GRADIENT ESTIMATORS FOR NORMALIZING FLOWS

Piotr Białas

Institute of Applied Computer Science, Jagiellonian University
Łojasiewicza 11, 30-348 Kraków Poland

Piotr Korcyl, Tomasz Stebel

Institute of Theoretical Physics, Jagiellonian University
Łojasiewicza 11, 30-348 Kraków Poland

Received 28 February 2024, accepted 9 March 2024, published online 22 March 2024

Recently, a machine learning approach to Monte-Carlo simulations called Neural Markov Chain Monte Carlo (NMCMC) is gaining traction. In its most popular form, it uses neural networks to construct normalizing flows which are then trained to approximate the desired target distribution. In this contribution, we present a new gradient estimator for the Stochastic Gradient Descent algorithm (and the corresponding PyTorch implementation) and show that it leads to better training results for the ϕ^4 model. For this model, our estimator achieves the same precision in approximately half of the time needed in the standard approach and ultimately provides better estimates of the free energy. We attribute this effect to the lower variance of the new estimator. In contrary to the standard learning algorithm, our approach does not require estimation of the action gradient with respect to the fields, thus has the potential of further speeding up the training for models with more complicated actions.

DOI:10.5506/APhysPolB.55.3-A2

1. Introduction

Despite the apparent simplicity of the original idea behind Monte-Carlo simulations by Stanislaw Ulam, this approach is one of the main pillars of computational sciences. Expressed in the form of an algorithm applied to study a simple classical statistical mechanics problem by Metropolis *et al.* [1], it is ubiquitous as a tool for dealing with complicated probability distributions (see, for example, [2]). In many cases, one resorts to the construction of an associated Markov chain of consecutive proposals which provides a mathematically grounded way of generating samples from a given distribution even when the proper normalization of the latter is not known. The

only limiting factor of the approach is the statistical uncertainty which directly depends on the number of statistically independent configurations. Hence, the effectiveness of any such simulation algorithm can be linked to its autocorrelation time which quantifies how many configurations are produced before a new, statistically independent configuration appears. For systems close to phase transitions, the increasing autocorrelation times, a phenomenon called critical slowing down, is usually the main factor that limits the statistical precision of outputs.

The recent interest in machine learning techniques has offered possible ways of dealing with this problem. For example, Ref. [3] and later Ref. [4] proposed autoregressive neural networks as a mechanism of generating independent configurations which can be used as proposals in the construction of the Markov chain. The new algorithm [5] was hence called Neural Markov Chain Monte Carlo (NMCMC). Once the neural network is sufficiently well-trained, one indeed finds that autocorrelation times are significantly reduced as was demonstrated in the context of the two-dimensional Ising model in Ref. [6].

For systems with continuous degrees of freedom, the NMCMC algorithm has to be appropriately modified and the predominant approach is to use *normalizing flows* to generate configurations, while at the same time calculating their probabilities. Both of these steps are necessary. Before any neural network can be used for that purpose, it must be trained, *i.e.* its weights should be tuned in such a way as to approximate the desired probability distribution. The standard approach for achieving this is using the stochastic gradient descent (SGD) algorithm which requires the estimation of gradients of the loss function with respect to the neural network weights. In this contribution, we propose to adapt the gradient estimator used for autoregressive networks applied for discrete models (*e.g.* Ising model) to the case of normalizing flows. We show that this estimator avoids calculating the derivative of the action and is only approximately 10% slower. Furthermore, due to its better convergence properties when applied in SGD, it outperforms the standard algorithm in terms of resulting autocorrelation time and the quality of calculations of the variational free energy. We attribute this effect to the lower variance of this new estimator. We demonstrate our idea using a solvable toy model and the scalar ϕ^4 field theory by comparing the proposed gradient estimator with the other two gradient estimators used in the literature.

This contribution is organized as follows. In order to be self-contained, we briefly introduce the NMCMC approach in Section 2. Then, we describe different gradient estimators in Section 3. We provide the definitions, as well as some characteristics. In Section 4, we discuss how the estimators introduced in Section 3 can be adapted to work with normalizing flows.

Section 5 provides a very simple toy example, where we can thoroughly compare all estimators. Finally, in Section 6, we compare all the estimators on the two-dimensional ϕ^4 model. We also include the snippets of Python code that implement our estimator using PyTorch framework [7].

2. Neural Markov Chain Monte Carlo

When using the Monte-Carlo methods, we are faced with the task of generating samples from some *target* distribution $p(\boldsymbol{\phi})$. In the majority of interesting applications, *e.g.* lattice field theories, it is impossible to generate samples independently from this distribution and we have to resort to Markov Chain Monte Carlo methods (MCMC) instead.

In this approach, given an initial configuration $\boldsymbol{\phi}_i$, a new trial configuration $\boldsymbol{\phi}_{\text{trial}}$ is proposed from the distribution $q(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i)$. This trial configuration is then accepted with probability $p_{\text{a}}(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i)$ or the previous configuration is repeated in the Markov chain. Usually, the configuration $\boldsymbol{\phi}_{\text{trial}}$ differs from $\boldsymbol{\phi}_i$ only on a small subset of degrees of freedom like *e.g.* single lattice site. If the so-called *detailed balance* condition

$$p(\boldsymbol{\phi}_i) q(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i) p_{\text{a}}(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i) = p(\boldsymbol{\phi}_{\text{trial}}) q(\boldsymbol{\phi}_i \,|\, \boldsymbol{\phi}_{\text{trial}}) p_{\text{a}}(\boldsymbol{\phi}_i \,|\, \boldsymbol{\phi}_{\text{trial}}) \tag{1}$$

is satisfied and provided that all available configurations can be reached, then asymptotically this procedure generates samples with distribution $p(\boldsymbol{\phi})$. One way of satisfying condition (1) is by the Metropolis–Hastings acceptance probability

$$p_{\text{a}}(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i) = \min \left\{ 1, \frac{p(\boldsymbol{\phi}_{\text{trial}})}{q(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i)} \frac{q(\boldsymbol{\phi}_i \,|\, \boldsymbol{\phi}_{\text{trial}})}{p(\boldsymbol{\phi}_i)} \right\} . \tag{2}$$

The biggest drawback of this algorithm is the fact that consecutive samples are highly correlated due to small incremental changes made at each step.

The idea of Metropolized Independent Sampling (MIS) method [8] is to generate samples *independently* from some distribution $q(\boldsymbol{\phi})$ *i.e.*

$$q(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i) = q(\boldsymbol{\phi}_{\text{trial}}) \,, \tag{3}$$

and then proceed with the Metropolis–Hastings accept/reject step

$$p_{\text{a}}(\boldsymbol{\phi}_{\text{trial}} \,|\, \boldsymbol{\phi}_i) = \min \left\{ 1, \frac{p(\boldsymbol{\phi}_{\text{trial}})}{q(\boldsymbol{\phi}_{\text{trial}})} \frac{q(\boldsymbol{\phi}_i)}{p(\boldsymbol{\phi}_i)} \right\} . \tag{4}$$

This also introduces correlations but if the distribution $q(\boldsymbol{\phi})$ is sufficiently close to $p(\boldsymbol{\phi})$ and the acceptance rate is close to one, then those correlations can be substantially smaller then in the case of MCMC (see Ref. [6] for discussion).

Seemingly, in the MIS approach, one has only replaced the problem of generating configurations from the distribution $p(\boldsymbol{\phi})$ with another hard problem of finding the distribution $q(\boldsymbol{\phi})$ that is close to the target distribution $p(\boldsymbol{\phi})$ and allows for fast generation of independent configurations. However, following the proposal of Neural Markov Chain Monte Carlo, one can use machine learning techniques, notably neural networks, to *learn* the distribution $q(\boldsymbol{\phi})$ [3, 4]. The general idea is that $q(\boldsymbol{\phi})$ is now parameterized by some (very large) set of parameters $\boldsymbol{\theta}$

$$q(\boldsymbol{\phi}) = q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \,. \tag{5}$$

The training consists in the tuning of the parameters $\boldsymbol{\theta}$ as to minimize a loss function that measures the difference between $q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})$ and target distribution $p(\boldsymbol{\phi})$. A natural choice for such a function is the Kullback–Leibler divergence

$$\begin{aligned} D_{\mathrm{KL}}(q \,|\, p) &= \int \mathrm{d}\boldsymbol{\phi}\, q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \left(\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log p(\boldsymbol{\phi})\right) \\ &= E[\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log p(\boldsymbol{\phi})]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})} \,. \end{aligned} \tag{6}$$

Please note that this function is not symmetric: $D_{\mathrm{KL}}(q \,|\, p) \neq D_{\mathrm{KL}}(p \,|\, q)$. This particular form (6) is chosen because we are sampling from the distribution $q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})$.

