TECHNISCHE UNIVERSITÄT WIEN

DISSERTATION

Symmetry-preserving neural networks in lattice field theories

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der Technischen Wissenschaften unter der Leitung von

Priv.Doz. Dr.techn. Andreas Ipp

und mit der Unterstützung von

Dr.techn. David I. Müller

eingereicht an der Technischen Universität Wien,
Fakultät für Physik

von
Dott. Mag. Matteo Favoni

Wien, am 16.12.2024

_____

_____ _____ _____
Dr. Andreas Ipp   Prof. Tilo Wettig   Prof. Kai Zhou
(Betreuer)   (Gutachter)   (Gutachter)

Abstract

This thesis deals with neural networks that respect symmetries and presents the advantages in applying them to lattice field theory problems. The concept of equivariance is explained, together with the reason why such a property is crucial for the network to preserve the desired symmetry. The benefits of choosing equivariant networks are first illustrated for translational symmetry on a complex scalar field toy model. The discussion is then extended to gauge theories, for which Lattice Gauge Equivariant Convolutional Neural Networks (L-CNNs) are specifically designed ad hoc. Regressions of physical observables such as Wilson loops are successfully solved by L-CNNs, whereas traditional architectures which are not gauge symmetric perform significantly worse. Finally, we introduce the technique of neural gradient flow, which is an ordinary differential equation solved by neural networks, and propose it as a method to generate lattice gauge configurations.

ii

Zusammenfassung

Diese Arbeit befasst sich mit neuronalen Netzen, die Symmetrien respektieren, und stellt die Vorteile ihrer Anwendung auf gitterfeldtheoretische Probleme dar. Das Konzept der Äquivarianz wird erklärt, zusammen mit der Begründung, warum eine solche Eigenschaft für das Netz entscheidend ist, um die gewünschte Symmetrie zu erhalten. Die Vorteile der Wahl äquivarianter Netze werden zunächst für die Translationssymmetrie eines komplexen Skalarfeld-Spielzeugmodells erläutert. Die Diskussion wird dann auf Eichtheorien ausgedehnt, für die Lattice Gauge Equivariant Convolutional Neural Networks (L-CNNs) speziell entworfen werden. Regressionen von physikalischen Observablen wie Wilson-Schleifen werden von L-CNNs erfolgreich gelöst, während traditionelle Architekturen, die nicht eichsymmetrisch sind, deutlich schlechter abschneiden. Schließlich stellen wir die Technik des neuronalen Gradientenflusses vor, bei der es sich um eine gewöhnliche Differentialgleichung handelt, die von neuronalen Netzen gelöst wird, und schlagen sie als eine Methode zur Generierung von Gittereichkonfigurationen vor.

iv

Preface

The techniques and results presented in this thesis are largely based on the following published articles:

- S. Bulusu, M. Favoni, A. Ipp D. Müller and D. Schuh, Phys. Lett. D **104**, 074504 (2021) [2103.14686] [1]

- M. Favoni, A. Ipp, D. Müller and D. Schuh, Phys. Rev. Lett. **128**, 032003 (2022) [2012.12901] [2]

- M. Favoni, A. Ipp and D. Müller, EPJ Web of Conferences **274**, 09001 (2022) [2212.00832] [3]

vi

Acknowledgements

First of all, I want to thank my supervisor Andreas Ipp for giving me the opportunity to work on this project. I also want to thank him along with my co-supervisor David Müller for their guidance over the years, especially for managing the pandemic situation seamlessly and making it less burdensome. My colleague Daniel Schuh has been another key presence along this journey; I thank him for his friendship and numerous exchanges on our doctoral projects and on a wide variety of topics. I also thank Srinath Bulusu, whose collaboration was very helpful at the beginning of this project. I want to express my deep gratitude to Alexander Rothkopf for hosting me at the University of Stavanger during my stay abroad. I also thank his collaborators Rasmus Larsen, Daniel Alvestad and Gaurang Parkar for useful discussions. Finally, I wholeheartedly thank my family, my mother Donata Danna, my father Dario Favoni, and my partner Letizia Savino for their constant support and for always believing in me.

viii

Contents

Chapter 1

Introduction

In the past century, quantum field theory has established itself as the appropriate mathematical framework to describe the fundamental constituents of nature. The Standard Model of particles [4, 5] is the most successful theory based on such a framework, in that it reproduces all the experimental results found at particle colliders within a 5σ confidence level. Tensions between this theory and experiments do exist, for example for the value of the anomalous magnetic dipole moment [6, 7], and, even more importantly, some crucial questions are left unaddressed, such as neutrino oscillations and dark matter. Precise predictions are paramount to reveal deviations from experiments and to identify promising candidate theories to explain physics beyond the Standard Model. Perturbative calculations are possible, but other approaches have been developed. One of the most prominent is lattice field theory, a discretization of quantum field theory on a fictitious grid. Its great appeal lies in the possibility of treating non-perturbative problems, arising for example in the low-energy regime of quantum chromodynamics (QCD), and, moreover, it enables to run simulations on a computer. These are typically performed with Monte Carlo methods [8], which are very effective but run into issues such as critical slowing down [9] and the sign problem [10, 11].

The scientific community has witnessed a deep surge of interest in artificial intelligence in the last decade, with a large variety of applications in many fields, such as image recognition [12], natural language processing [13], games like chess [14], protein folding [15] and cancer detection [16]. A very successful subfield of artificial intelligence is machine learning, which is an umbrella term for programs capable of gradually improving their performance on a given task without explicit instructions. This resembles the human way of learning from experience, and the brain structure has indeed inspired the main machine learning algorithm in use nowadays: artificial neural networks. The elementary unit of such models is an artificial neuron (also called unit, node or simply neuron), and is linked to other neurons

via edges, mimicking how biological neurons are connected to each other via synapses in the brain. Artificial neurons receive input signals, which are typically real numbers, multiply each of them by a weight, and output the sum of the result. Usually, neurons are followed by an activation function which introduces some non-linearity [17].

The perceptron [18, 19], an architecture comprising of just one neuron and an activation function, was proposed in 1957 for binary classification and represents one of the very first machine learning models to be implemented. In a multi-layer perceptron (MLP), also referred to as dense network, neurons are interconnected and organized in layers, with a visible layer representing the input that is then processed by a sequence of hidden layers leading to the output. The larger the number of hidden layers, the deeper the network is said to be, hence the name deep learning [20]. Stacking many layers one after the other allows the network to identify relevant pieces of information and patterns, starting from simple concepts and combining them to achieve higher levels of abstraction. MLPs are feed-forward neural networks, that is, the flow of information only goes from the input to the output. Therefore, no backward connections between neurons are present.

One of the strategies to deal with data is supervised learning, in which a network is provided with a dataset containing some input and the corresponding output data, also called labels. The parameters of the network (i.e. the aforementioned weights multiplying the input of each neuron) are usually randomly initialized and, consequently, the network's outputs are in general very distant from the labels. The distance is encoded in a loss function. It is common practice to split the dataset into training, validation and test set. The training samples are used to minimize the loss function by updating the network's weights using some form of gradient descent [21]. The validation set helps to avoid a phenomenon called overfitting [22], which causes the network to memorize the samples without truly extracting useful information. The test set reports on the ability of the network to generalize to unseen data. Given this procedure, a neural network serves as the best approximating function of the true underlying map between the input and the corresponding labels present in the dataset. If the true function is continuous, it is always possible to approximate it with arbitrary accuracy for a sufficiently wide or deep dense network with at least one hidden layer and an activation function, as guaranteed by various versions of the universal approximation theorem [23, 24].

Neural networks have recently garnered the attention of the physics community, with successful applications in many fields. In lattice field theory, a pioneering work in phase classification was conducted in [25], followed by applications to ϕ^4 scalar field theory, lattice QCD, Ising, XY, Potts and Yukawa models, where neural networks were applied to classical [26] and topological [27] phase transitions, to universality classes [28], and to infer order parameters [29], action parameters [30] and thermodynamic proper-

ties [31]. Several efforts have been made to alleviate the sign problem with the aid of neural-network-based algorithms [32–35]. Fruitful applications have been reported in the context of the renormalization group [36, 37] and also vice versa renormalization group transformations have been used to interpret machine learning models [38]. A thriving field is represented by generative models, in particular normalizing flows [39], although another option is represented by diffusion models [40]. A strong relationship between stochastic normalizing flows [41] and Jarzynski's equality was found [42], which helped to mitigate topological freezing with out-of-equilibrium simulations [43] and sampling the lattice Nambu-Goto string [44].

Since Noether published her theorem [45] more than a hundred years ago, the role of symmetries in physics has become pivotal, especially in field theories. For instance, the Standard Model describes the three fundamental interactions between elementary particles as gauge theories. Even though symmetry-agnostic neural networks have been proven to be a powerful tool also in lattice field theory, constraining them to respect the underlying symmetries can facilitate the learning process. This acknowledgement had an impact also in computer vision. Images are typically characterized by global translational equivariance, which induces convolutions [46]. These constitute the basis of convolutional neural networks (CNNs). A convolution is defined as the sum of the element-wise product of a kernel (or filter) with an equally-sized patch of the image. This operation is repeated over the whole input, such that the output is a new image rather than a single output. Since the parameters of the kernel do not change while being applied at each pixel, the convolution has the property of weight sharing, which is essential for translational symmetry. Also, a compact kernel enables the network to scan the image in search of local features. Many kernels with different weights can be applied and their outputs are organized in separate channels, and several convolutional layers can be stacked, usually interposing activation functions and pooling layers. These are usually employed for downsampling. Forms of the universal approximation theorem exist also for CNNs [47, 48]. The main ingredients of CNNs were introduced with the Neocognitron [49] in 1979, and more recently the advantages of CNNs became apparent. For instance, they played a major role in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [12], a competition held yearly from 2010 to 2017, with AlexNet [50] becoming the first CNN to win it in 2012. Despite the success, most of the CNN architectures used in ILSVRC do not preserve symmetry under translations. For example, a flattening layer completely breaks the symmetry. This can be avoided by substituting it with a global average pooling (GAP) [51], which is a feature of ResNet [52], the winning CNN architecture of ILSVRC in 2015. In lattice field theory too, CNNs that are not translationally invariant have been employed even though translational symmetry was a property of the system under consideration [26, 28, 29, 38, 53–55]. In Chapter 2, we will extensively discuss the symmetry properties of CNNs

and show the benefits of applying a symmetry-respecting network in the context of a complex scalar field theory.

Available a priori knowledge about specific problems has prompted researchers to design ad-hoc neural networks, as in the case of physics-informed neural networks [56], Hamiltonian neural networks [57] and Lagrangian neural networks [58, 59]. The commitment to incorporate symmetries stands out, and has given rise to a very active field, with a focus on global symmetries, with group equivariant convolutional networks [60–78] and on local ones, with gauge equivariant convolutional networks [2, 79–86] and equivariant transformers [87, 88]. Chapter 3 is devoted to the design of a neural network able to preserve gauge symmetry on the lattice.

In the context of configuration sampling, a particularly active field is represented by gauge equivariant normalizing flows [31, 89–93]. In Chapter 4, we propose a possible alternative method to generate gauge link configurations, similar to the one proposed in [94].

Finally, we summarize our findings in the Conclusion chapter 5 and present further details in the Appendix.

Chapter 2

Translationally-symmetric neural networks in scalar field theory

The numerous successful applications of neural networks in scientific contexts represent an empirical demonstration that they are a powerful and reliable tool. This is also supported from a theoretical standpoint, in that neural networks can learn to approximate any function. In practice, though, there can be many obstacles preventing a neural network from an effective learning procedure. For instance, a limited amount of samples in the dataset, the training phase requiring too many iterations, the optimization procedure getting stuck in unsatisfactory local minima of the loss function. An approach that aims at facilitating learning is to take into account the symmetries of the problem under examination and to design the network in a way that preserves these symmetries. In physics, and in particular in field theories, systems are often invariant under translations. An example is a complex scalar field theory, which is the focus of this Chapter. We first describe such a theory and discretize it on the lattice. Then we explain how a non-zero chemical potential gives rise to a sign problem and how it can be circumvented with the use of a duality transformation. After that, individual layers of CNNs are presented and their translational symmetry properties are discussed. Three architecture types are proposed and their performance is evaluated on three different tasks. For each task, we show how the incorporation of translational symmetry proves to be beneficial for the network's performance.

2.1   Dual formulation of a complex scalar field

The Lagrangian density of a complex scalar field $\phi(x^\mu)$ subject to the potential $V(\phi^*\phi)$ can be written as

$$\mathcal{L}(\partial_\mu\phi, \partial_\mu\phi^*, \phi, \phi^*) = \partial_\mu\phi^*(x)\partial^\mu\phi(x) - V(\phi^*(x)\phi(x)), \qquad (2.1.1)$$

with $\mu \in \{0, 1, ..., d-1\}$ running over the d spacetime indices and the convention used for the metric is $\eta_{\mu\nu} = \mathrm{diag}(+, -, \ldots, -)$. The action

$$S = \int \mathrm{d}^d x \, \mathcal{L} \qquad (2.1.2)$$

is then invariant under translations in time and space $x'^\mu = x^\mu + a^\mu$. The field transforms as

$$\phi'(x'^\mu) = \phi(x^\mu), \qquad (2.1.3)$$

while derivatives remain unchanged:

$$\partial'_\mu = \frac{\partial}{\partial x'^\mu} = \frac{\partial x^\nu}{\partial x'^\mu}\frac{\partial}{\partial x^\nu} = \frac{\partial(x'^\nu - a^\nu)}{\partial x'^\mu}\frac{\partial}{\partial x^\nu} = \delta^\nu_\mu\frac{\partial}{\partial x^\nu} = \frac{\partial}{\partial x^\mu} = \partial_\mu. \quad (2.1.4)$$

In the action of the field ϕ'

$$S' = \int \mathrm{d}^d x' \left(\partial'_\mu \phi'^*(x') \, \partial'^\mu \phi'(x') - V(\phi'^*(x') \, \phi'(x')) \right), \qquad (2.1.5)$$

the integration measure is not modified by a translation, nor are the kinetic term and the interacting term, as one can see from Eqs. (2.1.3) and (2.1.4). This proves that the action is invariant under translations. As a remark, if the Lagrangian density explicitly depends on the position x^μ the action is manifestly not translationally invariant.

As stated in Noether's first theorem [45], finite continuous symmetries of the action are associated with the existence of conserved quantities, which in the case of invariance under spacetime translations are represented by the energy-momentum tensor.

The action (2.1.2) also exhibits a global $U(1)$ symmetry under the transformation $\phi \to e^{i\alpha}\phi$, which implies the presence of another conserved charge. In this context, a chemical potential can be introduced with a modification of the time derivative given by $D_0 = \partial_0 - i\mu$ [95]. The action then becomes

$$S = \int \mathrm{d}x_0 \, \mathrm{d}^{d-1}x \left(|D_0\phi|^2 - |\partial_i\phi|^2 - V(|\phi|) \right), \qquad (2.1.6)$$

where $i \in \{1, 2, \ldots, d-1\}$ runs over the space indices. In the following, we will focus on a complex scalar field with quartic interaction subject to the potential

$$V(|\phi|) = m^2|\phi|^2 + \lambda|\phi|^4, \qquad (2.1.7)$$

where m is the mass and λ the coupling constant. The action takes the form

$$S = \int \mathrm{d}x_0 \, \mathrm{d}^{D-1}x \left(|D_0\phi|^2 - |\partial_i\phi|^2 - m^2|\phi|^2 - \lambda|\phi|^4 \right). \tag{2.1.8}$$

It is possible to introduce an imaginary time coordinate by performing the Wick rotation

$$x_0 = ix_d, \quad x_d \in \mathbb{R}. \tag{2.1.9}$$

We explicitly write the effect of this transformation on the time covariant derivative:

$$|D_0\phi|^2 = D_0\phi \, (D_0\phi)^\dagger = (\partial_0 - i\mu)\phi \, (\partial_0 + i\mu)\phi^* = -(\partial_d + \mu)\phi \, (\partial_d - \mu)\phi^*, \tag{2.1.10}$$

where we used the fact that $\partial_0 = -i\partial_d$. We can define the Euclidean action S_E as follows:

$$S_E = iS = \int \mathrm{d}^d x \left(|\partial_i\phi|^2 + (\partial_d + \mu)\phi \, (\partial_d - \mu)\phi^* + m^2|\phi|^2 + \lambda|\phi|^4 \right). \tag{2.1.11}$$

We now proceed to the discretization of the theory. The points that define the lattice are located at the positions $x^\mu = an^\mu$, with $n^\mu \in \mathbb{R}$ and $a = a^i \; \forall i \in \{1, \ldots, d\}$, thus assuming the lattice to be equally spaced in every dimension. The integration over spacetime is replaced by a sum over all lattice sites,

$$\int \mathrm{d}^d x \to a^d \sum_x. \tag{2.1.12}$$

Derivatives cannot be taken on the lattice because of its discontinuous nature, therefore a natural substitute is the difference quotient

$$\partial_\mu \phi(x) \to \frac{\phi_{x+a\hat{\mu}} - \phi_x}{a}, \tag{2.1.13}$$

which, in the $a \to 0$ limit, coincides with the definition of the derivative. The notation $\hat{\mu}$ refers to the unit vector pointing in the μ direction. It is possible to reformulate the two terms involving the chemical potential in Eq. (2.1.11) as follows:

$$\pm\mu = -\partial_d \left(\mathrm{e}^{\mp\mu x_d} \right) \mathrm{e}^{\pm\mu x_d} \to \frac{\mathrm{e}^{\mp\mu(x_d+a)} - \mathrm{e}^{\mp\mu x_d}}{a} \mathrm{e}^{\pm\mu x_d} = \frac{\mathrm{e}^{\mp\mu a} - 1}{a}, \tag{2.1.14}$$

thus yielding

$$(\partial_d + \mu)\phi \to \frac{\phi_{x+a\hat{d}} - \mathrm{e}^{-\mu a}\phi_x}{a} \tag{2.1.15}$$

$$(\partial_d - \mu)\phi^* \to \frac{\phi^*_{x+a\hat{d}} - \mathrm{e}^{\mu a}\phi^*_x}{a}. \tag{2.1.16}$$

