Overview of the L2HMC Algorithm*

Sam Foreman†
James Osborn‡
Xiao-Yong Jin§

June 30, 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.

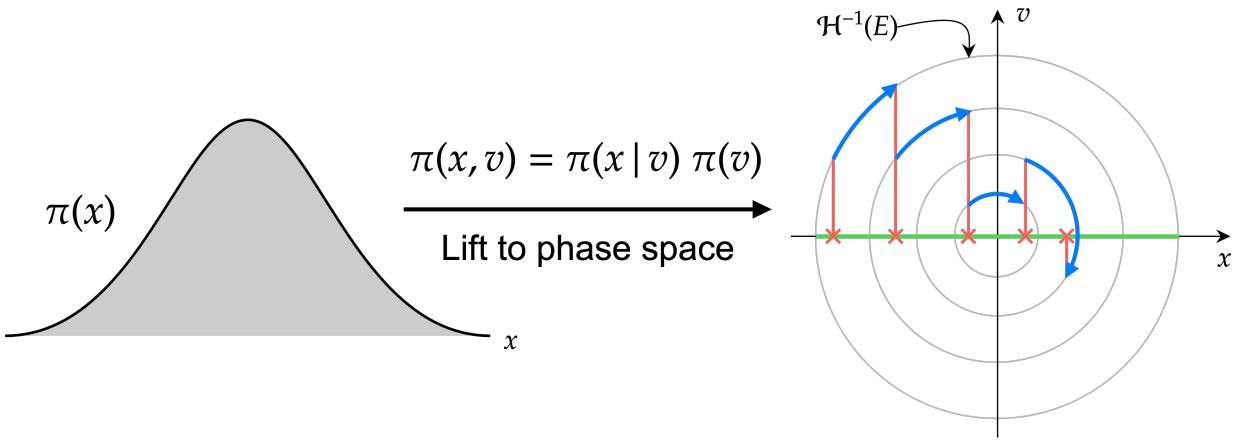


Figure 1: *Visualizing HMC for a 1D Gaussian* (example from [3], figure adapted with permission from [4]). Each Hamiltonian Markov transition lifts the initial state onto a random level set of the Hamiltonian, $H^{(-1)}(E)$, which can then be explored with a Hamiltonian trajectory before projecting back 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: $\exp\left(\frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta_1)\right)$. 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'':

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

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

with Jacobians: $\exp\left(\varepsilon m^t \cdot S_x(\zeta_2)\right)$, 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 $\exp\left(\frac{\varepsilon}{2}\mathbb{1} \cdot S_v(\zeta)\right)$ 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].

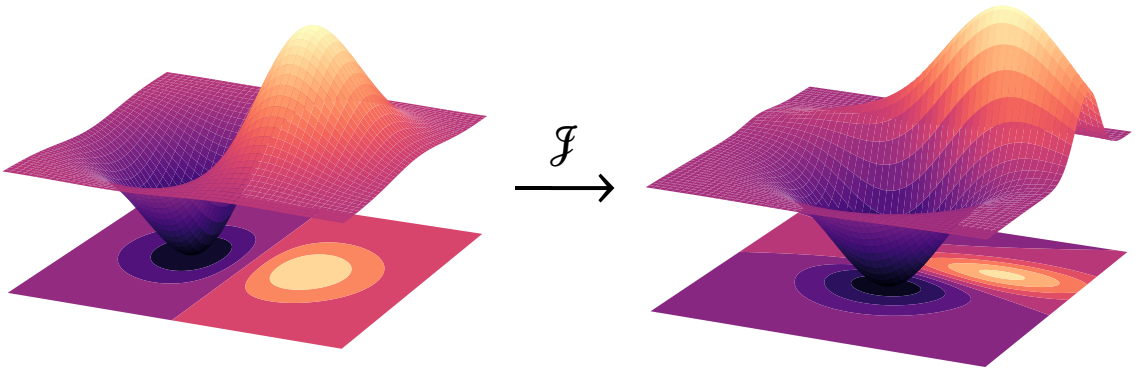


Figure 2: Example of how the determinant of the Jacobian can deform the energy landscape.

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 ($d = 1$):

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

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

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

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

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 ($d = -1$):

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

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

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

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

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:

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

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

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). Since both networks are identical, we describe the architecture of V_{net} below, and include a flowchart for X_{net} illustrative purposes in Fig 4.

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 $\tau(t) = \left(\cos\left(\frac{2\pi t}{N_{\mathrm{LF}}}\right),\ \sin\left(\frac{2\pi t}{N_{\mathrm{LF}}}\right)\right)$. Each of the inputs is then passed through a fully-connected ('dense' layer), consisting of n_h hidden units

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

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

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

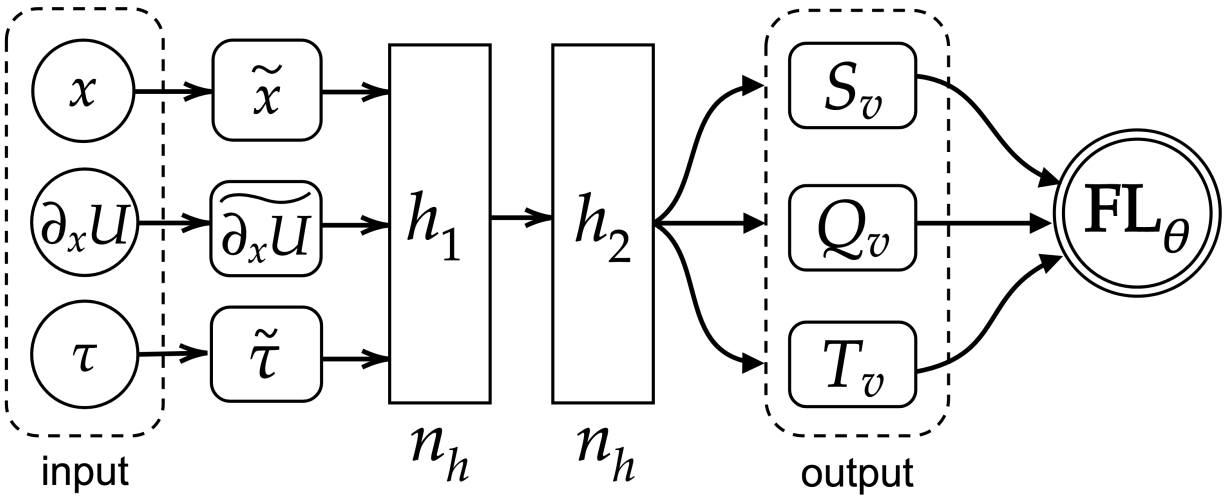


Figure 3: Illustration showing the generic (fully-connected) network architecture for training S_v, Q_v, and T_v. Figure adapted with permission from [4].

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:

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

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

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

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.

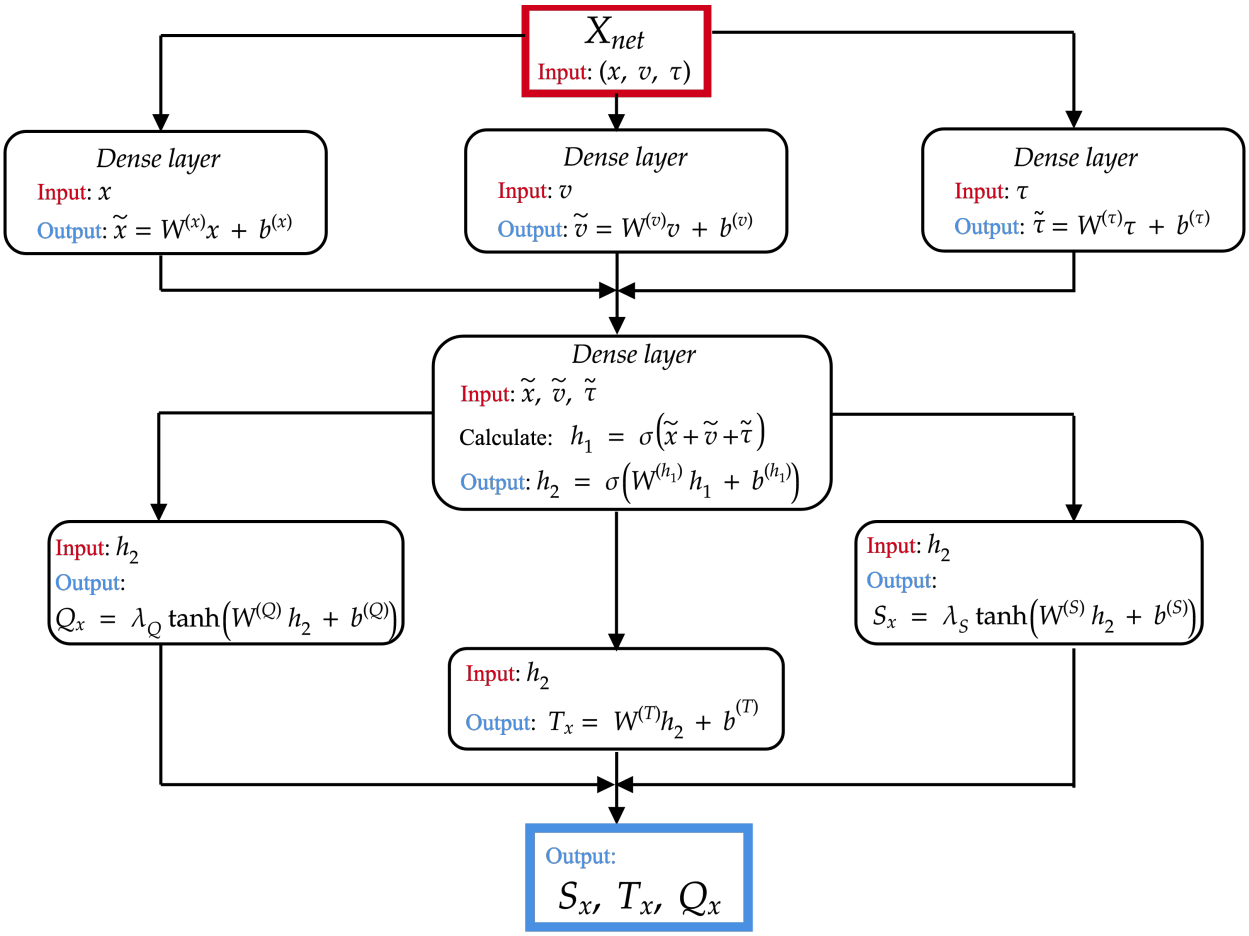


Figure 4: Flowchart illustrating the generic fully-connected network architecture including the intermediate variables computed at each hidden layer of the network.

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. 5. 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 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:

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

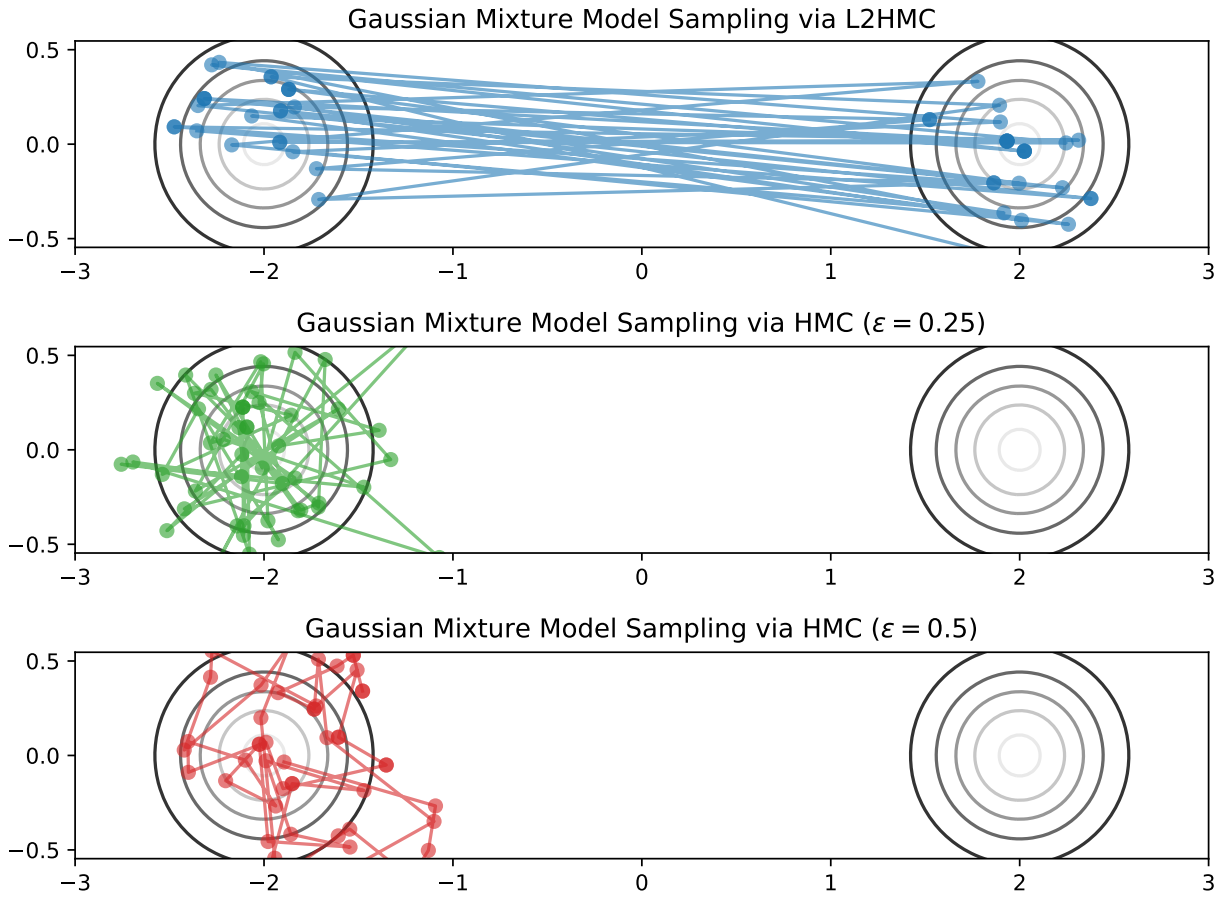


Figure 5: Comparison of trajectories generated using L2HMC (top), and traditional HMC with $\varepsilon = 0.25$ (middle) and $\varepsilon = 0.5$ (bottom). Note that L2HMC is able to successfully mix between modes, whereas HMC is not.

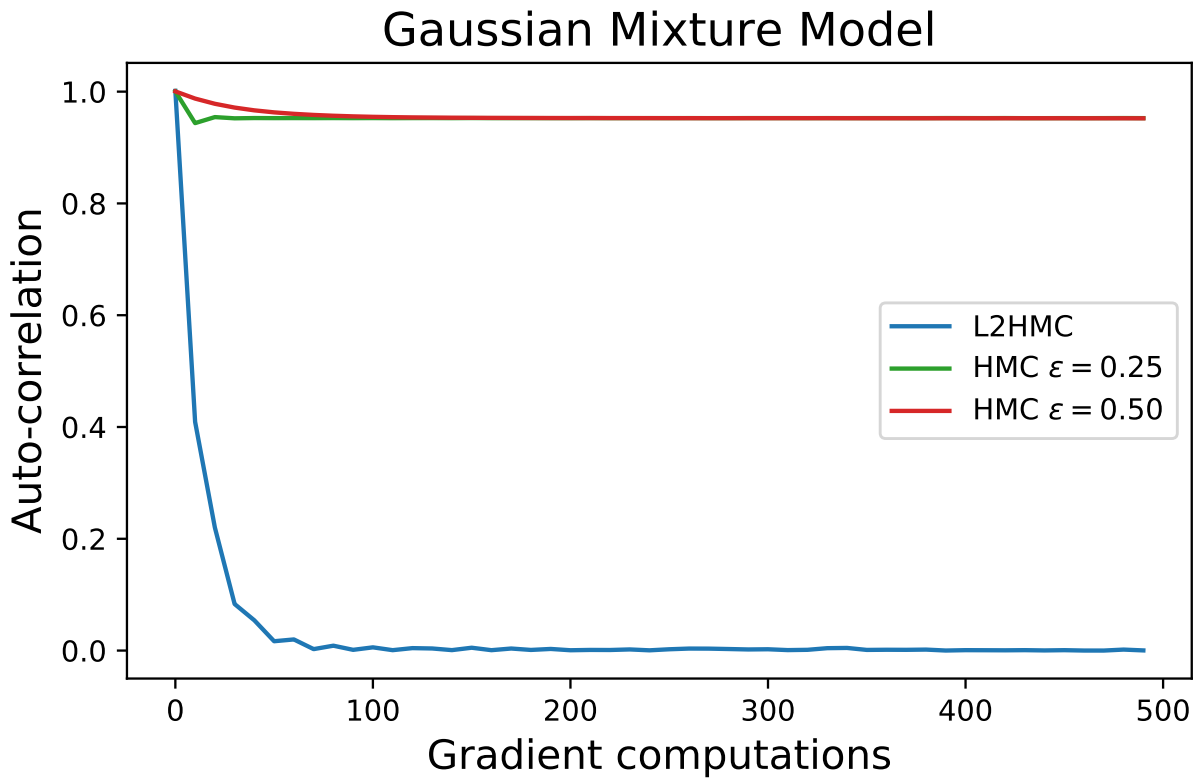


Figure 6: Autocorrelation vs. gradient evaluations (i.e. MD steps). Note that L2HMC (blue) has a significantly reduced autocorrelation after the same number of gradient evaluations when compared to either of the two HMC trajectories

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. 7.

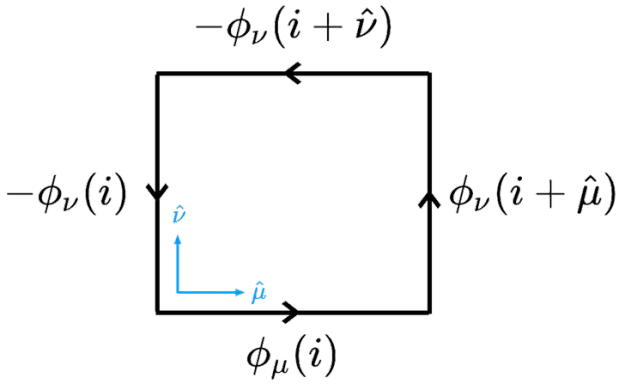


Figure 7: Illustration of an elementary plaquette on the lattice.

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*.

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.

