Deep Learning Hamiltonian Monte Carlo

Sam Foreman, Xiao-Yong Jin & James C. Osborn *
Leadership Computing Facility
Argonne National Laboratory

Abstract

We generalize the Hamiltonian Monte Carlo algorithm with a stack of neural network layers, and evaluate its ability to sample from different topologies in a two-dimensional lattice gauge theory. We demonstrate that our model is able to successfully mix between modes of different topologies, significantly reducing the computational cost required to generate independent gauge field configurations. Our implementation is available at https://github.com/saforem2/l2hmc-qcd.

1 Introduction

In Lévy et al. (2017), the authors propose the Learning to Hamiltonian Monte Carlo (L2HMC) algorithm, and demonstrate its ability to outperform traditional Hamiltonian Monte Carlo (HMC) on a variety of target distributions. They show that the trained L2HMC sampler is capable of mixing between modes by exploring regions of phase space which are inaccessible with traditional HMC. In this paper, we propose a generalized sampler using a deep neural network that is self-trained to propose new Markov Chain states, which is made exact with the help of Metropolis-Hastings (Hastings, 1970) algorithm. We target our Deep Learning Hamiltonian Monte Carlo (DLHMC) algorithm to simulating lattice gauge theories, where state-of-the-art simulations have billions of degrees of freedom and run on supercomputers with thousands of nodes, yet suffer an exponential slowdown when approaching continuum physics (Schaefer et al., 2009; Cossu et al., 2017).

2 Main Contributions

We propose the DLHMC algorithm as a generalized HMC algorithm, using a stack of distinct neural network layers replacing consecutive leapfrog steps. Each distinct neural network layer, carrying a discrete index $k = 0, 1, \ldots, N_{\text{LF}} - 1$ for N_{LF} leapfrog layers, performs the augmented leapfrog equations (Equation 1 and Equation 2), which are parametrized by different neural networks.

We apply the proposed method to a 2-dimensional $U(1)$ lattice gauge theory defined on a square lattice with periodic boundary conditions. We specifically designed a loss function for our physical system, encouraging the tunneling of topological properties. We see a significant reduction in the computational cost using DLHMC, as measured by the integrated autocorrelation time of the topological charge. We compare our results to traditional HMC across a variety of trajectory lengths and coupling constants, and show that our trained models consistently outperform traditional HMC (see Figure 3). Models trained on smaller coupling constants, when applied to a larger coupling constant, require only minimal retraining. We find that the efficiency in tunneling topological sectors of our trained DLHMC sampler is explained by the behavior of the physical system as it passes through the deep neural network, see Figure 5a and Figure 5b.

*foremans@anl.gov, xjin@anl.gov, osborn@alcf.anl.gov

3 Related Work:

Recently, there has been a growing interest in applying generative machine learning techniques to build smarter, more efficient scientific simulations. Following the development of the RealNVP (Dinh et al., 2017) architecture, there have been many proposed techniques that aim to take advantage of its invertible structure. In particular, simulations in lattice gauge theory and lattice QCD stand to benefit tremendously from this new approach, as evidenced by the rapidly-growing body of work in this direction ranging from network architectures (Dinh et al., 2017; Favoni et al., 2020; Toth et al., 2020), generative models (Albergo et al., 2019; 2021; Medvidovic et al., 2020; Boyda et al., 2020; Kanwar et al., 2020; Wehenkel & Louppe, 2020; Tomiya & Nagai, 2021), and novel MCMC approaches (Pasarica & Gelman, 2010; Tanaka & Tomiya, 2017; Hoffman et al., 2019; Neklyudov & Welling, 2020; Neklyudov et al., 2020; Rezende et al., 2020; Li et al., 2021)

4 DLHMC Algorithm

We provide a review of the generic Hamiltonian Monte Carlo (HMC) algorithm and set up some of the relevant notation in Section A.1. For simulating a system, x, using a theoretically given probability density $p(x)$ with likely intractable normalization, we augment the Markov chain state as $\xi = (x, v, d)$ with target distribution $p(\xi) = p(x, v, d) = p(x)p(v)p(d)$. The conjugate momentum v typically used in HMC algorithms has a known and easy to sample distribution. The binary direction variable, as introduced in L2HMC (Lévy et al., 2017), $d \in \mathcal{U}(+,-)$, denoting the "direction" (forward/backward) of our update.

