NEURAL TRANSFORMATIONS FOR EFFICIENT TOPOLOGICAL MIXING

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose a generalized version of the L2HMC algorithm (Lévy et al., 2018), and evaluate its ability to sample from different topologies in a two-dimensional lattice gauge theory. In particular, we demonstrate that our model is able to successfully mix between modes of different topology, significantly reducing the computational cost required to generate independent gauge configurations.

1 MAIN CONTRIBUTIONS

1. We propose a generalized version of the L2HMC algorithm that uses different networks (with different step sizes) for each distinct leapfrog step. We represent this generalization by carrying a discrete index $k = 0, 1, \ldots, N_{\text{LF}}$ through the augmented leapfrog equations, (Equation 2, Equation 3) indicating that these functions are allowed to vary.
2. We propose a modified loss function, defined in terms of the topological charge metric $\delta_{\mathcal{Q}_{\mathbb{R}}}(\xi', \xi)$,
$$\mathcal{L}_\theta[\xi', \xi, A(\xi'|\xi)] = \frac{-A(\xi'|\xi) \cdot \delta_{\mathcal{Q}_{\mathbb{R}}}(\xi', \xi)}{a^2}, \tag{1}$$
where $\delta_{\mathcal{Q}_{\mathbb{R}}}(\xi', \xi) \equiv (\mathcal{Q}'_{\mathbb{R}} - \mathcal{Q}_{\mathbb{R}})^2$, and $\mathcal{Q}_{\mathbb{R}} \in \mathbb{R}$ is the *real-valued topological charge*, defined in Section 5. We evaluate this loss at the end of each trajectory during training, and use the gradient information to update the weights θ parameterizing the auxiliary functions s_i^k, t_i^k, q_i^k, $i = x, v$ introduced in the augmented leapfrog updates in Section 3.1.
3. We apply the proposed method to a $1+1$-dimensional $U(1)$ lattice gauge theory defined on a $N_x \times N_t$ lattice with periodic boundary conditions, and report a significant reduction in the computational cost, as measured by the *integrated autocorrelation time* of the topological charge $\tau_{\text{int}}^{\mathcal{Q}}$. We compare our results to traditional HMC across a variety of trajectory lengths and inverse coupling constants β, and show that our trained model consistently outperforms traditional HMC (see Figure 2b).

2 RELATED WORK:

Recently, there has been a significant interest in applying probabilistic programming techniques with autodifferentiation capabilities to develop more efficient simulations. [not sure what this means: probabilistic programming techniques] In particular, following the development of the RealNVP (Dinh et al., 2016) architecture, there has been an explosion of different proposals that aim to take advantage of the invertible network architecture. Because of the enormous computational burden faced by generative techniques in lattice gauge theory, there have been multiple works (Albergo et al., 2019; 2021; Favoni et al., 2020; Medvidovic et al., 2020; Neklyudov & Welling, 2020; Neklyudov et al., 2020; Li et al., 2020; Boyda et al., 2020; Kanwar et al., 2020; Toth et al., 2019; Hoffman et al., 2019; Wehenkel & Louppe, 2020; Pasarica & Gelman, 2010; Dinh et al., 2016; Tanaka & Tomiya, 2017; Schaefer et al., 2009; Cossu et al., 2018; Rezende et al., 2020), that look at developing more efficient sampling techniques specifically tailored for lattice gauge/field theories.∗ ∗ Terrible sentence

3 BACKGROUND

We provide a review of the generic Hamiltonian Monte Carlo (HMC) algorithm and setup some of the relevant notation in Section A.1.

