Overview of the L2HMC Algorithm*

Sam Foreman†
James Osborn‡
Xiao-Yong Jin§

November 24, 2020

1 | Introduction

We describe a new technique for performing Hamiltonian Monte-Carlo (HMC) simulations called: 'Learning to Hamiltonian Monte Carlo' (L2HMC) [1] which expands upon the traditional HMC by using a generalized version of the leapfrog integrator that is parameterized by weights in a neural network. Hamiltonian Monte-Carlo improves upon the random-walk guess and check strategy of generic MCMC by integrating Hamilton's equations along approximate iso-probability contours of phase space. In doing so, we are able to explore the phase space more efficiently by taking larger steps between proposed configurations while maintaining high acceptance rates. In order to demonstrate the usefulness of this new approach, we use various metrics for measuring the performance of the trained (L2HMC) sampler vs. the generic HMC sampler.

First, we will look at applying this algorithm to a two-dimensional Gaussian Mixture Model (GMM). The GMM is a notoriously difficult distribution for HMC due to the vanishingly small likelihood of the leapfrog integrator traversing the space between the two modes. Conversely, we see that through the use of a carefully chosen training procedure, the trained L2HMC sampler is able to successfully discover the existence of both modes, and mixes ('tunnels') between the two with ease. Additionally, we will observe that the trained L2HMC sampler mixes much faster than the generic HMC sampler, as evidenced through their respective autocorrelation spectra.

This ability to reduce autocorrelations is an important metric for measuring the efficiency of a general MCMC algorithm, and is of great importance for simulations in lattice gauge theory and lattice QCD. Following this, we introduce the two-dimensional $U(1)$ lattice gauge theory and describe important modifications to the algorithm that are of particular relevance for lattice models. Ongoing issues and potential areas for improvement are also discussed, particularly within the context of high-performance computing and long-term goals of the lattice QCD community.

2 | Hamiltonian Monte Carlo

We can improve upon the random-walk guess and check strategy of the generic Markov Chain Monte Carlo algorithm by "guiding" the simulation according to the systems natural dynamics using a method known as Hamiltonian (Hybrid) Monte Carlo (HMC).

In HMC, model samples can be obtained by simulating a physical system governed by a Hamiltonian comprised of kinetic and potential energy functions that govern a particles dynamics. By transforming the density function to a potential energy function and introducing the auxiliary momentum variable v, HMC lifts the target distribution onto a joint probability distribution in phase space (x, v), where x is the original variable

*Source code can be found at: https://github.com/saforem2/l2hmc-qcd

†foremans@anl.gov, Argonne National Laboratory
‡osborn@alcf.anl.gov, Argonne National Laboratory
§xjin@anl.gov, Argonne National Laboratory

of interest (e.g. position in Euclidean space). A new state is then obtained by solving the equations of motion for a fixed period of time using a volume-preserving integrator (most commonly the *leapfrog integrator*). The addition of random (typically normally distributed) momenta encourages long-distance jumps in state space with a single Metropolis-Hastings (MH) step.

Let the 'position' of the physical state be denoted by a vector $x \in \mathbb{R}^n$ and the conjugate momenta of the physical state be denoted by a vector $v \in \mathbb{R}^n$. Then the Hamiltonian reads

$$H(x, v) = U(x) + K(v) \tag{1}$$

$$= U(x) + \frac{1}{2} v^T v, \tag{2}$$

where $U(x)$ is the potential energy, and $K(v) = \frac{1}{2} v^T v$ the kinetic energy.We assume without loss of generality that the position and momentum variables are independently distributed. That is, we assume the target distribution of the system can be written as $\pi(x, v) = \pi(x)\pi(v)$. Further, instead of sampling $\pi(x)$ directly, HMC operates by sampling from the canonical distribution $\pi(x, v) = \frac{1}{Z} \exp(-H(x, v)) = \pi(x)\pi(v)$, for some partition function Z that provides a normalization factor. Additionally, we assume the momentum is distributed according to an identity-covariance Gaussian given by $\pi(v) \propto \exp\left\{(-\frac{1}{2} v^T v)\right\}$ For convenience, we will denote the combined state of the system by $\xi \equiv (x, v)$. From this augmented state ξ, HMC produces a proposed state $\xi' = (x', v')$ by approximately integrating Hamiltonian dynamics jointly on x and v. This integration is performed along approximate iso-probability contours of $\pi(x, v) = \pi(x)\pi(v)$ due to the Hamiltonians energy conservation.

2.1 Hamiltonian Dynamics

One of the characteristic properties of Hamilton's equations is that they conserve the value of the Hamiltonian. Because of this, every Hamiltonian trajectory is confined to an energy *level set*,

$$H^{(-1)}(E) = \{x, v | H(x, v) = E\}. \tag{3}$$

Our state $\xi = (x, v)$ then proceeds to explore this level set by integrating Hamilton's equations, which are shown as a system of differential equations in Eq. 5.

$$\dot{x}_i = \frac{\partial H}{\partial v_i} = v_i \tag{4}$$

$$\dot{v}_i = -\frac{\partial H}{\partial x_i} = -\frac{\partial U}{\partial x_i} \tag{5}$$

It can be shown [2] that the above transformation is volume-preserving and reversible, two necessary factors to guarantee asymptotic convergence of the simulation to the target distribution. The dynamics are simulated using the leapfrog integrator, which for a single time step consists of:

$$v^{\frac{1}{2}} = v - \frac{\varepsilon}{2} \partial_x U(x) \tag{6}$$

$$x' = x + \varepsilon v^{\frac{1}{2}} \tag{7}$$

$$v' = v - \frac{\varepsilon}{2} \partial_x U(x'). \tag{8}$$

We write the action of the leapfrog integrator in terms of an operator $\mathbf{L} : \mathbf{L}\xi \equiv \mathbf{L}(x, v) \equiv (x', v')$, and introduce a momentum flip operator $\mathbf{F} : \mathbf{F}(x, v) \equiv (x, -v)$. The Metropolis-Hastings acceptance probability for the HMC proposal is given by:

$$A(\mathbf{FL}\xi|\xi) = \min\left(1, \frac{\pi(\mathbf{FL}\xi)}{\pi(\xi)} \left| \frac{\partial\left[\mathbf{FL}\xi\right]}{\partial \xi^T} \right| \right), \tag{9}$$

Where $\left|\frac{\partial[\mathbf{FL}\xi]}{\partial\xi^T}\right|$ denotes the determinant of the Jacobian describing the transformation, and is equal to 1 for traditional HMC. In order to utilize these Hamiltonian trajectories to construct an efficient Markov transition, we need a mechanism for introducing momentum to a given point in the target parameter space.

Fortunately, this can be done by exploiting the probabilistic structure of the system [3]. To lift an initial point in parameter space into one on phase space, we simply sample from the conditional distribution over the momentum,

$$v \sim \pi(x|v). \tag{10}$$

Sampling the momentum directly from the conditional distribution ensures that this lift will fall into the typical set in phase space. We can then proceed to explore the joint typical set by integrating Hamilton's equations as demonstrated above to obtain a new configuration $\xi \rightarrow \xi'$. We can then return to the target parameter space by simply projecting away the momentum,

$$(x, v) \rightarrow x \tag{11}$$

These three steps when performed in series gives a complete Hamiltonian Markov transition composed of random trajectories that rapidly explore the target distribution, as desired. An example of this process can be seen in Fig 1.

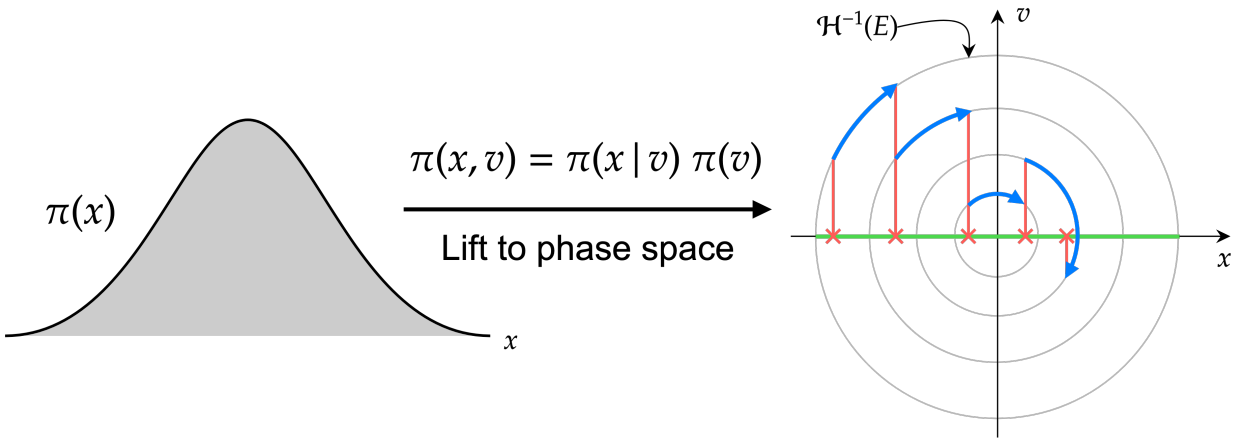


Figure 1: *Visualizing HMC for a 1D Gaussian* (example from [3], figure adapted with permission from [4]). Each Hamiltonian Markov transition lifts the initial state onto a random level set of the Hamiltonian, $H^{(-1)}(E)$, which can then be explored with a Hamiltonian trajectory before projecting back down to the target parameter space.

2.1.1 Properties of Hamiltonian Dynamics

There are three fundamental properties of Hamiltonian dynamics which are crucial to its use in constructing Markov Chain Monte Carlo updates.

1. **Reversibility:** Hamiltonian dynamics are *reversible* — the mapping from $\mathbf{L} : \xi(t) \rightarrow \xi' = \xi(t + s)$ is one-to-one, and consequently has an inverse $\mathbf{L}^{-1}$, obtained by negating the time derivatives in Eq. 5.

2. **Conservation of the Hamiltonian:** Moreover, the dynamics *keeps the Hamiltonian invariant*.

3. **Volume preservation:** The final property of Hamiltonian dynamics is that it *preserves volume* in (x, v) phase space (i.e. Liouville's Theorem).

All in all, HMC offers noticeable improvements compared to the 'random-walk' approach of generic MCMC, but tends to perform poorly on high-dimensional distributions. This becomes immediately apparent when it is used for simulations in lattice gauge theory and lattice QCD, where large autocorrelations and slow 'burn-in' can become prohibitively expensive.

3 | Generalizing the Leapfrog Integrator

As in the HMC algorithm, we start by augmenting the current state $x \in \mathbb{R}^n$ with a continuous momentum variable $v \in \mathbb{R}^n$ drawn from a standard normal distribution. Additionally, we introduce a binary direction variable $d \in \{-1, 1\}$, drawn from a uniform distribution. The complete augmented state is then denoted by $\xi \equiv (x, v, d)$, with probability density $p(\xi) = p(x)p(v)p(d)$. To improve the overall performance of our model, for each step t of the leapfrog operator $\mathbf{L}_\theta$, we assign a fixed random binary mask $m^t \in \{0, 1\}^n$ that will determine which variables are affected by each sub-update. The mask m^t is drawn uniformly from the set of binary vectors satisfying $\sum_{i=1}^n m_i^t = \lfloor \frac{n}{2} \rfloor$, i.e. half the entries of m^t are 0 and half are 1. Additionally, we write $\bar{m}^t = \mathbb{1} - m^t$ and $x_{m^t} = x \odot m^t$, where $\odot$ denotes element-wise multiplication, and $\mathbb{1}$ the vector of 1's in each entry.

We begin with a subset of the augmented space, $\zeta_1 \equiv (x, \partial_x U(x), t)$, independent of the momentum v. We introduce three new functions of ζ_1: T_v, Q_v, and S_v. We can then perform a single time-step of our modified leapfrog integrator $\mathbf{L}_\theta$.

First, we update the momentum v, which depends only on the subset ζ_1. This update is written

$$v' = v \odot \underbrace{\exp\left(\frac{\varepsilon}{2} S_v(\zeta_1)\right)}_{\textbf{Momentum scaling}} - \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \underbrace{\exp(\varepsilon Q_v(\zeta_1))}_{\textbf{Gradient scaling}} + \underbrace{T_v(\zeta_1)}_{\textbf{Translation}}\right] \tag{12}$$

and the corresponding Jacobian is given by: $\exp\left\{\left(\frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta_1)\right)\right\}$. Next, we update x by first updating a subset of the coordinates of x (determined according to the mask m^t), followed by the complementary subset (determined from $\bar{m}^t$). The first update affects only x_{m^t} and produces x'. This update depends only on the subset $\zeta_2 \equiv (x_{\bar{m}^t}, v, t)$. Following this, we perform the second update which only affects $x'_{\bar{m}^t}$ and depends only on $\zeta_3 \equiv (x'_{m^t}, v, t)$, to produce x'':

$$x' = x_{\bar{m}^t} + m^t \odot \left[x \odot \exp\{(\varepsilon S_x(\zeta_2))\} + \varepsilon \left(v' \odot \exp\{(\varepsilon Q_x(\zeta_2))\} + T_x(\zeta_2)\right)\right] \tag{13}$$

$$x'' = x'_{m^t} + \bar{m}^t \odot \left[x' \odot \exp\{(\varepsilon S_x(\zeta_3))\} + \varepsilon \left(v' \odot \exp\{(\varepsilon Q_x(\zeta_3))\} + T_x(\zeta_3)\right)\right]. \tag{14}$$

with Jacobians: $\exp\{(\varepsilon m^t \cdot S_x(\zeta_2))\}$, and $\exp\left\{(\varepsilon \bar{m}^t \cdot S_x(\zeta_3))\right\}$, respectively. Finally, we proceed to update v again, using the subset $\zeta_4 \equiv (x'', \partial_x U'', t)$:

$$v'' = v' \odot \exp\left(\frac{\varepsilon}{2} S_v(\zeta_4)\right) - \frac{\varepsilon}{2}\left[\partial_x U \odot \exp(\varepsilon Q_v(\zeta_4)) + T_v(\zeta_4)\right]. \tag{15}$$

In order to build some intuition about each of these terms, we discuss below some of the subtleties contained in this approach and how they are (carefully) dealt with.

The first thing to notice about these equations is that if $S_i = Q_i = T_i = 0$ $(i = x, v)$, we recover the previous equations for the generic leapfrog integrator (as we would expect since we are attempting to *generalize* HMC). We can also see a similarity between the equations for updating v and those for updating x: each update is generalized by *scaling* the previous value (v or x), and *scaling and translating* the updating value (either $\partial_x U(x)$ or x). It can be shown [1], that the scaling applied to the momentum in Eq 12 can enable, among other things, acceleration in low-density zones to facilitate mixing between modes, and that the scaling term applied to the gradient may allow better conditioning of the energy landscape (e.g., by learning a diagonal inertia tensor), or partial ignoring of the energy gradient for rapidly oscillating energies.

Second, note that because the determinant of the Jacobian appears in the Metropolis-Hastings (MH) acceptance probability, we require the Jacobian of each update to be efficiently computable (i.e. independent of the variable actually being updated). For each of the momentum updates, the input is a subset $\zeta = (x, \partial_x U(x), t)$ of the augmented space and the associated Jacobian is $\exp\left\{\left(\frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta)\right)\right\}$ which is independent of v as desired. For the position updates however, things are complicated by the fact that the input ζ is x-dependent. In order to ensure that the Jacobian of the x update is efficiently computable, it is necessary to break the update into two parts following the approach outlined in *Real-valued Non-Volume Preserving transformations (RealNVP)* [5].