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

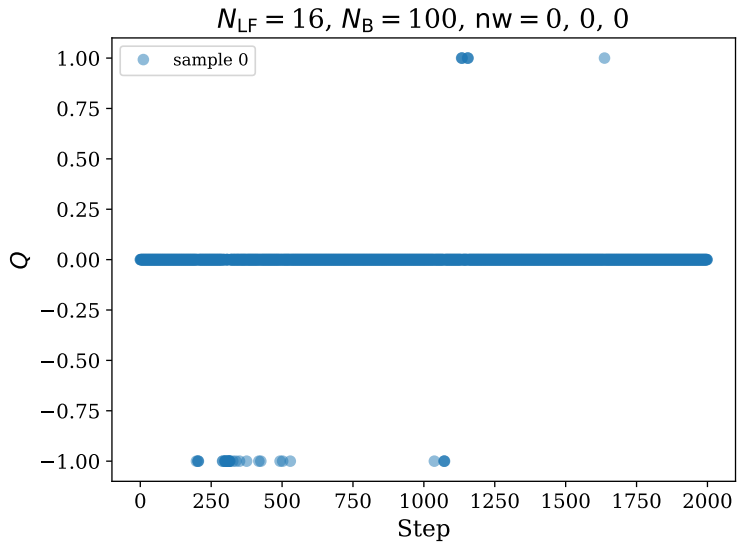


Figure 8: Example of topological freezing in the $2D$ $U(1)$ lattice gauge theory. The above result was generated using generic HMC sampling for a 8×8 lattice. Note that for the majority of the simulation $\mathbb{Q} = 0$, making it virtually impossible to get a reasonable estimate of χ.

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 $\xi \approx \xi'$.

10.3 Issues with the Average Plaquette

When running inference using the trained sampler, an issue was encountered in which the average plaquette, $\langle \phi_P \rangle$ seems to converge to a value which is noticeably different from the expected value in the infinite volume limit, and can be seen in Fig. 9. In order to quantify this unexpected behavior, we can calculate the difference between the observed value of the average plaquette, $\langle \phi_P \rangle$ and the expected value $\phi^{(*)}_P$ (calculated from the infinite volume limit):

$$\delta_{\phi_P}(\alpha_Q, N_{\text{LF}}) \equiv \langle \phi_P \rangle - \phi^{(*)}_P \neq 0. \tag{51}$$

[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.

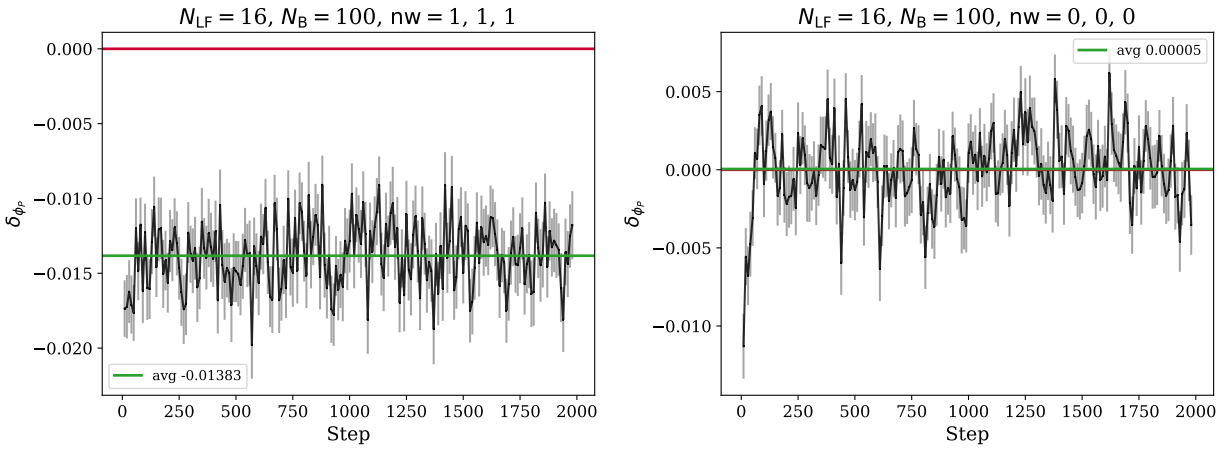


Figure 9: Difference between the observed and expected value of the average plaquette, δ_{ϕ_P}, **(left)**: using the trained L2HMC sampler, and **(right)**: using generic HMC.

11 | Debugging

11.1 Net Weights

In an attempt to better understand the source of this unexpected behavior, multiplicative weights (referred to as 'nw' for 'network weights' in Fig. 9) were introduced to scale the individual contribution from each of the 'learned' functions S, T, and Q, explicitly:

$$S \longrightarrow \alpha_S \, S \tag{52}$$

$$T \longrightarrow \alpha_T \, T \tag{53}$$

$$Q \longrightarrow \alpha_Q \, Q. \tag{54}$$

By varying each of these weights individually allows us to selectively 'tune' how much each of these functions contribute when running inference on the trained model. Note that in the limit $\alpha_S, \alpha_T, \alpha_Q \to 0$, we recover generic HMC, and as can be seen in Fig. 9, the error in the average plaquette $\delta_{\phi_P} \simeq 0$, as expected. As an additional sanity check, we looked at how the error in the average plaquette behaves for different values of the weights α_i $(i = S, T, Q)$. Explicitly, beginning with $\vec{\alpha} \equiv [\alpha_S, \alpha_Q, \alpha_T] = [0, 0, 0]$, we increase each of the weights one by one and compute the average value of the plaquette difference. For example, the blue line (Transformation (Q) function) in Fig. 10 was obtained by keeping both α_S and α_T fixed and 0 and varying $\alpha_Q \in [0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 5.0]$, and similarly for α_S and α_Q. These results seem to indicate that each of the individual functions contribute separately to the error, with the scaling (S) and translation (T) functions having the largest effect.

11.2 Updates (11/11/2019)

- ☑ Ensure reproducibility across training/inference runs when using same input parameters.
  - Essential for debugging the bias in the average plaquette.
  - Somewhat tricky problem due to the fact that tensorflow has two distinct methods of specifying a seed: graph-level and operation-level. Additionally, when using horovod for distributed training across multiple ranks, we must ensure that each rank gets a different seed otherwise they will all be training identical copies of the model.
- ☑ Implement reversibility checker that ensures that the L2HMC dynamics (i.e. the augmented HMC sampler) is reversible.

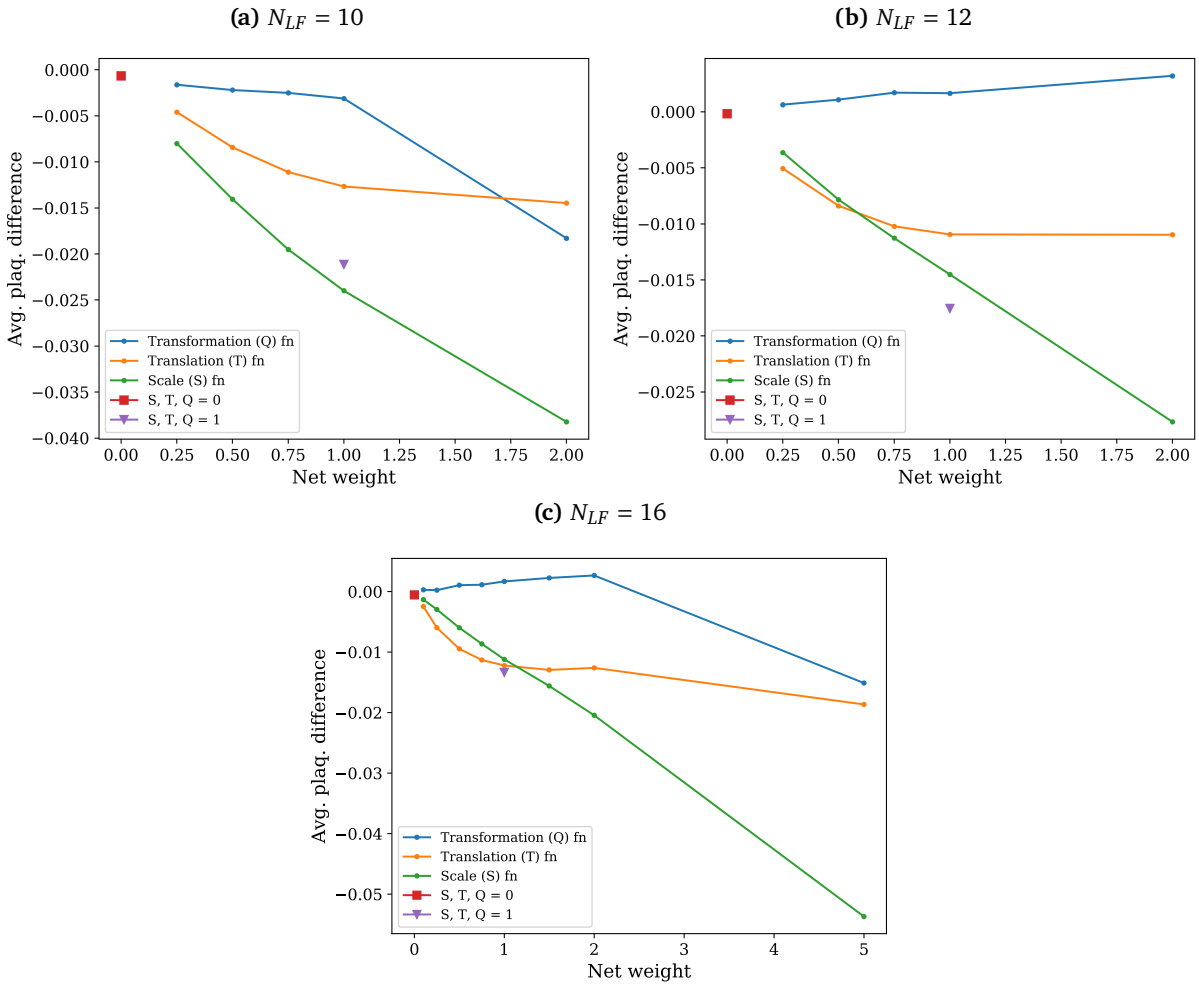


Figure 10: Plaquette difference δ_{ϕ_P} for different values of the net weights $\vec{\alpha} \equiv [\alpha_S, \alpha_T, \alpha_Q]$. Note that $\delta_{\phi_P} \longrightarrow 0$ as $\vec{\alpha} \longrightarrow [0, 0, 0]$, as expected.

- Starting with $\xi = (x, v, d)$, run the dynamics in the forward direction to get ξ'. If $\mathbf{L}_\theta \xi = \xi'$, and $\mathbf{F}\xi = \mathbf{F}(x, v, d) = (x, v, -d)$, then a complete (invertible) update step can be written as $\mathbf{F}\mathbf{L}_\theta \xi = \xi'$.
- If our sampler is reversible, running the dynamics backwards on ξ' should return us to the original state ξ.

$$\mathbf{F}\mathbf{L}_\theta \mathbf{F}\mathbf{L}_\theta \xi = \xi \tag{55}$$

☑ Try increasing floating point precision (`tf.float32 -> tf.float64`) (**Error still present.**)

☑ Try anti-symmetric Gaussian Mixture Model and see if the trained model is an accurate representation of the target distribution (e.g. by looking at the locations of the means).

☑ At James' suggestion, we decided to look at the kinetic/potential energies and the Hamiltonian at the beginning and end of each trajectory (**Solved! Issue was being caused by unexpected resampling of the momentum**).

12 | GMM Results

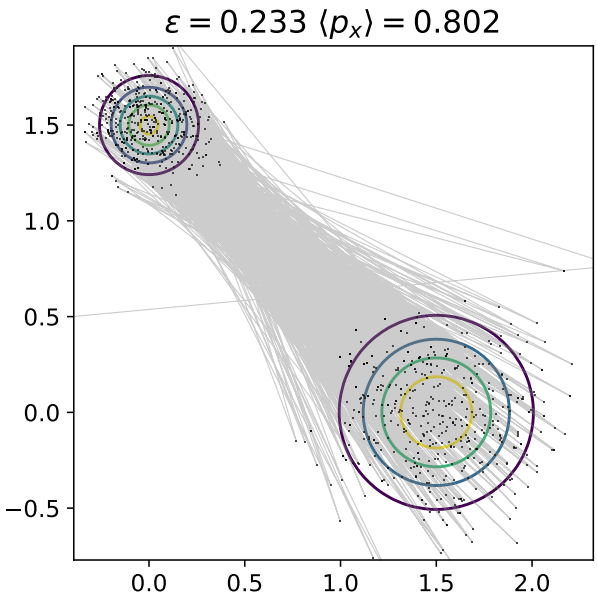


Figure 11: Inference run shown for a single chain using the L2HMC sampler trained on this gaussian mixture model.

13 | Updates 12/10/2019

- It was observed that if the model was trained (using simulated annealing) to a final β_{final} that was greater than the value of $\beta_{\text{inference}}$ at which we intend to run inference, the bias in the average plaquette decreased. For example, if $\beta_{\text{final}} = 5$ and $\beta_{\text{inference}} = 4$, the bias seemed to be smaller when running at $\beta_{\text{inference}} = 4$ than it was when running at $\beta_{\text{inference}} = 5$.
  - Try using larger value of β_{final} in annealing schedule to test if this is consistent.
- Reduce $N_{\text{LF}} \to 1$

(a) L2HMC samples **(b)** true target distribution

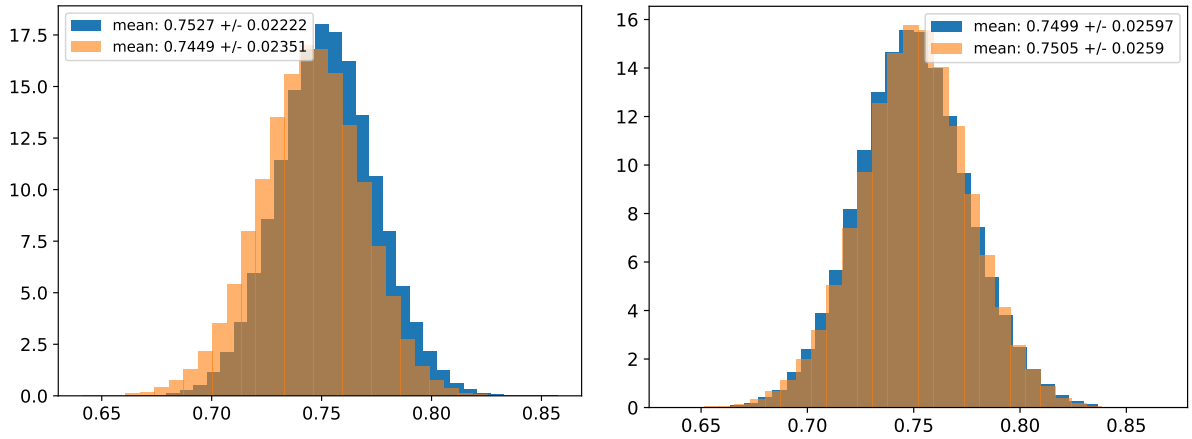


Figure 12: Histograms of $\langle x \rangle$ and $\langle y \rangle$ in this two-dimensional target space.

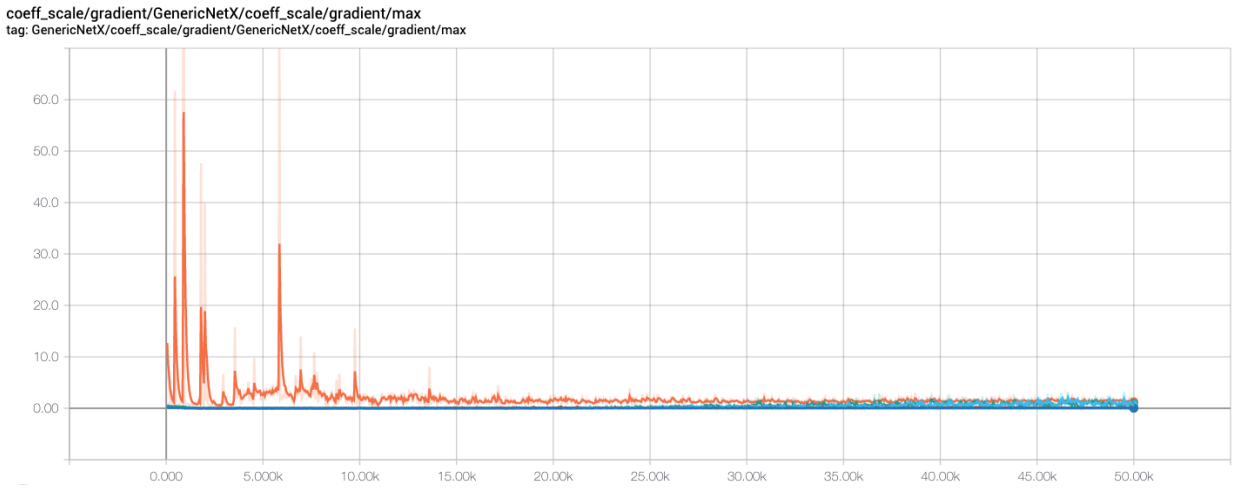


Figure 13: Example of a spiking gradient.