3.1 GENERALIZING THE LEAPFROG INTEGRATOR: L2HMC

In (Lévy et al., 2018), the authors propose the L2HMC ("Learning to Hamiltonian Monte Carlo") algorithm, and demonstrate its ability to outperform traditional Hamiltonian Monte Carlo (HMC) on a variety of two-dimensional target distributions. For example, the trained L2HMC sampler is shown to be capable of exploring regions of phase space which are typically inaccessible with traditional HMC. Additionally, they show that the trained sampler is efficient at mixing between modes of a multi-modal target distribution, a feature which is highly desirable for MCMC simulations of lattice gauge theory. ∗

∗ combine w/ prev sentence

We denote a complete state by $\xi = (x, v, d)$ with target distribution $p(\xi) = p(x, v, d) = p(x) \cdot p(v) \cdot p(d)$. Here we've introduced a binary direction variable (distributed independently of both x and v) $d \sim \mathcal{U}(+, -)$ which is resampled following each Metropolis-Hastings accept/reject step, and can be interpreted as determining the "direction" (forward/backward) of our update. **The key modification of the L2HMC algorithm is the introduction of six auxiliary functions s_i, t_i, q_i for $i = x, v$ into the leapfrog updates, which are parameterized by weights θ in a neural network.**

For simplicity, we consider the forward $d = +1$ direction, and introduce the notation:

$$v'_k \equiv \Gamma^+_k(v_k; \zeta_{v_k}) = v_k \odot \exp\left(\tfrac{\varepsilon^k_v}{2} s^k_v(\zeta_{v_k})\right) - \tfrac{\varepsilon^k_v}{2}\left[\partial_x S(x_k) \odot \exp\left(\varepsilon^k_v q^k_v(\zeta_{v_k})\right) + t^k_v(\zeta_{v_k})\right], \quad (2)$$

$$x'_k \equiv \Lambda^+(x_k; \zeta_{x_k}) = x_k \odot \exp(\varepsilon^k_x s^k_x(\zeta_{x_k})) + \varepsilon^k_x\left[v'_k \odot \exp(\varepsilon^k_x q^k_x(\zeta_{x_k})) + t^k_x(\zeta_{x_k})\right] \quad (3)$$

where (1.) $\zeta_{v_k} = (x_k, \partial_x S(x_k))$, $\zeta_{x_k} = (x_k, v_k)$ denote the (x, v respectively) networks' inputs,[1], and (2.) we indicate the direction of the update by the superscript $\pm$ on $\Gamma^\pm_k, \Lambda^\pm_k$. Using this notation, we can write the complete leapfrog update (in the forward $d = +1$ direction)[2] as:

1. Half-step momentum update: $v'_k = \Gamma^+_k(v_k; \zeta_{v_k})$
2. Full-step half-position update:[3] $x'_k = \bar{m}^t \odot x_k + m^t \odot \Lambda^+_k(x_k; \zeta_{x_k})$
3. Full-step half-position update: $x''_k = \bar{m}^t \odot \Lambda^+_k(x'_k; \zeta_{x'_k}) + m^t \odot x'_k$
4. Half-step momentum update: $v''_k = \Gamma^+(v'_k; \zeta_{v'_k})$

Note that in order to keep our leapfrog update reversible, we've split the x update into two sub-updates by introducing a binary mask $\bar{m}^t = \mathbb{1} - m^t$ that updates half of the components of x sequentially, as shown (with the general transformation $\Lambda^\pm$) in Figure 1. ∗

∗ move splitx fig to appendix?

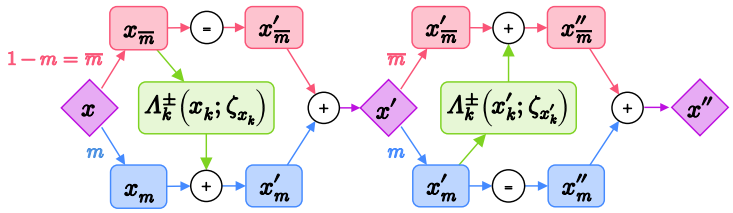


Figure 1: Illustration of the split x update. Here we include the general form of the transformation $\Lambda^\pm_k(x_k; \zeta_{x_k})$.