Actually, in most cases, we know the target distribution $p(\boldsymbol{\phi})$ only up to a normalizing constant. Let us assume that we only know $P(\boldsymbol{\phi})$,

$$P(\boldsymbol{\phi}) = Z \cdot p(\boldsymbol{\phi}) \,, \qquad Z = \int \mathrm{d}\boldsymbol{\phi} P(\boldsymbol{\phi}) \,, \tag{7}$$

where the constant Z is usually called the *partition function*. Inserting P instead of p into the Kullback–Leibler divergence definition, we obtain the *variational free energy*

$$F_q = E\left[\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log p(\boldsymbol{\phi}) - \log Z\right]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})} = F + D_{\mathrm{KL}}(q \,|\, p), \tag{8}$$

where $F = -\log Z$ is the *free energy*. As F does not depend on $\boldsymbol{\theta}$, minimizing F_q is equivalent to minimizing D_{KL}. In the following, we will use P and F_q instead of p and D_{KL}. The possibility of calculating F_q and thus estimating the free energy F is one of the major strengths of this approach as this is very hard to do in the classical MCMC simulations [9].

It is a non-trivial question as to how to define the $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ model in practice. It has to

(1) define a properly normalized probability distribution, and

(2) allow for sampling from this distribution.

We will shortly describe two common approaches: normalizing flows and autoregressive networks which can be used for systems with continuous and discrete degrees of freedom respectively.

2.1. Continuous degrees of freedom — normalizing flows

The normalizing flow can be thought of as a tuple of functions [5, 10, 11]

$$\mathbb{R}^D \ni \boldsymbol{z} \longrightarrow (q_{\mathrm{pr}}(\boldsymbol{z}), \boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})) \in \left(\mathbb{R}, \mathbb{R}^D\right) . \tag{9}$$

The function $q_{\mathrm{pr}}(\boldsymbol{z})$ is the probability density defining a *prior* distribution of random variable $\boldsymbol{z}$. The function $\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})$ must be a *bijection*, so if the input $\boldsymbol{z}$ is drawn from $q_{\mathrm{pr}}(\boldsymbol{z})$ then, the output $\boldsymbol{\phi}$ is distributed according to

$$q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = q_z(\boldsymbol{z}\,|\,\boldsymbol{\theta}) \equiv q_{\mathrm{pr}}(\boldsymbol{z}) J(\boldsymbol{z}\,|\,\boldsymbol{\theta})^{-1} , \qquad \boldsymbol{\phi} = \boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta}) , \tag{10}$$

where

$$J(\boldsymbol{z}\,|\,\boldsymbol{\theta}) = \det\left(\frac{\partial \boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{z}}\right) \tag{11}$$

is the determinant of the Jacobian of $\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})$. For this approach to be of practical use, the flows are constructed in such a way that the Jacobian determinant is relatively easy to compute. In terms of $q_{\mathrm{pr}}(\boldsymbol{z})$, $q_z(\boldsymbol{z}\,|\,\boldsymbol{\theta})$, and $\boldsymbol{\varphi}(\boldsymbol{z})$, the variational free energy F_q can be written as

$$\begin{aligned} F_q &= \int \mathrm{d}\boldsymbol{z}\, q_{\mathrm{pr}}(\boldsymbol{z}) \left(\log q_z(\boldsymbol{z}\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta}))\right) \\ &= E\left[\log q_z(\boldsymbol{z}\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta}))\right]_{q_{\mathrm{pr}}(\boldsymbol{z})} . \end{aligned} \tag{12}$$

When sampling from $q_{\mathrm{pr}}(\boldsymbol{z})$, this can be approximated as

$$F_q \approx \frac{1}{N} \sum_{i=1}^{N} \left(\log q_z(\boldsymbol{z}_i\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\varphi}(\boldsymbol{z}_i\,|\,\boldsymbol{\theta}))\right) , \quad \boldsymbol{z}_i \sim q_{\mathrm{pr}}(\boldsymbol{z}) , \tag{13}$$

where the $\sim$ symbol denotes that each $\boldsymbol{z}_i$ is drawn from the distribution $q_{\mathrm{pr}}(\boldsymbol{z})$.

2.2. Discrete degrees of freedom — autoregressive networks

For systems with discrete degrees of freedom, we cannot use normalizing flows. In such situation we can represent the distribution $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ via conditional probabilities

$$q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = q(\phi_1\,|\,\boldsymbol{\theta}) \prod_{i=2}^{\mathcal{N}} q(\phi_i\,|\,\phi_{i-1},\ldots,\phi_1,\boldsymbol{\theta})\,, \tag{14}$$

where $(\phi_1,\ldots,\phi_{\mathcal{N}})$ are the $\mathcal{N}$ components of the configuration $\boldsymbol{\phi}$ and have to represent *discrete* degrees of freedom. In the case of a simple spin system, $\phi_i = \pm 1$, the factorised probability (14) can be described by a neural network with $\mathcal{N}$ inputs $\phi_1,\ldots,\phi_{\mathcal{N}}$ and $\mathcal{N}$ outputs corresponding to conditional probabilities $q(\phi_i = 1\,|\,\phi_{i-1},\ldots,\phi_1)$. This can be generalized to the case when ϕ_i takes on more than two values [12]. To ensure that $q(\phi_i\,|\,\phi_{i-1},\ldots,\phi_1,\boldsymbol{\theta})$ depends only on the values of the preceding spins $\phi_{i-1},\ldots,\phi_1$, the so-called *autoregressive networks* are used [3, 13–16].

The configuration $\boldsymbol{\phi} = (\phi_1,\ldots,\phi_{\mathcal{N}})$ can be generated using ancestral sampling by successively generating the components ϕ_i one-by-one from distributions $q(\phi_i\,|\,\phi_{i-1},\ldots,\phi_1,\boldsymbol{\theta})$ starting at $i=1$ and feeding the generated components successively back to the network to obtain $q(\phi_{i+1}\,|\,\phi_i,\ldots,\phi_1,\boldsymbol{\theta})$, and so on.

In this formulation, we can obtain an estimate of F_q by sampling $\boldsymbol{\phi}$ directly from $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$

$$F_q \approx \frac{1}{N}\sum_{i=1}^{N} \left(\log q(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\phi}_i)\right)\,, \qquad \boldsymbol{\phi}_i \sim q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\,. \tag{15}$$

3. Gradient estimators

Minimizing F_q and thus training the machine learning model is done by the stochastic gradient descent (SGD) and requires the calculation of the gradient of F_q with respect to $\boldsymbol{\theta}$. Actually, we can only estimate the gradient based on the finite sample (batch) of N configurations $\{\boldsymbol{\phi}\} = \{\boldsymbol{\phi}_1,\ldots,\boldsymbol{\phi}_N\}$.

In the case of normalizing flows, this is pretty straightforward. We can directly differentiate expression (13) to obtain the first gradient estimator $\mathbf{g}_3[\{\boldsymbol{\phi}\}]$

$$\frac{\mathrm{d}F_q}{\mathrm{d}\boldsymbol{\theta}} \approx \mathbf{g}_3[\{\boldsymbol{\phi}\}] \equiv \frac{1}{N}\sum_{i=1}^{N} \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}} \left(\log q(\boldsymbol{z}_i\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\varphi}(\boldsymbol{z}_i\,|\,\boldsymbol{\theta}))\right)\,, \quad \boldsymbol{z}_i \sim q_{\mathrm{pr}}(\cdot\,|\,\boldsymbol{\theta})\,. \tag{16}$$

This derivative can be calculated by popular packages such as *e.g.* PyTorch [7] or TensorFlow [17] using automatic differentiation.

While conceptually simple, this estimator has a considerable drawback as it requires calculating the gradient of the distribution $P(\boldsymbol{\phi})$ with respect to the configuration $\boldsymbol{\phi}$,

$$\frac{\partial}{\partial \boldsymbol{\theta}} \log P(\boldsymbol{\varphi}(\boldsymbol{z}_i \,|\, \boldsymbol{\theta})) = \frac{\partial}{\partial \boldsymbol{\phi}} \log P(\boldsymbol{\phi}) \bigg|_{\boldsymbol{\phi}=\boldsymbol{\varphi}(\boldsymbol{z}_i|\boldsymbol{\theta})} \frac{\partial \boldsymbol{\varphi}(\boldsymbol{z}_i \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \,. \tag{17}$$

In lattice field theories the probability P is given by the *action* $S(\boldsymbol{\phi})$

$$\log P(\boldsymbol{\phi}(\boldsymbol{z} \,|\, \boldsymbol{\theta})) = -S(\boldsymbol{\phi}(\boldsymbol{z} \,|\, \boldsymbol{\theta})) \,, \tag{18}$$

and calculating the gradient of F_q requires the gradient of the action S with respect to the fields $\boldsymbol{\phi}$. This may not pose large complications for *e.g.* ϕ^4 theory where the action is just a polynomial in $\boldsymbol{\phi}$. Other lattice field theories however, notably the Quantum Chromodynamics with dynamical fermions, may have much more complicated actions including some representation of the non-local determinant of the fermionic matrix and the calculation of the action gradient may be impractical.