- Split `net weights`$(= [\alpha_S, \alpha_T, \alpha_Q])$ into separate components for x and v so that

$$\texttt{net weights} \equiv [\alpha_{S_x}, \alpha_{T_x}, \alpha_{Q_x}, \alpha_{S_v}, \alpha_{T_v}, \alpha_{Q_v}] \tag{56}$$

- Explicitly loop over different values of `net weights` to determine which (if any) of the functions S_x, T_x, Q_x, S_v, T_v, or Q_v has the largest contribution to the bias in the average plaquette.
- Slowly turn things off until results agree with generic HMC and then turn them back on individually until difference reappears.
- Looking at summaries in TensorBoard, it was observed that certain gradients experienced large spikes during training (See Fig. 13).
  - **Use gradient clipping!** (by global norm)

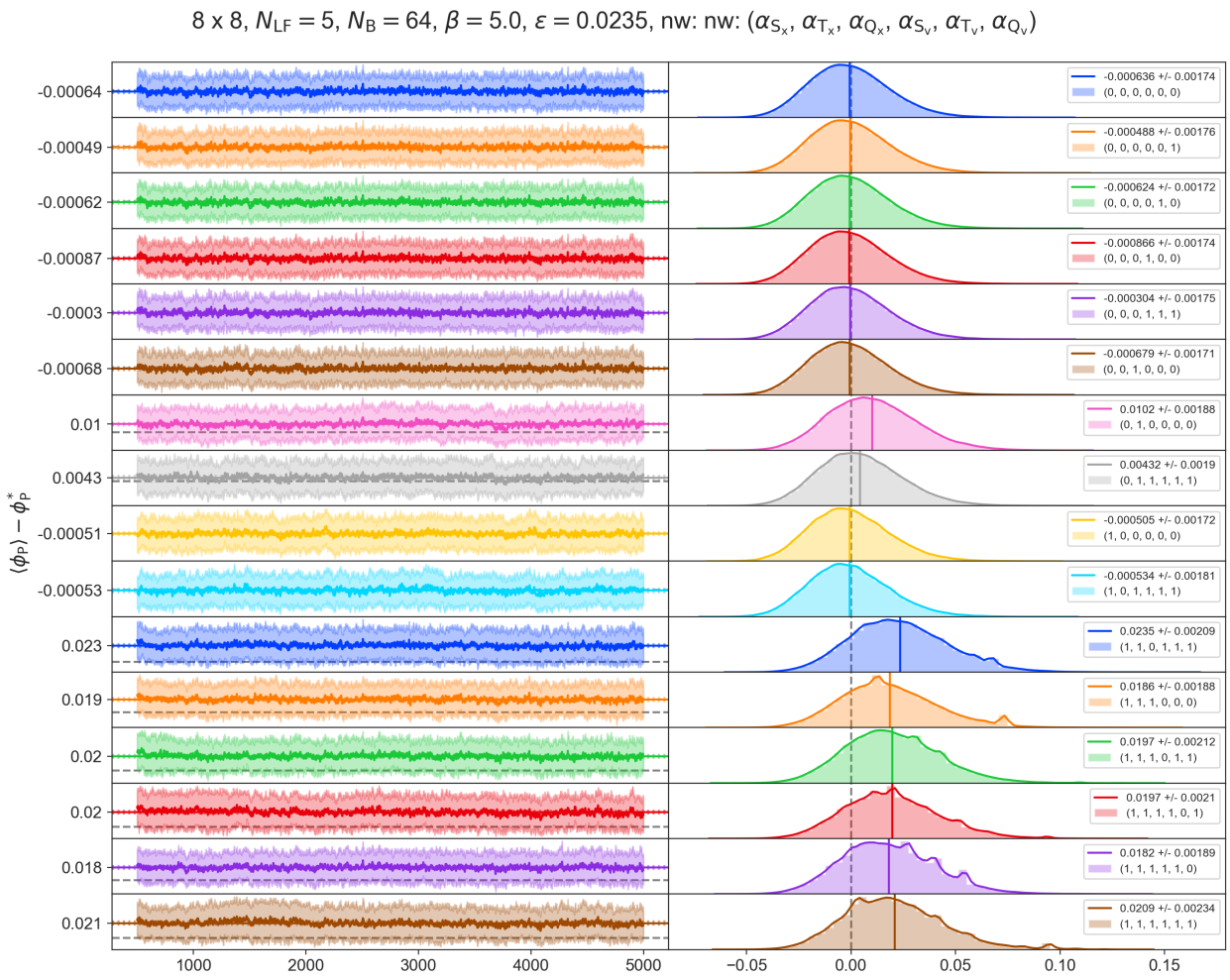


Figure 14: Trace plots and histograms from an inference run with $N_{\text{LF}} = 5$ and various values of `net weights`.

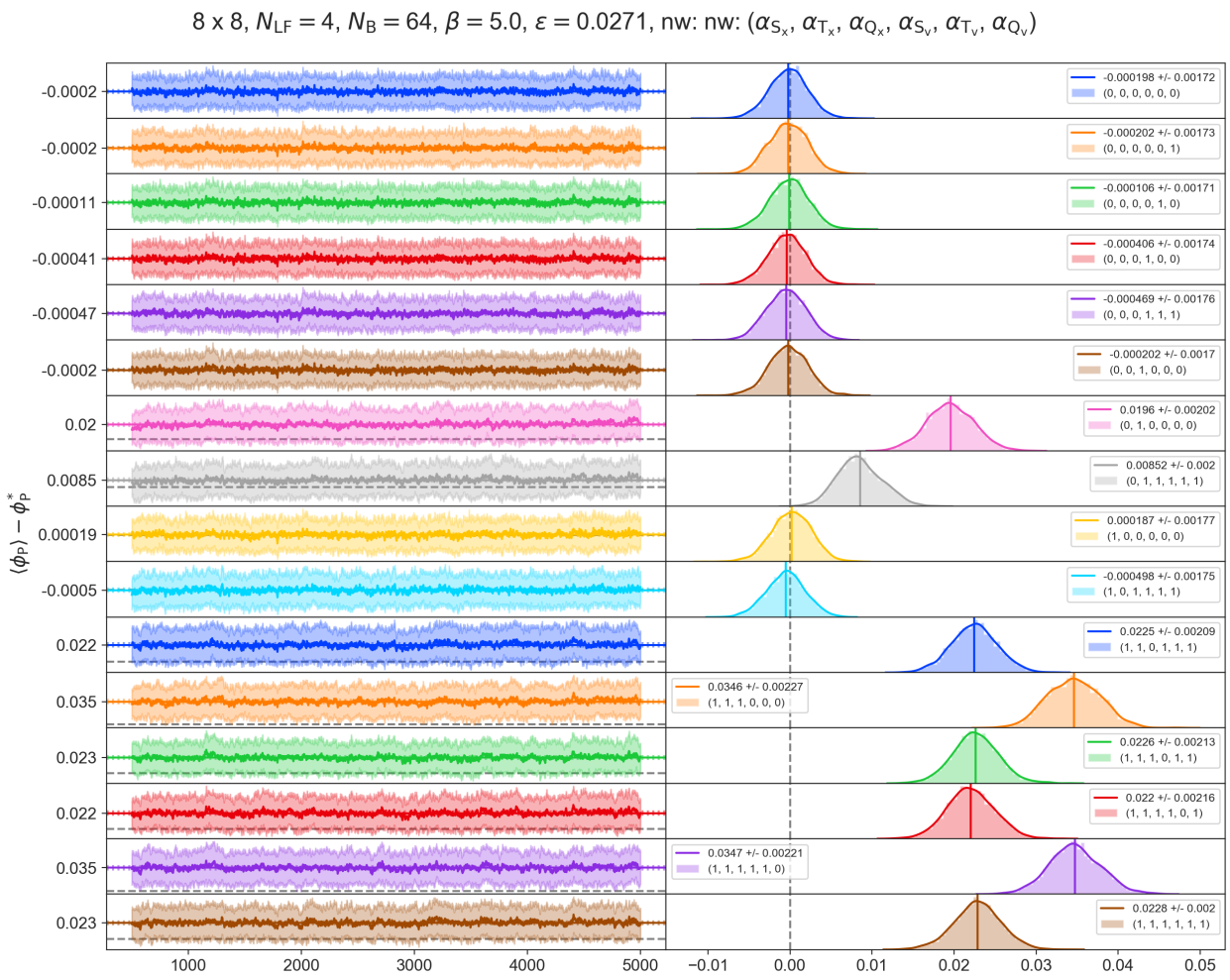


Figure 15: Trace plots and histograms from an inference run with $N_{LF} = 4$ and various values of `net weights`.

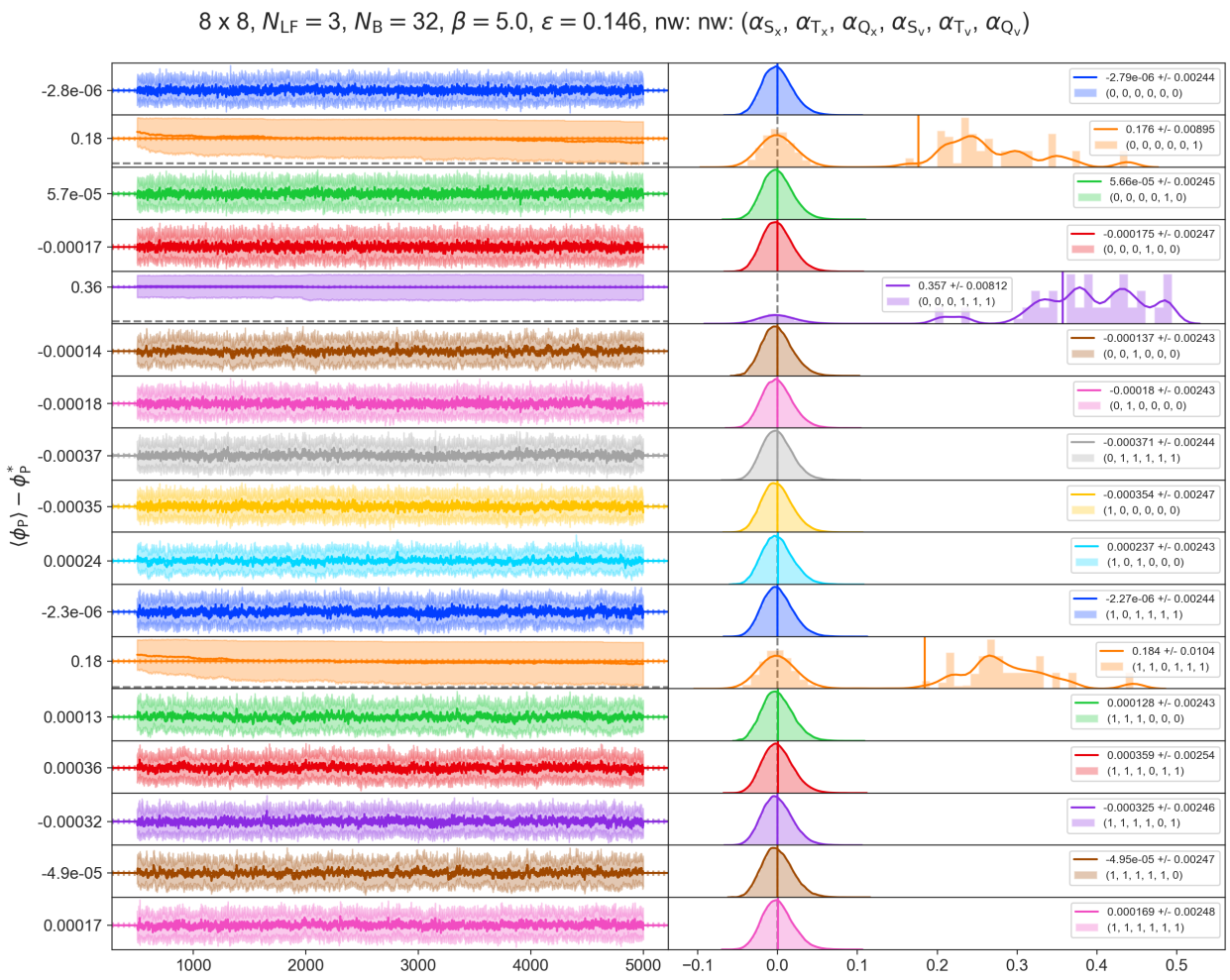


Figure 16: Trace plots and histograms from an inference run with $N_{\mathrm{LF}} = 3$ and various values of `net weights`.

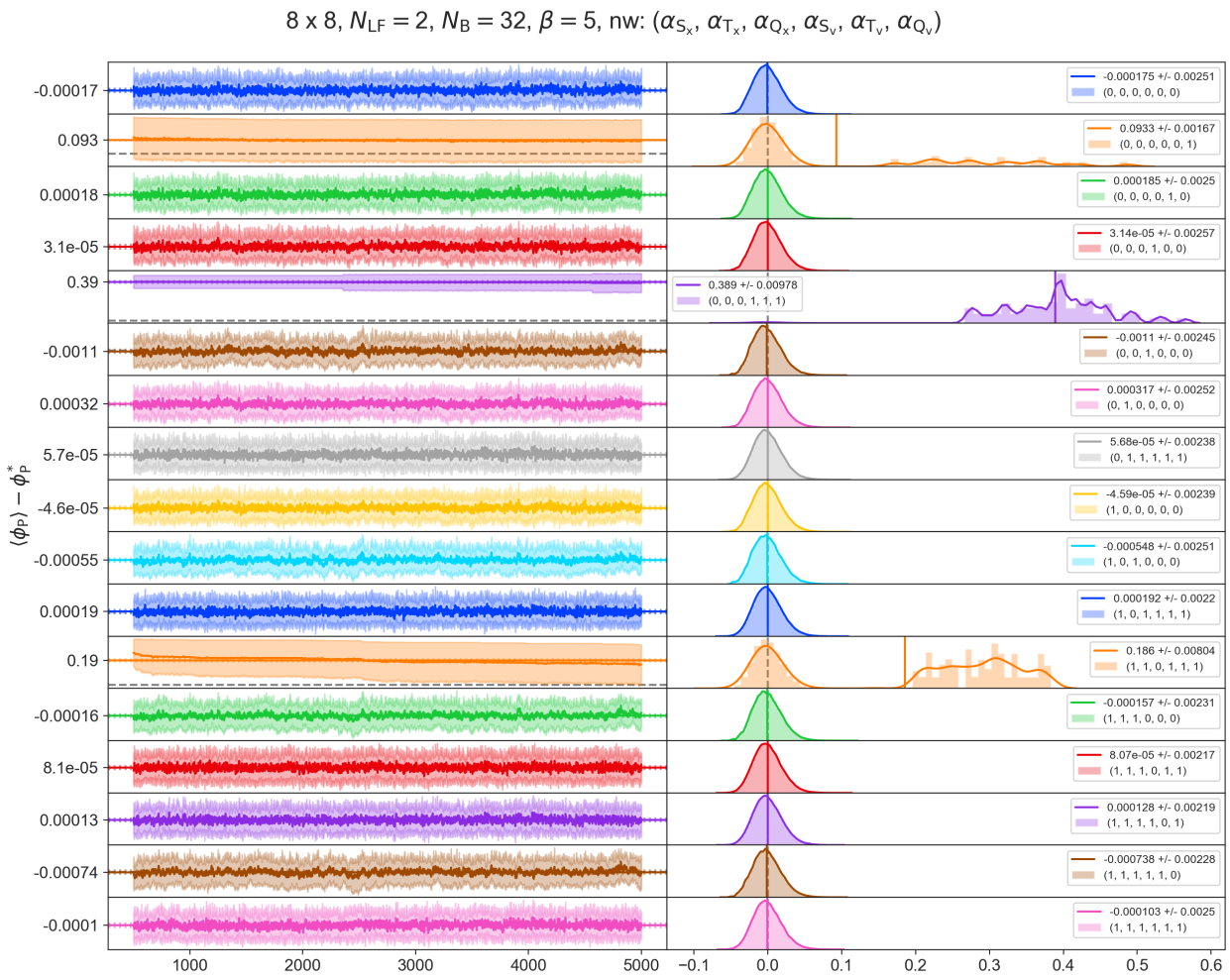


Figure 17: Trace plots and histograms from an inference run with $N_{\text{LF}} = 2$ and various values of `net weights`.

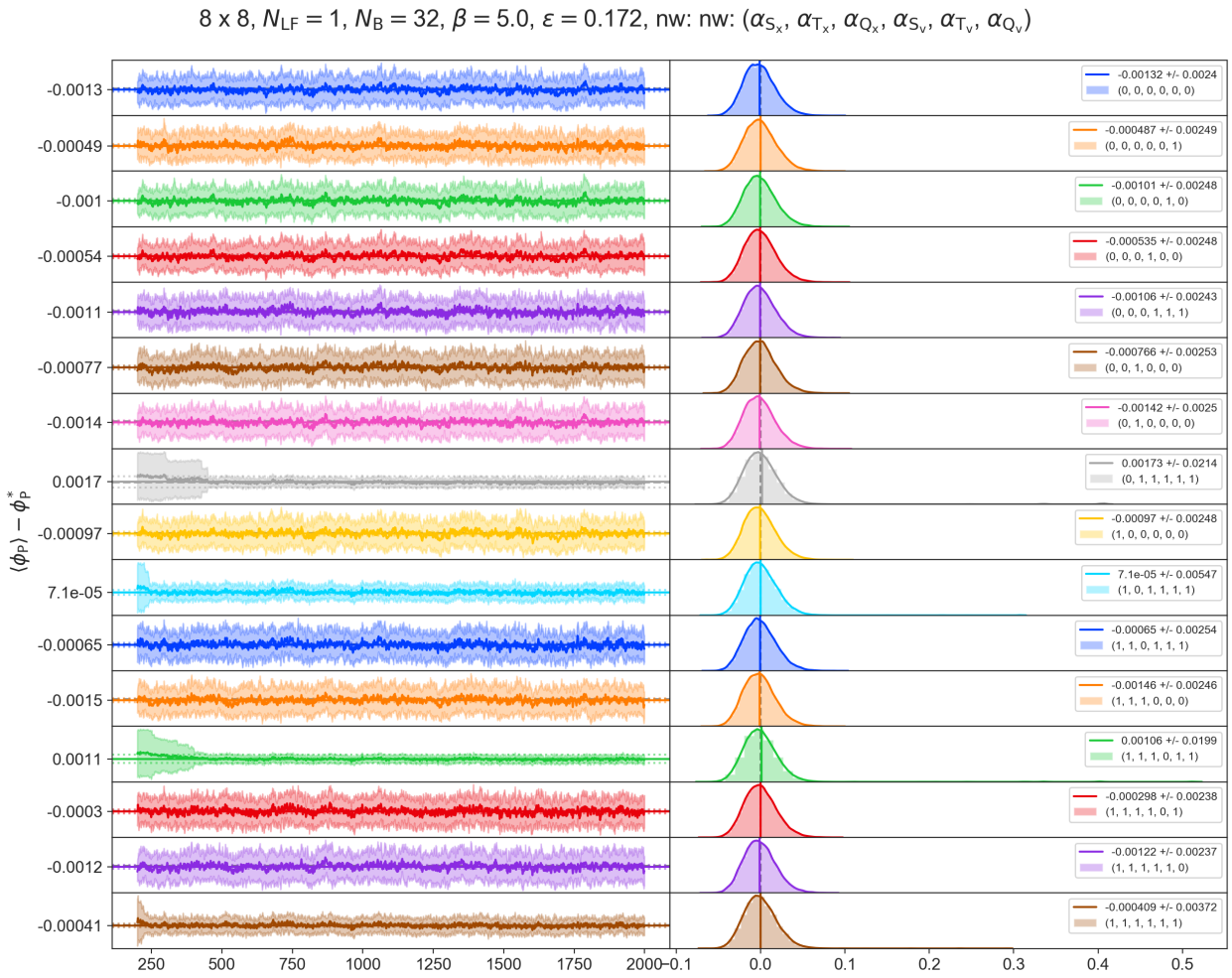


Figure 18: Trace plots and histograms of $\delta\phi_p$ from an inference run with $N_{\text{LF}} = 1$ and various values of `net weights`.