As in HMC, we form a complete trajectory by performing N_{LF} leapfrog steps in sequence, followed by a Metropolis-Hastings accept/reject step as described in Equation 8. However, unlike in the expression for HMC, we must take into account the overall Jacobian factor from the update $\xi \to \xi'$, which can be easily computed as $\left|\frac{\partial v'_k}{\partial v_k}\right| = \exp\left(\frac{1}{2}\varepsilon^k_v \cdot s^k_v(\zeta_{v_k})\right)$, $\left|\frac{\partial x'_k}{\partial x_k}\right| = \exp\left(\varepsilon^k_x s^k_x(\zeta_{x_k})\right)$. In

[1] Each of which is independent of the variable being updated.

[2] To obtain the expression for the reverse direction, we can invert each of the $\Gamma^- \equiv \left(\Gamma^+\right)^{-1}, \Lambda^- \equiv \left(\Lambda^+\right)^{-1}$ functions, and perform the updates in the opposite order.

[3] By this we mean we are performing a complete update on one-half of the indices of x (determined by $m^t \odot x$), followed by an analogous update of the complementary indices, $\bar{m}^t \odot x$.

order to perform the updates in the generalized leapfrog integrator, we need to evaluate each of the functions s, t, q. Without loss of generality[4], we temporarily ignore the discrete leapfrog index k, and restrict our attention to the s_x, t_x, q_x functions used in the x update, Equation 3. Each of the s_x, t_x, q_x functions takes as input $\zeta_x = (x, v)$, with $x \in \mathbb{R}^n$, $v \in \mathbb{R}^n$. The network first passes each component of the input (x, v) through a dense layer. These outputs are then summed together before being passed through an additional stack of d hidden layers, i.e.

$$z_d = \sigma(w_d^T z_{d-1} + b_{d-1}),\ z_{d-1} = \sigma(w_{d-1}^T z_{d-2} + b_{d-2}),\ \ldots,\ z_1 \equiv \sigma(w_x^T x + w_v^T v + b) \tag{4}$$

The network outputs s_x, t_x, q_x are then defined in terms of this final hidden variable z_n as

$$s_x(\zeta_x) = \alpha_s \tanh(w_s z_n + b_s), \quad t_x(\zeta_x) = w_t^T z_n + b_t, \quad q_x(\zeta_x) = \alpha_q \tanh(w_q z_n + b_q) \tag{5}$$

where α_s, and α_q are trainable scaling factors. The only requirement on the details of the network is that the dimensionality of the outputs s_x, t_x, q_x match the dimensionality of our physical variables $x, v \in \mathbb{R}^n$.

Next, we introduce a loss function, $\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}\left[-\frac{1}{a^2} \cdot \delta(\xi', \xi) \cdot A(\xi'|\xi)\right]$, where $\delta(\xi', \xi)$ is a suitably chosen metric function, and a is a scaling factor.

4 Annealing Schedule

To help our sampler overcome the large energy barriers between isolated modes, we introduce an *annealing schedule* during the training phase (N training steps) $\{\gamma_t\}_{t=0}^N = \{\gamma_0, \gamma_1, \ldots, \gamma_{N-1}, \gamma_N\}$, where $\gamma_0 < \gamma_1 < \cdots < \gamma_N \equiv 1$, $\gamma_{t+1} - \gamma_t \ll 1$. Note that we are free to vary γ during the initial training phase as long as we recover the true distribution with $\gamma \equiv 1$ at the end of training and evaluate our trained model without this factor. Explicitly, for $\gamma_t < 1$ this rescaling factor helps to reduce the height of the energy barriers, making it easier for our sampler to explore previously inaccessible regions of the phase space. In terms of this additional annealing schedule, our target distribution picks up an additional index t to represent our progress through the training phase, which can be written explicitly as

$$p_t(x) \propto e^{-\gamma_t S(x)}, \quad \text{for} \quad t = 0, 1, \ldots, N \tag{6}$$

