Neural Mode Jump Monte Carlo

Luigi Sbailò

Coffee talk
17/02/2020, FHI-Berlin

Markov chain Monte Carlo

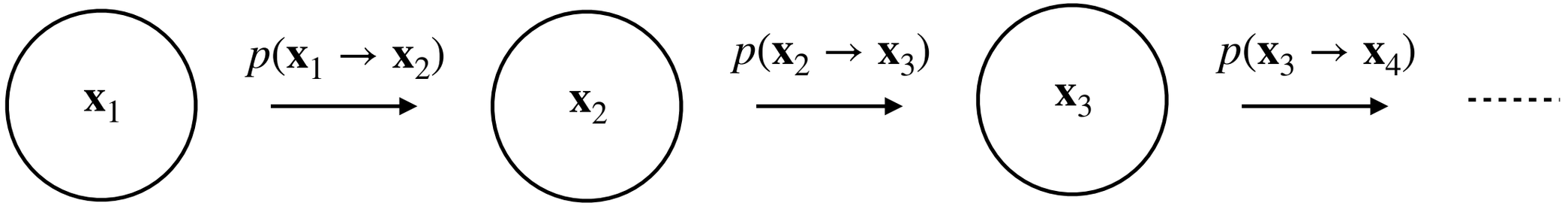
$$\langle A \rangle = \int_{\Omega} d\mathbf{x}\, \pi(\mathbf{x})\, A(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^{n} A(\mathbf{x}_i)$$

x sampled from $\pi(\mathbf{x})$

Markov chain Monte Carlo

$$\langle A \rangle = \int_{\Omega} d\mathbf{x}\, \pi(\mathbf{x})\, A(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^{n} A(\mathbf{x}_i)$$

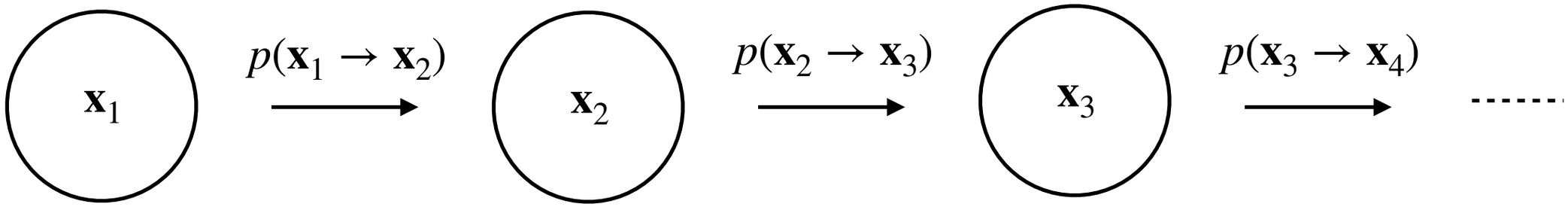
x sampled from $\pi(\mathbf{x})$



Markov chain Monte Carlo

$$\langle A \rangle = \int_{\Omega} d\mathbf{x}\, \pi(\mathbf{x})\, A(\mathbf{x}) = \frac{1}{n} \sum_{i=0}^{n} A(\mathbf{x}_i)$$

x sampled from $\pi(\mathbf{x})$



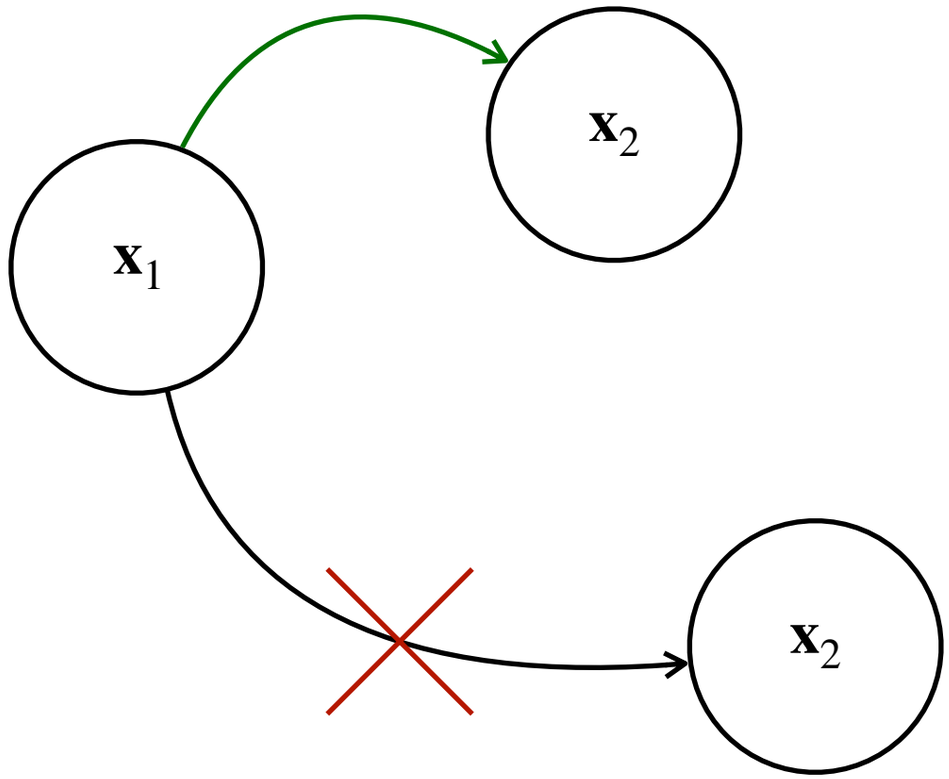
$$\pi(\mathbf{x}_1)\, p(\mathbf{x}_1 \to \mathbf{x}_2) = \pi(\mathbf{x}_2)\, p(\mathbf{x}_2 \to \mathbf{x}_1)$$

