PROCEEDINGS OF SCIENCE

LeapfrogLayers: A Trainable Framework for Effective Topological Sampling

Sam Foreman,[a,*] **Xiao-Yong Jin**[a] **and James C. Osborn**[a]

[a] *Argonne National Laboratory,*
Lemont, IL

E-mail: foremans@anl.gov, xjin@anl.gov, osborn@alcf.anl.gov

We introduce LeapfrogLayers, an invertible neural network architecture that can be trained to efficiently sample the topology of a 2D $U(1)$ lattice gauge theory. We show an improvement in the integrated autocorrelation time of the topological charge when compared with traditional HMC, and propose methods for scaling our model to larger lattice volumes.

The 38th International Symposium on Lattice Field Theory
26-30 July 2021
Zoom / Gather @ MIT, Cambridge MA, USA

*Speaker

1. Introduction

Background

One of the major goals of lattice field theory calculations is to evaluate integrals of the form

$$\langle \mathcal{O} \rangle \propto \int [\mathcal{D}x]\, \mathcal{O}(x) p(x), \tag{1}$$

for some target distribution $p(x) \propto e^{-S(x)}$. We can approximate the integral using Markov Chain Monte Carlo (MCMC) sampling techniques. This is done by sequentially generating a chain of configurations $\{x_1, x_2, \dots x_N\}$, with $x_i \sim p(x)$ and averaging the value of the function $\mathcal{O}(x)$ over the chain. Accounting for correlations between states in our chain, the sampling variance of this esimator is given by

$$\sigma^2 = \frac{\tau_{\mathcal{O}}^{\text{int}}}{N} \sum_{n=1}^{N} \text{Var}\left[\mathcal{O}(x)\right] \tag{2}$$

where $\tau_{\text{int}}^{\mathcal{O}}$ is the integrated autocorrelation time. This quantity can be interpreted as the additional time required for these induced correlations to become negligible.

1.1 Charge Freezing

The ability to efficiently generate independent configurations is currently a major bottleneck for lattice simulations. In this work we consider a $U(1)$ gauge model on a 2D lattice with periodic boundary conditions. The theory is defined in terms of the link variables $U_\mu(x) = e^{i x_\mu(n)} \in U(1)$ with $x_\mu(n) \in [-\pi, \pi]$. Our target distribution is given by $p(x) \propto e^{-S_\beta(x)}$, where $S_\beta(x)$ is the Wilson action

$$S_\beta(x) = \beta \sum_P 1 - \cos x_P, \text{ and} \tag{3}$$
$$x_P \equiv \left[x_\mu(n) + x_\nu(n+\hat{\mu}) - x_\mu(n+\hat{\nu}) - x_\nu(n)\right],$$

is the sum of the gauge variables around the elementary plaquette. For a given lattice configuration, we can calculate the topological charge $Q \in \mathbb{Z}$ using

$$Q_{\mathbb{Z}} = \frac{1}{2\pi} \sum_P \lfloor x_P \rfloor, \text{ where}$$
$$\lfloor x_P \rfloor \equiv x_P - 2\pi \left\lfloor \frac{x_P + \pi}{2\pi} \right\rfloor.$$

As $\beta \to \infty$, the $Q_{\mathbb{Z}} = 0$ mode becomes dominant and we see that the value of $Q_{\mathbb{Z}}$ remains fixed for large durations of the simulation.

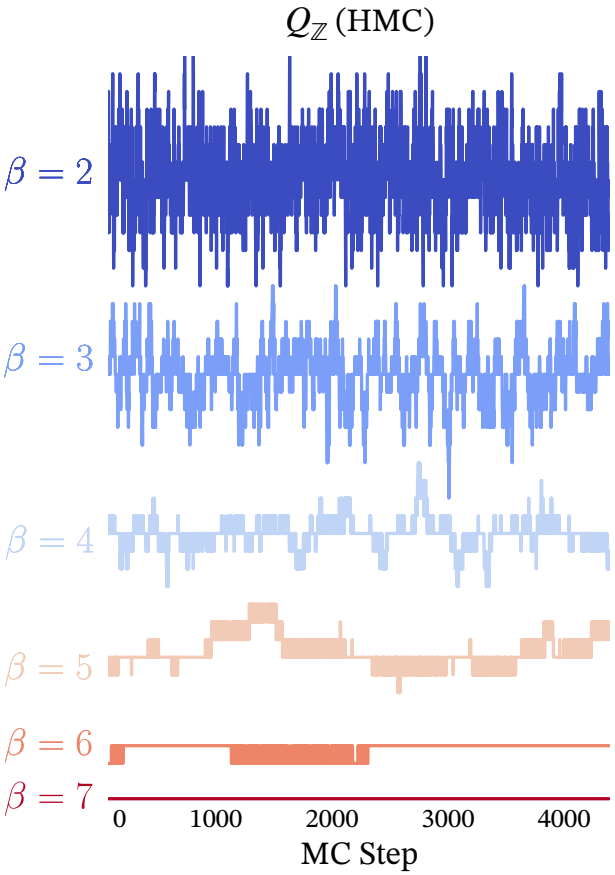


Figure 1: Illustration of the topological charge Q freezing as $\beta : 2 \to 7$ for traditional HMC.