5 Lattice Gauge Theory

Let $U_\mu(n) = e^{i x_\mu(n)} \in U(1)$, with $x_\mu(n) \in [-\pi, \pi]$ denote the *link variables*, where $x_\mu(n)$ is the link oriented in the $\hat{\mu}$-direction located at the site n. We can write our target distribution $p_t(x)$ in terms of the Wilson action, $S_\beta(x) = \beta \cdot \sum_P 1 - \cos(x_P)$ as

$$p_t(x) = e^{-\gamma_t \beta \cdot S(x)}, \quad \text{with} \quad \beta \cdot S(x) = \sum_P 1 - \cos(x_P) \tag{7}$$

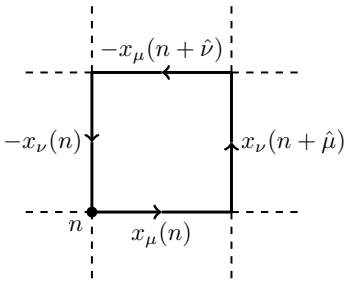
where $x_P \equiv x_\mu(n) + x_\nu(n + \hat{\mu}) - x_\mu(n + \hat{\nu}) - x_\nu(n)$ is the sum of the link variables around the 1×1 elementary plaquette, shown in Figure 2a. Here, $\beta = 2/g_0^2$ is the inverse coupling constant and $\beta \to \infty$ recovers the continuum limit of the theory. We consider two physical quantities of interest, namely the *real* and *integer* valued *topological charge*, ($\mathcal{Q}_\mathbb{R} \in \mathbb{R}$, and $\mathcal{Q}_\mathbb{Z} \in \mathbb{Z}$, respectively) defined as

$$\mathcal{Q}_\mathbb{R}(x) \equiv \frac{1}{2\pi} \sum_P \sin(x_P), \quad \mathcal{Q}_\mathbb{Z}(x) \equiv \frac{1}{2\pi} \sum_P \lfloor x_P \rfloor, \quad \text{where} \quad \lfloor x_P \rfloor = x_P - 2\pi \left\lfloor \frac{x_P + \pi}{2\pi} \right\rfloor$$

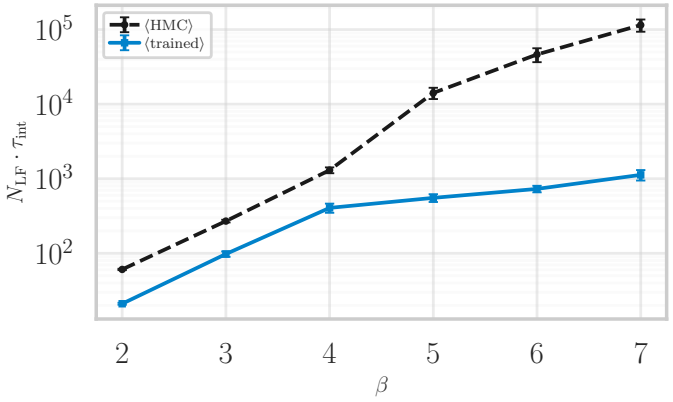
Acknowledgments

This research used resources of the Argonne Leadership Computing Facility (ALCF), which is a DOE office of science user facility supported under contract DE_AC02–06CH11357.This work describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the work do not necessarily represent the views of the u.s. doe or the united states government.