The discretized version of the Euclidean action can be reworked in the following way:

$$S_E \to a^d \sum_x \left[\sum_{\nu=1}^{d-1} \left(\frac{\phi_{x+a\hat\nu} - \phi_x}{a} \frac{\phi_{x+a\hat\nu}^* - \phi_x^*}{a} \right) \right.$$
$$\left. + \frac{\phi_{x+a\hat d} - e^{-\mu a}\phi_x}{a} \frac{\phi_{x+a\hat d}^* - e^{\mu a}\phi_x^*}{a} + m^2|\phi_x|^2 + \lambda|\phi_x|^4 \right] =$$
$$= a^{d-2} \sum_x \left[\left(2d + (am)^2 \right) |\phi_x|^2 + a^2\lambda|\phi_x|^4 \right.$$
$$\left. - \sum_{\nu=1}^{2} \left(e^{a\mu\,\delta_{\nu,2}} \phi_x^* \phi_{x+\hat\nu} + e^{-a\mu\,\delta_{\nu,2}} \phi_x^* \phi_{x-\hat\nu} \right) \right],$$
$$(2.1.17)$$

where $\delta_{\nu,2}$ denotes the Kronecker delta and we made use of the observation that

$$\sum_x \sum_{\nu=1}^{d} \phi_{x+a\hat\nu} \, \phi_{x+a\hat\nu}^* = \sum_x \sum_{\nu=1}^{d} \phi_x \, \phi_x^* = d \sum_x |\phi_x|^2. \qquad (2.1.18)$$

Dimensionless physical parameters are introduced by

$$m' = am,$$
$$\eta = 2d + m'^2,$$
$$\mu' = a\mu,$$
$$\lambda' = a^2\lambda. \qquad (2.1.19)$$

We restrict our interest to a $1+1$ dimensional system with extension L in the space direction and the inverse of the temperature $1/T$ in the imaginary time coordinate. The lattice is equipped with periodic boundary conditions and we set $a = 1$. These assumptions lead to the simplification of Eq. (2.1.17) given below:

$$S_{\text{lat}} = \sum_x \left[\eta|\phi_x|^2 + \lambda|\phi_x|^4 - \sum_{\nu=1}^{2} \left(e^{\mu\,\delta_{\nu,2}} \phi_x^* \phi_{x+\hat\nu} + e^{-\mu\,\delta_{\nu,2}} \phi_x^* \phi_{x-\hat\nu} \right) \right].$$
$$(2.1.20)$$

The statistical-mechanical interpretation of the path integral allows to write the partition function

$$Z = \int \mathcal{D}\phi \, e^{-S_{\text{lat}}} \qquad (2.1.21)$$

and treat the term $e^{-S_{\text{lat}}}$ as a probability distribution, which enables Monte Carlo sampling. This is possible only if the action S_{lat} is real, which is due to

the fact that complex weights cannot be interpreted in a probabilistic manner, ruling out Monte Carlo simulations as a tool to study the physics of the system. One often refers to systems exhibiting this type of behavior as suffering a sign problem. As apparent in (2.1.20), non-zero chemical potentials give rise to complex terms. Many approaches have been tried in order to get rid of the sign problem or at least alleviate it [10, 11], recently involving also machine learning techniques [32–35]. For the theory under examination, it can be completely removed by means of a duality transformation, that replaces the complex field ϕ with two integer fields $k_{x,\nu}$ and $l_{x,\nu}$, with one component per each direction $\nu = 1, 2$ and the index $x \in \{1, 2, \ldots, N\}$ is a label for the lattice site. This dual formulation is also called flux representation and the details of the derivation can be found in [96]. The partition function obtained with these new fields is

$$Z = \sum_{\{k,l\}} \left(\prod_{x,\nu} \frac{1}{(|k_{x,\nu}| + l_{x,\nu})! l_{x,\nu}!} \right) \left(\prod_x e^{\mu k_{x,2}} W(f_x) \right)$$
$$\times \left(\prod_x \delta \left(\sum_\nu (k_{x,\nu} - k_{x-\hat{\nu},\nu}) \right) \right), \tag{2.1.22}$$

where the summation is intended to run over all possible values of $k_{x,\nu}$ and $l_{x,\nu}$ at all lattice sites:

$$\sum_{\{k,l\}} = \prod_{x,\nu} \sum_{k_{x,\nu}=-\infty}^{\infty} \sum_{l_{x,\nu}=0}^{\infty}$$
$$= \sum_{k_{1,1}=-\infty}^{\infty} \sum_{l_{1,1}=0}^{\infty} \sum_{k_{1,2}=-\infty}^{\infty} \sum_{l_{1,2}=0}^{\infty} \cdots \sum_{k_{N,2}=-\infty}^{\infty} \sum_{l_{N,2}=0}^{\infty}. \tag{2.1.23}$$

As can be seen in the sum, the field k takes values in $\mathbb{Z}$, while l is a non-negative integer. The third term in Eq. (2.1.22) imposes that the flux conservation

$$\sum_\nu (k_{x,\nu} - k_{x-\hat{\nu},\nu}) = 0 \tag{2.1.24}$$

has to be respected, otherwise the Kronecker delta yields zero, meaning that configurations of k and l fields for which Eq. (2.1.24) is not satisfied do not contribute to the partition function and can therefore be considered unphysical. The function $W(f_x)$ has the expression

$$W(f_x) = \int_0^\infty \mathrm{d}t \, t^{f_x+1} \mathrm{e}^{-\eta t^2 - \lambda t^4}, \tag{2.1.25}$$

and its non-negative integer-valued argument is given by

$$f_x = \sum_\nu [|k_{x,\nu}| + |k_{x-\hat{\nu},\nu}| + 2(l_{x,\nu} + l_{x-\hat{\nu},\nu})]. \tag{2.1.26}$$

This quantity has the following useful properties:

$$\frac{\partial W}{\partial \eta} = -W(f_x + 2), \tag{2.1.27}$$

$$\frac{\partial W}{\partial \lambda} = -W(f_x + 4). \tag{2.1.28}$$

As in statistical mechanics, taking derivatives of the partition function with respect to physical parameters gives us observables that can be computed. For the system under examination, by making use of Eqs. (2.1.27) and (2.1.28) it is possible to compute their ensemble averages with

$$\langle n \rangle = \frac{T}{V} \frac{\partial \ln Z}{\partial \mu} = \frac{1}{N_x N_t} \left\langle \sum_x k_{x,2} \right\rangle, \tag{2.1.29}$$

$$\langle |\phi|^2 \rangle = -\frac{T}{V} \frac{\partial \ln Z}{\partial \eta} = \frac{1}{N_x N_t} \left\langle \sum_x \frac{W(f_x + 2)}{W(f_x)} \right\rangle, \tag{2.1.30}$$

$$\langle |\phi|^4 \rangle = -\frac{T}{V} \frac{\partial \ln Z}{\partial \lambda} = \frac{1}{N_x N_t} \left\langle \sum_x \frac{W(f_x + 4)}{W(f_x)} \right\rangle, \tag{2.1.31}$$

where N_x is the number of lattice sites in the space dimension and N_t refers to the extension in time. In the upcoming studies, the first two quantities will be used, namely the particle number density n and the squared absolute value of the field $|\phi|^2$.

For our purpose, it is crucial to notice that the observables in Eqs. (2.1.29), (2.1.30) and (2.1.31) are invariant under spacetime translations, thus the dual formulation inherits the same symmetry that was present at the level of the action in (2.1.2). The global $U(1)$ symmetry mentioned before is instead reflected in the flux conservation (2.1.24). Here, our focus of interest is translational invariance and in the next section we are going to explain how it is implemented into neural networks.

2.2   Symmetry properties of neural network layers

The physical system discussed so far is an ideal situation where neural networks can be applied. We can associate with a certain configuration $\{k_{x,\mu}, l_{x,\mu}\}$ the values of n and $|\phi|^2$ as given by Eqs. (2.1.29) and (2.1.30), which resembles a typical regression task in image processing, such as age prediction from two-dimensional pictures of faces [97]. There are two main differences worth to be noted, though: first, an image is characterized by one real value per pixel in the grayscale case or three in an RGB image, while here the lattice is defined by four integer values per site; second, the configurations that will be used are equipped with periodic boundary conditions, a property that pictures normally do not possess and is necessary for the system to by translationally invariant.

Neural networks are an excellent tool to approximate any functions, and the universal approximation theorem [23,24] supports this with a solid mathematical foundation. Given the ability of these networks to learn relevant features, it may be expected that also symmetries can be identified as an important aspect of a problem, but in general they cannot be learned exactly, only approximately. The core of this work is the following: if a network respects a certain symmetry by construction, it is expected to perform better than a network that has a similar structure but internally breaks the symmetry. We want to verify this statement focusing first on the aforementioned translational symmetry of the complex scalar field presented so far, and extend the discussion to gauge symmetry in the next chapter. The type of network that will be used here is a CNN. Requiring that such networks respect translational symmetry, formally corresponds to impose that their output is invariant under translations. This translates into a more general concept when discussing individual layers, and as a matter of fact the sufficient condition for a network to be invariant under translations is that its layers are equivariant under translations, which gives the network a higher degree of expressivity. Equivariance of a layer Φ can be defined generally for any symmetry group $\mathcal{G}$ as

$$\Phi(L_g\, x) = L'_g\, \Phi(x), \qquad (2.2.1)$$

where L_g is a group transformation and g is a group element. For situations in which $L_g = L'_g$, equivariance corresponds to the commutative property of the layer with a group transformation. Moreover, invariance is the special case for which $L'_g = 1$.

A network that respects invariance under a global symmetry group is called Group Convolutional Neural Network (G-CNN), as defined in [60] and on which the following discussion is based. If the group $\mathcal{G}$ is the translation group $\mathbb{T}$, the G-CNN structure coincides with one of a conventional CNN.

We will now analyze individual layers and their symmetry properties one by one.

2.2.1 Convolutional layers

The definition of a convolution on a two-dimensional lattice is given by [60]

$$[f \star \psi](x) = \sum_{y \in \mathbb{Z}^2} f(y)\, \psi(y - x) = \sum_{y \in \mathbb{Z}^2} f(x + y)\, \psi(y), \qquad (2.2.2)$$

where the feature map f and the kernel (or filter) ψ are real-valued functions:

$$f : \mathbb{Z}^2 \to \mathbb{R}, \qquad (2.2.3)$$
$$\psi : \mathbb{Z}^2 \to \mathbb{R}. \qquad (2.2.4)$$

The definition (2.2.2) is typically used in the machine learning community, and may differ in other fields, e.g. in mathematics, where the argument of ψ has the opposite sign. Since the kernel's weights are optimized independently of the orientation of the kernel, the two definitions are equivalent from the point of view of the learning process. We also note that in Eq. (2.2.2) a bias term b can be added, but it does not affect the discussion about layer symmetry.

Usually, the kernel is localized, therefore there is only a small subset of $\mathbb{Z}^2$ that is non-zero, but for simplicity we keep the sum running over the whole $\mathbb{Z}^2$ space. The domain of the feature map f is a finite subset $F \subset \mathbb{Z}^2$ equipped with periodic boundary conditions. We avoid writing them explicitly by assuming that the feature map repeats periodically outside F. A convenient way of realizing such boundary conditions is by customizing an appropriate padding. In *PyTorch* [98], it is already implemented and is called circular padding. It is worth noticing that periodicity guarantees that the output of the convolution Eq. (2.2.2) has the same size as the feature map f.

We define a translation of the feature map as

$$[L_t f](x) = f(x - t), \tag{2.2.5}$$

where t is an element of the translation group $\mathbb{T}$, which can be identified with an element of $\mathbb{Z}^2$. Equivariance under translations can be proved for the convolution with the following sequence of equalities:

$$
\begin{aligned}
[L_t f \star \psi](x) &= \sum_{y \in \mathbb{Z}^2} f(y - t)\psi(y - x) \\
&= \sum_{y' \in \mathbb{Z}^2} f(y')\psi(y' - (x - t)) \\
&= [f \star \psi](x - t) \\
&= [L_t[f \star \psi]](x).
\end{aligned}
\tag{2.2.6}
$$

What has been implied in the definition in Eq. (2.2.2) is that the convolution has a stride s of one. The stride of a convolution is the distance between the points where the kernel is applied.

More generally,

$$[f \star \psi]_s(x) = \sum_{y \in \mathbb{Z}^2} f(y)\psi(y - sx) \tag{2.2.7}$$

can be used as a definition for convolutions with any stride $s \geq 1$, which coincides with the previous one in Eq. (2.2.2) if $s = 1$. In the case $s \geq 2$, we will call the convolutions strided, and we highlight two relevant features: the size of the convolution output is smaller than the feature map, and

translational equivariance is broken in general. The latter can be shown by considering a translation $t \in \mathbb{T}$ with its components t_1 and t_2 satisfying both $|t_1| < s$ and $|t_2| < s$. For instance, if we choose $t = (1, 0)$, its action on the feature map results in

$$
\begin{aligned}
[L_t f \star \psi]_s(x) &= \sum_{y \in \mathbb{Z}^2} f(y - t)\psi(y - sx) \\
&= \sum_{y' \in \mathbb{Z}^2} f(y')\psi(y' - sx + t) \\
&= \sum_{y' \in \mathbb{Z}^2} f(y')\psi(y' - s(x - t/s)).
\end{aligned}
\tag{2.2.8}
$$

If equivariance was respected, it would be possible to rewrite the formula above in terms of a shifted position $x' = x - t/s \in \mathbb{Z}^2$, but this is impossible since $t/s \notin \mathbb{Z}^2$. However, strided convolutions are equivariant if we restrict ourselves to the subgroup $\mathbb{T}_s \subset \mathbb{T}$, whose elements are translations which are multiples of s. In such a situation, $t_1/s \in \mathbb{Z}$ and $t_2/s \in \mathbb{Z}$, and we can write the following:

$$
\begin{aligned}
[L_t f \star \psi]_s(x) &= \sum_{y \in \mathbb{Z}^2} f(y - t)\psi(y - sx) \\
&= \sum_{y' \in \mathbb{Z}^2} f(y')\psi(y' - s(x - t/s)) \\
&= \sum_{y' \in \mathbb{Z}^2} f(y')\psi(y' - sx') \\
&= [L_{t'}[f \star \psi]_s](x),
\end{aligned}
\tag{2.2.9}
$$

with $t' = t/s \in \mathbb{T}$. To summarize, a convolutional layer with a given stride is equivariant only under translations which are multiples of that stride. It is possible to achieve equivariance under any translation only in the case $s = 1$. The generalization to more than one feature map, which corresponds to having multiple channels, can be accommodated easily. It is worth noting that a strided convolution is equivalent to a convolution with $s = 1$ followed by a subsampling step, meaning that the input size gets reduced.

2.2.2 Spatial pooling layers

Typically, spatial pooling layers are employed as subsamplers, since the most common choice is a stride $s \geq 2$. Here, however, we will break down spatial pooling into two distinct steps: a pooling step and a subsampling step. For the former, we can start considering average pooling. It can be viewed as a special case of convolution for which the weights of the kernel ψ are all set to the same value, i.e. the reciprocal of the number of points in the

kernel, $1/N_\psi$. This does not affect the equivariance property of the layer, which is instead broken by the subsequent subsampling, except for the subset of translations multiple of the stride, as described previously.

This does not only apply to average pooling, but remains true for any kind of spatial pooling P, as can be shown by examining its action on the feature map f:

$$Pf(x) = \underset{y \in U_x}{P} f(y), \tag{2.2.10}$$

where U_x are subsets of the feature map domain F and the index x denotes how the kernel moves through the feature map. If we now apply a translation to the feature map, then

$$\begin{aligned}
PL_t f(x) &= \underset{y \in U_x}{P} f(y - t) \\
&= \underset{y' \in U_{x-t}}{P} f(y') \\
&= L_t P f(x), \tag{2.2.11}
\end{aligned}$$

which proves that equivariance as defined in Eq. (2.2.1) is preserved. Therefore, as it was for convolutions, also for spatial pooling layers we pinpointed that the symmetry-breaking aspect is the stride, not the layer structure itself.

It is important to emphasize that spatial pooling layers with $s = 1$ do respect translational equivariance and can be included in an architecture if one wishes this property to be incorporated, even though they do not perform subsampling as typically expected.

Flattening and global pooling

A widely spread practice in visual computing is to flatten the output of the convolutional part of the network and process it further with a fully connected linear part. A translation of the input is reflected into a permutation at the level of the flattening layer output, while the weights in the dense layers do not change their position, hence leading in general to a different final output and breaking translational symmetry. This can be avoided by means of a global pooling layer between the last layer in the convolutional part and the first linear layer, an example of which is the GAP, first introduced in [51]. There, a feature map was created for each class, and the average of each feature map was directly fed into a softmax layer. While this method effectively preserves translational symmetry, a more general approach is the insertion of dense layers between global pooling and the softmax operation in order to provide the network with a higher level of expressivity.