Autoregressive networks require calculating the derivative of expression (15). This is more tricky as in this case the sampling distribution $q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})$ also depends on $\boldsymbol{\theta}$. Following Ref. [3], we can, however, start by calculating the gradient of the exact expression (8)

$$\begin{aligned} \frac{\mathrm{d}F_q}{\mathrm{d}\boldsymbol{\theta}} \;&=\; \int \mathrm{d}\boldsymbol{\phi}\, \frac{\partial q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \left(\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log P(\boldsymbol{\phi})\right) \\ &\quad + \int \mathrm{d}\boldsymbol{\phi}\, q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \frac{\partial}{\partial \boldsymbol{\theta}} \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \,. \end{aligned} \tag{19}$$

The last term in the above expression is zero because it can be rewritten as the derivative of a constant

$$E\left[\frac{\partial \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})} = \int \mathrm{d}\boldsymbol{\phi}\, \frac{\partial q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \underbrace{\int \mathrm{d}\boldsymbol{\phi}\, q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}_{1} = 0 \,. \tag{20}$$

First term in expression (19) can be further rewritten as

$$\begin{aligned} \frac{\mathrm{d}F_q}{\mathrm{d}\boldsymbol{\theta}} \;&=\; \int \mathrm{d}\boldsymbol{\phi}\, q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \frac{\partial \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \left(\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log P(\boldsymbol{\phi})\right) \\ &=\; E\left[\frac{\partial \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \left(\log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log P(\boldsymbol{\phi})\right)\right]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})} , \end{aligned} \tag{21}$$

and approximated as

$$\frac{\mathrm{d}F_q}{\mathrm{d}\boldsymbol{\theta}} \approx \mathbf{g}_1[\{\boldsymbol{\phi}\}] \equiv \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \log q(\boldsymbol{\phi}_i \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \left(\log q(\boldsymbol{\phi}_i \,|\, \boldsymbol{\theta}) - \log P(\boldsymbol{\phi}_i)\right) \,, \tag{22}$$

which defines another gradient estimator $\mathbf{g}_1[\{\boldsymbol{\phi}\}]$ discussed in this work.

The authors of Ref. [3] introduce yet another gradient estimator, which we label $\mathbf{g}_2[\{\boldsymbol{\phi}\}]$, with the aim of reducing the variance, by subtracting the batch mean from the signal

$$\mathbf{g}_2[\{\boldsymbol{\phi}\}] = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \log q(\boldsymbol{\phi}_i \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \left(s(\boldsymbol{\phi}_i \,|\, \boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})_N} \right) , \tag{23}$$

where

$$s(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) \equiv \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) - \log P(\boldsymbol{\phi}) \quad \text{and} \quad \overline{s(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})_N} = \frac{1}{N}\sum_{i=1}^{N} s(\boldsymbol{\phi}_i \,|\, \boldsymbol{\theta}) \,. \tag{24}$$

Please note that expression (23) does not depend on Z.

Contrary to $\mathbf{g}_3$ and $\mathbf{g}_1$, the $\mathbf{g}_2$ estimator is slightly biased

$$E\left[\mathbf{g}_2[\{\boldsymbol{\phi}\}]\right] = \frac{N-1}{N} E\left[\mathbf{g}_1[\{\boldsymbol{\phi}\}]\right] \,. \tag{25}$$

The proof of this fact is presented in Appendix A. Of course, such multiplicative bias does not play any role when the estimator is used in the gradient descent algorithm and is very small anyway when $N \sim 10^3$. For all practical purposes, we can treat all estimators as unbiased, so any differences must stem from the higher moments, most importantly from the variance.

Although not much can be said about the variances of these estimators in general, we can show that for perfectly trained model *i.e.* when $q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) = p(\boldsymbol{\phi})$

$$\operatorname{var}\left[\mathbf{g}_1[\{\boldsymbol{\phi}\}]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})=p(\boldsymbol{\phi})}\right. = \frac{1}{N}(\log Z)^2 \operatorname{var}\left[\frac{\partial \log q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\right]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})=p(\boldsymbol{\phi})} . \tag{26}$$

As Z may be very large or very small depending on formulation of $P(\boldsymbol{\phi})$, the variance can be quite substantial. For $\mathbf{g}_2$, we obtain

$$\operatorname{var}\left[g_2[\{\boldsymbol{\phi}\}]_{q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta})=p(\boldsymbol{\phi})} = 0 \,. \right. \tag{27}$$

The proof is presented in Appendix B. This potentially very large reduction in variance was the actual rationale for introducing this estimator (see Ref. [3] supl. materials).

As for the estimator $\mathbf{g}_3$, we cannot make any claims about the value of its variance even for $q(\boldsymbol{\phi} \,|\, \boldsymbol{\theta}) = p(\boldsymbol{\phi})$ but we will show that it does not need to vanish in this case.

4. Eliminating action derivative

We notice that contrary to $\mathbf{g}_3$, the estimators $\mathbf{g}_1$ and $\mathbf{g}_2$ do not require calculating the derivatives of $P(\boldsymbol{\phi})$. This is due to the fact that we can first generate a configuration $\boldsymbol{\phi}$ from the distribution $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ and then obtain its probability directly (see figure 1 (a) for schematic picture). In the case of normalizing flows in the standard approach (estimator $\mathbf{g}_3$ described above), we do not have direct access to the function $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ since the probability of the configuration is determined simultaneously with the generation by passing $\boldsymbol{z}$ through the network (see figure 1 (b)). However, by leveraging the reversibility of normalizing flows, we can adapt the $\mathbf{g}_2$ estimator to that case (see figure 1 (c)). The $\mathbf{g}_2$ estimator requires the $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ function, and while it is not explicit in the normalizing flows formulation (9), it can be inferred from Eq. (10). Using the fact that the Jacobian determinant of the transformation $\boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$,

$$\bar{J}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) \equiv \det\left(\frac{\partial \boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\phi}}\right), \tag{28}$$

is the inverse of Jacobian determinant of $\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})$,

$$\bar{J}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = J(\boldsymbol{z}\,|\,\boldsymbol{\theta})^{-1}, \tag{29}$$

we can write $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ as

$$q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = q_{\mathrm{pr}}(\boldsymbol{z}')\bar{J}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}), \qquad \boldsymbol{z}' = \boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}). \tag{30}$$

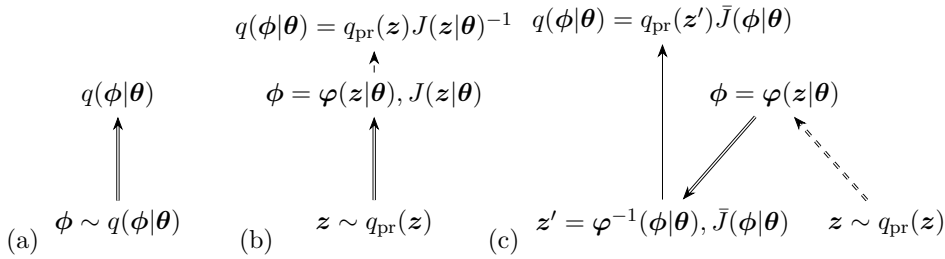

Fig. 1. Schematic picture of three algorithms for gradient estimation discussed in the paper: (a) autoregressive networks, (b) normalizing flows, (c) our proposition of adaptation of (a) into normalizing flows. Double-line arrows represent the flow: upward-pointing arrows represent forward propagation, and downward-pointing arrows represent backward propagation. Dashed arrows denote propagation which does not require the gradient.