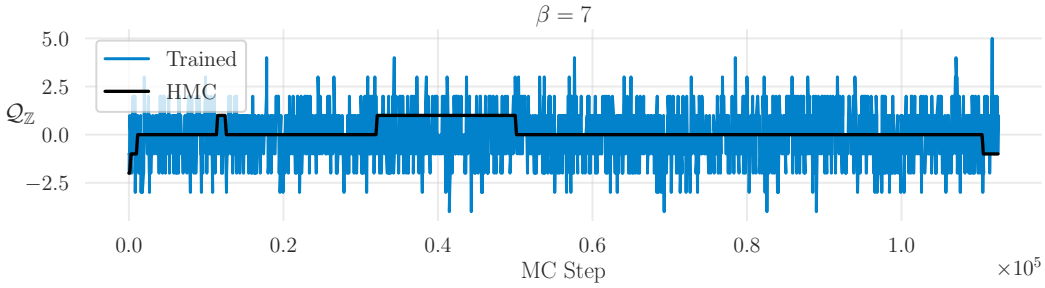
[4]Because we maintain a separate network with identical architecture for evaluating the s_v, t_v, q_v functions in Equation 2, the procedure is identical.



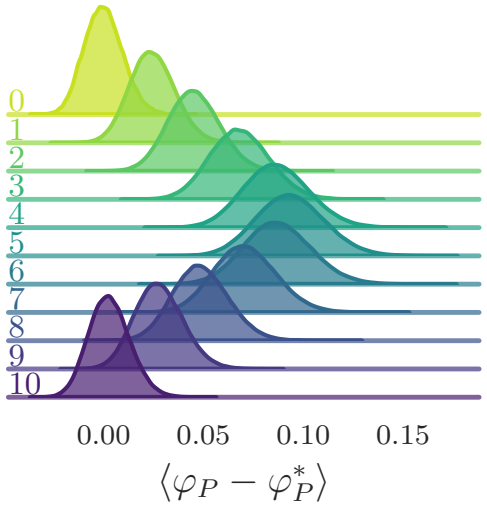
(a) Plaquette, P



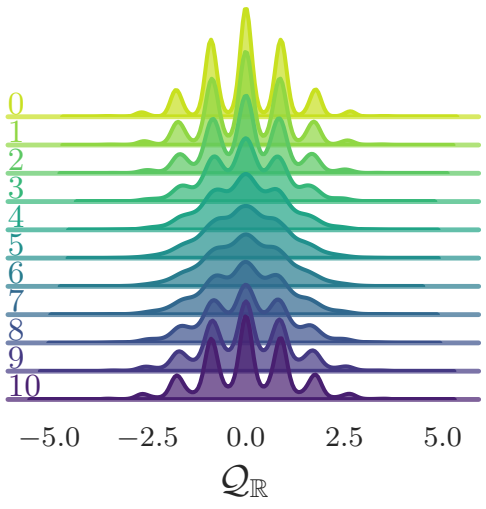
(b) Estimate of the integrated autocorrelation time $\tau_{\mathrm{int}}^{\mathcal{Q}_\mathbb{R}}$ vs β, scaled by N_{LF} to account for different trajectory lengths.



(c) Topological charge $\mathcal{Q}_\mathbb{R}$ vs MC step for both HMC (black line) and the trained model (blue line).



(d) Error in the average plaquette, $\langle x_P - x_P^* \rangle$ vs leapfrog step.



(e) The real-valued topological charge, $\mathcal{Q}_\mathbb{R}$ vs leapfrog step.

REFERENCES

Michael S Albergo, Denis Boyda, Daniel C Hackett, Gurtej Kanwar, Kyle Cranmer, Sébastien Racanière, Danilo Jimenez Rezende, and Phiala E Shanahan. Introduction to normalizing flows for lattice field theory. *arXiv preprint arXiv:2101.08176*, 2021.

MS Albergo, G Kanwar, and PE Shanahan. Flow-based generative models for markov chain monte carlo in lattice field theory. *Physical Review D*, 100(3):034515, 2019.

Denis Boyda, Gurtej Kanwar, Sébastien Racanière, Danilo Jimenez Rezende, Michael S Albergo, Kyle Cranmer, Daniel C Hackett, and Phiala E Shanahan. Sampling using $su(n)$ gauge equivariant flows. *arXiv preprint arXiv:2008.05456*, 2020.