DLHMC algorithm further generalize the leapfrog steps in L2HMC using individual neural networks. Each leapfrog step in DLHMC is a layer of the deep neural network that transforms the input $\xi_k = (x_k, v_k, d_k)$, for the k-th layer, to output (x''_k, v''_k, d_k), where d_k stays constant choosing either the forward or backward leapfrog layer. Subsequently the $k+1$-th layer uses $\xi_{k+1} = (x''_k, v''_k, d_k)$ as its input. For simplicity, we consider the forward $d = +1$ direction, and introduce the notation:

$$v'_k \equiv \Gamma^+_k(v_k; \zeta_{v_k}) = v_k \odot \exp\left(\tfrac{\varepsilon^k_v}{2} s^k_v(\zeta_{v_k})\right) - \tfrac{\varepsilon^k_v}{2}\left[\partial_x S(x_k) \odot \exp\left(\varepsilon^k_v q^k_v(\zeta_{v_k})\right) + t^k_v(\zeta_{v_k})\right], \quad (1)$$

$$x'_k \equiv \Lambda^+(x_k; \zeta_{x_k}) = x_k \odot \exp\left(\varepsilon^k_x s^k_x(\zeta_{x_k})\right) + \varepsilon^k_x\left[v'_k \odot \exp\left(\varepsilon^k_x q^k_x(\zeta_{x_k})\right) + t^k_x(\zeta_{x_k})\right] \quad (2)$$

where: (1.) $\zeta_{v_k} = (x_k, \partial_x S(x_k))$ and $\zeta_{x_k} = (\bar{m}^k \odot x_k, v_k)$, $\zeta_{x'_k} = (m^k \odot x'_k, v'_k)$ denote the (x, v respectively) networks' inputs[1], where $\odot$ denotes element-wise multiplication; (2.) we indicate the direction of the update by the superscript $\pm$ on $\Gamma^\pm_k, \Lambda^\pm_k$; (3.) $k \in \{0, 1, \ldots, N_{\text{LF}} - 1\}$ denotes the current leapfrog step along the trajectory. We include in Figure 1 an illustration of the network architecture of the k-th leapfrog layer used for the updates in Equation 1 and Equation 2.

Using this notation, we can write a 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] $x'_k = \bar{m}^k \odot x_k + m^k \odot \Lambda^+_k(x_k; \zeta_{x_k})$
3. Full-step half-position update: $x''_k = \bar{m}^k \odot \Lambda^+_k(x'_k; \zeta_{x'_k}) + m^k \odot x'_k$
4. Half-step momentum update: $v''_k = \Gamma^+_k(v'_k; \zeta_{v'_k})$

Collectively, we refer to this series of updates as a single leapfrog layer, which performs a single update $\xi \to \xi'$. Note that in order to keep our update reversible, we've split the x

[1] The v network is independent of the variables being updated, the x network will be made independent with the mask m^k.

[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 $m^k \odot x$), followed by an analogous update of the complementary indices, $\bar{m}^k \odot x$.