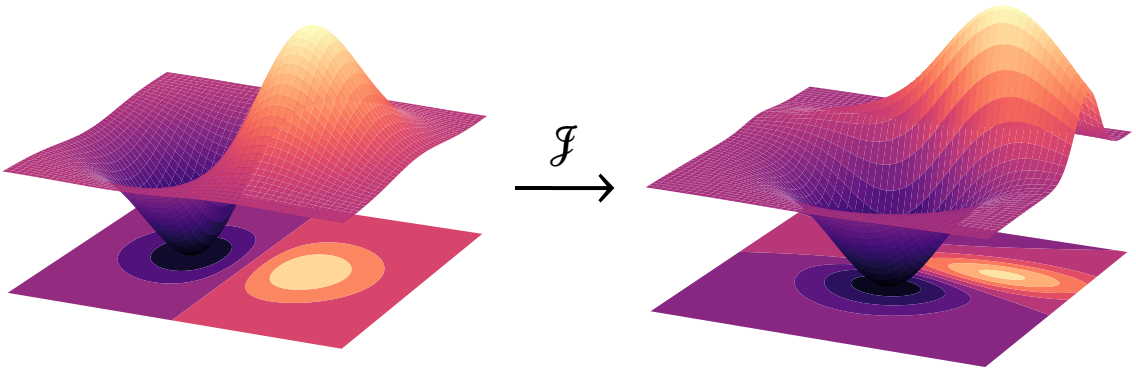


Figure 2: Example of how the determinant of the Jacobian can deform the energy landscape.

3.1 Metropolis-Hastings Accept/Reject

Written in terms of these transformations, the augmented leapfrog operator $\mathbf{L}_\theta$ consists of M sequential applications of the single-step leapfrog operator $\mathbf{L}_\theta \xi = \mathbf{L}_\theta(x, v, d) = (x''^{\times M}, v''^{\times M}, d)$, followed by the previously-defined momentum flip operator $\mathbf{F}$ which flips the direction variable d, i.e. $\mathbf{F}\xi = (x, v, -d)$. Using these, we can express a complete molecular dynamics update step as $\mathbf{FL}_\theta \xi = \xi'$, where now the Metropolis-Hastings acceptance probability for this proposal is given by

$$A(\mathbf{FL}\xi|\xi) = \min\left(1, \frac{p(\mathbf{FL}\xi)}{p(\xi)}\left|\frac{\partial\,[\mathbf{FL}\xi]}{\partial \xi^T}\right|\right), \tag{16}$$

Where $\left|\frac{\partial[\mathbf{FL}\xi]}{\partial \xi^T}\right|$ denotes the determinant of the Jacobian describing the transformation.

In contrast to generic HMC where $\left|\frac{\partial[\mathbf{FL}\xi]}{\partial \xi^T}\right| = 1$, we now have non-symplectic transformations (i.e. non-volume preserving) and so we must explicitly account for the determinant of the Jacobian. These non-volume preserving transformations have the effect of deforming the energy landscape, which, depending on the nature of the transformation, may allow for the exploration of regions of space which were previously inaccessible.

To simplify our notation, introduce an additional operator $\mathbf{R}$ that re-samples the momentum and direction, e.g. given $\xi = (x, v, d)$, $\mathbf{R}\,\xi = (x, v', d')$ where $v' \sim \mathcal{N}(0, I)$, $d' \sim \mathcal{U}(\{-1, 1\})$. A complete sampling step of our algorithm then consists of the following two steps:

1. $\xi' = \mathbf{FL}_\theta\, \xi$ with probability $A(\mathbf{FL}_\theta\, \xi|\xi)$ otherwise $\xi' = \xi$.
2. $\xi' = \mathbf{R}\,\xi$.

Note however, that for MH to be well-defined, this deterministic operator must be *invertible* and *have a tractable Jacobian* (i.e. we can compute its determinant). In order to make this operator invertible, we augment the state space (x, v) into (x, v, d), where $d \in \{-1, 1\}$ is drawn with equal probability and represent the direction of the update. All of the previous expressions for the augmented leapfrog updates represent the forward ($d = 1$) direction. We can derive the expressions for the backward direction ($d = -1$) by reversing the order of the updates (i.e. $v'' \to v'$, then $x'' \to x'$, followed by $x' \to x$ and finally $v' \to v$). For completeness, we include in Sec. 4 and Sec 5 all of the equations (both forward and backward directions) relevant for updating the variables of interest in our augmented leapfrog sampler.

4 | Forward Direction $(d = 1)$:

$$v' = v \odot \exp\left\{\left(\frac{\varepsilon}{2} S_v(\zeta_1)\right)\right\} - \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \exp\{(\varepsilon Q_v(\zeta_1))\} + T_v(\zeta_1)\right] \tag{17}$$

$$x' = x_{\bar{m}^t} + m^t \odot \left[x \odot \exp\{(\varepsilon S_x(\zeta_2))\} + \varepsilon\left(v' \odot \exp\{(\varepsilon Q_x(\zeta_2))\} + T_x(\zeta_2)\right)\right] \tag{18}$$

$$x'' = x'_{m^t} + \bar{m}^t \odot \left[x' \odot \exp\{(\varepsilon S_x(\zeta_3))\} + \varepsilon\left(v' \odot \exp\{(\varepsilon Q_x(\zeta_3))\} + T_x(\zeta_3)\right)\right] \tag{19}$$

$$v'' = v' \odot \exp\left\{\left(\frac{\varepsilon}{2} S_v(\zeta_4)\right)\right\} - \frac{\varepsilon}{2}\left[\partial_x U(x'') \odot \exp\{(\varepsilon Q_v(\zeta_4))\} + T_v(\zeta_4)\right] \tag{20}$$

With $\zeta_1 = (x, \partial_x U(x), t)$, $\zeta_2 = (x_{\bar{m}^t}, v, t)$, $\zeta_3 = (x'_{m^t}, v, t)$, $\zeta_4 = (x'', \partial_x U(x''), t)$.

5 | Backward Direction $(d = -1)$:

$$v' = \left\{v + \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \exp\{(\varepsilon Q_v(\zeta_1))\} + T_v(\zeta_1)\right]\right\} \odot \exp\left\{\left(-\frac{\varepsilon}{2} S_v(\zeta_1)\right)\right\} \tag{21}$$

$$x' = x_{m^t} + \bar{m}^t \odot \left[x - \varepsilon(\exp\{(\varepsilon Q_x(\zeta_2))\} \odot v' + T_x(\zeta_2))\right] \odot \exp\{(-\varepsilon S_x(\zeta_2))\} \tag{22}$$

$$x'' = x_{\bar{m}^t} + m^t \odot \left[x' - \varepsilon(\exp\{(\varepsilon Q_x(\zeta_3))\} \odot v' + T_x(\zeta_3))\right] \odot \exp\{(-\varepsilon S_x(\zeta_3))\} \tag{23}$$

$$v'' = \left\{v' + \frac{\varepsilon}{2}\left[\partial_x U(x'') \odot \exp\{(\varepsilon Q_v(\zeta_1))\} + T_v(\zeta_1)\right]\right\} \odot \exp\left\{\left(-\frac{\varepsilon}{2} S_v(\zeta_4)\right)\right\} \tag{24}$$

With $\zeta_1 = (x, \partial_x U(x), t)$, $\zeta_2 = (x_{m^t}, v, t)$, $\zeta_3 = (x'_{\bar{m}^t}, v, t)$, $\zeta_4 = (x'', \partial_x U(x''), t)$.

6 | Determinant of the Jacobian

In terms of the auxiliary functions S_i, Q_i, T_i, we can compute the Jacobian:

$$\log|\mathcal{J}| = \log\left|\frac{\partial[\mathbf{FL}_\theta \xi]}{\partial \xi^T}\right| \tag{25}$$

$$= d \sum_{t \le N_{\text{LF}}} \left[\frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta_1^t) + \varepsilon m^t \cdot S_x(\zeta_2^t) + \varepsilon \bar{m}^t \cdot S_x(\zeta_3^t) + \frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta_4^t)\right]. \tag{26}$$

where N_{LF} is the number of leapfrog steps, and ζ_i^t denotes the intermediary variable ζ_i^t at time step t and d is the direction of ξ, i.e. $d = 1$ (-1) for the forward (backward) update.

7 | Network Architecture

As previously mentioned, each of the functions Q, S, and T, are implemented using multi-layer perceptrons with shared weights. It's important to note that we keep separate the network responsible for paramterizing the functions used in the position updates ('X_{net}', i.e. Q_x, S_x, and T_x), and the network responsible for parameterizing the momentum updates ('V_{net}', i.e. Q_v, S_v, and T_v).

The network takes as input $\zeta_1 = (x, \partial_x U(x), t)$, where $x, v \in \mathbb{R}^n$, and t is encoded as $\tau(t) = \left(\cos\left(\frac{2\pi t}{N_{\text{LF}}}\right),\ \sin\left(\frac{2\pi t}{N_{\text{LF}}}\right)\right)$.

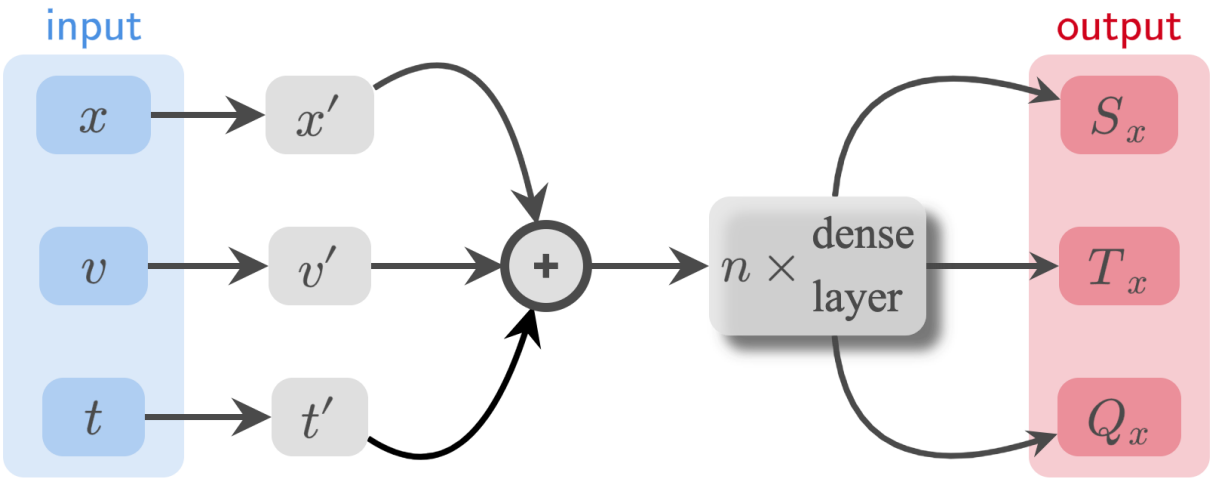


Figure 3: Illustration showing the generic (fully-connected) network architecture for training S_x, Q_x, and T_x.

Each of the inputs is then passed through a fully-connected ('dense' layer), consisting of n_h hidden units

$$\tilde{x} = W^{(x)}x + b^{(x)} \quad (\in \mathbb{R}^{n_h}) \tag{27}$$

$$\tilde{v} = W^{(v)}v + b^{(v)} \quad (\in \mathbb{R}^{n_h}) \tag{28}$$

$$\tilde{\tau} = W^{(\tau)}\tau + b^{(\tau)} \quad (\in \mathbb{R}^{n_h}). \tag{29}$$

Where $W^{(x)}, W^{(v)} \in \mathbb{R}^{n \times n_h}$, $W^{(t)} \in \mathbb{R}^{2 \times n_h}$, and $b^{(x)}$, $b^{(v)}$, $b^{(t)} \in \mathbb{R}^{n_h}$. From these, the network computes

$$h_1 = \sigma(\tilde{x} + \tilde{v} + \tilde{\tau}) \quad (\in \mathbb{R}^{n_h}). \tag{30}$$

Where $\sigma(x) = \max(0, x)$ denotes the rectified linear unit (ReLU) activation function. Next, the network computes

$$h_2 = \sigma\left(W^{(h_1)}h_1 + b^{(h_1)}\right) \quad (\in \mathbb{R}^{n_h}). \tag{31}$$

These weights (h_2) are then used to compute the network's output:

$$S_x = \lambda_S \tanh\left(W^{(S)}h_2 + b^{(S)}\right) \quad (\in \mathbb{R}^n) \tag{32}$$

$$Q_x = \lambda_Q \tanh\left(W^{(Q)}h_2 + b^{(Q)}\right) \quad (\in \mathbb{R}^n) \tag{33}$$

$$T_x = W^{(T)}h_2 + b^{(T)} \quad (\in \mathbb{R}^n), \tag{34}$$

Where $W^{(s)}, W^{(q)}$, and $W^{(T)} \in \mathbb{R}^{n_h \times n}$ and $b^{(s)}, b^{(q)}$, and $b^{(T)} \in \mathbb{R}^n$. The parameters λ_s and λ_q are additional trainable variables initialized to zero. The network used for parameterizing the functions T_v, Q_v and S_v takes as input $(x, \partial_x U(x), t)$ where again t is encoded as above. The architecture of this network is the same, and produces outputs T_v, Q_v, and S_v.

8 | Training Procedure

By augmenting traditional HMC methods with these trainable functions, we hope to obtain a sampler that has the following key properties:

1. Fast mixing (i.e. able to quickly produce uncorrelated samples).
2. Fast burn-in (i.e. rapid convergence to the target distribution).
3. Ability to mix across energy levels.
4. Ability to mix between modes.

Following the results in [6], we design a loss function with the goal of maximizing the expected squared jumped distance (or analogously, minimizing the lag-one autocorrelation). To do this, we first introduce

$$\delta(\xi, \xi') = \delta((x', v', d'), (x, v, d)) \equiv \|x - x'\|_2^2. \tag{35}$$

Then, the expected squared jumped distance is given by $\mathbb{E}_{\xi\sim p(\xi)}\left[\delta(\mathbf{FL}_\theta\xi, \xi)A(\mathbf{FL}_\theta\xi|\xi)\right]$. By maximizing this objective function, we are encouraging transitions that efficiently explore a local region of state-space, but may fail to explore regions where very little mixing occurs. To help combat this effect, we define a loss function

$$\ell_\lambda(\xi, \xi', A(\xi'|\xi)) = \frac{\lambda^2}{\delta(\xi, \xi')A(\xi'|\xi)} - \frac{\delta(\xi, \xi')A(\xi'|\xi)}{\lambda^2} \tag{36}$$

where λ is a scale parameter describing the characteristic length scale of the problem. Note that the first term helps to prevent the sampler from becoming stuck in a state where it cannot move effectively, and the second term helps to maximize the distance between subsequent moves in the Markov chain.