Guido Cossu, Peter Boyle, Norman Christ, Chulwoo Jung, Andreas Jüttner, and Francesco Sanfilippo. Testing algorithms for critical slowing down. In *EPJ Web of Conferences*, volume 175, pp. 02008. EDP Sciences, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv:1605.08803 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1605.08803. arXiv: 1605.08803.

Matteo Favoni, Andreas Ipp, David I Müller, and Daniel Schuh. Lattice gauge equivariant convolutional neural networks. *arXiv preprint arXiv:2012.12901*, 2020.

Matthew Hoffman, Pavel Sountsov, Joshua V Dillon, Ian Langmore, Dustin Tran, and Srinivas Vasudevan. Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport. *arXiv preprint arXiv:1903.03704*, 2019.

Gurtej Kanwar, Michael S Albergo, Denis Boyda, Kyle Cranmer, Daniel C Hackett, Sébastien Racaniere, Danilo Jimenez Rezende, and Phiala E Shanahan. Equivariant flow-based sampling for lattice gauge theory. *Physical Review Letters*, 125(12):121601, 2020.

Daniel Lévy, M. Hoffman, and Jascha Sohl-Dickstein. Generalizing hamiltonian monte carlo with neural networks. *ArXiv*, abs/1711.09268, 2018.

Zengyi Li, Yubei Chen, and Friedrich T Sommer. A neural network mcmc sampler that maximizes proposal entropy. *arXiv preprint arXiv:2010.03587*, 2020.

Matija Medvidovic, Juan Carrasquilla, Lauren E Hayward, and Bohdan Kulchytskyy. Generative models for sampling of lattice field theories. *arXiv preprint arXiv:2012.01442*, 2020.

Kirill Neklyudov and Max Welling. Orbital mcmc. *arXiv preprint arXiv:2010.08047*, 2020.

Kirill Neklyudov, Max Welling, Evgenii Egorov, and Dmitry Vetrov. Involutive mcmc: a unifying framework. In *International Conference on Machine Learning*, pp. 7273–7282. PMLR, 2020.

Cristian Pasarica and Andrew Gelman. Adaptively scaling the metropolis algorithm using expected squared jumped distance. *Statistica Sinica*, pp. 343–364, 2010.

Danilo Jimenez Rezende, George Papamakarios, Sébastien Racaniere, Michael Albergo, Gurtej Kanwar, Phiala Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. In *International Conference on Machine Learning*, pp. 8083–8092. PMLR, 2020.

Stefan Schaefer, Rainer Sommer, and Francesco Virotta. Investigating the critical slowing down of qcd simulations. *arXiv preprint arXiv:0910.1465*, 2009.

Akinori Tanaka and Akio Tomiya. Towards reduction of autocorrelation in hmc by machine learning. *arXiv preprint arXiv:1712.03893*, 2017.

Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks. *arXiv preprint arXiv:1909.13789*, 2019.

Antoine Wehenkel and Gilles Louppe. You say normalizing flows i see bayesian networks. *arXiv preprint arXiv:2006.00866*, 2020.

A APPENDIX

A.1 HAMILTONIAN MONTE CARLO (HMC)