(a) Equivariant architecture (EQ)

(b) Strided architecture (ST)

(c) Flattening architecture (FL)
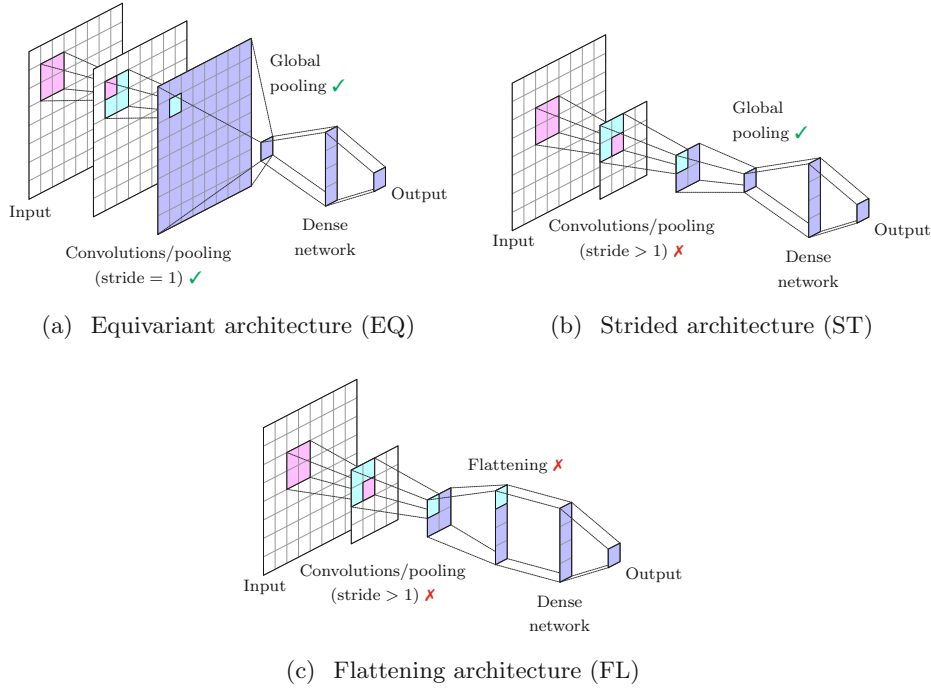
Figure 2.1:   The three different architecture types chosen for the comparison. Operations preserving translational symmetry are indicated by a check mark (✓), while those violating it are marked with a cross (✗). A convolutional or pooling layer with a stride greater than one or a flattening layer are responsible for breaking the symmetry. A global pooling layer allows for the application of the same network to different lattice sizes. The number of channels in each layer (not shown) does not impact the translational symmetry properties. Image from [1].

2.3   Architecture types

In order to study how significant symmetry is for the performance of neural networks, we make a comparison between architectures which respect it and others which break it. Here, we consider three types of architectures. The first one is translationally equivariant, and will be labeled as EQ. The other two do not preserve translational symmetry: a strided architecture, indicated by ST, contains spatial pooling layers with a stride larger than one, and the FL architecture, which uses a flattening step and spatial pooling layers with stride larger than one. After the global pooling or flattening step, it is possible to append a dense network without affecting the symmetry properties of the architecture. The three CNN types are shown in Fig. 2.1.

The FL type is vastly employed on image data, but is affected by a drawback which in lattice field theory contexts is highly relevant: this kind

of architecture is limited to the size of the configurations used for training, while the EQ and ST types can be applied to different sizes.

Lattice studies are typically conducted as approximations of field theories, where spacetime is continuous and infinite. It is important to find a balance between lattice size and computation time. This is because on one side lattice calculations need to yield results as close as possible to the physical case, while on the other larger lattices require longer simulations. For this reason, an advantageous strategy is to train on rather small configurations first and then use the optimized model on larger configurations, since training is the more time-consuming part. If the application on different lattice sizes is not possible, transfer learning [99] can be of assistance, though at the cost of additional training, as is the case for FL architectures.

Let us now examine the architectures in more detail. Input configurations are fed to a convolutional layer, whose stride is one for all architecture types. The kernel size can be odd or even, and the use of circular padding ensures that the output maintains the same size as the input and that translational equivariance is respected also at the boundaries. Next, an activation function such as ReLU is used to introduce non-linearity. This does not undermine equivariance, since activation functions act pointwise. This pair of layers constitutes the main core of a CNN and is consecutively repeated multiple times, meaning that the network becomes deeper and thus becomes more expressive. Non-equivariant architectures also feature the presence of a spatial pooling layer with stride larger than one between at least one of the activation function layers and the following convolution. In the EQ and the ST types, the convolutional part is concluded by a global pooling, while FL architectures end in a flattening layer. Global pooling removes position dependence, so any translational symmetry still existent is inherited by the output, while this is not the case for flattening. Afterwards, an MLP can be used to process the result even further before leading to the output.

There are a few more comments to be made about the architectures. It is possible to include longer-range correlations by the introduction of dilated convolutions [100]. If their stride is one, equivariance is preserved, as can be shown considering that dilated convolutions are equivalent to standard convolutions with a larger kernel in which some of the weights are set to zero.

As mentioned previously, spatial pooling layers with a stride of one do not break translational symmetry, but do not perform any subsampling either. If one is interested in subsampling respecting symmetries, coset pooling [61] meets the requirements. The problems we are tackling in this chapter only involve local quantities, while coset pooling is a non-local operation, hence it is not expected to be an effective choice here.

In the tasks that the CNNs will be given, different types of observables have to be predicted. Depending on whether such observables are extensive or intensive, there can be a fitting choice of global pooling that allows the

network to generalize better across different lattice sizes. In a 2D problem, extensive variables scale with the area, whereas intensive ones remain constant under area changes. This means that in a regression task on an extensive quantity, a global sum pooling is the most suitable option, while a global average pooling factors out the area dependence, which is ideal for an intensive observable. Classification tasks are connected to the prediction of decision boundaries between different categories, therefore the previous discussion is not applicable and in general it is not trivial to identify a priori an optimal global pooling layer.

A strategy that has been used in visual computing to overcome the limitations of a network that breaks a certain symmetry is data augmentation. There, a specific input is passed to the network multiple times after having undergone different group transformations. This is intended to teach the network the invariance under such transformations, effectively learning the underlying symmetry present in the data. We will test this approach for ST and FL architectures in the upcoming task. It has to be stressed though that this does not lead to an exactly invariant neural network, and also that it is not guaranteed to approximate the symmetry well enough on the test set.

2.4 Task I: predicting physical observables

This section deals with a regression task that has already been anticipated and revisits the study performed in [26]. From the ensemble averages in Eqs. (2.1.29) and (2.1.30), we can associate each configuration $\{k_{x,\mu}, l_{x,\mu}\}$ with two values

$$n = \frac{1}{N_x N_t} \sum_x k_{x,2} \,, \tag{2.4.1}$$

$$|\phi|^2 = \frac{1}{N_x N_t} \sum_x \frac{W(f_x + 2)}{W(f_x)}. \tag{2.4.2}$$

A configuration represents the input of the neural network whereas the output are two real numbers which approximate the true labels n and $|\phi|^2$. While n only depends linearly on one component of the fields and is therefore expected to be easy to learn, $|\phi|^2$ depends on a ratio of the highly non-linear function $W(f_x)$, given in Eq. (2.1.25), which depends on the fields through f_x in Eq. (2.1.26).

2.4.1 Physical parameters and lattice sizes

The physics of the system is governed by three parameters, namely λ, η and μ. In the experiments run for this task, we will use specific values of these parameters: the coupling constant and the mass are kept fixed at values $\lambda =$

1 and $\eta = 4 + m^2 = 4.01$, while the chemical potential μ ranges in the interval $[0.91, 1.05]$ with steps of $\Delta\mu = 0.005$ and in the interval $[1.1, 1.5]$ with values separated by $\Delta\mu = 0.1$. The combination of physical parameters with $\mu \in \{0.91, \ldots, 1.5\}$ is analogous to the one employed in [26] to create the dataset. There, the authors trained neural networks on configurations generated with the endpoints of such a range of the chemical potential, i.e. $\mu = \{0.91, 1.05\}$, while the whole interval was utilized for the test set. This is a standard approach in phase classification tasks, as pioneered in [25], and shows the ability of the network to extract information from configurations pertaining to values of the physical parameters not used during training. We argue though that for this system it is sufficient to train on configurations produced at only one of the two chemical potentials, specifically $\mu = 1.05$, in order to endow the networks with good generalization properties. Moreover, we want to check if these properties hold not only for lower μ, but also as the chemical potential is increased, which is the reason for creating the additional dataset with $\mu \in \{1.1, \ldots, 1.5\}$.

There is another parameter that needs to be chosen, namely the lattice size. In [26], it remained fixed to 200×10 throughout the study, which was convenient since an architecture of type FL was employed, while here it is possible to create differently sized configurations, train on a given lattice size, and, in the case of EQ and ST architectures, test the generalization capabilities to different lattice sizes without retraining. The lattice sizes we use are the following: 50×2, 60×4, 100×5, 125×8 and 200×10, where the first number refers to the time extension $N_t = 1/T$ and the second one to the spatial one, $N_x = L$. A different N_t corresponds to a different temperature T, which influences the properties of the phase transition.

2.4.2   Data generation and worm algorithm

The flux representation described in section 2.1 is characterized by the integer field k and the non-negative integer field l constrained by Eq. (2.1.24). Notice how this conservation law only involves the link variable k. The method we intend to use to generate configurations is a standard Monte Carlo algorithm, which is suitable for l. In fact, we can propose a Metropolis step

$$l_{x,\nu} \to l_{x,\nu} \pm 1 \tag{2.4.3}$$

for each site and direction independently and compute its Metropolis acceptance probability as a ratio of Boltzmann weights of the action (2.1.20). On the other hand, applying the same approach to k would disrupt the flux conservation (2.1.24), which would not be easy to restore afterwards. This is where an algorithm originally proposed by Prokof'ev and Svistunov in [101] comes into play. It performs a sequence of local Metropolis update steps each at an adjacent position with respect to the previous site. Concretely,

a random site x and a direction $\nu \in \{\pm 1, \pm 2\}$ are selected at the beginning, and the update

$$k_{x,\nu} \to k_{x,\nu} + \text{sign}(\nu)\xi \qquad (2.4.4)$$

is proposed, with $\xi = \pm 1$. Acceptance probabilities are calculated like in standard Monte Carlo as ratios of Boltzmann weights of the dual action, and if the step is accepted the proposal (2.4.4) is made at $x' = x + \nu$ keeping the same ξ, but randomly choosing a new ν. This is repeated until x' rejoins the initial site x. What we have described follows the prescriptions given in [96]. The resemblance to the movement of a worm led to this algorithm being known as worm algorithm and the paths formed by it as worms, where x' represents the head and x the tail. When the head meets the tail, the worm is said to be closed, otherwise, it is called open. For an open worm, the flux conservation is locally violated at its endpoints, while it is restored at all sites once the worm closes. In order to provide a clear understanding of the worm algorithm mechanism, a possible worm journey is portrayed in Fig. 2.2.

The two algorithms for updating l and k are run alternately starting from a configuration where every field value is set to zero at each lattice point. Initially, the system experiences a thermalization phase that we are not interested in. Configurations and their labels at equilibrium form the dataset, with the caveat that autocorrelated data have to be avoided as much as possible, in particular because networks can learn autocorrelation instead of physical information. For this reason, we keep autocorrelation below a certain threshold by performing a sufficient number of sweeps between each data saving.

2.4.3 Dataset

The whole training set comprises a total of $N_{\text{train}} = 20\,000$ samples, and the entire validation set consists of $N_{\text{val}} = 2\,000$ data points, both generated at $\mu = 1.05$ with a lattice size of 60×4. The test set contains $4\,000$ samples for every μ and lattice size that have been mentioned in 2.4.1, amounting to a total of 6.8×10^5 samples. We will refer to the test set pertaining to chemical potentials $\mu \in \{0.91, \ldots, 1.05\}$ as test set A, while test set B encompasses the values $\mu \in \{1.1, \ldots, 1.5\}$. This distinction is introduced because test set A can be used to make a direct comparison with [26], since the chemical potentials used are the same, while test set B informs about the generalization capabilities of the networks to μ higher than the one used during training. Thermalization effects are eliminated by discarding data generated in the first $1\,000$ sweeps, after which configurations and corresponding observables are saved every five sweeps. For a few combinations of chemical potential and lattice size, a high autocorrelation is observed. In these cases,
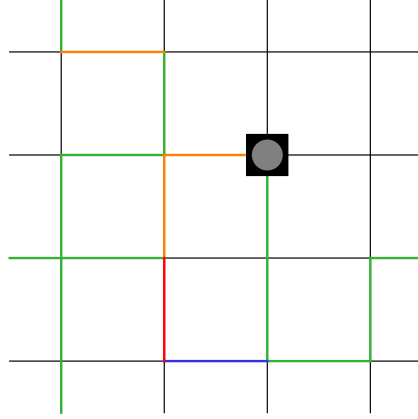
(a) We start from a physically allowed open path configuration, i.e. the flux is conserved at every site. All links are intended to be pointing in the positive directions of space (horizontal axis) and time (vertical axis). The head and tail of the worm are randomly selected.
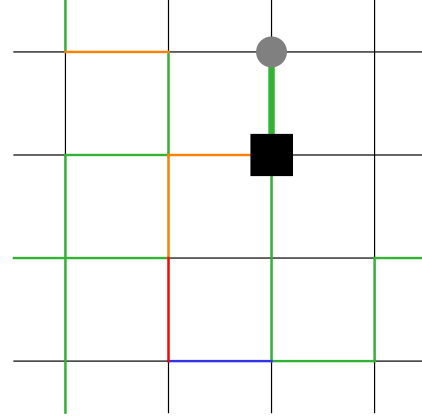
(b) The movement of the head of the worm upwards with the sign $\xi = +1$ is proposed and accepted, like all the next shifts. Links that have been just modified are highlighted with a greater thickness.
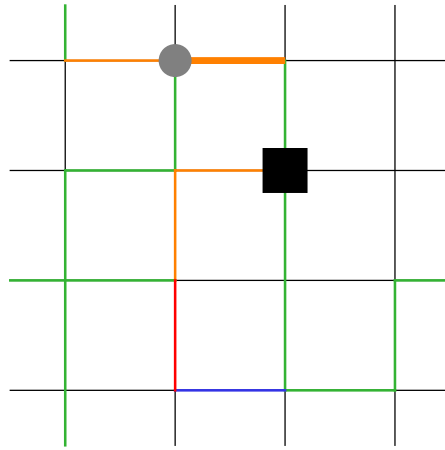
(c) The head is shifted to the left, therefore $\text{sign}(\nu) = -1$ and the link is decreased according to Eq. (2.4.4). It is important to stress that the value of ξ remains equal until the worm is closed.
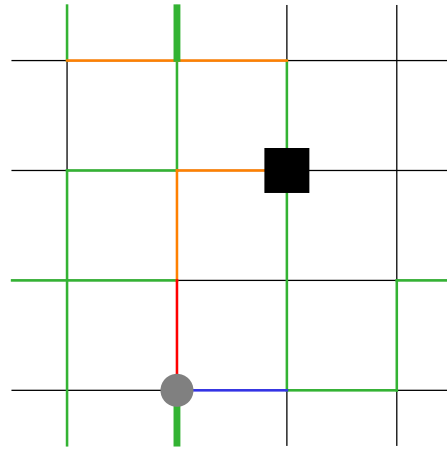
(d) This upward movement illustrates the worm algorithm mechanism on a lattice equipped with periodic boundary conditions.
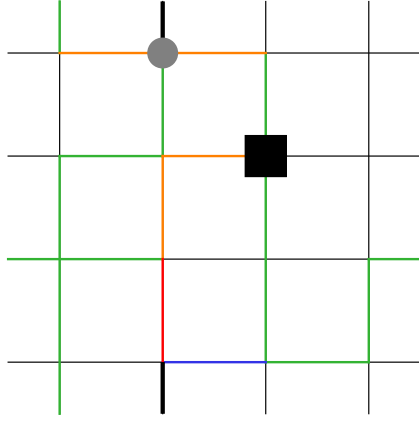
(e) The head is pushed back to the previous site, effectively undoing the previous movement. This highlights that the direction ν is always chosen randomly, hence these back-and-forth steps are possible.
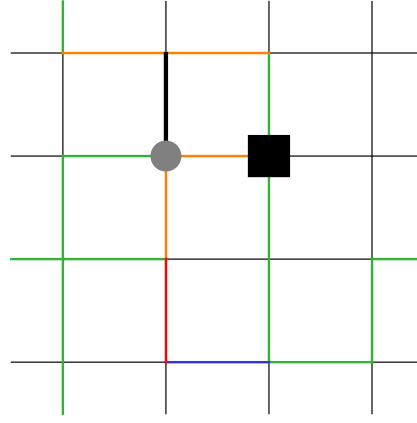
(f) Another downward movement is depicted, with the link value decreased by 1.
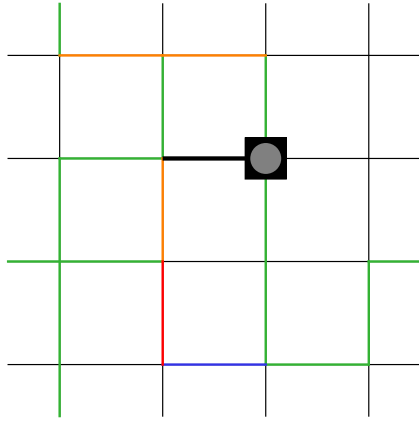
(g) The head shifts to the right rejoining the tail. The worm is closed, and, before starting a new one from a randomly chosen lattice site, a standard Monte Carlo update of the field $l_{x,\nu}$ is executed.
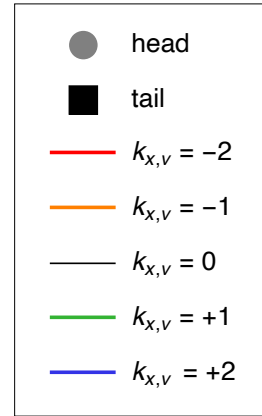
(h) Legend of the previous pictures, clarifying the worm movements and the link values $k_{x,\nu}$.

Figure 2.2: The pictures above illustrate six possible consecutive movements of the worm in a two-dimensional lattice with size $N_t = N_x = 4$.

the number of sweeps is increased to 50, such that autocorrelation becomes negligible.