Detailed balance + **Ergodicity** → **Asymptotically, the Markov chain samples the target distribution $\pi(\mathbf{x})$**

Metropolis-Hastings - local moves

$$\pi(\mathbf{x}_1)\, p(\mathbf{x}_1 \to \mathbf{x}_2) = \pi(\mathbf{x}_2)\, p(\mathbf{x}_2 \to \mathbf{x}_1)$$

$$p(\mathbf{x}_1 \to \mathbf{x}_2) = p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)\, p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)$$

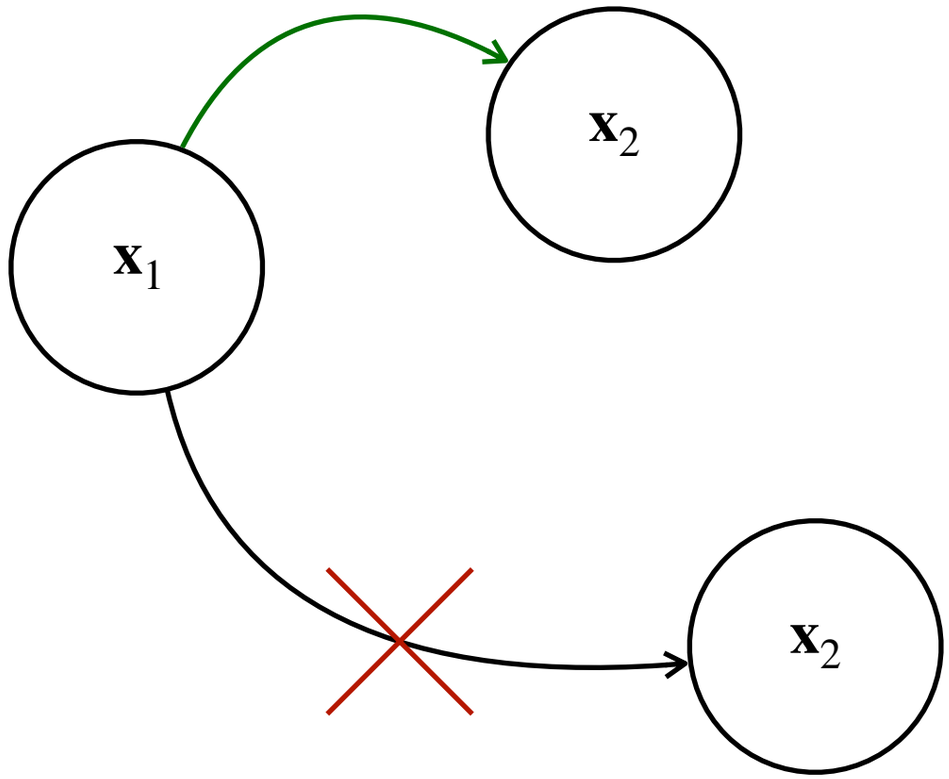


Metropolis-Hastings - local moves

$$\pi(\mathbf{x}_1)\, p(\mathbf{x}_1 \to \mathbf{x}_2) = \pi(\mathbf{x}_2)\, p(\mathbf{x}_2 \to \mathbf{x}_1)$$

$$p(\mathbf{x}_1 \to \mathbf{x}_2) = p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)\, p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)$$

$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, \frac{\pi(\mathbf{x}_2)\, p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1)}{\pi(\mathbf{x}_1)\, p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)}\right)$$