Given that, the calculation of $\mathbf{g}_2$ would proceed as follows:

1. First, use the function $\boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})$ to generate configurations $\boldsymbol{\phi}_i$ without any gradient calculations
$$\boldsymbol{\phi}_i = \boldsymbol{\varphi}(\boldsymbol{z}_i\,|\,\boldsymbol{\theta})\,, \qquad \boldsymbol{z}_i \sim q_{\mathrm{pr}}(\boldsymbol{z})\,. \tag{31}$$
2. Then switch on the gradient calculations and calculate $\boldsymbol{z}'$ by running the flow backward
$$\boldsymbol{z}'_i = \boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta}) \tag{32}$$
and use Eq. (30) to calculate the probability $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$. It is very important that we use the $\boldsymbol{z}'_i$ from step two and not $\boldsymbol{z}_i$ from step one, as the gradients have to be propagated through q_{pr}.
3. Finally, the gradient estimate is calculated as in (23)
$$\mathbf{g}_2[\{\boldsymbol{\phi}\}] = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \log q(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \times \Big(\log q(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\phi}_i) - \overline{\log q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\phi})}\Big)\,. \tag{33}$$

This will require running the flow two times: forward to obtain $\boldsymbol{\phi}_i$, then backward to calculate $\boldsymbol{z}'$, but the gradients have to be calculated only on the last pass. We illustrate this with the pseudocode in Algorithm 1 and schematically in figure 1 (c).

▷ `generate` ϕ
Switch off gradient calculations
$\boldsymbol{z} \sim q_{\mathrm{pr}}(\boldsymbol{z})$
$\boldsymbol{\phi} \leftarrow \boldsymbol{\varphi}(\boldsymbol{z}\,|\,\boldsymbol{\theta})$ ▷ `Forward pass`
▷ `Calculate signal`
$s \leftarrow \log q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\phi})$
▷ `Calculate` g_2
Switch on gradient calculations
$\boldsymbol{z}' \leftarrow \boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$ ▷ `Backward pass`
$q \leftarrow q_{\mathrm{pr}}(\boldsymbol{z}'\,|\,\boldsymbol{\theta}) \det\left(\frac{\partial \boldsymbol{\varphi}^{-1}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\phi}}\right)$
$loss \leftarrow \log q \times (s - \bar{s})$

Algorithm 1: Calculation of $\mathbf{g}_2$ estimator for normalizing flows. The resulting *loss* can be used for automatic differentiation.

5. Toy model

We will illustrate the concepts introduced in previous sections with a very simple, one-dimensional normalizing flow that generates an exponential distribution

$$(q_{\mathrm{pr}}(z), \varphi(z\,|\,\theta)) = \left(1, -\frac{1}{\theta}\log(1-z)\right), \qquad z \in [0,1)\,. \tag{34}$$

This will allow us to explicitly calculate the form of each estimator $\mathbf{g}_i$, as well as its variance. Using (10), we obtain

$$q_z(z\,|\,\theta) = 1 \cdot J(\boldsymbol{z}\,|\,\boldsymbol{\theta}) = \theta(1-z)\,. \tag{35}$$

Combining this with the inverse flow

$$z = \varphi^{-1}(\phi\,|\,\theta) \equiv 1 - \mathrm{e}^{-\phi\theta}\,, \tag{36}$$

we get the exponential distribution

$$q(\phi\,|\,\theta) \equiv q_z\left(\varphi^{-1}(\phi\,|\,\theta)\,|\,\theta\right) = \theta\,\mathrm{e}^{-\theta\phi}\,. \tag{37}$$

The Jacobian determinant for the inverse flow is

$$\bar{J}(\phi\,|\,\theta) = \theta\,\mathrm{e}^{-\phi\theta}\,, \tag{38}$$

thus using (30), we get the same result for $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})$.

Given the target distribution

$$p(\phi) = \lambda\,\mathrm{e}^{-\lambda\phi} \quad \text{and} \quad P(\phi) = Z \cdot p(\phi)\,, \tag{39}$$

the free energy can be easily calculated as

$$\begin{aligned} F_q &= \theta \int \mathrm{d}\phi\,\mathrm{e}^{-\theta\phi}\left(\log\theta - \log\lambda - \log Z - \phi\,(\theta-\lambda)\right) \\ &= \log\theta - \log\lambda - \log Z - \frac{1}{\theta}\,(\theta-\lambda)\,, \end{aligned} \tag{40}$$

as well as its gradient

$$\frac{\mathrm{d}F_q}{\mathrm{d}\theta} = \frac{1}{\theta^2}(\theta-\lambda)\,. \tag{41}$$

For the gradient estimator $\mathbf{g}_1$, we obtain

$$\begin{aligned} \mathbf{g}_1[\{\phi\}] &= \frac{1}{N}\sum_{i=1}^{N}\left(\frac{1}{\theta} - \phi_i\right)\left(\log\theta - \log\lambda - \log Z - \phi_i(\theta-\lambda)\right) \\ &= \frac{1}{\theta}(\log\theta - \log\lambda - \log Z) + (\theta-\lambda)\frac{1}{N}\sum_{i=1}^{N}\phi_i^2 \\ &\quad - \left[\frac{1}{\theta}(\theta-\lambda) + (\log\theta - \log\lambda - \log Z)\right]\frac{1}{N}\sum_{i=1}^{N}\phi_i\,. \end{aligned} \tag{42}$$

Since

$$E\left[\frac{1}{N}\sum_i \phi_i\right] = \frac{1}{\theta} \quad \text{and} \quad E\left[\frac{1}{N}\sum_i \phi_i^2\right] = \frac{2}{\theta^2}\,, \tag{43}$$

we obtain the correct expression (41) for $E[\mathbf{g}_1]$ which means that the estimator is unbiased as expected. The calculation of the variance is more involved and the final result is

$$\begin{aligned}\mathrm{var}\,[\mathbf{g}_1] \;=\; & \frac{13}{N}\frac{(\theta-\lambda)^2}{\theta^4} - \frac{6}{N}\frac{(\theta-\lambda)(\log\theta - \log\lambda - \log Z)}{\theta^3} \\ & + \frac{1}{N}\frac{(\log\theta - \log\lambda - \log Z)^2}{\theta^2}\,.\end{aligned} \tag{44}$$

Thus, for $\theta = \lambda$,

$$\mathrm{var}\,[\mathbf{g}_1]_{\theta=\lambda} = \frac{1}{N}\frac{(\log Z)^2}{\lambda^2}\,, \tag{45}$$

which is non-zero in the case of $Z \neq 1$ and can be arbitrarily large.

For the estimator $\mathbf{g}_2$, we have

$$\mathbf{g}_2[\{\phi\}] = (\theta-\lambda)\frac{1}{N}\sum_i \left(\phi_i - \frac{1}{\theta}\right)\left(\phi_i - \bar{\phi}_N\right)\,, \qquad \bar{\phi}_N = \frac{1}{N}\sum_{j=1}^{N}\phi_j\,. \tag{46}$$

Using the relations

$$E\left[\bar{\phi}_N\right] = E\left[\phi\right] = \frac{1}{\theta} \tag{47}$$

and

$$\frac{1}{N}\sum_{i=1}^{N} E\left[\left(\phi - \frac{1}{\theta}\right)\bar{\phi}_N\right] = \frac{1}{N}E\left[\left(\phi - \frac{1}{\theta}\right)\phi\right] + \frac{N-1}{N}E\left[\left(\phi - \frac{1}{\theta}\right)\right]E\left[\phi\right]\,, \tag{48}$$

we obtain that the expectation value of estimator $\mathbf{g}_2$ is

$$E\left[\mathbf{g}_2\right] = \frac{N-1}{N}(\theta-\lambda)\frac{1}{\theta^2}\,, \tag{49}$$

as predicted by Eq. (25). The calculations of the variance are tedious and we present them to first order in N^{-1},

$$\mathrm{var}\,[\mathbf{g}_2] = \frac{7}{N}\frac{(\theta-\lambda)^2}{\theta^4} + (\theta-\lambda)^2 O\left(\frac{1}{N^2}\right)\,. \tag{50}$$

Finally for estimator $\mathbf{g}_3$, we obtain

$$\begin{aligned}\mathbf{g}_3[\{\phi\}] &= \frac{1}{N}\sum_i \frac{\mathrm{d}}{\mathrm{d}\theta}\left(\log\theta - \log\lambda + \log(1-z_i)\left(1-\frac{\lambda}{\theta}\right)\right) \\ &= \frac{1}{\theta} + \frac{\lambda}{\theta^2}\frac{1}{N}\sum_i \log(1-z_i)\,. \end{aligned} \tag{51}$$

Since the random variable $-\log(1-z)$ is distributed according to the exponential distribution with mean equal to one

$$E[\log(1-z_i)] = -1 \quad \text{and} \quad \mathrm{var}\,[\log(1-z_i)] = 1\,, \tag{52}$$

we again obtain the correct result (41) for $E\,[\mathbf{g}_3]$. Similarly, variance can be calculated as

$$\mathrm{var}\,[\mathbf{g}_3] = \frac{1}{N}\frac{\lambda^2}{\theta^4}\,. \tag{53}$$

Please note that this expression does not vanish when $\theta = \lambda$.