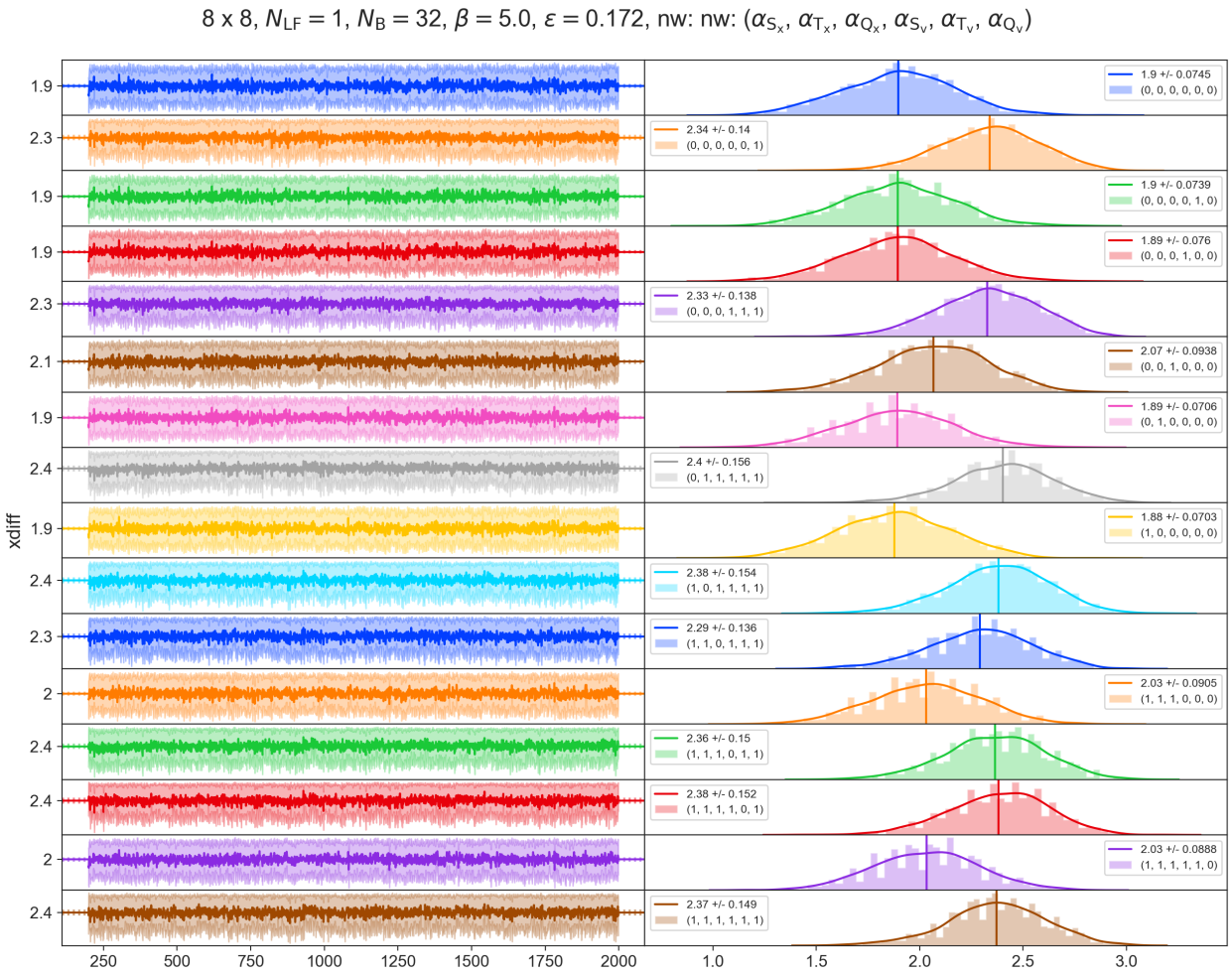


Figure 19: Trace plots and histograms of $\delta x = x^{(i+1)} - x^{(i)} \mod 2\pi$ from an inference run with $N_{\mathrm{LF}} = 1$ and various values of `net weights`.

14 | Updates 12/16/2019

All of the following results were trained using Horovod on COOLEY for 1×10^5 training steps (2× previous training length).

As a measure of how well the trained sampler performs, we can introduce the *tunneling rate* (γ), as

$$\gamma = \frac{1}{M} \sum_{m=1}^{M} |\mathbb{Q}^{(m+1)} - \mathbb{Q}^{(m)}| \tag{57}$$

where $\mathbb{Q} \in \mathbb{Z}$ is the topological charge, and M denotes the number of accept/reject steps the sampler was ran for. This tells us the amount by which we can expect the topological charge to change per step, and is a useful metric for measuring how well the sampler is able to explore different topological sectors.

15 | Updates 01/01/2020

- In order to further pin down the cause of the bias in the average plaquette, it is important to verify that for a given N_{LF}, the results are consistent when using different random seeds.
- From recent data, it seems that the function T_x has the largest contribution to the plaquette difference, as seen in Figures 14, 15.
  - Try training the model with different combinations of T_x, T_v set to 0.
  - For those models trained with T_x, $T_v = 1$, try running inference with T_x, $T_v = 0$.
- To further simplify the model, training runs were carried out at a fixed step size ε.
- In order to effectively interpret the results of a given training/inference run, it is useful to be able to visualize multiple distributions (e.g. the plaquette bias $\delta\phi_P$, tunneling rate γ, acceptance probability $A(\xi'|\xi)$, and the average distance traveled between subsequent configurations) simultaneously. Included below are violinplots of each of these distributions.

For all of the results included below the following parameters were used:

- $N_s = N_t = 8$ (i.e. 8×8 lattice)
- batch size $N_B = 64$
- learning rate $\alpha_{\text{init}} = 1 \times 10^{-3}$, decayed by 0.96 after 25,000 training steps with `staircase = True`.
- $N_{\text{train}} = 1 \times 10^5$ using 16 nodes, 2 workers / node via `horovod` on COOLEY.
- $N_{h_1} = N_{h_2} = 100$ nodes in each of the hidden layers.
- $\beta = 5$

In creating each of the below figures, data was included only if the average acceptance probability was greater than 10%.

Additionally, it can be seen in some of the figures that the distance traveled, (the rightmost column in the violinplots) is centered around 0. This is the result of a somewhat misleading calculation that failed to account for the direction of the leapfrog update, causing the forward and backward directions to be treated equally. Moreover, for some of the plots the average distance traveled was calculated using the Euclidean $L2$ metric, $\|x^{(i+1)} - x^{(i)}\|_2^2$ instead of the more appropriate cos metric, $1 - \cos(x^{(i+1)} - x^{(i)})$.

15.1 Statistics

For a given inference run consisting of M accept/reject steps, we obtain an array of lattice configurations with shape: $[M, N_b, 2V]$, where N_b is the batch size (i.e. number of chains ran in parallel), and $V = N_s \times N_t$ is the volume of the lattice.

For concreteness, we describe below how statistics are calculated for the plaquette difference $\delta\phi_P$, although an identical approach applies for each of the observables.

- From our array of configurations, we first compute $\delta\phi_P$ separately for each of the N_b chains, resulting in an array of shape $[M, N_b]$.
- In order to account for thermalization, the first 25% of the data is ignored, so we are left with $[3M/4, N_b]$ individual data points.
- For each of the plots in the *first row*, this data was flattened into a single array of shape $[1, 3M/4 \times N_b]$.
- For each of the plots in the *second row* however, statistics were calculated using bootstrap resampling as represented by the pair of angled brackets, e.g. $\langle \delta\phi_P \rangle$.

16 | Updates: 01/21/2020

In yet another attempt to identify the source of the bias in the plaquette, it was decided that it would be beneficial to develop tensorflow-independent code that is capable of running inference on a trained model. This was done by exporting the weights from the trained neural network to a `.pkl` file after training. From these weights, we are then able to reconstruct each of the functions $S_x, T_x, Q_x, S_v, T_v, Q_v$. By writing this alternative inference code in pure numpy, we are able to avoid some of the inherent complexities in tensorflow while also providing a more flexible interface. For example, when running inference using the saved tensorflow graph, we are forced to use the same number of leapfrog steps N_{LF}, whereas we are free to change this when using the numpy code.

- Once the numpy inference code was finished, it was then ran on all previously saved models.
- Initial results seemed to have a minor discrepancy when compared to the inference runs generated using tensorflow.
  - Systematic debugging led to the identification of an internal bug in tensorflow related to exporting the weights from the saved model.
  - Explicitly, when calling `layer.get_weights ()` on a particular layer in the model (as suggested by the official tensorflow documentation), the "weight" matrix was returned correctly, but the associated bias vector returned was always $[0, 0, \ldots, 0]$.
- Because of this, all of the existing inference data generated from this new numpy code was incorrect and needed to be regenerated.

17 | Updates: 03/16/2020

- **Symplectic**
  - With $\alpha_{S_i} = 0$ for $i = x, v$, (only T_i, Q_i terms), this should be trivial.
  - Even with S terms, the formulas are pretty simple, and should be easy to verify once we can be sure $S = 0$ is working properly.
  - For now it is probably still enough to only consider T_x since we mainly want to understand the source of the bias.
  - Once we understand that we can add more terms and work on tuning it better.
- **Reversibility**
  - We can check that the trained sampler is reversible using the following procedure:
    1. Randomly choose the direction d to create the initial state $\xi = (x, v, d)$.
    2. Run the dynamics and flip the direction:

$$\mathbf{FL}\xi = \mathbf{F}\xi' = (x', v', -d) \tag{58}$$

    3. Run the dynamics and flip the direction again:

$$\mathbf{FL}\xi' = \mathbf{F}\xi'' = (x'', v'', d) \tag{59}$$

    4. Check the difference:

$$\delta x = x - x'' \tag{60}$$
$$\delta v = v - v'' \tag{61}$$

  - **(AI2):** The violations should get worse as the volume increases, so it is probably best to formulate the network in terms of the group variables $(\cos\phi_\mu(i), \sin\phi_\mu(i)) = e^{i\phi_\mu(i)}$ instead of the algebra $\phi_\mu(i)$. This would easily apply to higher groups as well. For $U(1)$, this doubles the size of the inputs, but should work better overall.
  - Having the network treat 0 and 2π differently is a potential source of reversibility violation, though it may be small in practice. Continue looking for evidence of this.
  - Using the above criterion it was observed that the sampler does indeed violate reversibility, as shown in Fig 20.
  - In order to identify the root cause of the reversibility violation, I am currently working on stepping through the sub-updates of the dynamics code and checking reversibility at each step.
- **Ergodicity**
  - Technically, this may be an autocorrelation problem, but in practice it is difficult to distinguish them.
    * This may be the main problem. L2HMC seems to only learn certain types of updates, but fails at general mixing.
  - **(AI3)** In principle, there should be some initial conditions that give a negative bias, and some positive. Mapping out the bias distribution for different seeds may help confirm this, though the distribution may not be symmetric, and could have a large tail on one side.
  - **(AI4)** Alternating HMC with L2HMC is an idea to fix this.
  - If L2HMC mixes poorly on its own, then we may need to run mostly HMC.
  - Perhaps, running N updates of HMC and 1 L2HMC, for varying N, will avoid the L2HMC bias and show an improvement over either alone.

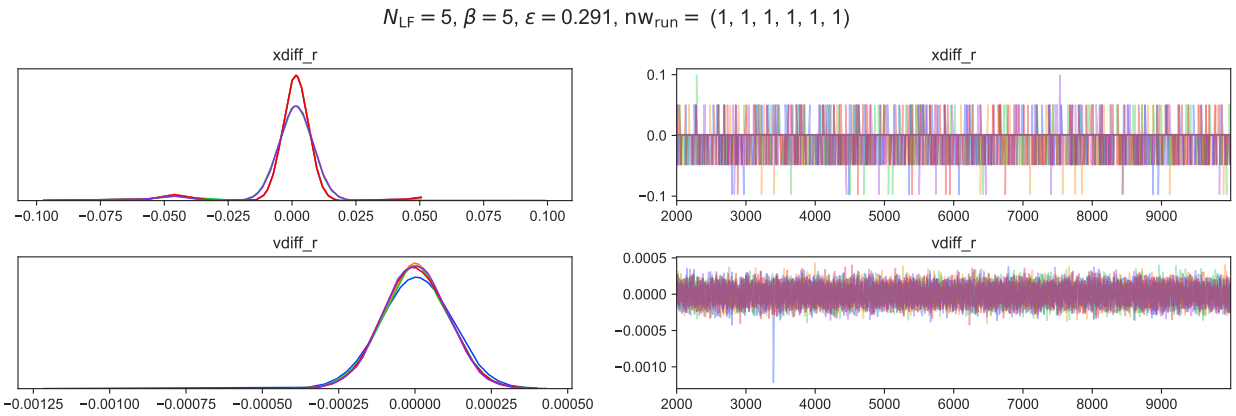


Figure 20: Traceplot of average differences $\langle \delta x \rangle$, $\langle \delta v \rangle$ demonstrating the reversibility violation.

- We can periodically switch between L2HMC and generic HMC during inference to see if there is any benefit. An example of this procedure is shown in Fig 21.

☑ **(AI3, AI4):** No evidence in recent data showing a negative bias, although it may be that the distribution is *very* one sided. Continuing to look for counter-examples.

- **Training**
  - **(AI5)** Run more tests with current code at $T_x = 1$ to further map this distribution and see how the average distance, δx_{out} and acceptance, $A(\xi'|\xi)$ after training correlate with the bias.
  - **(AI6)** The initial tests of alternate loss functions seemed promising. Continue to explore alternate loss functions.
    * Maybe $|\delta x| * A^2(\xi'|\xi)$
    * Or others that avoid anything we can hopefully determine from **(AI5)** (or other tests), that correlate with increased bias.
  - Try running on $O(2)$ model in 1D compare against Yannicks results.
- **Code/scaling**
  - Longer-term, we want to write L2HMC in QEX for Aurora. This would potentially be faster and easier to scale up.
    * However, the full dense layer won't scale well, and would need to be replaced eventually.
  - **(AI7)** One option is to replace the dense weight matrix with a low-rank approximation. This could be investigated by taking the SVD of the weight matrix and looking at how the singular values fall off.
    * Could also replace the weight matrix after training (on the dense matrix), then run inference on a low-rank approximation to see how it compares.
  - **(AI8)** We could experiment with a few other variants that would be easy to implement and scale well, like a local connection (stencil) in combination with a low-rank fully connected layer. This would be relatively easy to implement in QEX.
  - In the 2D case, the singular value decomposition (SVD) of a weight matrix W in the network can be written as:

$$W = USV^H \tag{62}$$

    where $S = \text{s}$ contains the singular values of W and U, V^H are unitary. The rows of V^H are the eigenvectors of $W^H W$ and the columns of U are the eigenvectors of WW^H. In both cases the corresponding (possibly non-zero) eigenvalues are given by s^2.

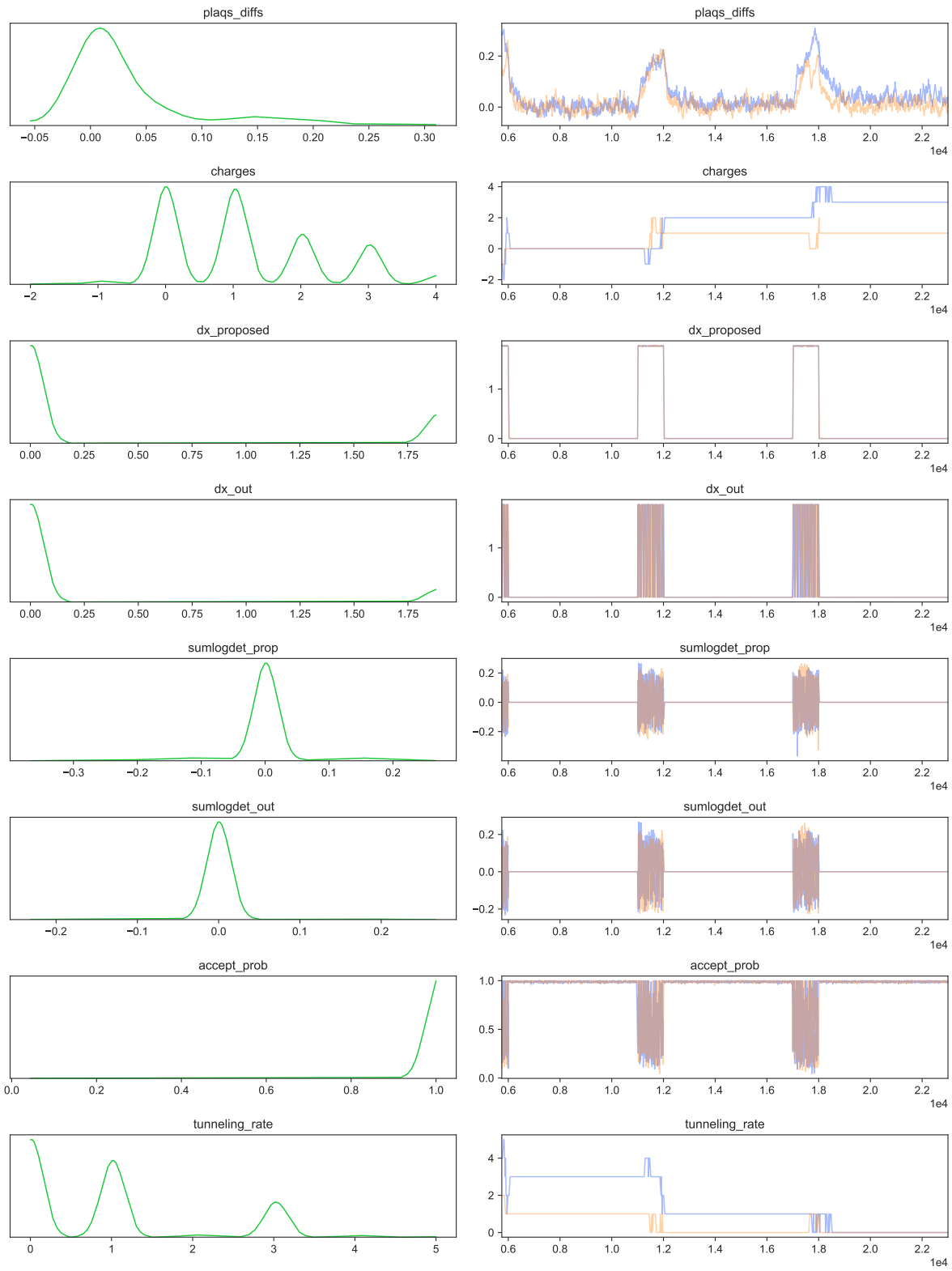


Figure 21: Results obtained by periodically mixing between L2HMC and HMC during inference.