We now take a closer look at the distributions of $k_{x,2}$ and f_x. These two quantities are crucial for the computation of the observables n and $|\phi|^2$ respectively, hence checking their distributions can provide a better understanding of the dataset properties and can also shed more light on the possible reasons behind the success or failure of a model. Let us remind again that $k_{x,2}$ is an integer number, while f_x can be either 0 or a positive even number, as can be checked by making use of its definition (2.1.26) and the flux conservation (2.1.24). In the rest of the section, we drop the lattice index x and adjust the notation as follows: $k_{x,1} \to k_x$, $k_{x,2} \to k_t$, $f_x \to f$.

In Fig. 2.3, we report the distribution of k_t for the training and testing phase. Specifically, the upper histogram displays the distribution of the training set, while the second and the third show the distributions of test sets A and B, respectively. The range of k_t in the training set and test set A is almost identical, meaning that values used for testing have also been encountered by the networks during training, but the distributions are fairly dissimilar, which makes this task non-trivial. For what concerns test set B, the domain includes higher values, making this test in fact an analysis on the ability of the networks to extrapolate to entirely unseen data. Similar considerations can be made also for f, whose distributions are shown in Fig.2.4.

2.4.4   Architecture search

In machine learning applications to physics, it is advisable to exploit any available knowledge about the system under examination to build a network, as learning can be facilitated and performance enhanced. Here, if we assume no prior knowledge of the exact expressions for n and $|\phi|^2$, two key considerations can guide us in designing an appropriate architecture to deal with the present task.

The first one deals with the fact that the partition function in Eq. (2.1.22) contains products over all spacetime points, and observables are calculated as derivatives of $\ln Z$, which means that a general expression for observables involves sums over lattice sites. If the observable does not scale with the volume, therefore its definition contains a factor $1/(N_x N_t)$ on a 1+1D lattice, it is intensive and global average pooling is the appropriate layer, while if the observable does scale with the volume, it is extensive and global sum pooling is the suitable operation. Given Eqs. (2.1.29) and (2.1.30), the observables studied in this task are intensive, therefore we use a GAP. The dense part after this layer does not affect the intensive nature of the prediction.

The second point is about the translational symmetry of the action, which implies invariance under translations of the observables, as already discussed in section 2.1. This dictates some restrictions in the architecture
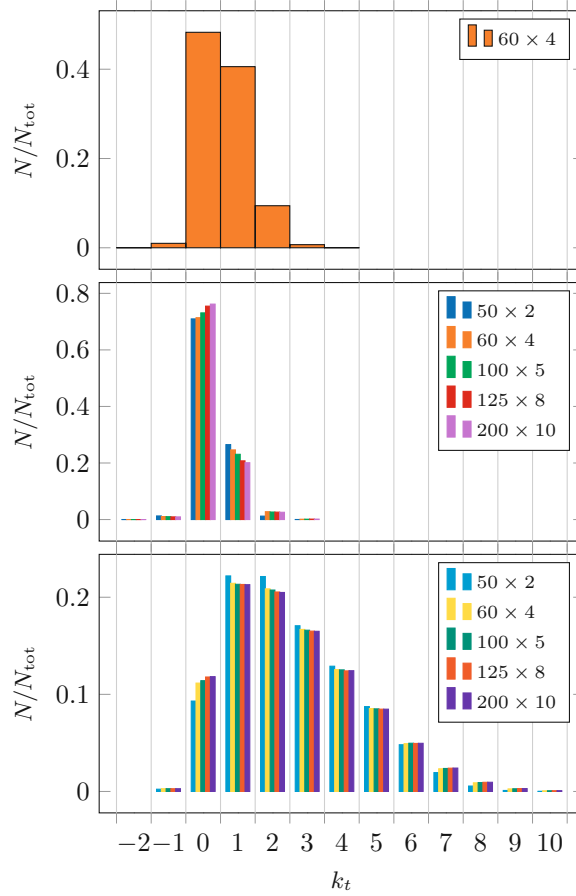
Figure 2.3: Distributions of the field component k_t. These histograms illustrate the distributions of k_t in the training set (top), test set A (middle) and test set B (bottom). The test sets noticeably maintain a consistent distribution across the different lattice sizes. The training set and test set A feature mostly the same values of k_t, although with largely different distributions. Test set B is characterized by a less peaked distribution, where higher values of k_t are reached. Densities below 10^{-4} are not shown. Image from [1].
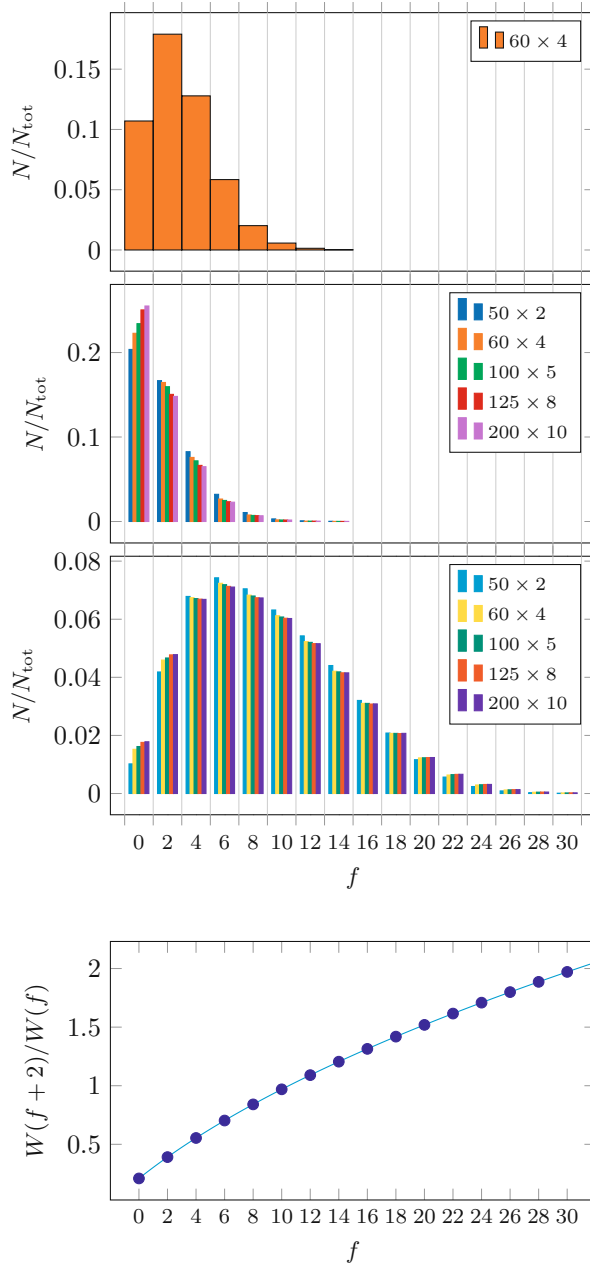
Figure 2.4: Distributions of f and ratio of $W(f)$. The histograms depict the distributions of f in the training set (top histogram), test set A (middle histogram), and test set B (bottom histogram). Similar remarks can be made for the distribution of f in Fig. 2.3. The last plot shows $W(f+2)/W(f)$ with $\eta = 4.01$ and $\lambda = 1$. The markers highlight even integer values of f, which directly enter the calculation of $|\phi|^2$. Weights lower than 10^{-4} are not shown. Image from [1].

Table 2.1: Search spaces for EQ architectures. This table lists the possible number of convolutional (conv, $s = 1$) and linear layers (lin), kernel sizes, the number of channels of the convolutional layers and the number of units in the linear layers. The insertion of spatial pooling layers with $s = 1$ has been avoided in these search spaces as it appears to degrade predictions. Table from [1]

	conv	lin	kernel size	channels/nodes
run 1	$[2, 3]$	$[0, 1]$	$\{(1{\times}1), (2{\times}2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$
run 2	$[2, 4]$	1	$\{(1{\times}1), (2{\times}2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$
extended search	$[2, 4]$	$[0, 3]$	$\{(1{\times}1), (2{\times}2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$

that lead to a preference of the EQ type. Whether such a preference is justified, is the subject of this chapter and will be empirically evaluated by testing the performance of the three architecture types presented in section 2.3.

A comparison can be made, for example, by searching for a very good EQ model, then add at least one spatial pooling layer to form its ST counterpart, and finally replace the global pooling layer with a flattening step to obtain the corresponding FL model. However, it is not guaranteed that the non-equivariant counterparts realized with this approach are representative of well-performing models of the same type, and the analysis could be biased towards equivariant models. In order to carry out a more meaningful and fair study, we define a set of possible hyperparameters and use an optimization strategy to identify the most promising models. The search space for each type can be found in table 2.1 for EQ architectures, table 2.2 for ST and table 2.3 for FL. In a preliminary phase, hyperparameters are tried out manually, which served to establish the search space in the first run. Its results are used to partially reduce the search space in the second run. In both these searches, 50 different combinations of hyperparameters are selected and the models are trained on each of the training sets that will be specified in section 2.4.5. The extended search is intended to explore a larger hyperparameter space with 100 trials, also here with unique combinations of hyperparameters, to check if a better architecture was overlooked because of a too restricted search space. This search is exclusive to the largest training set. Not shown in the tables is the fact that an activation function is employed after every convolution and after every linear layer, except right before the prediction is made. In the first run, different activation functions are tested: *ReLU, tanh, PReLU* and *LeakyReLU* [17]. Due to the better performances obtained, the latter is the only one employed in the second run and in the extended search. An advantage of *LeakyReLU* over *ReLU* lies in the prevention of the well-known problem of dead neurons, which are not active after initialization or become inactive during training.

The difference between ST architectures and EQ is the presence of spatial pooling layers with $s = 2$ in the convolutional part. The kernel of these layers

Table 2.2: Search spaces for ST architectures. Here are presented the possible number of convolutional ($s = 1$) and linear layers (abbreviated as in table 2.1), kernel sizes, the number of channels for the convolutional layers, the number of neurons in the dense part, the number of spatial pooling layers (SPL, $s = 2$) and the spatial pooling mode (SPM). Table from [1]

	conv	lin	kernel size	channels/nodes	SPL	SPM
run 1	[2, 4]	[0, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	$\{1, 2\}$	$\{$avg, max$\}$
run 2	[2, 4]	[0, 2]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	$\{1, 2\}$	avg
extended search	[2, 4]	[0, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	$\{1, 2\}$	$\{$avg, max$\}$

Table 2.3: Search spaces for FL architectures. This table displays the possible number of convolutional ($s = 1$) and linear layers, kernel sizes, the number of channels of the convolutional layers, the number of nodes in the linear layers, the number of spatial pooling layers ($s = 2$) and the spatial pooling mode. The amount of convolutions is not selected directly, but is conditioned on how many 1×1 convolutional layers are chosen by *optuna*. Two 2×2 convolutions and their respective spatial pooling layer are always present. It is possible to additionally insert 1×1 convolutions before each 2×2 convolutional layer or between each of them and the subsequent spatial pooling. These restrictions in the position of each convolution are signaled with an asterisk next to "kernel size". Abbreviations are intended as in table 2.2. Table from [1]

	conv	lin	kernel size*	channels/nodes	SPL	SPM
run 1	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	$\{$avg, max$\}$
run 2	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	avg
extended search	[2, 6]	[1, 3]	$\{(1 \times 1), (2 \times 2)\}$	$\{4, 8, 16, 24, 32, 48, 64, 80\}$	2	$\{$avg, max$\}$

is fixed to 2×2, and each of them can be either a max pooling or an average pooling. The number and the position of these layers is part of the search space, but some limitations are applied. For instance, a spatial pooling has to be inserted between an activation function and a convolutional layer, meaning that it is not allowed to apply it directly to the input or immediately before global pooling. Also, if the number of convolutions is two, i.e. the minimum permitted in the search space, the only choice is to interpose one spatial pooling layer between them.

We design FL architectures drawing inspiration from traditional CNNs used for similar machine learning problems. The building block is formed by a 2×2 convolution followed by a spatial pooling layer with a stride $s = 2$. The architecture consists of two of these building blocks one after the other, with the optional insertion of a 1×1 convolution before each 2×2 convolution and between each of them and the following spatial pooling, with a maximum of six convolutions in total. Activation functions are inserted after each convolutional layer except before flattening.

For the hyperparameter optimization, we choose an automatized optimizer called *optuna* [102]. The metric that has to be minimized is the validation loss averaged over three different parameter initializations. This averaging mitigates statistical fluctuations introduced by random parameter initializations, which is crucial because *optuna* dynamically adjusts its search space, with early results influencing the probability distributions guiding later hyperparameter value selections. This whole procedure is repeated separately for every training set, since for small training set sizes simpler models may perform better, while more complicated ones are likely to be favored for larger sizes.

Once the *optuna* runs are completed, the architectures scoring the lowest average validation loss are selected and ten new instances for each of them are trained from scratch, with the scope of further reducing the influence of random initializations. The best-performing architectures are reported for each type in table 2.4. These architectures can be considered as representatives of their type, because many of the architectures that have been retrained exhibit similar performance, which is the case independently of the training set size. The notation $\text{Conv}(K \times K, N_{\text{in}}, N_{\text{out}})$ refers to a two-dimensional convolution with equal kernel size K for both directions, N_{in} input channels and N_{out} output channels. Circular padding is applied before every convolutional operation to ensure periodic boundary conditions, and a stride of one is chosen for each convolution. Average pooling layers with kernel size K and stride s are denoted as $\text{AvgPool}(K \times K, s)$, and dense layers are written as $\text{Linear}(N_{\text{in}}, N_{\text{out}})$ with N_{in} input nodes and N_{out} output nodes.

Table 2.4: Best architectures for the prediction of n and $|\phi|^2$ for each architecture type. This table presents the most promising architectures obtained with the *optuna* searches. The network inputs consist of field configurations $k_{t,x}, l_{t,x}$ provided in the form of tensors with dimensions $(N_t, N_x, 4)$. The output is formed by two nodes which approximate the observable values. The last row specifies the amount of trainable parameters for each architecture. Table from [1].

EQ	ST	FL
Conv(1×1, 4, 64)	Conv(1×1, 4, 80)	Conv(1×1, 4, 64)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1, 64, 48)	Conv(1×1, 80, 80)	Conv(2×2, 64, 80)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1, 48, 80)	Conv(1×1, 80, 48)	AvgPool(2×2, 2)
LeakyReLU	LeakyReLU	Conv(1×1, 80, 48)
Conv(2×2, 80, 80)	AvgPool(2×2, 2)	LeakyReLU
LeakyReLU	Conv(2×2, 48, 80)	Conv(2×2, 48, 64)
GlobalAvgPool	LeakyReLU	LeakyReLU
Linear(80, 2)	GlobalAvgPool	AvgPool(2×2, 2)
	Linear(80, 2)	Conv(1×1, 64, 24)
		Flatten
		Linear(360, 24)
		LeakyReLU
		Linear(24, 2)
33202	26370	47394

2.4.5 Training and testing

In subsection 2.4.3, the dataset has been introduced and its main characteristics have been discussed. In this section, we give the details of how the data are used during training and testing.

The training phase is repeated for each architecture in Table 2.4 with different training set sizes. The mean squared error (MSE) is used as loss function, averaging over the contribution of the two observables:

$$\mathcal{L} = \frac{1}{2N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left[(n_{i,\text{true}} - n_{i,\text{pred}})^2 + (|\phi|^2_{i,\text{true}} - |\phi|^2_{i,\text{pred}})^2 \right]. \qquad (2.4.5)$$

Optimization is performed using the *AMSGrad* [103] variant of the *AdamW* optimizer [104] without weight decay. The reason for running the experiment with different training sets is to acquire information about sample efficiency. The utility of studying examples with fewer training samples is connected to real situations where generating configurations is highly time-consuming, e.g. in large-scale simulations. The training set size ranges from 100 to 20 000, with larger step sizes for a larger number of samples, as follows:

$$N_{\text{data}} = \{100, 150, 200, 250, 500, 750, 1\,000, 1\,500, 2\,000, 2\,500, 3\,000, 4\,000,$$
$$5\,000, 6\,000, 7\,000, 8\,000, 9\,000, 10\,000, 11\,000, 12\,000, 13\,000, 14\,000,$$
$$15\,000, 16\,000, 17\,000, 18\,000, 19\,000, 20\,000\}. \qquad (2.4.6)$$

Table 2.5: Loss, optimizer and early stopping settings for *PyTorch*. Table from [1].

loss	size_avg	reduce	reduction		
MSELoss	None	None	'mean'		
optimizer	lr	betas	eps	weight_decay	amsgrad
AdamW	0.001	$(0.9, 0.999)$	10^{-8}	0	True
	monitor	min_delta	patience	mode	
EarlyStopping	'val_loss'	0	25	'min'	

The validation sets used in combination with each training set contain a number of samples amounting to $1/10$ of the training set size Eq. (2.4.6). In order to avoid being too close to a batch training situation, the batch size is set to 50 for training set sizes smaller than 500 and 100 for bigger ones. We employ early stopping (see e.g. Section 7.8 of [105]) monitoring the validation loss with a patience of 25 to determine the number of epochs, which has a minimum and maximum threshold of 100 and 1 000 respectively. Patience refers to the number of epochs the training process is allowed to continue without improvement in a chosen metric before stopping. The model parameters are checkpointed when the lowest validation loss is reached. An overview of the settings used in this procedure is given in table 2.5.

Given the large number of maximum epochs and the early stopping criterion based on validation loss, the models are essentially trained until convergence to a very good minimum in the parameter space. As previously mentioned, we train on 60×4 configurations generated with a chemical potential $\mu = 1.05$. The impact of data augmentation in learning translational symmetry and improving the performance is checked for both ST and FL architectures. These are trained with and without data augmentation, which is accomplished by random shifts of the input data that depend on the architecture type. ST architectures include a maximum of two spatial pooling layers with a 2×2 kernel and a stride of 2, as reported in table 2.2. The discussion in Subsec. 2.2 implies that ST architectures feature translational equivariance under shifts that are multiples of 4, therefore data augmentation is achieved by means of shifts of $[0, 3]$ in both dimensions. At the same time, it brings to the conclusion that FL architectures do not incorporate translational equivariance no matter what the applied shift to the input is, meaning that the possible shifts are $[0, 59]$ in the time direction and $[0, 3]$ in the space direction, where the limits come from the lattice extension.