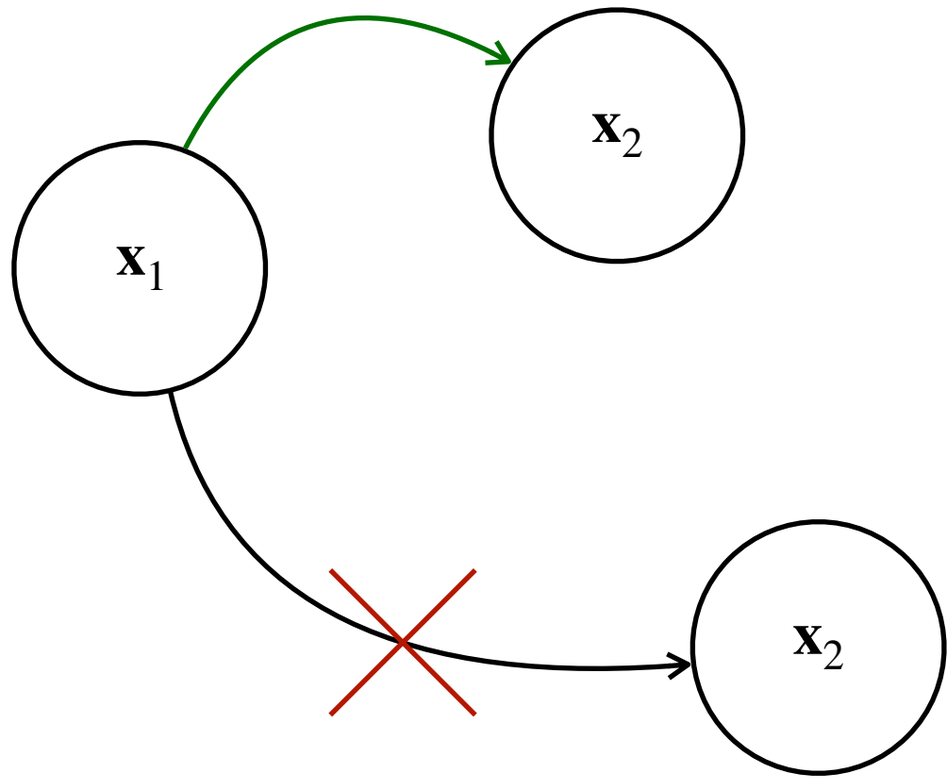


Metropolis-Hastings - local moves

$$\pi(\mathbf{x}_1)\, p(\mathbf{x}_1 \to \mathbf{x}_2) = \pi(\mathbf{x}_2)\, p(\mathbf{x}_2 \to \mathbf{x}_1)$$

$$p(\mathbf{x}_1 \to \mathbf{x}_2) = p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)\, p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)$$

$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, \frac{\pi(\mathbf{x}_2)\, p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1)}{\pi(\mathbf{x}_1)\, p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)}\right)$$



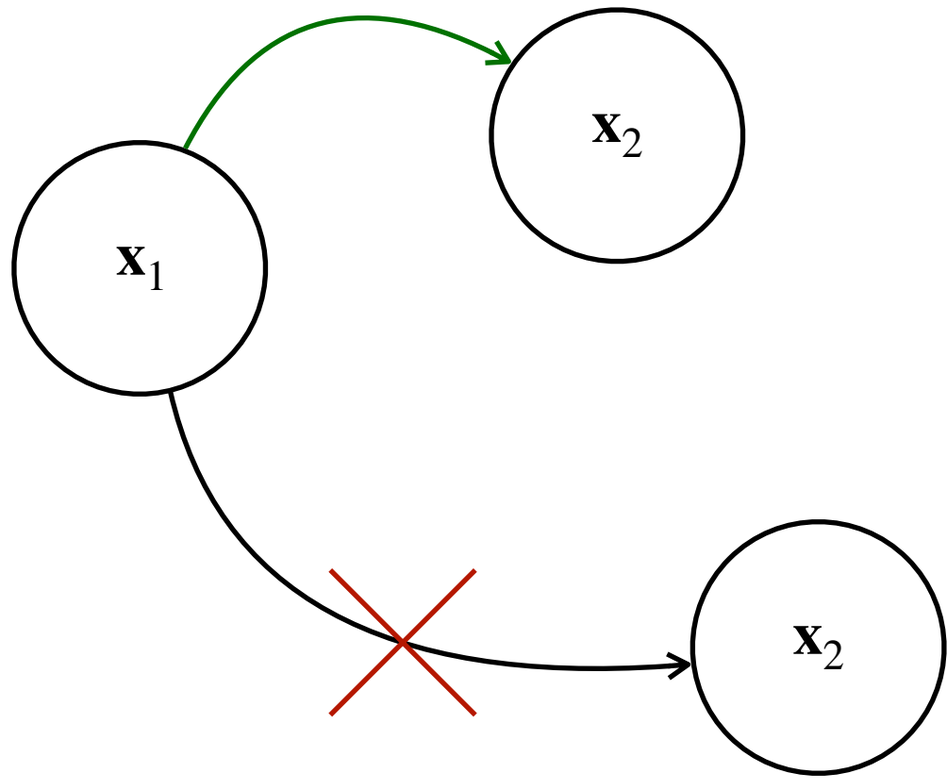
$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, e^{-\beta\left(V(\mathbf{x}_2) - V(\mathbf{x}_1)\right)} \frac{p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1)}{p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)}\right)$$

