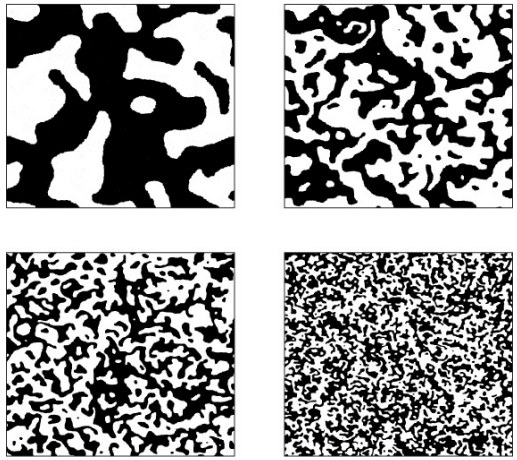
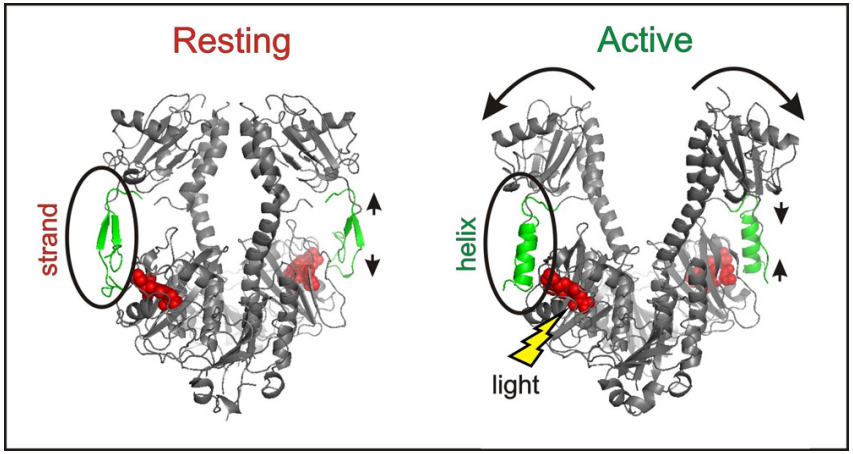
Monte Carlo Sampling with Deep Learning

F. Noé[1]

Group Seminar, July 28, 2018

Limitations of Monte Carlo Sampling



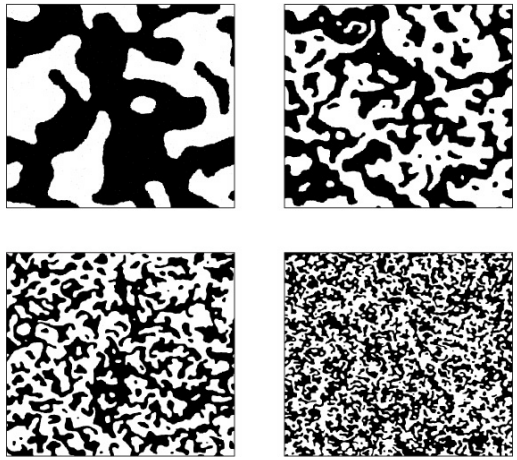


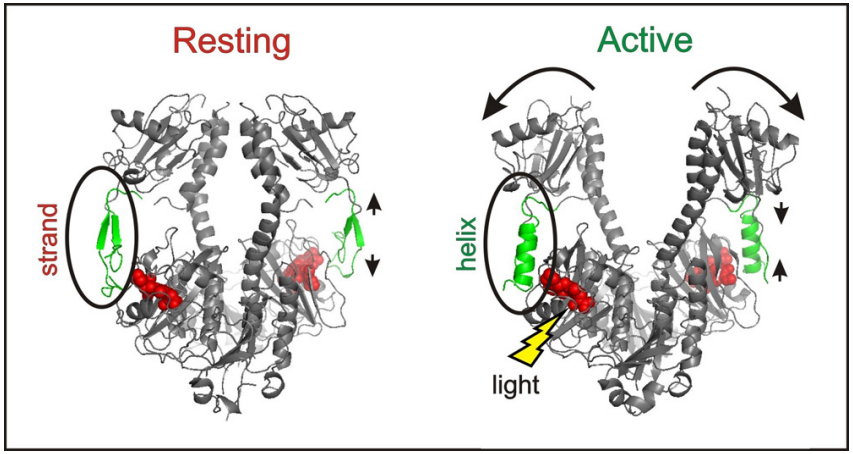
- **Input**: Reduced Potential Energy $u(\mathbf{x})$ in coordinates $\mathbf{x} \in \mathbb{R}^n$.
- **Aim**: Sample Equilibrium Distribution.

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

- **Problem 1**: For increasing n, the subvolume of low-energy configurations is vanishingly small compared to $\mathbb{R}^n$ and has a complex shape.
  - Direct MD sampling (with rejection or reweighting) in configuration space is hopeless .
- **Problem 2**: Metastable states or phases
  - MCMC or MD methods with small steps converge very slowly
  - Guessing large MCMC proposal steps is hard and problem-specific.

Limitations of Monte Carlo Sampling



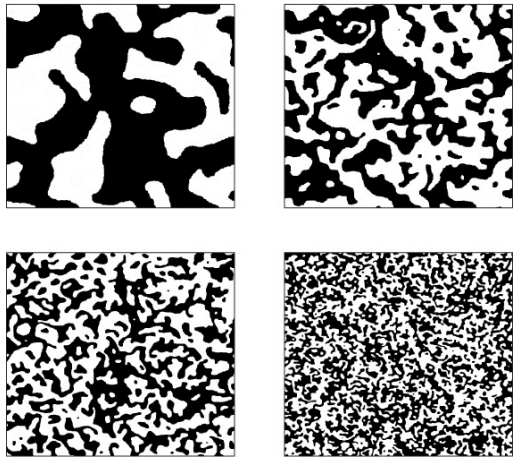


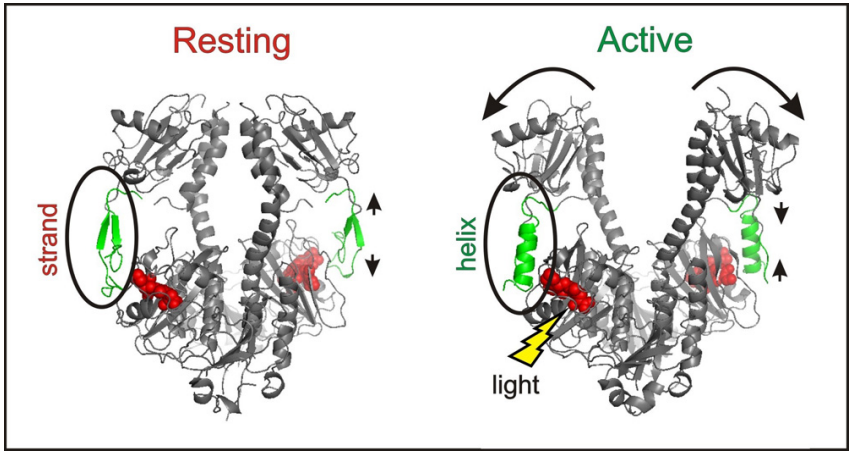
- **Input**: Reduced Potential Energy $u(\mathbf{x})$ in coordinates $\mathbf{x} \in \mathbb{R}^n$.
- **Aim**: Sample Equilibrium Distribution.

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

- **Problem 1**: For increasing n, the subvolume of low-energy configurations is vanishingly small compared to $\mathbb{R}^n$ and has a complex shape.
  - Direct MD sampling (with rejection or reweighting) in configuration space is hopeless .
- **Problem 2**: Metastable states or phases
  - MCMC or MD methods with small steps converge very slowly
  - Guessing large MCMC proposal steps is hard and problem-specific.

Limitations of Monte Carlo Sampling



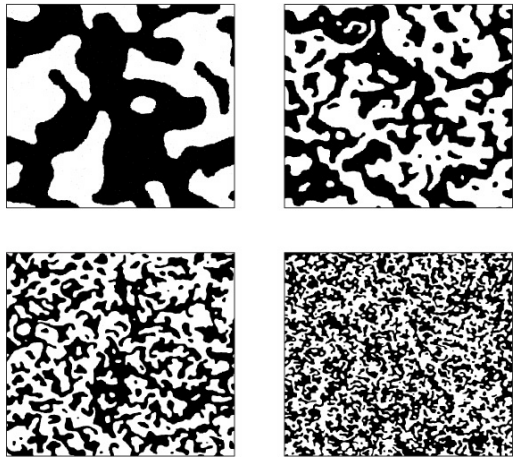


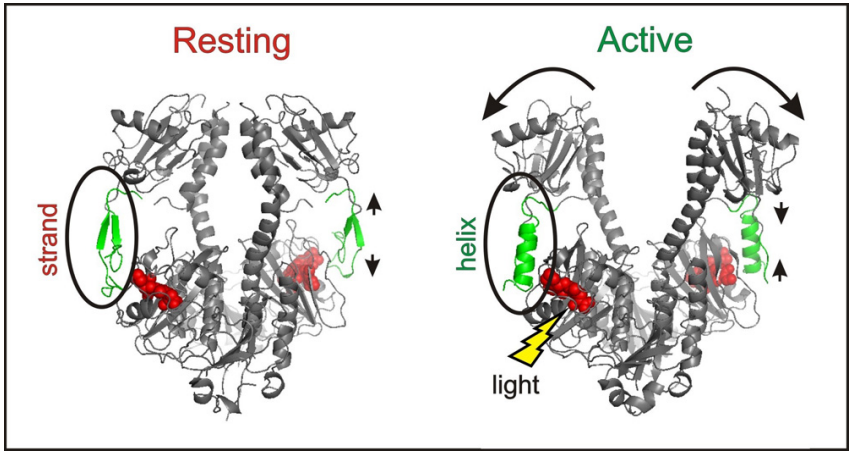
- **Input**: Reduced Potential Energy $u(\mathbf{x})$ in coordinates $\mathbf{x} \in \mathbb{R}^n$.
- **Aim**: Sample Equilibrium Distribution.

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

- **Problem 1**: For increasing n, the subvolume of low-energy configurations is vanishingly small compared to $\mathbb{R}^n$ and has a complex shape.
  - Direct MD sampling (with rejection or reweighting) in configuration space is hopeless .
- **Problem 2**: Metastable states or phases
  - MCMC or MD methods with small steps converge very slowly
  - Guessing large MCMC proposal steps is hard and problem-specific.

Limitations of Monte Carlo Sampling



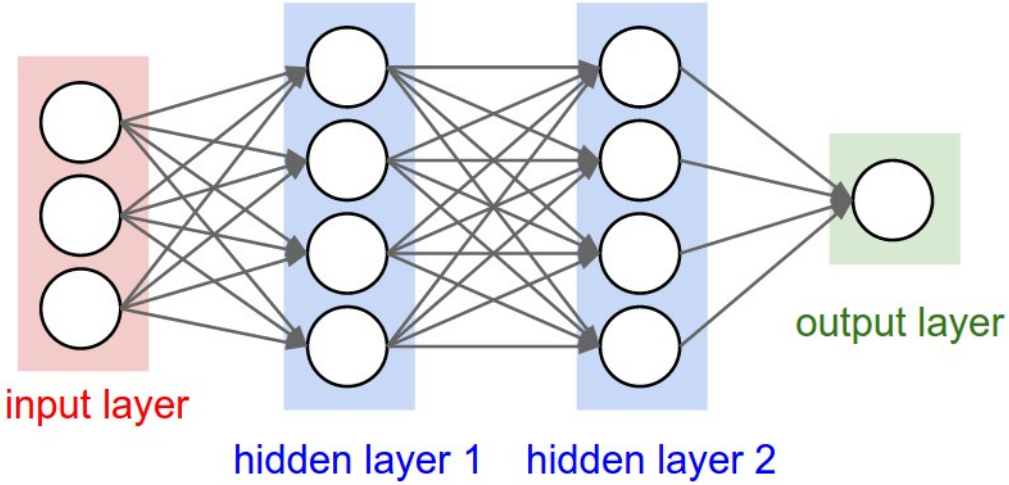


- **Input**: Reduced Potential Energy $u(\mathbf{x})$ in coordinates $\mathbf{x} \in \mathbb{R}^n$.
- **Aim**: Sample Equilibrium Distribution.

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

- **Problem 1**: For increasing n, the subvolume of low-energy configurations is vanishingly small compared to $\mathbb{R}^n$ and has a complex shape.
  - Direct MD sampling (with rejection or reweighting) in configuration space is hopeless .
- **Problem 2**: Metastable states or phases
  - MCMC or MD methods with small steps converge very slowly
  - Guessing large MCMC proposal steps is hard and problem-specific.