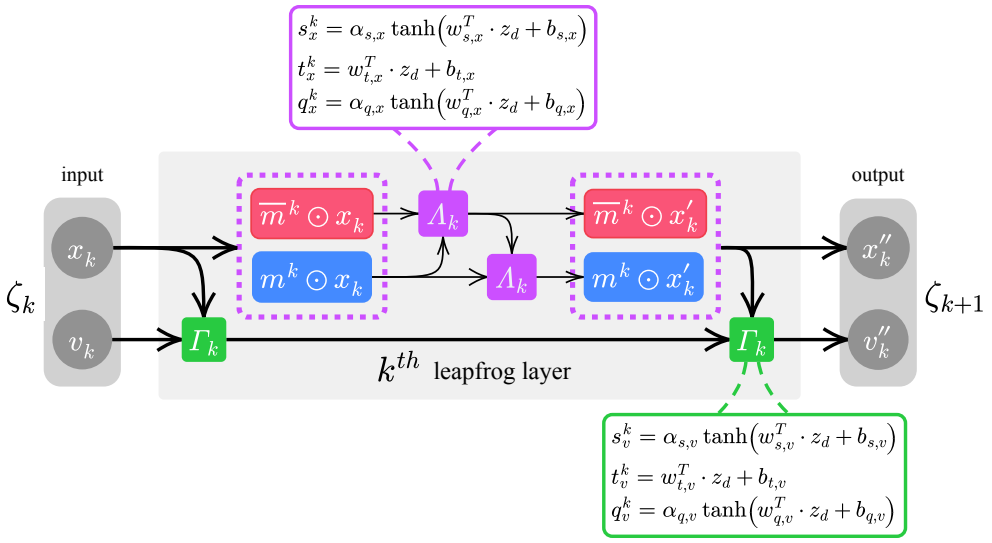


Figure 1: Illustration of the network architecture used in Equation 1 and Equation 2.

update into two sub-updates by introducing a binary mask $\bar{m}^k = \mathbb{1} - m^k$ that updates half of the components of x sequentially.

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 7. 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 $\left|\frac{\partial v''_k}{\partial v_k}\right| = \exp\left(\frac{1}{2}\varepsilon_v^k s_v^k(\zeta_{v_k})\right)$, $\left|\frac{\partial x''_k}{\partial x_k}\right| = \exp\left(\varepsilon_x^k s_x^k(\zeta_{x_k})\right)$. where we keep the Jacobian factor for individual degree of freedom tractable.

5 $U(1)$ Lattice Gauge Theory

Lattice gauge theory is one formulation of quantum field theories with gauge fields, eg. Quantum Electrodynamics (QED) and Quantum Chromodynamics (QCD). It is the predominant method that can be computed numerically, and widely used for simulating subatomic particles and nuclei. Here we describe the $U(1)$ gauge (QED), and apply our DLHMC in 2 dimensional systems. Let $U_\mu(n) = e^{ix_\mu(n)} \in U(1)$, 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 in terms of the Wilson action $S_\beta(x)$ as

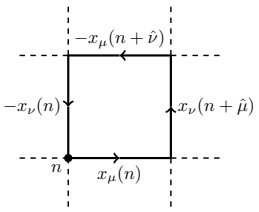


Figure 2: Plaquette

$$p_t(x) = \frac{1}{Z} e^{-\gamma_t S_\beta(x)}, \quad \text{with} \quad S_\beta(x) = \beta \sum_P 1 - \cos(x_P) \tag{3}$$

where Z is the intractable normalization; γ_t is a scaling factor ($\|\gamma_t\| \leq 1$) slowly annealed during training (Section B); $x_P \equiv x_\mu(n) + x_\nu(n+\hat{\mu}) - x_\mu(n+\hat{\nu}) - x_\nu(n)$ is a combination of link variables around the 1×1 elementary plaquette, shown in Figure 2; and the sum is over all such plaquettes on the lattice. Here, $\beta = 2/g_0^2$ is a coupling constant and $\beta \to \infty$ recovers the continuum limit of the theory.