The sampler is then trained by minimizing ℓ_λ over both the target and initialization distributions. Explicitly, for an initial distribution π_0 over $\mathcal{X}$, we define the initialization distribution as $q(\xi) = \pi_0(x)\mathcal{N}(v; 0, I)p(d)$, and minimize

$$\mathcal{L}(\theta) \equiv \mathbb{E}_{p(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi|\xi))\right] + \lambda_b\mathbb{E}_{q(\xi)}\left[\ell_\lambda(\xi, \mathbf{FL}_\theta\xi, A(\mathbf{FL}_\theta\xi|\xi))\right]. \tag{37}$$

For completeness, we include the full algorithm [1] used to train L2HMC in Alg. 1.

Algorithm 1: Training procedure for the L2HMC algorithm.

input :

1. A (potential) energy function, $U : \mathcal{X} \to \mathbb{R}$ and its gradient $\nabla_x U : \mathcal{X} \to \mathcal{X}$
2. Initial distribution over the augmented state space, q
3. Number of iterations, N_{train}
4. Number of leapfrog steps, N_{LF}
5. Learning rate schedule, $(\alpha_t)_{t\leq N_{\text{train}}}$
6. Batch size, N_{samples}
7. Scale parameter, λ
8. Regularization strength, λ_b

Initialize the parameters of the sampler, θ
Initialize $\{\xi_{p^{(i)}}\}_{i\leq N_{\text{samples}}}$ from $q(\xi)$
for $t = 0$ **to** N_{train} **:**
  Sample a minibatch, $\left\{\xi_q^{(i)}\right\}_{i\leq N_{\text{samples}}}$ from $q(\xi)$.
  $\mathcal{L} \leftarrow 0$
  for $i = 1$ **to** N_{LF} **:**
    $\xi_p^{(i)} \leftarrow \mathbf{R}\,\xi_p^{(i)}$
    $\mathcal{L} \leftarrow \mathcal{L} + \ell_\lambda\left(\xi_p^{(i)}, \mathbf{FL}_\theta\xi_p^{(i)}, A(\mathbf{FL}_\theta\xi_p^{(i)}|\xi_p^{(i)})\right) + \lambda_b\ell_\lambda\left(\xi_q^{(i)}, \mathbf{FL}_\theta\xi_q^{(i)}, A(\mathbf{FL}_\theta\xi_q^{(i)}|\xi_q^{(i)})\right)$
    $\xi_p^{(i)} \leftarrow \mathbf{FL}_\theta\xi_p^{(i)}$ with probability $A(\mathbf{FL}_\theta\xi_p^{(i)}|\xi_p^{(i)})$
  $\theta \leftarrow \theta - \alpha_t\nabla_\theta\mathcal{L}$

9 | Gaussian Mixture Model

The Gaussian Mixture Model (GMM) is a notoriously difficult example for traditional HMC to sample accurately due to the existence of multiple modes. In particular, HMC cannot mix between modes that are reasonably separated without recourse to additional tricks. This is due, in part, to the fact that HMC cannot easily traverse the low-density zones which exist between modes.

In the most general case, we consider a target distribution described by a mixture of $M > 1$ components in $\mathbb{R}^D$ for $D \geq 1$:

$$p(\mathbf{x}) \equiv \sum_{m=1}^{M} p(m)p(\mathbf{x}|m) \equiv \sum_{m=1}^{M} \pi_m p(\mathbf{x}|m) \quad \forall\ \mathbf{x} \in \mathbb{R}^D \tag{38}$$

where $\sum_{m=1}^{M} \pi_m = 1$, $\pi_M \in (0, 1)\ \forall m = 1, \ldots, M$ and each component distribution is a normal probability distribution in $\mathbb{R}^D$. So $\mathbf{x}|m \sim \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$, where $\boldsymbol{\mu}_m \equiv \mathbb{E}_{p(\mathbf{x}|m)}\{\mathbf{x}\}$ and $\boldsymbol{\Sigma}_m \equiv \mathbb{E}_{p(\mathbf{x}|m)}\left\{(\mathbf{x} - \boldsymbol{\mu}_m)(\mathbf{x} - \boldsymbol{\mu}_m)^T\right\} > 0$ are the mean vector and covariance matrix, respectively, of component m.

9.1 Example

Consider a simple 2D case consisting of two Gaussians

$$\mathbf{x} \sim \pi_1 \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \pi_2 \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2) \tag{39}$$

with $\pi_1 = \pi_2 = 0.5$, $\boldsymbol{\mu}_1 = (-2, 0)$, $\boldsymbol{\mu}_2 = (2, 0)$ and

$$\boldsymbol{\Sigma}_1 = \boldsymbol{\Sigma}_2 = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix} \tag{40}$$

The results of trajectories generated using both traditional HMC and the L2HMC algorithm can be seen in Fig. 4. a Note that traditional HMC performs poorly and is unable to mix between the two modes, whereas L2HMC is able to correctly sample from the target distribution without getting stuck in either of the individual modes.

The L2HMC sampler was trained using simulated annealing using the schedule shown in Eq 41 with a starting temperature of $T = 10$, for 5, 000 training steps. By starting with a high temperature, the chain is able to move between both modes ('tunnel') successfully. Once it has learned this, we can lower the temperature back to $T = 1$ and recover the initial distribution while preserving information about tunneling in the networks "memory".

$$T(n) = \left(T_i - T_f\right) \cdot \left(1 - \frac{n}{N_{\text{train}}}\right) + T_f \tag{41}$$

10 | 2D $U(1)$ Lattice Gauge Theory

All lattice QCD simulations are performed at finite lattice spacing a and need an extrapolation to the continuum in order to be used for computing values of physical quantities. More reliable extrapolations can be done by simulating the theory at increasingly smaller lattice spacings. The picture that results when the lattice spacing is reduced and the physics kept constant is that all finite physical quantities of negative mass dimension diverge if measured in lattice units. In statistical mechanics language, this states that the continuum limit is a critical point of the theory since correlation lengths diverge. MCMC algorithms are known to encounter difficulties when used for simulating theories close to a critical point, an issue known as the *critical slowing down* of the algorithm. This effect is most prominent in the topological charge, whose auto-correlation time increases dramatically with finer lattice spacings. As a result, there is a growing interest in developing new sampling techniques for generating equilibrium configurations. In particular, algorithms that are able to offer improvements in efficiency through a reduction of statistical autocorrelations are highly desired. We begin with the two-dimensional $U(1)$ lattice gauge theory with dynamical variables $U_\mu(i)$ defined on the links of

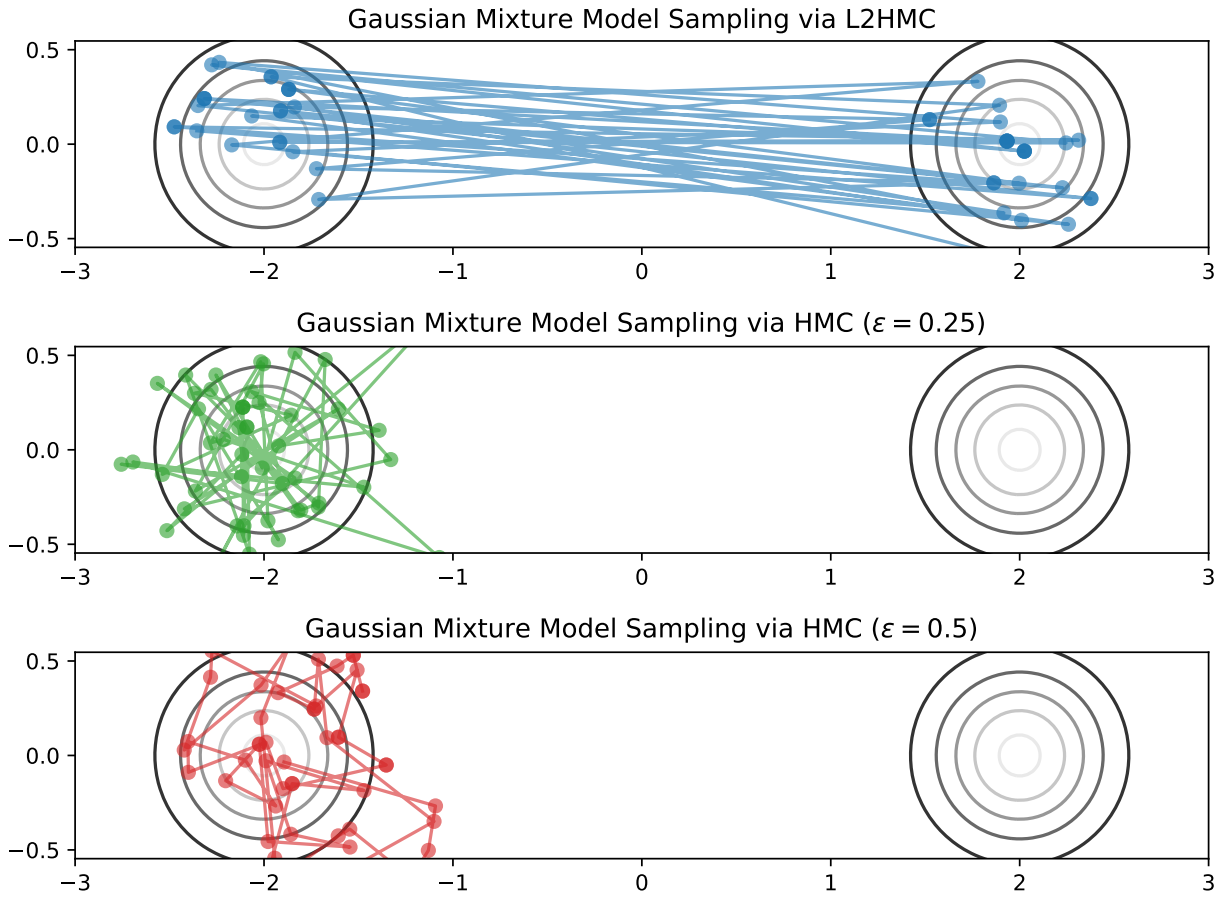


Figure 4: Comparison of trajectories generated using L2HMC (top), and traditional HMC with $\epsilon = 0.25$ (middle) and $\epsilon = 0.5$ (bottom). Note that L2HMC is able to successfully mix between modes, whereas HMC is not.

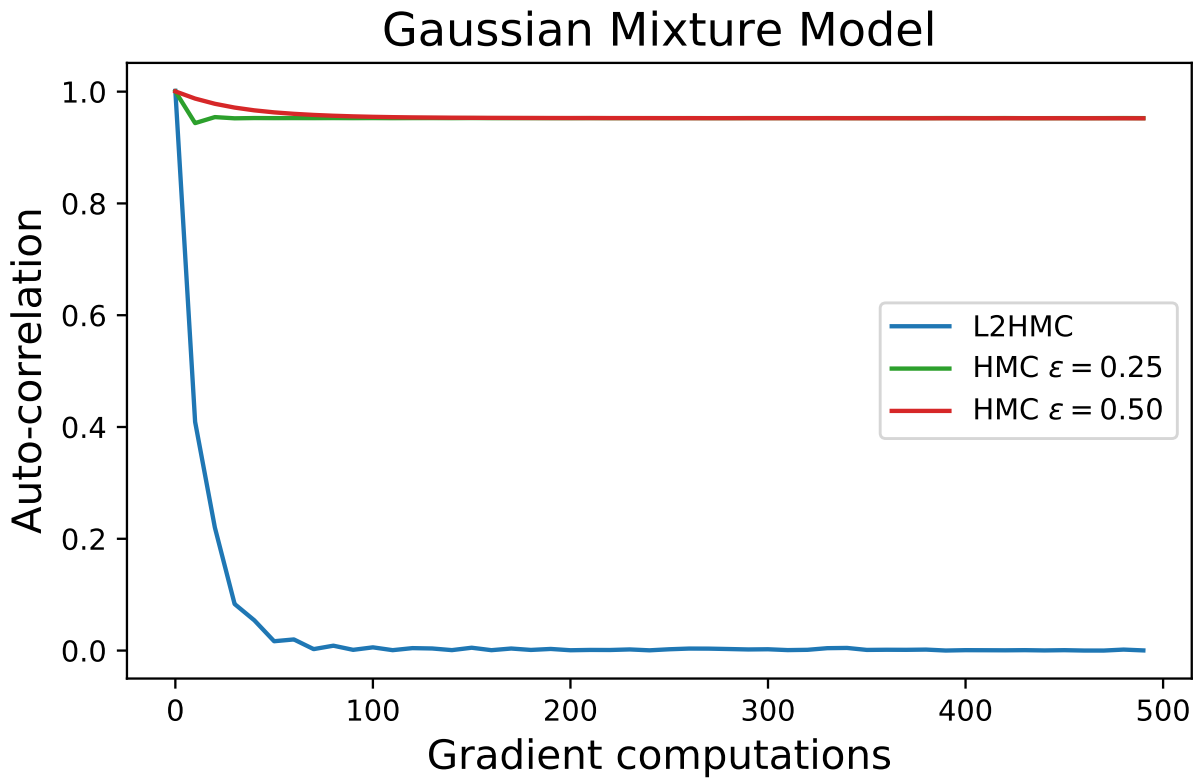


Figure 5: Autocorrelation vs. gradient evaluations (i.e. MD steps). Note that L2HMC (blue) has a significantly reduced autocorrelation after the same number of gradient evaluations when compared to either of the two HMC trajectories

a lattice, where i labels a site and μ specifies the direction. Each link $U_\mu(i)$ can be expressed in terms of an angle $0 < \phi_\mu(i) \leq 2\pi$.

$$U_\mu(i) = e^{i\phi_\mu(i)} \tag{42}$$

with the Wilson action defined as:

$$\beta S = \beta \sum_P (1 - \cos(\phi_P)) \tag{43}$$

where

$$\phi_P \equiv \phi_{\mu\nu}(i) = \phi_\mu(i) + \phi_\nu(i + \hat{\mu}) - \phi_\mu(i + \hat{\nu}) - \phi_\nu(i) \tag{44}$$

and $\beta = 1/e^2$ is the gauge coupling, and the sum $\sum_P$ runs over all plaquettes of the lattice. An illustration showing how these variables are defined for an elementary plaquette is shown in Fig. 6.

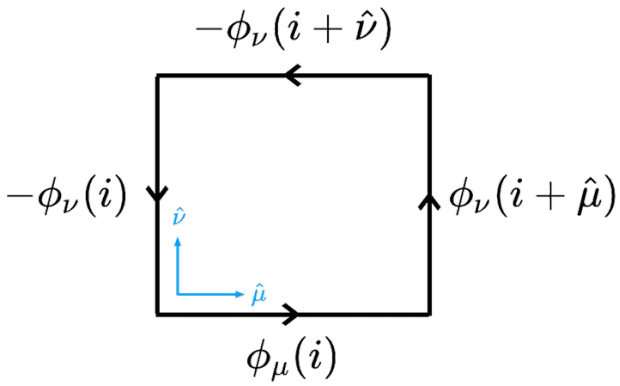


Figure 6: Illustration of an elementary plaquette on the lattice.