Reminder: Multilayer Neural Networks

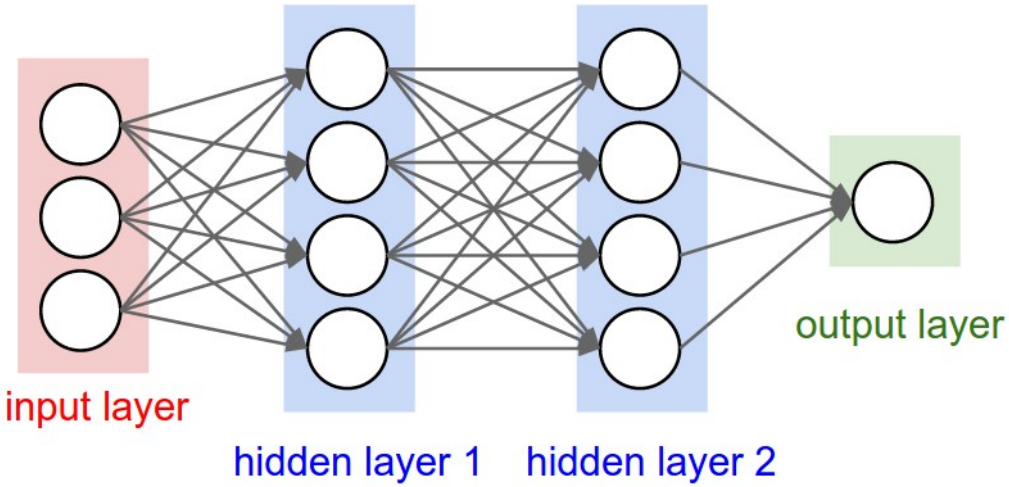


Sequence of linear and nonlinear transforms defined by the recursion:

$$\mathbf{x}^{l+1} = \sigma\left(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l\right)$$

- L layers indexed by $l = 1, ..., L$. Input vector $\mathbf{x}^{(0)}$ does not count as a layer.
- $\mathbf{x}^l \in \mathbb{R}^{n_l}$: Activations of n_l neurons at layer l
- Trainable weights $\mathbf{W}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$ at each layer. W^l_{ij} is connecting neuron j of layer $l-1$ with neuron i of layer l.
- Nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$.
- Output vector: $\hat{\mathbf{y}} = \mathbf{x}^L$ ($\hat{\mathbf{y}}$: network predictions. $\mathbf{y}$: training values)

Reminder: Multilayer Neural Networks

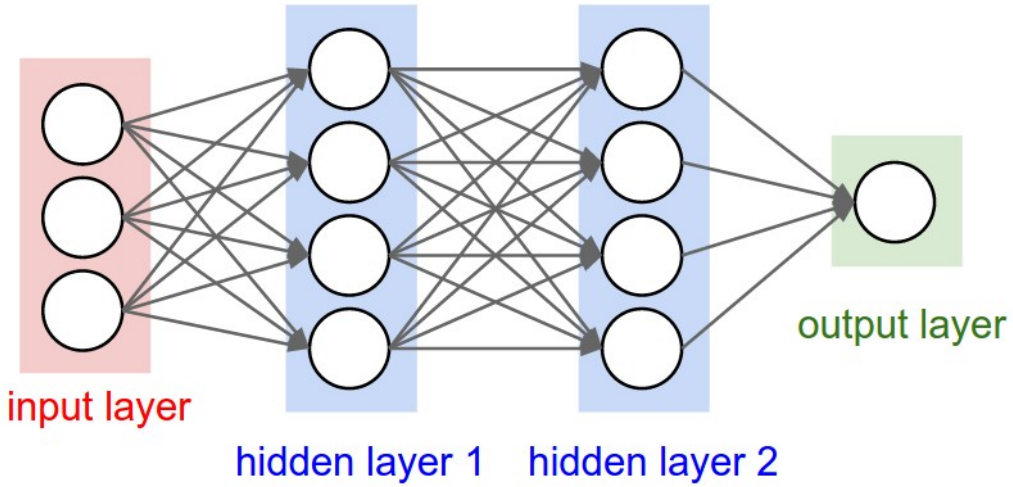


Sequence of linear and nonlinear transforms defined by the recursion:

$$\mathbf{x}^{l+1} = \sigma\left(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l\right)$$

- L layers indexed by $l = 1, \ldots, L$. Input vector $\mathbf{x}^{(0)}$ does not count as a layer.
- $\mathbf{x}^l \in \mathbb{R}^{n_l}$: Activations of n_l neurons at layer l
- Trainable weights $\mathbf{W}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$ at each layer. W^l_{ij} is connecting neuron j of layer $l-1$ with neuron i of layer l.
- Nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$.
- Output vector: $\hat{\mathbf{y}} = \mathbf{x}^l$ ($\hat{\mathbf{y}}$: network predictions. $\mathbf{y}$: training values)

Reminder: Multilayer Neural Networks

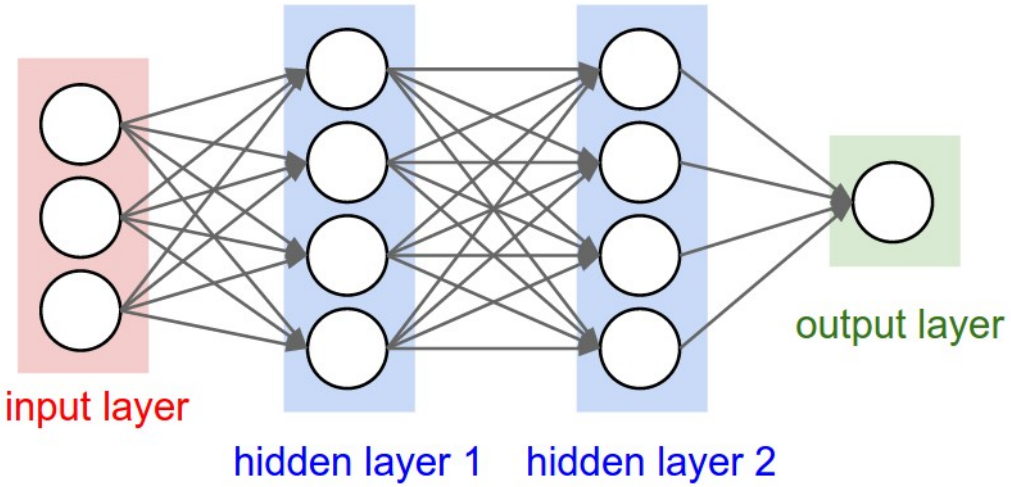


Sequence of linear and nonlinear transforms defined by the recursion:

$$\mathbf{x}^{l+1} = \sigma\left(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l\right)$$

- L layers indexed by $l = 1, \ldots, L$. Input vector $\mathbf{x}^{(0)}$ does not count as a layer.
- $\mathbf{x}^l \in \mathbb{R}^{n_l}$: Activations of n_l neurons at layer l
- Trainable weights $\mathbf{W}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$ at each layer. W^l_{ij} is connecting neuron j of layer $l-1$ with neuron i of layer l.
- Nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$.
- Output vector: $\hat{\mathbf{y}} = \mathbf{x}^l$ ($\hat{\mathbf{y}}$: network predictions. $\mathbf{y}$: training values)

Reminder: Multilayer Neural Networks

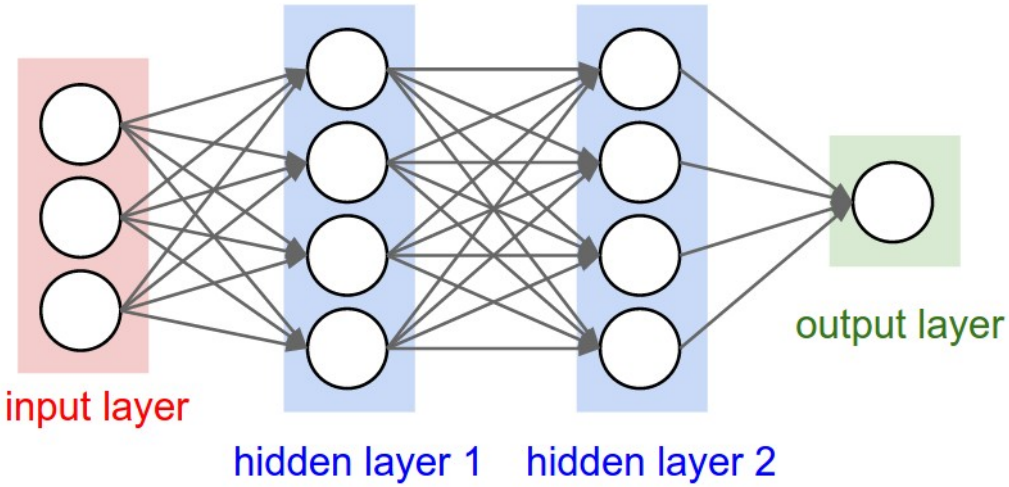


Sequence of linear and nonlinear transforms defined by the recursion:

$$\mathbf{x}^{l+1} = \sigma\left(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l\right)$$

- L layers indexed by $l = 1, \ldots, L$. Input vector $\mathbf{x}^{(0)}$ does not count as a layer.
- $\mathbf{x}^l \in \mathbb{R}^{n_l}$: Activations of n_l neurons at layer l
- Trainable weights $\mathbf{W}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$ at each layer. W^l_{ij} is connecting neuron j of layer $l-1$ with neuron i of layer l.
- Nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$.
- Output vector: $\hat{\mathbf{y}} = \mathbf{x}^l$ ($\hat{\mathbf{y}}$: network predictions. $\mathbf{y}$: training values)

Reminder: Multilayer Neural Networks



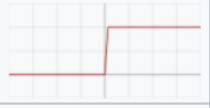
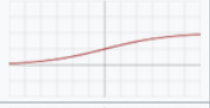
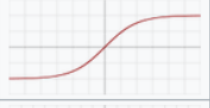
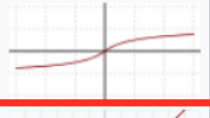
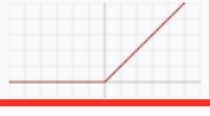
Sequence of linear and nonlinear transforms defined by the recursion:

$$\mathbf{x}^{l+1} = \sigma\left(\mathbf{W}^l \mathbf{x}^l + \mathbf{b}^l\right)$$

- L layers indexed by $l = 1, \ldots, L$. Input vector $\mathbf{x}^{(0)}$ does not count as a layer.
- $\mathbf{x}^l \in \mathbb{R}^{n_l}$: Activations of n_l neurons at layer l
- Trainable weights $\mathbf{W}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ and biases $\mathbf{b}^l \in \mathbb{R}^{n_l}$ at each layer. W^l_{ij} is connecting neuron j of layer $l-1$ with neuron i of layer l.
- Nonlinear function $\sigma : \mathbb{R} \to \mathbb{R}$.
- Output vector: $\hat{\mathbf{y}} = \mathbf{x}^l$ ($\hat{\mathbf{y}}$: network predictions. $\mathbf{y}$: training values)

Reminder: Multilayer Neural Networks

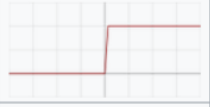
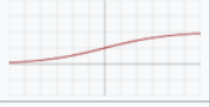
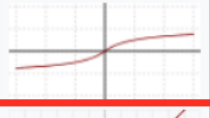
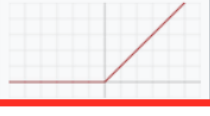
- **Universal Represenation Theorem**: *(Under some mild conditions), a neural network with **one** hidden layer and sufficiently many hidden neurons can approximate any continuous function $F : \mathbb{R}^{n_0} \mapsto \mathbb{R}^{n_2}$ with arbitrary accuracy.*
- **Deeper neural networks** often perform better as they can model complex functions with less hidden neurons.
- Avoid vanishing/exploding gradients: **nonsaturating nonlinearity**

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

Currently most widely used. Empirically easier to train and results in sparse networks.