Physically, each lattice configuration has its topological charge, $\mathcal{Q}_\mathbb{Z} \in \mathbb{Z}$, defined as

$$\mathcal{Q}_\mathbb{Z}(x) \equiv \frac{1}{2\pi} \sum_P \lfloor x_P \rfloor, \quad \text{where} \quad \lfloor x_P \rfloor = x_P - 2\pi \left\lfloor \frac{x_P + \pi}{2\pi} \right\rfloor. \tag{4}$$

Current methods are severely limited in their ability to mix between different topologies, a phenomenon known as topological freezing, when simulation approaches the continuum limit. For this reason, we wish to construct a loss function that encourages our sampler to explore different topological sectors. In order to do so, we introduce a continuous version of the topological charge, $\mathcal{Q}_{\mathbb{R}} \in \mathbb{R}$, by replacing the projection in Equation 4 to give:

$$\mathcal{Q}_{\mathbb{R}} \equiv \frac{1}{2\pi} \sum_{P} \sin(x_P). \tag{5}$$

This quantity has the advantage of being continuously differentiable, which is important for training the deep neural network. Our loss function is then defined as

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}[\, -\delta(\xi', \xi)\, A(\xi'|\xi)\,] \tag{6}$$

where $\delta(\xi', \xi) \equiv (\mathcal{Q}_{\mathbb{R}}(x') - \mathcal{Q}_{\mathbb{R}}(x))^2$ and $A(\xi'|\xi)$ is given in Equation 7.

6 Results

We apply our DLHMC algorithm with the training procedure in Section C and the annealing schedule from Section B to the 2D $U(1)$ gauge theory with parameters gradually approaching the continuum limit. We then run inference to generate the Markov chain using the trained model and compare its efficiency with HMC. We trained our models using Horovod (Sergeev & Balso, 2018) with TensorFlow (Abadi et al., 2016) on the ThetaGPU supercomputer at the Argonne Leadership Computing Facility. A typical training run on 1 node (8× NVIDIA A100 GPUs) using a batch size $M = 2048$, hidden layer shapes $= [256, 256, 256]$ for each of the $N_{\text{LF}} = 10$ leapfrog layers, on a 16×16 lattice, for 5×10^5 training steps takes roughly 24 hours to complete.

To measure the improvement of our model, we evaluate the integrated autocorrelation time of the topological charge which can be interpreted roughly as the number of trajectories needed (on average) before an independent sample is drawn. In order to more accurately capture the computational effort between the two approaches we scale our estimate of the integrated autocorrelation time by the number of leapfrog steps as $N_{\text{LF}}\tau$. We can see in Figure 3, that the trained model consistently outperforms generic HMC across $\beta = 2, 3, \ldots, 7$. To ensure the validity of our results, each trained run was compared to multiple HMC runs across a range of N_{LF} and ε values. We see in Figure 4 that HMC remains stuck at a particular value of $\mathcal{Q}_{\mathbb{Z}}$ for large sections of the simulation, whereas the trained sampler rapidly jumps between values. In order to better understand the mechanism driving this improved behavior, we evaluate different quantities of the system during its passing through the N_{LF} leapfrog layers. Figure 5a and Figure 5b show the distribution over batches of the average plaquette value and the modified energy of the Markov state, during the transformation via $N_{\text{LF}} = 10$ leapfrog layers. Our sampler artificially increased the energy density of the physical system during the first half of the trajectory before returning back to its original physical value. This is not a prescribed behavior, but self-learned during the training. We believe that this ability to vary the energy during the trajectory helps the sampler to overcome energy barriers between topological sectors whereas HMC remains stuck. As can be seen in Figure 5c, the real-valued topological charge exhibits mixing behavior at the middle of the deep neural network.

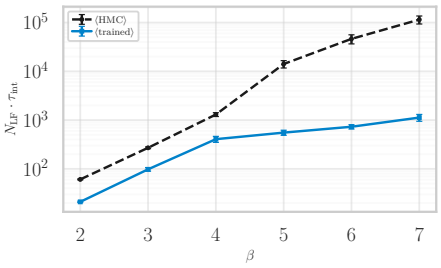


Figure 3: Estimate of the integrated autocorrelation time $\tau_{\text{int}}^{\mathcal{Q}_{\mathbb{R}}}$ vs β, scaled by N_{LF} to account for simulation cost.