This can be seen by introducing the *tunneling rate*[1]

$$\delta Q \equiv |Q_{i+1} - Q_i| \in \mathbb{Z}. \tag{4}$$

This quantity serves as a measure for how efficiently our chain is able to jump (tunnel) between sectors of distinct topological charge. From Fig 1, we can see that $\delta Q \to 0$ as $\beta \to \infty$.

2. Hamiltonian Monte Carlo (HMC)

The Hamiltonian Monte Carlo algorithm begins by introducing a fictitious momentum $v \sim \mathcal{N}(0, \mathbb{1})$ distributed independently of x. This allows us to write the joint target density of the $\xi \equiv (x, v)$ system as

$$p(x, v) = p(x) \cdot p(v) = e^{-S_\beta(x)} \cdot e^{-v^T v/2} = e^{-H(x,v)} \tag{5}$$

where $H(\xi) = H(x, v) = S_\beta(x) + \frac{1}{2} v^T v$ is the Hamiltonian of the system. We use the *leapfrog integrator* to numerically integrate Hamilton's equations

$$\dot{x} = \frac{\partial H}{\partial v} \quad , \quad \dot{v} = -\frac{\partial H}{\partial x} \tag{6}$$

along iso-probability contours of $H = \text{const.}$ from $\xi = (x, v) \to (x^*, v^*) = \xi^*$.

2.1 Leapfrog Integrator

1. Starting from x_0, resample the momentum $v_0 \sim \mathcal{N}(0, \mathbb{1})$ and construct the state $\xi_0 = (x_0, v_0)$.

2. Generate a *proposal configuration* ξ^* by integrating $\dot{\xi}$ along $H = \text{const.}$ for N leapfrog steps. i.e.

$$\xi_0 \to \xi_1 \to \ldots \to \xi_N \equiv \xi^*, \tag{7}$$

   where a single leapfrog step $\xi_i \to \xi_{i+1}$ above consists of:

$$\textbf{(a.)}\ \tilde{v} \leftarrow v - \frac{\varepsilon}{2}\partial_x S(x), \qquad \textbf{(b.)}\ x' \leftarrow x + \varepsilon\tilde{v}, \qquad \textbf{(c.)}\ v' \leftarrow \tilde{v} - \frac{\varepsilon}{2}\partial_x S(x). \tag{8}$$

3. At the end of the trajectory, accept or reject the proposal configuration ξ^* using the Metropolis-Hastings (MH) test

$$x_{i+1} \leftarrow \begin{cases} x^* \text{ with probability } A(\xi^*|\xi) \equiv \min\left\{1, \frac{p(\xi^*)}{p(\xi)}\left|\frac{\partial \xi^*}{\partial \xi^T}\right|\right\} \\ x_i \text{ with probability } 1 - A(\xi^*|\xi). \end{cases} \tag{9}$$

An illustration of this procedure can be seen in Fig 2.

[1] As measured between subsequent states in our chain $i, i+1$.