– The amount of overall variance explained by the i^{th} pair of SVD vectors is given by $s_i^2 / \sum_j s_j^2$, where s_j are the singular values (diagonal of S).

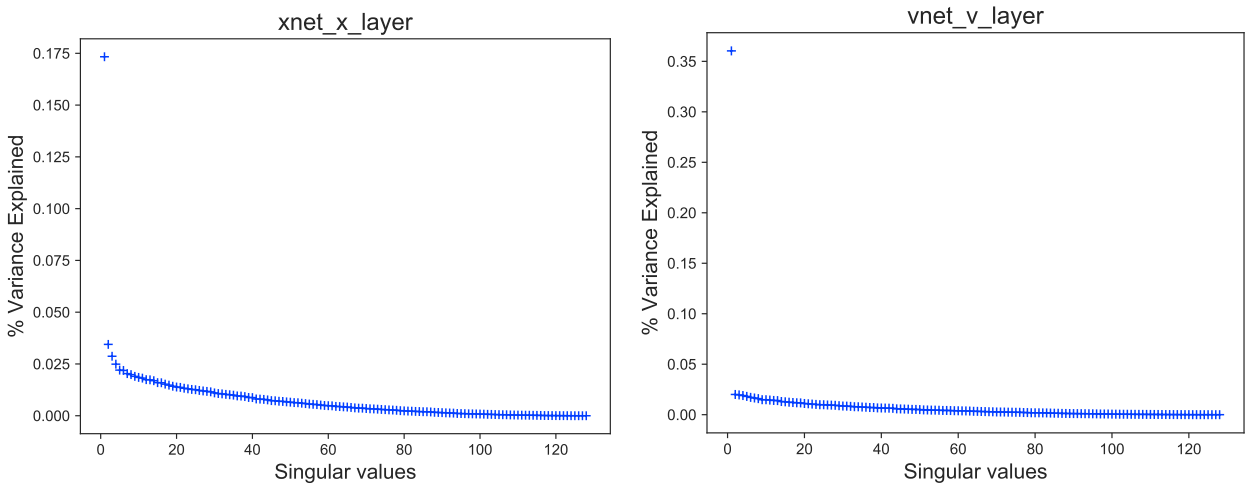


Figure 22: Plots showing the percent of the total variance explained by the i^{th} singular value for two layers in our neural network.

17.1 Simplifying the x-update

One technique for determining the source of the bias in the plaquette is to remove/simplify individual components in the network, and see if any of these changes fixes the issue.

One possible simplification that we have begun to explore is to combine the two x sub-updates into a single update by explicitly setting the "site" masks in Eq. 20 24 be

$$m^t = [1, 1, \ldots, 1] \tag{63}$$
$$\bar{m}^t = [0, 0, \ldots, 0] \tag{64}$$

for $t = 1, \ldots, N_{\text{LF}}$.

In this case, the forward update ($d = 1$), becomes:

$$x' = x \odot \exp\left(\varepsilon S_x(\zeta_2)\right) + \varepsilon \left(v' \odot \exp\left(\varepsilon Q_x(\zeta_2)\right) + T_x(\zeta_2)\right) \tag{65}$$
$$x'' = x' \tag{66}$$
$$= x \odot \exp\left(\varepsilon S_x(\zeta_2)\right) + \varepsilon \left(v' \odot \exp\left(\varepsilon Q_x(\zeta_2)\right) + T_x(\zeta_2)\right) \tag{67}$$
$$\tag{68}$$

And similarly for the backwards (d=-1) update:

$$x' = x \tag{69}$$
$$x'' = \left[x' - v' \odot \varepsilon \left(\exp(\varepsilon Q_x(\zeta_3)) + T_x(\zeta_3)\right)\right] \odot \exp\left(-\varepsilon S_x(\zeta_3)\right) \tag{70}$$

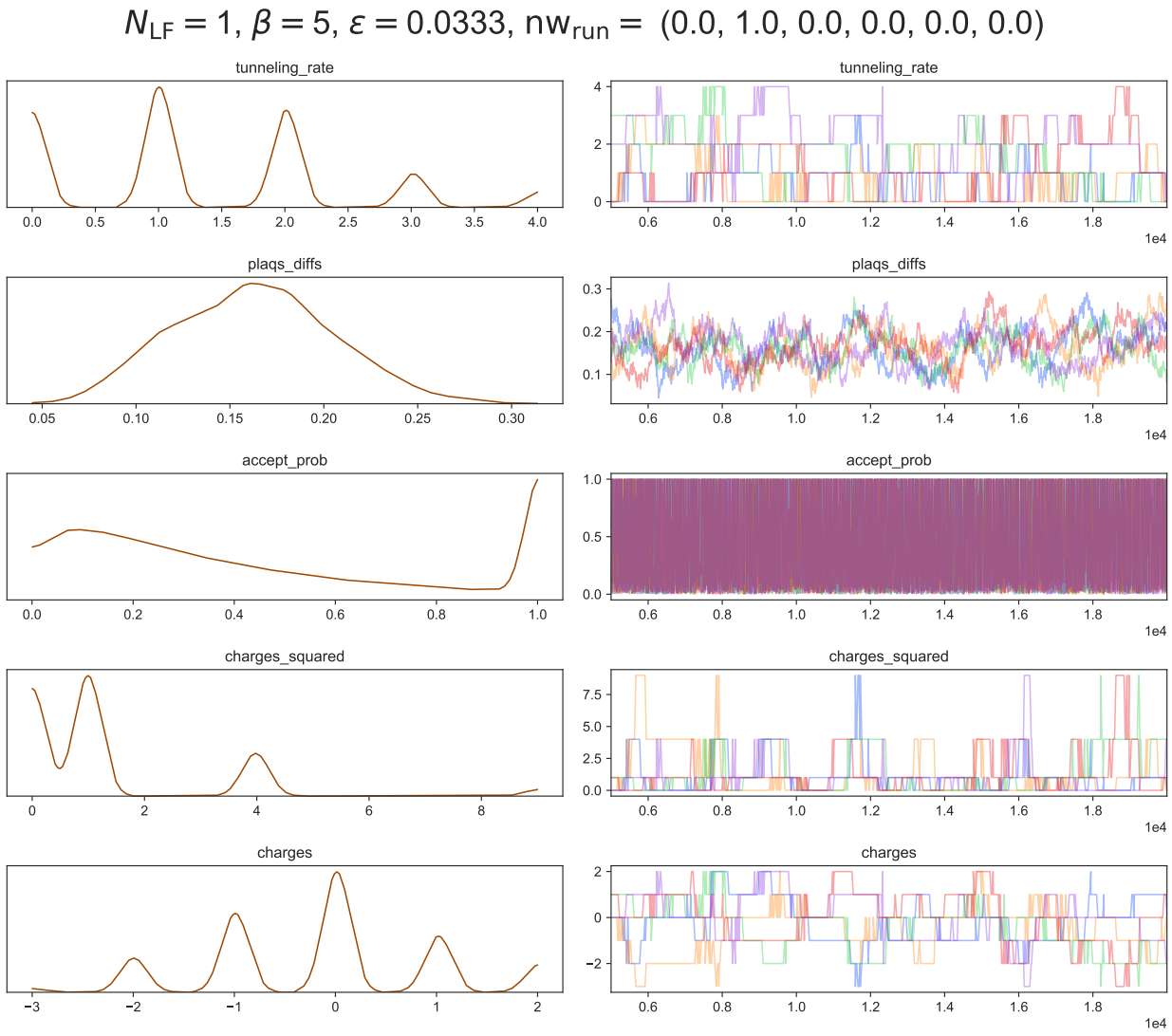


Figure 23: Using the original (randomly) assigned site masks, we see the bias is present.

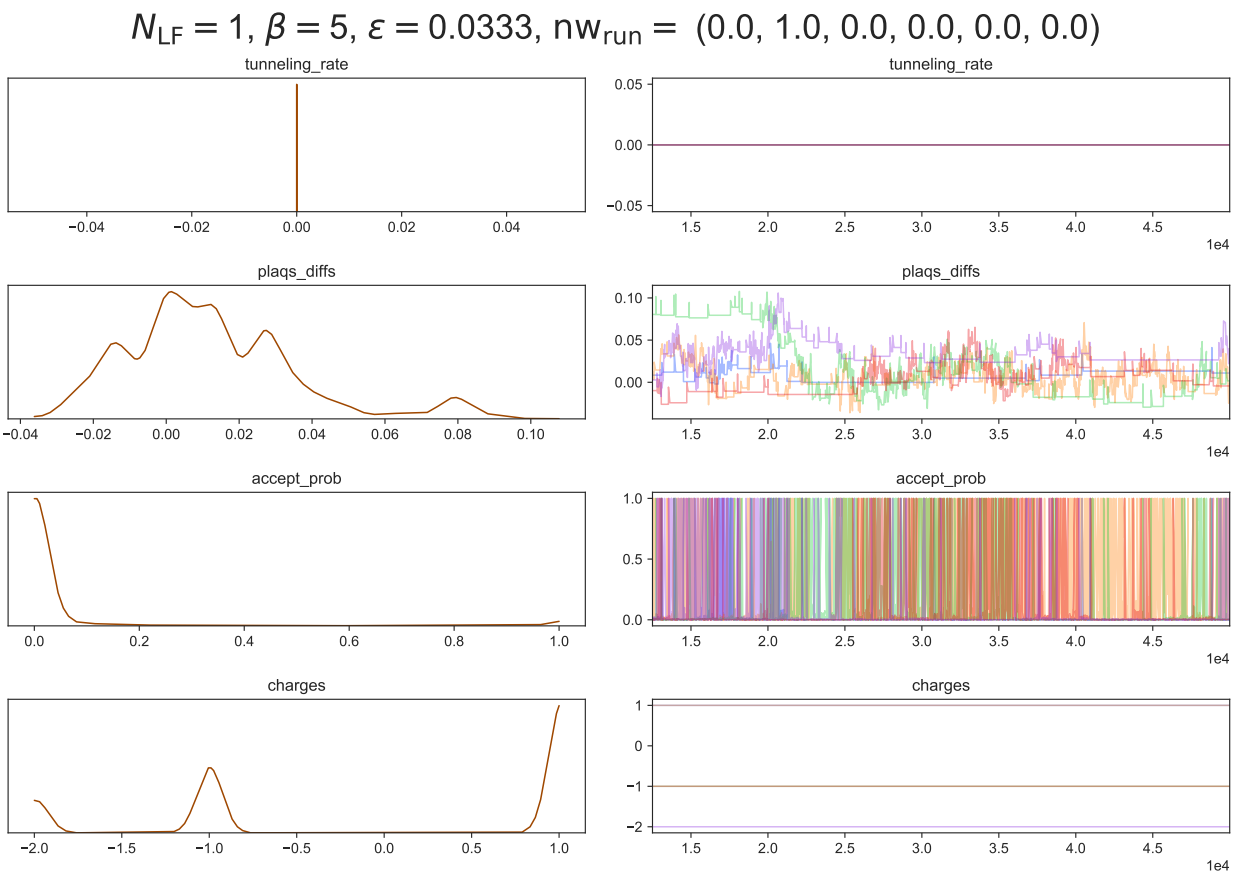


Figure 24: Using the combined x sub-updates with $m^t = [1, 1, \ldots, 1]$ and $\bar{m}^t = [0, 0, \ldots, 0]$. We see that the bias has slightly improved, although both the acceptance rate and tunneling rate appear to suffer dramatically.

18 | Updates: 03/31/2020

18.1 Periodicity

- When using the angular representation, $x \equiv \phi_\mu(k) \in [0, 2\pi)$ of the link variables $U_\mu(k) = e^{i\phi_\mu(k)} \in U(1)$, it seems like the main problem (possibly in addition to issues with reversibility) is that the output is discontinuous when the input moves between 0, and $2\pi - \varepsilon$ ($\varepsilon \ll 1$).
- Additionally, the network itself is not periodic, i.e. if $x \longrightarrow x + n\pi$, the $x \odot \exp(\varepsilon S_x(\zeta_1))$ term in the x-update becomes
$$x \odot \exp(\varepsilon S_x(\zeta_1)) \longrightarrow (x + n\pi) \odot \exp(\varepsilon S_x(\zeta_1)) \tag{71}$$
and we're left with an additional $n\pi \odot \exp(\varepsilon S_x(\zeta_1))$ term.
- We can avoid this issue by using the $\vec{x} \equiv \big[\cos\phi_\mu(k), \sin\phi_\mu(k)\big]$ representation as input to the network, which then updates the cos and sin terms separately.

19 | Updates: 04/13/2020

19.1 Changes to Network

- Use Cartesian representation $\big[\cos\phi_\mu(x), \sin\phi_\mu(x)\big]$ 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.
- Under this new representation, the bias in the average plaquette no longer seems to be an issue.
- However, there is no noticeable improvement in the tunneling rate when compared to generic HMC.
- **TODO:**
  - ☐ Continue testing with additional hidden layers.
  - ☐ Add convolutional/pooling layers at beginning of network to ensure translational invariance.

	tunneling events	tunneling rate	accept prob
chain 1	1	0.000125	0.684 ± 0.003
chain 2	4	0.000500	0.695 ± 0.003
chain 3	4	0.000500	0.696 ± 0.003
chain 4	3	0.000375	0.698 ± 0.003
average	**2.4**	**0.0003**	**0.696 ± 0.003**

Table 1: Inference results for trained **L2HMC** sampler ran for $N = 1 \times 10^4$ accept/reject steps at $\beta = 5$.

	tunneling events	tunneling rate	accept prob
chain 1	2	0.000250	0.279 ± 0.004
chain 2	1	0.000125	0.271 ± 0.004
chain 3	3	0.000375	0.281 ± 0.004
chain 4	4	0.000500	0.286 ± 0.004
average	**2**	**0.00025**	**0.282 ± 0.004**

Table 2: Inference results for generic **HMC** sampler ran for $N = 1 \times 10^4$ accept/reject steps at $\beta = 5$.

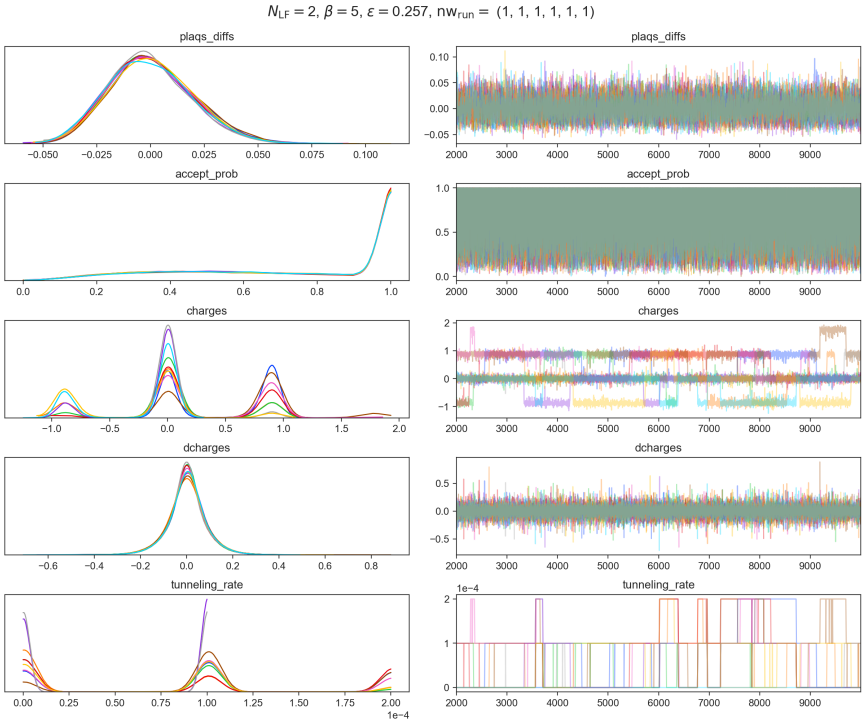


Figure 25: Inference results from trained **L2HMC** sampler.

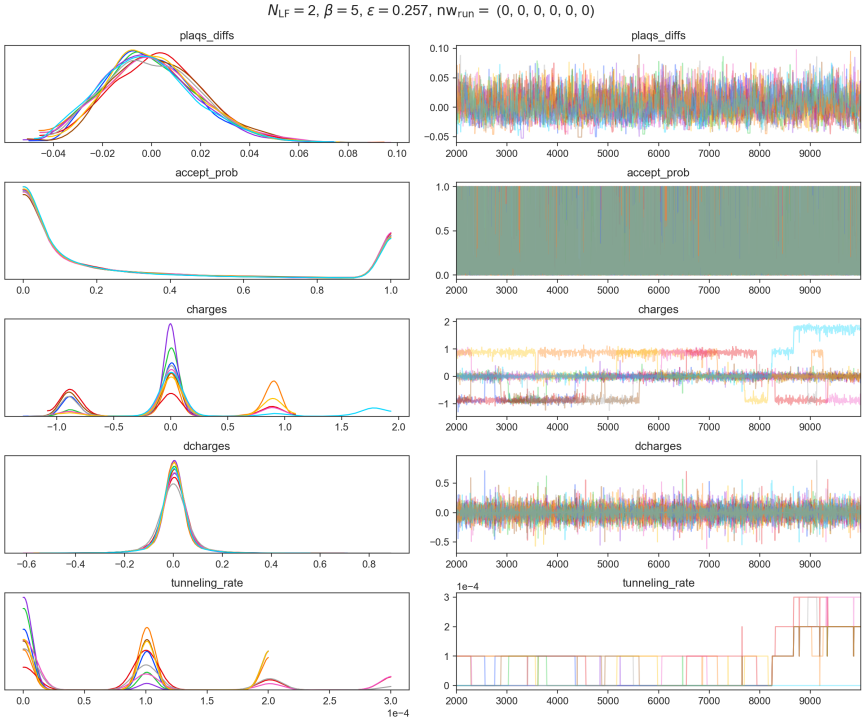


Figure 26: Inference results from generic **HMC** sampler.