7 Discussion and Further Research

We propose DLHMC as an efficient algorithm for MCMC, and observed a dramatic improvement in simulating 2D $U(1)$ gauge theory. Being able to efficiently tunnel topological sectors

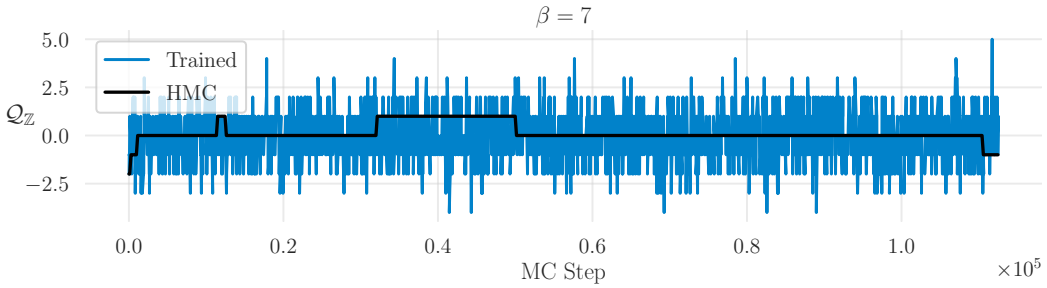
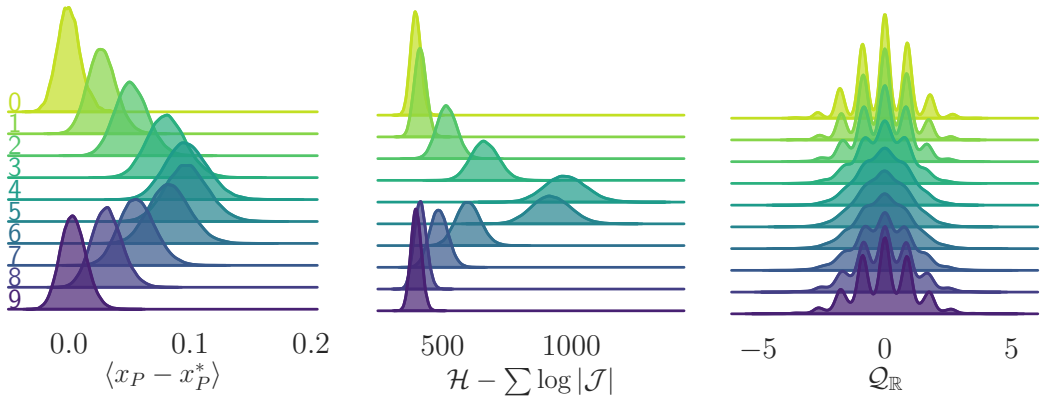


Figure 4: Topological charge $\mathcal{Q}_{\mathbb{Z}}$ vs MC step for both HMC (black) and the trained model (blue).



(a) Difference in the average plaquette, $\langle x_P - x_P^* \rangle$ at each leapfrog layer.

(b) The adjusted energy, $\mathcal{H} - \sum \log |\mathcal{J}|$ at each leapfrog layer.

(c) The real-valued topological charge, $\mathcal{Q}_{\mathbb{R}}$ at each leapfrog layer.

Figure 5: Plots showing how various quantities of interest are transformed as they're passed through each of the leapfrog layers in the generalized leapfrog update.

is a significant first step toward efficient simulating lattice gauge theories describing our universe. We plan to continue developing this approach with the goal of eventually scaling up to the 4D $SU(3)$ gauge theory of QCD.

Acknowledgments

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. This research was performed using 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. Results presented in this research were obtained using the Python (Van Rossum & Drake, 2012), programming language and its many data science libraries (Hunter, 2007; Harris et al., 2020; Abadi et al., 2016; Waskom et al., 2017; Pérez & Granger, 2007; Kumar et al., 2019)

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2016. URL http://arxiv.org/abs/1603.04467.

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. URL http://arxiv.org/abs/2101.08176.

M. S. Albergo, G. Kanwar, and P. E. Shanahan. Flow-based generative models for markov chain monte carlo in lattice field theory. Physical Review D, 100(3), Aug 2019. ISSN 2470-0029. doi: 10.1103/physrevd.100.034515. URL http://dx.doi.org/10.1103/PhysRevD.100.034515.