The Hamiltonian Monte Carlo algorithm is a widely used technique that allows us to sample from an analytically known target distribution $p(x)$ by constructing a chain of states $\{x^{(0)}, x^{(1)}, \ldots, x^{(n)}\}$, such that $x^{(n)} \sim p(x)$ in the limit $n \to \infty$. For our purposes, we assume that our target distribution can be expressed as a Boltzmann distribution, $p(x) = \frac{1}{\mathcal{Z}} e^{-S(x)} \propto e^{-S(x)}$, where $S(x)$ is the *action* of our theory, and $\mathcal{Z}$ is often referred to as the *partition function*, which ensures our target distribution is correctly normalized to unity. In this case, HMC begins by augmenting the state space with a fictitious momentum variable v, normally distributed independently of x, i.e. $v \sim \mathcal{N}(0, \mathbb{1})$. Our joint distribution can then be written as $p(x, v) = p(x) \cdot p(v) \propto e^{-S(x)} \cdot e^{-\frac{1}{2} v^T v} = e^{-\mathcal{H}(x,v)}$, where $\mathcal{H}(x, v)$ is the Hamiltonian of the joint (x, v) system. This system obeys Hamilton's equations: $\dot{x} = \frac{\partial \mathcal{H}}{\partial v}$, $\dot{v} = -\frac{\partial H}{\partial x}$, which can be numerically integrated using the leapfrog integrator along iso-probability contours of $\mathcal{H} = \text{const.}$. Explicitly, for a step size ε and initial state $\xi = (x, v)$, the leapfrog integrator generates a proposal configuration $\xi' \equiv (x', v')$ by performing the following series of updates:

1. Half-step momentum update: $\quad v^{1/2} \equiv v\big(t + \frac{\varepsilon}{2}\big) = v - \frac{\varepsilon}{2} \partial_x S(x)$
2. Full-step position update: $\quad x' \equiv x(t + \varepsilon) = x + \varepsilon v^{1/2}$
3. Half-step momentum update: $\quad v' \equiv v(t + \varepsilon) = v^{1/2} - \frac{\varepsilon}{2} \partial_x S(x')$

We can then construct a complete *trajectory* of length $\lambda = \varepsilon \cdot N_{\text{LF}}$ by performing N_{LF} leapfrog steps in sequence. At the end of our trajectory, we either accept or reject the proposal configuration according to the Metropolis-Hastings acceptance criteria,

$$x_{i+1} = \begin{cases} x' & \text{with probability } A(\xi'|\xi) \\ x & \text{with probability } (1 - A(\xi'|\xi)), \end{cases} \quad \text{and} \quad A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)} \left|\frac{\partial \xi'}{\partial \xi^T}\right|\right\}. \tag{8}$$

The generic leapfrog integrator is known to be symplectic (conserves energy), so the Jacobian factor reduces to $\left|\frac{\partial \xi'}{\partial \xi^T}\right| = 1$.

A.2 TRAINING ALGORITHM

Algorithm 1: Training procedure

input :

1. `Target distribution,` $p_t(x) \propto e^{-\beta_t U(x)}$
2. `Loss function,` $\mathcal{L}_\theta(\xi', \xi, A(\xi'|\xi))$
3. `Learning rate schedule,` $\{\alpha_t\}_{t=0}^{N_{\text{train}}}$
4. `Annealing schedule,` $\{\gamma_t\}_{t=0}^{N_{\text{train}}}$
5. `Batch of initial states,` x

`Initialize weights` θ

for $0 \leq t < N_{\text{train}}$ **:**

- `update:` $p_t(x) \propto e^{-\beta_t U(x)}$
- `resample:` $v \sim \mathcal{N}(0, \mathbb{1})$
- `resample:` $d \sim \mathcal{U}(+, -)$
- `construct:` $\xi \equiv (x, v, d)$
- **for** $0 \leq \ell < N_{\text{LF}}$ **:**
  - `propose:` $\xi'_\ell \leftarrow \mathbf{FL}_\ell^{\pm} \xi_\ell$
- `compute:` $A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)} \left|\frac{\partial \xi'}{\partial \xi^T}\right|\right\}$
- `update:` $\mathcal{L} \leftarrow \mathcal{L}_\theta(\xi', \xi, A(\xi'|\xi))$
- `backprop:` $\theta \leftarrow \theta - \alpha_t \nabla_\theta \mathcal{L}$
- `assign:` $x_{t+1} \leftarrow \begin{cases} x' & \text{with probability } A(\xi'|\xi) \\ x & \text{with probability } (1 - A(\xi'|\xi)). \end{cases}$