Metropolis-Hastings - local moves

$$\pi(\mathbf{x}_1)\, p(\mathbf{x}_1 \to \mathbf{x}_2) = \pi(\mathbf{x}_2)\, p(\mathbf{x}_2 \to \mathbf{x}_1)$$

$$p(\mathbf{x}_1 \to \mathbf{x}_2) = p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)\, p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2)$$

$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, \frac{\pi(\mathbf{x}_2)\, p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1)}{\pi(\mathbf{x}_1)\, p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)}\right)$$

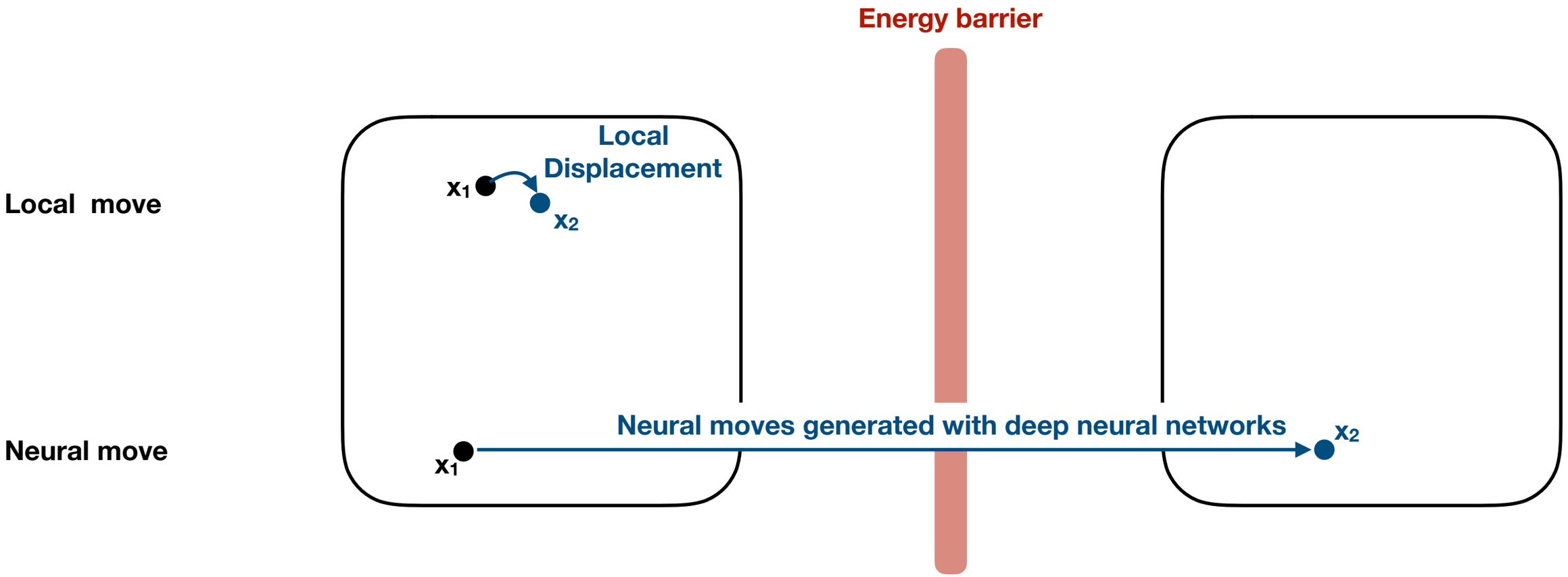


$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, e^{-\beta\left(V(\mathbf{x}_2) - V(\mathbf{x}_1)\right)} \frac{p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1)}{p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)}\right)$$

If moves are generated with random displacements:
$p_{prop}(\mathbf{x}_2 \to \mathbf{x}_1) = p_{prop}(\mathbf{x}_1 \to \mathbf{x}_2)$

$$p_{acc}(\mathbf{x}_1 \to \mathbf{x}_2) = \min\left(1, e^{-\beta\left(V(\mathbf{x}_2) - V(\mathbf{x}_1)\right)}\right)$$