5.1. Numerical results

In order to see how these three different estimators behave when employed in the SGD algorithm, we have optimized the $q(\boldsymbol{\phi}\,|\,\theta)$*model* to match the distribution $p(\boldsymbol{\theta})$ using the PyTorch framework [7]. The target distribution parameter λ was set to $1/3$ and Z to λ^{-1}. The starting θ value was set to 1. We have performed 500 steps, where by one step we understand a single update of the parameter θ. At each step, we have sampled a batch of $N = 100$ elements from the distribution $q(\phi\,|\,\theta)$ which we have used to calculate the gradient estimate using one of the $\mathbf{g}_i$ estimators. The actual step *i.e.* adjustment of θ was performed using the Adam optimizer with a learning rate set to 0.01. The results are presented in the left panel of figure 2. As we can see, all estimators give similar performance and θ converges to the true value. However, after approximate convergence, we note that the use of the estimators $\mathbf{g}_1$ and $\mathbf{g}_3$ results in a rather large "wandering" of the θ around its target value, which is due to the non-vanishing variance of the gradient in this case (see the right panel of figure 2).

To estimate the variance, we have generated 1000 additional batches at each step. On each batch, we calculated the gradient estimator and used those 1000 samples to estimate the variance. The results are presented in the right panel of figure 2, where we show the standard deviation (square root of variance) of gradient estimators calculated during simulations. They are consistent with our analytical calculations and, as predicted, the variance of estimator $\mathbf{g}_2$ does vanish as $\theta \to \lambda$. In contrast, the variance of the remaining estimators is substantially bigger than zero. While this is a contrived

example, it serves as an indicator that while unbiased, different estimators can have dramatically different statistical properties. Of course, increasing the batch size would result in decreased variance for all estimators.

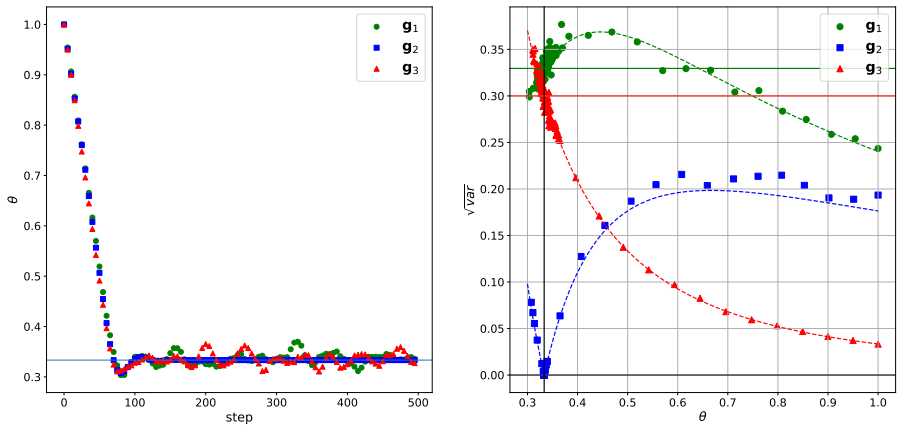

Fig. 2. Optimizing the $q(\cdot\,|\,\theta)$ distribution. Left: Evolution of the parameter θ, the blue horizontal line indicates the true value of $\theta = 1/3$. Right: Standard deviation (square root of variance) of the gradient estimator *versus* θ, dotted lines represent analytic results. Vertical black line corresponds to $\theta = 1/3$ and horizontal lines correspond to values of standard deviation at $\theta = \lambda$ for the $\mathbf{g}_1$ and $\mathbf{g}_3$ estimators.

6. Lattice ϕ^4 theory

The second example is the two dimensional scalar ϕ^4 field theory with Euclidean action

$$S\left[\boldsymbol{\phi}\,|\,m^2,\lambda\right] = \int \mathrm{d}x^2 \left(\sum_{\mu=0,1} \left(\partial_\mu \phi(x)\right)^2 + m^2\phi^2(x) + \lambda\phi^4(x) \right) \tag{54}$$

which, following [18], we discretize as

$$\begin{aligned} S\left(\boldsymbol{\phi}\,|\,m^2,\lambda\right) \;=\; & \sum_{i,j=0}^{L-1} \phi_{i,j}\left(2\phi_{i,j} - \phi_{i-1,j} - \phi_{i+1,j} + 2\phi_{i,j} - \phi_{i,j-1} - \phi_{i,j+1}\right) \\ & + \sum_{i,j=0}^{L-1} \left(m^2\phi_{i,j} + \lambda\phi_{i,j}^4\right) , \end{aligned} \tag{55}$$

where the lattice has the size of $L \times L$. The probability distribution p is given by the Boltzmann distribution

$$p(\boldsymbol{\phi}) = Z\left(m^2,\lambda\right)^{-1} \mathrm{e}^{-S\left(\boldsymbol{\phi}|m^2,\lambda\right)}\,, \qquad Z\left(m^2,\lambda\right) = \int \mathrm{d}\boldsymbol{\phi}\, \mathrm{e}^{-S\left(\boldsymbol{\phi}|m^2,\lambda\right)} \tag{56}$$

thus, $P(\boldsymbol{\phi}) = \exp(-S(\boldsymbol{\phi}))$.

We have used PyTorch normalizing flows implementation provided in the excellent tutorial [18]. It uses the *affine coupling layers* to implement the flow [10]. Field $\boldsymbol{\phi}$ is split using a checkerboard pattern into two parts $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$. Part $\boldsymbol{\phi}_2$ is frozen and does not change during the transformation but is used as an input to the functions $\boldsymbol{t}$ and $\boldsymbol{s}$ which are then used to transform part $\boldsymbol{\phi}_1$

$$\begin{aligned} \boldsymbol{\phi}_1' &\leftarrow \boldsymbol{\phi}_1\, \mathrm{e}^{\boldsymbol{s}(\boldsymbol{\phi}_2)} + \boldsymbol{t}(\boldsymbol{\phi}_2)\,, \\ \boldsymbol{\phi}_2' &\leftarrow \boldsymbol{\phi}_2\,. \end{aligned} \tag{57}$$

The outputs of functions $\boldsymbol{t}$ and $\boldsymbol{s}$ have the same size as $\boldsymbol{\phi}_1$ and all arithmetic operations are performed pointwise. The Jacobian determinant of this transformation is easily calculable

$$\log J(\boldsymbol{z}\,|\,\boldsymbol{\theta}) = \sum_i s_i(\boldsymbol{\phi}_2)\,, \tag{58}$$

where the sum runs over all components of $\boldsymbol{s}$.

Please note that this is a bijection with the inverse transformation given by

$$\begin{aligned} \boldsymbol{\phi}_1 &\leftarrow \left(\phi_1' - \boldsymbol{t}\left(\phi_2'\right)\right) \mathrm{e}^{-\boldsymbol{s}\left(\boldsymbol{\phi}_2'\right)}\,, \\ \boldsymbol{\phi}_2 &\leftarrow \boldsymbol{\phi}_2'\,, \end{aligned} \tag{59}$$

and

$$\log \bar{J}(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = -\sum_i s_i\left(\boldsymbol{\phi}_2'\right)\,. \tag{60}$$

In the next layer parts $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$ are interchanged. The functions $\boldsymbol{t}$ and $\boldsymbol{s}$ in each layer are implemented using a convolutional neural network with two output channels. The architecture of this network is presented in Table 1. We use 16 coupling layers. The prior distribution q_{pr} is taken as the standard normal distribution $\mathcal{N}(0,1)$ independently on each component of $\boldsymbol{z}$.

Table 1. Convolutional neural network architecture used in coupling layers.