Reminder: Multilayer Neural Networks

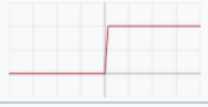
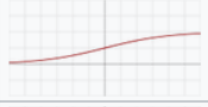
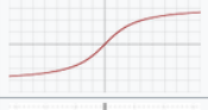
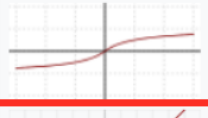
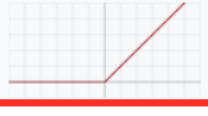
- **Universal Represenation Theorem**: *(Under some mild conditions), a neural network with **one** hidden layer and sufficiently many hidden neurons can approximate any continuous function* $F : \mathbb{R}^{n_0} \mapsto \mathbb{R}^{n_2}$ *with arbitrary accuracy.*
- **Deeper neural networks** often perform better as they can model complex functions with less hidden neurons.
- Avoid vanishing/exploding gradients: **nonsaturating nonlinearity**

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

Currently most widely used. Empirically easier to train and results in sparse networks.

Reminder: Multilayer Neural Networks

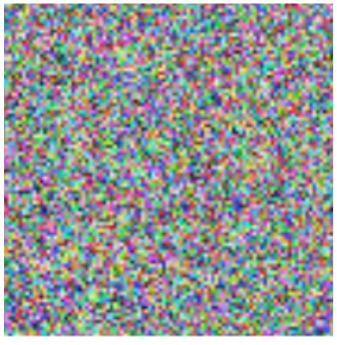
- **Universal Represenation Theorem**: *(Under some mild conditions), a neural network with **one** hidden layer and sufficiently many hidden neurons can approximate any continuous function* $F : \mathbb{R}^{n_0} \mapsto \mathbb{R}^{n_2}$ *with arbitrary accuracy.*
- **Deeper neural networks** often perform better as they can model complex functions with less hidden neurons.
- Avoid vanishing/exploding gradients: **nonsaturating nonlinearity**

Identity		$f(x) = x$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1+e^{-x}}$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$
ArcTan		$f(x) = \tan^{-1}(x)$
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$
Rectified linear unit (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$

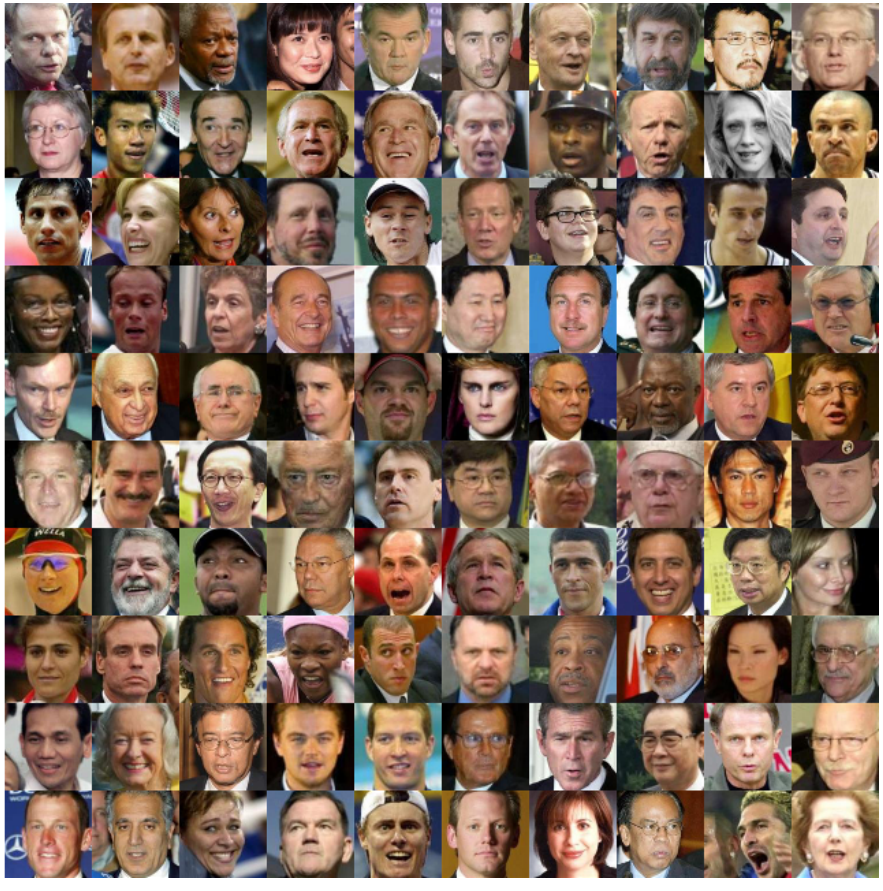
Currently most widely used. Empirically easier to train and results in sparse networks.

Reminder: Generative Neural Networks

Noise ~ N(0,1)



Generative Model



Reminder: Generative Neural Networks

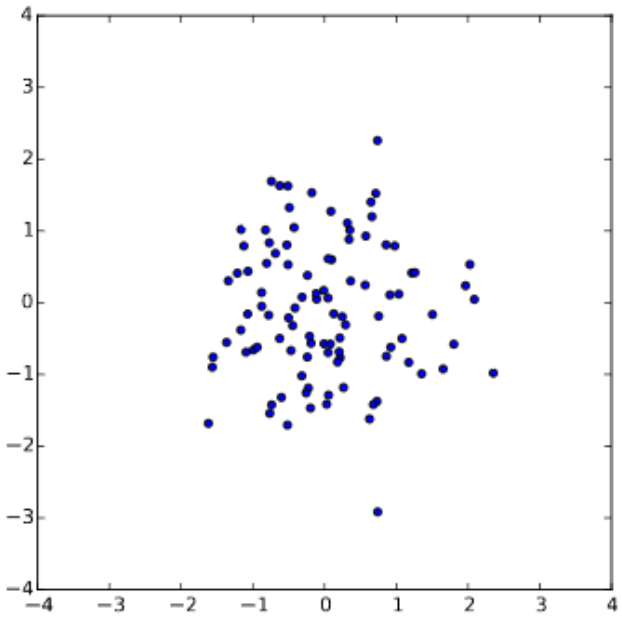
- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

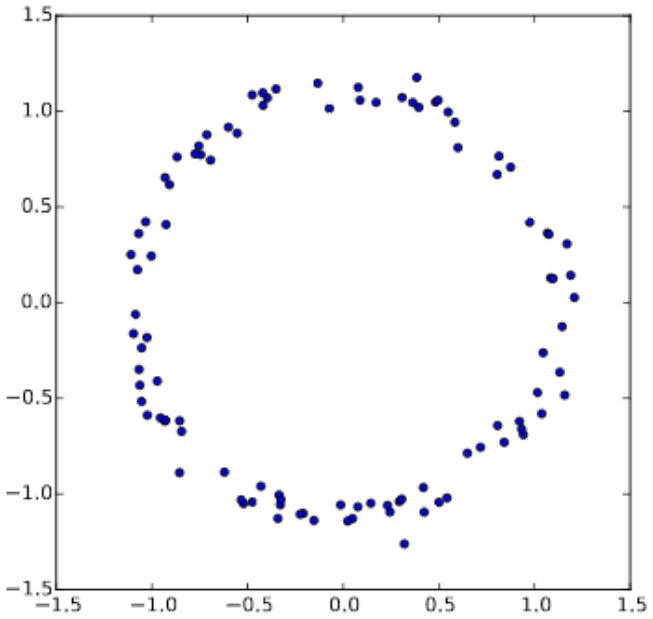
$$\mathbf{z} \sim p(\mathbf{z})$$

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.





Reminder: Generative Neural Networks

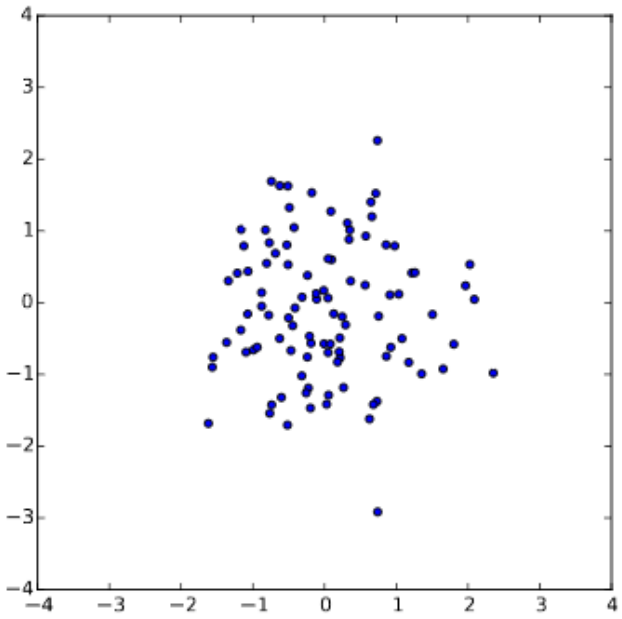
- **Idea**: Learn to sample intractable $p(\mathbf{x})$ by sampling tractable latent distribution

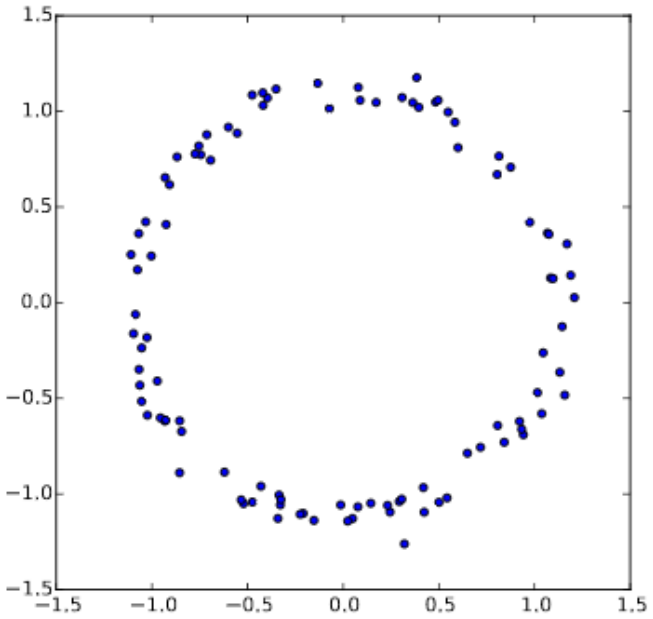
$$\mathbf{z} \sim p(\mathbf{z})$$

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.

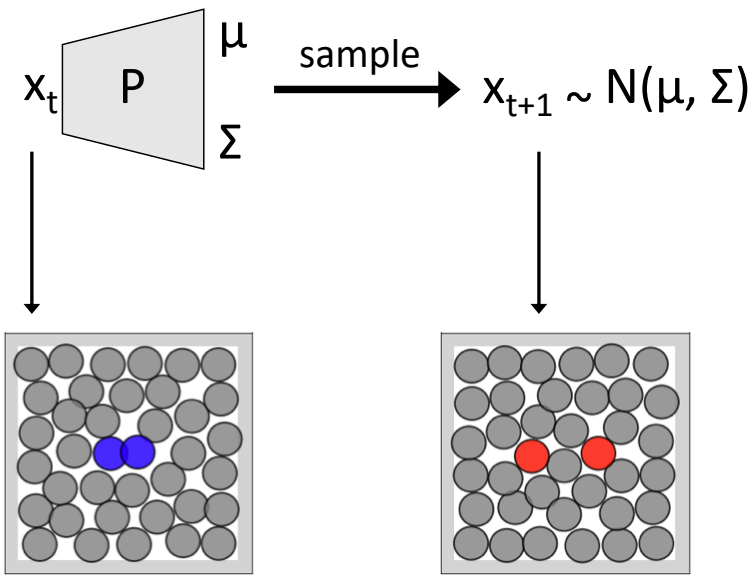




Two New Monte Carlo Approaches

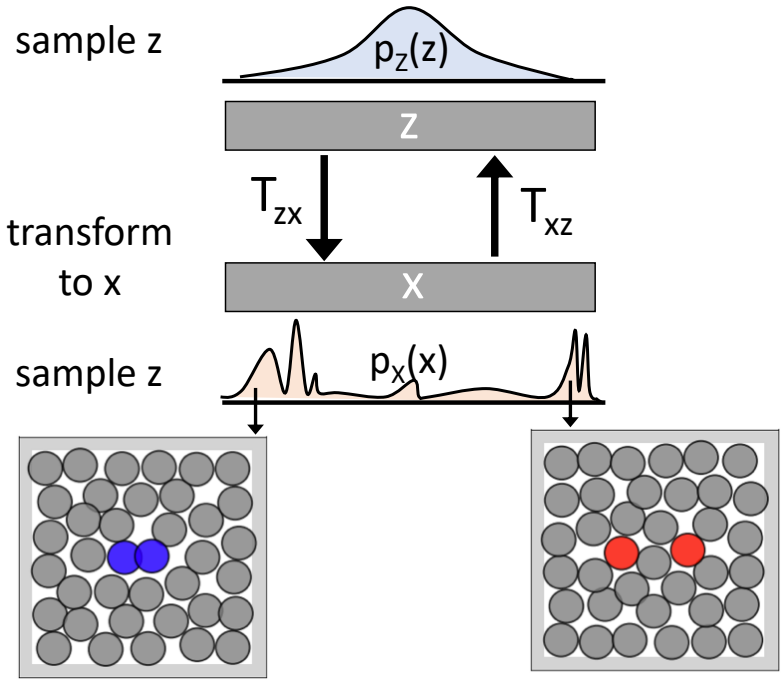
Adaptive MCMC

Aim: Learn proposal scheme P
to achieve efficient MCMC



Latent MC

Aim: Learn transformation Txz of data to latent
space such that direct MC is efficient



Neural Adaptive MCMC

- **Gaussian Encoder** - proposal step:

$$\begin{array}{llll} \mathbf{x} & \overset{F}{\longmapsto} \mu(\mathbf{x}), \mathbf{A}(\mathbf{x}) & \searrow & \\ \mathbf{w} & \sim \mathcal{N}(\cdot \mid \mathbf{0}, \mathbf{Id}) & \rightarrow \quad \mathbf{y} & = \mu(\mathbf{x}) + \mathbf{A}(\mathbf{x})\mathbf{w} \end{array}$$

  with $\mathbf{A}$ an upper diagonal matrix or diagonal matrix, and $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^\top$.