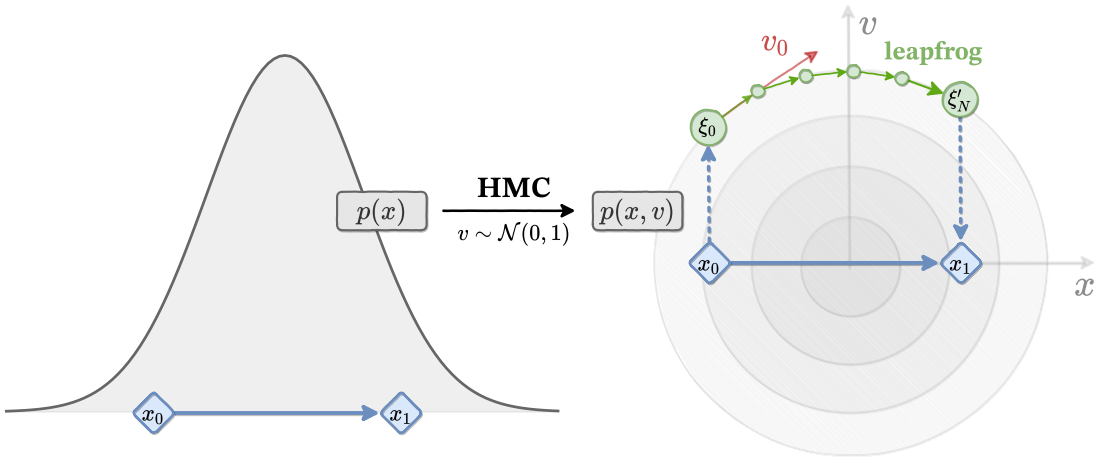


Figure 2: High-level overview of the HMC algorithm.

2.2 Issues with HMC

Re-sampling the momentum at the start of each trajectory causes the energy levels we explore to be randomly selected. This is reminiscient of the "random-walk" behavior of traditional MCMC and leads to a slow exploration of our target distribution (i.e. long autocorrelations). Additionally, the HMC sampler is known to have difficulty traversing low-density zones, resulting in poor performance for distributions which have multiple isolated modes. This is particularly relevant in the case of sampling topological quantities in lattice gauge models.

3. Generalizing HMC: LeapfrogLayers

In order to preserve the asymptotic behavior of HMC, our update must be explicitly reversible with a tractable Jacobian determinant. To simplify notation, we introduce two functions, Γ (Λ) to denote the v (x) updates. As in HMC, we follow the general pattern of performing alternating updates of v and x.

We can ensure our update is reversible by splitting the x-update into two parts and sequentially updating complementary subsets using a binary mask m and its complement $\bar{m}$. Additionally, we introduce $d \sim \mathcal{U}(+,-)$, distributed independently of both x and v, to determine the "direction" of our update[2]. Here, we associate $+$ ($-$) with the forward (backward) direction and note that running sequential updates in opposite directions has the effect of inverting the update. We denote the complete state by $\xi = (x, v, \pm)$, with target density given by $p(\xi) = p(x) \cdot p(v) \cdot p(\pm)$.

Explicitly, we can write this series of updates as[3]

$$(\mathbf{1}.) \quad v' = \Gamma^{\pm}[v; \zeta_v] \tag{10}$$

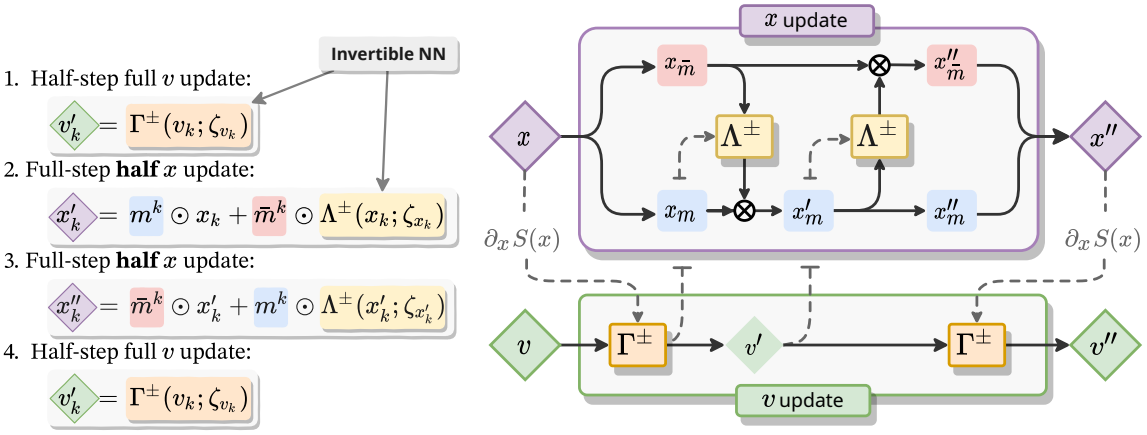
$$(\mathbf{2}.) \quad x' = m \odot x + \bar{m} \odot \Lambda^{\pm}[x_{\bar{m}}; \zeta_{\bar{x}}] \tag{11}$$

$$(\mathbf{3}.) \quad x'' = \bar{m} \odot x' + m \odot \Lambda^{\pm}[x_m; \zeta_{x'}] \tag{12}$$

$$(\mathbf{4}.) \quad v'' = \Gamma^{\pm}[v', \zeta_{v'}] \tag{13}$$

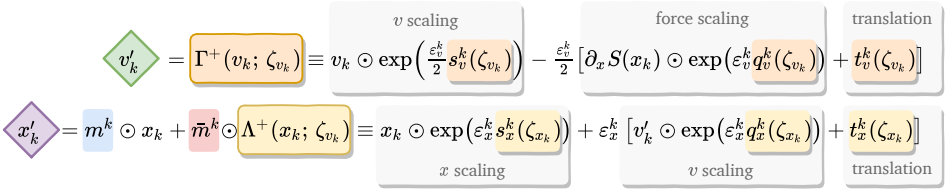
[2] As indicated by the superscript $\pm$ on $\Gamma^{\pm}, \Lambda^{\pm}$ in the update functions.