Other than the distinction of test set A and test set B mentioned in Sec. 2.4.3, the models are evaluated both on the whole test set and on the subset that includes only 60×4 configurations in the attempt to check the generalization ability to other lattice sizes and chemical potential separately. Because of the FL architectures being restricted to one lattice size, the second test can only be performed for the EQ and ST types.

Let us point out that the winning ST architecture can be applied to the 50×2 lattice because it features only one spatial pooling layer with a 2×2 kernel and a stride $s = 2$ (see table 2.4).

2.4.6   Results

In the following subsections, the details of various test results are discussed. The architectures are initially evaluated on the subset of test set A containing solely 60×4 configurations, later inspecting the performance on all lattice sizes. An analysis of the Silver Blaze phase transition is also carried out for various lattice sizes. At last, we will evaluate the models on test set B first on the 60×4 lattice then on all lattice sizes, exactly like with test set A, to study the extrapolation to chemical potential larger than the one used while training.

Results on the same lattice size as training

The same loss used for training in Eq. (2.4.5) is employed as a metric for the test set. Its values are plotted against the training set sizes listed in (2.4.6) in Fig. 2.5. The different amount of training samples shows how the three architecture types behave under limited information for smaller training set sizes and the extent of the performance improvement as the information is increased. The top plot indicates that the EQ models achieve a better test loss when the size of the training set becomes larger, as one would expect. On the other hand, the two non-equivariant architectures surprisingly do not benefit from having access to a greater number of samples in the training set. The center and bottom plots report on another noteworthy finding: data augmentation does not help the non-equivariant architectures to improve their results. If we only take into consideration the median of the test loss, we may come to the conclusion that the ST and the FL architectures are not able to reach a precision in the predictions as their equivariant counterpart. A non-equivariant model that has already converged to a very good local minimum in the loss function landscape is not going to benefit from the addition of more training samples, regardless of whether they are generated by Monte Carlo sampling or by data augmentation. The downward spikes in the bands of the ST architecture, though, hint that it is possible for some models to achieve a good approximation of the observables and even compete with the EQ models. Therefore, we conclude that although possible, it is less likely for the ST and FL models than for the EQ models to learn a satisfactory approximation of n and $|\phi|^2$.

At this point, we select the best EQ model according to the test loss, and evaluate it separately at all the chemical potentials present in test set A. In Fig. 2.6, we plot the predictions of the observables averaged over the configuration ensemble that pertains to each μ, together with the exact en-
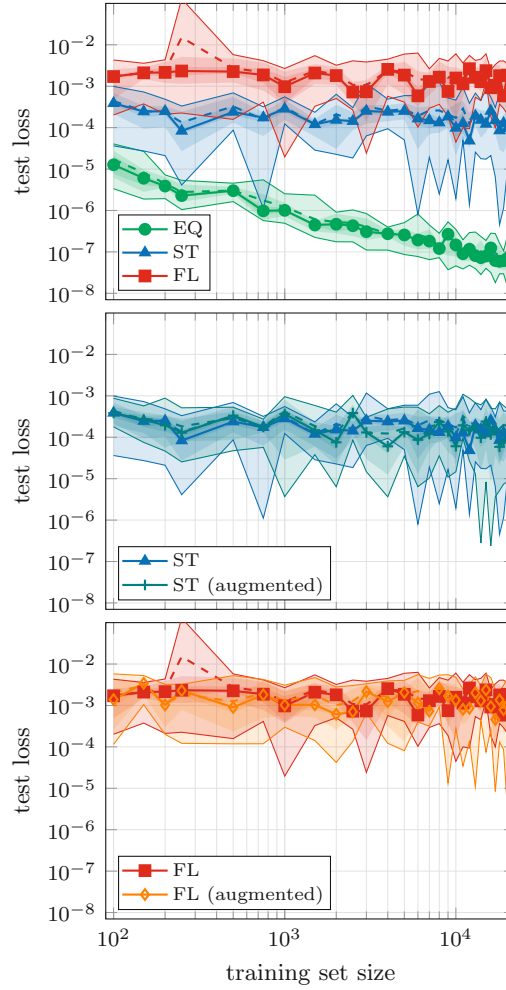
Figure 2.5: Test loss on the test set of all 60×4 configurations as a function of the samples used to train. The top plot displays the results of the three architecture types trained without data augmentation. In the middle and bottom plot, the impact of training with data augmentation is shown respectively for ST and FL models. In each plot, the solid lines represent the best and worst loss, the dashed lines denote the average over the ten models that have been initialized independently, and the shaded regions indicate the 20% quantiles. The symbols signal the median and are connected by a continuous line to guide the eye. The conventions just discussed will be used in similar plots later on. Image from [1].
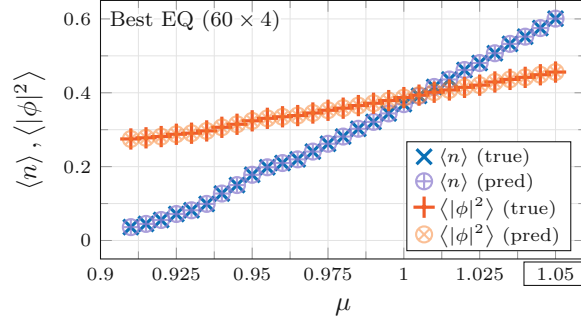
Figure 2.6: Predicted and true values for the ensemble averages $\langle n \rangle$ and $\langle |\phi|^2 \rangle$ as functions of the chemical potential μ on a 60×4 lattice. The predictions shown are generated by the EQ model which achieved the smallest test loss. Notably, the model has been trained solely on configurations generated at $\mu = 1.05$ but exhibits remarkable generalization capabilities to other values of μ. The training point is highlighted by a rectangle in this and subsequent plots. Image from [1].

semble average. The figure illustrates that the network can generalize to all the chemical potentials in the given range, despite being trained uniquely on samples generated with $\mu = 1.05$. This remarkable generalization ability is explained by the fact that the model is not really generalizing from one value of the chemical potential to all the others, but rather from the training samples to other samples, each consisting of a configuration and the corresponding value of the observables. The samples present in the training set feature a range of values for n and $|\phi|^2$ which is sufficiently large to cover most of the observable values in the test set. A similar observation has been made about the input distributions in subsection 2.4.3, where k_t and f_x span the same range in the training set and test set.

To illustrate this point, we employ ST models that have been trained on 18 000 samples. In figure 2.7, the test results are reported in the form of a scatter plot of true values versus the predicted ones of both observables, which is done for the best and the worst performing (according to the test loss) ST model respectively in the top and middle plot. The bottom plot displays the performance of the worst ST model on the training data. In these scatter plots the model's predictions for each sample are drawn, not just the average over the ensemble. Both networks can predict the larger values of both observables, but the worst one largely mispredicts the smaller values, because they are not present in the training set. Therefore, we can conclude that the better ST models are able to generalize to configurations and ranges of the two labels that were missing in the training phase, while the worse ones do not possess this ability. The relatively large values of the median in Fig. 2.5 suggest that the instances of the ST architecture have a
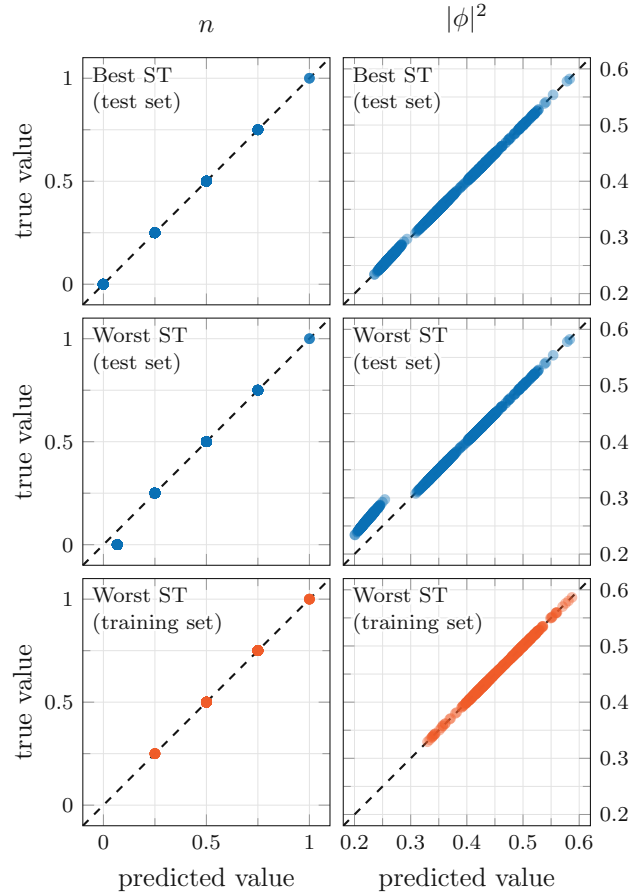
Figure 2.7: True versus predicted observables for the best and the worst ST network, both trained on 18 000 samples. The two top plots illustrate that the best instance of the ST architecture accurately estimates the observable values over the their whole range. Conversely, the middle plots demonstrate that the worst instance fails to predict correctly the smaller values of n and $|\phi|^2$. The noticeable discrepancy between ground truth and prediction arises from the training set including only larger values of the observables, as shown in the bottom plot, and the worst model struggling to generalize beyond that range. The top and middle plots show 1% of the test data, while the bottom plots display 4% of the training data. Image from [1].
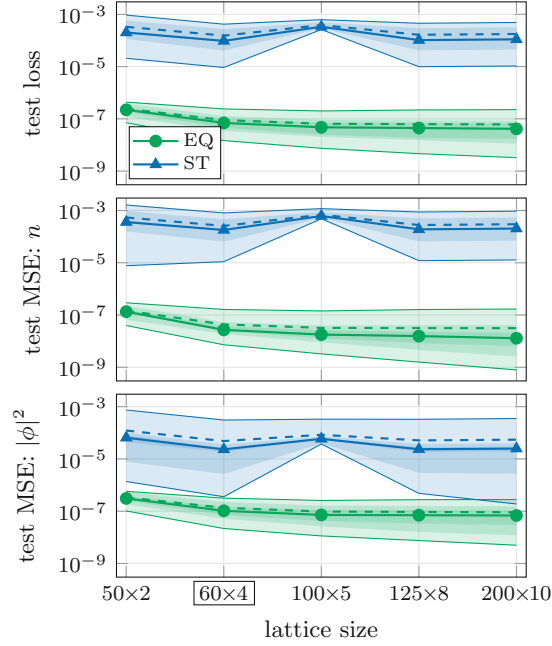
Figure 2.8: Overall test loss (top) and its contributions coming from each observable (middle and bottom) on various lattice sizes. Training was executed on 60×4 configurations. On average, the EQ type consistently outperforms the non-equivariant counterpart. The performance does not deteriorate when testing on lattice sizes that were not used for training for neither of the two types, with the only exception of the ST architecture (depicted in blue) when tested on the 100×5 lattice. What spoils the generalization capabilities of the models is the stride of the spatial pooling layer present in the ST architecture, which allows the use of the first four rows but not of the fifth one, consequently discarding 20% of the information which results in a higher loss. Image from [1].

low probability of achieving good generalization. Concerning the FL models, their behavior is similar to the ST models, but overall the predictions are less accurate.

Results on different lattice sizes

Different lattice sizes are not suitable for the same FL architecture that was used in the previous tests, therefore the comparison here is made between the EQ and ST type. Since data augmentation does not change the performance of the ST architecture significantly, the test is run only for models trained without data augmentation. Additionally, we will set the training set size to 20 000 training samples for this comparison.

In Fig. 2.8 the total test loss in Eq. (2.4.5) is shown, while the losses of each observable are portrayed in the middle and bottom plot. Despite the ST architecture maintaining a worse performance for all lattice sizes with respect to the EQ architecture, both types display a remarkable generalization ability. The only exception is found for the ST networks on the 100×5 lattice, where a kink in the blue curve is present in the prediction of both observables, an anomaly that does not manifest itself for EQ architecture. This issue arises from the odd number in the lattice dimension, which in combination with the ST architecture leads to an information loss. Let us review the effect of the ST architecture in Table 2.4 on configurations with such a lattice size: the first three convolutions do not affect the input size because of the application of circular padding. Then, the 100×5 feature maps are passed to a spatial pooling layer with a 2×2 kernel and a stride of 2. This layer operates on the first four rows and discards the last one, effectively wasting 20% of the information, and outputs a feature map with size 50×2. For this reason, the ST networks cannot use all of the data to make predictions, resulting in a less accurate test loss. An analogous observation can be made for the 125×8 lattice, but in that case the missing information only amounts to $1/125$ of the total available, whose effect is not visible in Fig. 2.8.

Silver Blaze phase transition

The Silver Blaze [106] phenomenon is a second-order phase transition occurring at zero temperature T, for which thermodynamical quantities are independent of the chemical potential μ below a critical value μ_c [96]. Specifically, the observables $\langle n \rangle$ and $\langle |\phi|^2 \rangle$ remain constant when $\mu < \mu_c$, and they start increasing once the chemical potential goes above the μ_c threshold. The particle density $\langle n \rangle$ serves as an order parameter for the Silver Blaze phase transition. Due to the finite size of the lattices, the temperature cannot vanish completely, hence the transition is not guaranteed to be sharp. Given that the networks are trained to approximate two observables that are relevant to study this phenomenon, their predictions are expected to reproduce the Silver Blaze phase transition.

In Figure 2.9 are illustrated the predictions of the EQ model that has been trained on $20\,000$ training samples and has achieved the lowest validation loss along with the true values. The symbols correspond to the ensemble average of each observable for every chemical potential in test set A on the 100×5, 125×8 and 200×10 lattices respectively in the top, middle and bottom plot. On the larger lattices, the phase transition is visible, while on the smaller ones no phase transition is detected within the range of μ under examination. The reason for this is that the critical value μ_c decreases with increasing temperature.

The phase transition is accurately predicted also by the ST models that
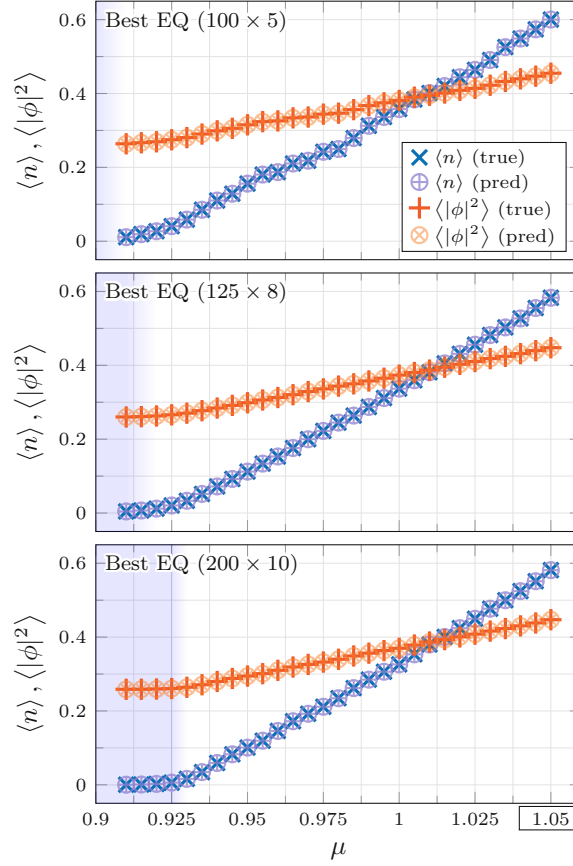
Figure 2.9: Predicted and true ensemble averages of the physical observables at each μ on the larger lattices. The EQ model trained on $20\,000$ samples with the best performance on the test set has been selected to provide the predictions shown in the plot. Training was executed only at $\mu = 1.05$. Notably, the kinks in the curves offer an estimate of the Silver Blaze phase transition, highlighted by the color gradient from the shaded region to the white background. The kinks become more evident as the lattice size increases. Image from [1].

are able to generalize to smaller values of the observables, such as the one that was used in Fig. 2.7, but this is not the case for all the ST models.

Extrapolation to larger chemical potentials

The results obtained by the EQ architecture and some ST models on test set A have demonstrated their ability to generalize to different chemical potentials and lattice sizes. As we have argued, in such a test set the fields and the observables span values that are also found in the training set, which allows some networks to generalize well. We are now interested in investigating the behavior of the models when we increase the value of the chemical potential, reaching field and observable values that were not present while training. Test set B, as described in subsection 2.4.3, serves this scope, therefore we evaluate the models on it without retraining.

Figure 2.10 shows scatter plots of the true values against the predicted ones for both observables on the 60×4 lattice. Each row displays the outcome respectively for the best EQ, ST and FL type, according to the validation loss. All three architectures struggle with the prediction of higher values of $|\phi|^2$ more than with the prediction of higher n. This can be understood considering the simple linear dependence on k_t in Eq. (2.4.1) and the non-linear dependence on all fields in Eq. (2.4.2), as pointed out at the beginning of this section. However, the predictions made by the best EQ model exhibit a closer alignment with the optimal results represented by the identity line compared to the predictions generated by the non-equivariant counterparts, of which the FL architecture displays a more pronounced deviation from the ground truth. This leads to a noticeable discrepancy in the ensemble averages of the observables only for $\mu = 1.5$, and this trend is also consistent across all considered lattice sizes for EQ and ST architectures. The selection criterion for the best architecture is based on the validation loss and should not involve test results, but we point out that there exist some models for each architecture that extrapolated better than the respective ones used in Fig. 2.10.