We can define the topological charge, $\mathbb{Q} \in \mathbb{Z}$, as

$$\mathbb{Q} \equiv \frac{1}{2\pi} \sum_P \tilde{\phi}_P = \frac{1}{2\pi} \sum_{\substack{i;\mu,\nu \\ \nu>\mu}} \tilde{\phi}_{\mu\nu}(i) \tag{45}$$

where

$$\tilde{\phi}_P \equiv \phi_P - 2\pi \left\lfloor \frac{\phi_P + \pi}{2\pi} \right\rfloor \tag{46}$$

is the sum of the link variables around the elementary plaquette, projected onto the interval $[0, 2\pi)$. From this, we can define topological susceptibility

$$\chi \equiv \frac{\langle \mathbb{Q}^2 \rangle - \langle \mathbb{Q} \rangle^2}{V} \tag{47}$$

By parity symmetry, $\langle \mathbb{Q} \rangle = 0$, so we have that

$$\chi = \frac{\langle \mathbb{Q}^2 \rangle}{V} \tag{48}$$

Unfortunately, the measurement of χ is often difficult due to the fact that the autocorrelation time with respect to $\mathbb{Q}$ tends to be extremely long. This is a consequence of the fact that the Markov chain tends to get stuck in a topological sector (characterized by $\mathbb{Q} = const.$), a phenomenon known as *topological freezing*.

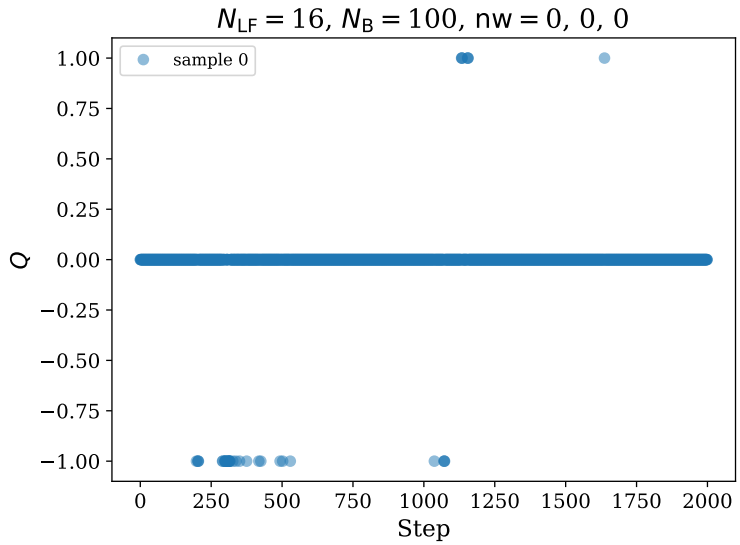


Figure 7: Example of topological freezing in the $2D$ $U(1)$ lattice gauge theory. The above result was generated using generic HMC sampling for a 8×8 lattice. Note that for the majority of the simulation $\mathbb{Q} = 0$, making it virtually impossible to get a reasonable estimate of χ.

10.1 Annealing Schedule

Proceeding as in the example of the Gaussian Mixture Model, we include a simulated annealing schedule in which the value of the gauge coupling β is continuously updated according to the annealing schedule shown in Eq. 49. This was done in order to encourage sampling from multiple different topological charge sectors, since our sampler is less 'restricted' at lower values of β.

$$\frac{1}{\beta(n)} = \left(\frac{1}{\beta_i} - \frac{1}{\beta_f}\right)\left(\frac{1-n}{N_{\text{train}}}\right) + \frac{1}{\beta_f} \tag{49}$$

Here $\beta(n)$ denotes the value of β to be used for the n^{th} training step ($n = 1, \ldots, N_{\text{train}}$), β_i represents the initial value of β at the beginning of the training, and β_f represents the final value of β at the end of training. For a typical training session, $N_{\text{train}} = 25,000$, $\beta_i = 2$ and $\beta_f = 5$.

10.2 Modified metric for $U(1)$ Gauge Model

In order to more accurately define the "distance" between two different lattice configurations, we redefine the metric in Eq. 35 to be

$$\delta(\xi, \xi') \equiv 1 - \cos(\xi - \xi') \tag{50}$$

where now $\xi \equiv \left(\phi^x_\mu(i),\ \phi^v_\mu(i),\ d\right)$, with ϕ^x_μ representing the lattice of ('position') gauge variables (what we called x previously), and ϕ^v_μ representing the lattice of ('momentum') gauge variables (what we called v previously). Note that i runs over all lattice sites[1] and $\mu = 0, 1$ for the two dimensional case. We see that this metric gives the expected behavior, since $\delta \to 0$ for $\xi \approx \xi'$.

[1] In what follows, we will refrain from explicitly including the site index and make the assumption that it implicitly extends over all sites on the lattice.

11 | Updates: 04/13/2020

11.1 Changes to Network

- Use Cartesian representation $\left[\cos\phi_\mu(x), \sin\phi_\mu(x)\right]$ instead of angular representation $\phi_\mu(x) \in [0, 2\pi)$.
  - While this doubles the size of our inputs, it avoids complications that arise from angles near 0 and 2π.

11.2 Changes to the loss function

Since our main goal is to obtain a sampler that is able to efficiently sample from different topological sectors, we can design a loss function around this idea. Recall that the topological charge $\mathbb{Q} \in \mathbb{Z}$ is computed as

$$\mathbb{Q} \equiv \frac{1}{2\pi} \sum_{\substack{x;\mu,\nu \\ \nu>\mu}} \sin\left(\phi_{\mu\nu}(x)\right) \tag{51}$$

for

$$\phi_{\mu\nu}(x) = \phi_\mu(x) + \phi_\nu(x+\hat{\mu}) - \phi_\mu(x+\hat{\nu}) - \phi_\nu(x) \tag{52}$$

Instead of maximizing the expected squared jump distance (ESJD) between configurations, it makes more sense to maximize quantities related to the plaquette sums, e.g. the *plaquette distance*, $\delta_P(\xi', \xi)$

$$\delta_P(\xi', \xi) = \sum 1 - \cos\left(\phi'_{\mu\nu}(x) - \phi_{\mu\nu}(x)\right) \tag{53}$$

or the topological charge difference squared, $\delta_{\mathbb{Q}}(\xi', \xi)$

$$\delta_{\mathbb{Q}}(\xi', \xi) = \left[\overbrace{\frac{1}{2\pi}\sum \sin\left(\phi'_{\mu\nu}(x)\right)}^{\mathbb{Q}'} - \overbrace{\frac{1}{2\pi}\sum \sin\left(\phi_{\mu\nu}(x)\right)}^{\mathbb{Q}}\right]^2 \tag{54}$$

$$= \left(\mathbb{Q}' - \mathbb{Q}\right)^2 \tag{55}$$

where $\phi'_{\mu\nu}(x)$ denotes the proposed configuration (before applying Metropolis-Hastings accept/reject). From these we can then define

$$\ell_{\lambda_P}(\xi', \xi, A(\xi'|\xi)) = \frac{\lambda_P^2}{\delta_P \cdot A(\xi'|\xi)} - \frac{\delta_P \cdot A(\xi'|\xi)}{\lambda_P^2} \tag{56}$$

$$\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi)) = \frac{\lambda_{\mathbb{Q}}^2}{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)} - \frac{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)}{\lambda_{\mathbb{Q}}^2} \tag{57}$$

where $\lambda_P, \lambda_{\mathbb{Q}}$ are scaling factors used to control the contribution from each of the $\delta_P, \delta_{\mathbb{Q}}$ terms. Finally, our loss function becomes

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}\left[\alpha_P \cdot \ell_{\lambda_P} + \alpha_{\mathbb{Q}} \cdot \ell_{\lambda_{\mathbb{Q}}}\right] + \mathbb{E}_{q(\xi)}\left[\alpha_P \cdot \ell_{\lambda_P} + \alpha_{\mathbb{Q}} \cdot \ell_{\lambda_{\mathbb{Q}}}\right] \tag{58}$$

where $\alpha_P, \alpha_{\mathbb{Q}}$ are weights to control the respective terms contribution to the overall loss function.

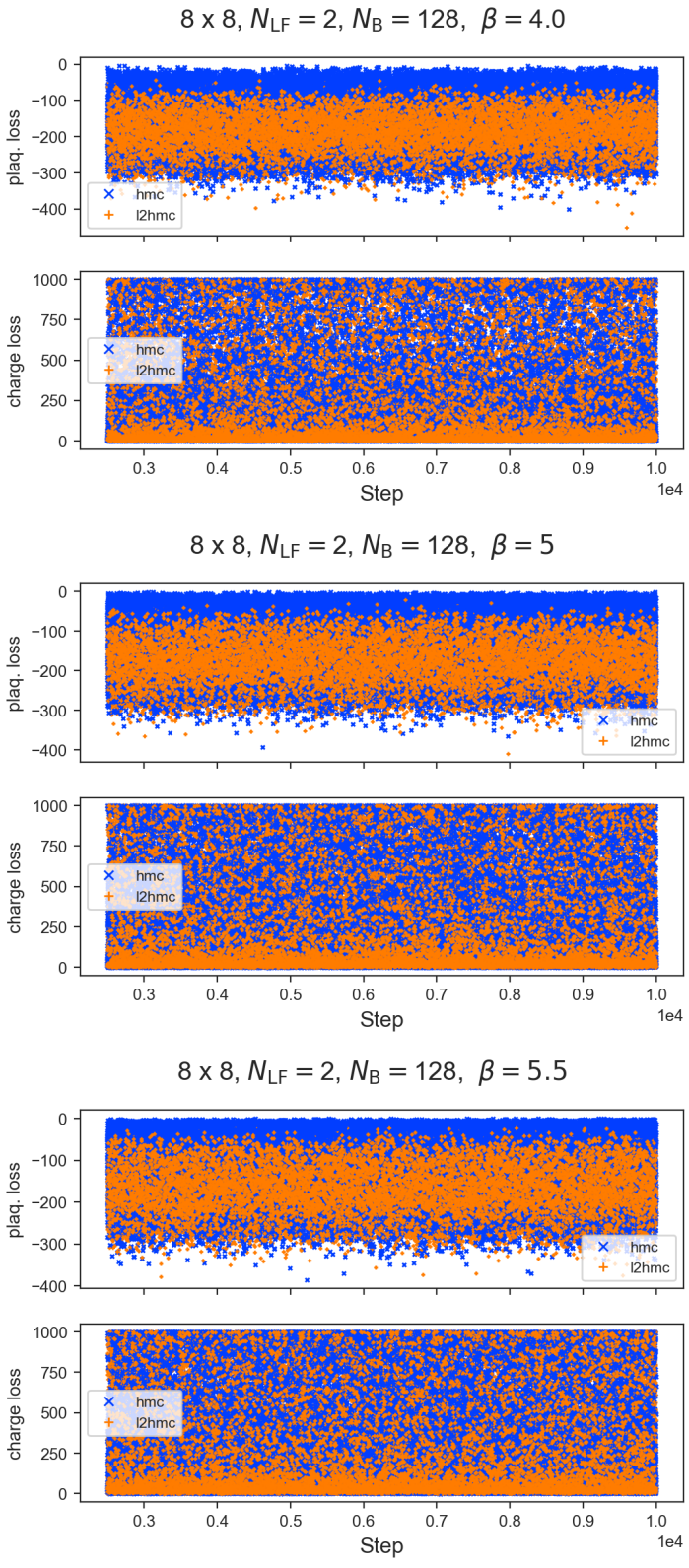


Figure 8: Loss comparisons between L2HMC and HMC at different values of β.

12 | Updates: 04/28/2020

12.1 Additional changes to loss function

Instead of working with the mixed loss function, $\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi))$ from Eq.57, we can focus exclusively on the second term, which is directly related to the tunneling rate. The equation then becomes

$$\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi)) = -\frac{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)}{\lambda^2_{\mathbb{Q}}} \tag{59}$$

$$= -\left(\frac{\mathbb{Q}' - \mathbb{Q}}{\lambda_{\mathbb{Q}}}\right)^2 \cdot A(\xi'|\xi) \tag{60}$$

13 | Non Compact Projection

Recall the expression for the x update in the augmented L2HMC sampler,

$$x' = m^t \odot x + \bar{m}^t \odot \left[x \odot \exp\left(\varepsilon S_x(\zeta_2^t)\right) + \varepsilon * \left(v \odot \exp\left(\varepsilon Q_x(\zeta_2^t)\right) + T_x(\zeta_2^t)\right)\right] \tag{61}$$

$$= m^t \odot x + \bar{m}^t \odot [\alpha x + \beta] \tag{62}$$

where $\alpha = \exp(\varepsilon S_x)$, $\beta = \varepsilon \cdot (v \odot \exp(\varepsilon Q_x) + T_x)$, depend on $\zeta_2 = (m^t \odot x, v, t)$, but are independent of $\bar{m}^t \odot x$.

Let $z = h(x) : [-\pi, \pi] \to \mathbb{R}$ be given by

$$z = \tan\left(\frac{x}{2}\right). \tag{63}$$

We can perform the affine transformation in $\mathbb{R}$ and then project back to $[-\pi, \pi]$ via $x = h^{-1}(z)$ to complete the update:

$$x' = m^t \odot x + \bar{m}^t \odot \left[2\tan^{-1}\left(\alpha \tan\left(\frac{x}{2}\right)\right) + \beta\right] \tag{64}$$

With Jacobian factor

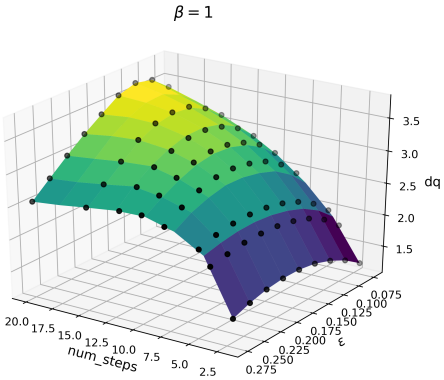
$$\frac{\partial x'}{\partial x} = \frac{\alpha}{\cos^2(\frac{x}{2}) + \alpha^2 \sin^2(\frac{x}{2})} \tag{65}$$

$$= \frac{\exp\{(\varepsilon S_x)\}}{\cos^2(\frac{x}{2}) + \exp\{(2 \cdot \varepsilon S_x)\} \sin(\frac{x}{2})} \tag{66}$$

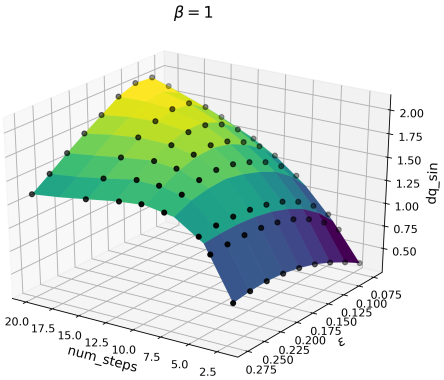
14 | Updates: 08/13/2020

14.1 HMC Results

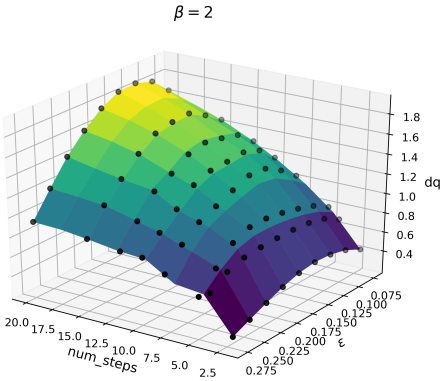
For a fixed value of β, we are interested in finding the values of $N_{\text{LF}}, \varepsilon$ that give the optimal tunneling rate, δQ.



