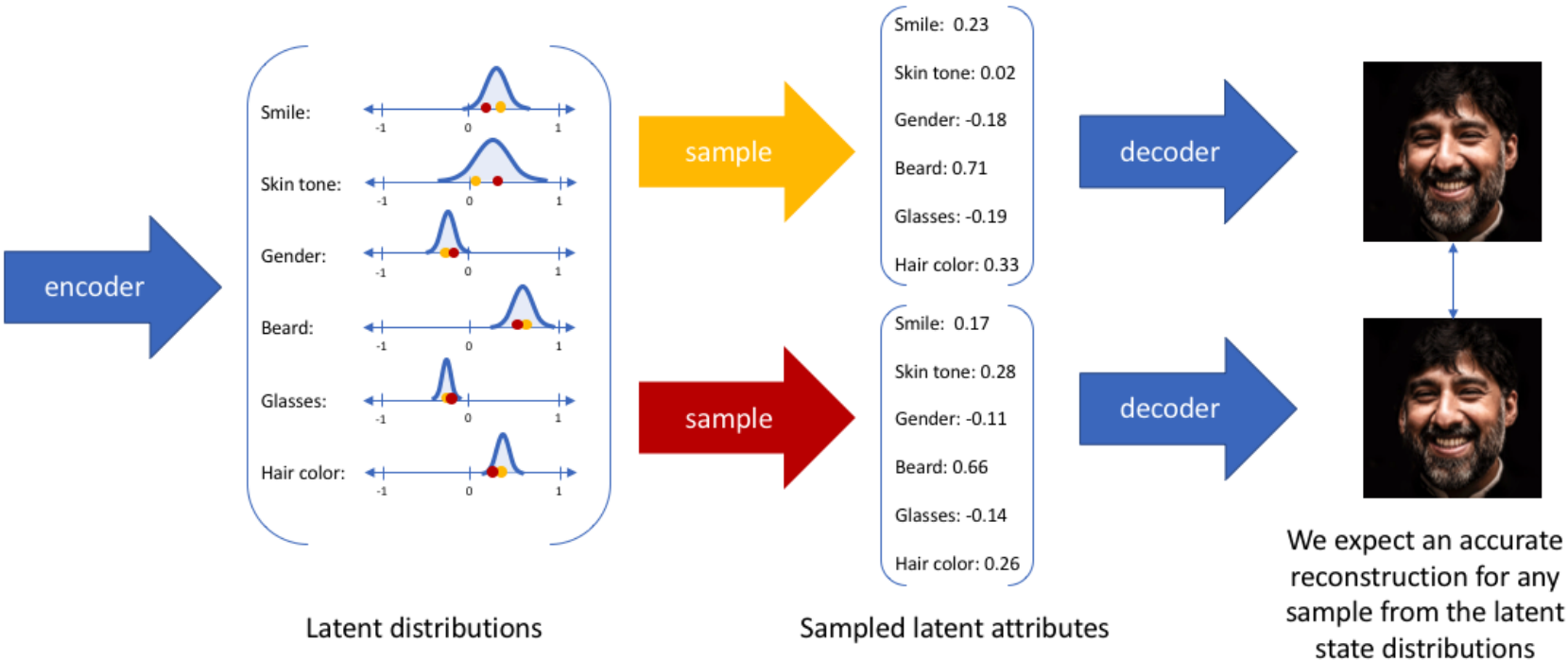
- **Encoder**: define probability distribution of latent variables
- **Sample**: latent variables **z** given the encoding of input **x**
- **Decode**: **z** so as to reconstruct corresponding input **x**.
- Enforces a continuous, smooth latent space representation.
- Values which are nearby to one another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



Variational Autoencoders (VAEs)

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

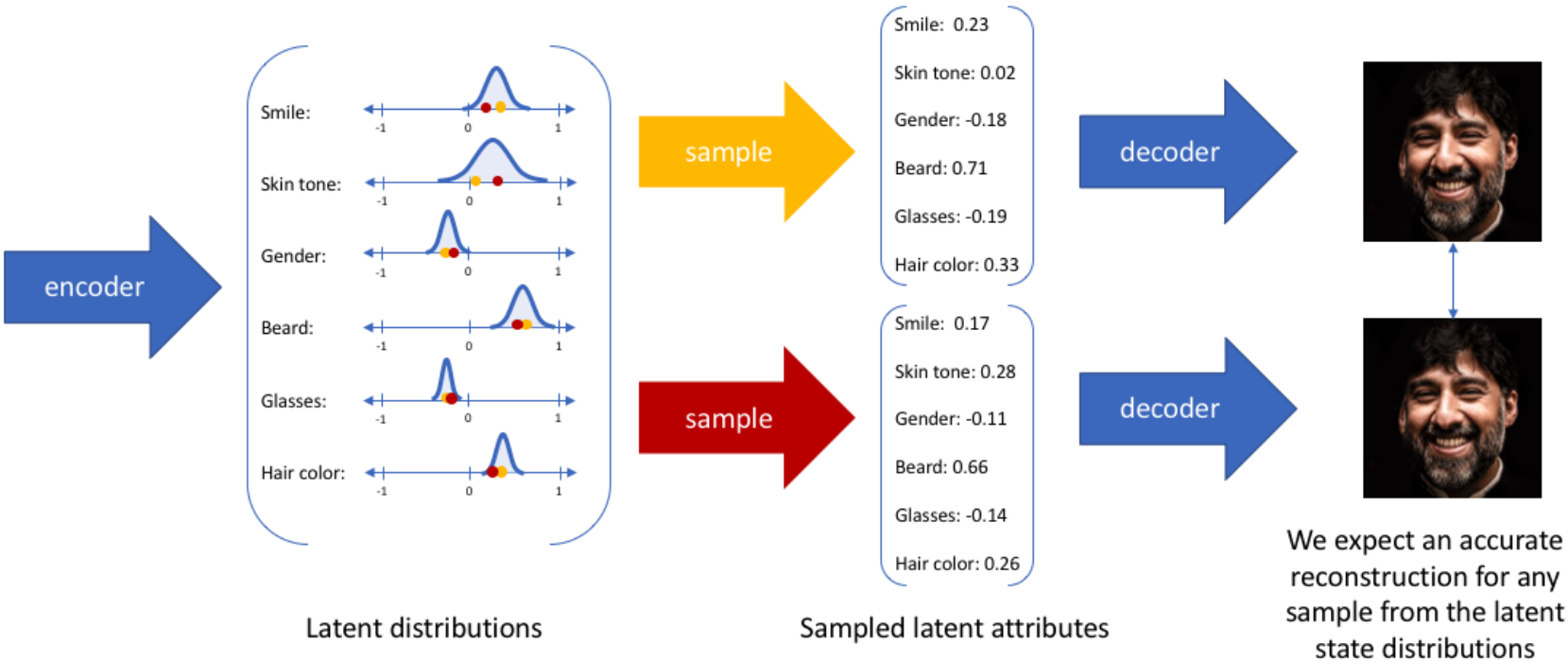
- **Encoder**: define probability distribution of latent variables
- **Sample**: latent variables **z** given the encoding of input **x**
- **Decode**: **z** so as to reconstruct corresponding input **x**.
- Enforces a continuous, smooth latent space representation.
- Values which are nearby to one another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



Variational Autoencoders (VAEs)

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

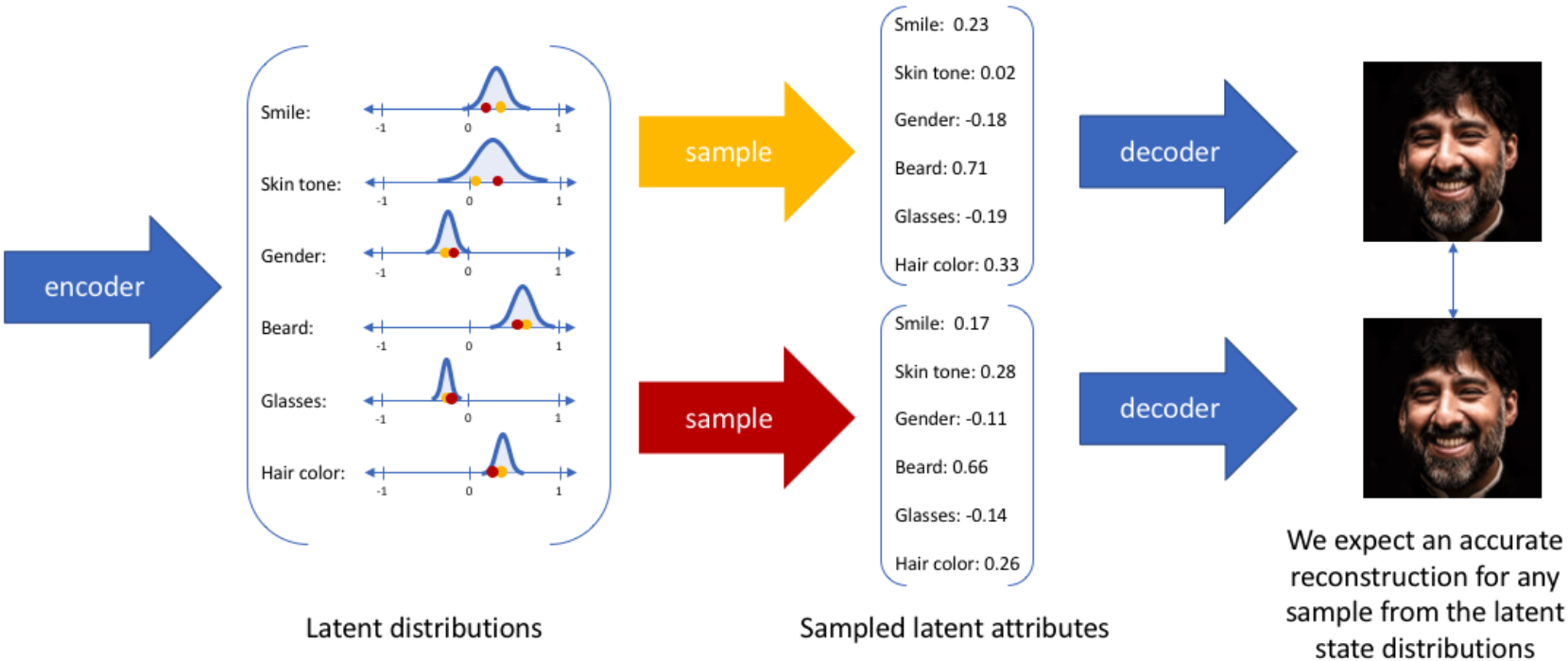
- **Encoder**: define probability distribution of latent variables
- **Sample**: latent variables **z** given the encoding of input **x**
- **Decode**: **z** so as to reconstruct corresponding input **x**.
- Enforces a continuous, smooth latent space representation.
- Values which are nearby to one another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



Variational 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.



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.

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.

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.

Variational Approximation

- Physical system with configuration $\mathbf{x}$ and energy function $u(\mathbf{x})$.

$$p(\mathbf{x}) = Z^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

with partition function:

$$Z = \int \mathrm{e}^{-u(\mathbf{x})} \mathrm{d}\mathbf{x}$$

- Free energy:

$$F_p = -\log Z_p = \langle u(\mathbf{x}) \rangle_p - H_p$$

with entropy $H_p = \int p(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$

- Approximate $p(\mathbf{x})$ with variational distribution $q(\mathbf{x}; \theta)$ with variational free energy

$$F_q = \langle u(\mathbf{x}) \rangle_q - H_q$$

- Variational free energy difference:

$$F_q(\theta) - F_p = \int q(\mathbf{x}; \theta) \left[\frac{\log q(\mathbf{x}; \theta)}{\log p(\mathbf{x})} \right] \mathrm{d}\mathbf{x} = KL(q \parallel p)$$

$KL \geq 0$ implies $F_q \geq F_p \rightarrow$ we can minimize variational free energy $F_q(\theta)$.

Variational Approximation

- Physical system with configuration $\mathbf{x}$ and energy function $u(\mathbf{x})$.

$$p(\mathbf{x}) = Z^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

with partition function:

$$Z = \int \mathrm{e}^{-u(\mathbf{x})} \mathrm{d}\mathbf{x}$$

- Free energy:

$$F_p = -\log Z_p = \langle u(\mathbf{x}) \rangle_p - H_p$$

with entropy $H_p = \int p(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$

- Approximate $p(\mathbf{x})$ with variational distribution $q(\mathbf{x}; \theta)$ with variational free energy

$$F_q = \langle u(\mathbf{x}) \rangle_q - H_q$$

- Variational free energy difference:

$$F_q(\theta) - F_p = \int q(\mathbf{x}; \theta) \left[\frac{\log q(\mathbf{x}; \theta)}{\log p(\mathbf{x})} \right] \mathrm{d}\mathbf{x} = KL(q \parallel p)$$

$KL \geq 0$ implies $F_q \geq F_p \rightarrow$ we can minimize variational free energy $F_q(\theta)$.

Variational Approximation

- Physical system with configuration $\mathbf{x}$ and energy function $u(\mathbf{x})$.

$$p(\mathbf{x}) = Z^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

with partition function:

$$Z = \int \mathrm{e}^{-u(\mathbf{x})} \mathrm{d}\mathbf{x}$$

- Free energy:

$$F_p = -\log Z_p = \langle u(\mathbf{x}) \rangle_p - H_p$$

with entropy $H_p = \int p(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$

- Approximate $p(\mathbf{x})$ with variational distribution $q(\mathbf{x}; \theta)$ with variational free energy

$$F_q = \langle u(\mathbf{x}) \rangle_q - H_q$$

- Variational free energy difference:

$$F_q(\theta) - F_p = \int q(\mathbf{x}; \theta) \left[\frac{\log q(\mathbf{x}; \theta)}{\log p(\mathbf{x})} \right] \mathrm{d}\mathbf{x} = KL(q \parallel p)$$

$KL \geq 0$ implies $F_q \geq F_p \rightarrow$ we can minimize variational free energy $F_q(\theta)$.

Variational Approximation

- Physical system with configuration $\mathbf{x}$ and energy function $u(\mathbf{x})$.

$$p(\mathbf{x}) = Z^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

with partition function:

$$Z = \int \mathrm{e}^{-u(\mathbf{x})} \mathrm{d}\mathbf{x}$$

- Free energy:

$$F_p = -\log Z_p = \langle u(\mathbf{x}) \rangle_p - H_p$$

with entropy $H_p = \int p(\mathbf{x}) \log p(\mathbf{x}) \mathrm{d}\mathbf{x}$

- Approximate $p(\mathbf{x})$ with variational distribution $q(\mathbf{x}; \theta)$ with variational free energy

$$F_q = \langle u(\mathbf{x}) \rangle_q - H_q$$

- Variational free energy difference:

$$F_q(\theta) - F_p = \int q(\mathbf{x}; \theta) \left[\frac{\log q(\mathbf{x}; \theta)}{\log p(\mathbf{x})} \right] \mathrm{d}\mathbf{x} = KL(q \parallel p)$$

$KL \geq 0$ implies $F_q \geq F_p \rightarrow$ we can minimize variational free energy $F_q(\theta)$.

Variational Mean Field of Ising Model

- Ising model energy, spin variables $x_i \in \{-1, +1\}$.

$$u(\mathbf{x}) = -\frac{1}{2}\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$$

- Mean field approximation: choose spins to be independent:

$$q(\mathbf{x};\theta) = \frac{1}{Z_q}\exp\left(\sum_i \theta_i x_i\right) = \prod_i \frac{e^{\theta_i x_i}}{2\cosh\theta_i}.$$

- Entropy separates into one-body terms:

$$H_q = -\sum_i q_i \log q_i + (1-q_i)\log(1-q_i)$$

with probability of spin i to be $+1$:

$$q_i = e^{\theta_i}/2\cosh\theta_i$$

Variational Mean Field of Ising Model

- Ising model energy, spin variables $x_i \in \{-1, +1\}$.

$$u(\mathbf{x}) = -\frac{1}{2}\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$$

- Mean field approximation: choose spins to be independent:

$$q(\mathbf{x}; \theta) = \frac{1}{Z_q} \exp\left(\sum_i \theta_i x_i\right) = \prod_i \frac{e^{\theta_i x_i}}{2\cosh\theta_i}.$$

- Entropy separates into one-body terms:

$$H_q = -\sum_i q_i \log q_i + (1 - q_i)\log(1 - q_i)$$

with probability of spin i to be $+1$:

$$q_i = e^{\theta_i}/2\cosh\theta_i$$

Variational Mean Field of Ising Model

- Ising model energy, spin variables $x_i \in \{-1, +1\}$.

$$u(\mathbf{x}) = -\frac{1}{2}\sum_{i,j} J_{ij} x_i x_j - \sum_i h_i x_i$$

- Mean field approximation: choose spins to be independent:

$$q(\mathbf{x}; \theta) = \frac{1}{Z_q} \exp\left(\sum_i \theta_i x_i\right) = \prod_i \frac{e^{\theta_i x_i}}{2\cosh\theta_i}.$$

- Entropy separates into one-body terms:

$$H_q = -\sum_i q_i \log q_i + (1 - q_i)\log(1 - q_i)$$

with probability of spin i to be $+1$:

$$q_i = e^{\theta_i}/2\cosh\theta_i$$

Variational Mean Field of Ising Model

- Mean energy separates into one-body terms:

$$\langle u(\mathbf{x})\rangle_q = -\frac{1}{2}\sum_{i,j} J_{ij} m_i m_j - \sum_i h_i m_i$$

with mean value of spin i:

$$m_i = \langle x_i \rangle_q = \sum_{x_i=\pm 1} x_i q_i = \tanh \theta_i \tag{1}$$

- Minimize variational free energy $F_q = \langle u(\mathbf{x})\rangle_q - H_q$ by taking derivatives and setting to zero leads to:

$$\theta_i = \sum_j J_{ij} m_j(\theta_j) + h_i \tag{2}$$

- Optimize θ by iterating (1-2).

Variational Mean Field of Ising Model

- Mean energy separates into one-body terms:

$$\langle u(\mathbf{x})\rangle_q = -\frac{1}{2}\sum_{i,j} J_{ij} m_i m_j - \sum_i h_i m_i$$

  with mean value of spin i:

$$m_i = \langle x_i\rangle_q = \sum_{x_i=\pm 1} x_i q_i = \tanh\theta_i \tag{1}$$

- Minimize variational free energy $F_q = \langle u(\mathbf{x})\rangle_q - H_q$ by taking derivatives and setting to zero leads to:

$$\theta_i = \sum_j J_{ij} m_j(\theta_j) + h_i \tag{2}$$

- Optimize θ by iterating (1-2).

Variational Mean Field of Ising Model

- Mean energy separates into one-body terms:

$$\langle u(\mathbf{x})\rangle_q = -\frac{1}{2}\sum_{i,j} J_{ij} m_i m_j - \sum_i h_i m_i$$

  with mean value of spin i:

$$m_i = \langle x_i\rangle_q = \sum_{x_i=\pm 1} x_i q_i = \tanh\theta_i \tag{1}$$

- Minimize variational free energy $F_q = \langle u(\mathbf{x})\rangle_q - H_q$ by taking derivatives and setting to zero leads to:

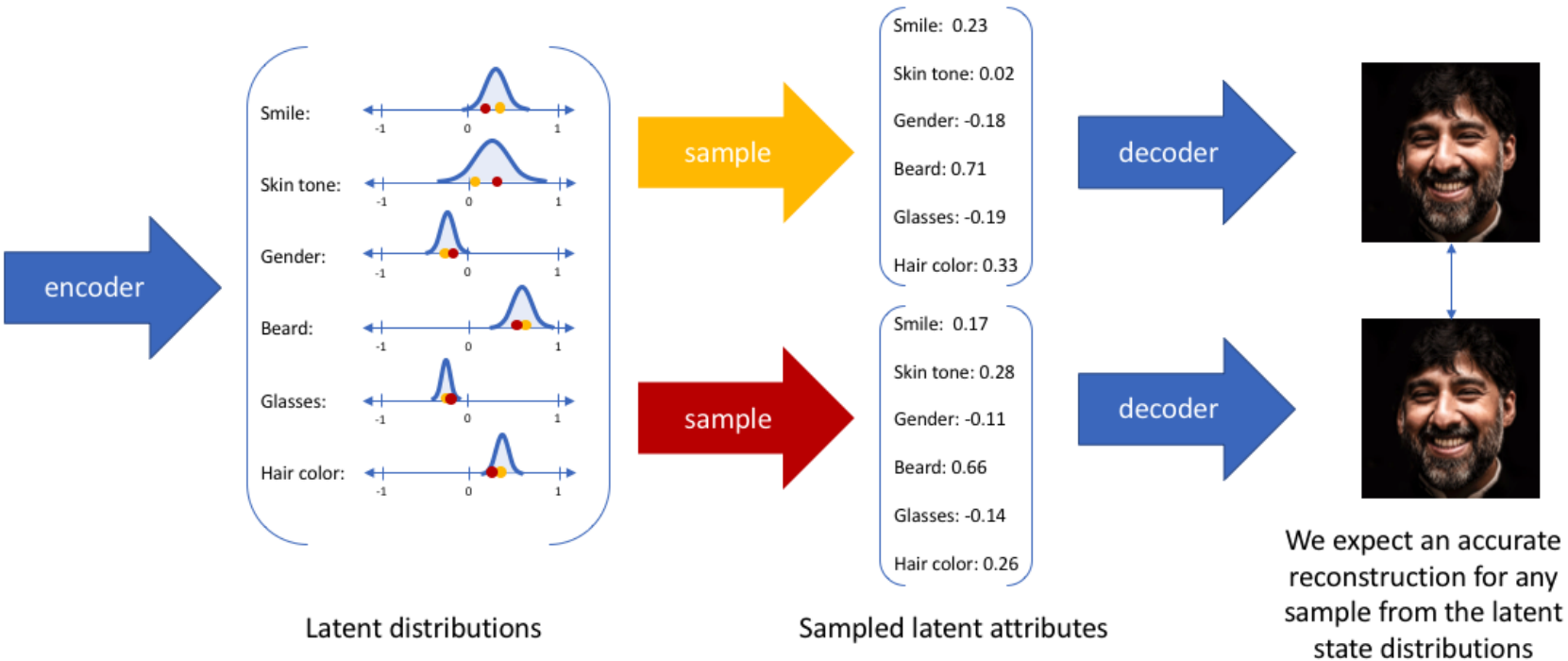
$$\theta_i = \sum_j J_{ij} m_j(\theta_j) + h_i \tag{2}$$

- Optimize θ by iterating (1-2).