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, 2020. ISSN 23318422. URL https://arxiv.org/abs/2008.05456.

Guido Cossu, Peter Boyle, Norman Christ, Chulwoo Jung, Andreas Jüttner, and Francesco Sanfilippo. Testing algorithms for critical slowing down. In arXiv, volume 175, pp. 2008. EDP Sciences, 2017. URL https://arxiv.org/abs/1710.07036.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, may 2017. URL http://arxiv.org/abs/1605.08803.

Matteo Favoni, Andreas Ipp, David I. Müller, and Daniel Schuh. Lattice gauge equivariant convolutional neural networks. arXiv preprint arXiv:2012.12901, 2020. URL http://arxiv.org/abs/2012.12901.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi,

Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, sep 2020. ISSN 14764687. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

W. K. Hastings. Monte carlo sampling methods using Markov chains and their applications. Biometrika, 57(1):97–109, 1970. ISSN 00063444. doi: 10.1093/biomet/57.1.97.

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, 2019. ISSN 23318422. URL https://arxiv.org/abs/1903.03704.

John D. Hunter. Matplotlib: A 2D graphics environment. Computing in Science and Engineering, 9(3):90–95, 2007. ISSN 15219615. doi: 10.1109/MCSE.2007.55.

Gurtej Kanwar, Michael S. Albergo, Denis Boyda, Kyle Cranmer, Daniel C. Hackett, Sébastien Racanière, Danilo Jimenez Rezende, and Phiala E. Shanahan. Equivariant flow-based sampling for lattice gauge theory. Physical Review Letters, 125(12), Sep 2020. ISSN 1079-7114. doi: 10.1103/physrevlett.125.121601. URL http://dx.doi.org/10.1103/PhysRevLett.125.121601.

Ravin Kumar, Colin Carroll, Ari Hartikainen, and Osvaldo Martin. Arviz a unified library for exploratory analysis of bayesian models in python. Journal of Open Source Software, 4 (33):1143, 2019. doi: 10.21105/joss.01143. URL https://doi.org/10.21105/joss.01143.

Daniel Lévy, Matthew D. Hoffman, Jascha Sohl-Dickstein, Daniel Levy, Matthew D. Hoffman, and Jascha Sohl-Dickstein. Generalizing Hamiltonian Monte Carlo with neural networks. arXiv, abs/1711.0, nov 2017. ISSN 23318422. URL http://arxiv.org/abs/1711.09268.

Zengyi Li, Yubei Chen, and Friedrich T. Sommer. A neural network mcmc sampler that maximizes proposal entropy. Entropy, 23(3):1–18, 2021. doi: 10.3390/e23030269. URL https://arxiv.org/abs/2010.03587.

Matija Medvidovic, Juan Carrasquilla, Lauren E. Hayward, and Bohdan Kulchytskyy. Generative models for sampling of lattice field theories. arXiv preprint arXiv:2012.01442, 2020. URL http://arxiv.org/abs/2012.01442.

Kirill Neklyudov and Max Welling. Orbital MCMC. arXiv preprint arXiv:2010.08047, 2020. URL http://arxiv.org/abs/2010.08047.

Kirill Neklyudov, Max Welling, Evgenii Egorov, and Dmitry Vetrov. Involutive mcmc: A unifying framework. In arXiv, pp. 7273–7282. PMLR, 2020. URL https://arxiv.org/abs/2006.16653.

Cristian Pasarica and Andrew Gelman. Adaptively scaling the metropolis algorithm using expected squared jumped distance. Statistica Sinica, 20(1):343–364, 2010. ISSN 10170405. doi: 10.2139/ssrn.1010403. URL http://www.jstor.org/stable/24308995.

Fernando Pérez and Brian E. Granger. IPython: A system for interactive scientific computing. Computing in Science and Engineering, 9(3):21–29, 2007. ISSN 15219615. doi: 10.1109/MCSE.2007.53.