[3] Here we denote by $x_m = m \odot x$ and $x_{\bar{m}} = \bar{m} \odot x$ with $\mathbb{1} = m + \bar{m}$.



(a) Generalized leapfrog update.

(b) Schematic diagram showing the data flow through a leapfrog layer.



(c) Detailed view of the first two steps in Fig 3a.

Figure 3: Various illustrations of the generalized leapfrog update. Note that k indexes the leapfrog step along our trajectory.

where $\zeta_{\bar{x}} = [\bar{m} \odot x, v]$, $\zeta_x = [m \odot x, v]$ ($\zeta_v = [x, \partial_x S(x)]$) is independent of x (v) and is passed as input to the update functions $\Lambda^{\pm}$ ($\Gamma^{\pm}$).

3.1 Network Details

Normalizing flows [3] are an obvious choice for the structure of the update functions. These architectures are easily invertible while maintaining a tractable Jacobian determinant, and have also been shown to be effective at approximating complicated target distributions in high-dimensional spaces [1–9],

Note that we maintain separate networks Γ, Λ with identical architectures for updating v and x, respectively. Without loss of generality, we describe below the details of the x-update for the forward ($d = +$) direction, $\Lambda^{+}[x_{\bar{m}}; \zeta_{\bar{x}}]$[4]. For simplicity, we describe the data flow through a single leapfrog layer, which takes as input $\zeta_{\bar{x}} = (x_{\bar{m}}, v)$. For the 2D $U(1)$ model, the gauge links are

[4]To obtain the expression for the $d = -$ direction we simply invert the update function and perform the updates in opposite order.

encoded as $[\cos(x), \sin(x)]$ for $x \in [-\pi, \pi]$. Explicitly[5],

$$h_1 = \sigma\left(w_x^T x + w_v^T v + b_1\right) \quad \in \mathbb{R}^{n_1} \tag{14}$$

$$h_2 = \sigma\left(w_2^T h_1 + b_2\right) \quad \in \mathbb{R}^{n_2} \tag{15}$$

$$\vdots$$

$$h_k = \sigma\left(w_k^T h_{k-1} + b_{k-1}\right) \quad \in \mathbb{R}^{n_k} \implies \tag{16}$$

$$\textbf{[1.]}\ s_x = \lambda_s \tanh\left(w_s^T h_k + b_s\right); \quad \textbf{[2.]}\ t_x = w_t^T h_k + b_t; \quad \textbf{[3.]}\ q_x = \lambda_q \tanh\left(w_q^T h_k + b_q\right);$$

where the outputs s_x, t_x, q_x are of the same dimensionality as x, and λ_s, λ_q are trainable parameters. Note that the shapes of the hidden layers h_k are determined by the sizes of each of the weight matrices w_k^T. These outputs are then used to update x, as shown in Fig 3.

3.2 Training Step

1. Resample $v \sim \mathcal{N}(0, \mathbb{1})$, $d \sim \mathcal{U}(+, -)$, and construct initial state $\xi_0 = (x_0, v_0, \pm)$
2. Generate the proposal configuration ξ^* by passing the initial state sequentially through N *leapfrog layers* to construct a trajectory: $\xi_0 \to \xi_1 \to \ldots \to \xi_N = \xi^*$
3. Compute the Metropolis-Hastings acceptance $A(\xi^*|\xi) = \min\left\{1, \frac{p(\xi^*)}{p(\xi)}\left|\frac{\partial \xi'}{\partial \xi^T}\right|\right\}$
4. Evaluate the loss function $\mathcal{L} \leftarrow \mathcal{L}_\theta(\xi^*, \xi)$, and backpropagate gradients to update weights
5. Evaluate Metropolis-Hastings criteria and assign the next state in the chain according to $x_{t+1} \leftarrow \begin{cases} x^* \text{ with prob. } A(\xi^*|\xi) \\ x \text{ with prob. } 1 - A(\xi^*|\xi). \end{cases}$