- **Proposal probabilities**:

$$p_{\text{prop}}(\mathbf{x} \to \mathbf{y}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}\mathbf{w}^\top \mathbf{w}}$$

$$p_{\text{prop}}(\mathbf{y} \to \mathbf{x}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\mu(\mathbf{y}))^\top \left(\mathbf{A}(\mathbf{y})\mathbf{A}^\top(\mathbf{y})\right)^{-1}(\mathbf{x}-\mu(\mathbf{y}))}$$

- **Network**:

$$\begin{array}{ccccccccc} \mathbf{x} & \overset{F_\mu}{\to} & \mu(\mathbf{x}) & \to & \oplus & \to & \mathbf{y} & \overset{F_\mu}{\to} & \mu(\mathbf{y}) \\ & & & & \uparrow & & F_A & \searrow & \\ \mathbf{x} & \overset{F_A}{\to} & \mathbf{A}(\mathbf{x}) & \to & \otimes & & & & \mathbf{A}(\mathbf{y}) \\ & & & & \nearrow & & & & \\ \mathbf{w} & \to & \to & & & & & & \end{array}$$

Neural Adaptive MCMC

- **Gaussian Encoder** - proposal step:

$$\begin{array}{llll} \mathbf{x} & \overset{F}{\longmapsto} \mu(\mathbf{x}), \mathbf{A}(\mathbf{x}) & \searrow & \\ \mathbf{w} & \sim \mathcal{N}(\cdot \mid \mathbf{0}, \mathbf{Id}) & \rightarrow \quad \mathbf{y} & = \mu(\mathbf{x}) + \mathbf{A}(\mathbf{x})\mathbf{w} \end{array}$$

  with $\mathbf{A}$ an upper diagonal matrix or diagonal matrix, and $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^\top$.

- **Proposal probabilities**:

$$p_{\text{prop}}(\mathbf{x} \to \mathbf{y}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}\mathbf{w}^\top \mathbf{w}}$$

$$p_{\text{prop}}(\mathbf{y} \to \mathbf{x}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\mu(\mathbf{y}))^\top \left(\mathbf{A}(\mathbf{y})\mathbf{A}^\top(\mathbf{y})\right)^{-1}(\mathbf{x}-\mu(\mathbf{y}))}$$

- **Network**:

Neural Adaptive MCMC

- **Gaussian Encoder** - proposal step:

$$\begin{array}{llll} \mathbf{x} & \overset{F}{\longmapsto} \mu(\mathbf{x}), \mathbf{A}(\mathbf{x}) & \searrow & \\ \mathbf{w} & \sim \mathcal{N}(\cdot \mid \mathbf{0}, \mathbf{Id}) & \rightarrow \quad \mathbf{y} & = \mu(\mathbf{x}) + \mathbf{A}(\mathbf{x})\mathbf{w} \end{array}$$

  with $\mathbf{A}$ an upper diagonal matrix or diagonal matrix, and $\mathbf{\Sigma} = \mathbf{A}\mathbf{A}^\top$.

- **Proposal probabilities**:

$$p_{\text{prop}}(\mathbf{x} \to \mathbf{y}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}\mathbf{w}^\top \mathbf{w}}$$

$$p_{\text{prop}}(\mathbf{y} \to \mathbf{x}) = \frac{1}{\sqrt{2\pi}\prod_{i=1}^{n} A_{ii}} \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\mu(\mathbf{y}))^\top \left(\mathbf{A}(\mathbf{y})\mathbf{A}^\top(\mathbf{y})\right)^{-1}(\mathbf{x}-\mu(\mathbf{y}))}$$

- **Network**:

$$\begin{array}{lllllllll} \mathbf{x} & \overset{F_\mu}{\to} & \mu(\mathbf{x}) & \to & \oplus & \to & \mathbf{y} & \overset{F_\mu}{\to} & \mu(\mathbf{y}) \\ & & & & \uparrow & & F_A & \searrow & \\ \mathbf{x} & \overset{F_A}{\to} & \mathbf{A}(\mathbf{x}) & \to & \otimes & & & & \mathbf{A}(\mathbf{y}) \\ & & & & \nearrow & & & & \\ \mathbf{w} & \to & \to & & & & & & \end{array}$$

Neural Adaptive MCMC

- **Loss function** - MCMC efficiency:

$$S(\mathbf{x} \to \mathbf{y}) = p_{acc}(\mathbf{x} \to \mathbf{y}) \, \|\mathbf{x} - \mathbf{y}\|^2$$

- Use **Barker acceptance probability**.

$$p_{\text{acc}}(\mathbf{x} \to \mathbf{y}) = \frac{\mu(\mathbf{y}) p_{\text{prop}}(\mathbf{y} \to \mathbf{x})}{\mu(\mathbf{x}) p_{\text{prop}}(\mathbf{x} \to \mathbf{y}) + \mu(\mathbf{y}) p_{\text{prop}}(\mathbf{y} \to \mathbf{x})}$$

Neural Adaptive MCMC

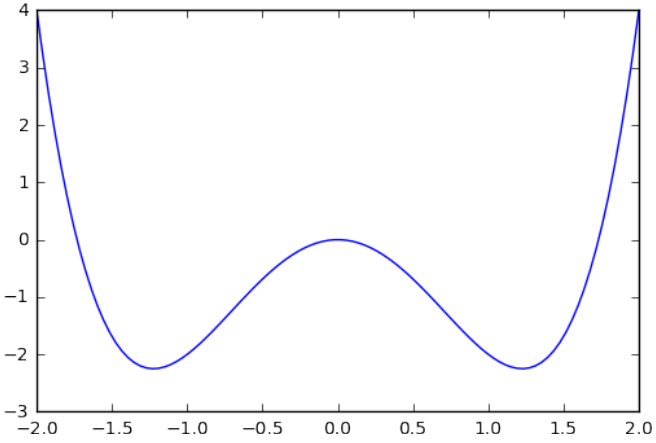
- **Loss function** - MCMC efficiency:

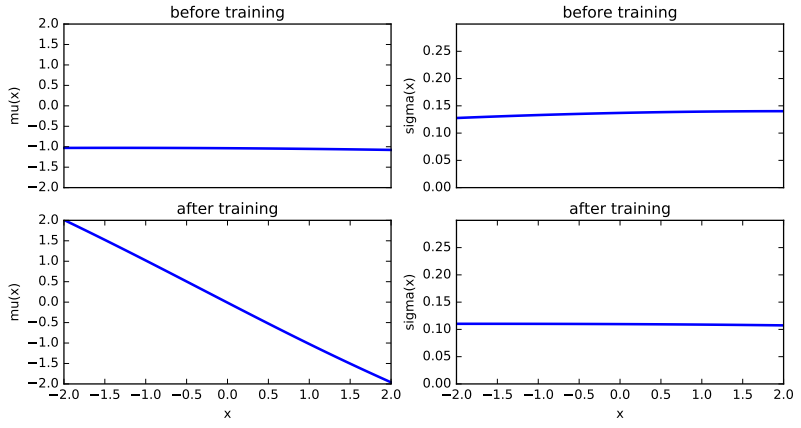
$$S(\mathbf{x} \to \mathbf{y}) = p_{acc}(\mathbf{x} \to \mathbf{y}) \, \|\mathbf{x} - \mathbf{y}\|^2$$

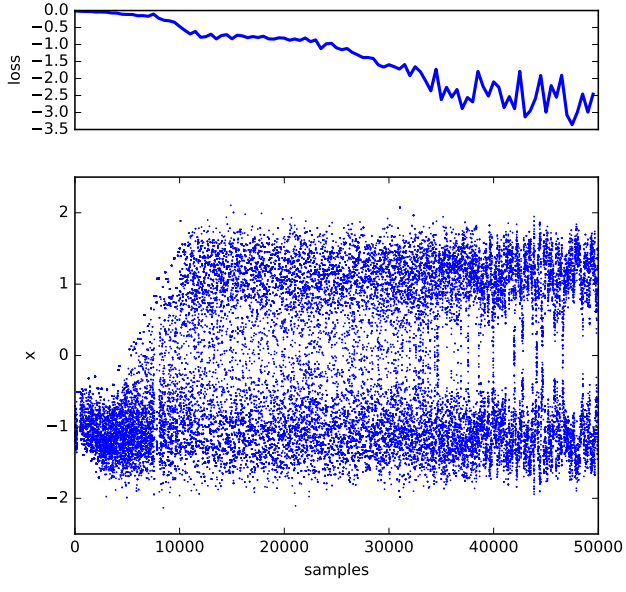
- Use **Barker acceptance probability**.

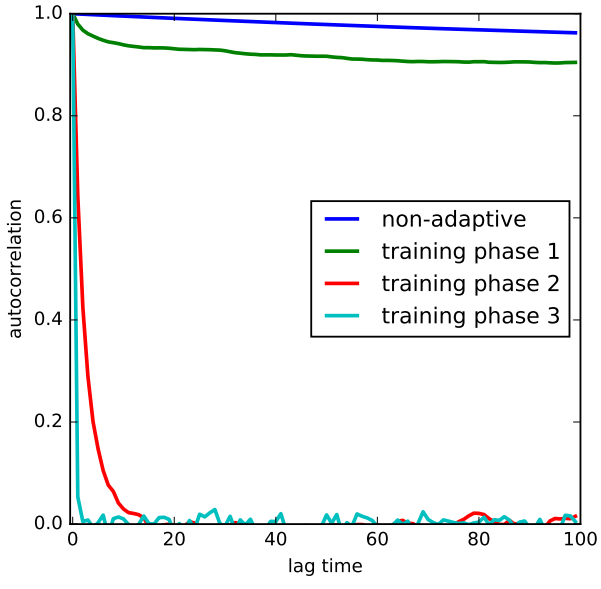
$$p_{\text{acc}}(\mathbf{x} \to \mathbf{y}) = \frac{\mu(\mathbf{y}) p_{\text{prop}}(\mathbf{y} \to \mathbf{x})}{\mu(\mathbf{x}) p_{\text{prop}}(\mathbf{x} \to \mathbf{y}) + \mu(\mathbf{y}) p_{\text{prop}}(\mathbf{y} \to \mathbf{x})}$$