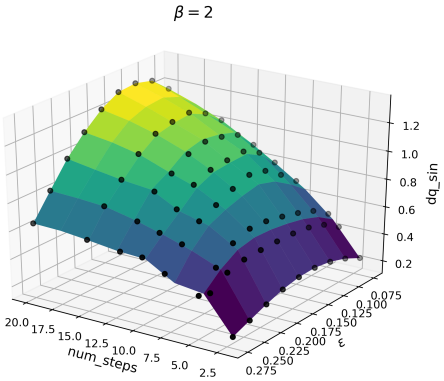
(a) δQ vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 1$



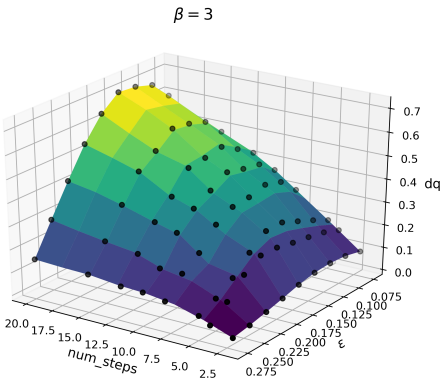
(b) δQ_{sin} vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 1$



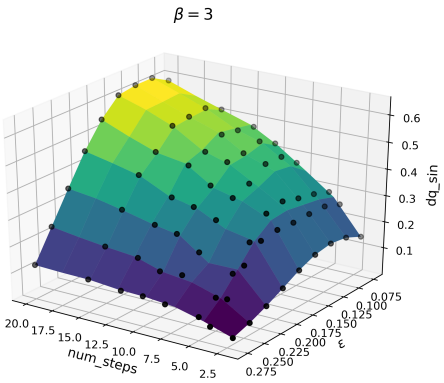
(a) δQ vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 2$



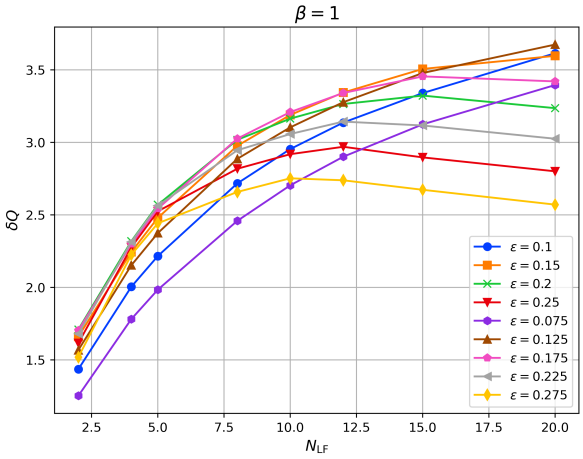
(b) δQ_{sin} vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 2$



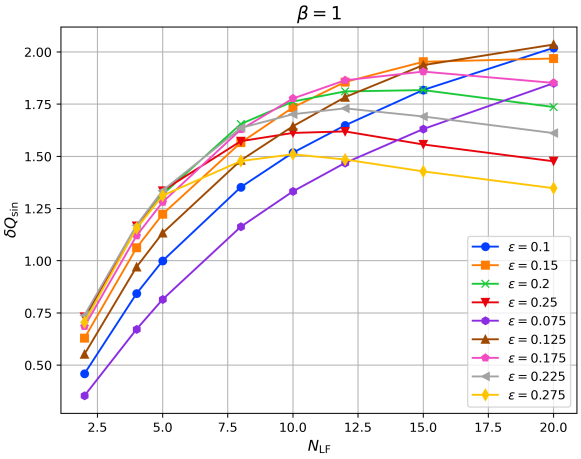
(a) δQ vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 3$



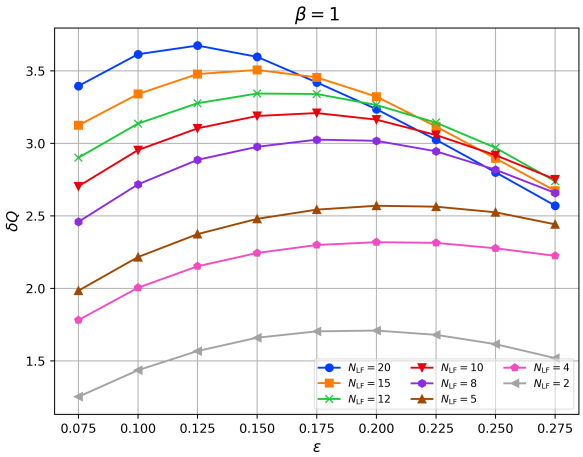
(b) δQ_{sin} vs. $N_{\text{LF}}, \varepsilon$ at $\beta = 3$



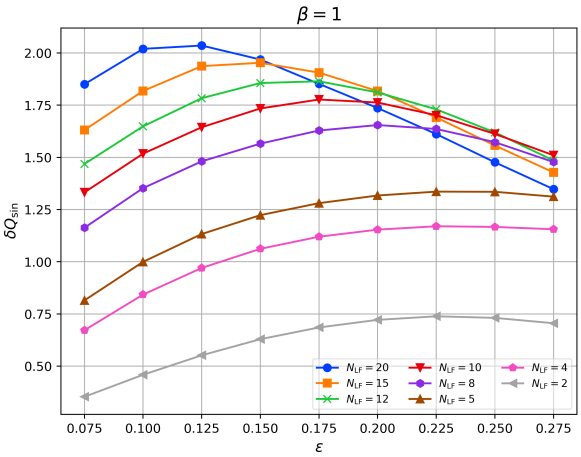
(a) δQ vs. N_{LF} at $\beta = 1$



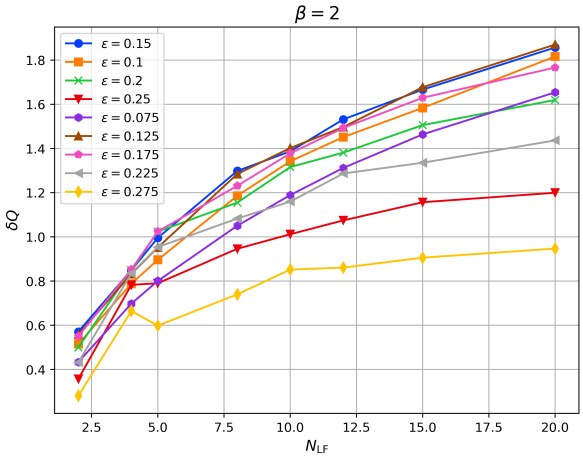
(b) δQ_{sin} vs. N_{LF} at $\beta = 1$



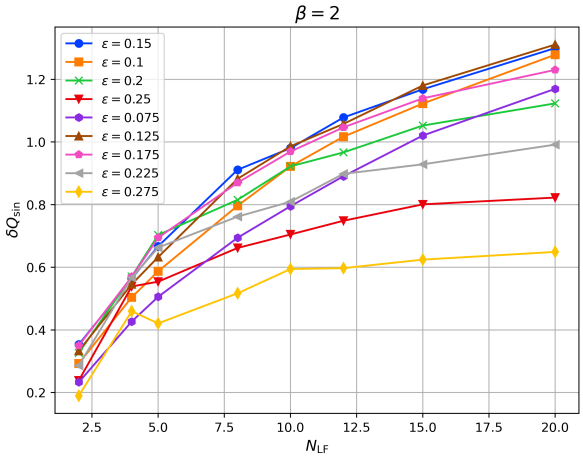
(a) δQ vs. ε at $\beta = 1$



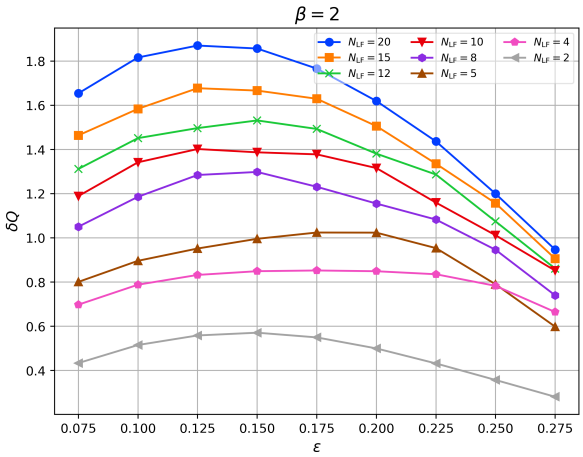
(b) δQ_{sin} vs. ε at $\beta = 1$



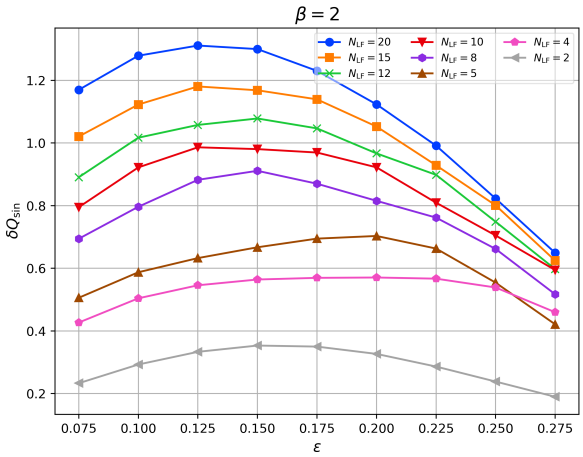
(a) δQ vs. N_{LF} at $\beta = 2$



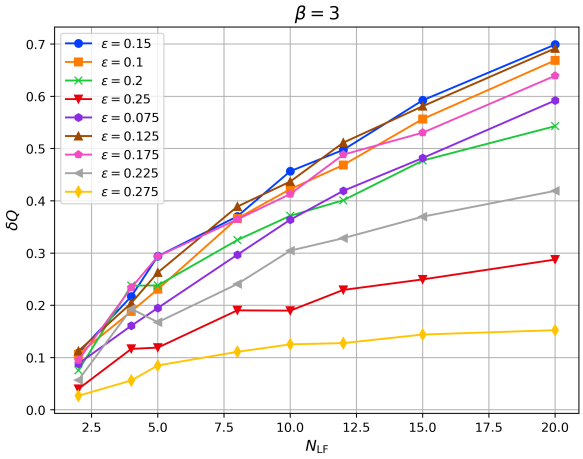
(b) δQ_{sin} vs. N_{LF} at $\beta = 2$



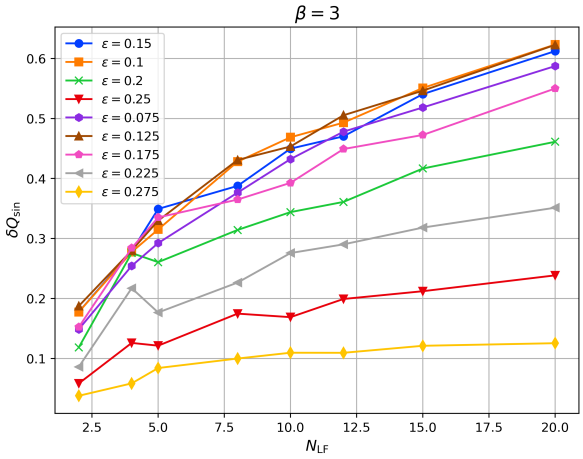
(a) δQ vs. ε at $\beta = 2$



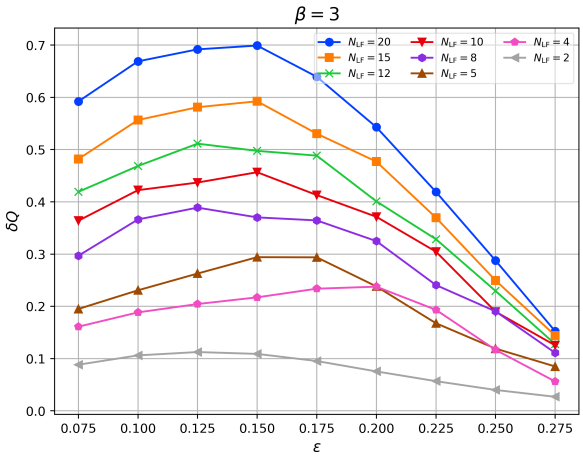
(b) δQ_{sin} vs. ε at $\beta = 2$



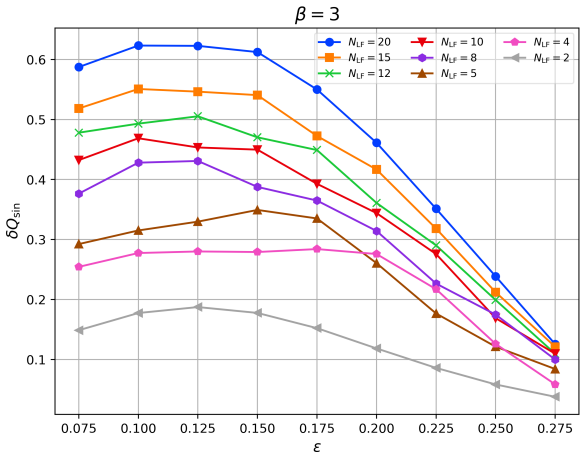
(a) δQ vs. N_{LF} at $\beta = 3$



(b) δQ_{sin} vs. N_{LF} at $\beta = 3$

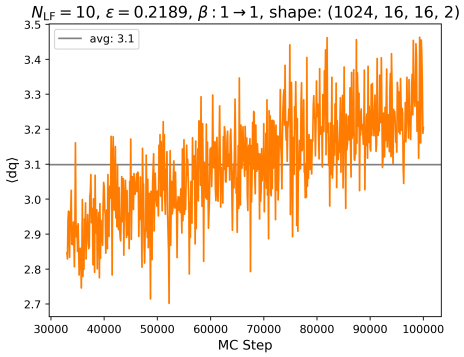


(a) δQ vs. ε at $\beta = 3$

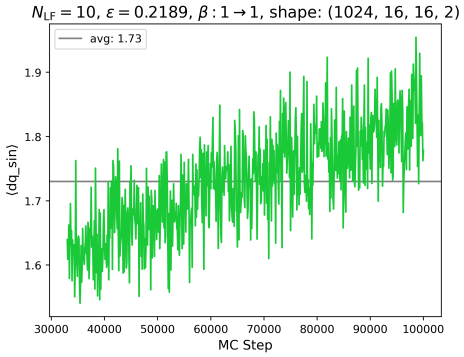


(b) δQ_{sin} vs. ε at $\beta = 3$

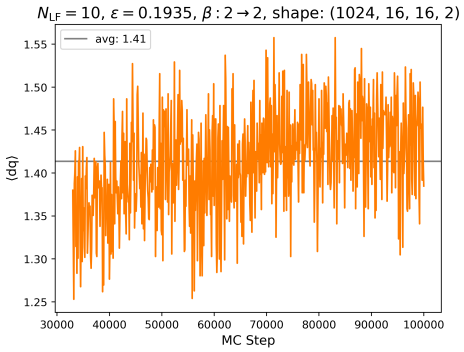
14.2 Training results



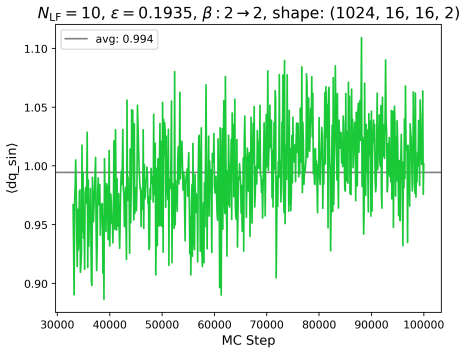
(a) $\langle \delta Q \rangle$ vs training step at $\beta = 1$