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**.



Variational Inference for latent variable models

- 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.

Variational Inference for latent variable models

- 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.

Variational Inference for latent variable models

- 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.

Variational Inference for latent variable models

- 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.

- **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?

- **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?

- **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 Bayes equation and some basic algebra:

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

- **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):

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

- Maximize model likelihood by minimizing $D_{\mathrm{KL}}(q \parallel p)$ or maximizing the ELBO:

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

- Using Bayes equation and some basic algebra:

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

- **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):

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

- Maximize model likelihood by minimizing $D_{\mathrm{KL}}(q \parallel p)$ or maximizing the ELBO:

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

- Using Bayes equation and some basic algebra:

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

- **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):

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

- Maximize model likelihood by minimizing $D_{\mathrm{KL}}(q \parallel p)$ or maximizing the ELBO:

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

- Using Bayes equation and some basic algebra:

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

- **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):

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

- Maximize model likelihood by minimizing $D_{\mathrm{KL}}(q \parallel p)$ or maximizing the ELBO:

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

- Using Bayes equation and some basic algebra:

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

- Interpreting these terms:

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

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

- Using Bayes equation and some basic algebra:

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

- 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)

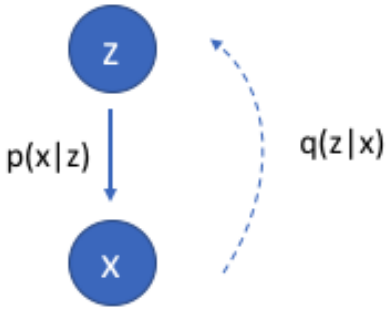
- Using Bayes equation and some basic algebra:

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

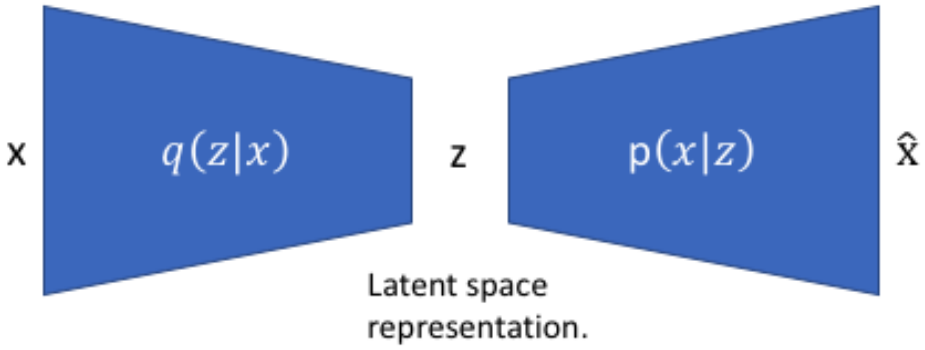
- Interpreting these terms:

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

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



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



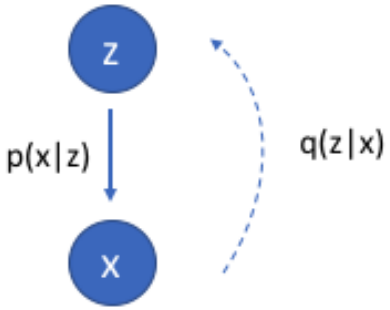
Latent space representation.

Neural network mapping x to z.

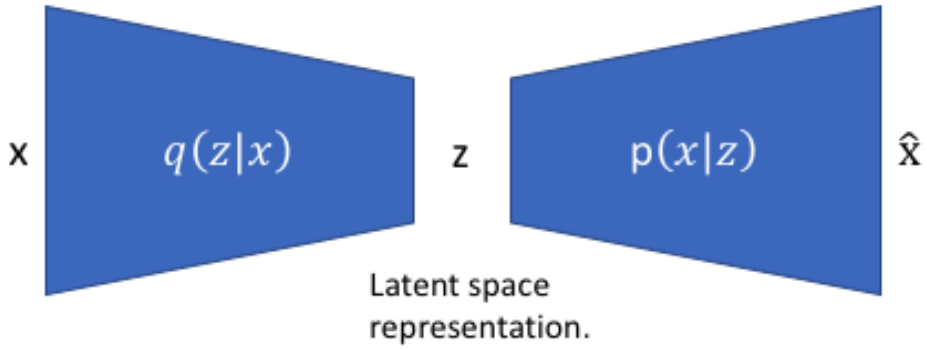
Neural network mapping z to x.

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

- **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



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



Latent space representation.

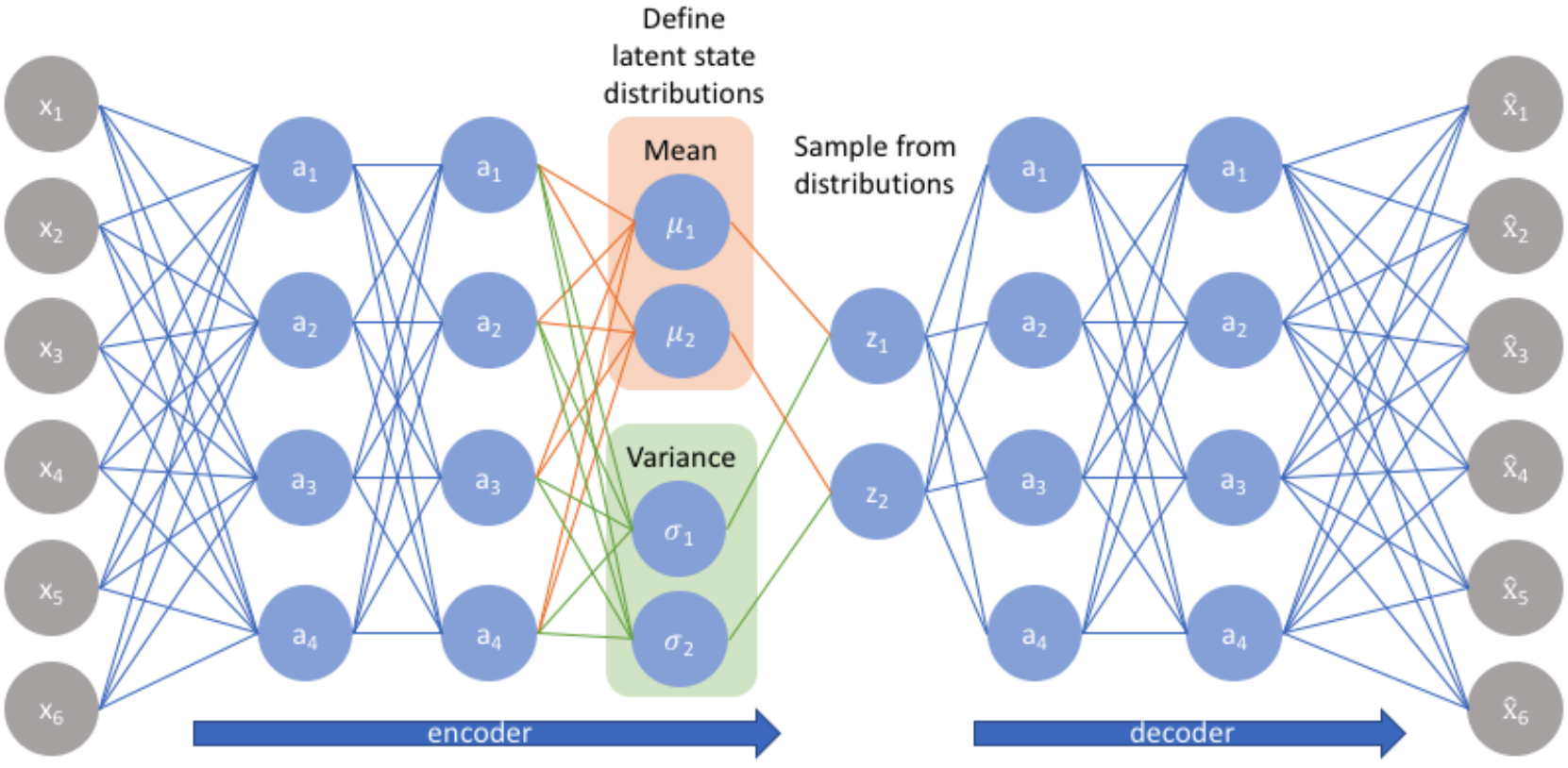
Neural network mapping x to z.

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.

- **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.

- **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

- 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

- 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

- 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 $\mathbf{x}$:

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

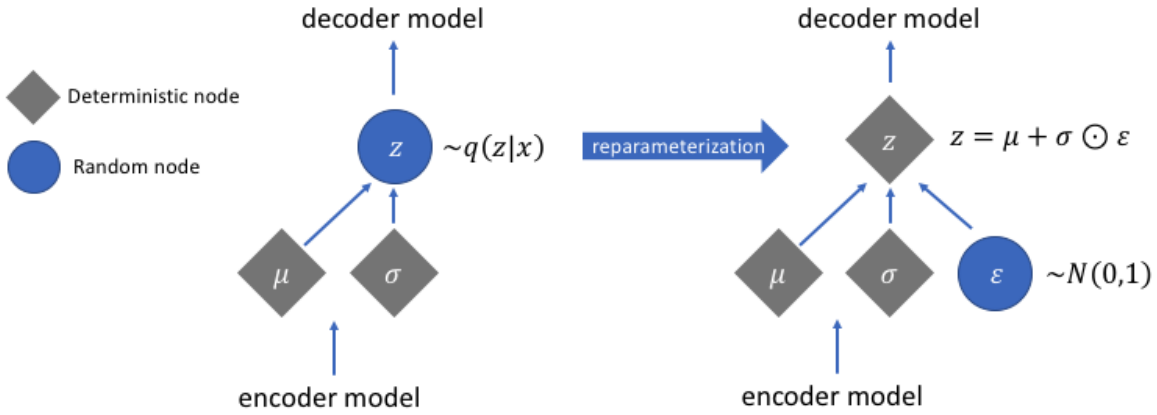
- However, the process of sampling a pdf itself is not differentiable. In order to compute derivatives, we use the reparametrization trick:
  1. Randomly sample ε from a unit Gaussian

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

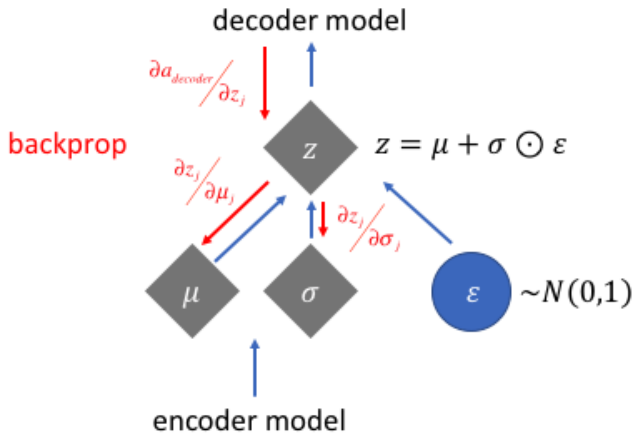
  2. Shift ε by mean and scale it by variance of the latent distribution:

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

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 exp.

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)

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

- 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!

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!

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)

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

- 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!

Computing the Reconstruction Loss

Evaluations Reconstruction Loss by Sample

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

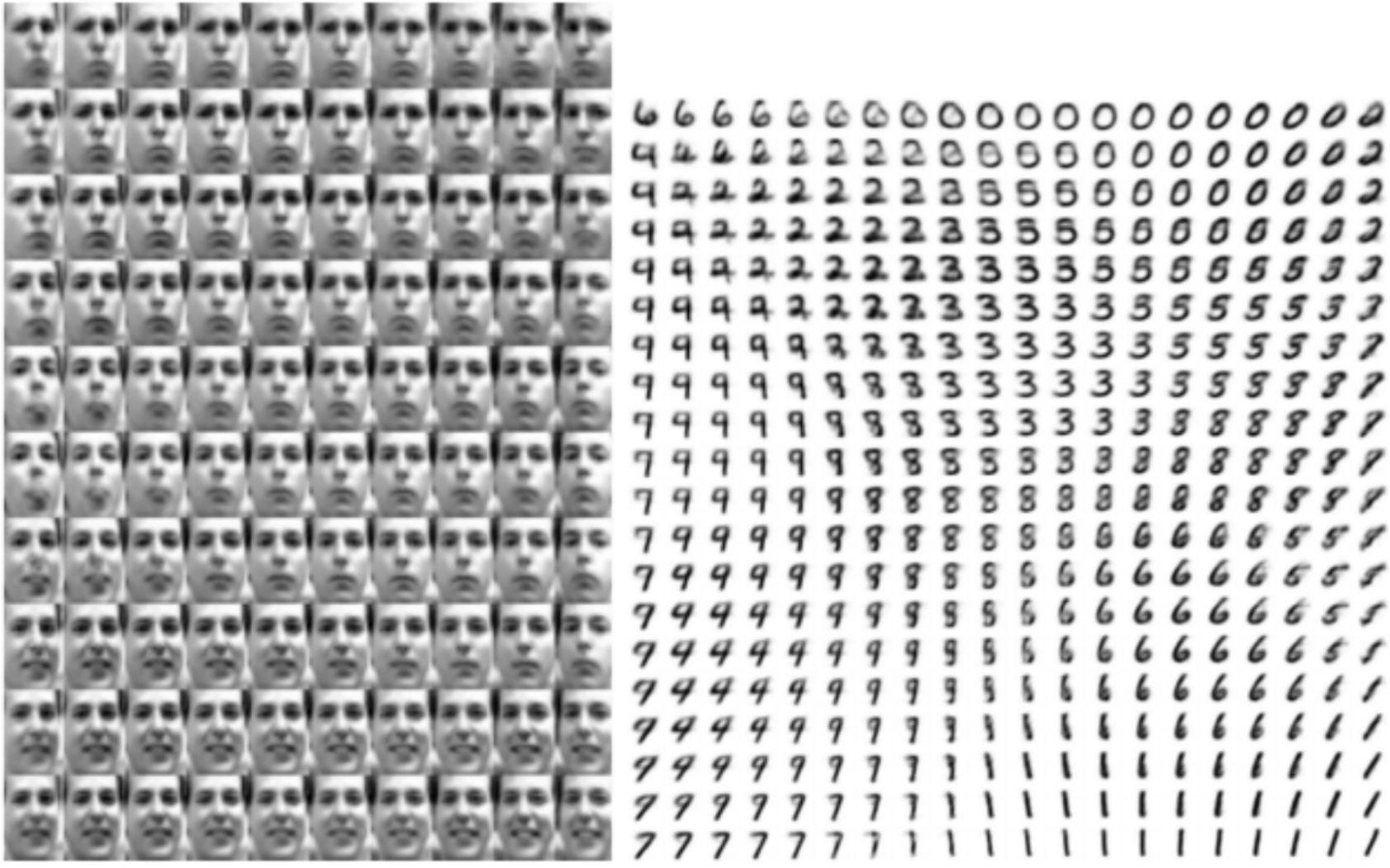
$$\log p(\mathbf{x} \mid \mathbf{z}; \theta)$$

- **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)

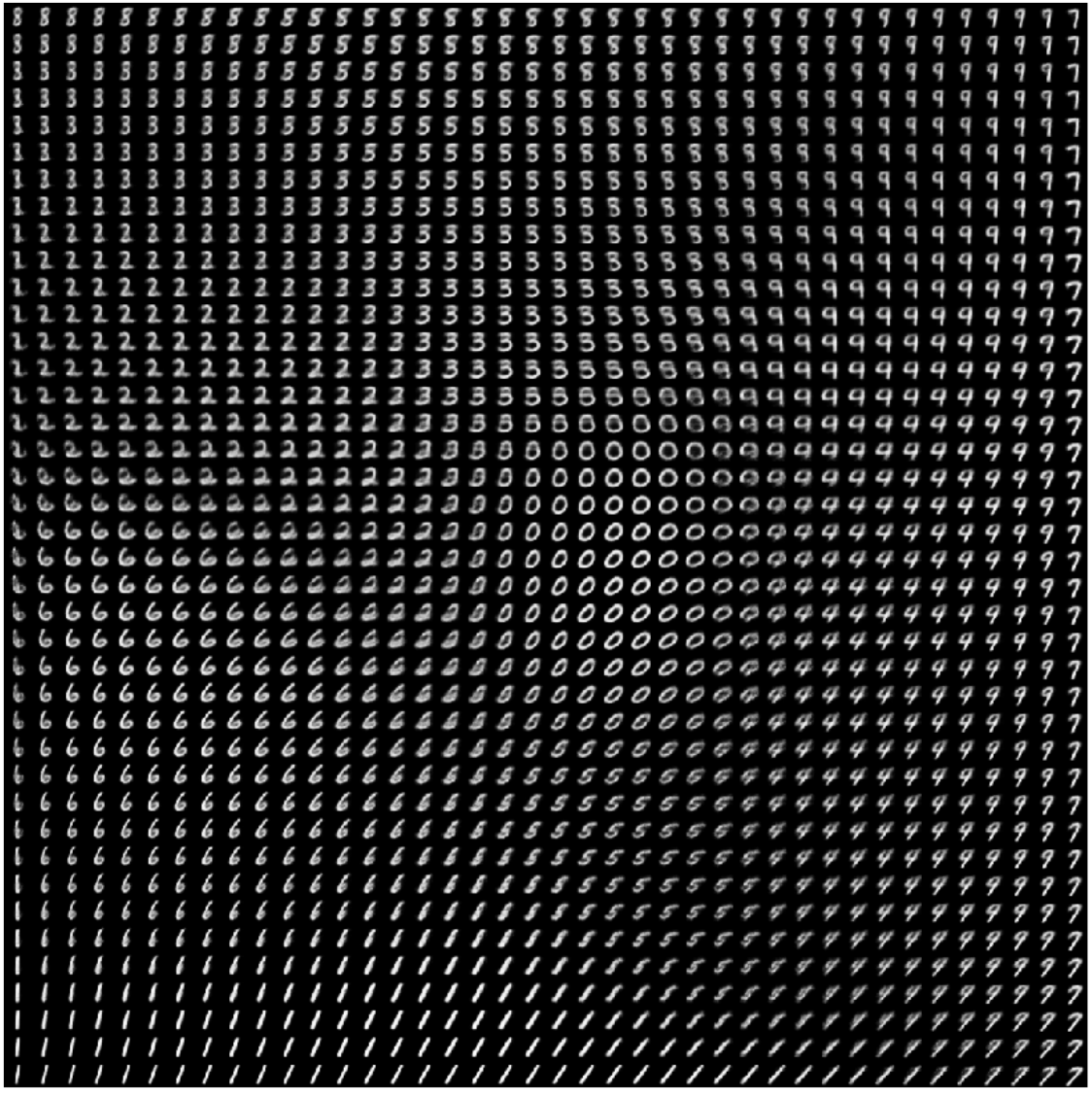
$$L(\theta) = \sum_{i=1}^{N} x_i \log \hat{x}_i(\mathbf{x}, \theta) + (1 - x_i) \log \left[1 - \hat{x}_i(\mathbf{x}, \theta)\right]$$

- 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

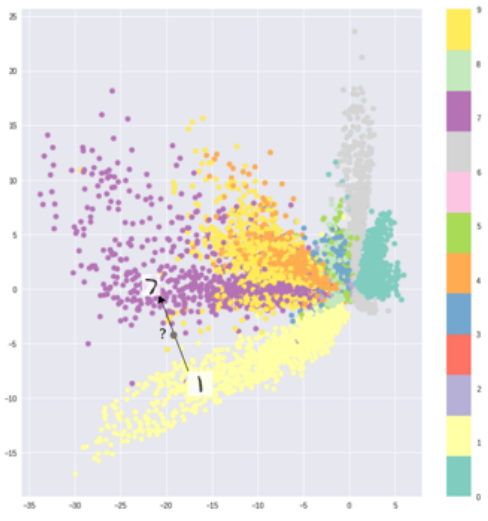


Variational Autoencoders

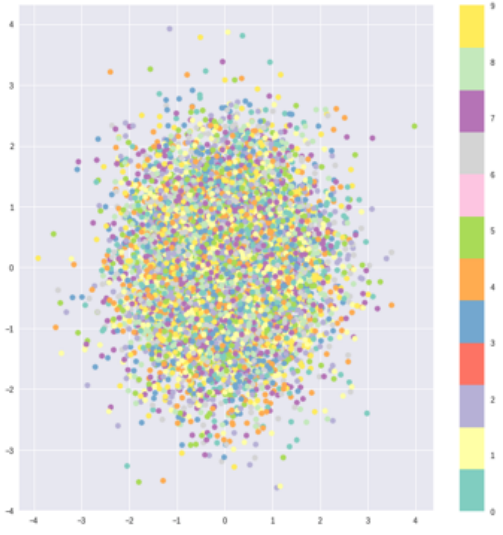


$$\min \left\{ -\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z}) + D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})) \right\}$$