19.2 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

$$\mathbb{Q} \equiv \frac{1}{2\pi} \sum_{\substack{x;\mu,\nu \\ \nu>\mu}} \sin\left(\phi_{\mu\nu}(x)\right) \tag{72}$$

for

$$\phi_{\mu\nu}(x) = \phi_\mu(x) + \phi_\nu(x+\hat{\mu}) - \phi_\mu(x+\hat{\nu}) - \phi_\nu(x) \tag{73}$$

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*, $\delta_P(\xi', \xi)$

$$\delta_P(\xi', \xi) = \sum 1 - \cos\left(\phi'_{\mu\nu}(x) - \phi_{\mu\nu}(x)\right) \tag{74}$$

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

$$\delta_{\mathbb{Q}}(\xi', \xi) = \left[\overbrace{\frac{1}{2\pi} \sum \sin\left(\phi'_{\mu\nu}(x)\right)}^{\mathbb{Q}'} - \overbrace{\frac{1}{2\pi} \sum \sin\left(\phi_{\mu\nu}(x)\right)}^{\mathbb{Q}} \right]^2 \tag{75}$$

$$= (\mathbb{Q}' - \mathbb{Q})^2 \tag{76}$$

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

$$\ell_{\lambda_P}(\xi', \xi, A(\xi'|\xi)) = \frac{\lambda_P^2}{\delta_P \cdot A(\xi'|\xi)} - \frac{\delta_P \cdot A(\xi'|\xi)}{\lambda_P^2} \tag{77}$$

$$\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi)) = \frac{\lambda_{\mathbb{Q}}^2}{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)} - \frac{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)}{\lambda_{\mathbb{Q}}^2} \tag{78}$$

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

$$\mathcal{L}(\theta) = \mathbb{E}_{p(\xi)}\left[\alpha_P \cdot \ell_{\lambda_P} + \alpha_{\mathbb{Q}} \cdot \ell_{\lambda_{\mathbb{Q}}}\right] + \mathbb{E}_{q(\xi)}\left[\alpha_P \cdot \ell_{\lambda_P} + \alpha_{\mathbb{Q}} \cdot \ell_{\lambda_{\mathbb{Q}}}\right] \tag{79}$$

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

20 | Updates: 04/28/2020

20.1 Additional changes to loss function

Instead of working with the mixed loss function, $\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi))$ from Eq.78, we can focus exclusively on the second term, which is directly related to the tunneling rate. The equation then becomes

$$\ell_{\lambda_{\mathbb{Q}}}(\xi', \xi, A(\xi'|\xi)) = -\frac{\delta_{\mathbb{Q}} \cdot A(\xi'|\xi)}{\lambda_{\mathbb{Q}}^2} \tag{80}$$

$$= -\left(\frac{\mathbb{Q}' - \mathbb{Q}}{\lambda_{\mathbb{Q}}}\right)^2 \cdot A(\xi'|\xi) \tag{81}$$

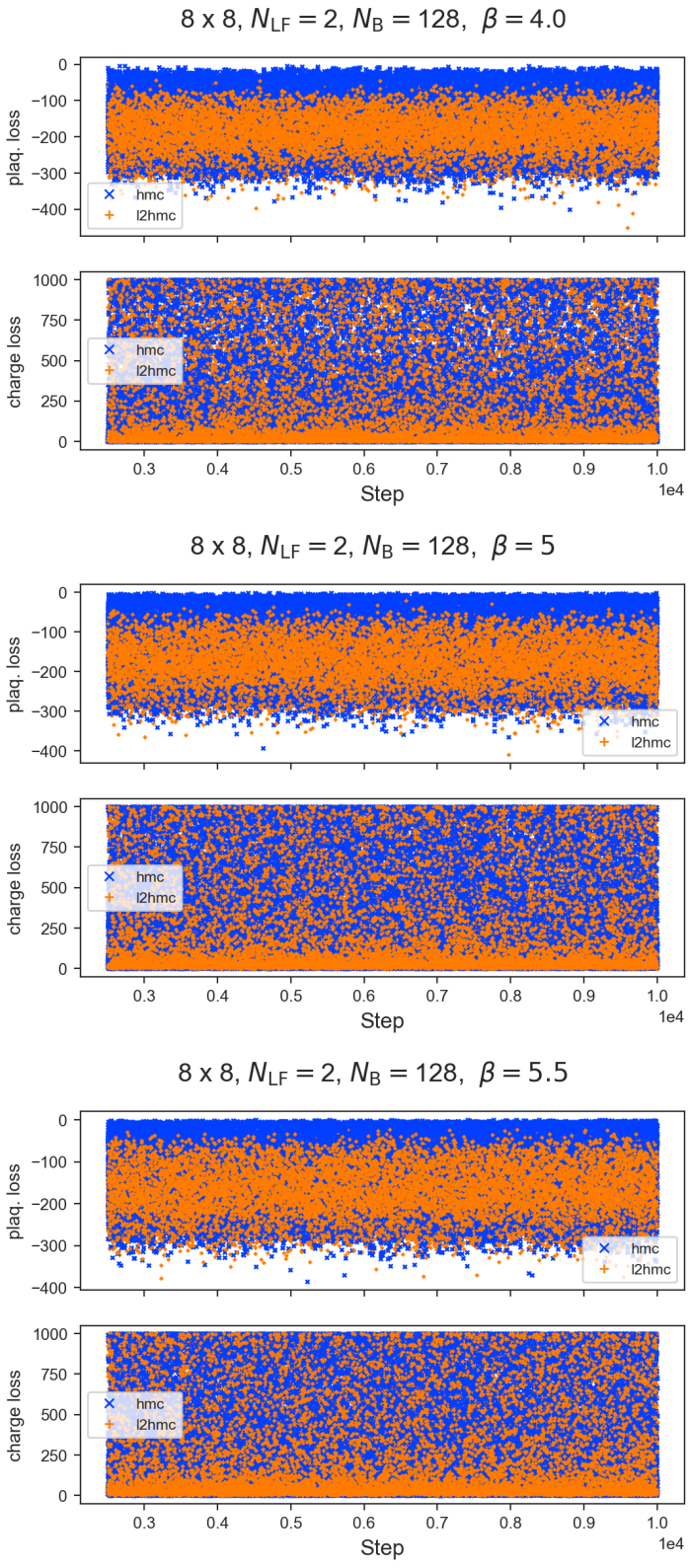
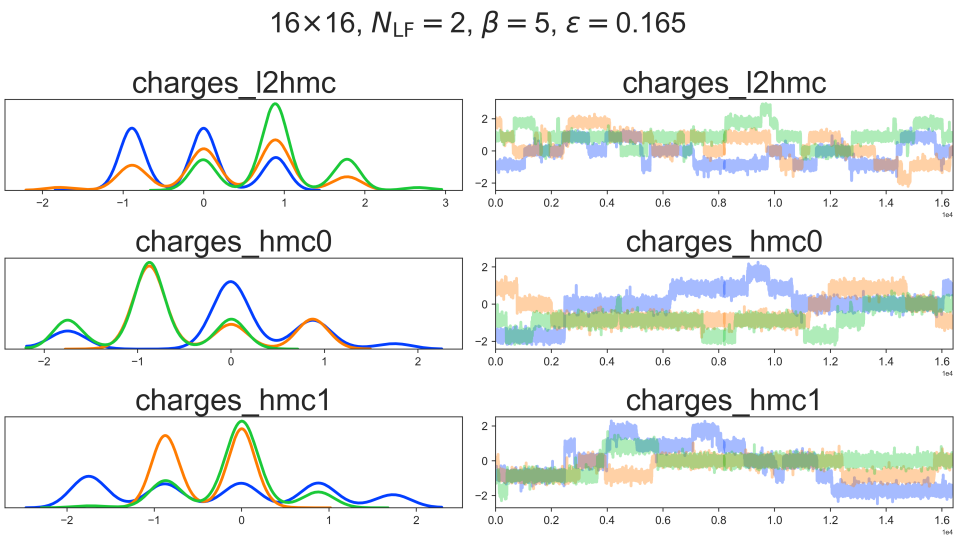
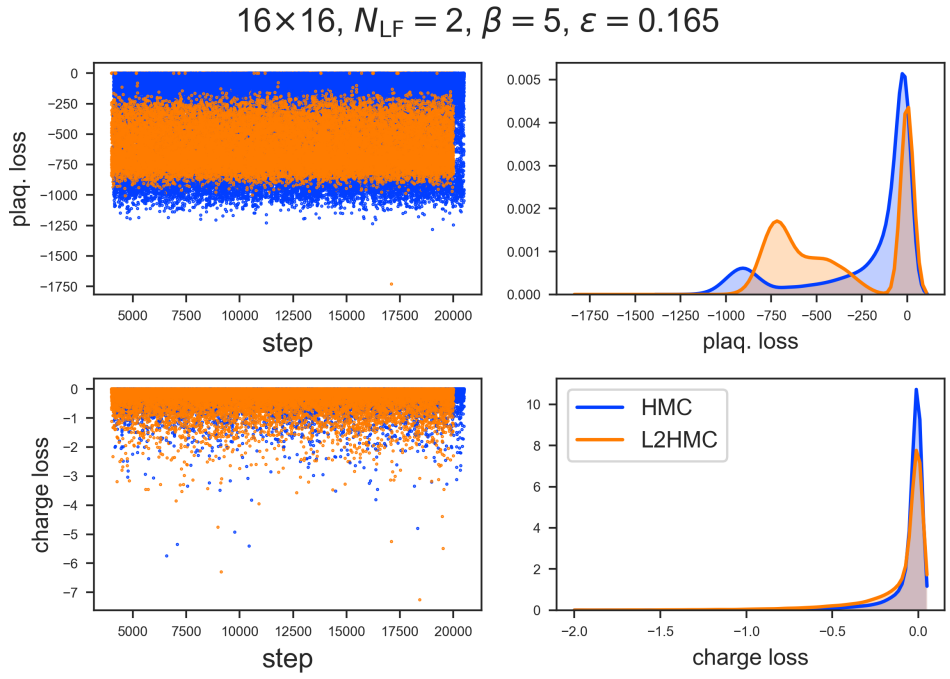


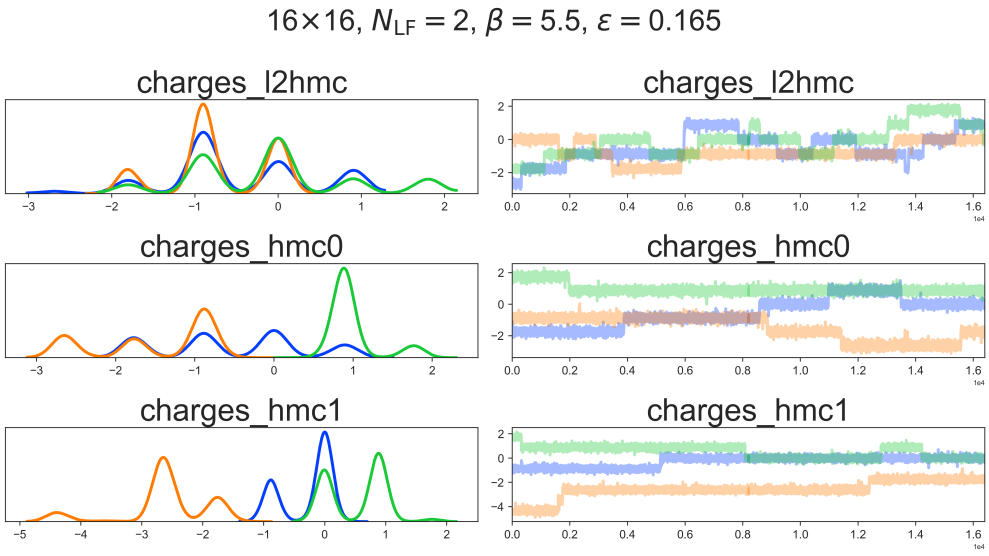
Figure 27: Loss comparisons between L2HMC and HMC at different values of β.



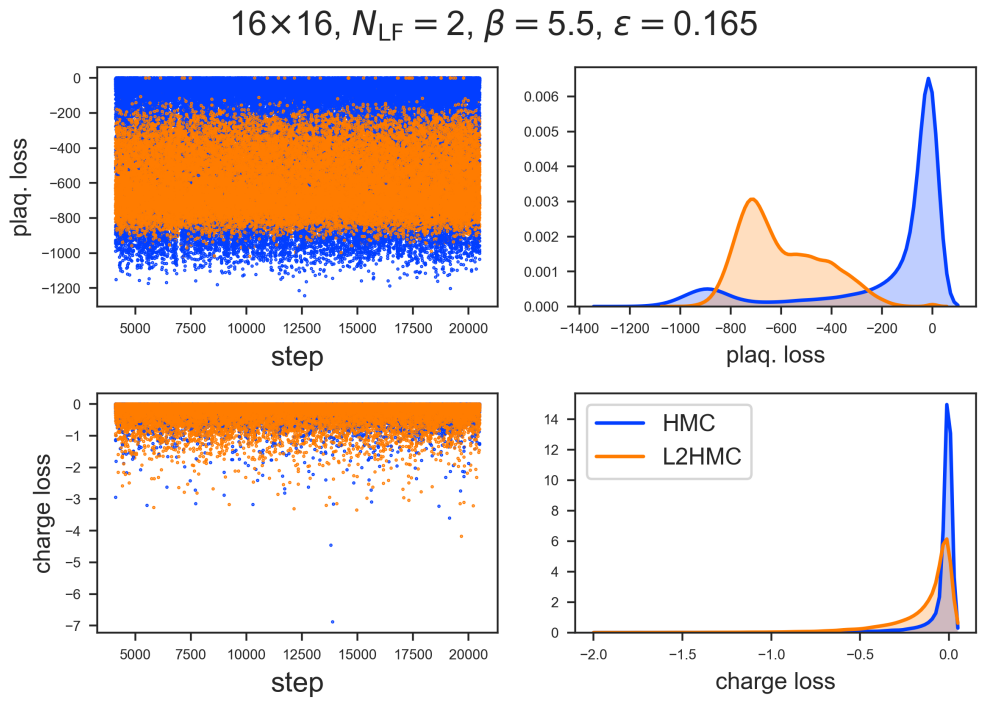
(a) Topological charges for both HMC and L2HMC runs at $\beta = 5$.



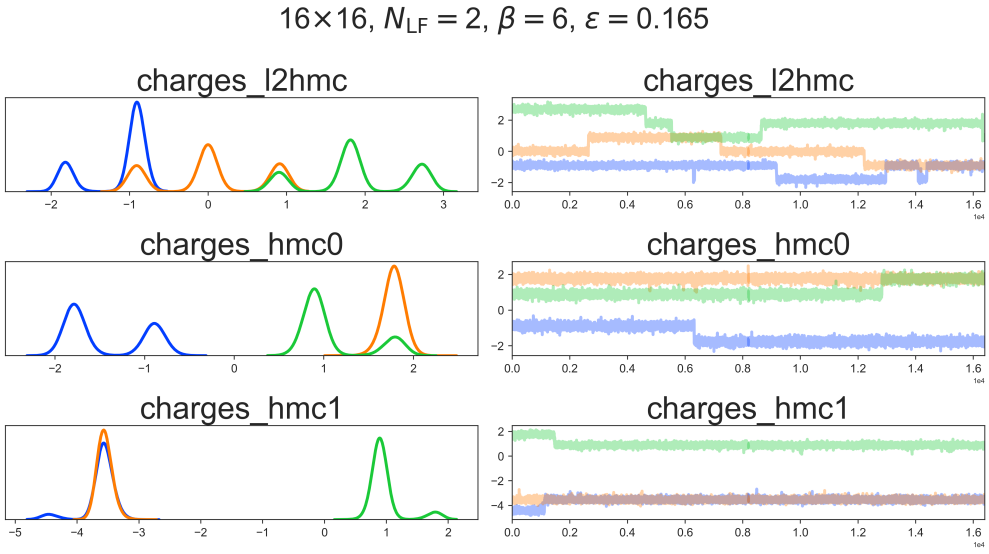
(b) Comparison of the plaquette and topological loss terms at $\beta = 5$.



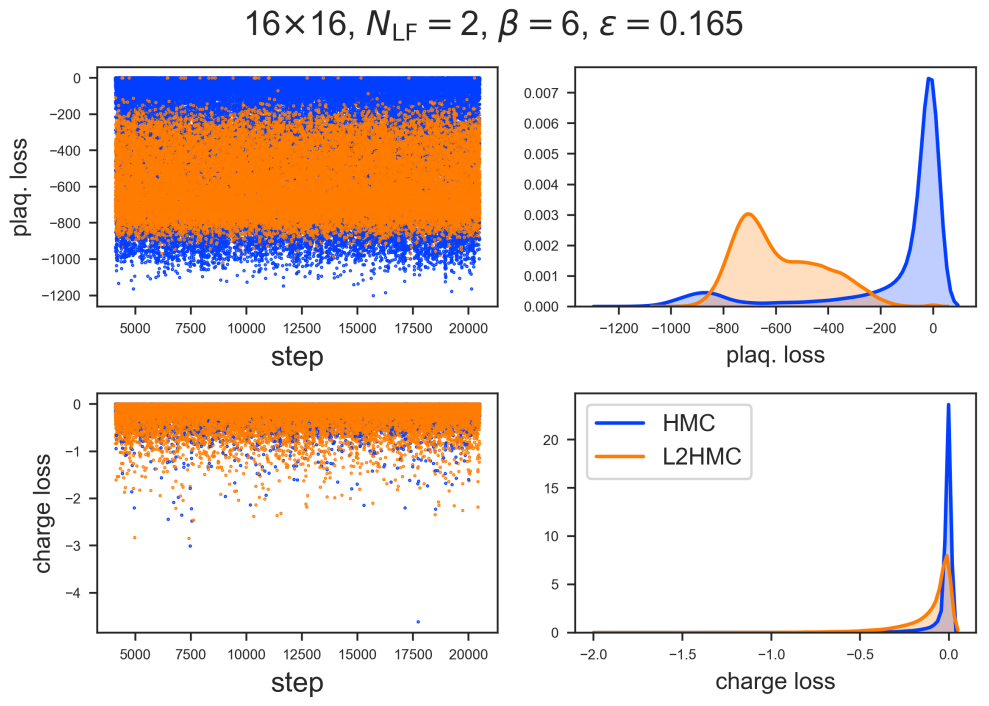
(a) Topological charges for both HMC and L2HMC runs at $\beta = 5.5$.