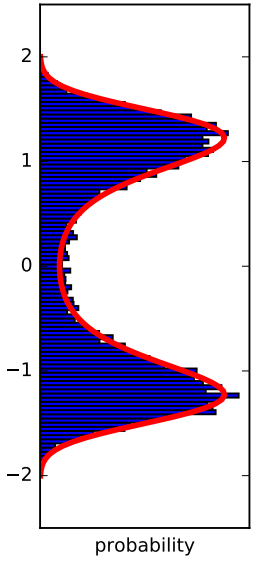
Neural Adaptive MCMC - 1D Example



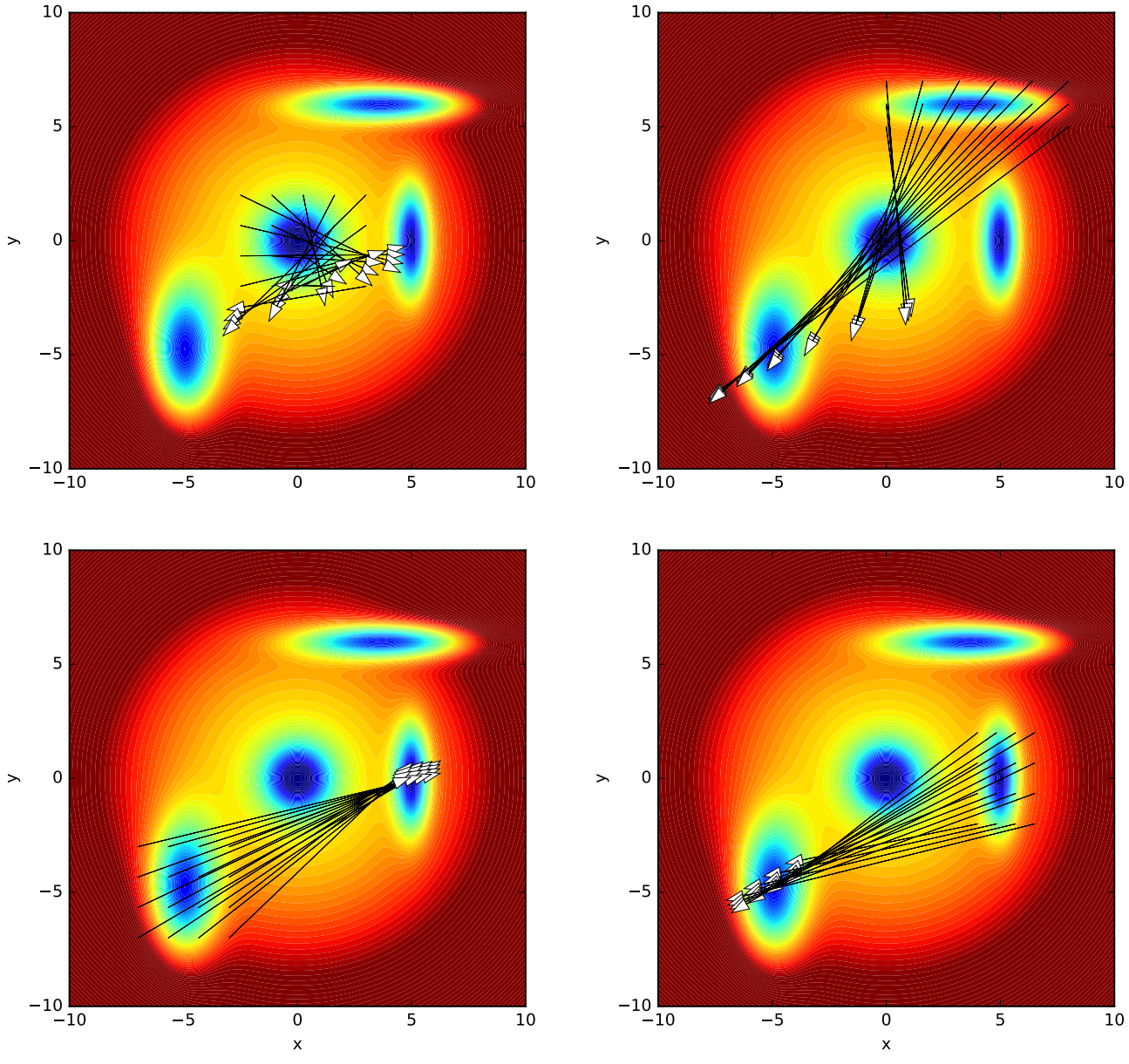








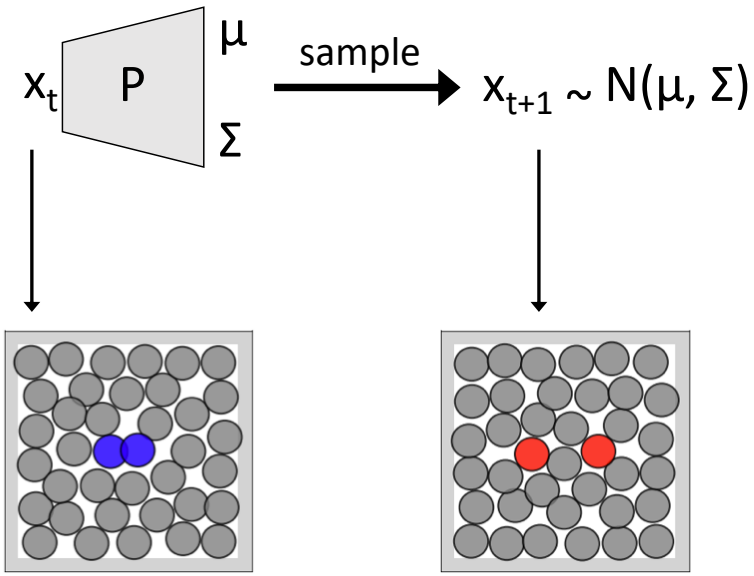
Neural Adaptive MCMC - 2D example



Two New Monte Carlo Approaches

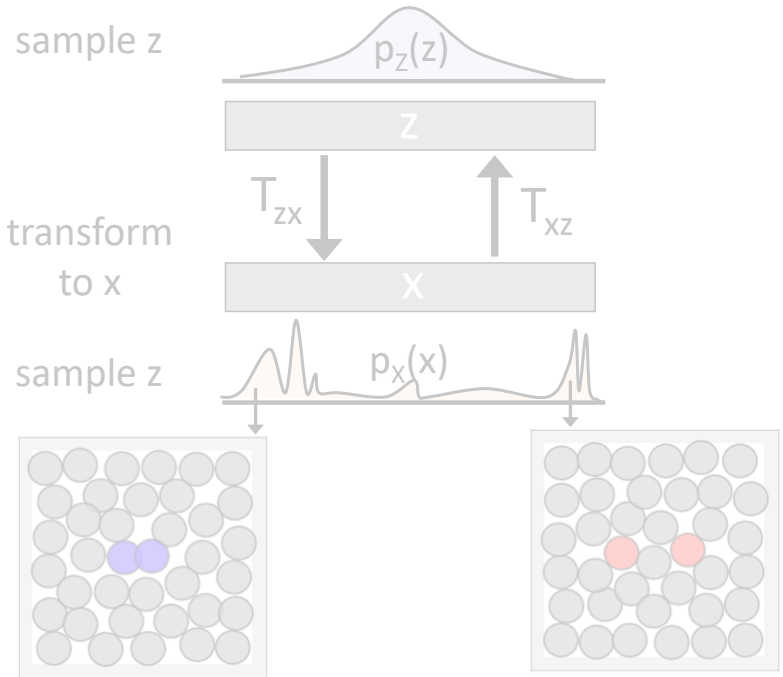
Adaptive MCMC

Aim: Learn proposal scheme P
to achieve efficient MCMC



Latent MC

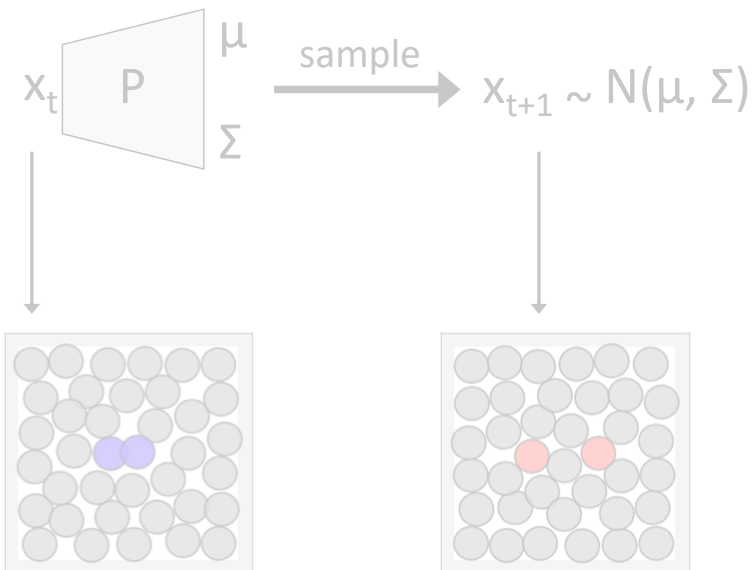
Aim: Learn transformation Txz of data to latent
space such that direct MC is efficient



Neural Latent MC

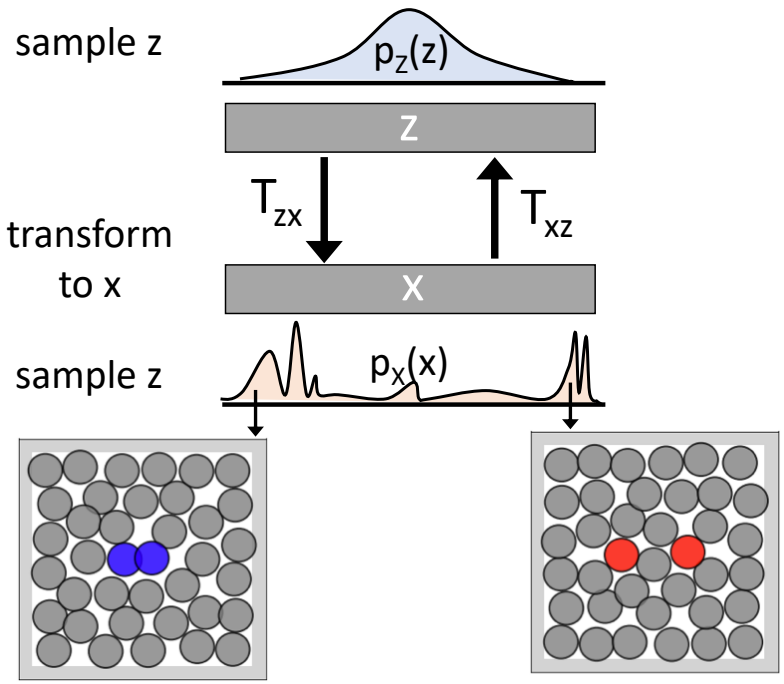
Adaptive MCMC

Aim: Learn proposal scheme P
to achieve efficient MCMC



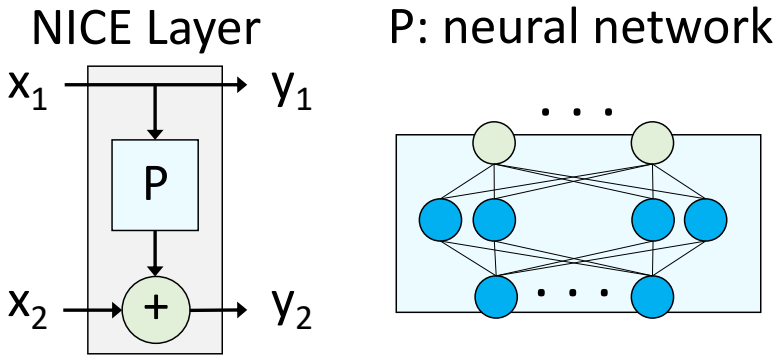
Latent MC

Aim: Learn transformation Txz of data to latent space such that direct MC is efficient



NICE: Non-linear Independent Components Estimation

Dinh, Krueger, Bengio, ICLR 2015



- Forward transformation:

$$y_1 = x_1$$
$$y_2 = x_2 + P(x_1)$$

- Inverse transformation:

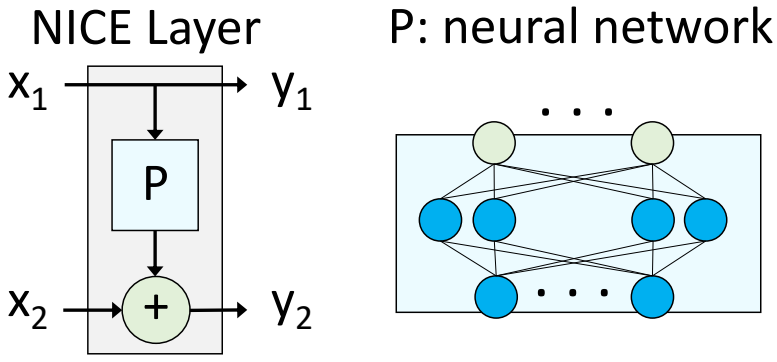
$$x_1 = y_1$$
$$x_2 = y_2 - P(y_1)$$

- Bijective transform with unit Jacobian:

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \det\left(\frac{\partial y}{\partial x}\right) = 1.$$

NICE: Non-linear Independent Components Estimation

Dinh, Krueger, Bengio, ICLR 2015



- Forward transformation:

$$y_1 = x_1$$
$$y_2 = x_2 + P(x_1)$$

- Inverse transformation:

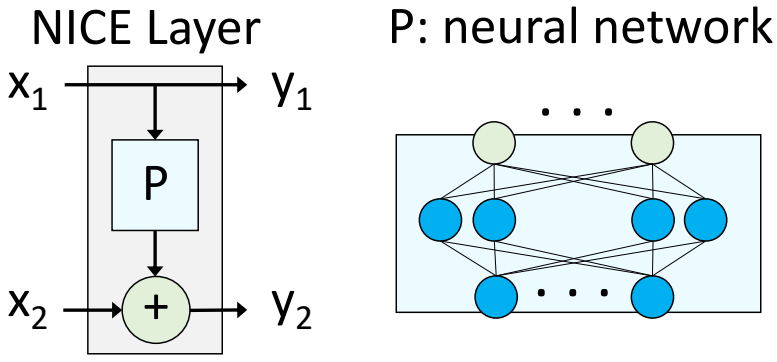
$$x_1 = y_1$$
$$x_2 = y_2 - P(y_1)$$

- Bijective transform with unit Jacobian:

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \det\left(\frac{\partial y}{\partial x}\right) = 1.$$

NICE: Non-linear Independent Components Estimation

Dinh, Krueger, Bengio, ICLR 2015



- Forward transformation:

$$y_1 = x_1$$
$$y_2 = x_2 + P(x_1)$$

- Inverse transformation:

$$x_1 = y_1$$
$$x_2 = y_2 - P(y_1)$$

- Bijective transform with unit Jacobian:

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \det\left(\frac{\partial y}{\partial x}\right) = 1.$$

Probability distributions

- Define an invertible and continuous differentiable **variable transformation**

$$\mathbf{z} = T_{xz}(\mathbf{x}) \qquad \mathbf{x} = T_{zx}(\mathbf{z}) = T_{xz}^{-1}(\mathbf{z})$$

- Probability density transforms as:

$$p_X(\mathbf{x}) = \left| \det \left(\frac{\partial T_{xz}}{\partial \mathbf{x}} \right) \right| p_Z(T_{xz}(\mathbf{z}))$$

- With unit Jacobian:

$$p_X(\mathbf{x}) = p_Z(T_{xz}(\mathbf{x}))$$

Key Idea

Given a **complicated distribution** $p_X(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$
learn $\mathbf{z} = T_{xz}(\mathbf{x})$ such that $p_Z(\mathbf{z}) = p_Z(T_{xz}(\mathbf{x}))$ is **simple**, e.g.:

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

Probability distributions

- Define an invertible and continuous differentiable **variable transformation**

$$\mathbf{z} = T_{xz}(\mathbf{x}) \qquad \mathbf{x} = T_{zx}(\mathbf{z}) = T_{xz}^{-1}(\mathbf{z})$$

- Probability density transforms as:

$$p_X(\mathbf{x}) = \left| \det \left(\frac{\partial T_{xz}}{\partial \mathbf{x}} \right) \right| p_Z(T_{xz}(\mathbf{z}))$$

- With unit Jacobian:

$$p_X(\mathbf{x}) = p_Z(T_{xz}(\mathbf{x}))$$

Key Idea

Given a **complicated distribution** $p_X(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$
learn $\mathbf{z} = T_{xz}(\mathbf{x})$ such that $p_Z(\mathbf{z}) = p_Z(T_{xz}(\mathbf{x}))$ is **simple**, e.g.:

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

Probability distributions

- Define an invertible and continuous differentiable **variable transformation**
$$\mathbf{z} = T_{xz}(\mathbf{x}) \qquad \mathbf{x} = T_{zx}(\mathbf{z}) = T_{xz}^{-1}(\mathbf{z})$$
- Probability density transforms as:
$$p_X(\mathbf{x}) = \left|\det\left(\frac{\partial T_{xz}}{\partial \mathbf{x}}\right)\right| p_Z(T_{xz}(\mathbf{z}))$$
- With unit Jacobian:
$$p_X(\mathbf{x}) = p_Z(T_{xz}(\mathbf{x}))$$

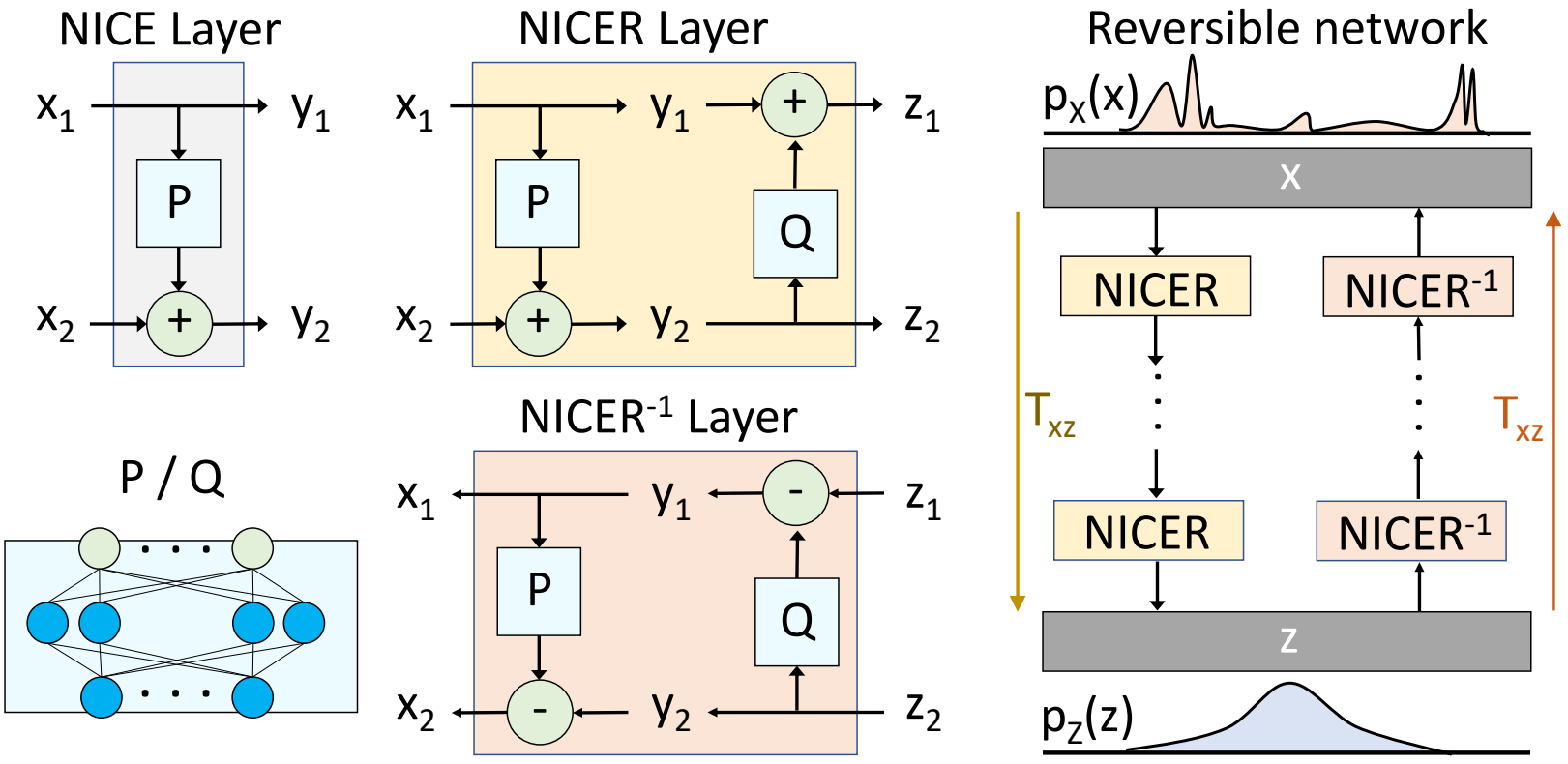
Key Idea

Given a **complicated distribution** $p_X(\mathbf{x}) \propto \mathrm{e}^{-u(\mathbf{x})}$
learn $\mathbf{z} = T_{xz}(\mathbf{x})$ such that $p_Z(\mathbf{z}) = p_Z(T_{xz}(\mathbf{x}))$ is **simple**, e.g.:

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

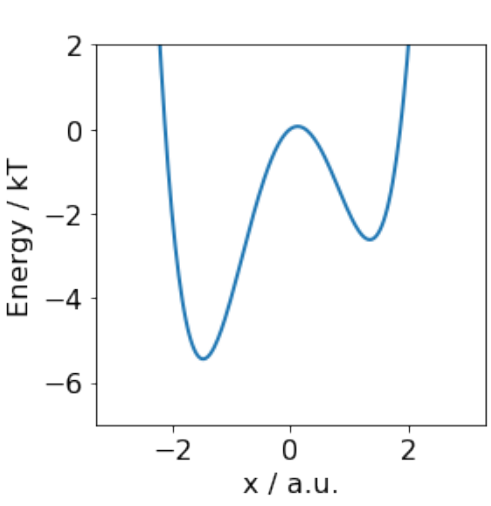
Neural Latent MC

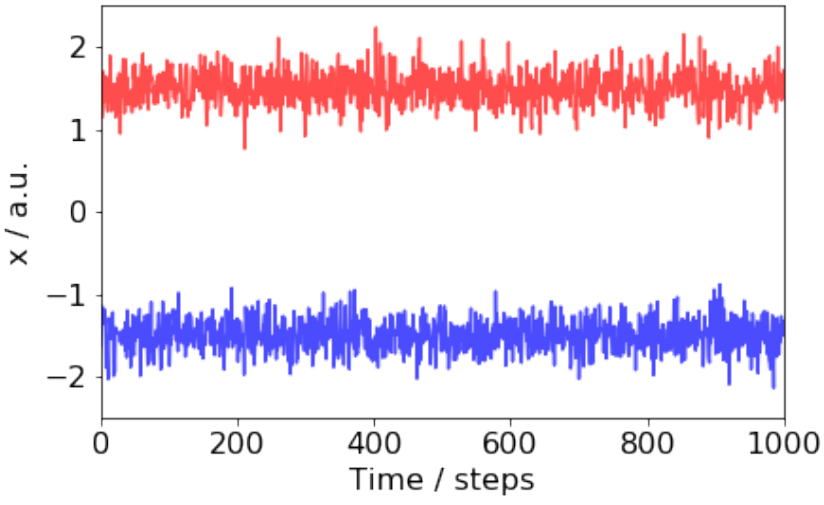
Motivation

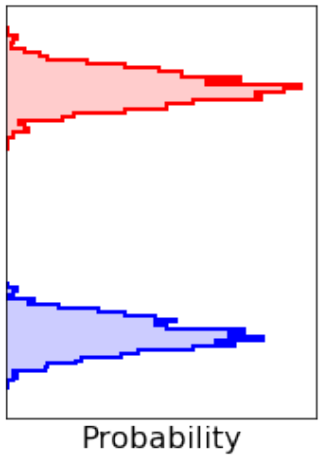


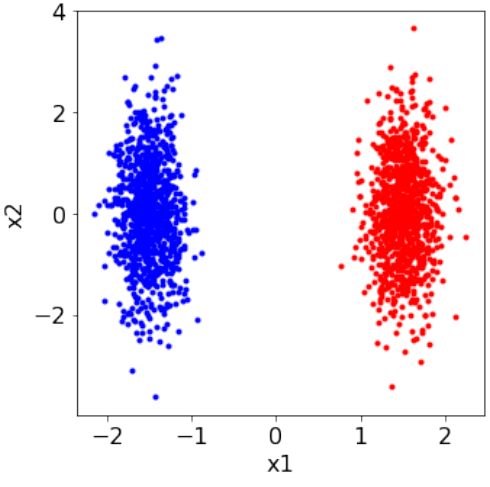
Neural Latent MC - 1D Example

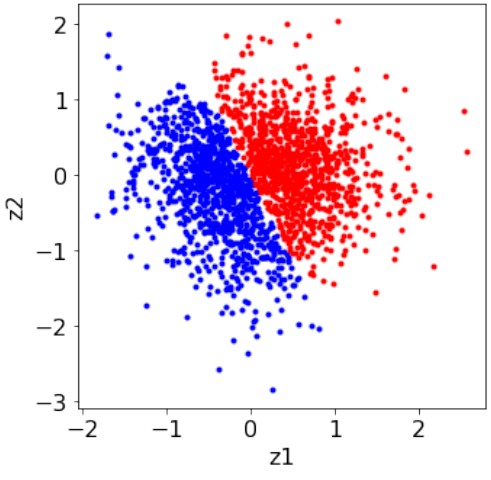
Motivation

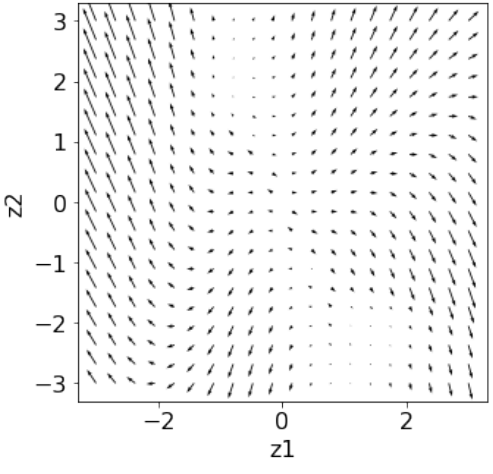






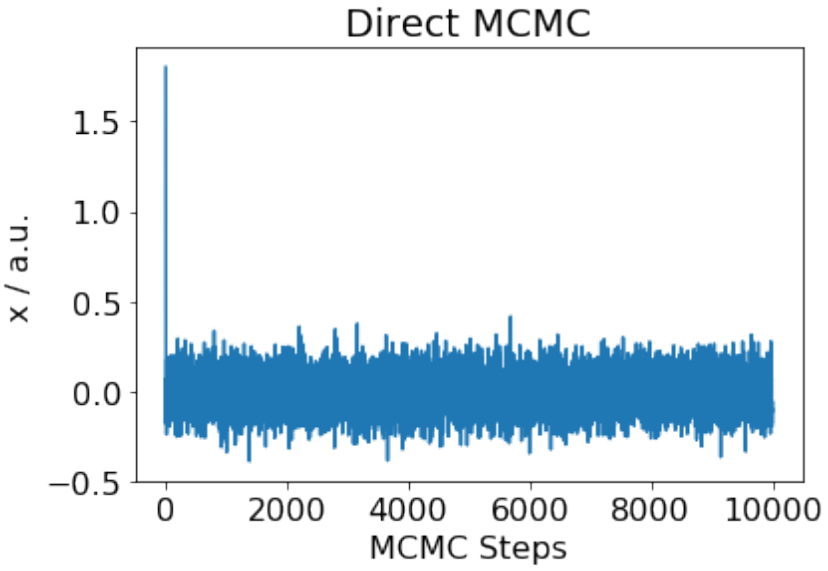


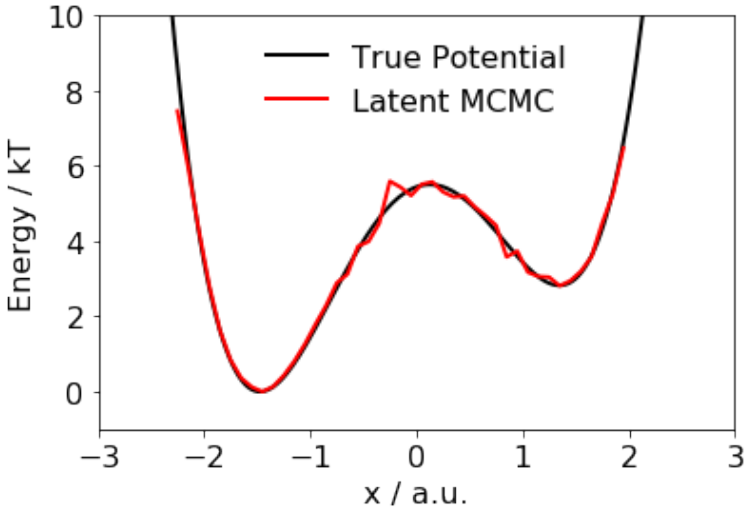


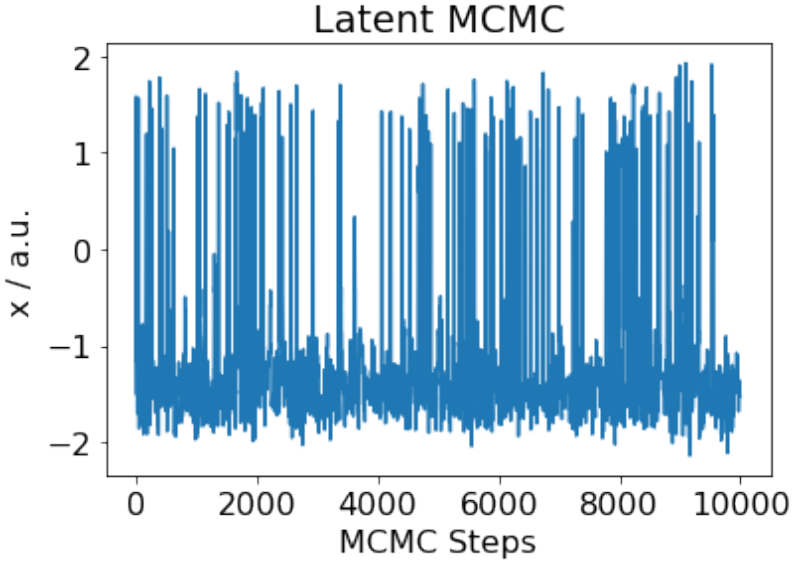


Neural Latent MCMC - 1D Example

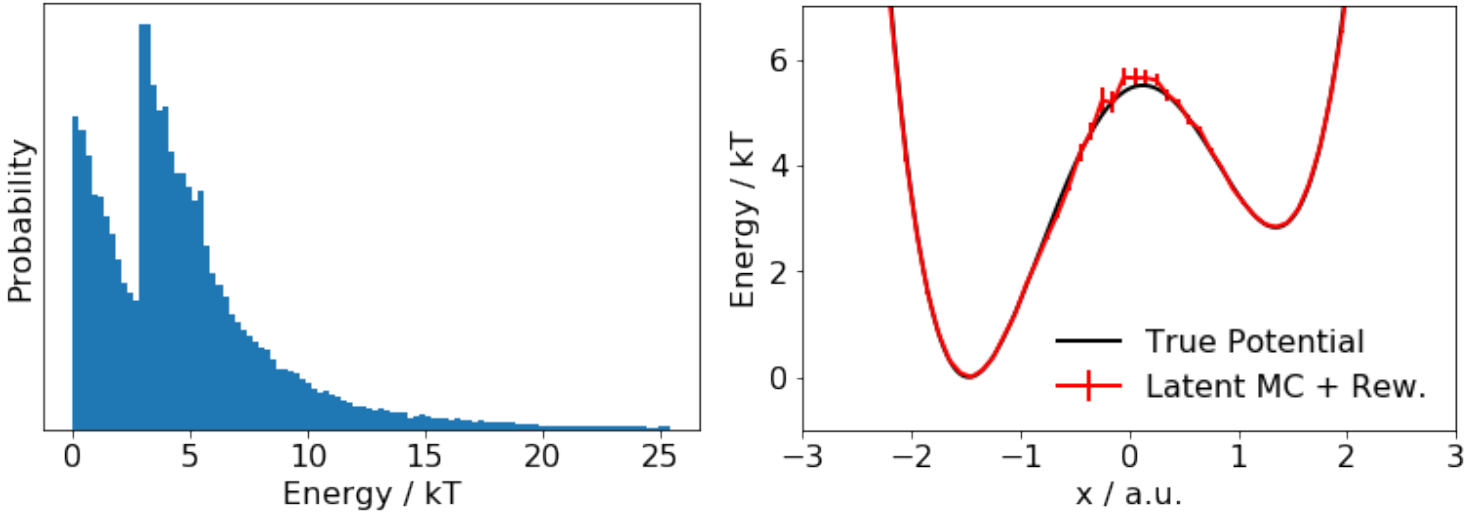
Motivation







Neural Latent MC - 1D Example



- **Sample prior**:

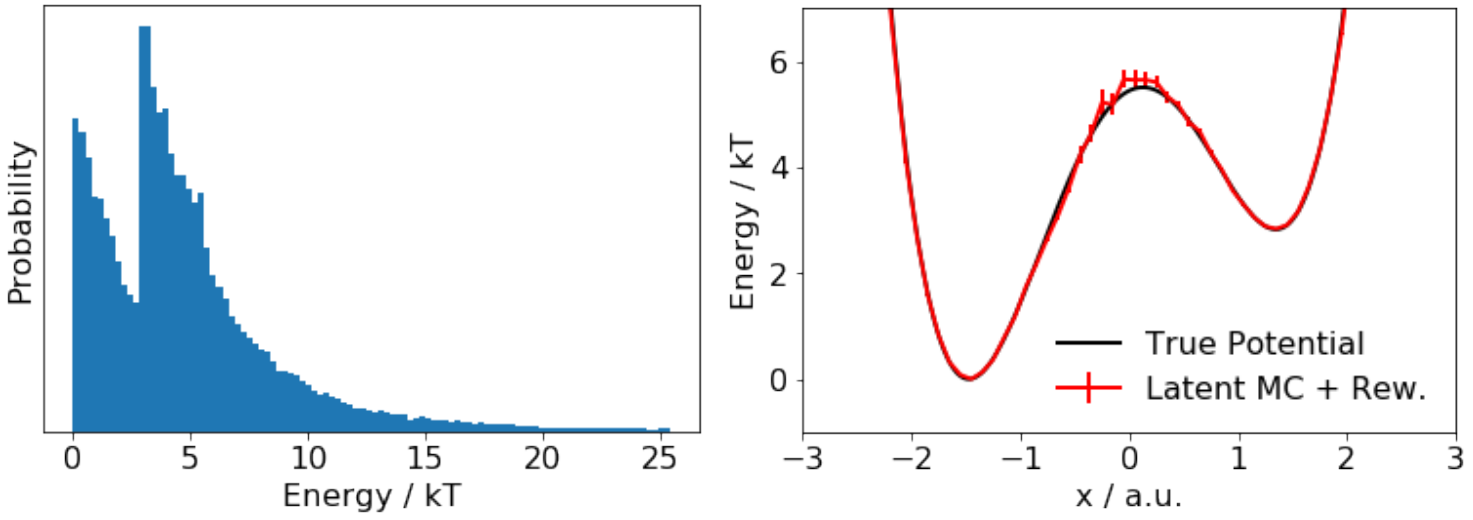
$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \propto \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{z}\|^2\right)$$

  This corresponds to a prior energy of $u_Z(z) = \frac{1}{2\sigma^2} \|\mathbf{z}\|^2$

- **Reweighting**:

$$w_X(\mathbf{x}) = \frac{p_X(T_{zx}(\mathbf{z}))}{p_Z(\mathbf{z})} = \mathrm{e}^{u_Z(\mathbf{z}) - u_X(T_{zx}(\mathbf{z}))}.$$

Neural Latent MC - 1D Example



- **Sample prior**:

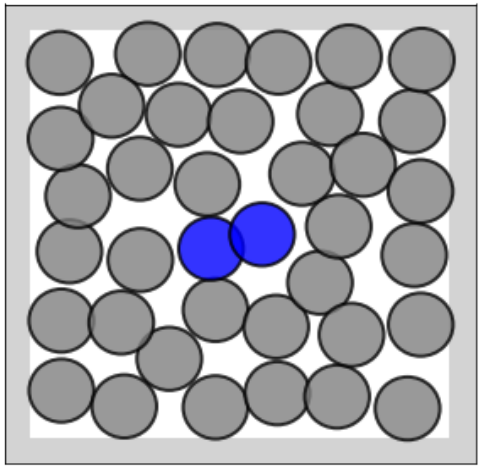
$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \propto \exp\left(-\frac{1}{2\sigma^2} \|\mathbf{z}\|^2\right)$$

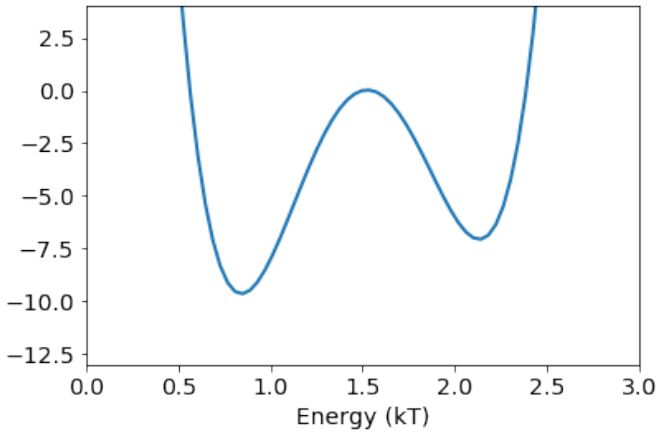
  This corresponds to a prior energy of $u_Z(z) = \frac{1}{2\sigma^2} \|\mathbf{z}\|^2$