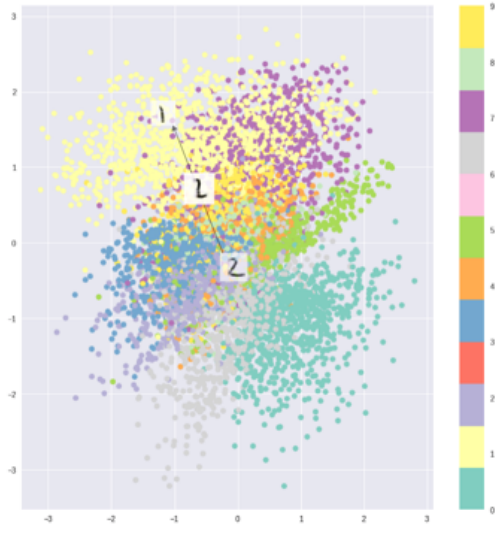
Only reconstruction loss

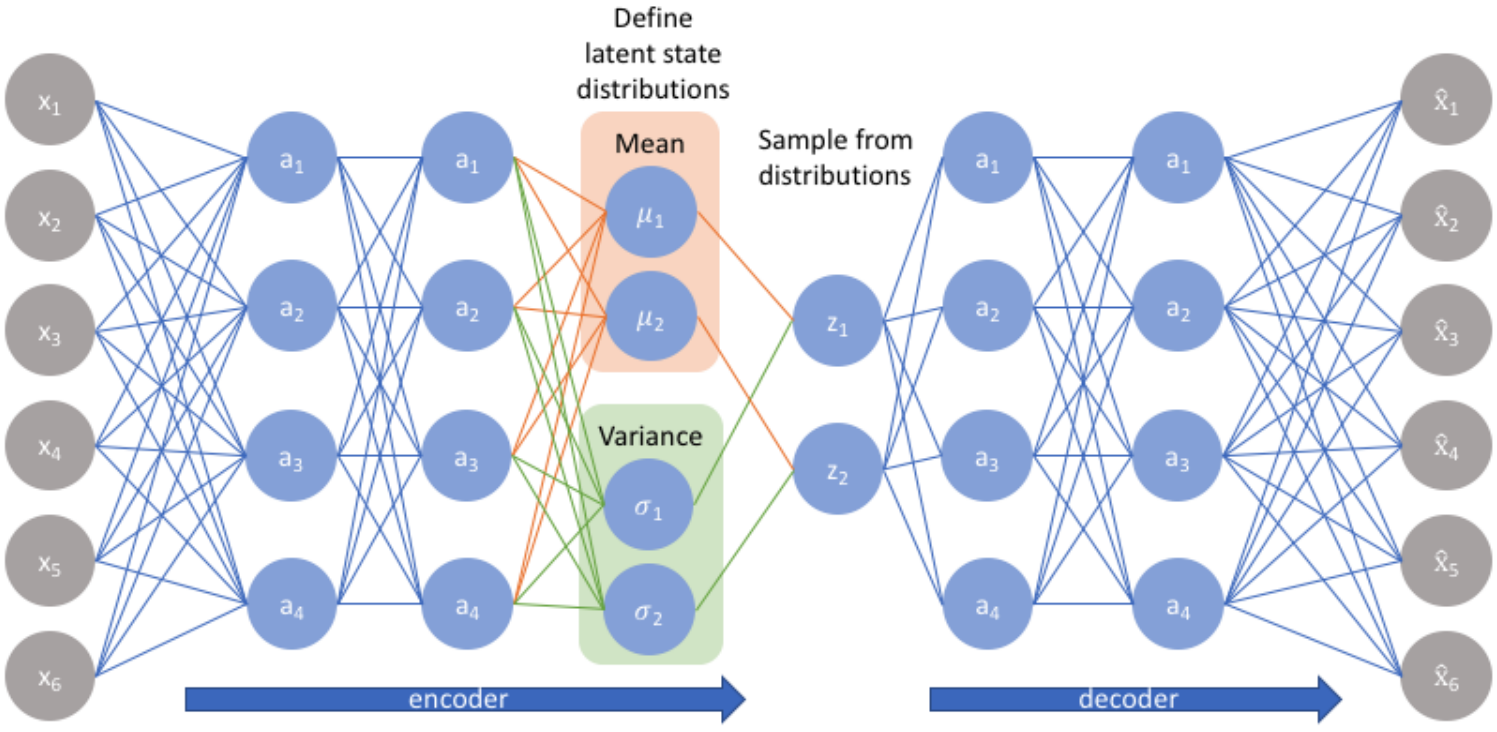


Only KL divergence



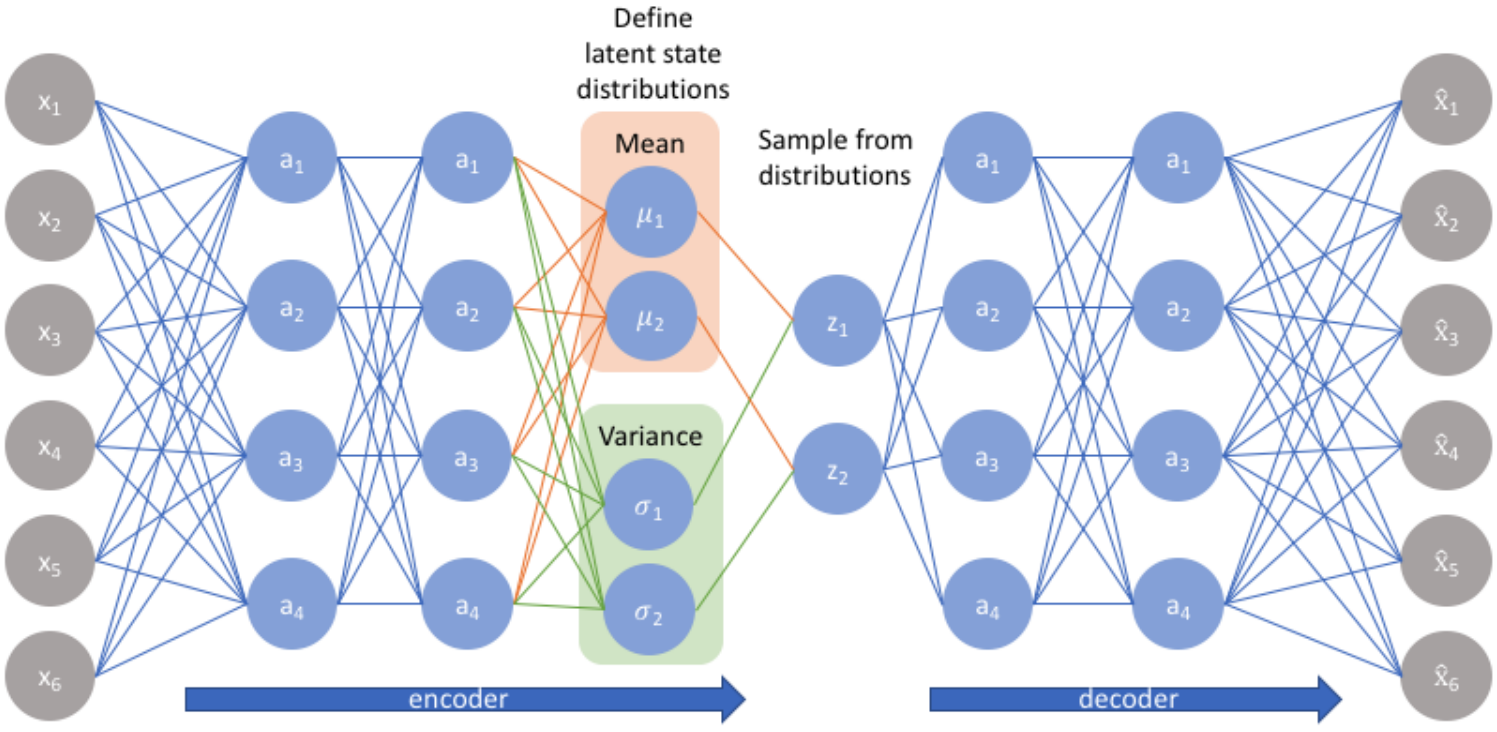
Combination





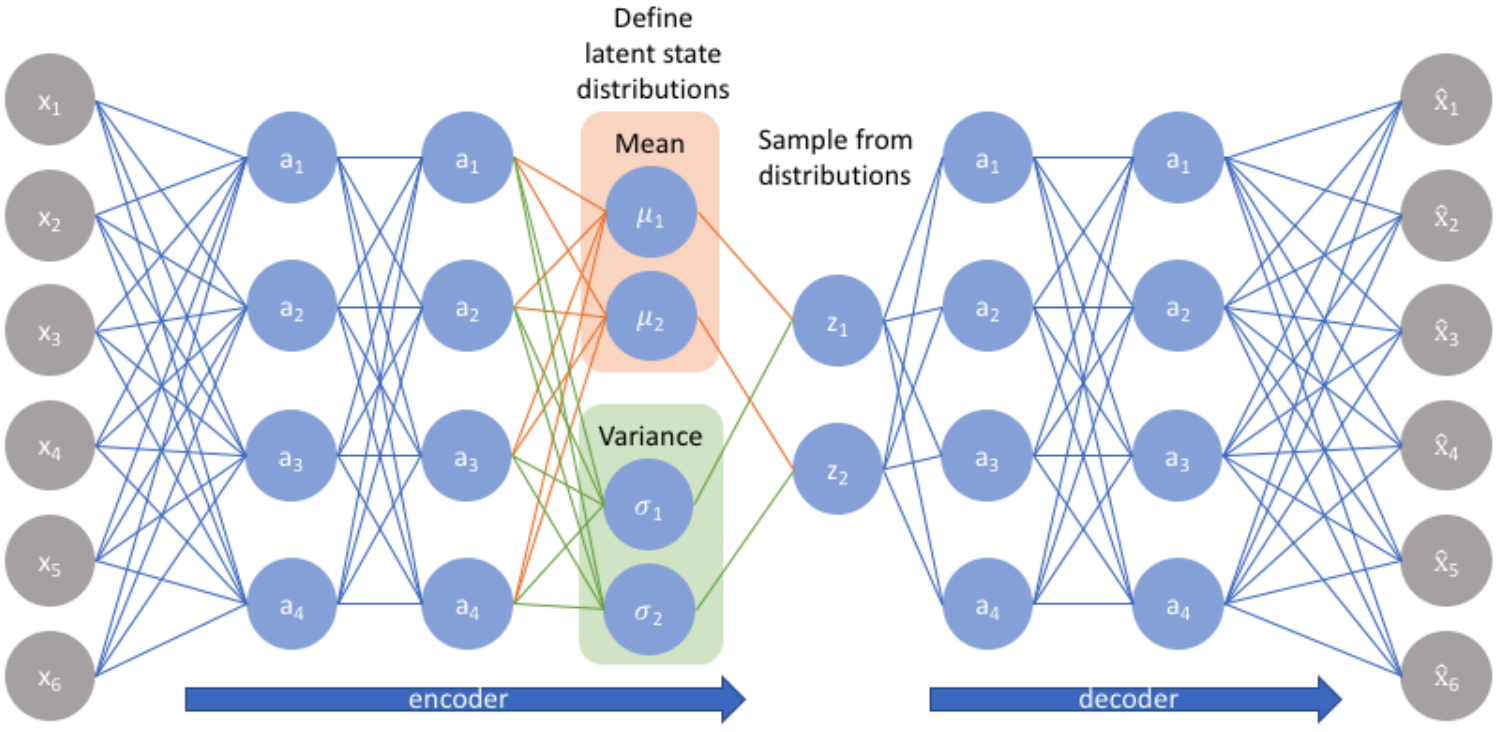
VAE advantages:

- 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.



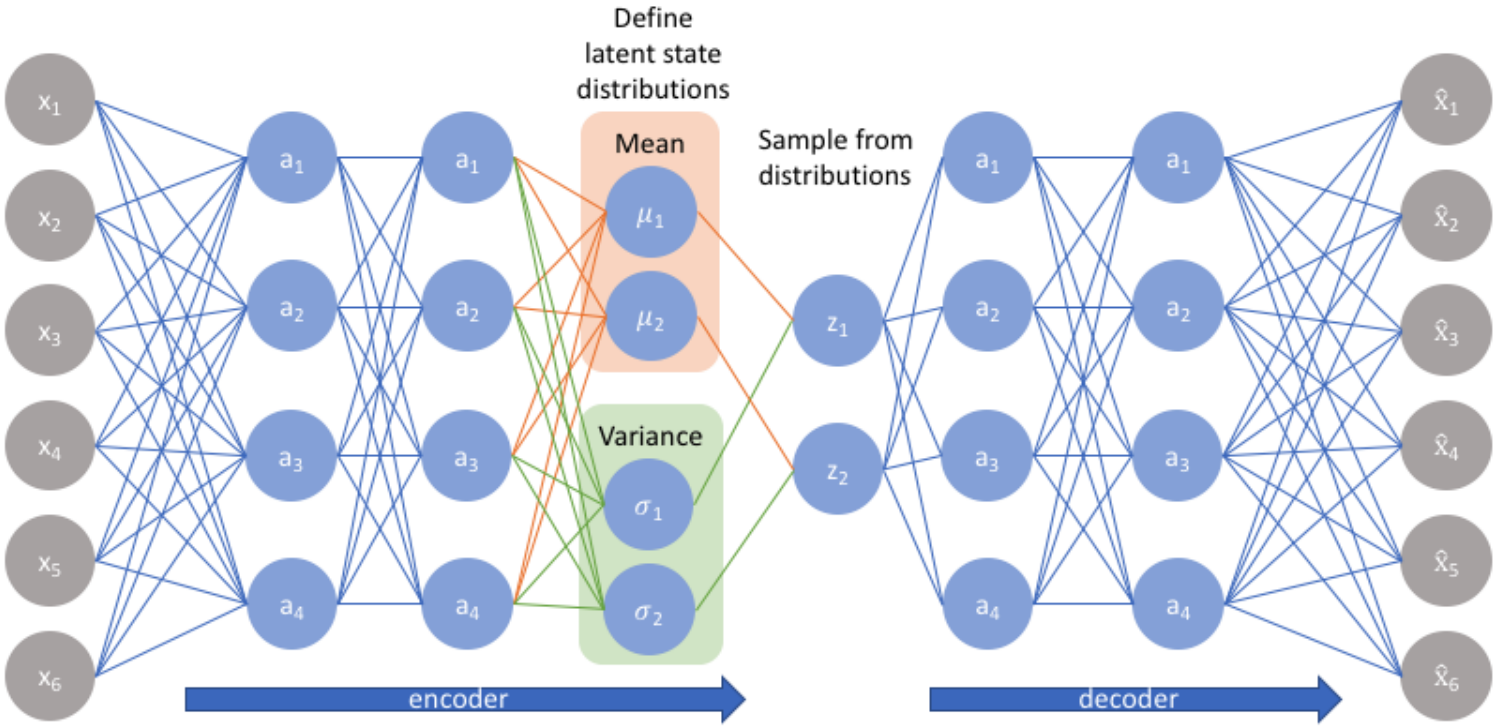
VAE advantages:

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



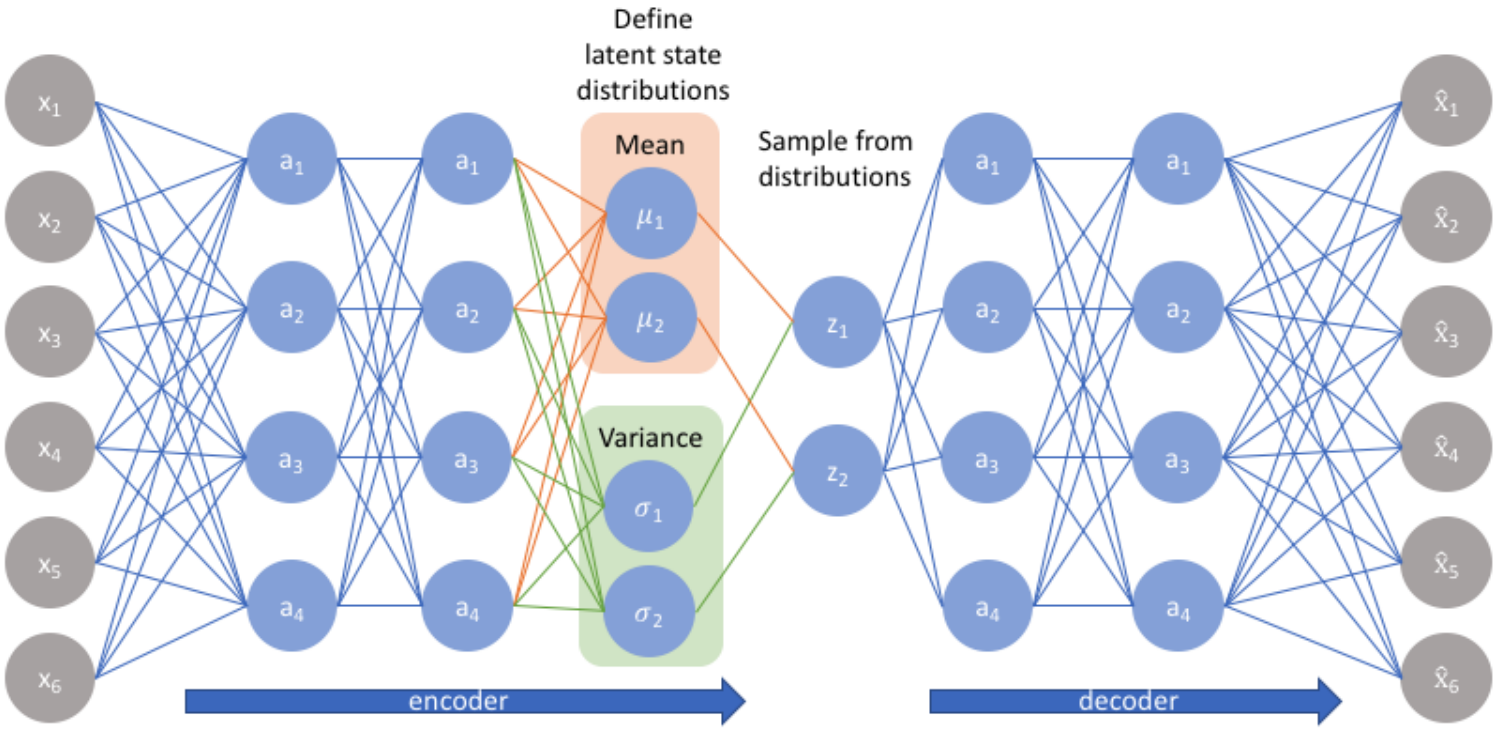
VAE advantages:

- 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.



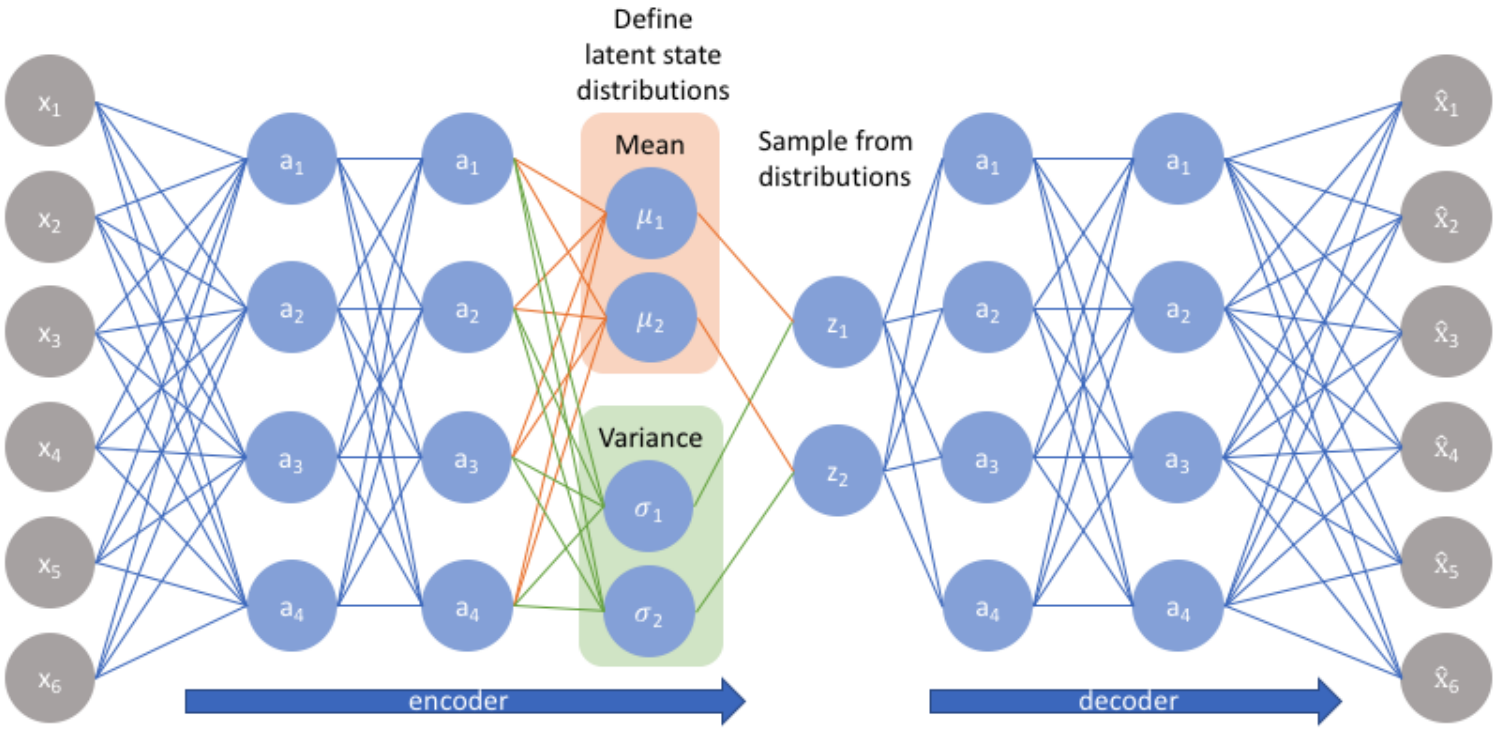
VAE advantages:

- 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.