Neural Mode Jump Monte Carlo



1. Select Kernel.
2. Generate trial configuration.
3. Accept or reject trial configuration.

$$p(\mathbf{x} \to \mathbf{y}) = p_{acc}(\mathbf{x} \to \mathbf{y}) \sum_i p^i(\mathbf{x}) p^i(\mathbf{x}) p^i_{prop}(\mathbf{x} \to \mathbf{x})$$

Local move

$$p_{acc}(\mathbf{x} \to \mathbf{y}) = \min\left\{1, e^{-\beta\left(V(\mathbf{y}) - V(\mathbf{x})\right)}\right\}$$

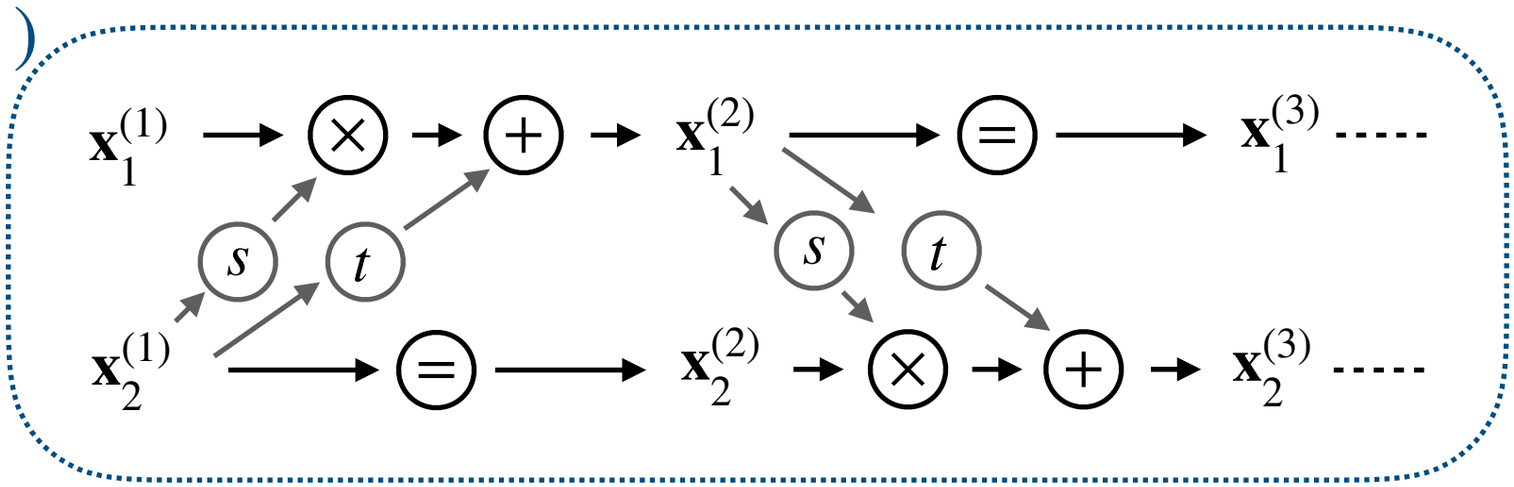
Neural move

$$p_{acc}(\mathbf{x} \to \mathbf{y}) = \min\left\{1, \frac{p_i(\mathbf{y})}{p_j(\mathbf{x})} e^{-\beta\left(V(\mathbf{y}) - V(\mathbf{x})\right)} \det|J(\mu(\mathbf{x}))|\right\}$$

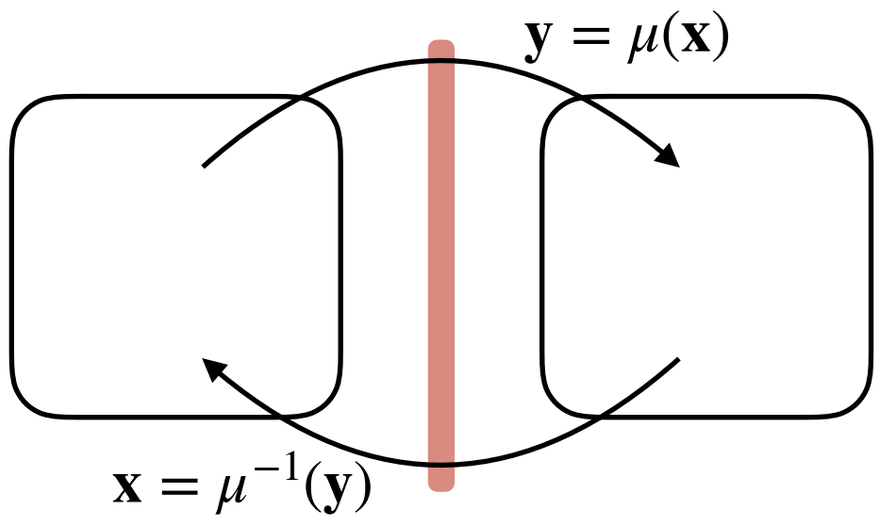
Sbailò L, Dibak M, Noé F (2019) arXiv (submitted)