4. Results

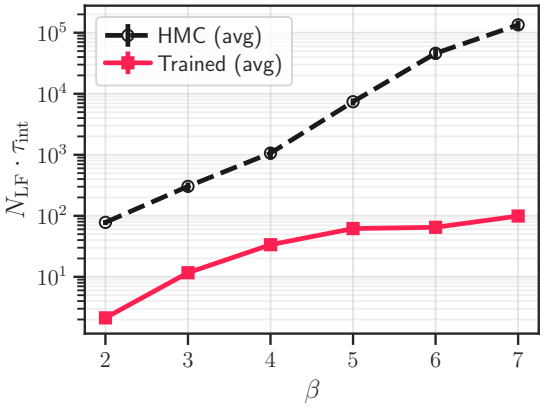
We can measure the performance of our approach by comparing the integrated autocorrelation time against traditional HMC. We see in Fig 4a that our estimate of the integrated autocorrelation time is much shorter for the trained model across $\beta \in [2, 7]$. To better understand how these transformations effect the HMC update, we can look at how various quantities evolve over the course of a trajectory, as shown in Fig 5.

References

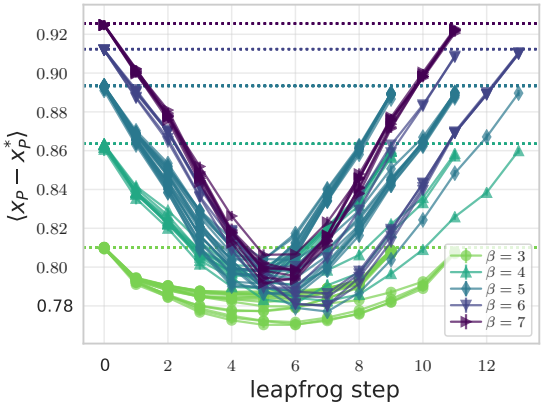
[1] MS Albergo, G Kanwar, and PE Shanahan. Flow-based generative models for Markov chain Monte Carlo in lattice field theory. 100(3):034515.

[2] 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.

[5] Here $\sigma(\cdot)$ is a generic nonlinear activation function.



(a) Plot of the integrated autocorrelation time $\tau_{\text{int}}^{\mathcal{Q}}$ of the topological charge for both HMC (black, dashed line) and the trained model (solid, red line).



(b) Deviation in the average plaquette vs leapfrog step.

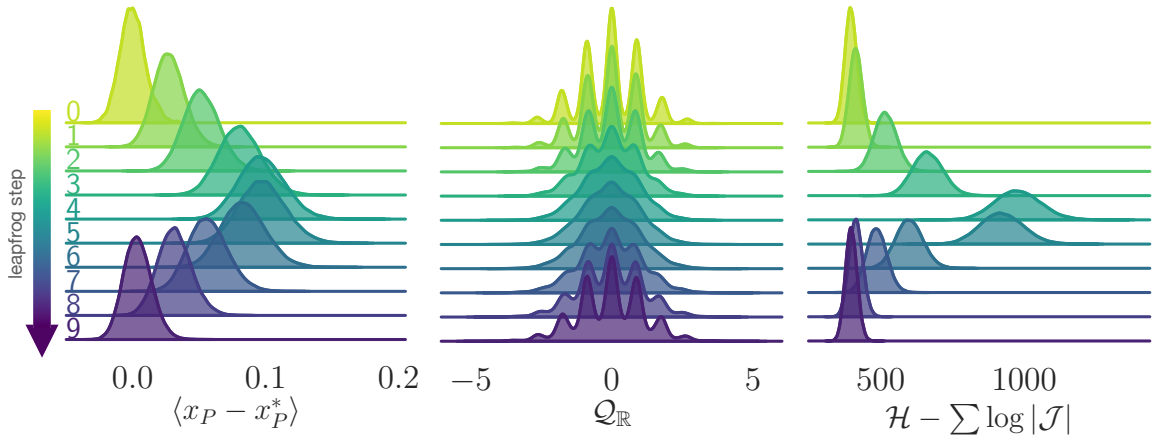


Figure 5: Evolution of the density of lattice observables over the course of a trajectory.

[3] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP.

[4] Sam Foreman, Xiao-Yong Jin, and James C. Osborn. Deep Learning Hamiltonian Monte Carlo.

[5] 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. 125(12):121601.

[6] Daniel Lévy, M. Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with Neural Networks. abs/1711.09268.

[7] Kirill Neklyudov and Max Welling. Orbital MCMC.

[8] Kirill Neklyudov, Max Welling, Evgenii Egorov, and Dmitry Vetrov. Involutive mcmc: A unifying framework. In *International Conference on Machine Learning*, pages 7273–7282. PMLR.

[9] Antoine Wehenkel and Gilles Louppe. You say normalizing flows i see bayesian networks.