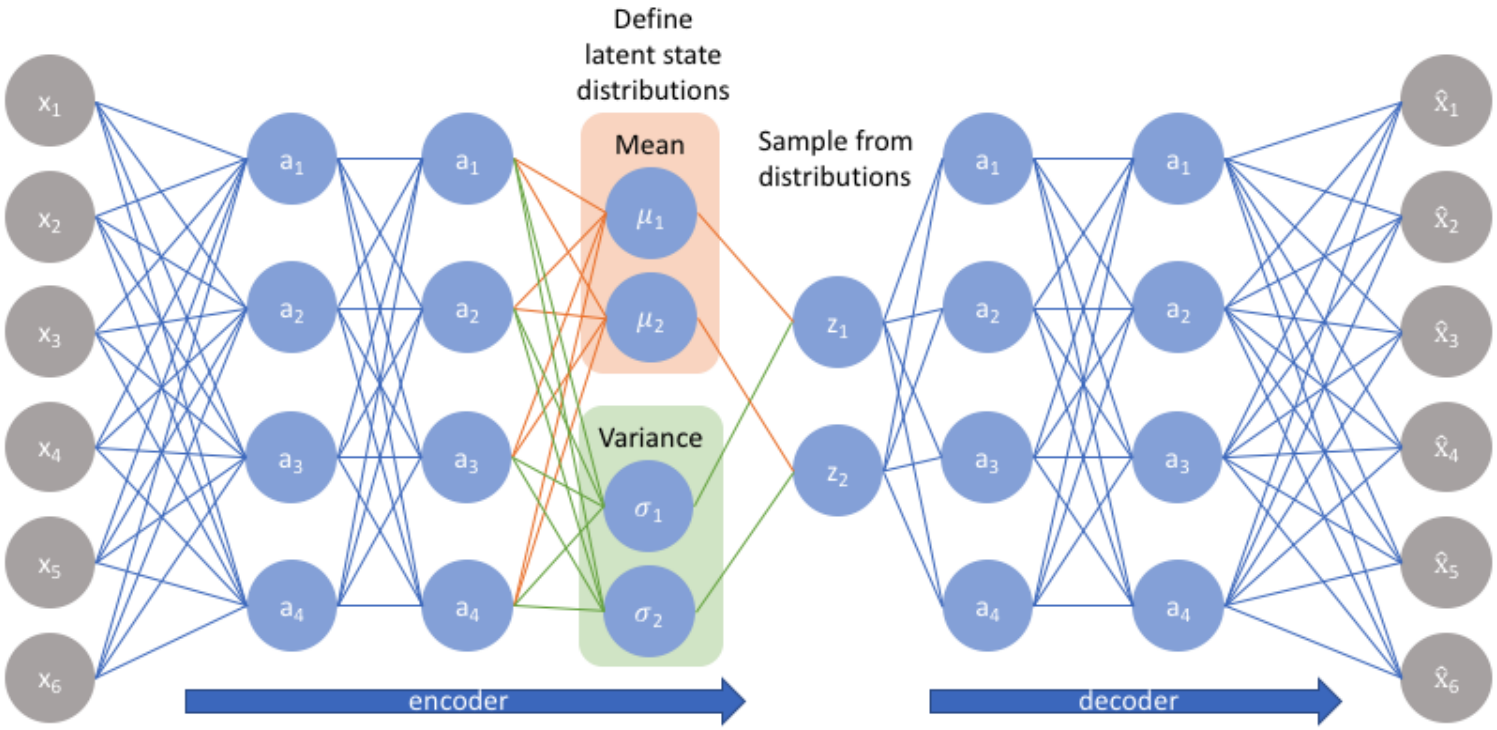
VAE disadvantages:

- 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 $\mathbf{z}$.



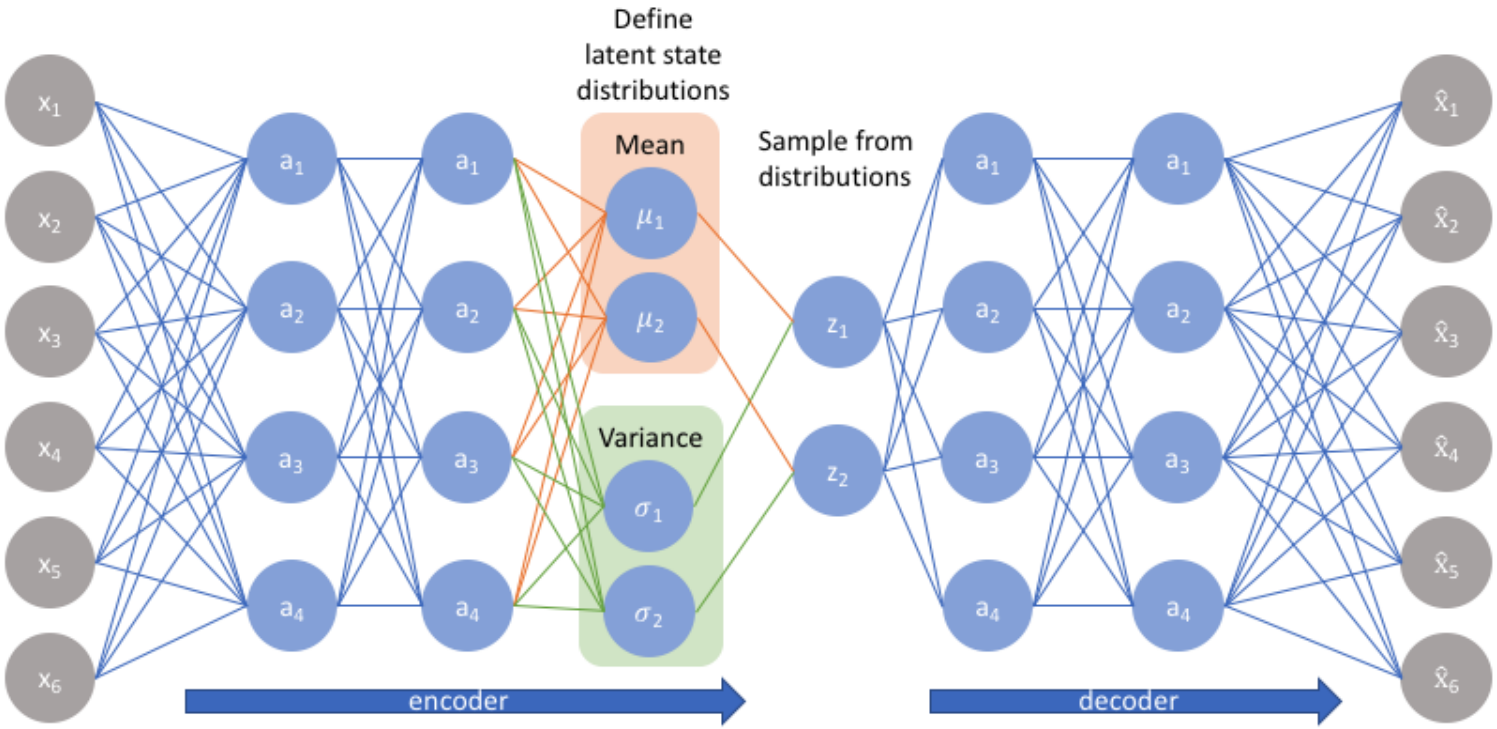
VAE disadvantages:

- 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 $\mathbf{z}$.



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 $\mathbf{z}$.

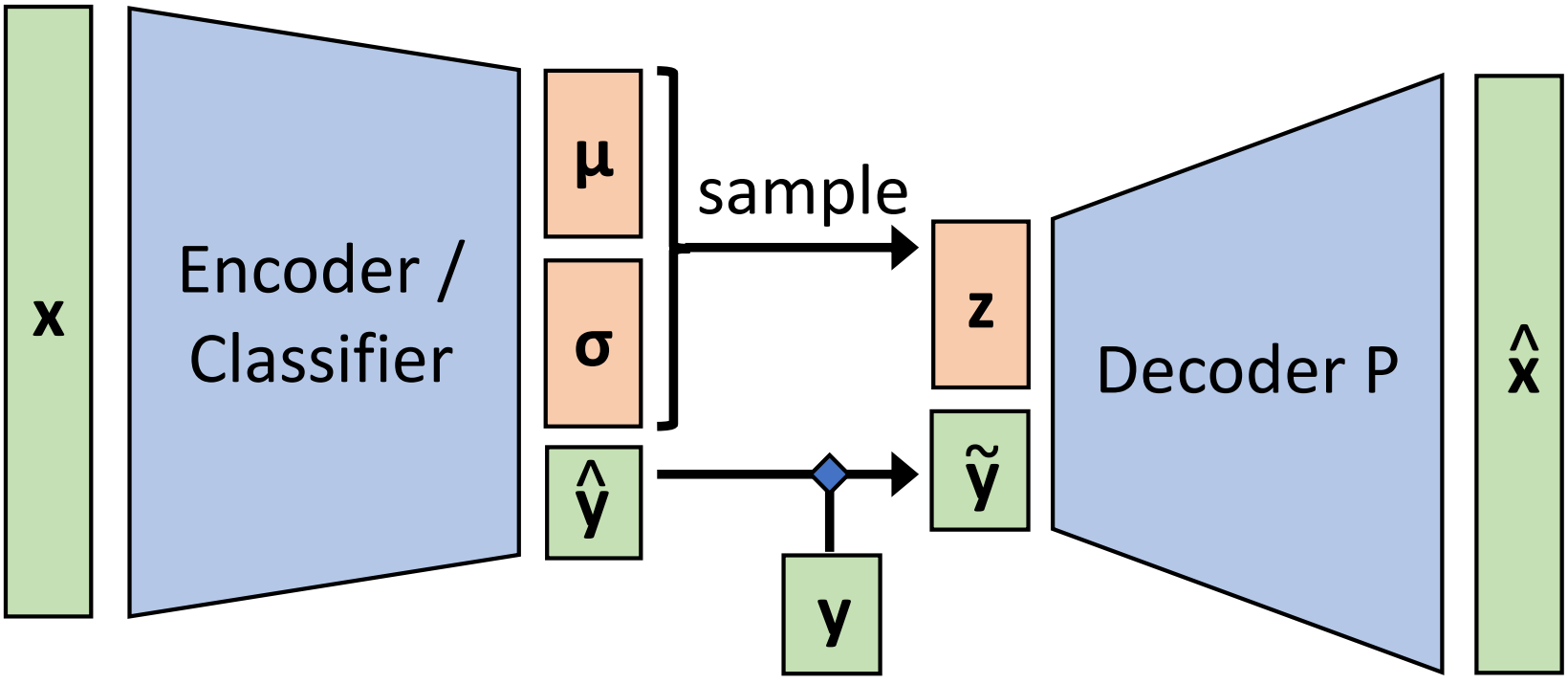


VAE disadvantages:

- 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 $\mathbf{z}$.

Classification Variational Autoencoder

Structure



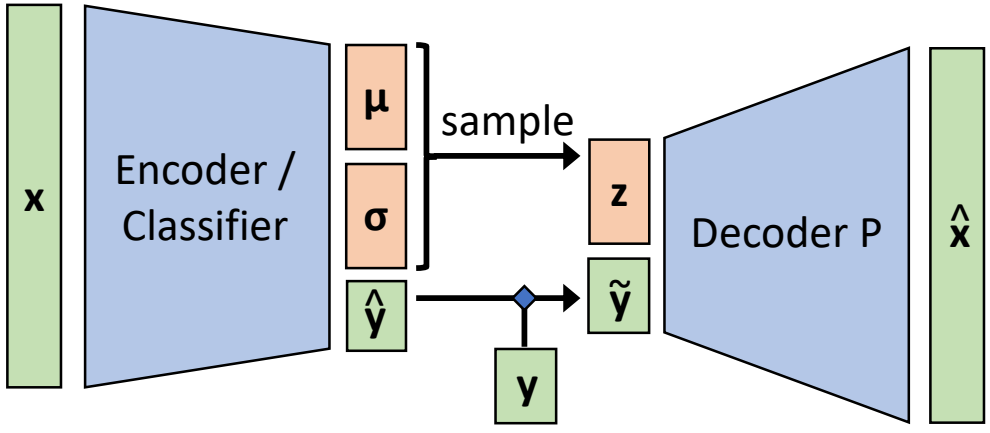
- **Supervised training**:

$$\min \left\{ \underbrace{-\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})} \log p(\mathbf{x} \mid \mathbf{z})}_{\text{Reconstruction loss}} + \underbrace{D_{\mathrm{KL}}(q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z}))}_{\text{Regularization loss}} + \underbrace{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}_{\text{Classification loss}} \right\}$$

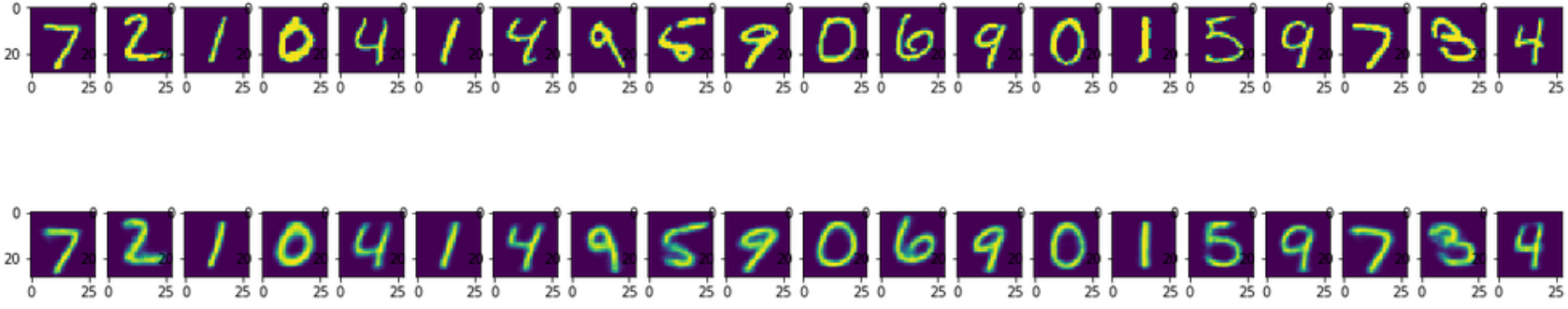
- **Unsupervised training** without classification loss.

Classification Variational Autoencoder

Input-output encoding (not optimized...)

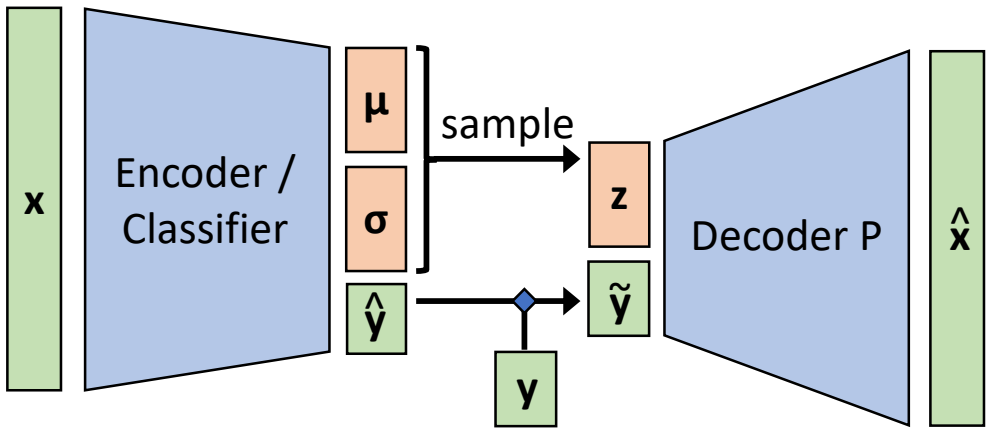


Reconstruction:

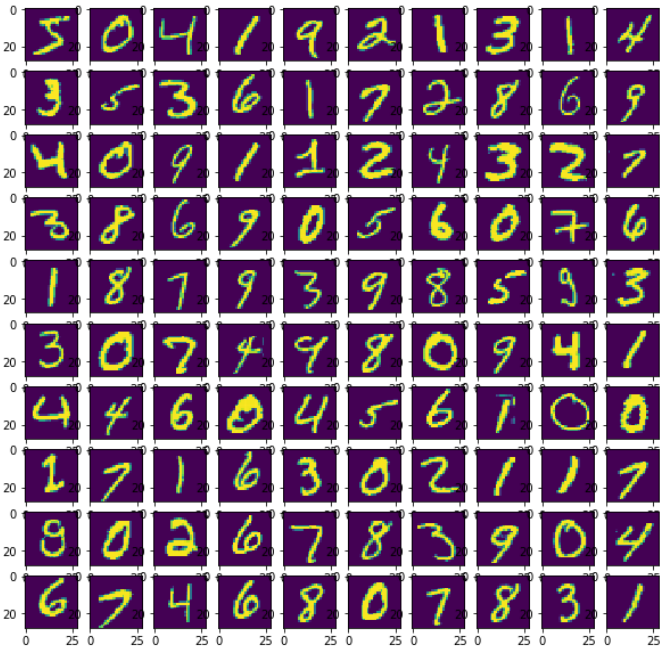


Classification Variational Autoencoder

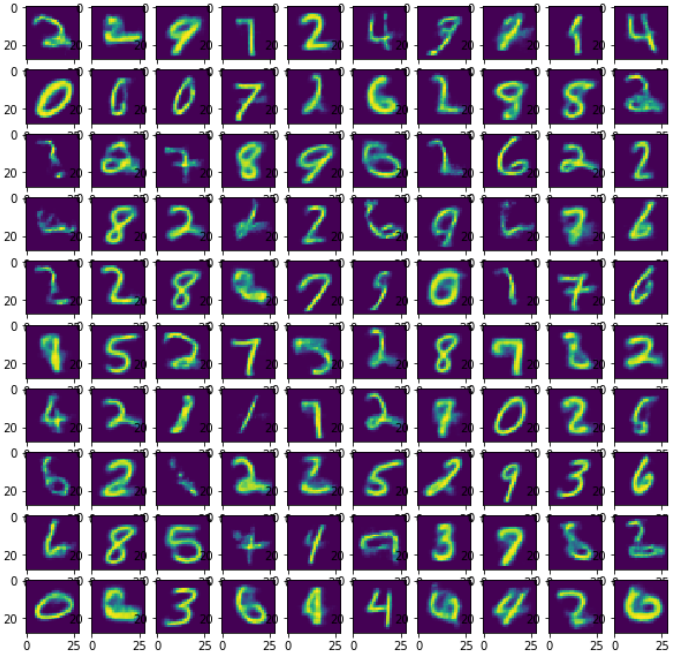
Sampling (not optimized...)



Input Data

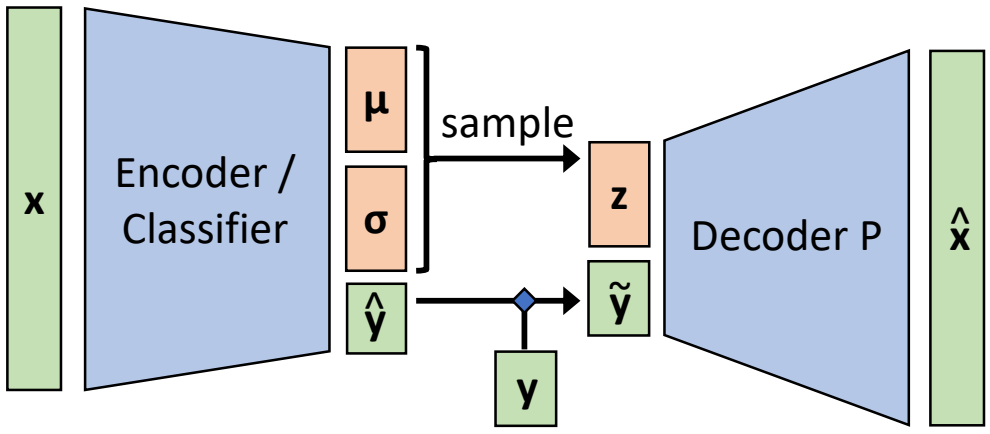


Generated Data

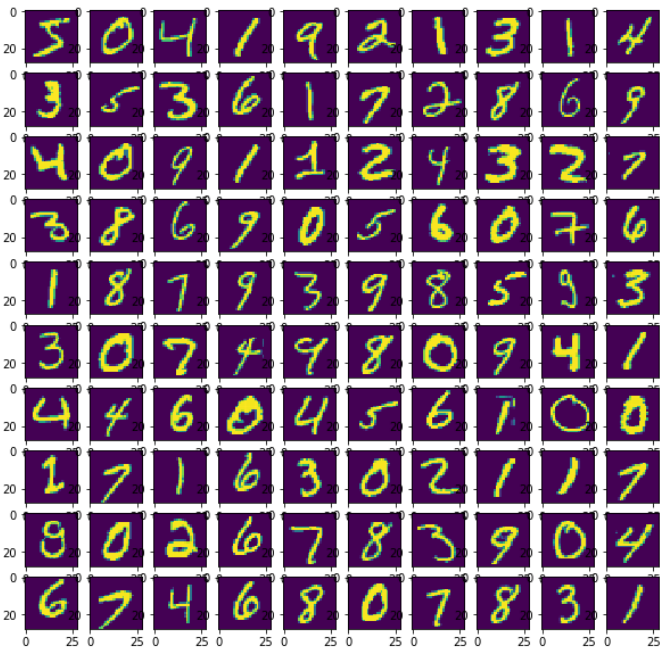


Classification Variational Autoencoder

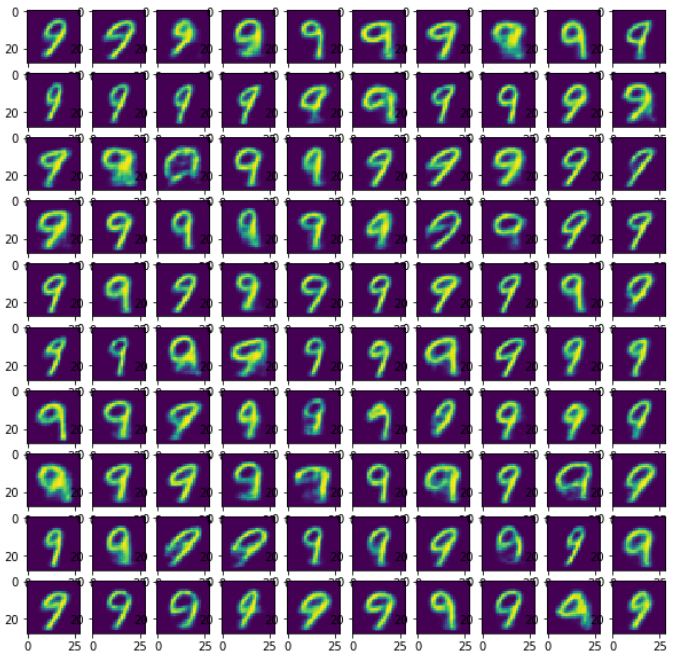
Conditional Sampling (not optimized...)