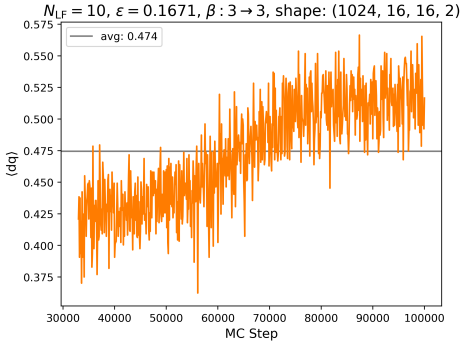
(b) $\langle \delta Q_{\sin} \rangle$ vs training step at $\beta = 1$



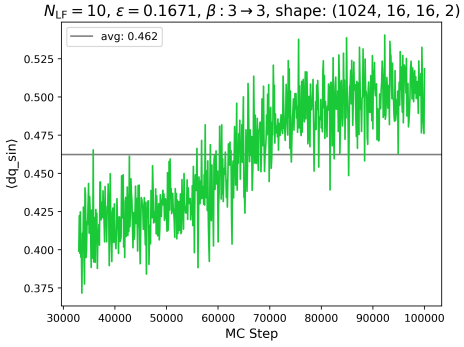
(a) $\langle \delta Q \rangle$ vs training step at $\beta = 2$



(b) $\langle \delta Q_{\sin} \rangle$ vs training step at $\beta = 2$



(a) $\langle \delta Q \rangle$ vs training step at $\beta = 3$



(b) $\langle \delta Q_{\sin} \rangle$ vs training step at $\beta = 2$

15 | Separate Networks

Recall that a single leapfrog step (in the *forward* ($d = +1$) direction) of the L2HMC algorithm consists of the following 4 updates:

1. **vNet** : $\zeta_1 \coloneqq (x, \partial_x U(x), t) \longrightarrow (S_v, T_v, Q_v)$:

$$v' = v \odot \exp\left\{\left(\frac{\varepsilon}{2} S_v(\zeta_1)\right)\right\} - \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \exp\{(\varepsilon Q_v(\zeta_1))\} + T_v(\zeta_1)\right] \tag{67}$$

2. **xNet** : $\zeta_2 \coloneqq (x_{\bar{m}^t}, v', t) \longrightarrow (S_x, T_x, Q_x)$:

$$x' = x_{\bar{m}^t} + m^t \odot \left[x \odot \exp\{(\varepsilon S_x(\zeta_2))\} + \varepsilon\left(v' \odot \exp\{(\varepsilon Q_x(\zeta_2))\} + T_x(\zeta_2)\right)\right] \tag{68}$$

3. **xNet** : $\zeta_3 \coloneqq (x'_{m^t}, v', t) \longrightarrow (S_x, T_x, Q_x)$:

$$x'' = x'_{m^t} + \bar{m}^t \odot \left[x' \odot \exp\{(\varepsilon S_x(\zeta_3))\} + \varepsilon\left(v' \odot \exp\{(\varepsilon Q_x(\zeta_3))\} + T_x(\zeta_3)\right)\right] \tag{69}$$

4. **vNet** : $\zeta_4 \coloneqq (x'', \partial_x U(x''), t) \longrightarrow (S_v, T_v, Q_v)$:

$$v'' = v' \odot \exp\left\{\left(\frac{\varepsilon}{2} S_v(\zeta_4)\right)\right\} - \frac{\varepsilon}{2}\left[\partial_x U(x'') \odot \exp\{(\varepsilon Q_v(\zeta_4))\} + T_v(\zeta_4)\right]. \tag{70}$$

A complete trajectory (of N_{LF} leapfrog steps) is then obtained via N_{LF} successive applications of the above procedure, followed by a MH accept/reject step. Note that in this approach we use the **same two networks for all leapfrog steps**, namely: {**xNet**, **vNet**}, which update $\{x, v\}$ respectively. This can be generalized by introducing **different networks for each leapfrog step** (e.g. the first LF step uses {$\mathbf{xNet}^{(1)}$, $\mathbf{vNet}^{(1)}$}, the second uses {$\mathbf{xNet}^{(2)}$, $\mathbf{vNet}^{(2)}$}, ..., and the final step uses {$\mathbf{xNet}^{(N_{\text{LF}})}$, $\mathbf{vNet}^{(N_{\text{LF}})}$}). While this approach dramatically increases the number of trainable parameters, it is also vastly more expressive and allows the model to learn independent updates for each leapfrog step.

Explicitly, let $\mathbf{L}_\theta{}^{(i)} : \xi \to \xi'$ denote the operator which performs the i^{th} (augmented) leapfrog step in the direction d, and introduce $\Gamma_\pm^{(i)} : (v; \zeta_v) \to v'$, $\Lambda_\pm^{(i)} : (x; \zeta_x) \to x'$, where $\zeta_v = (x, \partial_x U(x), t)$, $\zeta_x = (x, v, t)$. We can then write the full position and momentum updates in terms of these functions

$$\Gamma_\pm^{(i)}(v; \zeta_v) = \begin{cases} v \odot \exp\left\{\left(\frac{\varepsilon}{2} S_v^{(i)}(\zeta_v)\right)\right\} - \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \exp\left\{\left(\varepsilon Q_v^{(i)}(\zeta_v)\right)\right\} + T_v^{(i)}(\zeta_v)\right] & (d = +1) \\ \left\{v + \frac{\varepsilon}{2}\left[\partial_x U(x) \odot \exp\left\{\left(\varepsilon Q_v^{(i)}(\zeta_1)\right)\right\} + T_v^{(i)}(\zeta_1)\right]\right\} \odot \exp\left\{\left(-\frac{\varepsilon}{2} S_v^{(i)}(\zeta_1)\right)\right\} & (d = -1) \end{cases} \tag{71}$$

$$\Lambda_\pm^{(i)}(x; \zeta_x) = \begin{cases} x \odot \exp\left\{\left(\varepsilon S_x^{(i)}(\zeta_x)\right)\right\} + \varepsilon\left[v' \odot \exp\left\{\left(\varepsilon Q_x^{(i)}(\zeta_x)\right)\right\} + T_x^{(i)}(\zeta_x)\right] & (d = +1) \\ \left\{x - \varepsilon\left[v' \odot \exp\left\{\left(\varepsilon Q_x^{(i)}(\zeta_x)\right)\right\} + T_x^{(i)}(\zeta_x)\right]\right\} \odot \exp\left\{\left(-\varepsilon S_x^{(i)}(\zeta_x)\right)\right\} & (d = +1) \end{cases} \tag{72}$$

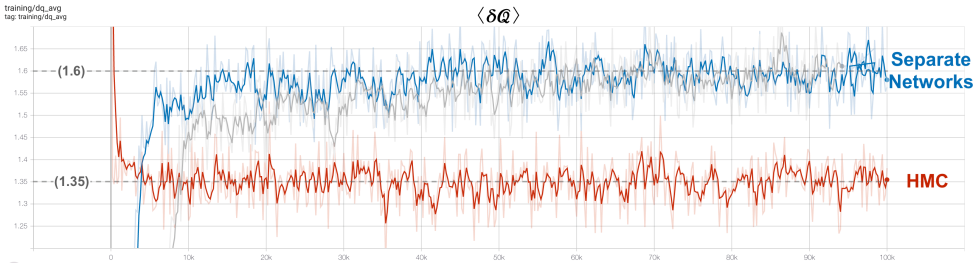
$$\tag{73}$$

We can write the i^{th} leapfrog step as:

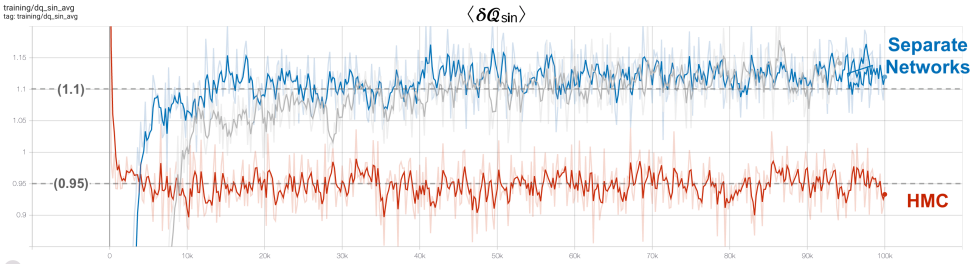
1. $v \longrightarrow v' = \Gamma_\pm^{(i)}(v; \zeta_v)$
2. $x \longrightarrow x' = x_{\bar{m}^t} + \Lambda_\pm^{(i)}(x_{m^t}; \zeta_x)$
3. $x' \longrightarrow x'' = x'_{\bar{m}^t} + \Lambda_\pm^{(i)}(x'_{m^t}; \zeta'_x)$
4. $v' \longrightarrow v'' = \Gamma_\pm^{(i)}(v'; \zeta''_v)$

and we can obtain a complete trajectory (of N_{LF} leapfrog steps) by succesively applying the above steps.

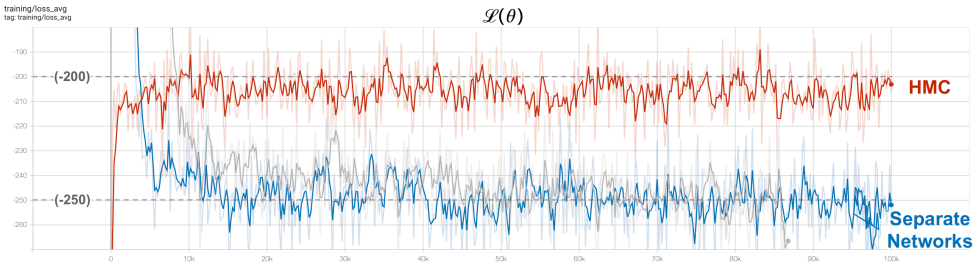
15.1 *(Preliminary) results:*



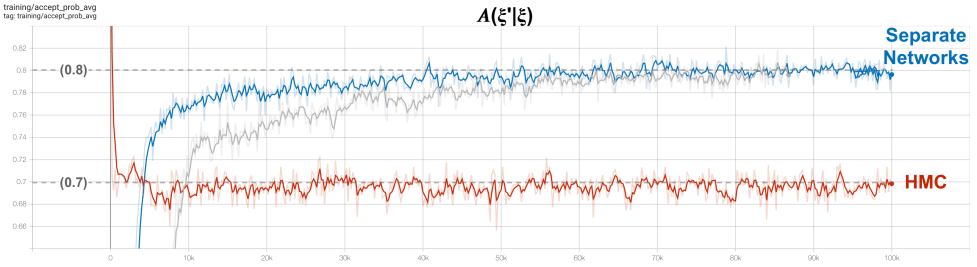
(a) sin tunneling rates, $\delta\mathbb{Q}_{\text{sin}}$ vs training step



(b) Tunneling rates, $\delta\mathbb{Q}$, $\delta\mathbb{Q}_{\text{sin}}$ vs training step



(c) Loss, $\mathscr{L}(\theta)$ vs training step



(d) Acceptance probability, $A(\xi'|\xi)$ vs training step

16 | Tunneling Rates

For inference, we generate $M \times N$ configurations (batch of M chains in parallel, for N MH-Accept/Rejects), and drop the first 25% percent of the data before computing observables to account for burn-in[2]. We compute the topological charge, $\mathbb{Q} = \frac{1}{2\pi}\sum_x \text{Arg}\left(\phi_{\mu\nu}(x)\right) \in \mathbb{Z}$, with $\text{Arg}\left(\phi_{\mu\nu}(x)\right) \in [-\pi, \pi]$ for each configuration.

For the m^{th} chain in our batch, we define

$$\delta\mathbb{Q}_m(n) := \mathbb{Q}(n+1) - \mathbb{Q}(n) \tag{74}$$

$$\langle\delta\mathbb{Q}_m\rangle := \frac{1}{N}\sum_{n=0}\delta\mathbb{Q}_m(n). \tag{75}$$

We can then compute the average tunneling rate over all chains (i.e. the number of tunneling events per step) as

$$\langle\delta\mathbb{Q}\rangle := \frac{1}{M}\sum_{m=1}^{M}\langle\delta\mathbb{Q}_m\rangle = \frac{1}{M}\sum_{m=1}^{M}\left[\frac{1}{N}\sum_{n=0}^{N}\delta\mathbb{Q}_m(n)\right] \tag{76}$$

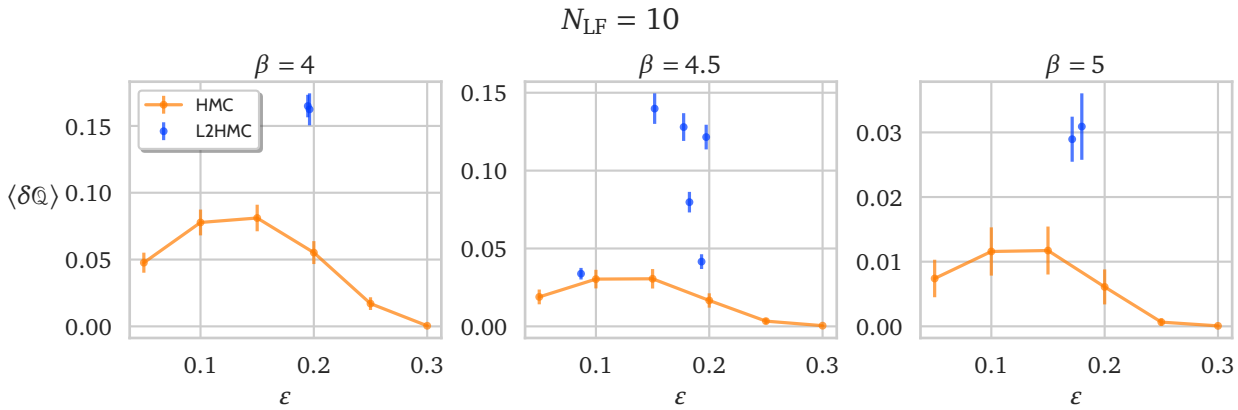


Figure 22: Comparison of the tunneling rates, $\langle\delta\mathbb{Q}\rangle$ for HMC and L2HMC

17 | Transformations

As we increase β, it becomes increasingly unlikely for the sampler to tunnel between different topological sectors, resulting in overall poor performance. In order to combat this effect, we have been focusing on better understanding the mechanism by which our model is able to "learn" how to sample from different topological sectors. Specifically, we looked at how the effective energy, $\mathcal{H} - \log|\mathcal{J}|$ varies during a trajectory, as shown in Fig. **??**. We immediately notice that the energy tends to increase during the first-half of the trajectory, before decreasing back towards its initial value. It appears that by increasing the energy towards the middle of the trajectory, the sampler is able to overcome the potential barriers seperating regions of distinct topological charge. In [2] the authors introduced this idea, referred to as "tempering during the trajectory" and methods for manually producing a similar effect are discussed. In Fig. 25, we look at how the continuously-valued topological charge,

$$\sin\mathbb{Q} \equiv \frac{1}{2\pi}\sum_P \sin\phi_P \tag{77}$$

varies during a trajectory.