In Fig. 2.11 the total and individual test losses on 60×4 configurations are plotted against the chemical potential. As expected, the predictions become worse for all the architectures as μ increases, but while for $\mu < \mu_c$ the difference between equivariant and non-equivariant architectures is considerable, for $\mu > \mu_c$ the loss grows more similarly for the three types, although maintaining the same ranking, with EQ models achieving the best results, followed by ST and then FL networks. For instance, at $\mu = 1.5$ the mean and median losses scored by the EQ architecture are lower than those of the non-equivariant ones, but the lowest loss is reached by the ST best model. A similar behavior is observed on the other lattice sizes when testing the EQ and the ST architectures, apart from the 100×5 lattice, where the ST models encounter the same problem shown in Fig. 2.8. We note that
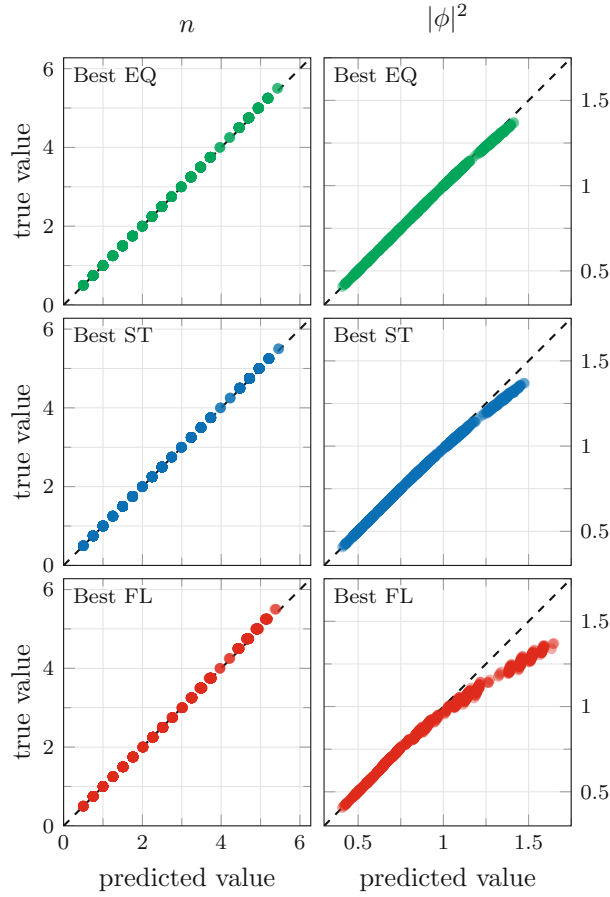
Figure 2.10: Scatter plot of predicted versus true observable values for the model achieving the lowest validation loss for each architecture type, evaluated on 60×4 configurations generated with $\mu \in [1.1, 1.5]$. The particle number density is predicted with good accuracy also for values larger than the ones used during training by all three models, while for $|\phi|^2$ the generalization capability deteriorates as the values lie further away from those belonging to the training set. In the plots, 6.25% of the test data has been employed. Image from [1].
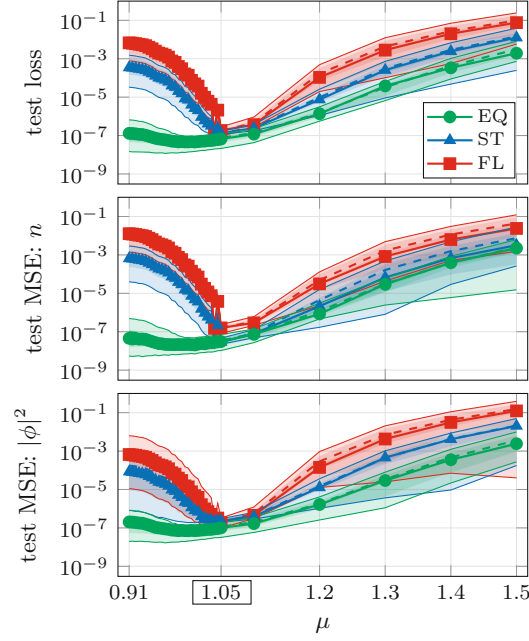
Figure 2.11: Total test loss and its contributions given by the physical observables n and $|\phi|^2$ as functions of the chemical potential on the 60×4 lattice. For each architecture type, the test results of an ensemble of the best architecture trained on 20 000 training samples are illustrated. Together with Fig. 2.5, these plots confirm a substantial difference in the accuracy of the predictions for $\mu \leq 1.05$. For larger chemical potentials, the loss seems to increase in a more similar way for each type, although in the range $1.2 \leq \mu \leq 1.5$ the separation between the average of the total test loss of EQ and ST models is approximately one order of magnitude, just as is between the average loss of ST and EQ models. Image from [1].

Fig. 2.10 visualizes the results of the model with the lowest validation loss associated with each individual architecture. For the EQ type, it is a model of the ensemble that has been trained with 20 000 samples, while for the ST and the FL types, it is a model that has been trained with 18 000 samples. On the other hand, in Fig. 2.11 we used the ensemble of models trained on 20 000 samples for every architecture.

Results summary

To summarize, the best translationally equivariant architecture outperforms the respective best model of the two non-equivariant types on the lattice size they have been trained on. Only a part of the ST networks are found to generalize beyond values of observables they have been shown during training,

while no EQ network has exhibited limitations. The behavior of FL models is analogous to the behavior of ST networks, but their predictions are less accurate overall. The tests on different lattice sizes are not directly possible for FL architectures, and are thus executed only for the EQ and the ST types. These are both capable of generalizing to different lattice sizes, although the ST models score the higher average test loss on the 60×4 lattice, which can be explained by their lack of generalization to observable values absent in the training set. Another restriction is that ST architectures are not suited to make predictions on any arbitrary lattice size: depending on the lattice dimension and the stride of the spatial pooling layers in the network, parts of the information can be lost. In contrast, EQ architectures have no such restriction. Despite being trained solely on $\mu = 1.05$ on the 60×4 lattice, many models effectively predict the Silver Blaze phase transition on a different lattice size, with $\mu_c \ll 1.05$. The EQ models accomplish this result remarkably well. Data augmentation does not provide significant benefits in training ST and FL architectures, which is why this approach is not employed in the next two tasks.

As a last important consideration, the results found on test set A can be compared with the ones in [26], where this regression task was performed on a test set generated with the same physical parameters. The architecture employed there belongs to the FL category, and can therefore only be applied to the lattice size used for training, which in that case was chosen to be 200×10. Training was performed on configurations pertaining to $\mu = 0.91$ in addition to $\mu = 1.05$, thus providing information from both phases. Furthermore, the number of trainable parameters of the architecture in [26] has an order of magnitude of 10^7, much more than our best model needs, which is about 3×10^4. Other well-performing models we found contain about an order of magnitude fewer parameters than our best one. The test loss achieved by the architecture in [26] is of the order of 10^{-6}, while the best EQ model achieves a loss of about an order of magnitude smaller.

2.5   Task II: detecting flux violations

The first task was successfully tackled by relatively simple CNN models, which were favored by the hyperparameter optimizer *optuna*. This is not surprising if we consider that the observable n can be reproduced exactly by an equivariant linear model with a single 1×1 convolution. Since information from next-neighboring sites is required for an exact representation of $|\phi|^2$, it is impossible for a simple 1×1 convolution to perfectly reproduce this observable, however we empirically observed that architectures featuring only 1×1 convolutions were able to satisfactorily approximate it.

In this section, we would like to extend our architecture comparison to a task where a trivial 1×1 convolutional layer cannot be trained to

(a)   Example   field   configuration

(b) Feature maps of convolutional network in best EQ and ST models
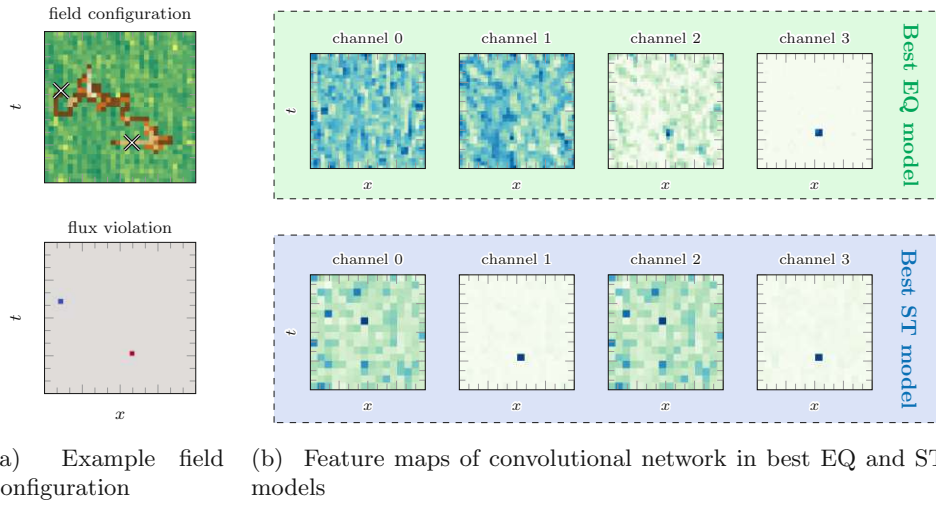
Figure 2.12: Visualization of an open worm field configuration and of one of the best models' feature maps. (a) A field configuration illustrating an open worm (depicted in brown) created with movements like the ones in Fig. 2.2, and the corresponding flux violation computed with Eq. (2.5.1). The violations are located at the two open ends of the worm, marked with crosses in the upper plot. (b) First four channels out of 32 and 16 respectively for the best EQ (top, green) and ST (bottom, blue) models in one of the feature maps of the convolutional part. In some of the channels one of the two flux violations is detected (e.g., channels 2 and 3 for EQ and 1 and 3 for ST), while for others (e.g., 0 and 1 for EQ and 0 and 2 for ST) the output is not straightforwardly interpretable. Image from [1].

provide almost perfect predictions and information from multiple lattice sites is simultaneously required. The quantity we identified as a good candidate is the local flux

$$\mathcal{F}_x \equiv \sum_{\nu=1}^{D} (k_{x,\nu} - k_{x-\hat{\nu},\nu}) \in \mathbb{Z}. \tag{2.5.1}$$

This is also a local quantity, but unlike $|\phi|^2$ it requires field values from nearest neighbors, which means that a 1×1 convolution is not sufficient for a good approximation.

We tackle a classification task where the map between a given field configuration $X = \{k_{x,\mu}, l_{x,\mu}\}$ and its label $y(X)$:

$$y(X) = \begin{cases} 0 & \mathcal{F}_x = 0, \quad \forall x, \\ 1 & \text{else} \end{cases} \tag{2.5.2}$$

has to be found. The worm algorithm delineated in subsection 2.4.2 generates physical field configurations, for which the flux constraint $\mathcal{F}_x = 0$, $\forall x$ is respected. Relaxing this condition and allowing the existence of open worms, hence flux violations, is achieved by a modification of the original worm algorithm. This is done after an initial thermalization performed with the alternate update of the fields l and k. On top of the resulting configuration, a new worm is started and the configuration with the open worm is saved. The worm moves on the lattice and the new configuration replaces the previous one with probability $1/\ell$, where ℓ is the current worm length, until the worm closes. With this method, each open worm configuration has equal probability of being picked. [1] As also visible in Fig. 2.12, the flux conservation is violated at the two ends of the open worm. In this work, open worms will serve as labels in the tasks the neural networks deal with, but they have physical relevance on their own, since they enter the calculation of n-point functions of ϕ [107, 108].

2.5.1 Physical parameters, lattice sizes and dataset

The dataset consists of two classes, closed worm and open worm configurations. The first is obtained with the original worm algorithm, while the second one results from a distinct run with the modified version described earlier. Both datasets are characterized by the same values of the

---

[1]This can be proven by induction. Starting from a closed worm configuration, the head of the worm moves and the corresponding configuration is saved. After the next accepted movement (provided the worm stays open) the resulting configuration is stored with a probability of $1/2$, hence the previous configuration remains stored with a probability of $1/2$. After a third movement, the new configuration replaces the old one with a probability of $1/3$. The other two have a probability of $(1 - 1/3) = 2/3$ of remaining saved. Since they had the same probability before the last step, they will equally share the probability of $2/3$, meaning that each of the three possible configurations has the same probability of being stored. This can be generalized to an arbitrary number of movements.

physical parameters and lattice dimensions, specifically the coupling constant is fixed to $\lambda = 1$, the mass m can take three values corresponding to $\eta = 4 + m^2 \in \{4.01, 4.04, 4.25\}$, the chemical potential can be set to $\mu \in \{1, 1.25, 1.5\}$, and the lattices are square, with $N_t = N_x \in \{8, 16, 32, 64\}$. Only a subset of these possible values is used for the training set, namely the two combinations $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$ and the smallest lattice size, i.e. 8×8, with a total number of $N_{\text{train}} = 4\,000$ instances equally distributed among each class and parameter combination. The validation set is structured in the same way, but contains $N_{\text{val}} = 400$ samples. All combinations of parameters and lattice sizes are employed to generate the test set, and for each of them 100 samples pertain to closed worm configurations and another 100 pertain to open worm configurations, with a total of $7\,200$ data. The thermalization phase is taken care of by discarding data in the first $2\,000$ sweeps, after which measurements are saved every 100 sweeps. As in subsection 2.4.3, we checked the distribution of the input fields. The findings are very similar to what will be shown in the next task, where they are also discussed.

2.5.2 Architecture search, training and testing

We compare the performance of the three different architecture types reported in Fig. 2.1 on this task. To ensure a fair comparison, we again rely on *optuna* to perform a search for promising architectures based on validation loss, which is chosen to be the binary cross entropy loss

$$\mathcal{L} = -\frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} \left[y_i \log p(y_i) + (1 - y_i) \log(1 - p(y_i)) \right], \qquad (2.5.3)$$

where y_i denotes the class (closed or open worm) and p the probability of belonging to such a class. For every type, the maximum number of convolutions is $N_{\text{conv,max}} = 3$, all with a maximum kernel size of $K = 3$, $N_{\text{ch}} \in \{4, 8, 16, 32\}$ possible channels and equipped with circular padding. After each convolutional layer a *LeakyReLU* activation function is applied. Furthermore, every convolution except the last one can be followed by a pooling layer (average or max pooling) with a stride $s = 1$ in the case of EQ architectures and $s = 2$ in the case of ST and FL architectures. Non-equivariant architectures need to feature at least one pooling layer with $s = 2$. At the end of the convolutional part of the network, a global max pooling layer is used for EQ and ST architectures, while a flattening step characterizes the FL ones. Other global pooling layers can be employed, like average pooling or sum pooling, but given that the task deals with the detection of local defects, global max pooling is a more natural choice. Another option that is investigated in the search is to explicitly set the bias terms to zero in every convolutional layer. The resulting output is further

Table 2.6: Best architectures for the detection of flux violations. This table reports the architectures for each type resulting from the *optuna* searches. The input of every model are configurations reshaped as $(N_t, N_x, 4)$ tensors. The output is a binary classification probability. The row at the end indicates the number of learnable parameters for each architecture. The asterisk (*) signals layers where no bias is used. Table from [1].

EQ	ST	FL
Conv(2×2, 4, 32)	Conv*(2×2, 4, 16)	Conv*(3×3, 4, 8)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1, 32, 32)	MaxPool(2×2, 2)	MaxPool(2×2, 2)
LeakyReLU	Conv(1×1, 16, 16)	Conv(2×2, 8, 32)
GlobalMaxPool	LeakyReLU	LeakyReLU
Linear(32, 32)	Conv(1×1, 16, 8)	AvgPool(2×2, 2)
LeakyReLU	LeakyReLU	Conv(2×2, 32, 32)
Linear*(32, 1)	GlobalMaxPool	LeakyReLU
Sigmoid	Linear*(8, 32)	Flatten
	Linear(32, 1)	Linear*(128, 1)
	Sigmoid	Sigmoid
2657	953	5600

processed by a dense network with up to $N_{\text{dense,max}} = 2$ layers, each with $N_{\text{nodes}} \in \{4, 8, 16, 32\}$ possible nodes. Similarly to the convolutional part, also in the dense part of the architecture *LeakyReLU* follows each linear layer. A final linear layer leads to a single node, followed by a sigmoid activation function. The bias in each linear layer can be set to zero in the search.

For each architecture type, we conduct two *optuna* runs with 400 trials each. The combinations of hyperparameters that are picked are trained starting from five random initializations of the weights, in order to reduce the impact of random fluctuations in the hyperparameter search. The best-performing architecture of the two runs according to the validation loss is selected and trained 50 times with different parameter initializations to provide an ensemble of models.

The training process follows a similar prescription to the regression task in section 2.4. We employ the *AMSGrad* variant of the *AdamW* optimizer without weight decay, the learning rate is chosen to be $\lambda_{lr} = 10^{-3}$, the batch size 100 and the epochs are set to 200. Early stopping based on validation loss is used with a patience of 50.

In Table 2.6, the best architectures resulting from the *optuna* search for each type are displayed.