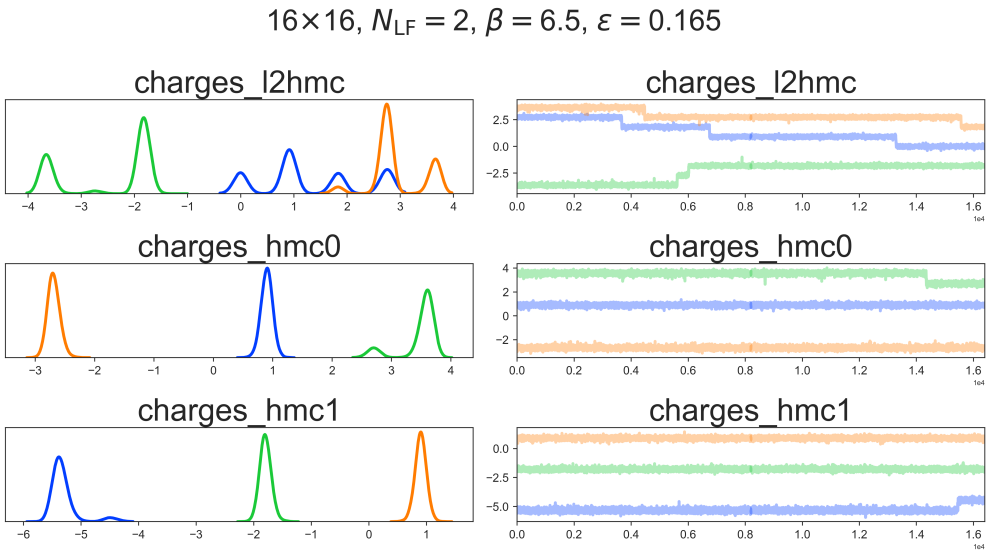
(b) Comparison of the plaquette and topological loss terms at $\beta = 5.5$.



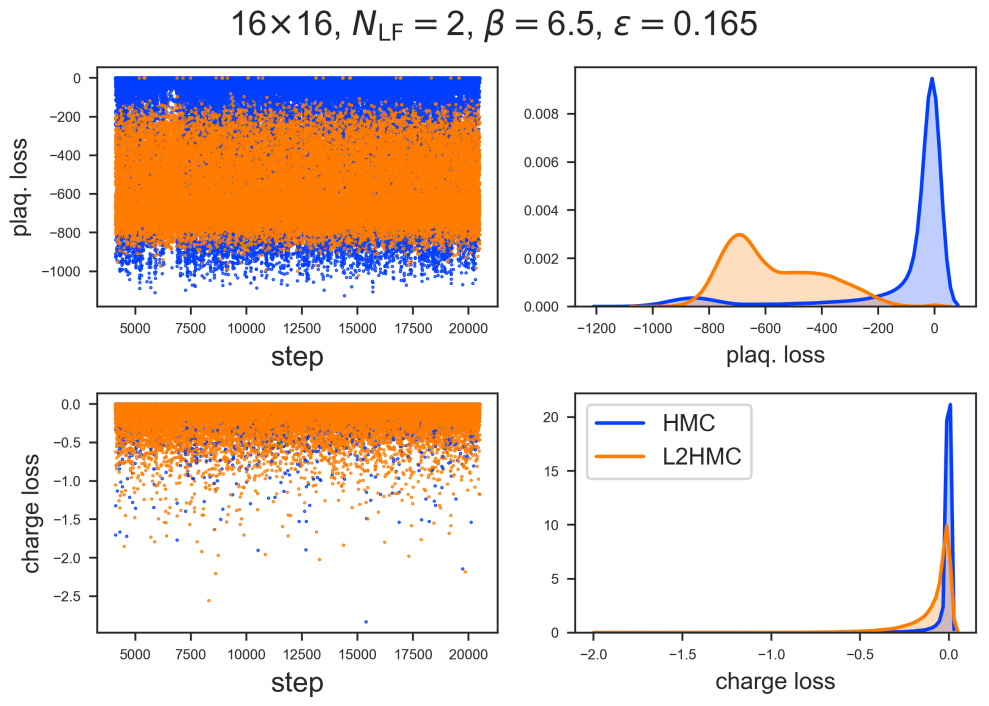
(a) Topological charges for both HMC and L2HMC runs at $\beta = 6$.



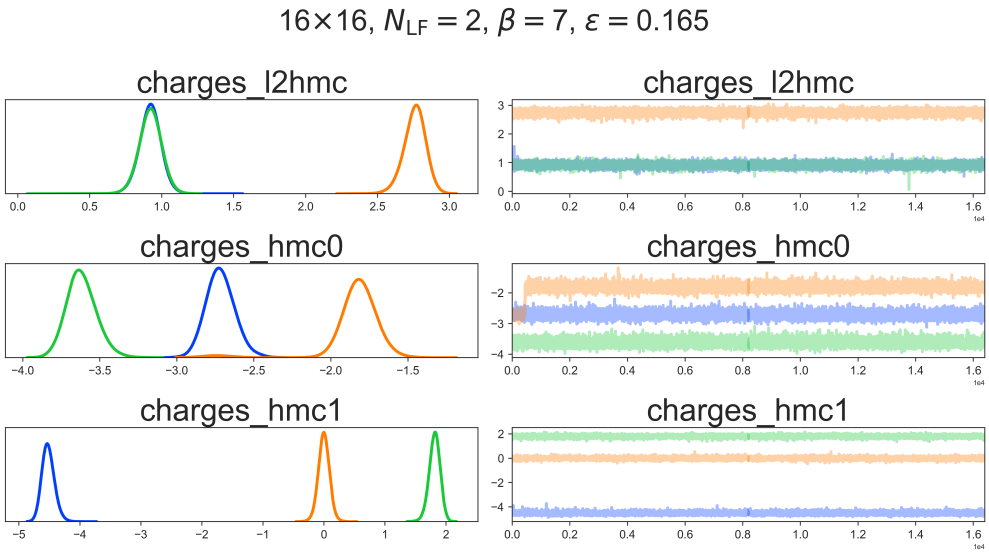
(b) Comparison of the plaquette and topological loss terms at $\beta = 6$.



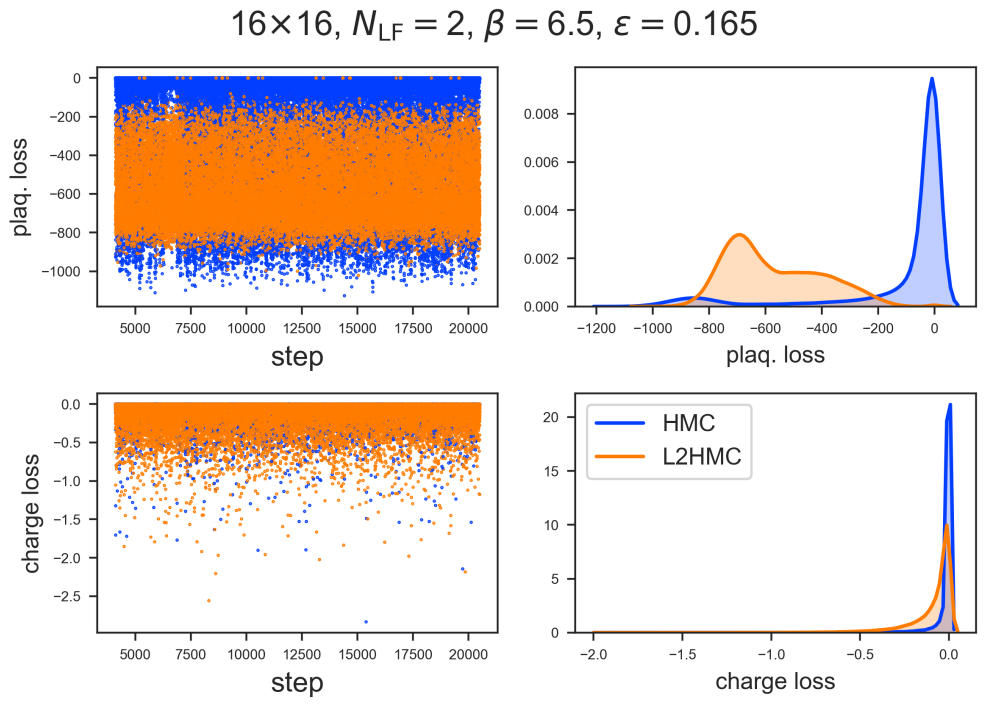
(a) Topological charges for both HMC and L2HMC runs at $\beta = 6.5$.



(b) Comparison of the plaquette and topological loss terms at $\beta = 6.5$.



(a) Topological charges for both HMC and L2HMC runs at $\beta = 7$.



(b) Comparison of the plaquette and topological loss terms at $\beta = 6.5$.

22 | Updates: 06/01/2020

22.1 Tunneling Rate

We define the *tunneling rate*, γ to be the average rate of change of the topological charge $\mathbb{Q} \in \mathbb{Z}$ between subsequent states in the Markov chain. For an individual chain of length M, we compute the tunneling rate as:

$$\gamma = \frac{1}{M} \sum_{i} \delta\mathbb{Q}^{(i)} = \frac{1}{M} \sum_{i=1}^{M} |\mathbb{Q}^{(i)} - \mathbb{Q}^{(i-1)}| \tag{82}$$

In Fig 34, we compare the tunneling rate for an L2HMC run to the tunneling rate for multiple HMC runs, for $\beta = 4, 4.5, 5$, each with a different step size ε.

Recent Progress:

- Generated database of generic HMC data for a range of different values of $N_{\text{LF}}, \beta, \varepsilon$ as a baseline to compare against.
- New even / odd masking scheme for splitting x updates instead of randomly assigned.
- Rewrote neural network code to be flexible and defined as a command line flag.
  - For example, if we want to use 3 hidden layers (with 128, 256, 512 units respectively), we can just pass `–units '128, 256, 512'` to the training script.
- Since the L2HMC algorithm aims to *generalize* HMC, we set all the weights in the neural network to be zero initially.
  - In addition, we begin by fixing all the network weights and allow the step-size ε to be a trainable parameter.
  - We then train/optimize ε until the loss plateaus, and then use this tuned value (along with the thermalized configuration) as the initial states for training the L2HMC algorithm.

TODO:

- ☐ Due to the annealing schedule, β changes by a small amount each training step. Because of this, our chains aren't able to completely thermalize.
  - It's not clear how much of an effect this has, but it might be beneficial to run a few thermalization steps (with all parameters fixed) in between each training step to help stabilize the configurations.
- ☐ Look at alternative network architectures, and/or custom (equivariant?) convolutional layers that can account for the geometry of the lattice.
- ☐ Try running longer training on larger models.
  - Because of the time limit on COOLEY, we are forced to periodically pause and resume the training every 12 hours (in addition to waiting in the queue).
- ☐ Continue adjusting parameters and training models. Immediate directions include: longer training (slower annealing schedule), more leapfrog steps, N_{LF}, deeper / wider neural networks,

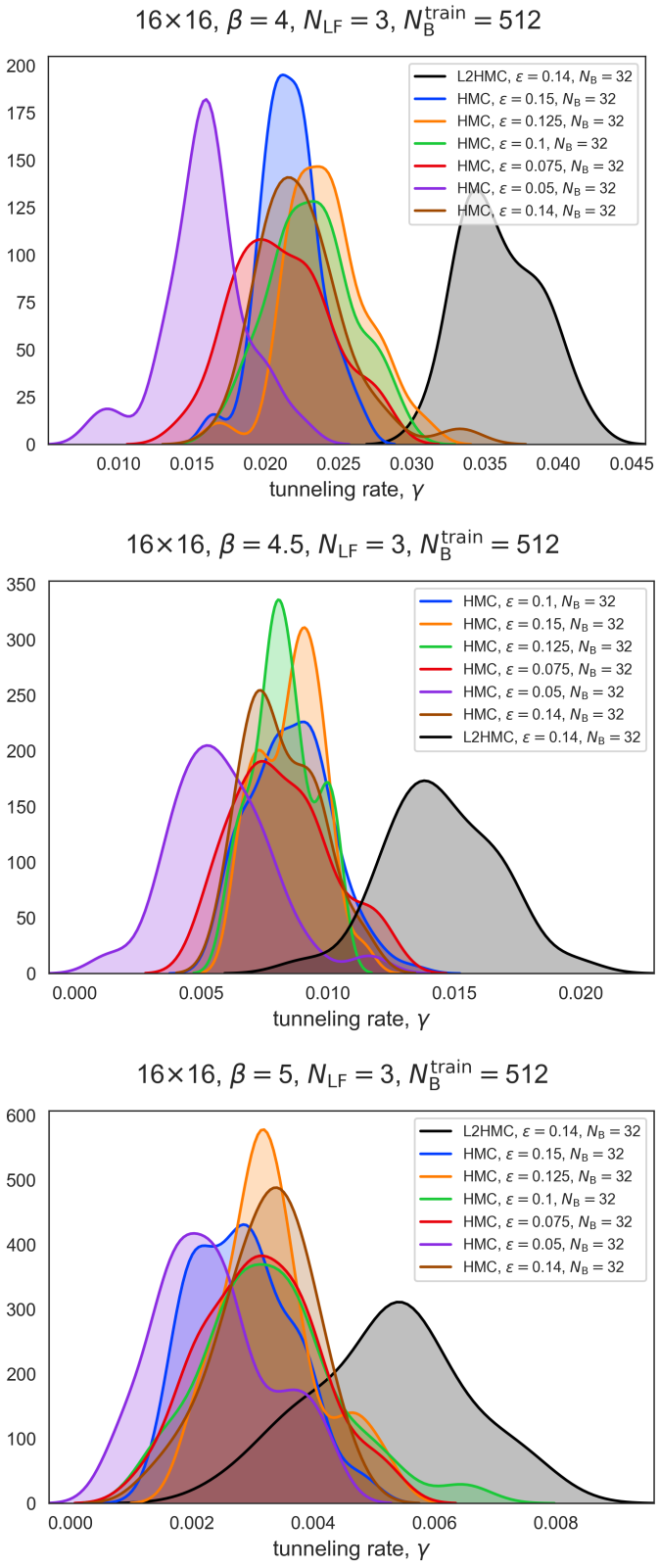
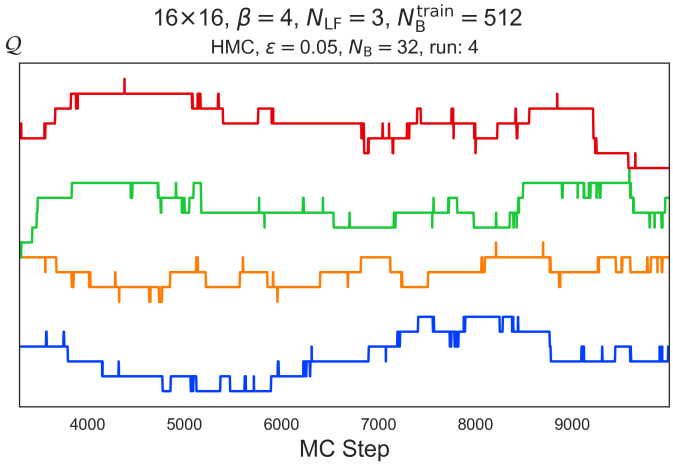
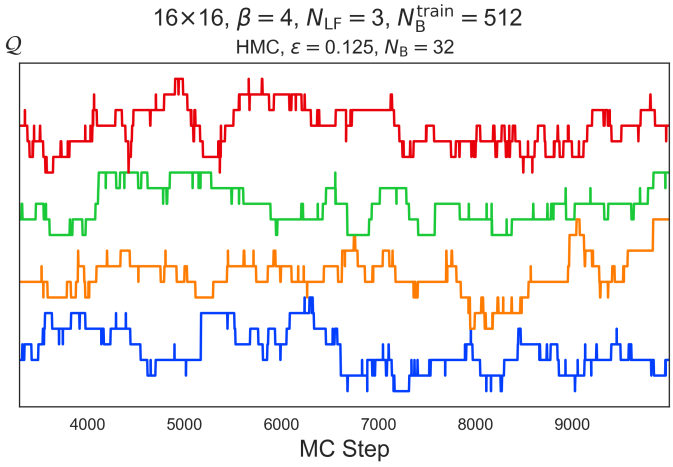


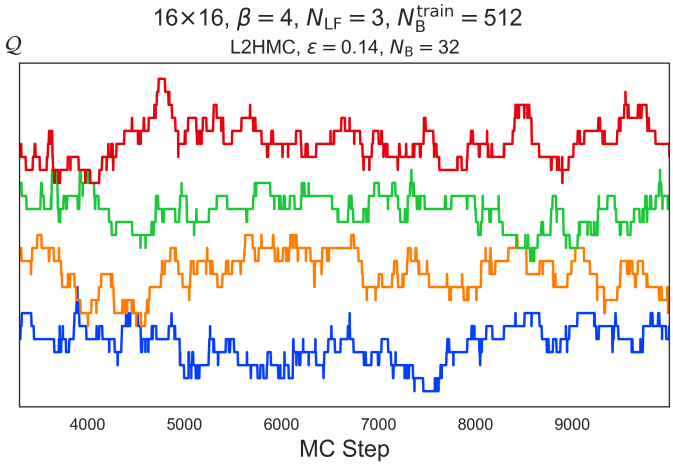
Figure 33: Figures showing the *tunneling rate*, γ for both L2HMC and HMC (with different step sizes, ε, for comparison.)



(a) HMC run with $\varepsilon = 0.05$



(b) HMC run with $\varepsilon = 0.125$



(c) L2HMC run with $\varepsilon = 0.14$

Figure 34: Figures showing the *tunneling rate*, γ for both L2HMC and HMC (with different step sizes, ε, for comparison.)