[2]For computing statistics of observables, we use bootstrap resampling.

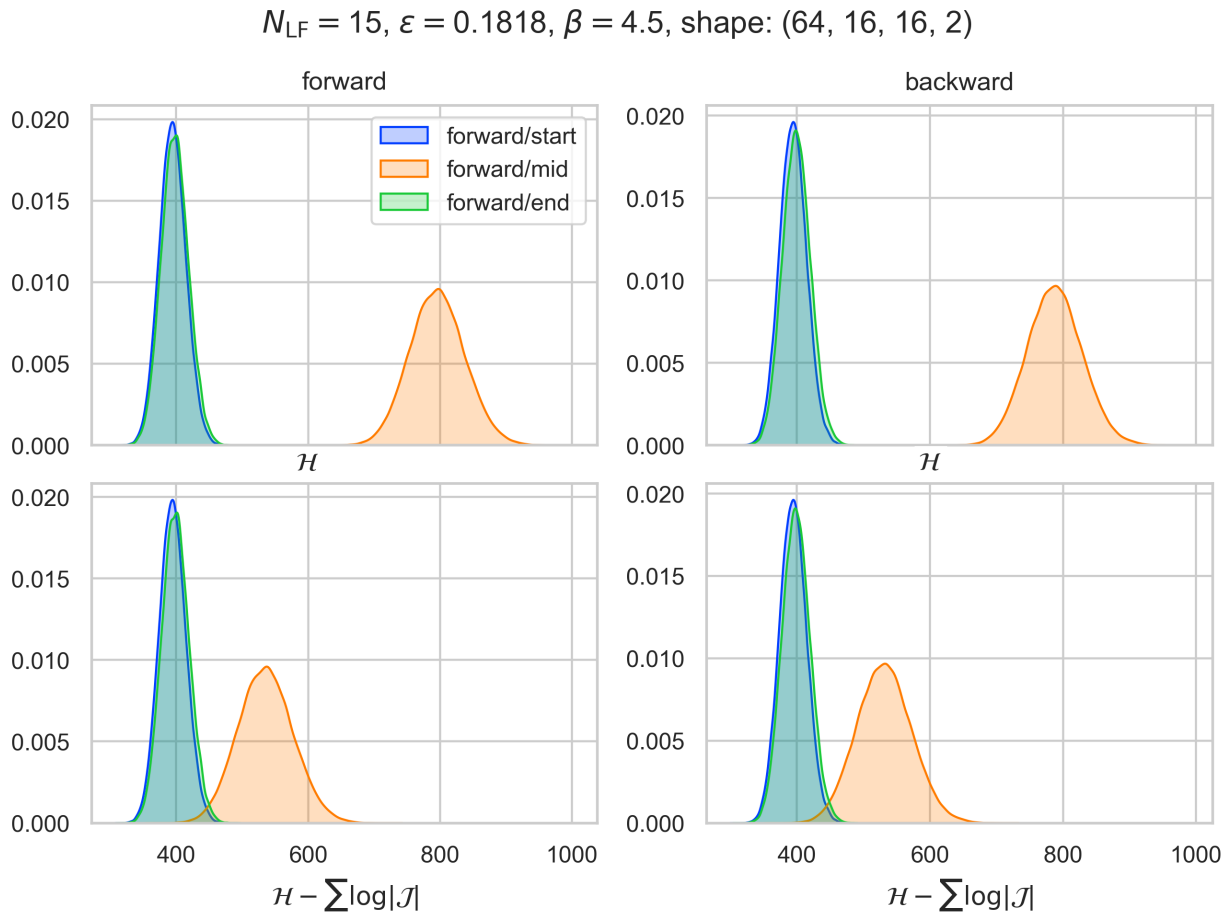
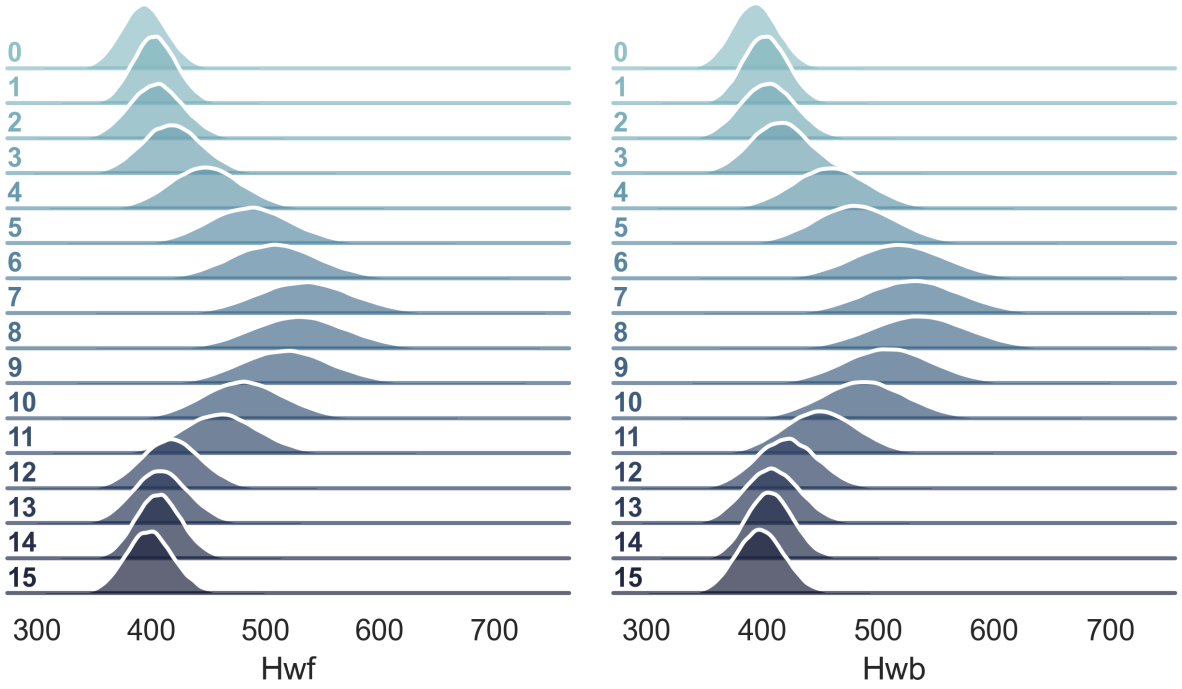


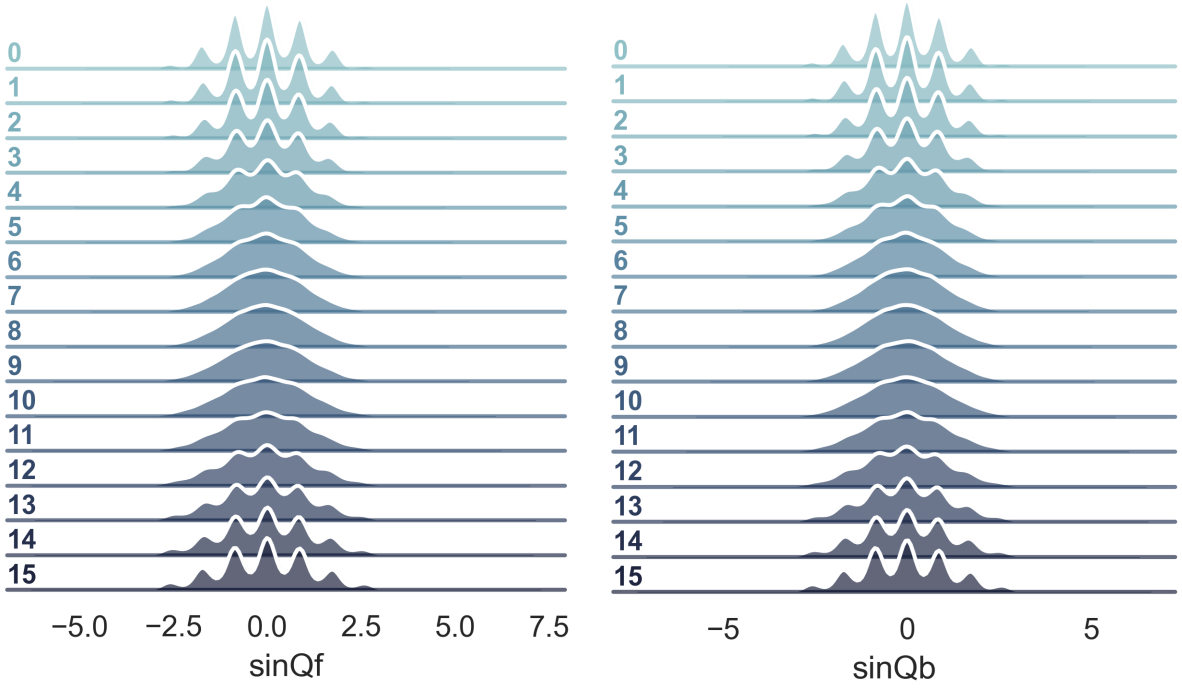
Figure 23: Illustration of the energy $\mathcal{H}$ (top row) and re-scaled energy $\mathcal{H} - \sum \log|\mathcal{J}|$ (bottom row) at the start, middle, and end of a trajectory during inference.



(a) Forward direction, $d = +1$

(b) Backward direction, $d = -1$

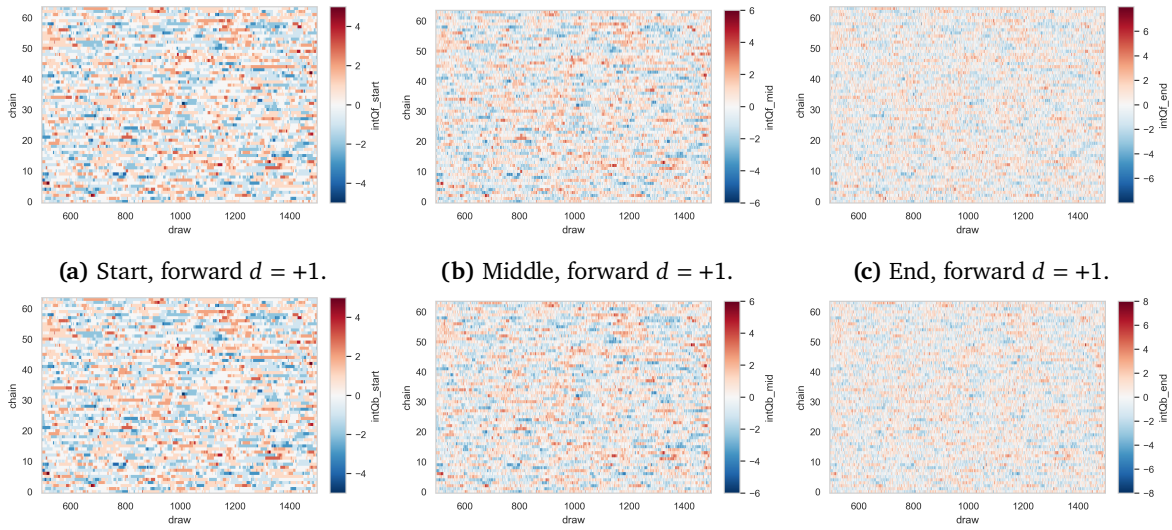
Figure 24: Illustration of the effective energy $\mathcal{H} - \sum \log |\mathcal{J}|$ at each leapfrog step during the trajectory.



(a) Forward direction, $d = +1$

(b) Backward direction, $d = -1$

Figure 25: Illustration of the continuously-valued topoological charge, $\sin \mathbb{Q}$ at each leapfrog step during the trajectory.



(a) Start, forward $d = +1$. **(b)** Middle, forward $d = +1$. **(c)** End, forward $d = +1$.

(d) Start, backward $d = -1$. **(e)** Middle, backward $d = -1$. **(f)** End, backward $d = -1$.

Figure 26: Illustration of the topological charge, $\mathbb{Q}$ for each configuration at the beginning, middle and end of a trajectory proceeding in the forward $d = +1$ (top row) and backward $d = -1$ (bottom row) direction.

References

[1] D. Levy, M. D. Hoffman, and J. Sohl-Dickstein, arXiv:1711.09268 [cs, stat], arXiv: 1711.09268 (2017).

[2] R. M. Neal, arXiv:1206.1901 [physics, stat], arXiv: 1206.1901 (2012).

[3] M. Betancourt, arXiv:1701.02434 [stat], arXiv: 1701.02434 (2017).

[4] *30 joeylitalien/l2hmc: iclr 2018 reproducibility challenge: generalizing hamiltonian monte carlo with neural networks*, https://joeylitalien.github.io/assets/reports/l2hmc.pdf, (Accessed on 06/21/2019).

[5] L. Dinh, J. Sohl-Dickstein, and S. Bengio, arXiv:1605.08803 [cs, stat], arXiv: 1605.08803 (2016).

[6] C. Pasarica and A. Gelman, Statistica Sinica **20**, 343 (2010).

[7] S. Ioffe and C. Szegedy, arXiv:1502.03167 [cs], arXiv: 1502.03167 (2015).

Appendices

In Appendices A-B, we describe some of the additional work that has been done in an effort to improve the algorithms performance when applied to models in lattice gauge theory and lattice QCD. Unfortunately, neither of these approaches seemed to significantly improve the quality of the sampler nor eliminate the error present in the average plaquette. Since each of these new ideas only further complicate the situation, they have been (temporarily) put on hold until the issues with the average plaquette can be resolved.

A | Modified Network Architecture

In order to better account for the rectangular geometry of the lattice, the previously described architecture was modified to include a stack of convolutional layers immediately following the input layer, as shown in Fig 27. The output from this convolutional structure is then fed to the generic network shown in Fig. **??**. This is a natural direction to pursue given the inherent translational invariance of both convolutional neural networks and rectangular lattices (with periodic boundary conditions). Additionally, the network architec-

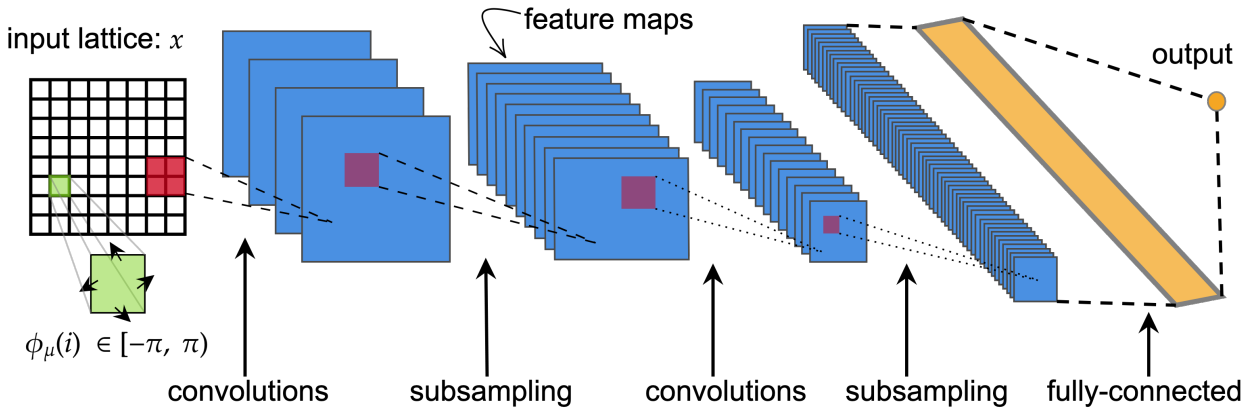


Figure 27: Convolutional structure used for learning localized features of rectangular lattice.

ture was modified to include a batch normalization layer after the second MaxPool layer. Introducing batch normalization is a commonly used technique in practice and is known to:

1. Help prevent against diverging gradients[3], (an issue that was occasionally encountered during the training procedure).
2. Generally improve model performance by achieving similar performance in fewer training steps when compared to models trained without it [7].