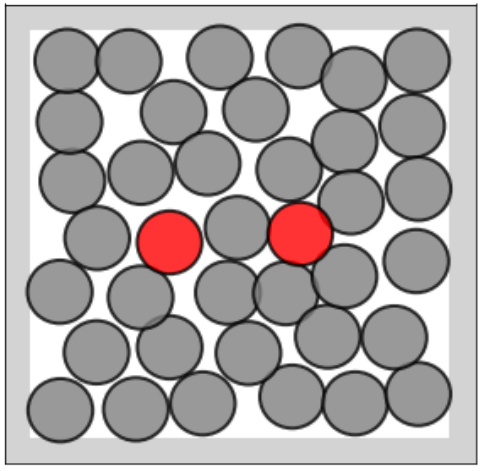
- **Reweighting**:

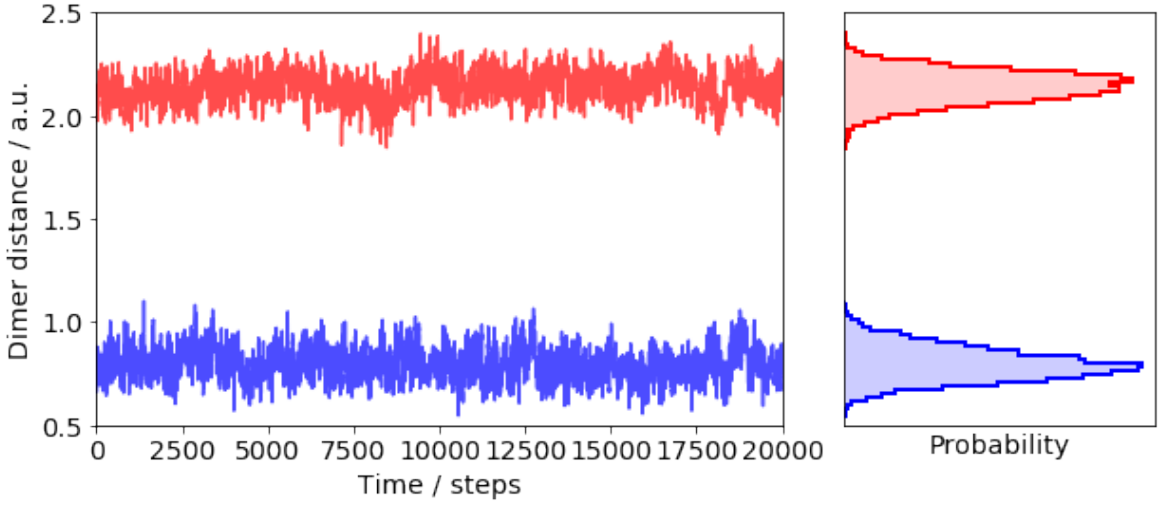
$$w_X(\mathbf{x}) = \frac{p_X(T_{zx}(\mathbf{z}))}{p_Z(\mathbf{z})} = \mathrm{e}^{u_Z(\mathbf{z}) - u_X(T_{zx}(\mathbf{z}))}.$$

Neural Latent MC - Particle Example







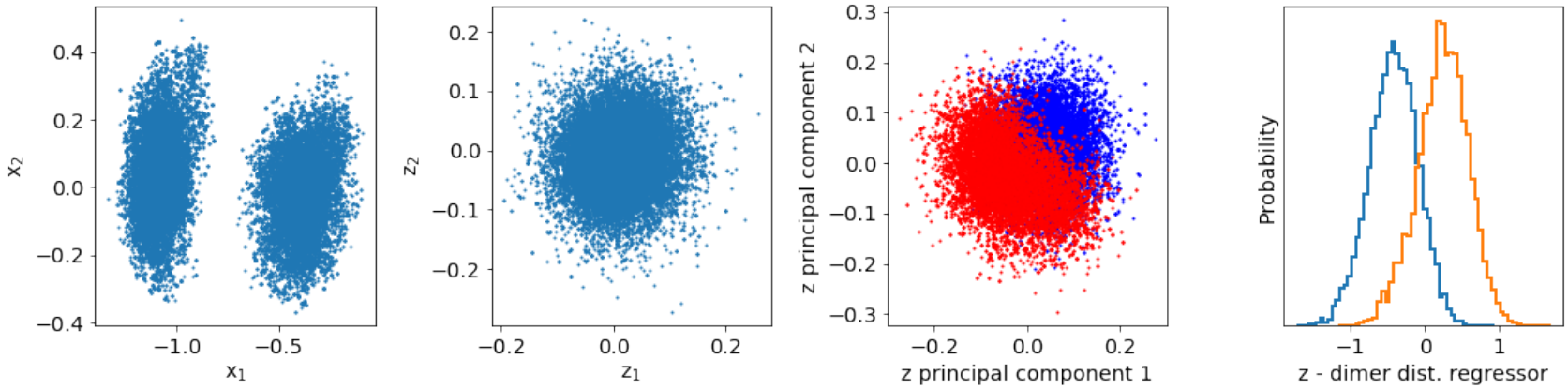


Neural Latent MC - Particle Example

- **Input**: Simulation data $\mathbf{X} = (\mathbf{x}_1^{\text{closed}}, \ldots, \mathbf{x}_{20000}^{\text{closed}}, \mathbf{x}_1^{\text{open}}, \ldots, \mathbf{x}_{20000}^{\text{open}})$
- **Train** $\mathbf{z}_i = T_{xy}(\mathbf{x}_i, \theta)$ until convergence (3000 epochs) using:

$$\min_\theta \underbrace{\left\| \frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu}) \right\|^2}_{\text{force mean 0}} + \underbrace{\left\| \mathbf{I} - \frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu})(\mathbf{z}_i - \hat{\mu})^\top \right\|_F^2}_{\text{force covariance I}} + \underbrace{\frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu})^2}_{\text{make compact}}$$

where $\hat{\mu} = B^{-1} \sum_j \mathbf{z}_j$, B batchsize, i, j sample indices in batch.

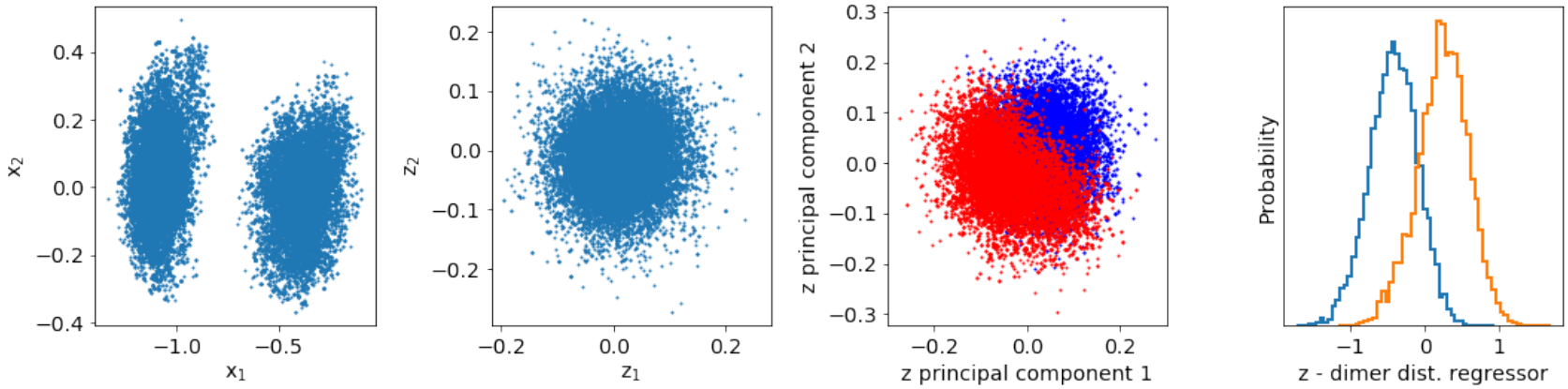


Neural Latent MC - Particle Example

- **Input**: Simulation data $\mathbf{X} = (\mathbf{x}_1^{\text{closed}}, \ldots, \mathbf{x}_{20000}^{\text{closed}}, \mathbf{x}_1^{\text{open}}, \ldots, \mathbf{x}_{20000}^{\text{open}})$
- **Train** $\mathbf{z}_i = T_{xy}(\mathbf{x}_i, \theta)$ until convergence (3000 epochs) using:

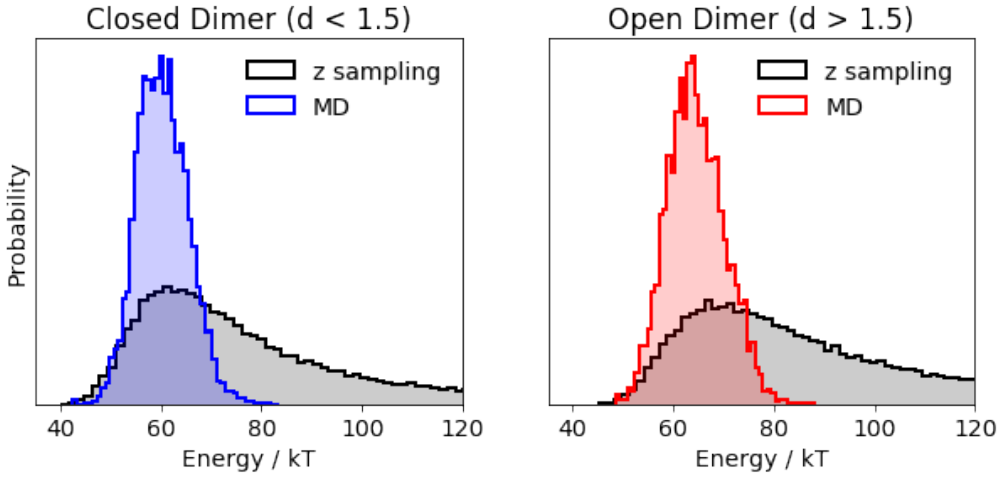
$$\min_\theta \underbrace{\left\| \frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu}) \right\|^2}_{\text{force mean 0}} + \underbrace{\left\| \mathbf{I} - \frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu})(\mathbf{z}_i - \hat{\mu})^\top \right\|_F^2}_{\text{force covariance I}} + \underbrace{\frac{1}{B} \sum_i (\mathbf{z}_i - \hat{\mu})^2}_{\text{make compact}}$$

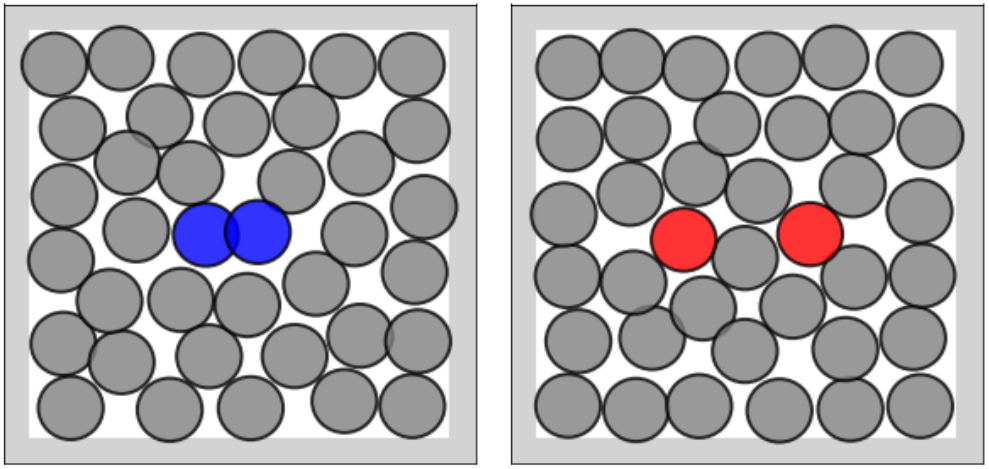
where $\hat{\mu} = B^{-1} \sum_j \mathbf{z}_j$, B batchsize, i, j sample indices in batch.



Neural Latent MC - Particle Example

1. Sample prior:
$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}) \propto \exp\left(\frac{1}{2\sigma^2} \|\mathbf{z}\|^2\right)$$
2. Evaluate posterior energy $u(\mathbf{x}[\mathbf{z}])$





Comparison with umbrella sampling