2.5.3 Results

The main test results are depicted in Figs. 2.13 and 2.14. In Fig. 2.13, we provide a comparison of the three architecture types evaluated on 8×8 lattices showing the dependence on the chemical potential. On average,
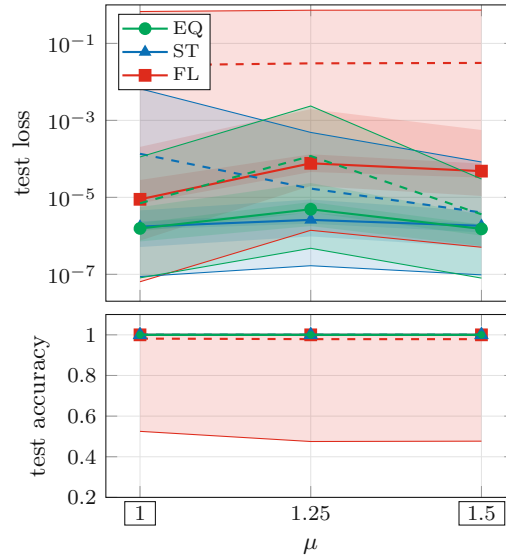
Figure 2.13: Test loss (top) and test accuracy (bottom) of the best EQ (green), ST (blue) and FL (red) architecture types over the chemical potential μ on the 8×8 lattice for the detection of flux violations. The training phase was carried out using data generated at $\mu = 1$ and 1.5 exclusively. The shaded regions indicate where the ensemble of all 50 randomly initialized models ranges, while the conventions for quantiles, symbols and lines are identical to the ones in Fig. 2.8. According to both metrics, the EQ and ST architectures achieve comparable performances and outclass the FL architecture. Image from [1].
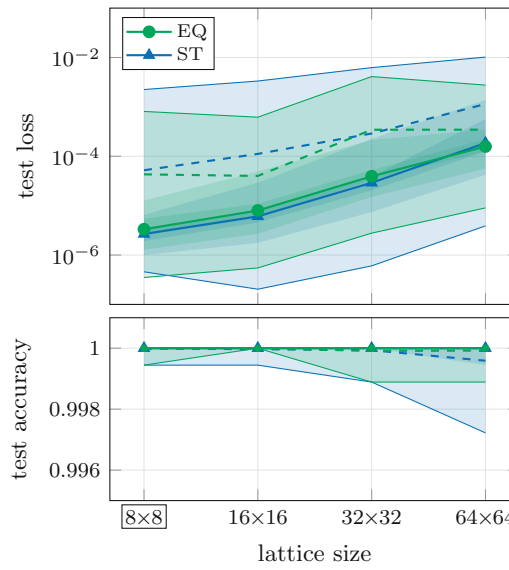
Figure 2.14: Test loss (top) and test accuracy (bottom) of the best EQ (green) and ST (blue) architecture types as functions of lattice size for the detection of flux violations. Training took place only on the 8×8 lattice. The colored bands and the solid and dashed lines have the same meaning as in Fig. 2.13. Both types of architecture demonstrate robust generalization across different lattice sizes, with a slightly reduced variance in the performance of the EQ models. Image from [1].

the EQ and ST models exhibit an excellent accuracy and a remarkable test loss, while the ensemble of FL models contains statistical outliers which contribute to worsen the average performance. Similar results are found when the test loss is plotted as a function of η. The comparison over all lattice sizes between EQ and ST types is plotted in Fig. 2.14, and shows that both architectures are able to effectively generalize to larger lattices, as well as to other physical parameters. FL models are excluded from this test, because no retraining has been run and therefore they could be applied to 8×8 lattices, where training happened. While the FL architecture scores worse results on average than the other two, the EQ and ST networks achieve a comparable performance, in contrast to the previous task. The breaking of translational symmetry caused by pooling layers with stride $s > 1$ does not ruin the performance of ST models in distinguishing closed from open worm configurations.

The results in Figs. 2.13 and 2.14 can be analyzed in more detail trying to understand and interpret how the EQ and ST models can reach such a high degree of accuracy. In order to do this, we inspect the feature maps of already trained EQ and ST models that appear in the convolutional part of the network, some examples of which are shown in Fig. 2.12 (b). We observe that in some of the channels of these feature maps flux violations in the proximity of one of the open ends of the worm are highlighted, as visible in Fig. 2.12 (a). Interestingly, it is sufficient for the network to detect a single defect to accurately predict the right class. This is reasonable if we consider that the networks did not have direct access to the local flux in Eq. (2.5.1) during training, but were provided with global information about whether a configuration features a flux violation.

Finally, we highlight that the number of weights of the best architectures found with *optuna* have an order of magnitude of 10^3, as reported in Table 2.6, which is rather small compared to the neural networks typically used.

2.6 Task III: counting flux violations

The task tackled in the previous section is generalized here to include more than one open worm, so that the networks need to solve the regression problem of finding a map between field configurations and the number of open worms present in it.

As discussed earlier, when an open worm is added, its endpoints show a flux violation, such that the flux in Eq. (2.5.1) is $\mathcal{F}_x = \pm 1$. If the algorithm for generating a new open worm is run on top of an open worm configuration, the second worm might cross the flux violations already present, where the Metropolis acceptance probability cannot be defined. Therefore, we realize a traffic-light system which forbids every new worm to move into a flux

violation. With this caveat, the algorithm for generating an open worm is repeated until the desired number of open worms is reached, then the resulting configuration is saved. Afterwards, we go back to the last configuration without open worms and perform 100 waiting sweeps before adding open worms again.

Since lattice sites that are characterized by a flux violation are not accessible to new open worms, the flux can take three values: 0, +1 and −1. This implies that a configuration with N_{worms} open worms features $2N_{\text{worms}}$ points where $\mathcal{F}_x = \pm 1$. Therefore, the function that needs to be approximated can be expressed as

$$y(X) = \frac{1}{2} \sum_x |\mathcal{F}_x| \,, \tag{2.6.1}$$

where X is a lattice configuration $\{k_{x,\mu}, l_{x,\mu}\}$. It is worth mentioning that this task can be viewed as a counting problem, such as crowd counting [109], although in a simplified version.

2.6.1 Physical parameters, lattice sizes and dataset

We keep using the same set of physical parameters and lattice sizes of the previous section, with the addition of a number of open worms $N_{\text{worms}} \in \{0, 1, \ldots, 10\}$. In order to investigate the generalization capabilities of the neural networks as in the other tasks, the training set consists of data generated at a small subset of all the possible combinations, namely $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$, the lattice size 8×8 (the same that were used in the classification problem), and a number of worms $N_{\text{worms}} \in \{0, 5\}$. The training samples are $N_{\text{train}} = 20\,000$ and are distributed equally between two different numbers of open worms and physical parameters. The validation set is structured analogously but consists of $N_{\text{val}} = N_{\text{train}}/10 = 2\,000$ samples. Every parameter combination is employed for the test set generation, with 100 samples each, leading to a total of $39\,600$ instances.

We now drop the lattice index x to use the notation k_t and k_x for the two fields that are involved in the computation of the flux, as indicated by Eq. (2.1.24). The distributions of such fields are illustrated in Figs. 2.15 and 2.16, respectively. We notice that for k_t the distribution is visibly different for training and testing configurations, even though the range of possible values is the same. The selection of the two (η, μ) pairs used for training is specifically designed to encompass the lower and higher values of k_t. This choice ensures that the domain covered is the same for training and testing. Each of the two peaks in the k_t training distribution in the top histogram in Fig. 2.15 is associated with one of the two (η, μ) pairs. The behavior of k_t is related to its coupling with the chemical potential, whereas k_x lacks this feature and in fact does not exhibit large differences between training and testing distributions.
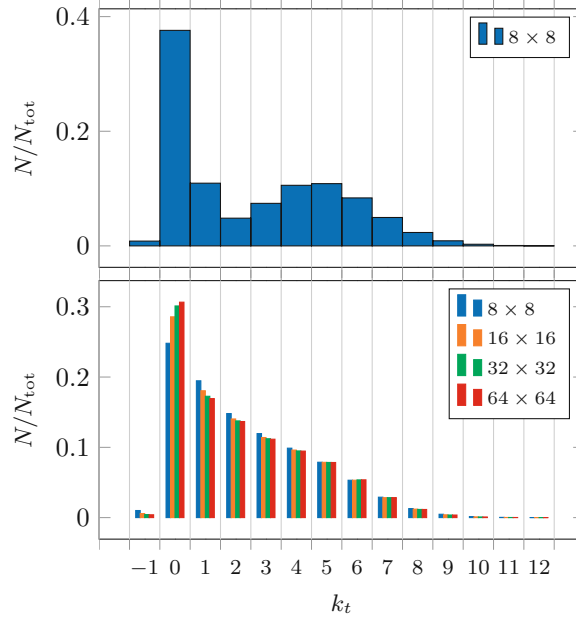
Figure 2.15: Distributions of the link field k_t. These two histograms feature the distributions of k_t in the training set (top) and in the test set (bottom). Image from [1].

2.6.2 Architecture search, training and testing

An initial exploration phase is conducted to examine trends with different hyperparameter choices, alongside an empirical check that a global sum pooling is the most favorable layer after the convolutional part of the networks. This is because the quantity that needs to be predicted is extensive, as discussed in Section 2.3. The insights gained during this preliminary stage are used to define the architecture search space of each of the three architecture types depicted in Fig. 2.1. To reduce bias in favor of a particular architecture type as much as possible, the search spaces are designed to be similar.

The search space for the EQ architecture involves $N_{\mathrm{conv}} \in \{2, 3, 4\}$ convolutional layers with a kernel size $K \in \{1, 2, 3\}$, followed by a global sum pooling layer leading to a dense network consisting of $N_{\mathrm{dense}} \in \{0, 1, 2\}$ layers. For the ST architecture, the search space is the same except for the insertion of $N_{\mathrm{pool}} \in \{1, 2\}$ spatial pooling layers with a stride of 2. Given that training takes place on 8×8 lattices, we limit the spatial pooling layers to a maximum of two, ensuring that the lattice is not reduced to only one site, as would happen with three spatial pooling layers, preserving the effectiveness of global sum pooling. The search space of the FL architecture type includes two mandatory convolutions with a kernel size $K \in \{2, 3\}$,
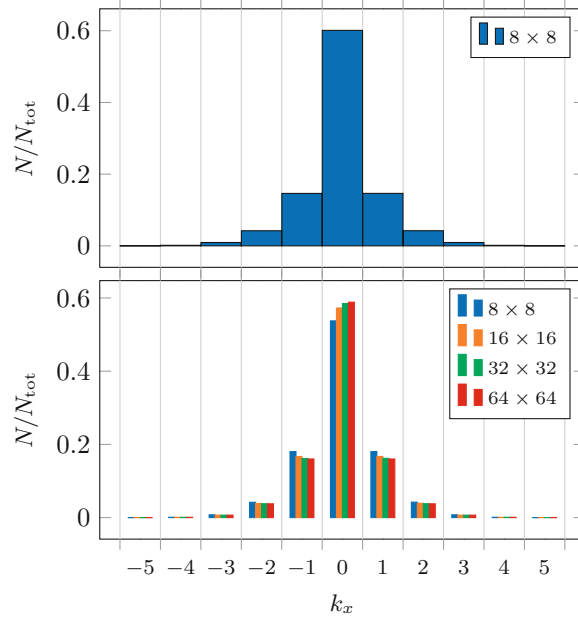
Figure 2.16: Distributions of the link field k_x. These two histograms feature the distributions of k_x in the training set (top) and in the test set (bottom). Image from [1].

and features a spatial pooling layer after each of them. An optional 1×1 convolution is inserted before and after each mandatory convolution, resulting in $N'_{\text{conv}} \in \{2, 3, 4, 5, 6\}$ convolutions in total. The convolutional part is followed by a flattening step and a dense network with $N'_{\text{dense}} \in \{1, 2, 3\}$ layers. The maximum number of layers is increased compared to the other two architecture types to compensate for the potential absence of 1×1 convolutions. For all search spaces, the channels in the convolutions and the nodes in the dense layers take the following values: $N_{\text{ch/nodes}} \in \{4, 8, 16, 32\}$.

Other hyperparameter choices are not part of the search space and are fixed based on the indications provided by the exploratory phase, with an outcome very similar to the other tasks. Circular padding is employed in all convolutions to take care of the periodicity of the lattice. We choose *LeakyReLU* as activation function throughout the network, and apply it after every convolution and every linear layer except the last one. The convolutions and the linear layers do not have a bias term.

It is worth mentioning that another *optuna* search was carried out for the EQ type with the same structure we have described for ST apart from having a stride $s = 1$ instead of $s = 2$. However, none of the models suggested in this run outperform the EQ models found in the initial search.

In order to evaluate the network performances, we make use of two metrics similarly to the previous task, the MSE loss and the accuracy. For the

Table 2.7: Best architectures for counting flux violations. This table reports the three most promising architectures obtained with an *optuna* search, ordered according to the average validation loss over 20 instances trained from scratch. As in the previous tasks, the input tensor shape is $(N_t, N_x, 4)$, while the output is a real number that approximates the amount of open worms in the configuration. The final row informs about the number of learnable parameters for each architecture. Table from [1].

1st EQ	**2nd EQ**	**3rd EQ**
Conv(1×1, 4, 32)	Conv(2×2, 4, 8)	Conv(1×1, 4, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(2×2, 32, 8)	Conv(2×2, 8, 8)	Conv(2×2, 4, 8)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(2×2, 8, 16)	Conv(1×1, 8, 4)	Conv(2×2, 8, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1, 16, 8)	Conv(1×1, 4, 8)	Conv(3×3, 4, 1)
LeakyReLU	LeakyReLU	LeakyReLU
GlobalSumPool	GlobalSumPool	GlobalSumPool
Linear(8, 1)	Linear(8, 1)	
1800	456	308

1st ST	**2nd ST**	**3rd ST**
Conv(2×2, 4, 16)	Conv(2×2, 4, 4)	Conv(2×2, 4, 4)
LeakyReLU	LeakyReLU	LeakyReLU
Conv(1×1, 16, 32)	MaxPool(2×2, 2)	AvgPool(2×2, 2)
LeakyReLU	Conv(2×2, 4, 4)	Conv(3×3, 4, 16)
Conv(1×1, 32, 32)	LeakyReLU	LeakyReLU
LeakyReLU	GlobalSumPool	GlobalSumPool
AvgPool(2×2, 2)	Linear(4, 1)	Linear(16, 32)
Conv(1×1, 32, 8)		LeakyReLU
LeakyReLU		Linear(32, 1)
GlobalSumPool		
Linear(8, 32)		
LeakyReLU		
Linear(32, 1)		
2336	132	1184

1st FL	**2nd FL**	**3rd FL**
Conv(2×2, 4, 4)	Conv(2×2, 4, 8)	Conv(2×2, 4, 32)
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool(2×2, 2)	AvgPool(2×2, 2)	AvgPool(2×2, 2)
Conv(3×3, 4, 8)	Conv(3×3, 8, 4)	Conv(3×3, 32, 4)
LeakyReLU	LeakyReLU	LeakyReLU
AvgPool(2×2, 2)	AvgPool(2×2, 2)	AvgPool(2×2, 2)
Flattening	Flattening	Flattening
Linear(8, 4)	Linear(4, 4)	Linear(4, 32)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(4, 32)	Linear(4, 32)	Linear(32, 16)
LeakyReLU	LeakyReLU	LeakyReLU
Linear(32, 1)	Linear(32, 1)	Linear(16, 1)
640	640	2704

latter, the predictions are rounded to the closest integer. During the optimization process, the monitored metric is the validation loss. Two *optuna* runs are executed with the aim of mitigating the risk of overlooking promising combinations of hyperparameters. For each of these, three models are trained for 200 epochs without early stopping to reduce the impact of random initializations. We used a batch size of 16, a learning rate $\lambda_{lr} = 10^{-3}$ and the *AMSGrad* variant of the *AdamW* optimizer with zero weight decay.

From the set of 100 different architectures resulting from the two *optuna* searches, we identify the top three for each architecture type based on the average validation loss across their three initializations. We train these architectures from 20 different random initializations using the same training and validation sets, and keeping the same hyperparameters, except for the amount of epochs which is increased to 500. We keep track of the validation loss and checkpoint the weight values of the best model. For each type, the architectures are sorted according to the average of the validation loss over the ensemble of models. The results are detailed in Table 2.7.

2.6.3   Results

We divide the results in a test performed on the subset containing 8×8 configurations and another which includes the whole test set. The former test is executed for all the architecture types and its results are shown in Fig. 2.17, while the FL architecture is excluded from the latter, whose results are depicted in Fig. 2.18.

The plots indicate that the EQ architecture consistently outperforms its non-equivariant counterparts. An interesting observation comes from Fig. 2.17, where ST and even more so FL models struggle particularly with the prediction of one, two and three of open worms. The reason for this can be associated to the fact that the training set includes samples with only $N_{\text{worms}} \in \{0, 5\}$, therefore the same architectures have also been trained on a dataset that features $N_{\text{worms}} \in \{0, 1, 5\}$. Surprisingly, the outcome is similar to the one reported in Fig. 2.17, suggesting that the problem for non-equivariant architectures is not simply a lack of sufficient information about the problem.

In table 2.8 the mean and median losses of the model ensemble on the validation and test sets are displayed for the three types. The best architecture for each metric and type is highlighted in bold. We notice that depending on whether we choose the mean or the median of the validation loss to select the best architecture we get a different outcome. An even more relevant observation is related to the generalization capabilities of the networks, specifically to the fact that performing well on the validation set does not guarantee that the networks achieve good results when testing. For this reason, we analyze the relationship between the validation and the test losses for every individual model of each type in Figs. 2.19 and 2.20.
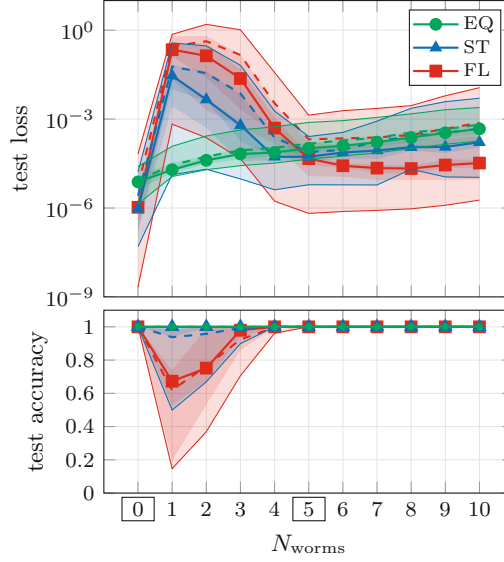
Figure 2.17: Test loss (top) and test accuracy (bottom) of the architectures which scored the lowest mean of the validation loss, evaluated on 8×8 configurations given as functions of the number of open worms. Training and validation are executed with $N_{\text{worms}} = 0$ and $N_{\text{worms}} = 5$, while the test set comprises a number of open worms in the interval $N_{\text{worms}} \in [0, 10]$. Conventions for shaded regions, quantiles, symbols, solid and dashed lines are the same as in Fig. 2.8 except that the number of instances used to create a model ensemble is 20. While the accuracy of every EQ model is perfect and its loss increases slightly with the number of open worms, the non-equivariant models fail to accurately predict a small number of flux violations. Image from [1].