Additionally, it has been shown to improve model performance and generally requires fewer training steps to achieve similar performance as models trained without it [7].

A.1 Network Diagrams

For completeness, we include below the two possible network diagrams for a single **xNet**.

B | Topological Loss Term

While the alternative metric introduced in Eq. 50 helps to better measure distances in this configuration space, it does nothing to encourage the exploration of different topological sectors since there may be config-

[3] a numerical issue in which infinite values are generated when calculating the gradients in backpropagation

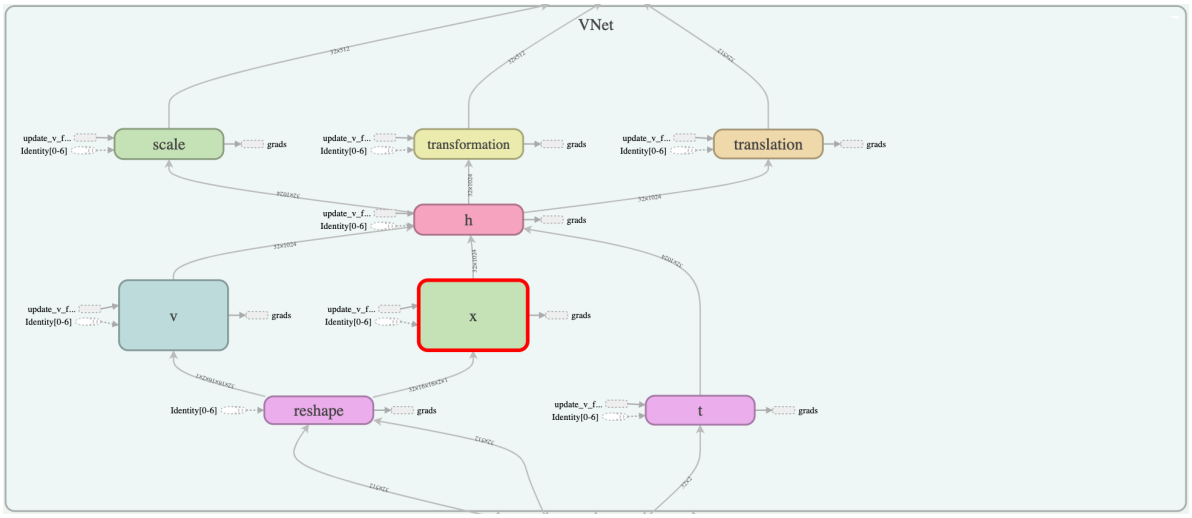


Figure 28: Illustration taken from TensorBoard showing an overview of the network architecture for VNet. Note that the architecture is identical for XNet.

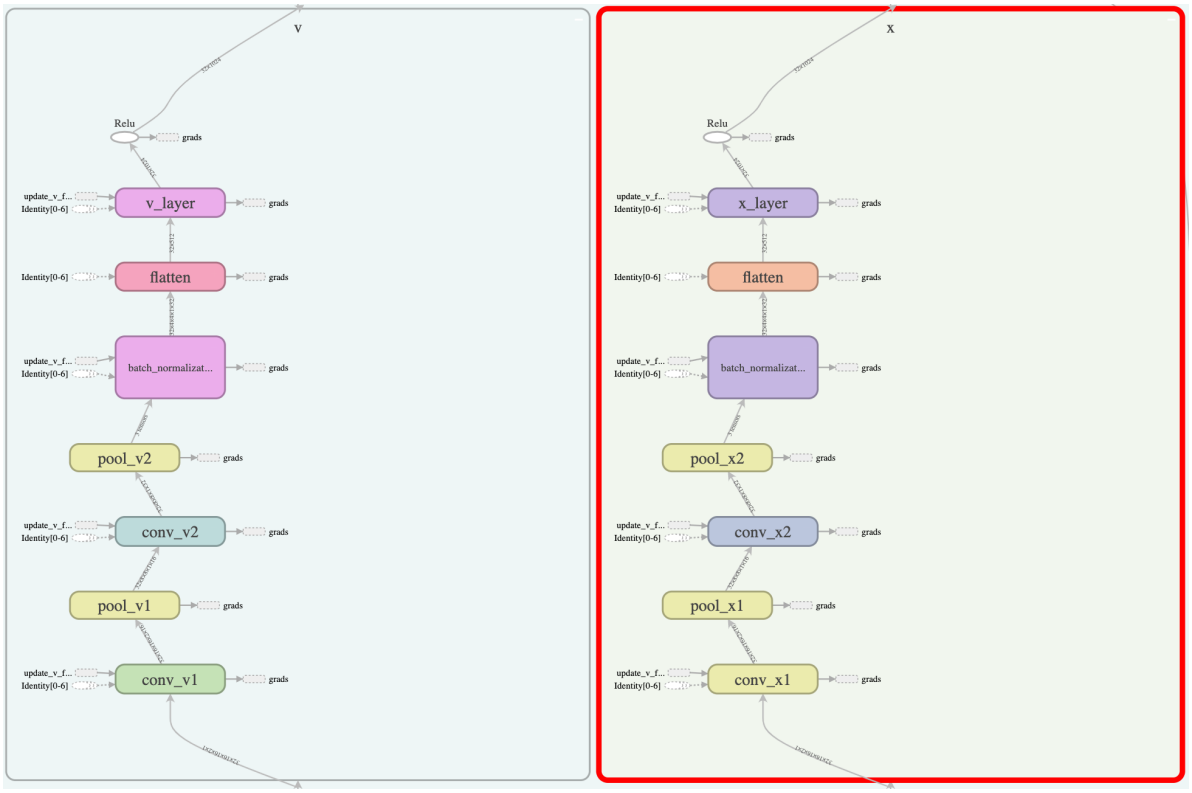


Figure 29: Detailed view of additional convolutional structure included to better account for rectangular geometry of lattice inputs.

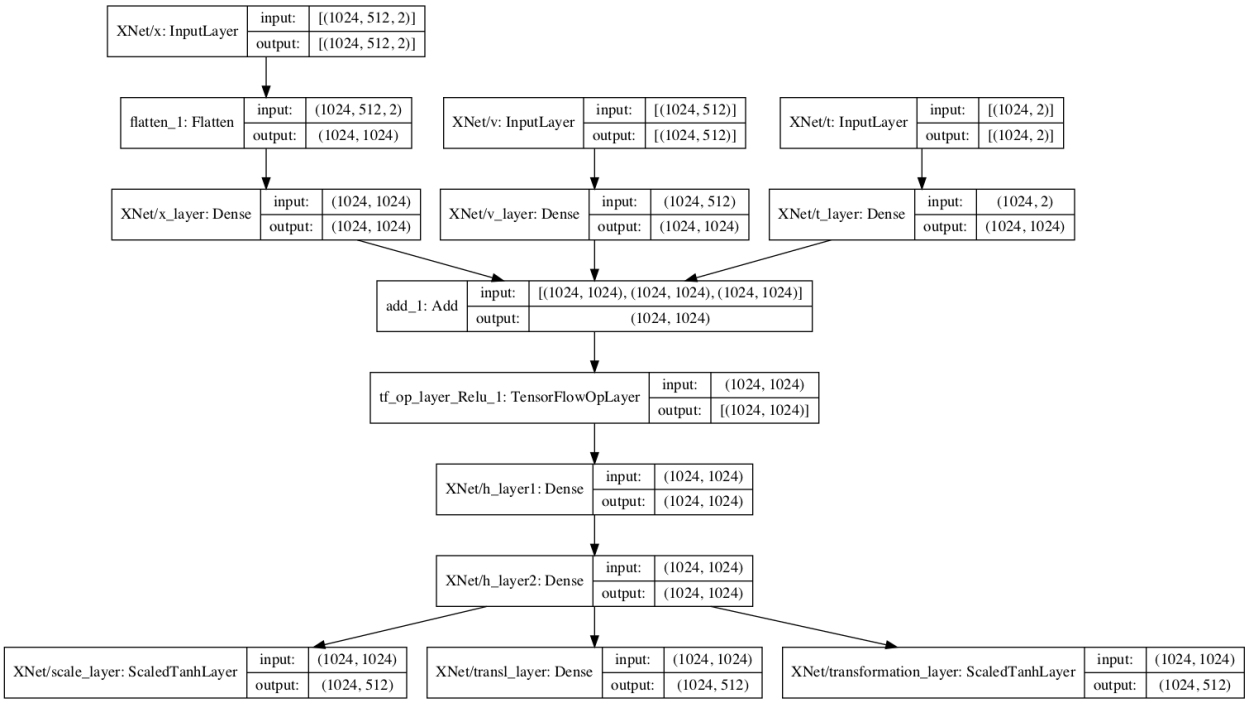


Figure 30: Network diagram for generic **xNet** (without convolutional structure).

urations for which $\delta(\xi, \xi') \approx 1$ but $Q(\xi) = Q(\xi')$. In order to potentially address this issue, we consider the following modification to the original loss function.

First define $\xi' \equiv \mathbf{FL}_\theta \xi$ as the resultant configuration proposed by the augmented leapfrog integrator, and

$$\delta_Q(\xi, \xi') = |Q(\xi) - Q(\xi')| \tag{78}$$

$$\ell_Q(\xi, \xi', A(\mathbf{FL}_\theta \xi|\xi)) = \delta_Q(\xi, \xi') \times A(\xi'|\xi). \tag{79}$$

So we have that δ_Q measures the difference in topological charge between the initial and proposed configurations, and ℓ_Q gives the expected topological charge difference. Proceeding as before, we include an additional auxiliary term which is identical in structure to the one above, except the input is now a configuration of link variables ϕ_μ drawn from the initialization distribution q, which for our purposes was chosen to be the standard random normal distribution on $[0, 2\pi$.

We can then write the topological loss term as

$$\mathscr{L}_Q(\theta) \equiv \mathbb{E}_{p(\xi)}\left[\ell_Q(\xi, \mathbf{FL}_\theta \xi, A(\mathbf{FL}_\theta \xi|\xi))\right] + \alpha_{\text{aux}}\, \mathbb{E}_{q(\xi)}\left[\ell_Q(\xi, \mathbf{FL}_\theta \xi, A(\mathbf{FL}_\theta \xi|\xi))\right] \tag{80}$$

If we denote the standard loss (with the modified metric function) defined in Eq. 37 as $\mathscr{L}_{\text{std}}(\theta)$, we can write the new total loss as a combination of these two terms,

$$\mathscr{L}(\theta) = \alpha_{\text{std}}\, \mathscr{L}_{\text{std}}(\theta) + \alpha_Q\, \mathscr{L}_Q(\theta) \tag{81}$$

where $\alpha_{\text{std}}, \alpha_Q$ are multiplicative factors that weigh the relative contributions to the total loss from the standard and topological loss terms respectively, and α_{aux} in Eq. 80 weighs the contribution of configurations drawn from the initialization distribution.

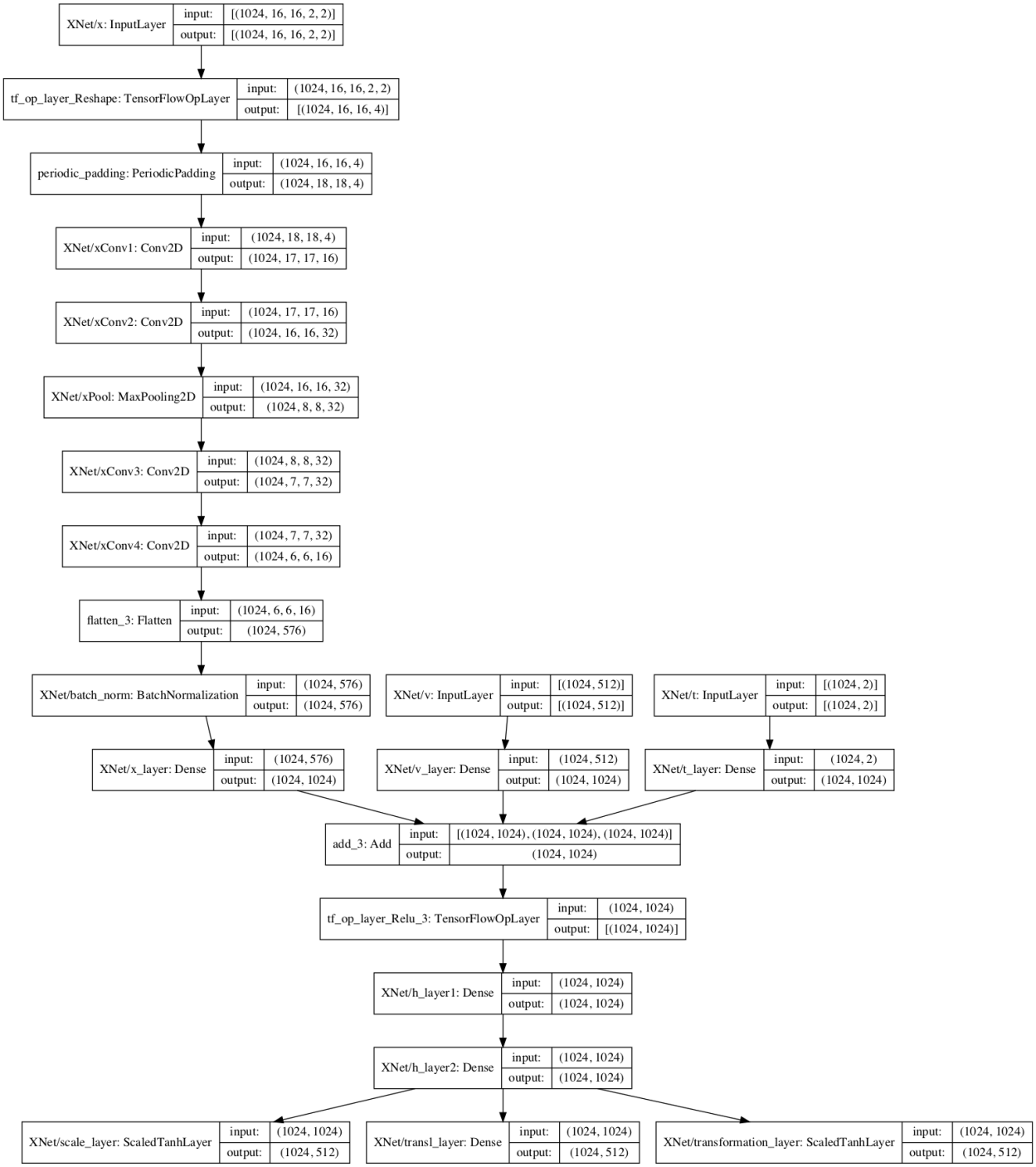


Figure 31: Network diagram for convolutional **xNet**.