Danilo Jimenez Rezende, George Papamakarios, Sébastien Racanière, Michael S. Albergo, Gurtej Kanwar, Phiala E. Shanahan, and Kyle Cranmer. Normalizing flows on tori and spheres. In arXiv, pp. 8083–8092. PMLR, 2020. URL https://arxiv.org/abs/2002.02428.

Stefan Schaefer, Rainer Sommer, and Francesco Virotta. Investigating the critical slowing down of QCD simulations. Proceedings of Science, 91, 2009. ISSN 18248039. doi: 10.22323/1.091.0032.

Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in tensorflow. arXiv, 2018. ISSN 23318422. URL https://arxiv.org/abs/1802.05799.

Akinori Tanaka and Akio Tomiya. Towards reduction of autocorrelation in HMC by machine learning. arXiv, 2017. ISSN 23318422. URL https://arxiv.org/abs/1712.03893.

Akio Tomiya and Yuki Nagai. Gauge covariant neural network for 4 dimensional non-abelian gauge theory, 2021.

Peter Toth, Danilo Jimenez Rezende, Andrew Jaegle, Sébastien Racanière, Aleksandar Botev, and Irina Higgins. Hamiltonian generative networks, 2020.

Guido Van Rossum and Fred L Drake. Python Tutorial. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 2012.

Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, and Adel Qalieh. mwaskom/seaborn: v0.8.1 (September 2017), sep 2017. URL https://doi.org/10.5281/zenodo.883859.

Antoine Wehenkel and Gilles Louppe. You say Normalizing Flows I see Bayesian Networks. arXiv, 2020. ISSN 23318422. URL https://arxiv.org/abs/2006.00866.

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 a normalization factor (the partition function). 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 $p(x, v) = p(x)p(v) \propto e^{-S(x)} e^{-\frac{1}{2} v^T v} = e^{-\mathcal{H}(x,v)}$, 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 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: $v^{1/2} \equiv v\big(t + \frac{\varepsilon}{2}\big) = v - \frac{\varepsilon}{2} \partial_x S(x)$
2. Full-step position update: $x' \equiv x(t + \varepsilon) = x + \varepsilon v^{1/2}$
3. Half-step momentum update: $v' \equiv v(t + \varepsilon) = v^{1/2} - \frac{\varepsilon}{2} \partial_x S(x')$

We can then construct a complete trajectory of length $\lambda = \varepsilon N_{\text{LF}}$ 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,

$$x_{i+1} = \begin{cases} x' & \text{with probability } A(\xi'|\xi) \\ x & \text{with probability } (1 - A(\xi'|\xi)), \end{cases} \quad \text{and} \quad A(\xi'|\xi) = \min\left\{1, \frac{p(\xi')}{p(\xi)} \left|\frac{\partial \xi'}{\partial \xi^T}\right|\right\}. \tag{7}$$

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

B 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 $\gamma_0 < \gamma_1 < \cdots < \gamma_N \equiv 1$, $\gamma_{t+1} - \gamma_t \ll 1$. 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

$$p_t(x) \propto e^{-\gamma_t S(x)}, \quad \text{for} \quad t = 0, 1, \ldots, N \tag{8}$$

C Training Algorithm

Algorithm 1: Training procedure

input :

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

Initialize weights θ

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

- update $p_t(x) \propto e^{-\gamma_t S_\beta(x)}$
- resample $v \sim \mathcal{N}(0, \mathbb{1})$
- resample $d \sim \mathcal{U}(+, -)$
- construct $\xi_0 \equiv (x_0, v_0, d_0)$
- **for** $0 \leq k < N_{\text{LF}}$:
  - propose (leapfrog layer) $\xi'_k \leftarrow \xi_k$
- 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 $x_{t+1} \leftarrow \begin{cases} x' & \text{with probability } A(\xi'|\xi) \\ x & \text{with probability } (1 - A(\xi'|\xi)). \end{cases}$