23 | Updates: 06/29/2020

Recent Progress:

- **Major rewrite of existing codebase**
  - Upgraded to be compatible with `tensorflow 2.x`
  - Makes it easier to resume a training session from a checkpoint
  - Can take advantage of tensorflows built-in profiler
  - Simplified (imperative!) interface for training / inference
  - Being *imerative-by-default* simplifies debugging and allows for rapid prototyping
  - Removes (a lot of!) redundant code
- **Modified network architecture to use separate networks for each leapfrog step**
  - Still working on direct comparison between this new architecture (with separate networks) and previous architecture (with single network).
- **Training without annealing schedule**
  - Avoids issue caused by β constantly changing (i.e. unable to fully "thermalize")
  - By training at a fixed β, we can directly compare against HMC with ε tuned to an optimal value (*testing* phase)
- Built custom `WarmupLR` learning rate schedule that slowly increases the learning rate from zero to prevent exploding gradients at the beginning of training

TODO:

- ☐ Include inference data from trained models in Figs. 36,37
- ☐ Implement `ReduceLROnPlateau` learning rate schedule to dynamically decrease the learning rate when the training loss encounters a plateau.
- ☐ Continued comparison between single/separate network architectures with all other parameters kept fixed.
- ☐ Look at alternative network architectures, and/or custom (equivariant?) convolutional layers that can account for the geometry of the lattice.
- ☐ Continue adjusting parameters and training models. Immediate directions include: longer training (slower annealing schedule), more leapfrog steps, N_{LF}, deeper / wider neural networks,
- ☐ Look at DeepHyper for hyperparameter/network architecture optimization

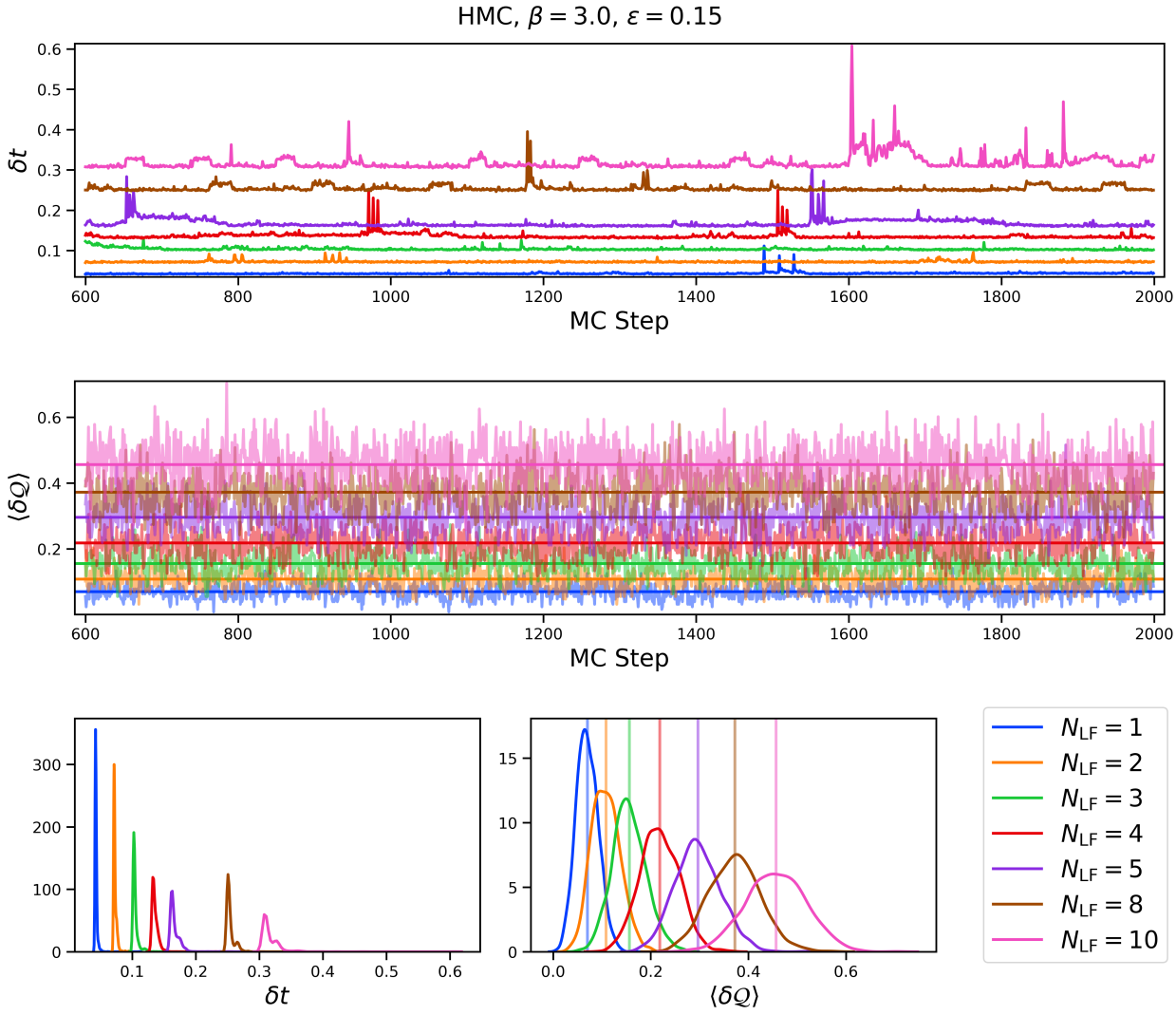


Figure 35: Cost (δt) benefit ($\delta \mathbb{Q}$) analysis of increasing the number of leapfrog steps N_{LF} in generic HMC at $\beta = 3.5$.

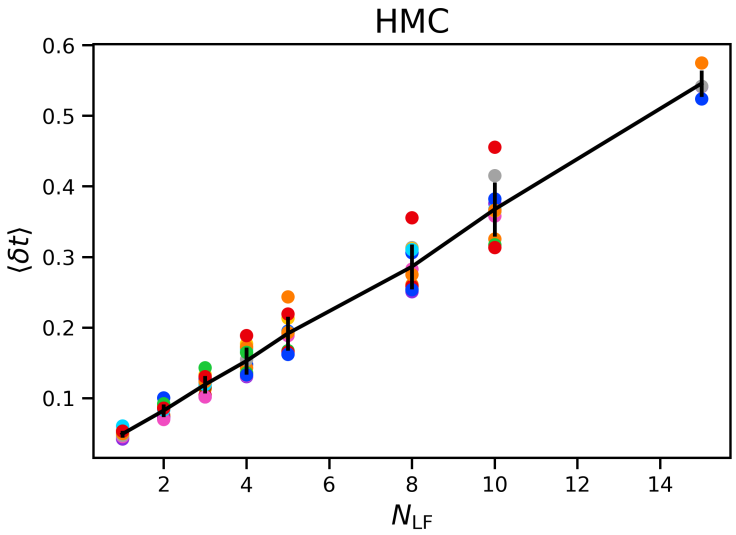
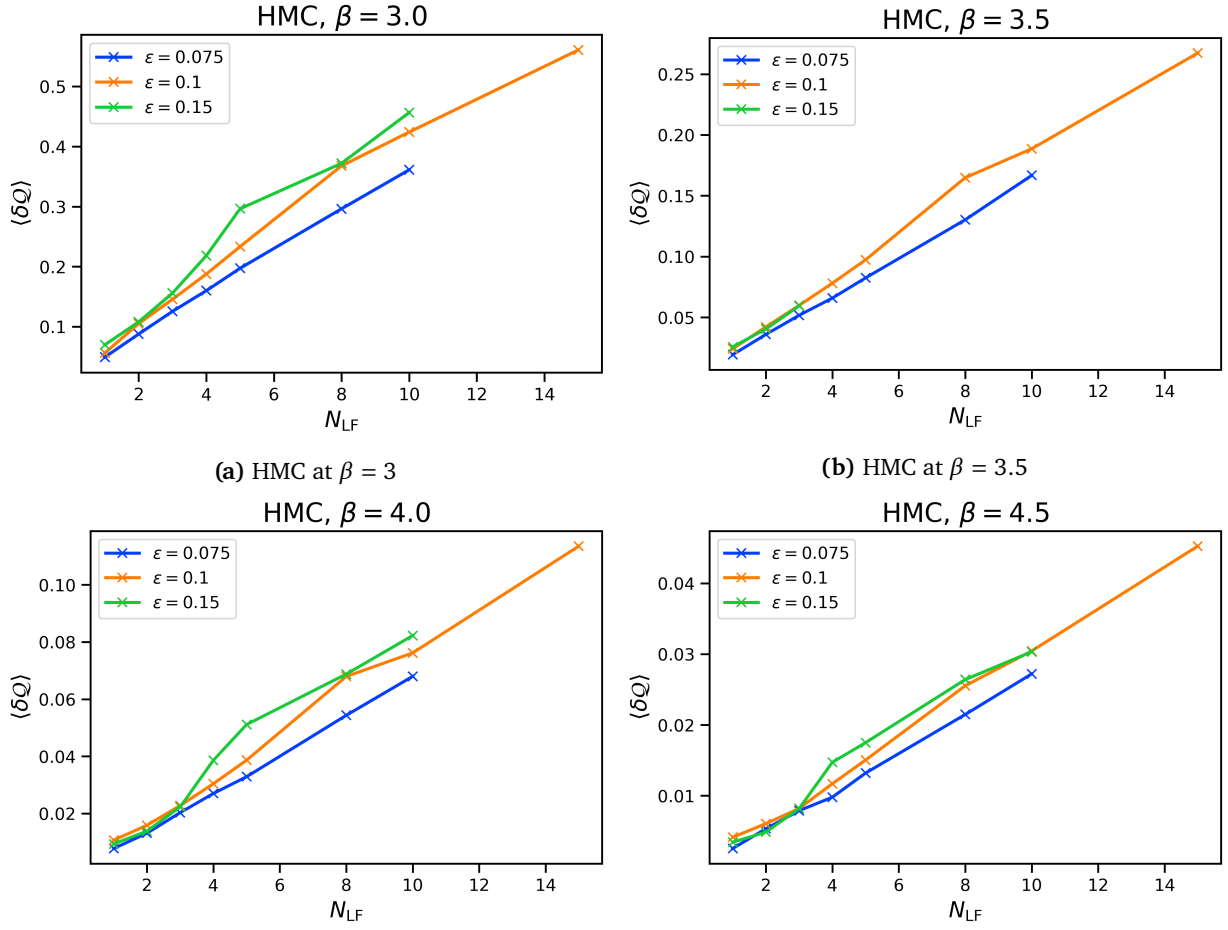


Figure 36: Average time per step vs. N_{LF} for generic HMC. Individual data points represent average values obtained using different values of β and ε.



(a) HMC at $\beta = 3$

(b) HMC at $\beta = 3.5$

(c) HMC at $\beta = 4$

(d) HMC at $\beta = 4.5$

Figure 37: Plots of $\langle \delta \mathbb{Q} \rangle$ vs N_{LF} for generic HMC.

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 38. The output from this convolutional structure is then fed to the generic network shown in Fig. 3. 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-

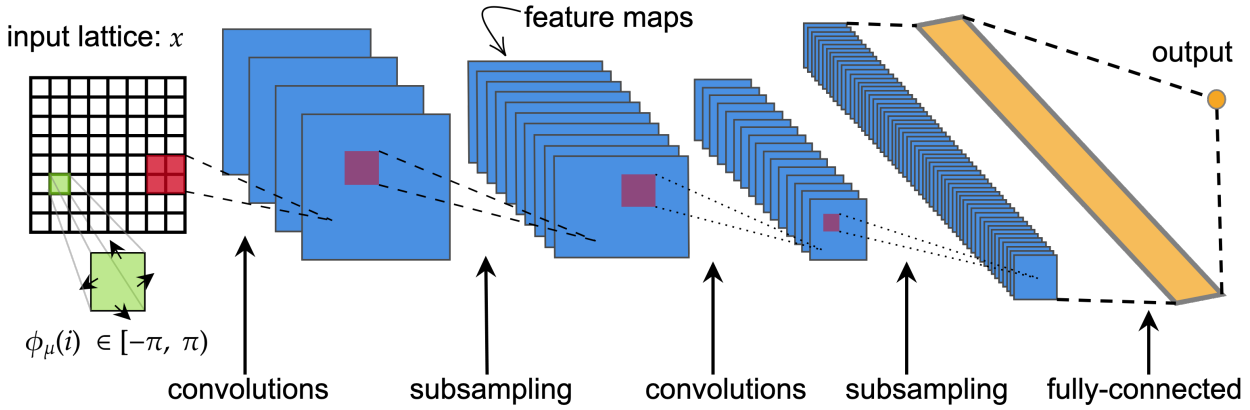


Figure 38: Convolutional structure used for learning localized features of rectangular lattice.

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[2], (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].

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 configurations 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.

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

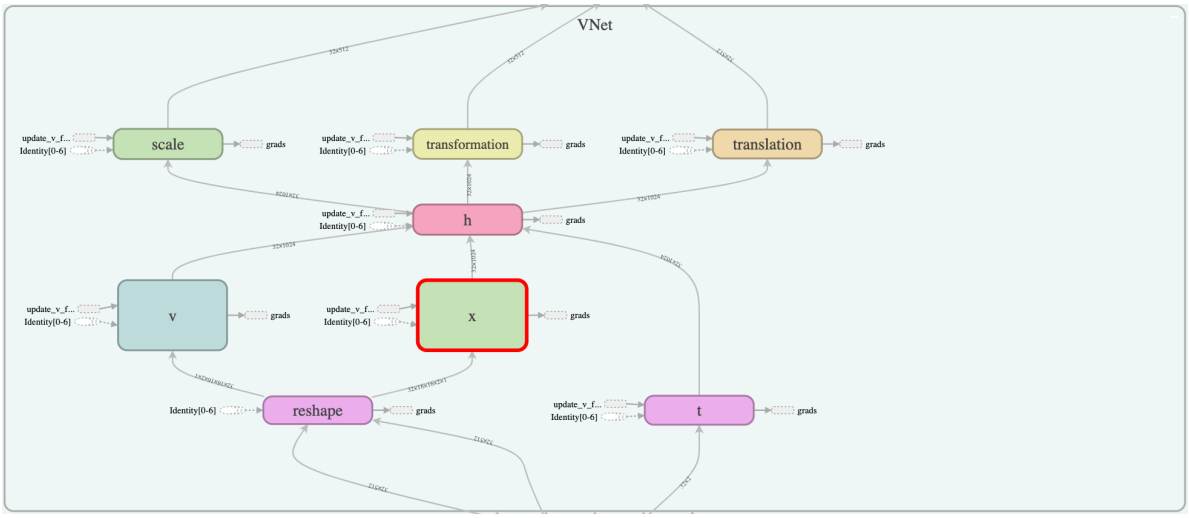


Figure 39: Illustration taken from TensorBoard showing an overview of the network architecture for VNet. Note that the architecture is identical for XNet.

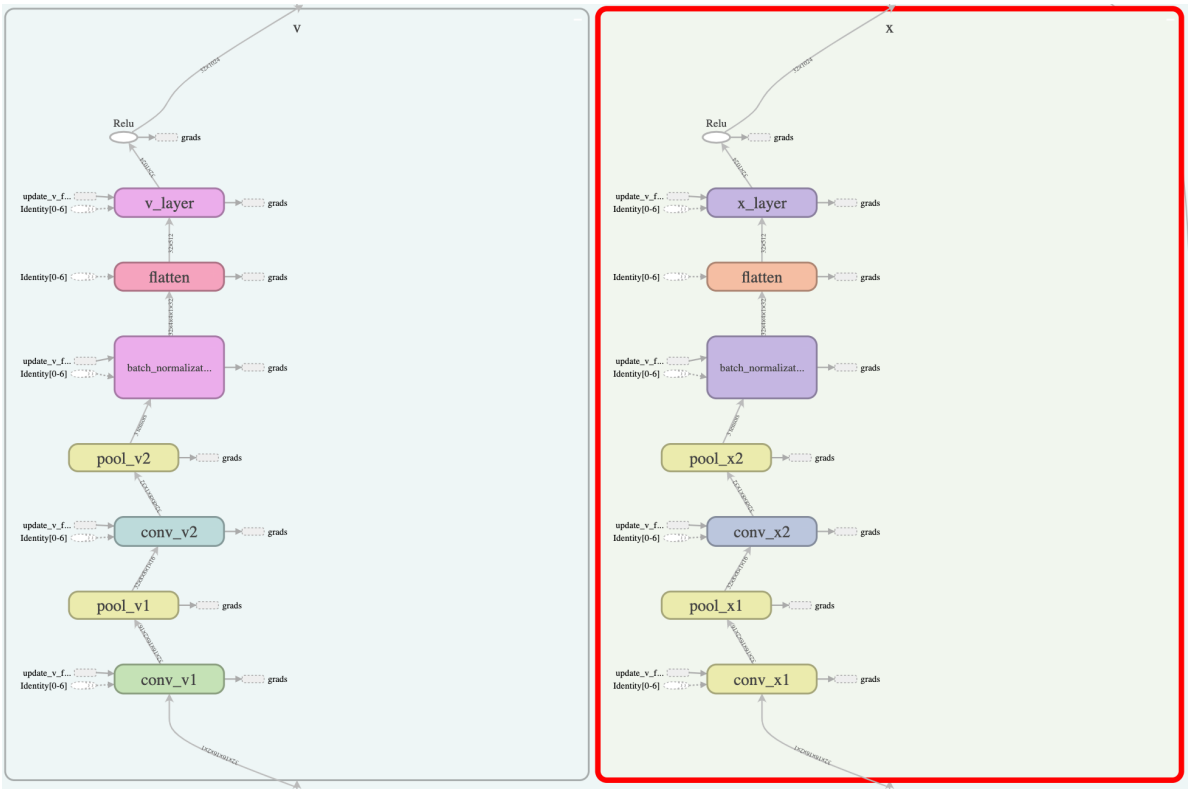


Figure 40: Detailed view of additional convolutional structure included to better account for rectangular geometry of lattice inputs.

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

$$\delta_Q(\xi, \xi') = |Q(\xi) - Q(\xi')| \tag{83}$$

$$\ell_Q(\xi, \xi', A(\mathbf{FL}_\theta \xi | \xi)) = \delta_Q(\xi, \xi') \times A(\xi' | \xi). \tag{84}$$

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

$$\mathscr{L}_Q(\theta) \equiv \mathbb{E}_{p(\xi)}\left[\ell_Q(\xi, \mathbf{FL}_\theta \xi, A(\mathbf{FL}_\theta \xi | \xi))\right] + \alpha_{\text{aux}}\, \mathbb{E}_{q(\xi)}\left[\ell_Q(\xi, \mathbf{FL}_\theta \xi, A(\mathbf{FL}_\theta \xi | \xi))\right] \tag{85}$$

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,

$$\mathscr{L}(\theta) = \alpha_{\text{std}}\, \mathscr{L}_{\text{std}}(\theta) + \alpha_Q\, \mathscr{L}_Q(\theta) \tag{86}$$

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. 85 weighs the contribution of configurations drawn from the initialization distribution.