Layer	ch_{in}	ch_{out}	Kernel
1	1	16	(3,3)
	Leaky ReLU		
2	16	16	(3,3)
	Leaky ReLU		
3	16	16	(3,3)
	Leaky ReLU		
4	16	2	(3,3)
	tanh		

The Python code for the single update step is presented in listing 1. The function `train_step` is parameterized by the `loss_fn` function which implements the loss used to calculate estimators $\mathbf{g}_i$. The code for loss estimators is presented in listings 2 and 3. The `reverse_apply_flow` and `apply_flow` functions are presented in listing 4.

```
def train_step(sub_mean, batch_size,
               *, model, action, loss_fn, optimizer):
    optimizer.zero_grad()
    loss, logq, logp = loss_fn(sub_mean, model=model,
                               action=action)
    loss.backward()
    optimizer.step()
```

Listing 1. A single update step.

```
def g_1_2_loss(sub_mean, batch_size, *, model, action):
    layers, prior = model["layers"], model["prior"]

    with torch.no_grad():
        z = prior.sample_n(batch_size)
        log_p_z = prior.log_prob(z)
        phi, logq = nf.apply_flow(layers, z, log_p_z)
        logp = -action(phi)
        signal =  logq - logp

    z, log_q_phi = nf.reverse_apply_flow(layers,
                   phi, torch.zeros(batch_size,
                   device=phi.device))
    log_q_phi+= prior.log_prob(z)
```

```
    if sub_mean:
        loss = torch.mean(log_q_phi
                          * (signal - signal.mean()))
    else:
        loss = torch.mean(log_q_phi * signal)

    return loss, logq, logp
```

Listing 2. Loss for the estimators $\mathbf{g}_1$ and $\mathbf{g}_2$. The only difference is the subtraction of the mean from the signal in the case of estimator $\mathbf{g}_2$. `layers` implements the affine coupling layers normalizing flow, `prior` implements the $q_{\mathrm{pr}}(\boldsymbol{z})$ distribution

```
def g_3_loss(sub_mean, batch_size, *, model, action):
    layers, prior = model["layers"], model["prior"]
    x, logq = nf.apply_flow_to_prior(
                        prior, layers,
                        batch_size=batch_size)
    logp = -action(x)
    loss = torch.mean(logq-logp)
    return loss, logq, logp
```

Listing 3. Loss for the estimator $\mathbf{g}_3$. The `sub_mean` parameter is provided for compatibility with the $\mathbf{g}_2$ and $\mathbf{g}_3$ loss implementation.

```
def apply_flow(coupling_layers, z, logq):
    for layer in coupling_layers:
        z, logJ = layer.forward(z)
        logq = logq - logJ
    return z, logq

def apply_flow_to_prior(prior, coupling_layers,
                        *, batch_size):
    z = prior.sample_n(batch_size)
    logq = prior.log_prob(z)
    return apply_flow(coupling_layers, z, logq)

def reverse_apply_flow(coupling_layers, phi, logq):
    for layer in reversed(coupling_layers):
        phi, logJ = layer.reverse(phi)
        logq += logJ
    return phi, logq
```

Listing 4. Applying the flow in forward and in reverse directions.

6.1. Numerical results

In this section, we present results for the following values of action (54) parameters: $m^2 = -4$, $\lambda = 8$. For those values, the system is in the disordered phase where the absolute magnetization $\langle\,|\,\sum_i \phi_i\,|\,\rangle$ is small and no mode-seeking phenomenon appears [19]. Results obtained for other values of physical parameters m^2 and λ are presented in Appendix C.

For each estimator, we have made four different training runs of 4000 epochs each, where each epoch consisted of 100 simulation steps, and in each step, we have sampled a batch of 1024 $\boldsymbol{\phi}$ configurations. The training parameters as well as timings are presented in Table 2. As expected, the $\mathbf{g}_2$ estimator is slower as it has to make one more pass through the network. However, it is only $\sim 10\%$ slower, indicating that it is the gradient calculation that takes up most of the time.

Table 2. Parameters and timings of the runs. We have omitted the $\mathbf{g}_1$ estimator due to its poor performance. lr denotes the learning rate used.

$L = 16$	$m^2 = -4$	$\lambda = 8$
optimizer	Adam	lr = 0.001
Num. epochs	Steps per epoch	Batch size
4000	100	1024
GPU	V100	32 GB
estimator	time	t/step
$\mathbf{g}_2$	19:00:00	0.17 s
$\mathbf{g}_3$	17:10:00	0.15 s

In Figs. 3 and 4, we present the evolution of two metrics: effective sample size (ESS) and variational free energy F_q (Eq. (8)). The ESS is defined as

$$\mathrm{ESS} = \frac{E\left[w(\boldsymbol{\phi})\right]^2_{q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})}}{E\left[w(\boldsymbol{\phi})^2\right]_{q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})}} \approx \frac{\left(\sum_{i=1}^N w(\boldsymbol{\phi}_i)\right)^2}{N\sum_{i=1}^N w(\boldsymbol{\phi}_i)^2}\,, \tag{61}$$

where

$$w(\boldsymbol{\phi}) = \frac{p(\boldsymbol{\phi})}{q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})} \quad \text{and} \quad \boldsymbol{\phi}_i \sim q(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta})\,, \tag{62}$$

and it is an estimate of the fraction of samples that can be considered independent [8]. Obviously, $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = p(\boldsymbol{\phi})$ entails $w(\boldsymbol{\phi}_i) = 1$ and $\mathrm{ESS} = 1$. The values presented are the averages over one epoch. The first thing one can notice is that the estimator $\mathbf{g}_1$ is converging very slowly and we have

decided to stop training after 1000 epochs and do not consider this estimator in further studies. The estimators $\mathbf{g}_2$ and $\mathbf{g}_3$ achieve comparable results with the estimator $\mathbf{g}_2$ systematically converging faster.

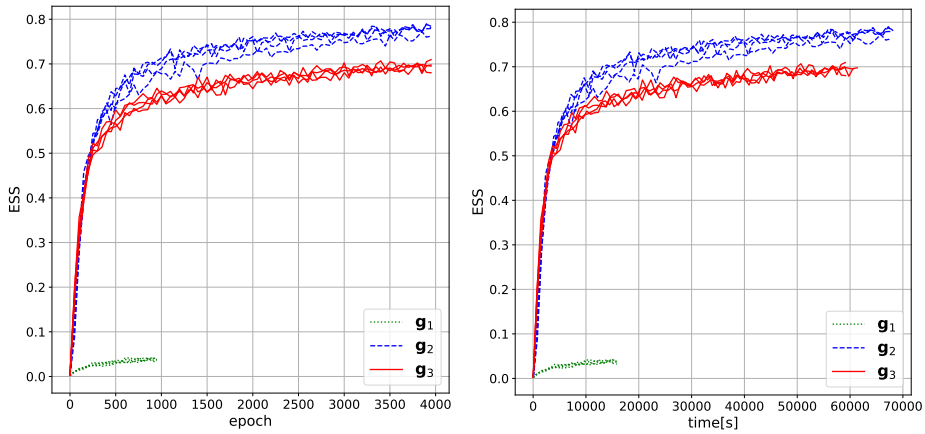

Fig. 3. Left: ESS as the function of epoch. One epoch consisted of 100 steps. In each step, the `train_step` function was called once. Right: ESS as a function of wall time in seconds.

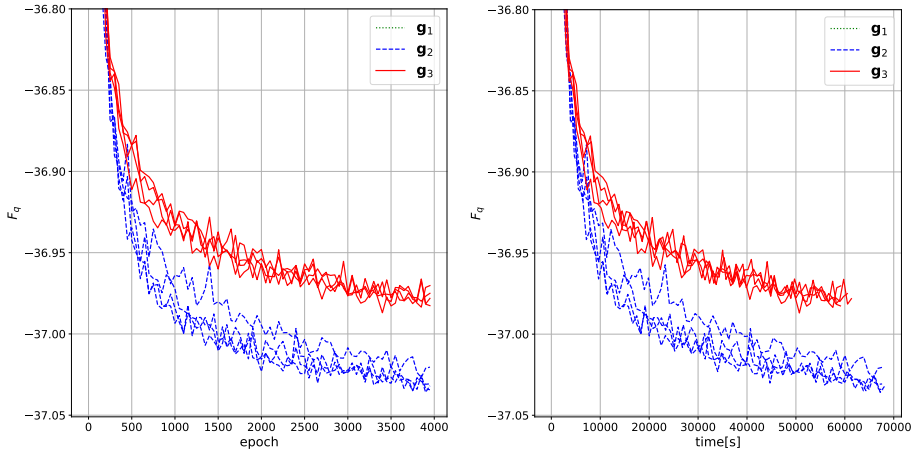

Fig. 4. Left: F_q as the function of epoch. One epoch consisted of 100 steps. In each step, the `train_step` function was called once. Right: F_q as a function of wall time in seconds. The estimator $\mathbf{g}_1$ is not shown as it results in F_q greater than -36.0.

To better compare the two estimators for each run, we have looked at the last 10 epochs (1000 steps) to find the lowest achieved value of F_q. We have saved the corresponding model. Each model was then used to generate a sample of 10^5 $\boldsymbol{\phi}$ configurations. We used those samples to calculate the variational free energy F_q (Eq. (8)). We have calculated the standard deviation of the signal $s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N}$ over each sample. This can be used as an indicator of the quality of training as it is zero when $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = p(\boldsymbol{\phi})$. While it is not clear how to quantify this, we can assume that a lower standard deviation indicates better-trained flow [3].

Next, we used the Metropolis–Hastings rejection step (4) to obtain the Monte-Carlo samples from the distribution (56). We have calculated the acceptance and the integrated autocorrelation time τ (see [20, pages 137, 143–145]). The results for each run are presented in Table 3. For the estimator $\mathbf{g}_2$, we have used models obtained after training for 2000 epochs or 4000 epochs. Looking at the table, we see that $\mathbf{g}_2$ systematically outperforms $\mathbf{g}_3$ for every metric even for much shorter training times.