Input Data

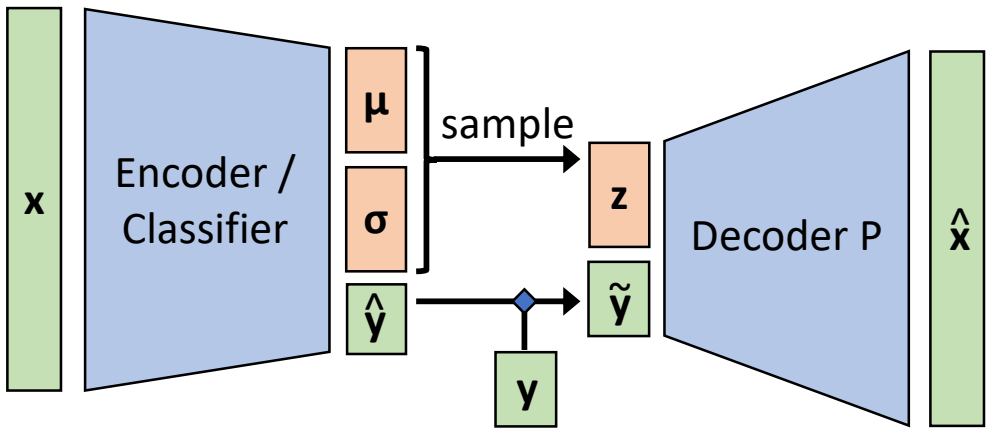


Generated Data



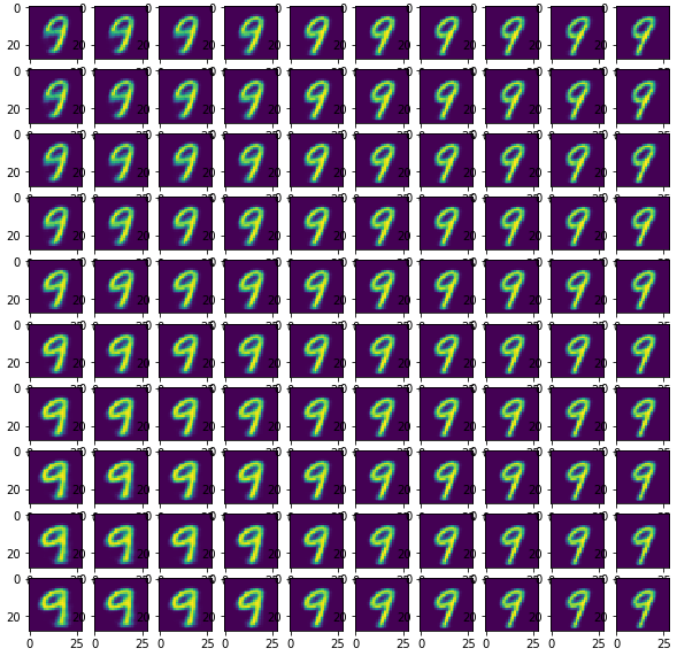
Classification Variational Autoencoder

Interpolation (not optimized...)



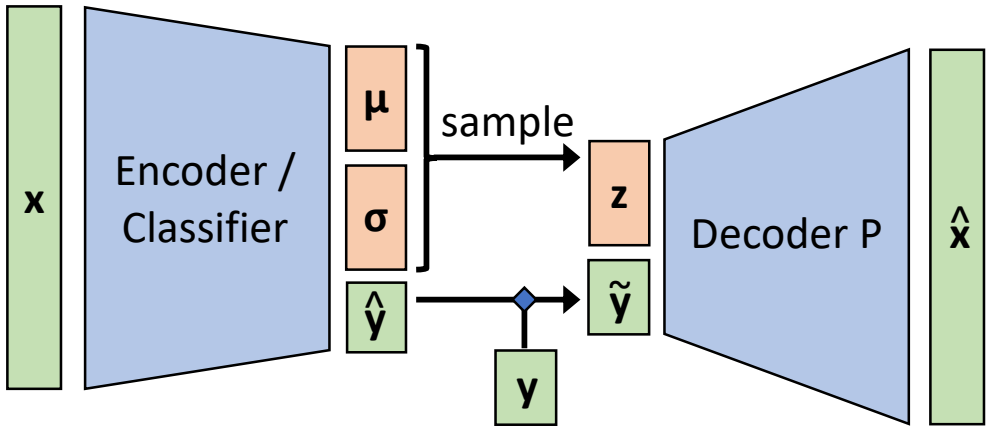
Input Data

Generated Data



Classification Variational Autoencoder

Semi-supervised learning (not optimized...)

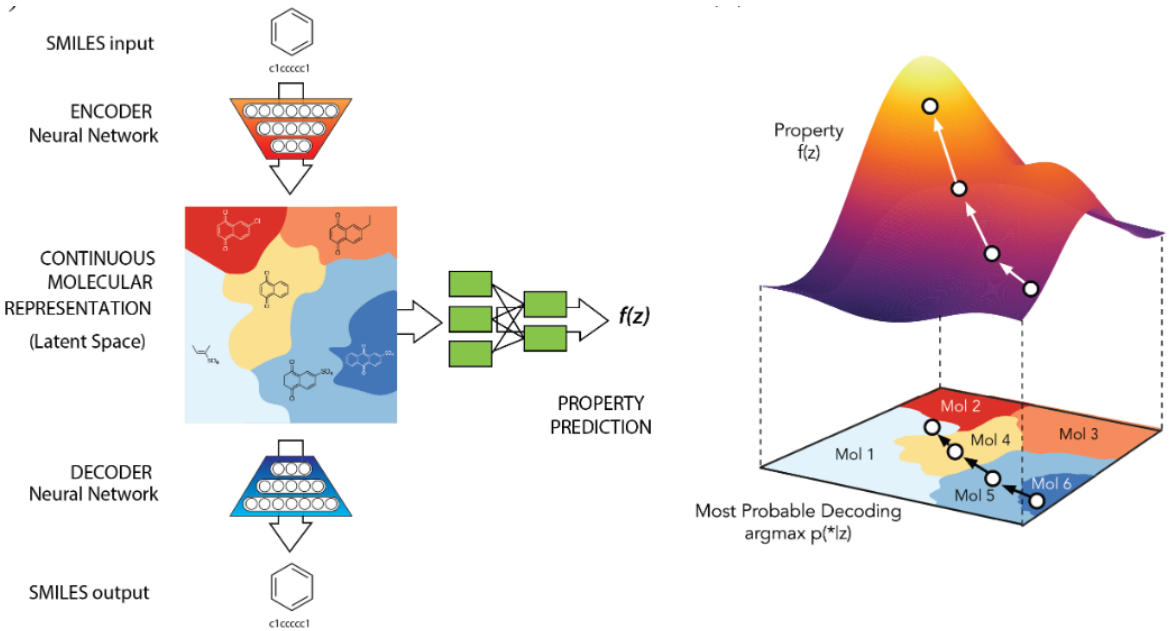


Semi-supervised learning ($N_{\text{train}}^{\text{supervised}} + N_{\text{train}}^{\text{unsupervised}} = 60,000$)

$N_{\text{train}}^{\text{supervised}}$	100	400	1000	4000	10000	40000	60000
Test error	0.2569	0.3577	0.7047	0.9291	0.9697	0.9878	0.9920

Examples

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

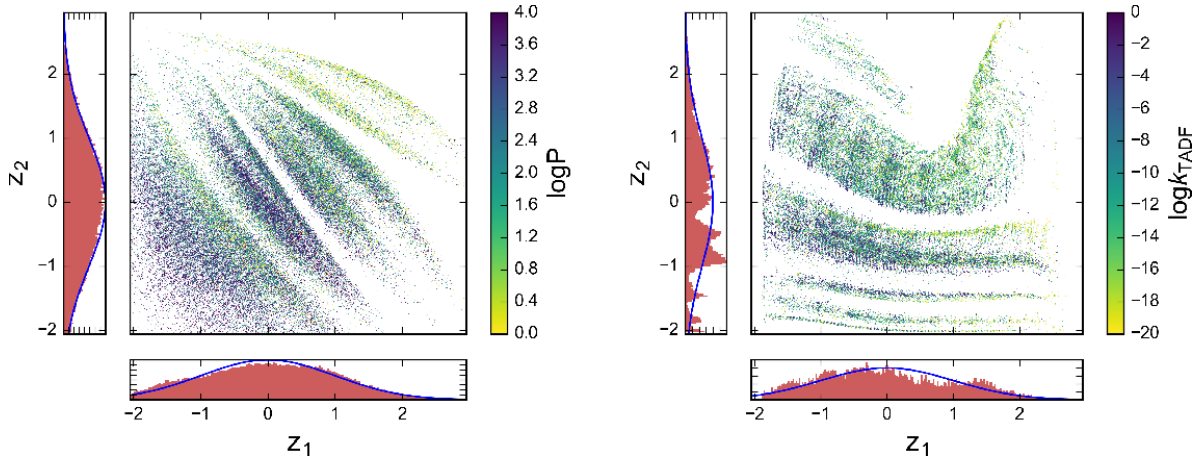


Interpolation. Starting from a discrete molecular representation, such as a SMILES string, the encoder network converts each molecule into a vector in the latent space, which is effectively a continuous molecular representation. Given a point in the latent space, the decoder network produces a corresponding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

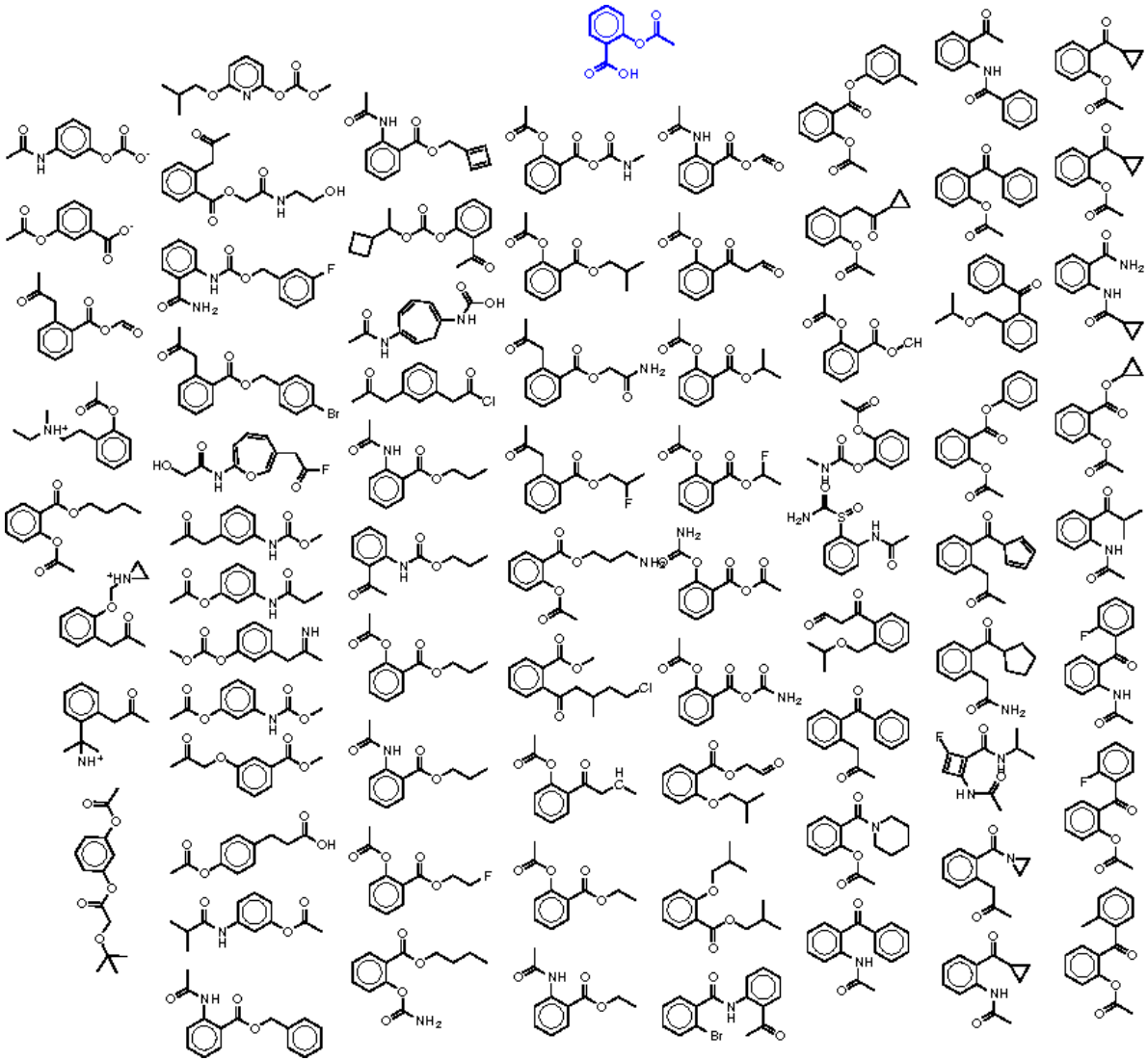
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



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

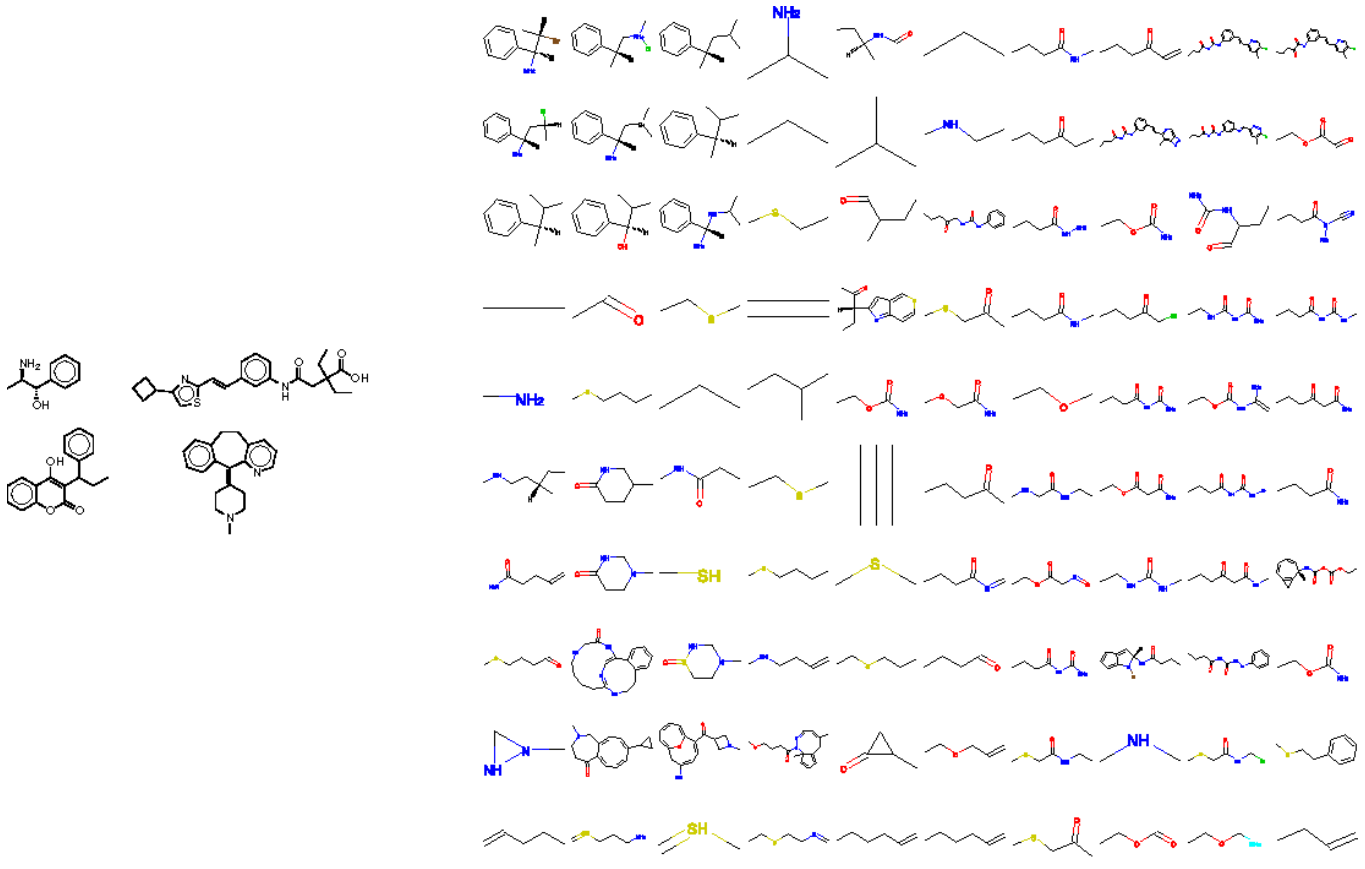
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



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

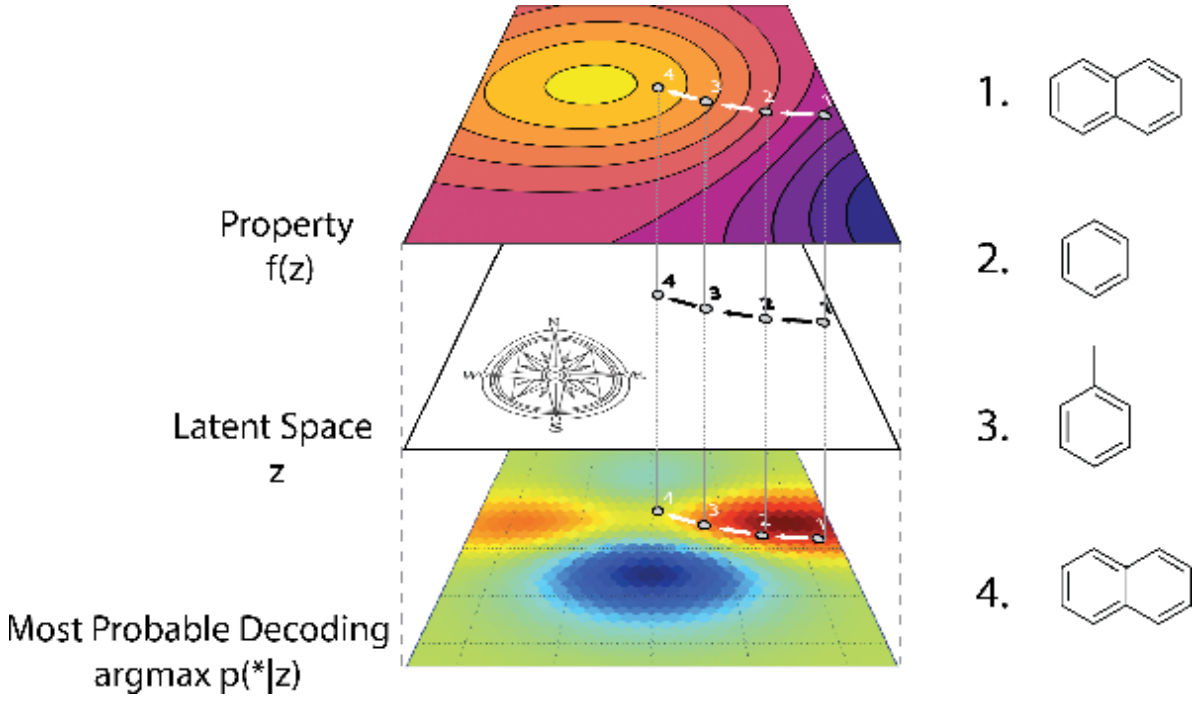
Automatic Chemical Design using Variational Autoencoders (Gómez-Bombarelli et al, 2016)



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.

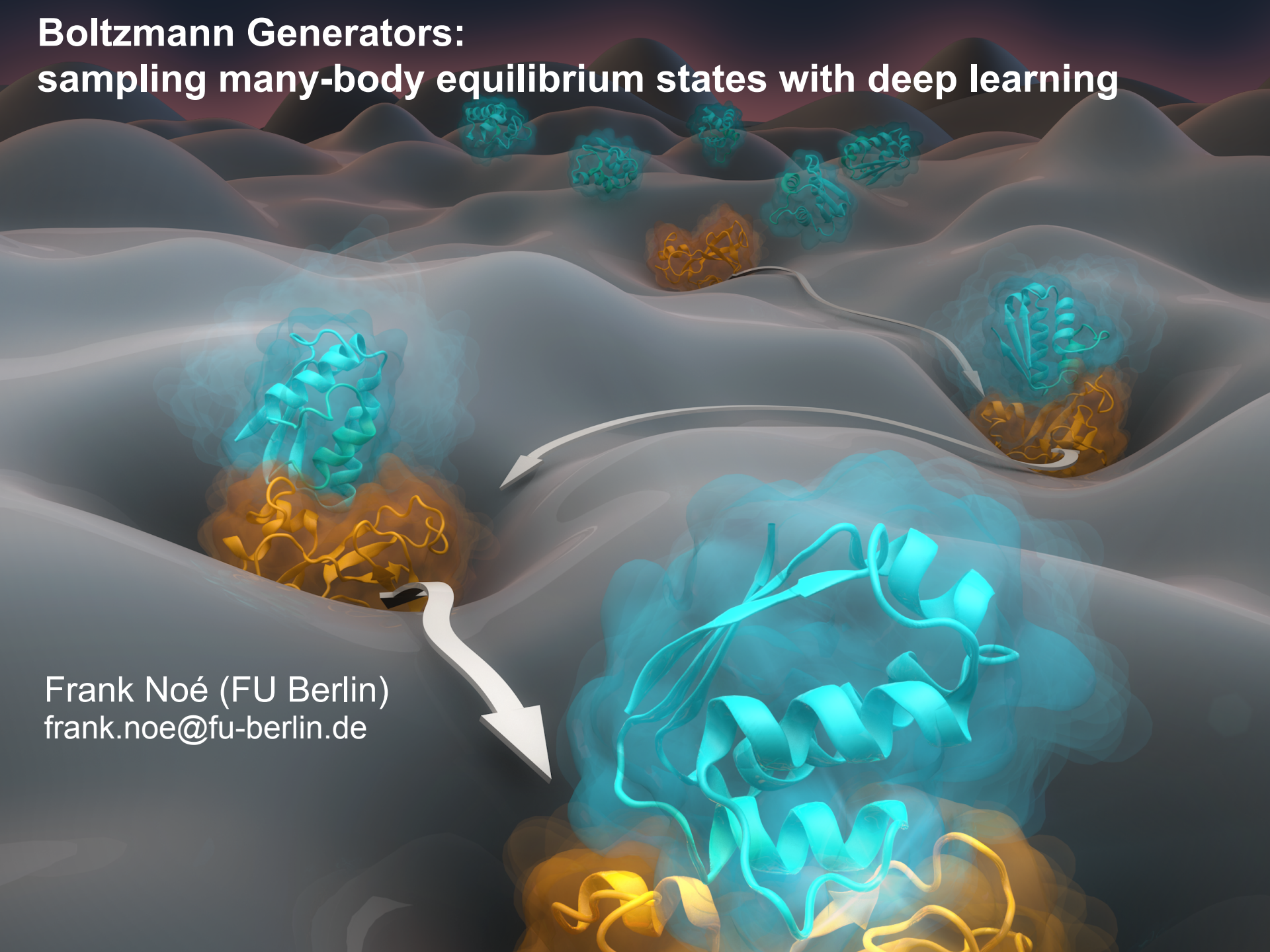
Examples

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.