DNNs generate uncorrelated moves

The input vector is divided into two disjoint sets of coordinates $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, then only simple operations are performed on the defined sets .
The resulting neural network is invertible:
We can construct $\mu^{-1}(\,\cdot\,)$ from $\mu(\,\cdot\,)$.



- Training set is collected with local MCMC simulations performed in each metastable state.
- Each batch contains configurations from both metastable states and training is performed simultaneously for both directions.
- Training is supervised: one reference configuration from the target metastable state is taken as label.
- The free energy difference is minimised during training to maximise acceptance probability.



$$p_{acc}(\mathbf{x} \to \mathbf{y}) = \min\left\{1, \frac{p_i(\mathbf{y})}{p_j(\mathbf{x})} e^{-\beta\left(V(\mathbf{y}) - V(\mathbf{x})\right)} \det\left|J(\mu(\mathbf{x}))\right|\right\}$$

Cost function: $C = C_{sup} + \gamma C_{acc}$

$$C_{sup} = \mu(\mathbf{x}) - \bar{\mathbf{y}}$$

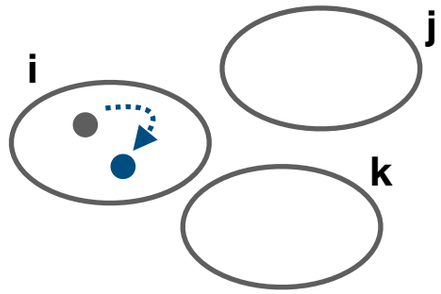
$$C_{acc} = -\beta\left(V(\mu(\mathbf{x})) - V(\mathbf{x})\right) + \log\left(\det J_\mu(\mathbf{x})\right)$$

Sbailò L, Dibak M, Noé F (2019) arXiv (submitted)

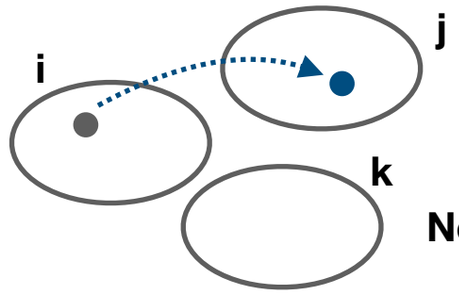
Algorithm outline

Select Kernel-i,j,k from initial state $\mathbf{x}$ in core-i with probability $p_{i,j,k}(\mathbf{x})$

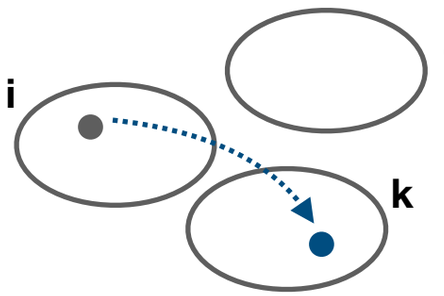
Local move



Neural move



Neural move



Sample state $\mathbf{y}$ from Gaussian extraction $\mathcal{N}(\mathbf{x}, \sigma^2)$