Table 3. var denotes the variance of the signal $s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N}$ and acc. the acceptance, τ is the integrated autocorrelation time.

Estimator(epochs) time [hh:mm]											
$\mathbf{g}_2$(2000) 9:30				$\mathbf{g}_2$(4000) 19:00				$\mathbf{g}_3$(4000) 17:10			
F_q	$\sqrt{\text{var}}$	acc.	τ	F_q	$\sqrt{\text{var}}$	acc.	τ	F_q	$\sqrt{\text{var}}$	acc.	τ
−36.98	0.76	0.69	1.28	−37.03	0.70	0.73	1.09	−36.96	0.77	0.66	1.43
−37.00	0.74	0.70	1.21	−37.02	0.71	0.72	1.09	−36.97	0.76	0.67	1.41
−37.01	0.72	0.71	1.18	−37.04	0.68	0.74	1.03	−37.00	0.74	0.69	1.38
−36.98	0.76	0.67	1.29	−37.04	0.68	0.75	1.00	−36.99	0.74	0.69	1.27

All three estimators are (practically) unbiased, so the differences in performance must stem from the difference in higher moments. To verify this, we estimated the variance of each estimator. Given a model, we have generated $N_\text{b} = 1000$ batches $\{\boldsymbol{\phi}\}_i$ of 1024 samples each (that was the batch size used in training). For each batch, we calculated the gradient estimate and the variance of every term which we then averaged

$$\text{var}\,[\mathbf{g}] \approx \frac{1}{N_\theta}\sum_{j=1}^{N_\theta}\frac{1}{N_\text{b}}\sum_{i=1}^{N_\text{b}}\left(g_j[\{\boldsymbol{\phi}\}_i] - \overline{g_j}\right)^2 , \tag{63}$$

where $\mathbf{g}$ is any of three gradients estimators and $g_j[\{\boldsymbol{\phi}\}]$ is its j^{th} component calculated for batch $\{\boldsymbol{\phi}\}$, $\overline{g_j}$ is the average of this component over all batches.

The results are presented in figure 5. We plot the square root of variance (63) as the function of the training time of the model. As we can see, the values for estimator $\mathbf{g}_1$ are almost two orders of magnitude larger than for the other two. That explains why it is performing so poorly. The picture on the right shows the same data but on different vertical scales so we can see the difference between the $\mathbf{g}_2$ and $\mathbf{g}_3$ estimators. The estimator $\mathbf{g}_2$ has clearly a lower variance which explains its better performance for this model.

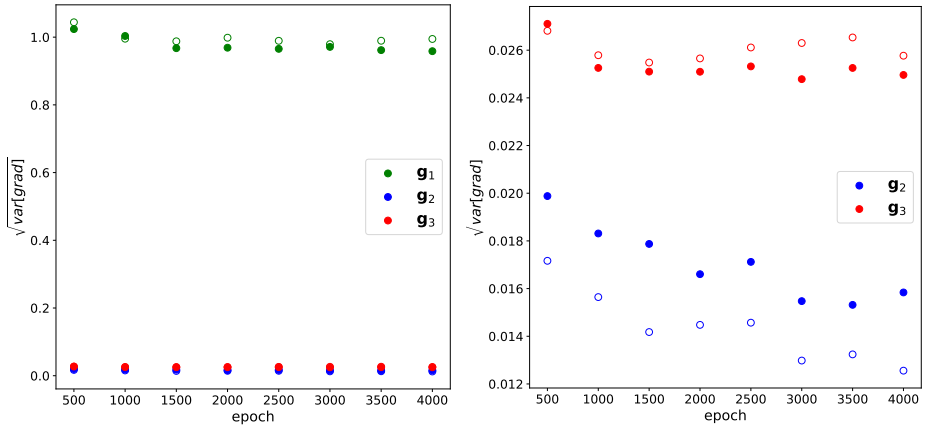

Fig. 5. Square root of the variance of gradient estimators as the function of training time. The right panel shows the same data but on different vertical scales. Open and closed symbols show independent runs.

7. Summary

In this work, we have described two estimators of the gradient of the loss function used in the literature in two different contexts: one is usually discussed together with systems with discrete degrees of freedom (we denoted it $\mathbf{g}_2$ in the text), while the other together with systems with continuous degrees of freedom (called $\mathbf{g}_3$). The machine learning architectures used for these two classes of systems are also different: one uses autoregressive neural networks in the first case, while normalizing flows in the second case. We pointed out that the two gradient estimators differ conceptually, namely the estimator typically used in the context of normalizing flows requires an explicit computation of derivatives of the action with respect to the fields, while the other does not. We, therefore, described how to adapt the gradient estimator $\mathbf{g}_2$ to the case of normalizing flows, rendering the computation of action derivatives unnecessary. This has the potential to speed up the training for models with more complicated actions. We supplemented our discussion with numerical experiments: in a one-dimensional toy model,

where all relevant quantities have been calculated analytically as well as in the two-dimensional scalar ϕ^4 field theory model. The proposed estimator $\mathbf{g}_2$ takes only 10% more time to calculate, this is more than offset by its better convergence properties. We have shown that given the same training time, it can outperform the standard estimator by a large margin and provide similar results in half of the time. Our results suggest that this is due to the lower variance of the $\mathbf{g}_2$ estimator compared to $\mathbf{g}_3$.

It should be noted that the training of the flow can be regarded as estimating the free energy by a variational approach. Such a task is notoriously hard, error-prone, and time-consuming in classical MCMC, this fact can warrant the use of normalizing flows for F_q calculation even when using them in NMCMC may be unpractical [9]. We have shown that using our approach we have obtained lower, and thus better, values of F_q.

Note added: After completion of this contribution our group compared performance of the $\mathbf{g}_2$ and $\mathbf{g}_3$ estimators using the 2D Schwinger model with the Wilson fermions [21]. Contrary to the ϕ^4 lattice theory, evaluation of action in the Schwinger model is computationally intensive as it contains the determinant of the Dirac operator. For such a model, the $\mathbf{g}_2$ estimator, which does not require differentiating of the action, has a significant advantage over $\mathbf{g}_3$ as far as numerical cost and memory usage are concerned.

An analysis related to the one presented in this manuscript was performed by the authors of Ref. [22], where several gradient estimators were considered including $\mathbf{g}_2$ and $\mathbf{g}_3$. Reference [22] proposed another estimator, called path gradient. Detailed comparison of $\mathbf{g}_2$ and the path gradient deserves a separate study, in particular in the context of lattice field theories with fermions.

Computer time allocation grant `plgnnformontecarlo` on the Prometheus supercomputer hosted by AGH Cyfronet in Kraków, Poland was used through the Polish PLGRID consortium. T.S. kindly acknowledges support of the National Science Center (NCN), Poland grant No. 2021/43/D/ST2/03375 and support of the Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University grant No. 2021-N17/MNS/000062. This research was partially funded by the Priority Research Area Digiworld under the program Excellence Initiative — Research University at the Jagiellonian University in Kraków.

Appendix A

Bias of estimator $\mathbf{g}_2$

First, recall that (Eq. (24))

$$s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) \equiv \log q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \log P(\boldsymbol{\phi}) \quad \text{and} \quad \overline{s(\boldsymbol{\phi})_N} = \frac{1}{N}\sum_{i=1}^{N} s(\boldsymbol{\phi}_i)\,. \tag{A.1}$$

From (23), we have

$$E\left[\mathbf{g}_2[\{\boldsymbol{\phi}\}]\right] = E\left[\mathbf{g}_1[\{\boldsymbol{\phi}\}]\right] - \frac{1}{N}\sum_{i=1}^{N} E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N}\right]\,, \tag{A.2}$$

where we have introduced a shortened notation

$$\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = \frac{\partial \log q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\,. \tag{A.3}$$

The second term in expression (A.2) is equal to

$$\begin{aligned} E\left[\frac{1}{N^2}\sum_{i,j}\delta(\boldsymbol{\phi}_i\,|\,\boldsymbol{\theta})s(\boldsymbol{\phi}_j\,|\,\boldsymbol{\theta})\right] &= \frac{1}{N}E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\right] \\ &\quad + \frac{N-1}{N}E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\right]E\left[s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\right] \\ &= \frac{1}{N}E\left[\mathbf{g}_1[\{\boldsymbol{\phi}\}]\right]\,, \end{aligned} \tag{A.4}$$

where we have used formula (20) which entails $E\left[\delta(\boldsymbol{\phi})\right] = 0$. Putting this back into (A.2), we obtain the stated result (25).