Boltzmann Generators:
sampling many-body equilibrium states with deep learning



Frank Noé (FU Berlin)
frank.noe@fu-berlin.de

Thermodynamics **Sampling problem**

Boltzmann Generators
sampling equilibrium states of many-body systems with deep learning

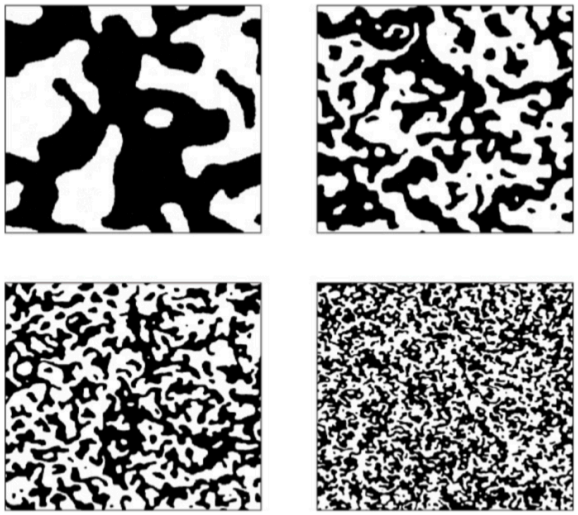
Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019)

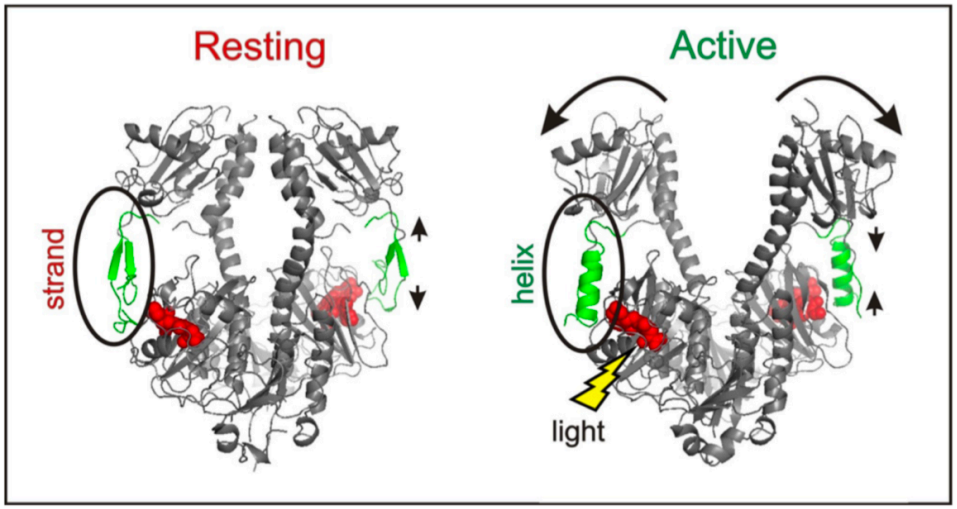

Simon Olsson


Jonas Köhler


Hao Wu

Boltzmann Generators





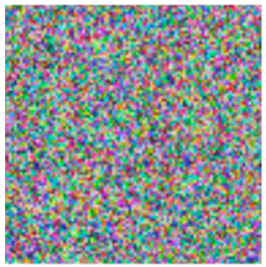
- **Input**: Reduced Potential Energy $u(\mathbf{x})$ in coordinates $\mathbf{x} \in \mathbb{R}^n$, e.g. $u(\mathbf{x}) = U(\mathbf{x})/k_B T$ (canonical ensemble).
- **Aim**: Generate *independent* Samples from Equilibrium Distribution.

$$\mu(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$$

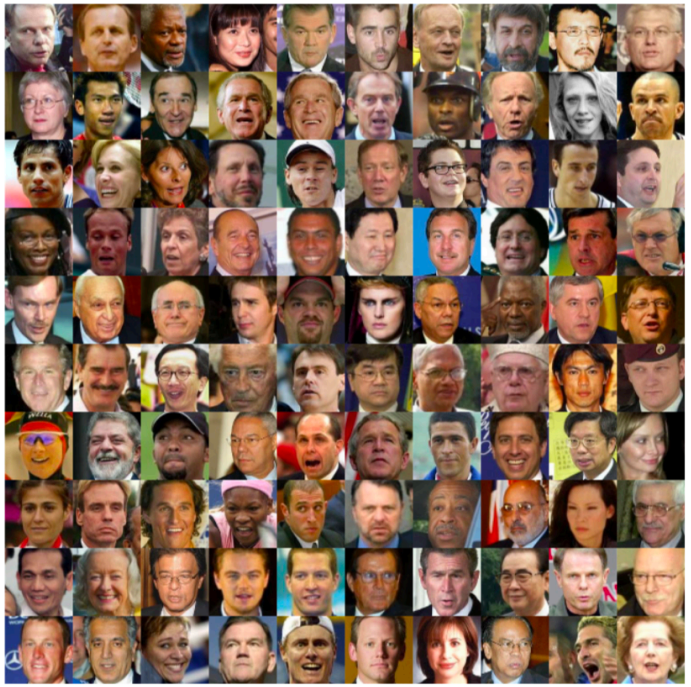
- **Problem**:
  - Direct MC (proposal for all n degrees of freedom + rejection or reweighting) not available.
  - Standard approach: MD/MCMC with local moves $\rightarrow$ sampling problem.

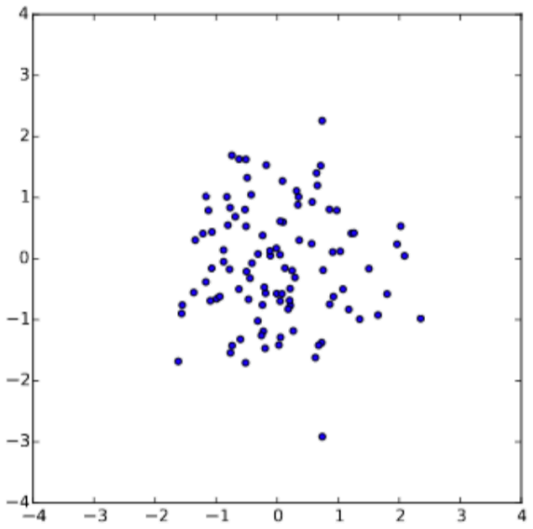
Boltzmann Generators

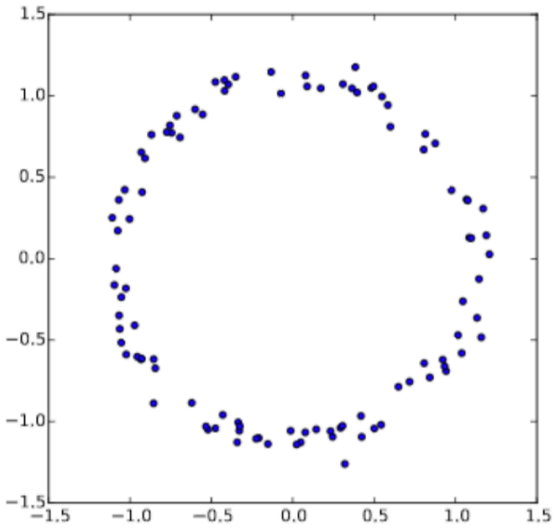
Noise ~ N(0,1)



Generative Model

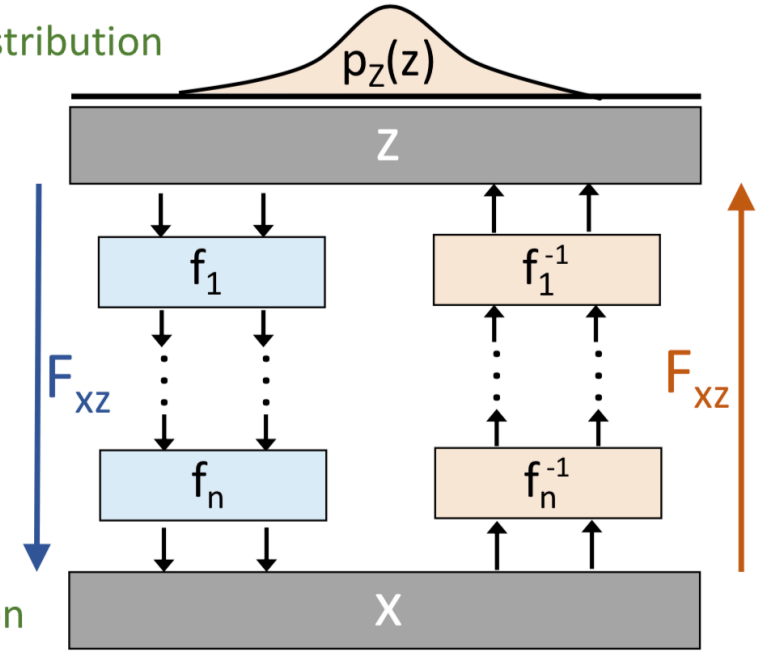




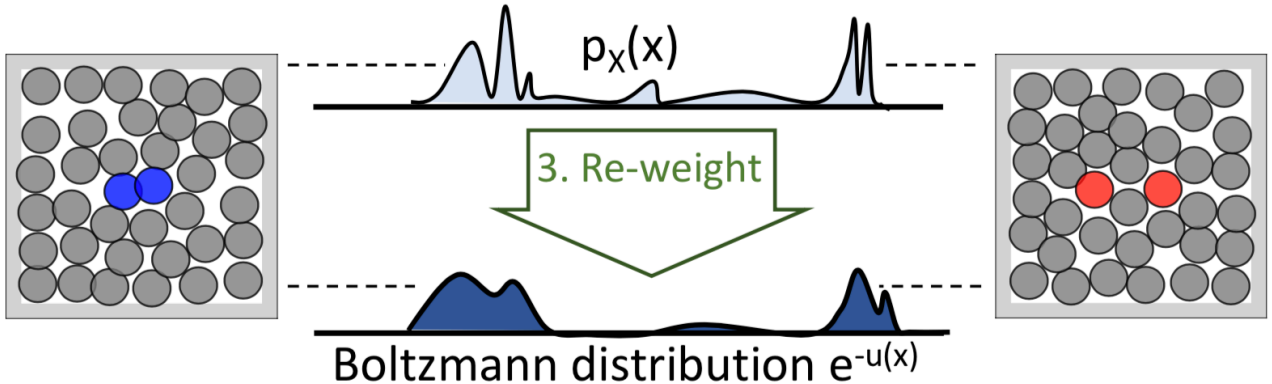


Boltzmann Generators

1. Sample Gaussian distribution



2. Generate distribution



Related work:

Tabak & Vanden-Eijnden:
Density estimation with flows

Dinh, Bengio, Rezende:
Invertible networks

Rezende & Mohamed:
Normalizing Flows

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators

- Invertible transformation:

$$\mathbf{z} = F_{xz}(\mathbf{x};\theta)$$
$$\mathbf{x} = F_{zx}(\mathbf{z};\theta).$$

- Jacobians:

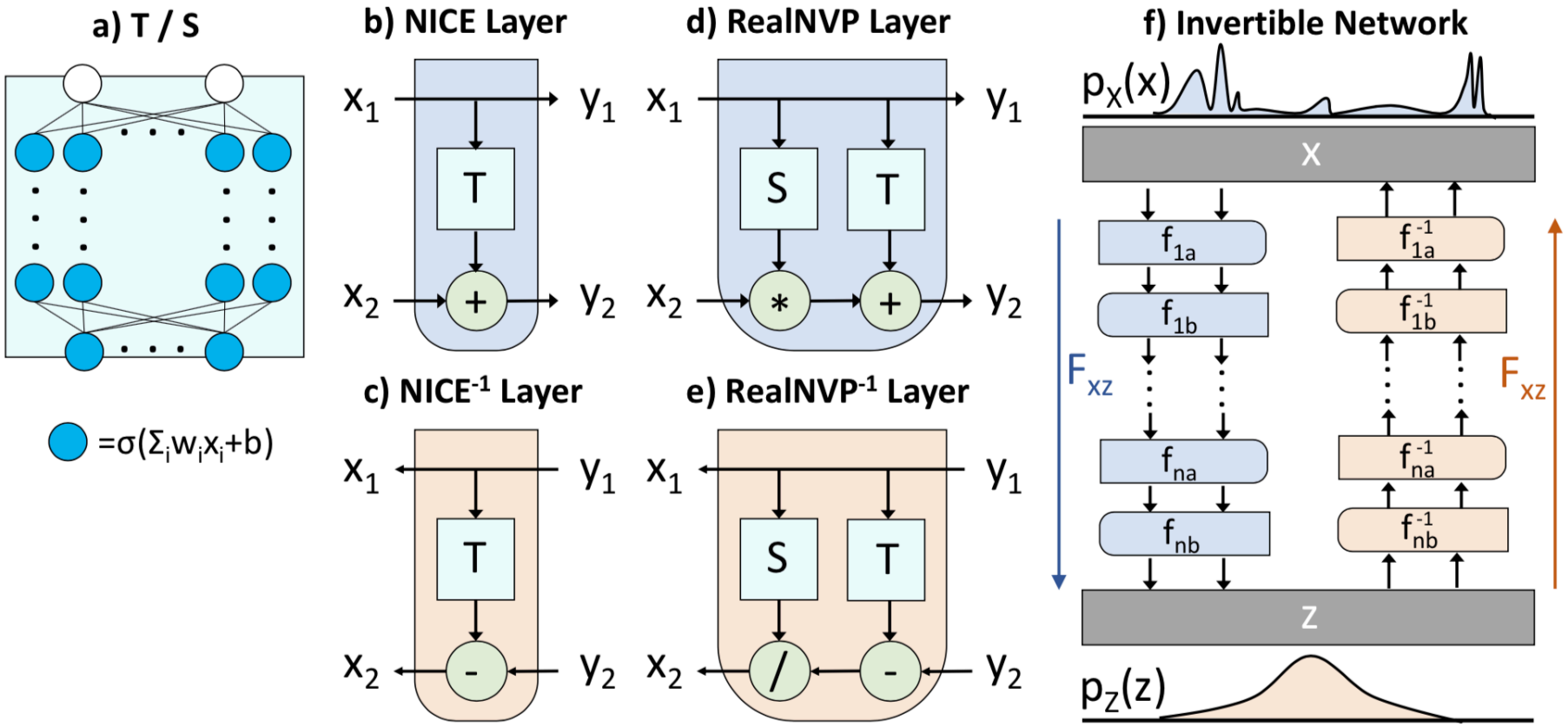
$$\mathbf{J}_{zx}(\mathbf{z};\theta) = \left[\frac{\partial F_{zx}(\mathbf{z};\theta)}{\partial z_1},\ldots,\frac{\partial F_{zx}(\mathbf{z};\theta)}{\partial z_n}\right]$$
$$\mathbf{J}_{xz}(\mathbf{x};\theta) = \left[\frac{dF_{xz}(\mathbf{x};\theta)}{dx_1},\ldots,\frac{dF_{xz}(\mathbf{x};\theta)}{dx_n}\right]$$

- Transformation of random variables:

$$p_X(\mathbf{x}) = p_Z(T_{xz}(\mathbf{x}))\,|\det \mathbf{J}_{xz}(\mathbf{x})|$$
$$p_Z(\mathbf{z}) = p_X(T_{zx}(\mathbf{z}))\,|\det \mathbf{J}_{zx}(\mathbf{z})|$$

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators



- **NICE**: Dinh, Krueger, Y. Bengio, ICLR 2015
- **RealNVP**: Dinh, Sohl-Dickstein, S. Bengio, ICLR 2017
- More advanced flow operations (e.g., normalizing flows)

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators

- **Distributions**:

$$\begin{aligned} \text{Prior} \quad & \mu_Z(\mathbf{z}) \xrightarrow{F_{zx}} p_X(\mathbf{x}) \quad \text{generated} \\ \text{Boltzmann} \quad & \mu_X(\mathbf{x}) \xrightarrow{F_{xz}} p_Z(\mathbf{z}) \quad \text{generated} \end{aligned}$$

- **Aim**: sample configurations **x** from **Boltzmann distribution**

$$\mu_X(\mathbf{x}) = Z_X^{-1} \mathrm{e}^{-u(\mathbf{x})}$$

- **Prior distribution**: Sample input in **z** from isotropic Gaussian:

$$q_Z(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) = Z_Z^{-1} \mathrm{e}^{-\frac{1}{2}\|\mathbf{z}\|^2/\sigma^2},$$

Prior energy:

$$u_Z(\mathbf{z}) = -\log q_Z(\mathbf{z}) = \frac{1}{2\sigma^2} \|\mathbf{z}\|^2 + \text{const}$$

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators

- Loss function:

$$J = \underbrace{w_{ML} J_{ML}}_{\text{max likelihood}} + \underbrace{w_{KL} J_{KL}}_{\text{Kullback-Leibler}} + \underbrace{w_{RC} J_{RC}}_{\text{reaction coordinate}}.$$

- KL divergence between generated $p_X(\mathbf{x})$ and Boltzmann distribution:

$$\begin{aligned}\mathrm{KL}_\theta\left[\mu_Z \parallel p_Z\right] &= \int \mu_Z(\mathbf{z})\left[\log \mu_Z(\mathbf{z}) - \log p_Z(\mathbf{z};\theta)\right] \mathrm{d}\mathbf{z}, \\ &= \underbrace{-H_Z + \log Z_X}_{\text{const}} + \underbrace{\mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\theta)) - \log\left|\mathbf{J}_{zx}(\mathbf{z};\theta)\right|\right]}_{\text{free energy}}\end{aligned}$$

- KL loss:

$$J_{KL} = \mathbb{E}_{\mathbf{z}\sim\mu_Z(\mathbf{z})}\left[u(F_{zx}(\mathbf{z};\theta)) - \log\left|\mathbf{J}_{zx}(\mathbf{z};\theta)\right|\right]. \qquad (5)$$

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators

- Loss function:

$$J = \underbrace{w_{ML} J_{ML}}_{\text{max likelihood}} + \underbrace{w_{KL} J_{KL}}_{\text{Kullback-Leibler}} + \underbrace{w_{RC} J_{RC}}_{\text{reaction coordinate}}.$$

- KL divergence between generated distribution $q_Z(\mathbf{z})$ and Gaussian:

$$\begin{aligned}\mathrm{KL}_\theta(\mu_X \parallel p_X) &= -H_X - \int \mu_X(\mathbf{x}) \log p_X(\mathbf{x};\theta) \mathrm{d}\mathbf{x} \\ &= \underbrace{-H_X + \log Z_Z}_{\text{const}} + \mathbb{E}_{\mathbf{x}\sim\mu(\mathbf{x})}\left[\frac{1}{\sigma^2}\left\|F_{xz}(\mathbf{x};\theta)\right\|^2 - \log\left|\mathbf{J}_{xz}(\mathbf{x};\theta)\right|\right]\end{aligned}$$

  Problem: sampling $\mathbf{x} \sim \mu(\mathbf{x})$ is difficult and our goal!