Generate state $\mathbf{y}$ with neural network $\mu_{ij}(\mathbf{x})$

Generate state $\mathbf{y}$ with neural network $\mu_{ik}(\mathbf{x})$

Accept $\mathbf{y}$ with probability

$$\min\left\{1, \frac{\pi(\mathbf{y})}{\pi(\mathbf{x})}\right\}$$

Accept $\mathbf{y}$ with probability

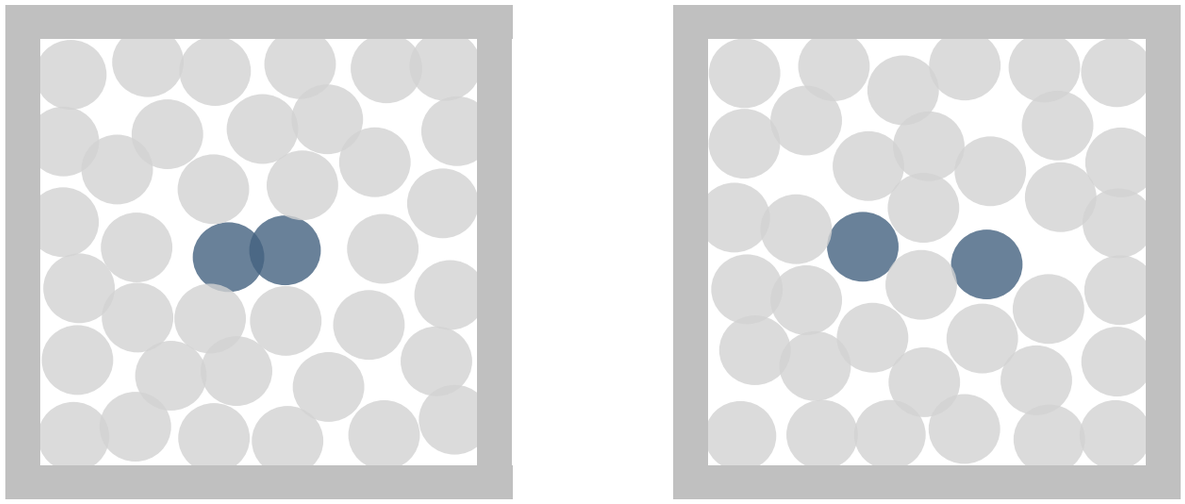
$$\min\left\{1, \frac{p_{ij}(\mathbf{y})\pi(\mathbf{y})}{p_{ji}(\mathbf{x})\pi(\mathbf{x})} \det|J(\mu_{ij}(\mathbf{x}))|\right\}$$