Appendix B

Variance of the estimators

Since $\boldsymbol{\phi}$s are independent from (22), we obtain

$$\mathrm{var}\left[\mathbf{g}_1\right] = \frac{1}{N}\left(E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})^2 s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})^2\right] - E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\right]^2\right)\,, \tag{B.1}$$

when $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = p(\boldsymbol{\phi})$, then $s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = -\log Z$ and we obtain expression (26).

For the estimator $\mathbf{g}_2$ by the same reasoning, we obtain

$$\begin{aligned} \mathrm{var}\left[\mathbf{g}_2\right] &= \frac{1}{N}\left(E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})^2\left(s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N}\right)^2\right]\right. \\ &\quad \left. - E\left[\delta(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})\left(s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N}\right)\right]^2\right)\,, \end{aligned} \tag{B.2}$$

when $q(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) = p(\boldsymbol{\phi})$, then $s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta}) - \overline{s(\boldsymbol{\phi}\,|\,\boldsymbol{\theta})_N} = 0$ and variance vanishes.

Appendix C

Results for other values of ϕ^4 theory parameters

In the main part of the manuscript, we considered one set of parameters of scalar field action, $m^2 = -4$, $\lambda = 8$. In this appendix, we shall investigate other choices of parameters, we shall fix $m^2 = -4$ and vary λ.

In figure 6, we compare ESS as a function of training epochs for two gradient estimators $\mathbf{g}_2$ and $\mathbf{g}_3$. We choose three values of λ: 3 (top pair of curves), 4.5 (bottom pair of curves), and 8 (middle pair of curves). These values were chosen so that the system is in the ordered phase, near the phase transition[1] and the disordered state, respectively. We first note that far from phase transition, the network can be efficiently trained no matter which estimator is used. We systematically observe better performance of $\mathbf{g}_2$ there. In the critical region, $\lambda = 4.5$, the network is much harder to train and both estimators perform comparably. For $\lambda = 3$, where two modes of the distribution are well separated, the mode-seeking phenomenon is observed, namely networks break the symmetry and sample only one mode, see separated study of this phenomenon [19].

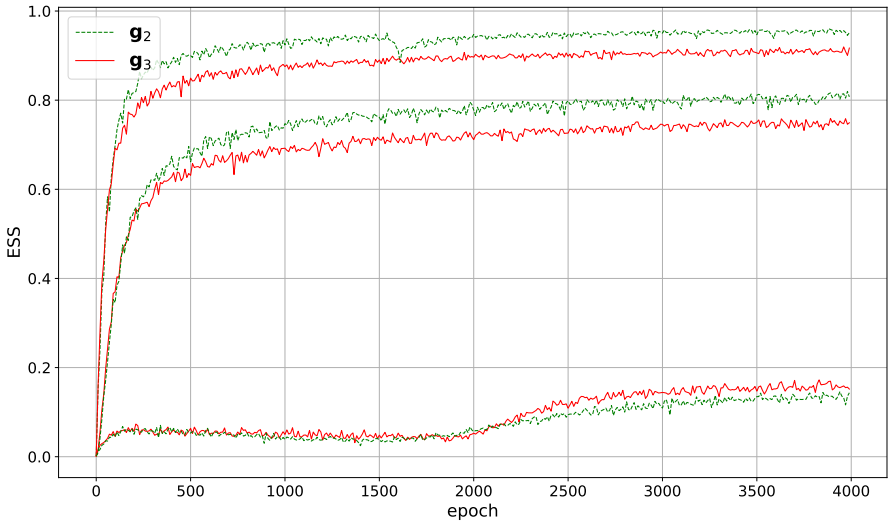

Fig. 6. ESS as the function of epoch (one epoch consisted of 100 steps) during training. The top pair of curves was obtained for $\lambda = 3$, the middle pair for $\lambda = 8$, and the bottom pair for $\lambda = 4.5$.

[1] Since we consider here small sizes of lattice, $L = 16$, we are not exposed to strong critical behavior of the system.

The above results were obtained by calculating ESS at the batch size = 1024 and then averaging such ESS over 100 batches to eliminate large fluctuations of this observable. We have noticed that if the network is not well-trained (here, close to the phase transition), the resulting values of ESS strongly depend on the batch size. We attribute this behavior to a large variance of the weights of (62). This phenomenon deserves separate studies which are currently in progress and will be presented elsewhere.

REFERENCES

[1] N. Metropolis *et al.*, «Equation of State Calculations by Fast Computing Machines», *J. Chem. Phys.* **21**, 1087 (1953).

[2] K. Binder, D. Heermann, «Monte Carlo Simulation in Statistical Physics: An Introduction», *Springer International Publishing*, Cham 2019.

[3] D. Wu, L. Wang, P. Zhang, «Solving Statistical Mechanics Using Variational Autoregressive Networks», *Phys. Rev. Lett.* **122**, 080602 (2019).

[4] K.A. Nicoli *et al.*, «Asymptotically unbiased estimation of physical observables with neural samplers», *Phys. Rev. E* **101**, 023304 (2020).

[5] M.S. Albergo, G. Kanwar, P.E. Shanahan, «Flow-based generative models for Markov chain Monte Carlo in lattice field theory», *Phys. Rev. D* **100**, 034515 (2019).

[6] P. Białas, P. Korcyl, T. Stebel, «Analysis of autocorrelation times in Neural Markov Chain Monte Carlo simulations», *Phys. Rev. E* **107**, 015303 (2023), arXiv:2111.10189 [cond-mat.stat-mech].

[7] A. Paszke *et al.*, «PyTorch: An Imperative Style, High-Performance Deep Learning Library», in: «Advances in Neural Information Processing Systems 32», *Curran Associates, Inc.*, 2019, pp. 8024–8035.

[8] J.S. Liu, «Metropolized independent sampling with comparisons to rejection sampling and importance sampling», *Stat. Comput.* **6**, 113 (1996).

[9] K.A. Nicoli *et al.*, «Estimation of Thermodynamic Observables in Lattice Field Theories with Deep Generative Models», *Phys. Rev. Lett.* **126**, 032001 (2021).

[10] L. Dinh, J. Sohl-Dickstein, S. Bengio, «Density estimation using Real NVP», arXiv:1605.08803 [cs.LG].

[11] I. Kobyzev, S. Prince, M. Brubaker, «Normalizing Flows: An Introduction and Review of Current Methods», *IEEE Trans. Pattern Anal. Mach. Intell.* **43**, 3964 (2021).

[12] P. Białas, P. Czarnota, P. Korcyl, T. Stebel, «Simulating first-order phase transition with hierarchical autoregressive networks», *Phys. Rev. E* **107**, 054127 (2023).

[13] B.J. Frey, «Graphical Models for Machine Learning and Digital Communication», *MIT Press*, Cambridge, MA 1998.

[14] B. Uria *et al.*, «Neural Autoregressive Distribution Estimation», *J. Mach. Learn. Res.* **17**, 1 (2016).

[15] M. Germain, K. Gregor, I. Murray, H. Larochelle, «MADE: Masked Autoencoder for Distribution Estimation», in: F. Bach, D. Blei (Eds.) «Proceedings of the 32nd International Conference on Machine Learning», Vol. 37, *PMLR*, Lille, France, 07–09 July, 2015, pp. 881–889.

[16] A.V. Oord, N. Kalchbrenner, K. Kavukcuoglu, «Pixel Recurrent Neural Networks», in: M.F. Balcan, K.Q. Weinberger (Eds.) «Proceedings of the 33rd International Conference on Machine Learning», Vol. 38, *PMLR*, New York, USA, 20–22 June, 2016, pp. 1747–1756.

[17] M. Abadi *et al.*, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems», 2015. Software available from `https://www.tensorflow.org/`

[18] M.S. Albergo *et al.*, «Introduction to Normalizing Flows for Lattice Field Theory», `arXiv:2101.08176 [hep-lat]`.

[19] D.C. Hackett *et al.*, «Flow-based sampling for multimodal distributions in lattice field theory», `arXiv:2107.00734 [hep-lat]`.

[20] A. Sokal, «Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms», in: C. DeWitt-Morette, P. Cartier, A. Folacci (Eds.) «Proceedings of Functional Integration: Basics and Applications», *Springer US*, Boston, MA 1997, pp. 131–192.

[21] P. Białas, P. Korcyl, T. Stebel, «Training normalizing flows with computationally intensive target probability distributions», *Comput. Phys. Commun.* **298**, 109094 (2024).

[22] L. Vaitl, K.A. Nicoli, S. Nakajima, P. Kessel, «Gradients should stay on path: better estimators of the reverse- and forward KL divergence for normalizing flows», *Mach. Learn.: Sci. Technol.* **3**, 045006 (2022).