- **Maximum Likelihood** loss for initial data distribution $\rho(\mathbf{x})$:

$$\begin{aligned}J_{ML} &= -\mathbb{E}_{\mathbf{x}\sim\rho(\mathbf{x})}\left[\log p_X(\mathbf{x};\theta)\right] \\ &= \mathbb{E}_{\mathbf{x}\sim\rho(\mathbf{x})}\left[\frac{1}{\sigma^2}\left\|F_{xz}(\mathbf{x};\theta)\right\|^2 - \log\left|\mathbf{J}_{xz}(\mathbf{x};\theta)\right|\right]\end{aligned}$$

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators

- Loss function:

$$J = \underbrace{w_{ML} J_{ML}}_{\text{max likelihood}} + \underbrace{w_{KL} J_{KL}}_{\text{Kullback-Leibler}} + \underbrace{w_{RC} J_{RC}}_{\text{reaction coordinate}}.$$

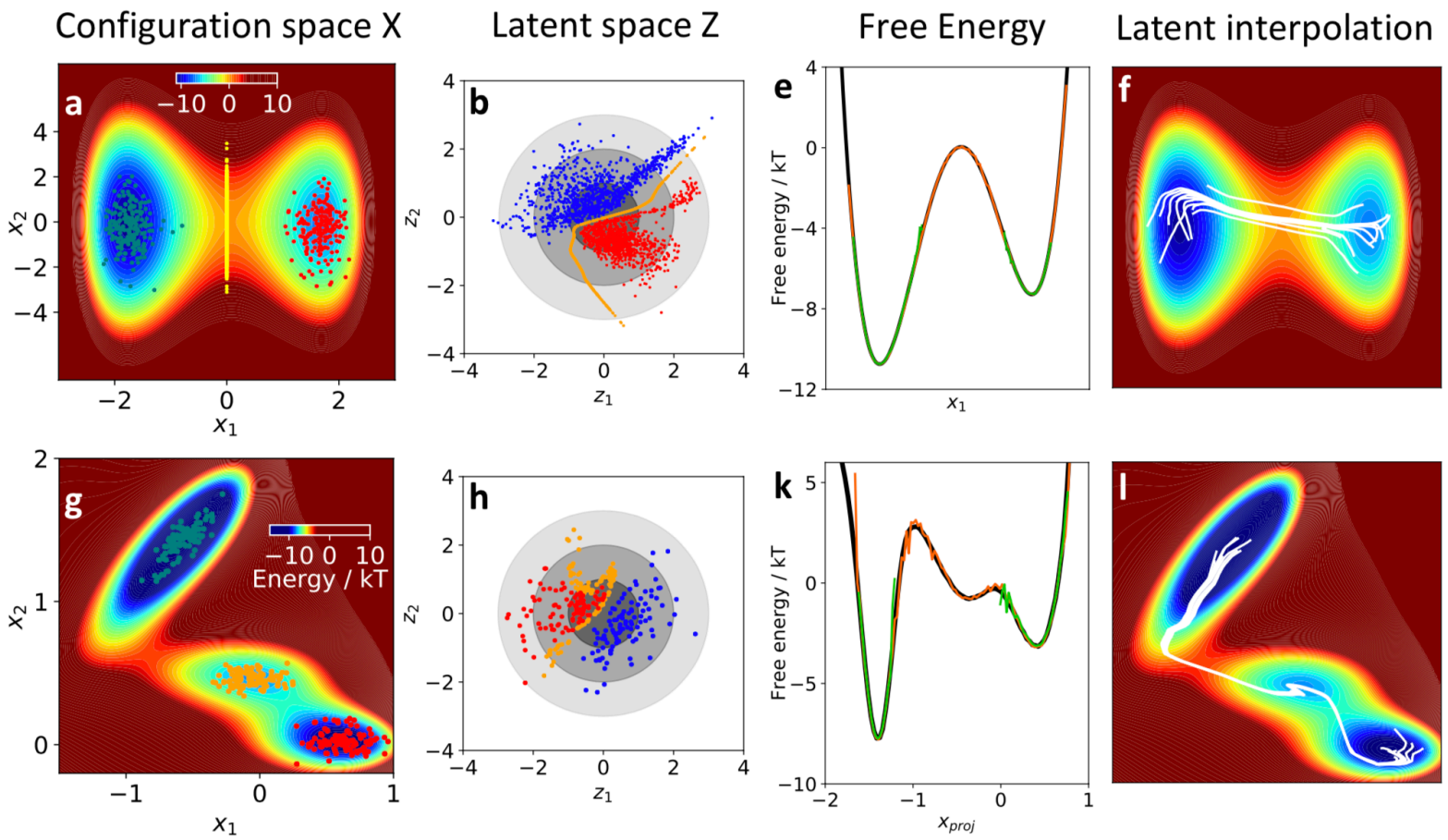
- Reaction coordinate loss:

$$\begin{aligned} J_{RC} &= \int p(R(\mathbf{x})) \log p(R(\mathbf{x})) \, \mathrm{d}R(\mathbf{x}) \\ &= \mathbb{E}_{\mathbf{x} \sim p_X(\mathbf{x})} \log p(R(\mathbf{x})). \end{aligned}$$

- Implementation:
  - Reaction coordinate function R is user input
  - Min and max bounds are given
  - $p(R(\mathbf{x}))$ is computed as a batchwise kernel density estimate.

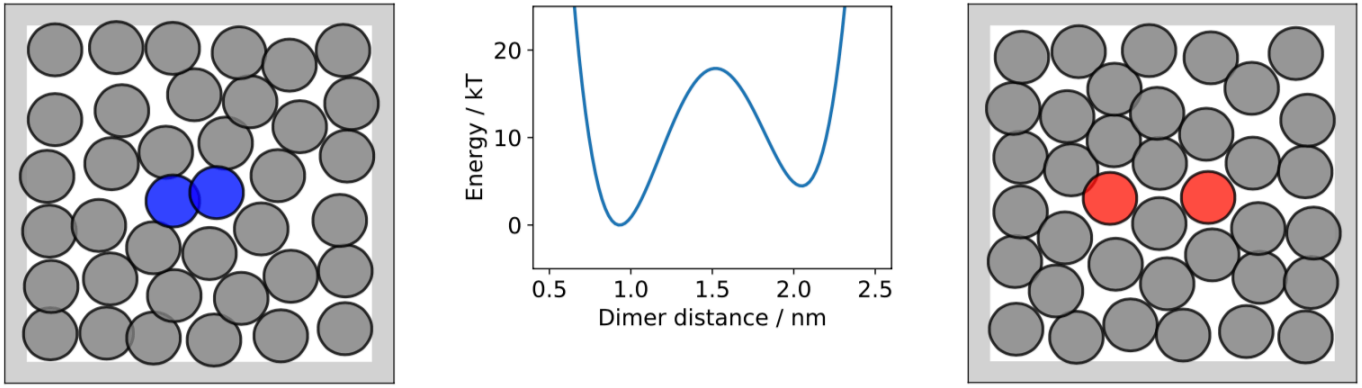
Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: Double well and Mueller potential

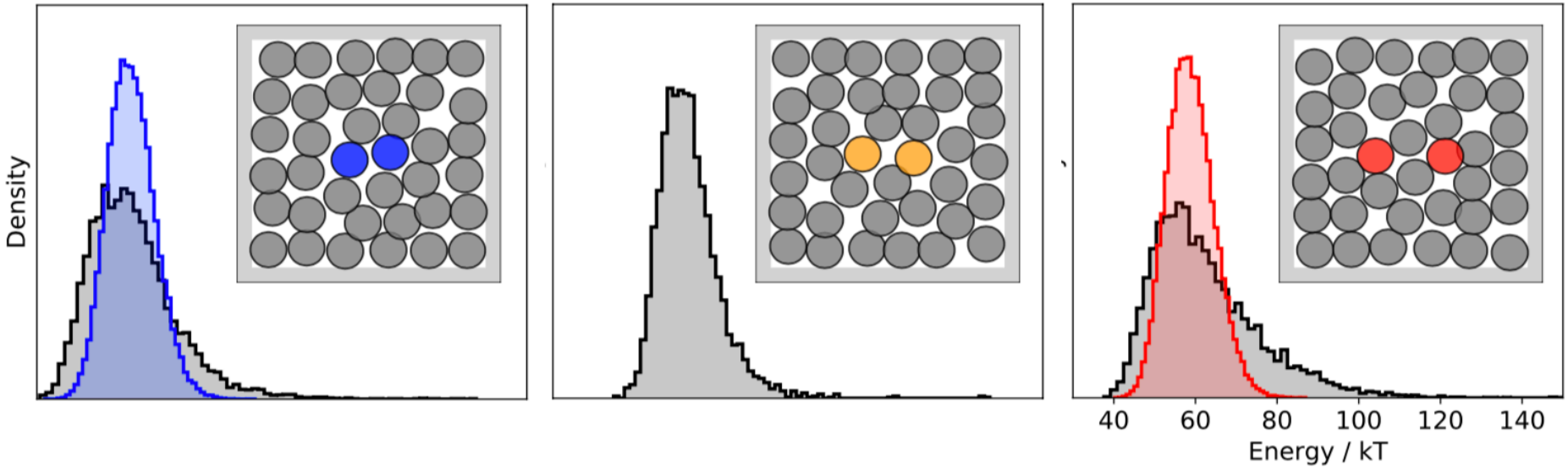


Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: Bistable dimer in dense colloid solvent

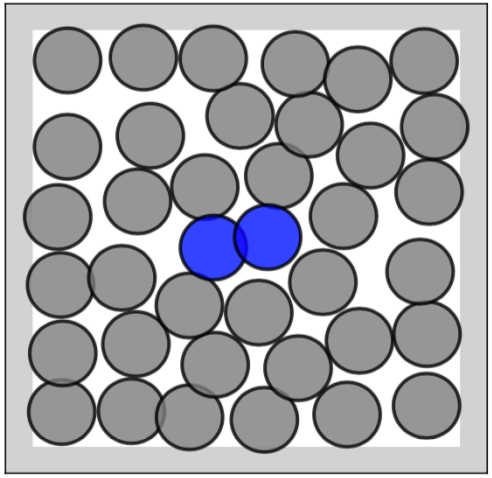


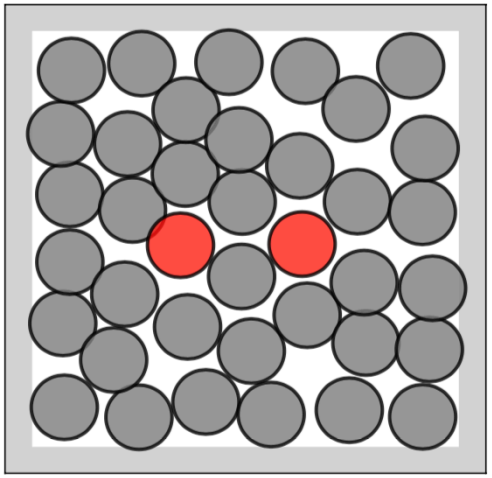
Boltzmann-Generated samples:

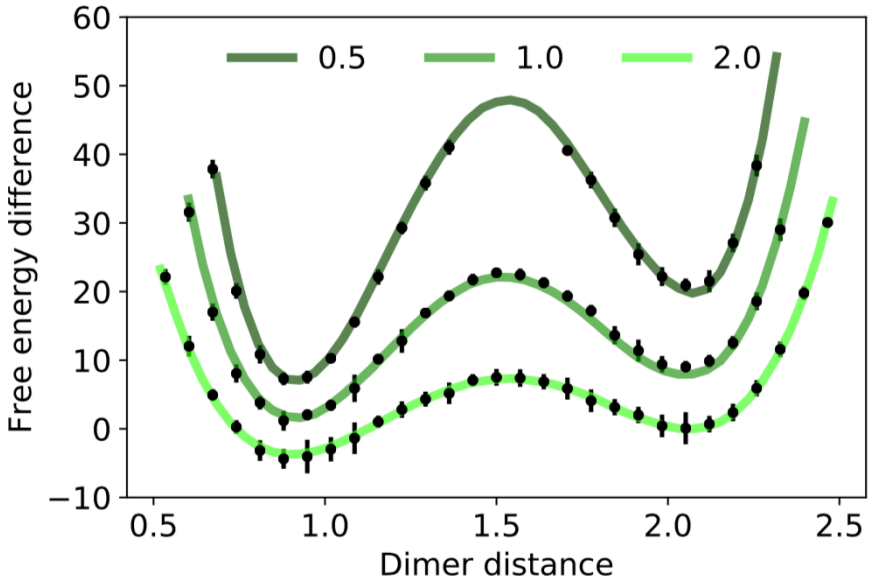


Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: Bistable dimer in dense colloid solvent

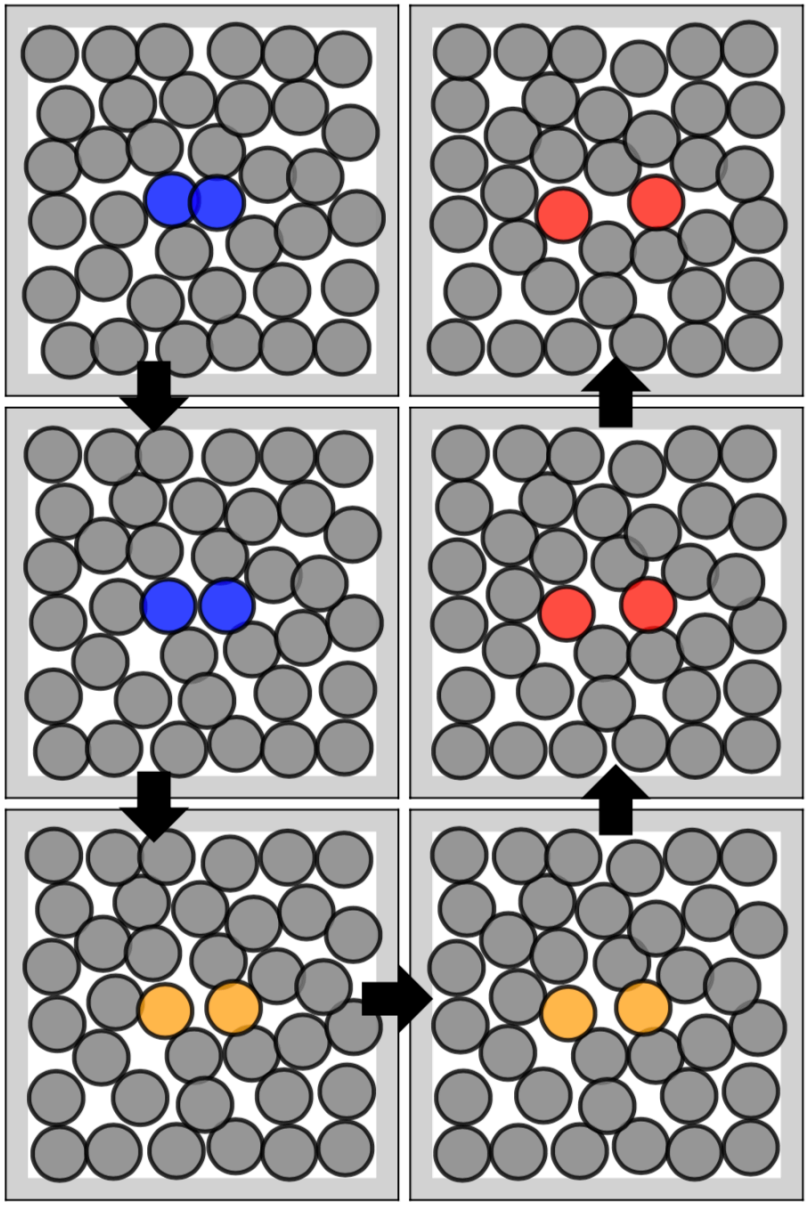






Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: Bistable dimer in dense colloid solvent



Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: Exploration

1. Sample batch $\{\mathbf{x}_1, ..., \mathbf{x}_B\}$ from X.

2. Update Boltzmann Generator parameters $\boldsymbol{\theta}$ by training on batch.

3. For each $\mathbf{x}$ in batch, propose a Metropolis Monte Carlo step in latent space with step-size s:

$$\mathbf{z}' = T_{xz}(\mathbf{x}) + s\mathcal{N}(\mathbf{0}, \mathbf{I}).$$

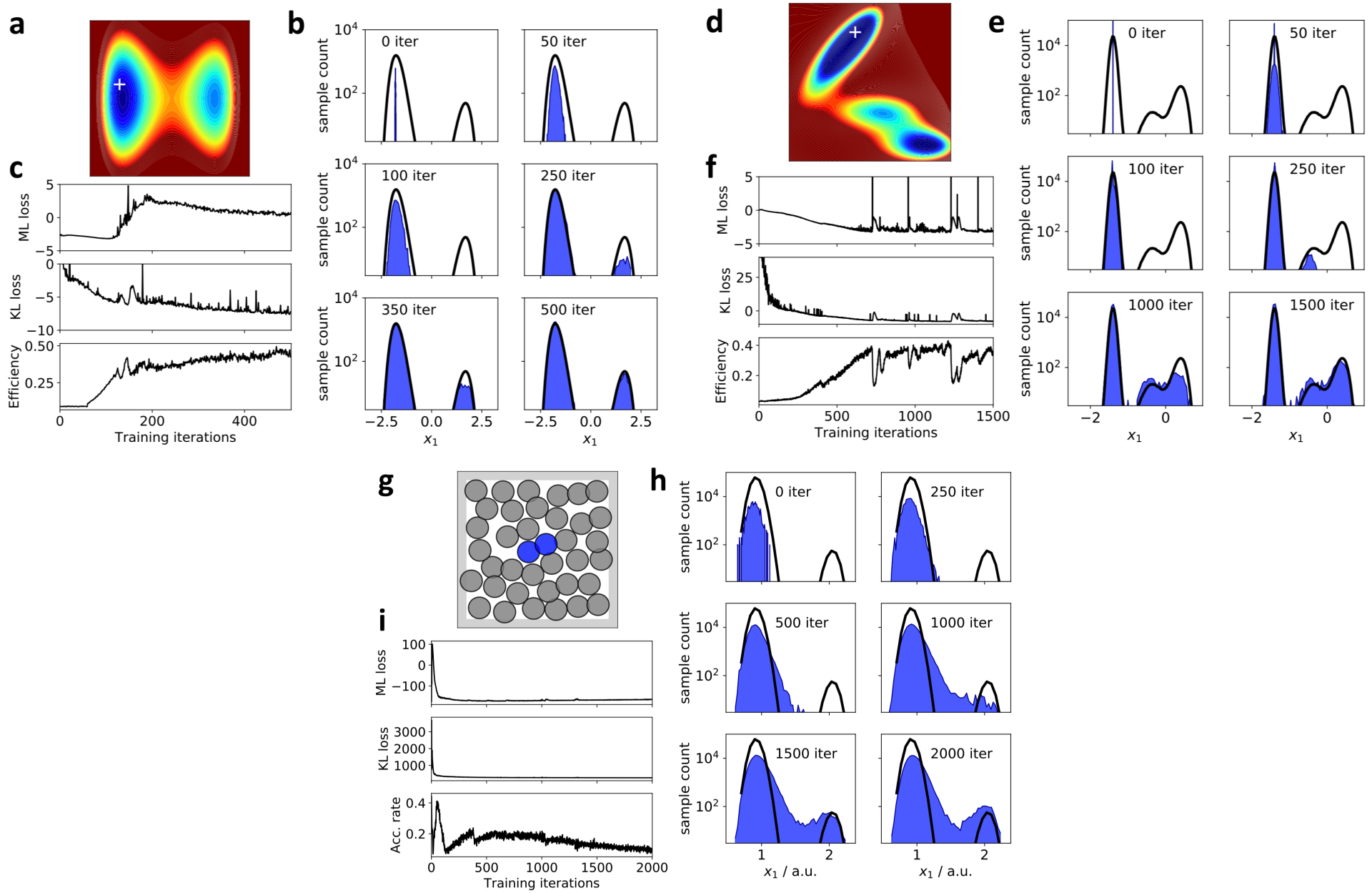
4. Accept or reject proposal with probability $\min\{1, \exp(-\Delta E)\}$ using the energy difference:

$$\Delta E = u(T_{zx}(\mathbf{z}')) - u(\mathbf{x}) - \log R_{zx}(\mathbf{z}'; \boldsymbol{\theta}) + \log R_{xz}(\mathbf{x}; \boldsymbol{\theta})$$

For the accepted samples, replace $\mathbf{x}$ by $\mathbf{x}' = T_{zx}(\mathbf{z}')$.

Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

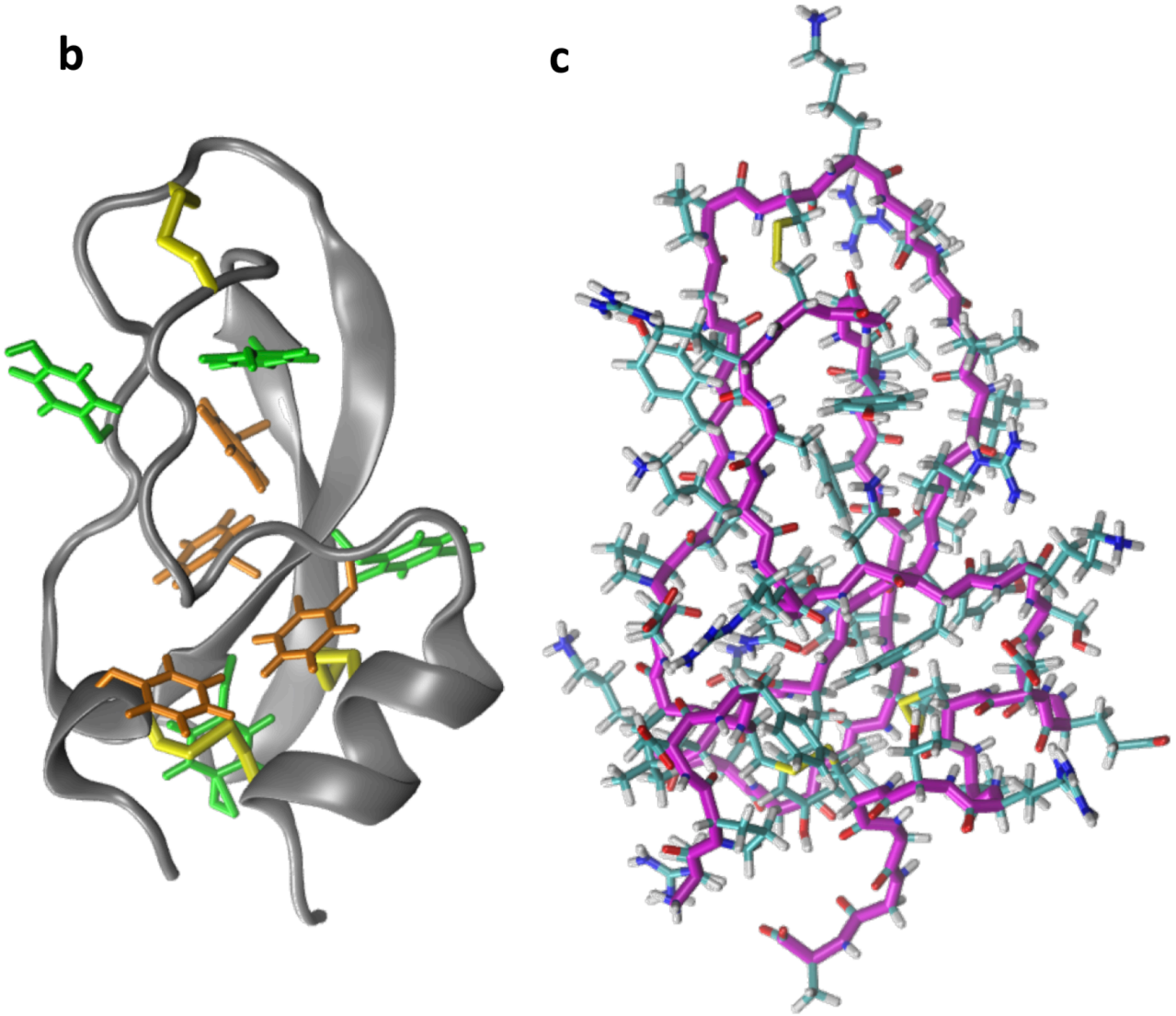
Boltzmann Generators: Exploration



Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

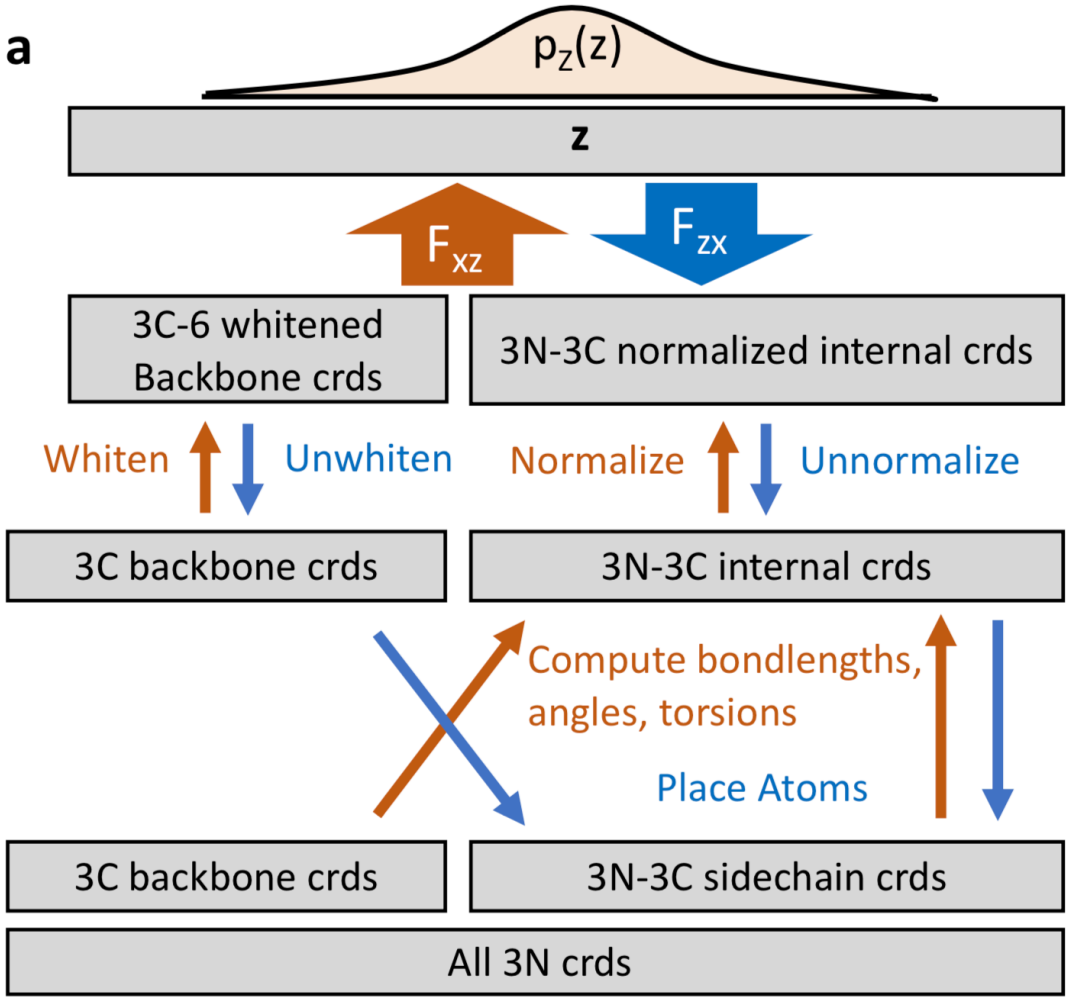
Freie Universität Berlin

Boltzmann Generators: towards proteins



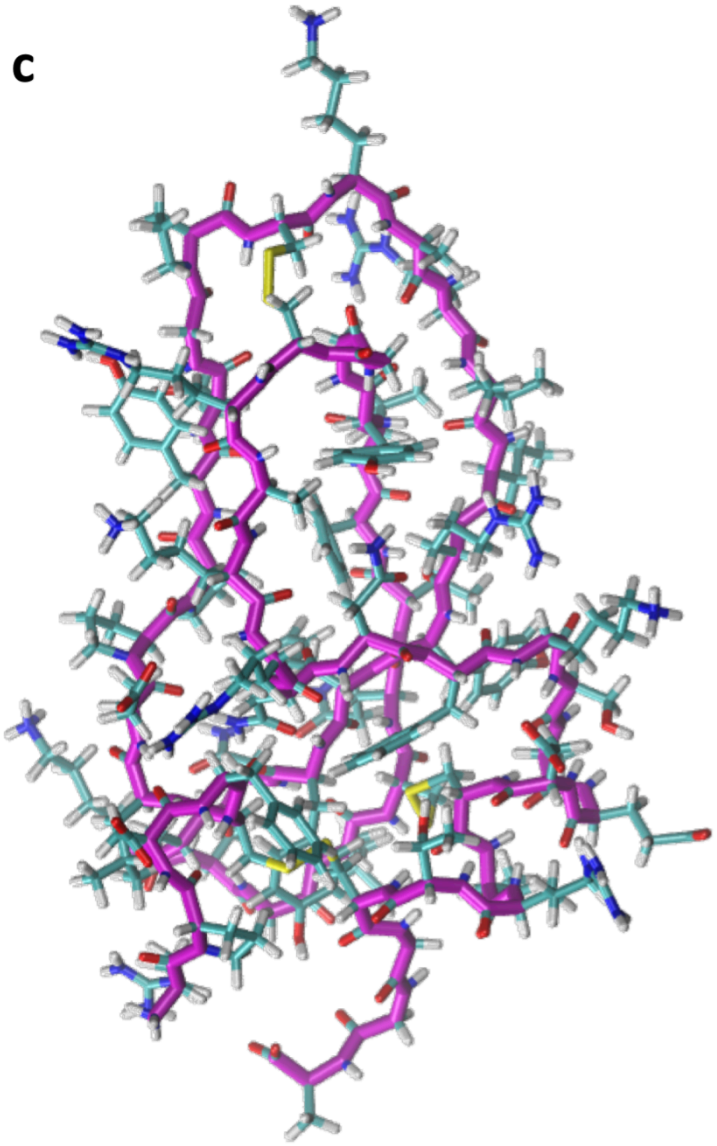
Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

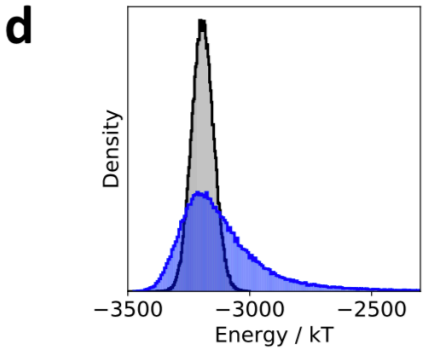
Boltzmann Generators: proteins

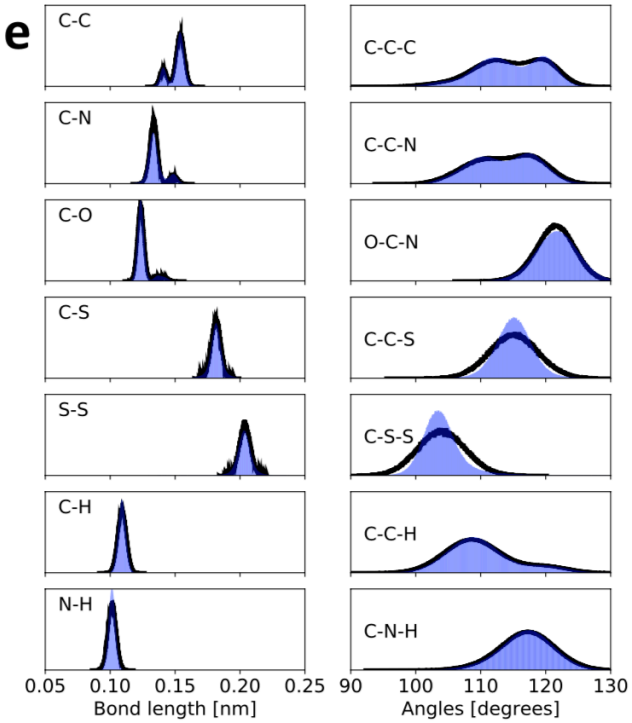


Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: proteins



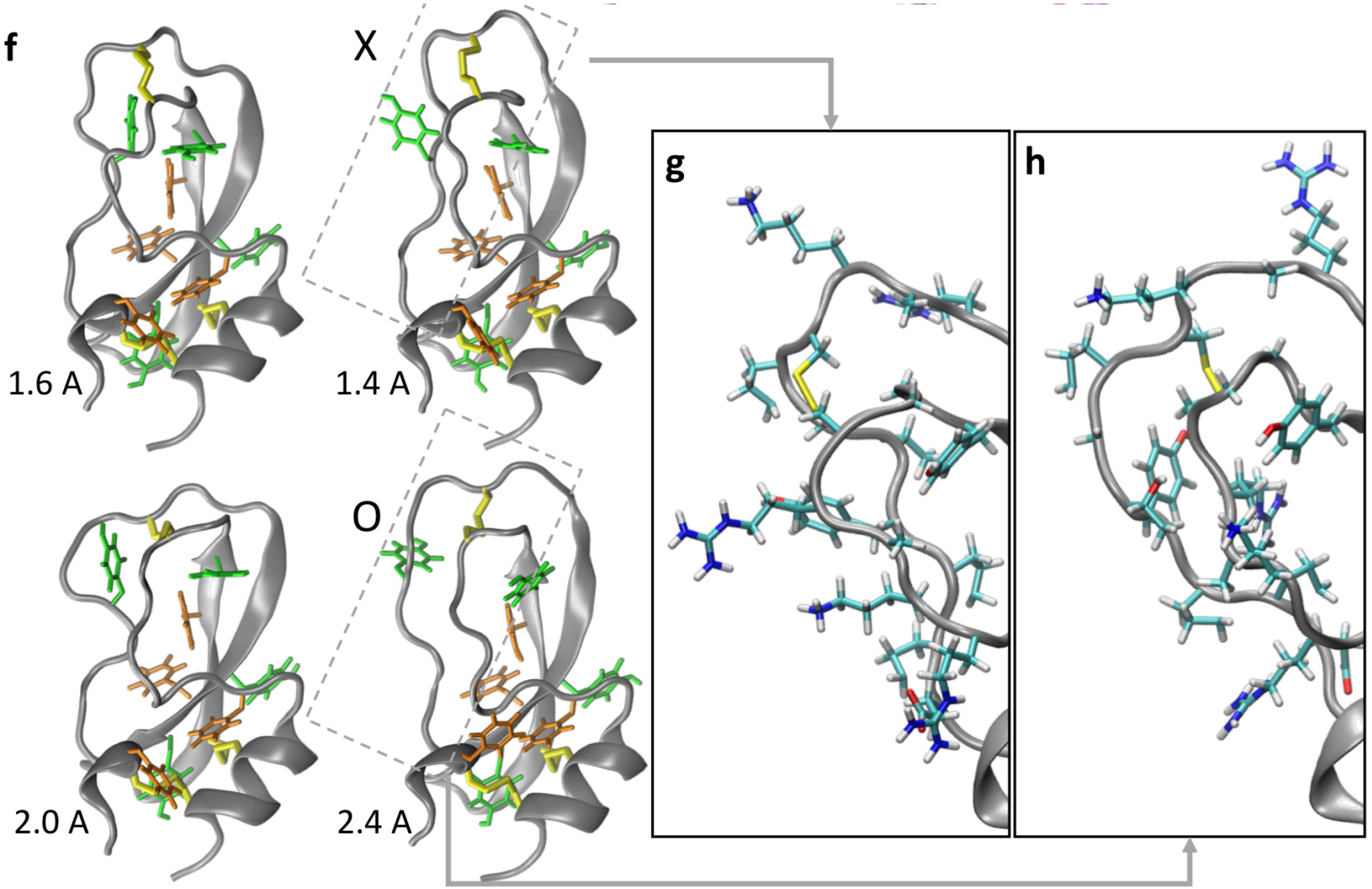




Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Freie Universität Berlin

Boltzmann Generators: proteins



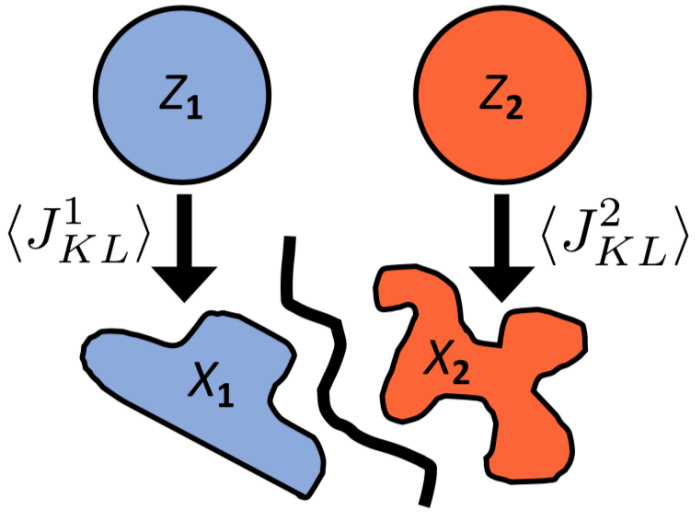
Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

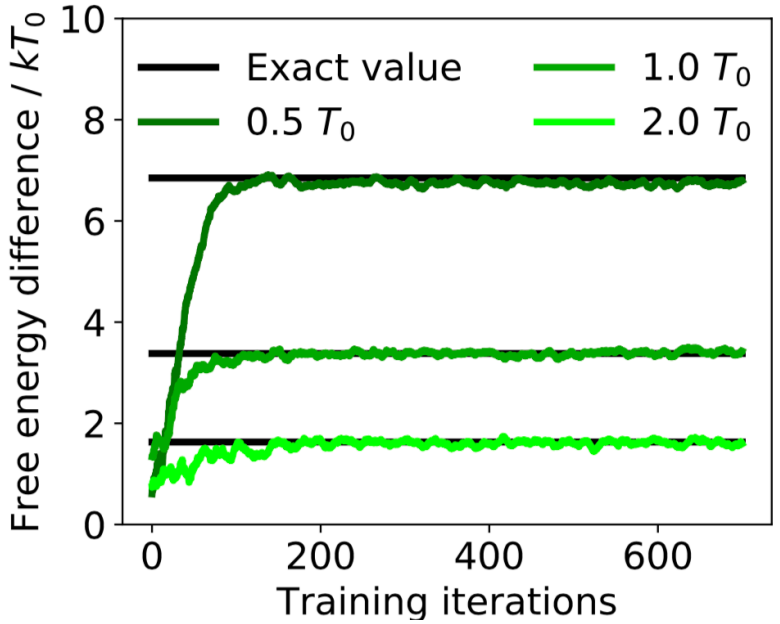
Boltzmann Generators: free energy differences

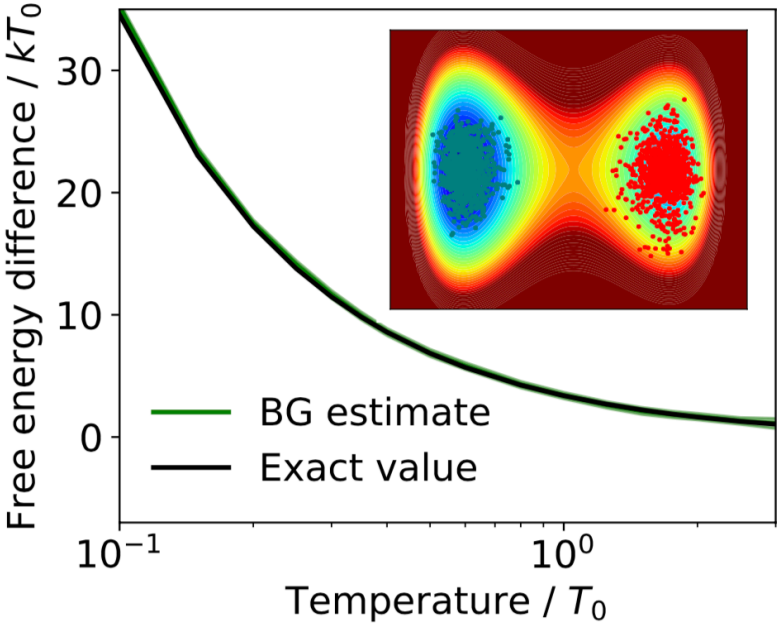
$$J_{KL} = \mathbb{E}_{\mathbf{z}} \left[u(F_{zx}(\mathbf{z})) - \log R_{zx}(\mathbf{z}) \right]$$

Free Energy difference from two independent Boltzmann Generators

$$\Delta A_{12} = \langle J^2_{KL} \rangle - \langle J^1_{KL} \rangle$$

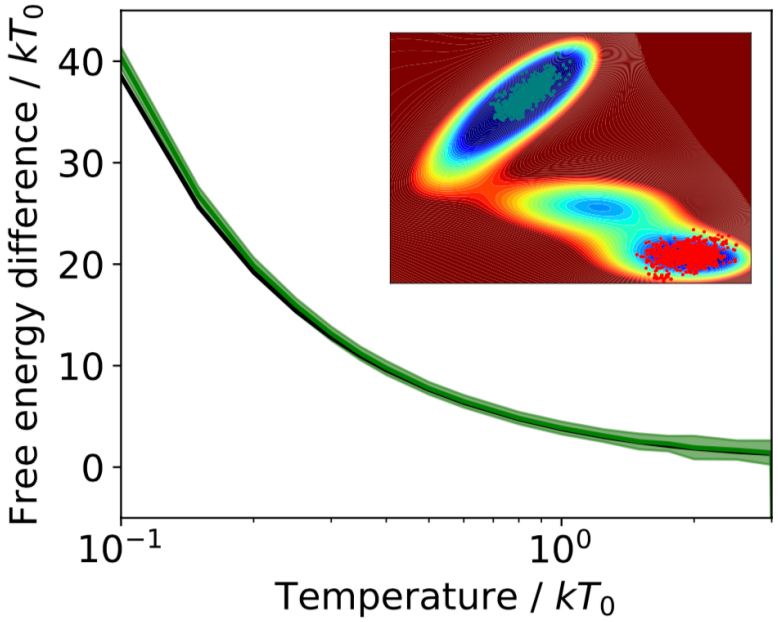


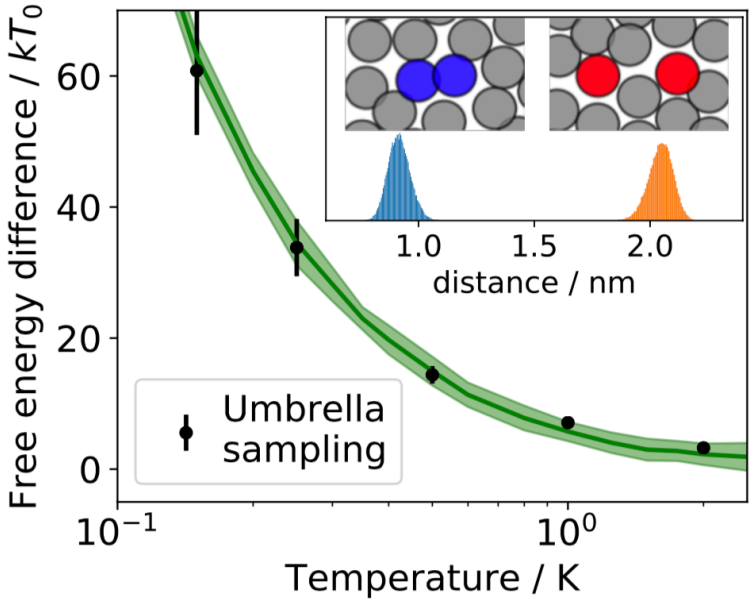


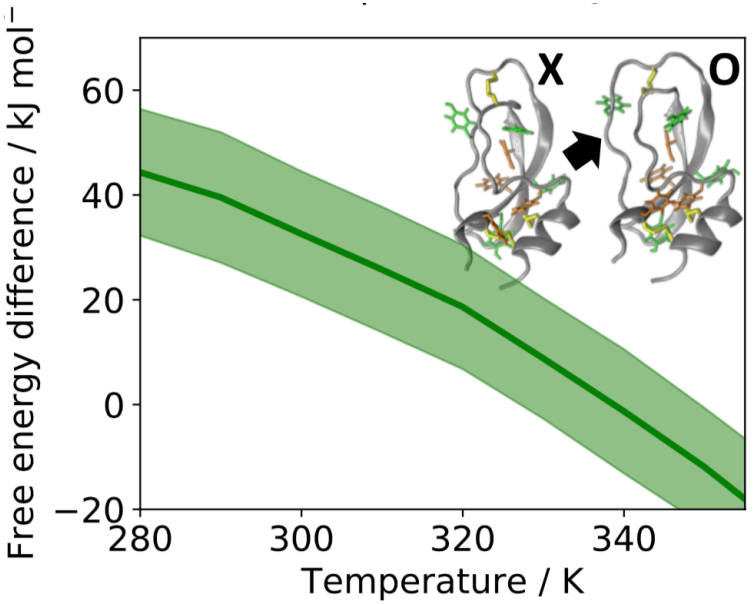


Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Boltzmann Generators: free energy differences







Noé, Olsson, Köhler, Wu, **Science** 365: eaww1147 (2019). arXiv:1812.01729

Acknowledgements

Collaborations

Cecilia Clementi (Rice University)
Christof Schütte (FU Berlin)
Eric Vanden-Eijnden (Courant Institut NY)
Thomas Weikl (MPI Potsdam)
Edina Rosta (King's College London)

Vijay Pande (Stanford)
Volker Haucke (FMP Berlin)
Stephan Sigrist (FU Berlin)
Katja, Faelber, Oliver Daumke (MDC)
John Chodera (MSKCC NY)
Gianni de Fabritiis (Barcelona)

Funding

erc
EINSTEIN Foundation.de
DFG Deutsche Forschungsgemeinschaft
Freie Universität Berlin