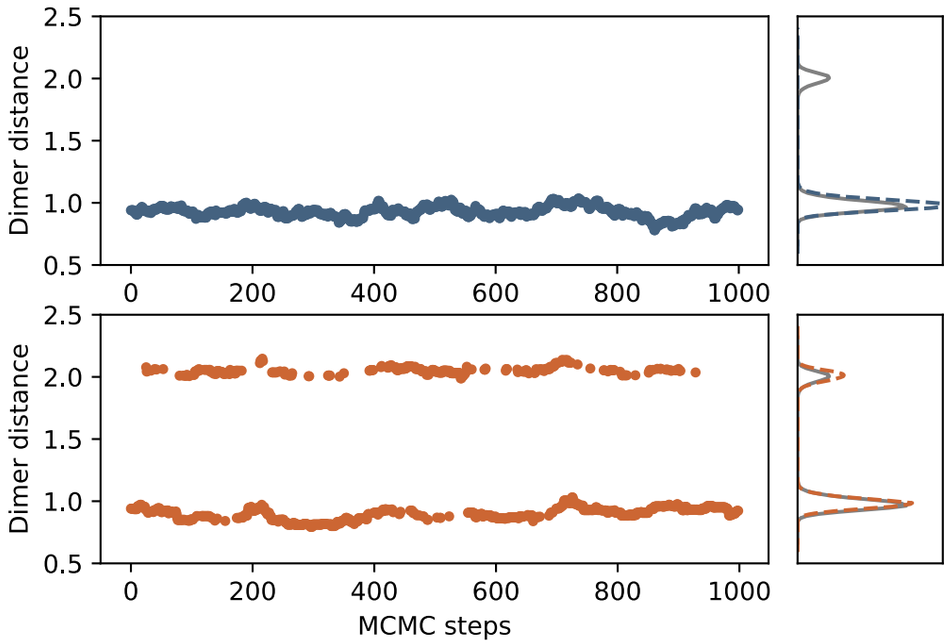
Accept $\mathbf{y}$ with probability

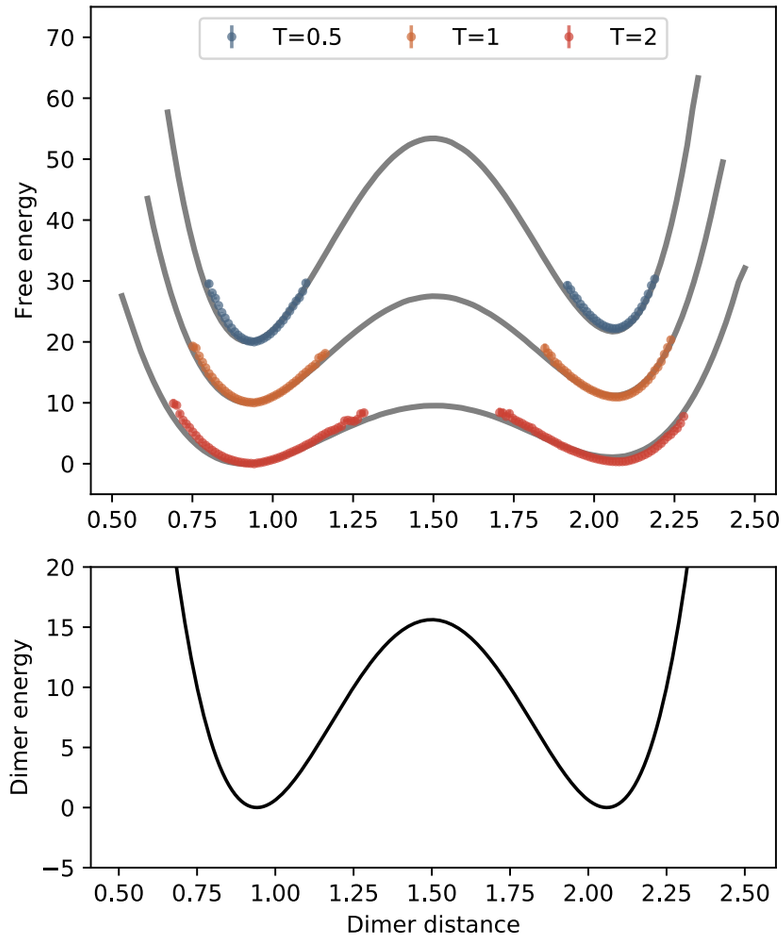
$$\min\left\{1, \frac{p_{ik}(\mathbf{y})\pi(\mathbf{y})}{p_{ki}(\mathbf{x})\pi(\mathbf{x})} \det|J(\mu_{ik}(\mathbf{x}))|\right\}$$

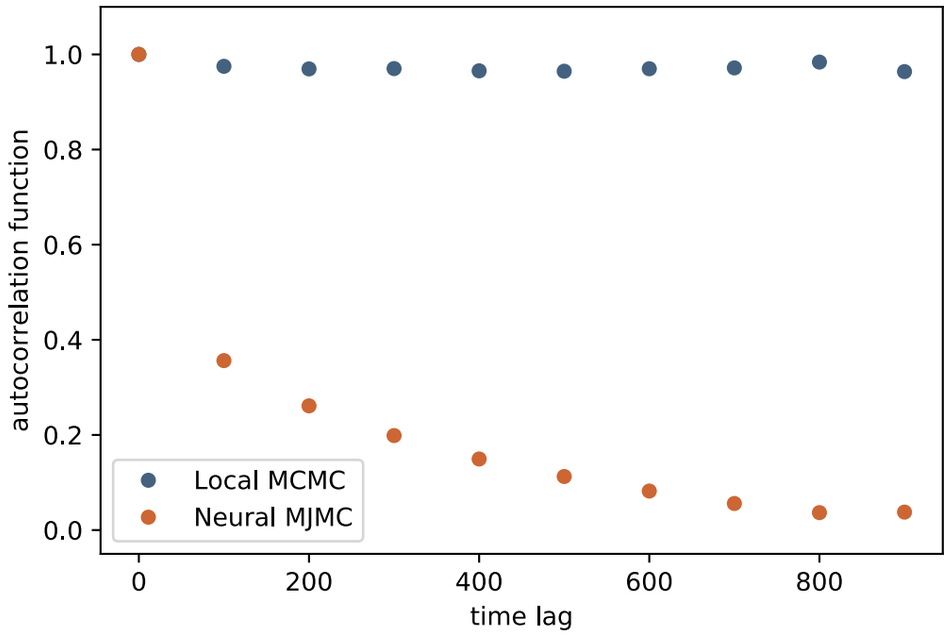
If accepted: add $\mathbf{y}$ to the Markov chain, and move the system to $\mathbf{y}$;
Otherwise: add $\mathbf{x}$ to the Markov chain, and remain in $\mathbf{x}$.

Results




Dimer distance
2.5
2.0
1.5
1.0
0.5
0
200
400
600
800
1000
MCMC steps


T=0.5
T=1
T=2
Free energy
70
60
50
40
30
20
10
0
0.50
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50
Dimer energy
20
15
10
5
0
−5
Dimer distance


autocorrelation function
1.0
0.8
0.6
0.4
0.2
0.0
Local MCMC
Neural MJMC
0
200
400
600
800
time lag

Conclusion

- Markov chains Monte Carlo asymptotically sample the equilibrium distribution if ergodic and the condition of detailed balance is satisfied.

- Local Metropolis-Hastings algorithm selects configurations with local random displacements, selected states are then accepted or rejected to enforce detailed balance.

- Local selection is efficient at sampling local conformations of the system, but transitions between different modes of the system are arduous to sample.

- Neural Mode Jump Monte Carlo is a combined scheme that alternates local moves and neural moves, where neural moves are generated with neural networks and produce direct transitions between different modes of the system. Markov chains are shown to be ergodic and satisfy the condition of detailed balance. The method is tested on benchmark systems.