Table 2.8: Metrics of the best architectures for counting flux violations. For each type and metric, the result of the best architecture is highlighted in bold. Table from [1].

	validation loss on 8×8		test loss on 8×8		test loss up to 64×64	
	mean	median	mean	median	mean	median
1st EQ	$\mathbf{4.676 \times 10^{-5}}$	4.137×10^{-5}	$\mathbf{2.108 \times 10^{-4}}$	1.483×10^{-4}	$\mathbf{1.008 \times 10^{-3}}$	8.308×10^{-4}
2nd EQ	1.042×10^{-4}	$\mathbf{2.440 \times 10^{-5}}$	3.525×10^{-4}	$\mathbf{8.783 \times 10^{-5}}$	1.807×10^{-3}	$\mathbf{7.936 \times 10^{-4}}$
3rd EQ	8.992×10^{-3}	3.072×10^{-4}	2.105×10^{-2}	9.163×10^{-4}	1.925	4.031×10^{-2}
1st ST	$\mathbf{2.331 \times 10^{-5}}$	2.173×10^{-5}	9.438×10^{-3}	3.576×10^{-3}	4.446	3.026
2nd ST	8.479×10^{-5}	4.372×10^{-5}	$\mathbf{2.545 \times 10^{-4}}$	$\mathbf{9.340 \times 10^{-5}}$	$\mathbf{3.738 \times 10^{-3}}$	$\mathbf{1.171 \times 10^{-3}}$
3rd ST	2.869×10^{-4}	$\mathbf{2.171 \times 10^{-5}}$	1.676×10^{-2}	1.381×10^{-3}	2.943	9.580×10^{-1}
1st FL	$\mathbf{2.602 \times 10^{-5}}$	1.787×10^{-5}	7.837×10^{-2}	3.817×10^{-2}	-	-
2nd FL	4.004×10^{-5}	1.117×10^{-5}	$\mathbf{5.300 \times 10^{-2}}$	$\mathbf{1.285 \times 10^{-3}}$	-	-
3rd FL	5.805×10^{-5}	$\mathbf{1.031 \times 10^{-5}}$	6.382×10^{-2}	3.556×10^{-2}	-	-
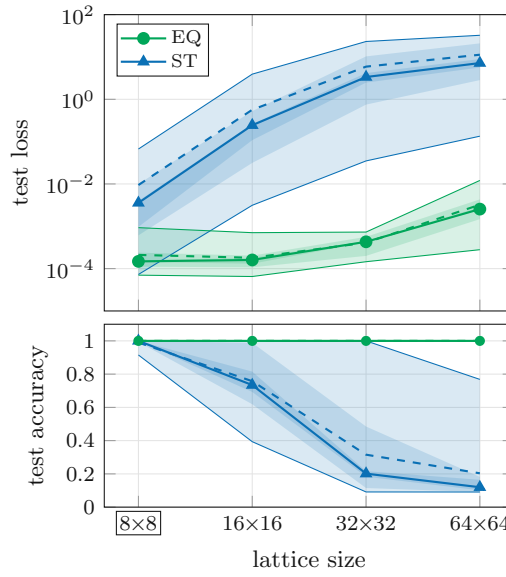
Figure 2.18: Test loss (top) and test accuracy (bottom) of the architectures which scored the lowest mean of the validation loss, evaluated on any number of worms in the range $N_{\text{worms}} \in [0, 10]$ given as functions of the lattice size. Training and validation are executed on the smallest lattice (8×8), while testing takes place on all lattice sizes. Except for a few ST models accomplishing good results on 8×8 configurations, the EQ architecture outperforms its non-equivariant counterpart across all lattice sizes. Image from [1].
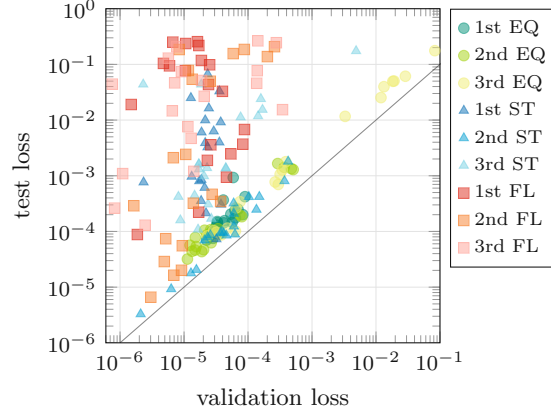
Figure 2.19: Test loss on the 8×8 lattice versus validation loss for every instance of each architecture. The test loss is shown for all 20 instances of the winning architectures displayed in table 2.7. Training and validation were executed with $N_{\text{worms}} \in \{0, 5\}$ and $(\eta, \mu) \in \{(4.01, 1.5), (4.25, 1)\}$ on 8×8 configurations. The test set included other physical parameter values: $N_{\text{worms}} \in [0, 10]$, $\mu \in \{1.0, 1.25, 1.5\}$, $\eta \in \{4.01, 4.04, 4.25\}$. The diagonal black line denotes where test loss and validation loss are equal. If a model has good generalization properties, the test loss has to be comparable with the validation loss, and therefore has to lie close to the black line, which happens for most of the EQ models (green circles). Image from [1].
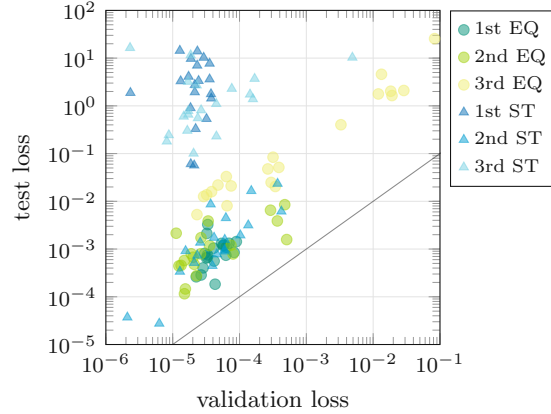
Figure 2.20: Test loss over the whole test set versus validation loss of every instance for EQ and ST architectures. We analyze our models in a similar fashion to Fig. 2.19, extending the study to lattice sizes up to 64×64. EQ models (green circles) tend to be closer to the black line where test loss and validation loss are the same, showing their ability of generalizing to different lattice sizes and physical parameters. Image from [1].

Most ST and FL models are scattered vertically, meaning that for many of them the test loss is much worse than the validation loss, which is a sign of generalization issues. EQ models, on the other hand, are distributed closer to the black line, where test and validation loss would match. These results suggest that if an EQ architecture scores a low validation loss we can expect that the test loss is also low, which means that it generalized more reliably than the non-equivariant counterparts.

Remarkably, a non-equivariant architecture also exhibits this behavior, namely the 2nd ST architecture. Its best two instances even achieve results better by almost an order of magnitude compared to the best EQ models both when validating and testing. One factor that may have contributed to its success is the presence of a spatial max pooling layer, which can be useful for detecting local defects, as mentioned also in the previous task. Another possible reason is the very small number of parameters compared to all the architectures in table 2.7, given the relative simplicity of this counting problem. This is confirmed also by the fact that *optuna* favors in general small architectures for this task, as happened for the previous ones, with a number of weights between ~ 100 and $\sim 3\,000$.

Chapter 3

Gauge-symmetric neural networks in lattice gauge theory

The results found in the previous Chapter have shown that incorporating translational symmetry in a CNN architecture gives a boost to the performance in a problem symmetric under translations. This serves as an additional motivation for the scope of this Chapter: designing neural networks that respect gauge symmetries by construction. After initially reviewing Yang-Mills theory and its discretization on the lattice, we define equivariance for lattice gauge theory. Layers such as convolutions, that are readily available in common frameworks, break gauge equivariance, therefore we introduce appropriate layers that possess such a property. The architectures resulting from stacking these layers respect gauge symmetry by construction, and they can in principle predict arbitrarily sized Wilson or Polyakov loops, making these networks capable of approximating any function on the lattice. We then show how these neural networks perform on regression tasks involving e.g. Wilson loops of various sizes in 1+1D and in 3+1D. In the former case, we compare their results with standard CNNs that do not respect gauge symmetry.

3.1 Yang-Mills theory and lattice gauge theory

In this section, we introduce Yang-Mills theory and lattice gauge theory with particular focus on gauge symmetry. A broader introduction can be found in standard textbooks [110, 111].

Let us consider the $SU(N_c)$ Yang-Mills theory at finite temperature T in $D+1$ dimensions. The expectation value of an observables $\mathcal{O}$ is calculated

by means of the path integral

$$\langle \mathcal{O} \rangle = \frac{1}{\mathcal{Z}} \int \mathcal{D}[A] \, \mathcal{O}(A_\mu) e^{-S_\mathrm{E}[A_\mu]}, \tag{3.1.1}$$

with the partition function

$$\mathcal{Z} = \int \mathcal{D}[A] \, e^{-S_\mathrm{E}[A_\mu]} \tag{3.1.2}$$

acting as a normalization constant, and the Euclidean Yang-Mills action

$$S_\mathrm{E}[A_\mu] = \frac{1}{2g^2} \int\limits_0^\beta d\tau \int d^D x \, \mathrm{Tr}\left[F_{\mu\nu}(x)F_{\mu\nu}(x)\right]. \tag{3.1.3}$$

The inverse temperature of the system is $\beta = T^{-1}$ and $g \in \mathbb{R}$ is the Yang-Mills coupling constant. The non-Abelian field strength tensor reads

$$F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu + i\left[A_\mu, A_\nu\right], \tag{3.1.4}$$

where $A_\mu(x)$ is a gauge field living in the algebra $\mathfrak{su}(N_c)$. We write the $(D+1)$-dimensional spacetime point as $x = (\tau, \mathbf{x})$. The D spatial dimensions spanned by $\mathbf{x}$ are real, while the imaginary time dimension τ extends from 0 to the inverse temperature, $\tau \in [0, \beta)$. The metric on the spacetime is Euclidean: $\eta_{\mu\nu} = \mathrm{diag}(+, +, \ldots, +)$. The gauge fields satisfy periodic boundary conditions along τ, i.e. $A_\mu(\tau + \beta, \mathbf{x}) = A_\mu(\tau, \mathbf{x})$.

A characteristic property of a gauge theory is gauge invariance, which refers to the fact that observables are invariant under local symmetry transformations. For Yang-Mills theory, a gauge transformation T_Ω applied to the gauge field $A_\mu(x)$ can be written as

$$T_\Omega A_\mu(x) = \Omega(x)\left(A_\mu(x) - i\partial_\mu\right)\Omega^\dagger(x), \tag{3.1.5}$$

where $\Omega(x)$ is an element of the $\mathrm{SU}(N_c)$ group, hence it is unitary and has unit determinant:

$$\Omega(x)\Omega^\dagger(x) = \mathbf{1}, \tag{3.1.6}$$
$$\det \Omega(x) = 1. \tag{3.1.7}$$

The application of this transformation to the field strength tensor yields

$$T_\Omega F_{\mu\nu}(x) = \Omega(x)F_{\mu\nu}(x)\Omega^\dagger(x), \tag{3.1.8}$$

where extra terms coming from the derivatives of the gauge field simplify exactly with the extra terms coming from the commutator. Given the cyclic property of the trace, the integrand in Eq. (3.1.3) is invariant under

gauge transformations, thus making the Yang-Mills action itself invariant under (3.1.5):

$$S_{\mathrm{E}}[T_\Omega A_\mu] = S_{\mathrm{E}}[A_\mu]. \tag{3.1.9}$$

As a consequence, the set of gauge fields $\{T_\Omega A_\mu\}$ that are related via arbitrary gauge transformations forms a class of physically equivalent fields. Moreover, any physical observable $\mathcal{O}(A_\mu)$ must be invariant under gauge transformations, which guarantees that also its expectation value Eq. (3.1.1) has the same property.

Apart from perturbation theory, where the coupling constant g is small, an infinite dimensional functional integral of the form of Eq. (3.1.1) is generally intractable. As in the scalar field theory case, we can tackle the problem by approximating the continuous spacetime with a lattice

$$\Lambda = \left\{ x = \sum_{\mu=0}^{D} a^\mu n^\mu \hat{\mu} \,|\, n^\mu \in \mathbb{Z} \right\}, \tag{3.1.10}$$

where a^μ are the spacings between the lattice sites in the direction $\hat{\mu}$. In the following, we set the lattice constant a^μ to unity for every direction. A naive discretization of the gauge fields fails to respect gauge symmetry. The regularization on the lattice circumvents this problem by replacing the gauge fields $A_\mu(x)$ with

$$U_{x,\mu} = \mathcal{P} \exp\left(ig \int_x^y \mathrm{d}z^\nu A_\nu(z) \right) \in \mathrm{SU}(N_c), \tag{3.1.11}$$

where $\mathcal{P}$ indicates the path ordering scheme. These quantities are defined along the edges of the lattice Λ that connect the lattice site x and its neighbor $y = x + \hat{\mu}$, hence the name gauge links. The definition above can be formally expressed by dividing the path into N infinitesimal segments $(z_n, z_n + \mathrm{d}z_n)$, $n \in \{0, 1, \ldots, N-1\}$ in the following way:

$$\begin{aligned}
U_{x,\mu} &= \lim_{N\to\infty} \exp\left[-i \int_{z_{N-1}}^y \mathrm{d}z^\nu_{N-1} A_\nu(z) \right] \ldots \exp\left[-i \int_{z_n}^{z_{n+1}} \mathrm{d}z^\nu_n A_\nu(z) \right] \\
&\quad \ldots \exp\left[-i \int_x^{z_1} \mathrm{d}z^\nu_0 A_\nu(z) \right] \\
&= \lim_{N\to\infty} [1 - i\,\mathrm{d}z^\nu_0 A_\nu(z_0)] \ldots [1 - i\,\mathrm{d}z^\nu_{N-1} A_\nu(z_{N-1})] + O(\mathrm{d}z^2), \tag{3.1.12}
\end{aligned}$$

where $z_0 = x$ and $z_N = y$. If we apply the transformation Eq. (3.1.5) at the point z_n, we find

$$\begin{aligned}
1 - i\,\mathrm{d}z^\nu_n A'_\nu(z_n) &= \Omega(z_n)\Omega^\dagger(z_n) - i\mathrm{d}z^\nu_n \Omega(z_n)\left[A_\nu(z_n)\Omega^\dagger(z_n) + i\,\partial_\nu\Omega^\dagger(z_n) \right] \\
&= \Omega(z_n)\left[1 - i\,\mathrm{d}z^\nu_n A_\nu(z_n) \right]\Omega^\dagger(z_{n+1}) + O(\mathrm{d}z^2). \tag{3.1.13}
\end{aligned}$$

The derivative of Ω is defined as in Eq. (2.1.13). Using this transformation rule for each z_n, the terms involving $\Omega(z_n)$ simplify except at the endpoints, thus yielding the gauge transformation of the link

$$T_\Omega U_{x,\mu} = \Omega_x U_{x,\mu} \Omega^\dagger_{x+\mu}, \tag{3.1.14}$$

where Ω_x indicates the transformation at lattice site x and $x + \mu$ is a short-hand for $x + \hat{\mu}$. In terms of these new variables, the Yang-Mills action can be approximated by the Wilson action

$$S_W[U] = \frac{2}{g^2} \sum_{x \in \Lambda} \sum_{\mu < \nu} \mathrm{Tr}\left[\mathbf{1} - U_{x,\mu\nu}\right], \tag{3.1.15}$$

where the so-called plaquette variables

$$U_{x,\mu\nu} = U_{x,\mu} U_{x+\mu,\nu} U^\dagger_{x+\nu,\mu} U^\dagger_{x,\nu} = \begin{array}{c} \fbox{} \end{array} \tag{3.1.16}$$

are 1×1 Wilson loops on the lattice. We made use of the definition of a link pointing in the negative direction $-\rho$ as $U_{y,-\rho} = U^{-1}_{y-\rho,\rho} = U^\dagger_{y-\rho,\rho}$. Here-after, Wilson loops are intended to be untraced matrices, unless specified otherwise. The plaquette variables transform locally at x under the effect of Eq. (3.1.14), i.e.

$$T_\Omega U_{x,\mu\nu} = \Omega_x U_{x,\mu\nu} \Omega^\dagger_x, \tag{3.1.17}$$

which implies that the Wilson action is invariant:

$$S_W[T_\Omega U] = S_W[U]. \tag{3.1.18}$$

From the Wilson action (3.1.15), it is possible to recover the Yang-Mills action (3.1.3) by expanding $U_{x,\mu}$ in powers of $A_\mu(x)$ with lattice artifacts of order $O(a^2)$, where a denotes the lattice spacing. Lattice gauge theory therefore represents the appropriate formalism to describe gauge fields on a lattice while respecting gauge symmetry. Similarly to the continuum case, expectation values are given by

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathcal{D}[U]\, \mathcal{O}[U]\, e^{-S_W[U]}, \tag{3.1.19}$$

$$Z = \int \mathcal{D}[U]\, e^{-S_W[U]}, \tag{3.1.20}$$

where $\mathcal{D}[U]$ is the Haar measure and Z is the partition function of the new degrees of freedom. With the lattice formalism, the infinite-dimensional functional integrals are substituted by finite (although high) dimensional integrals. Furthermore, we restrict the lattice to a finite sublattice charac-terized by a number of

$$N_{\mathrm{lat}} = N_t \cdot N_s^D \tag{3.1.21}$$

lattice sites, with N_t cells in the imaginary time direction and N_s cells in the spatial directions. The lattice is also equipped with periodic boundary conditions in every dimension. This enables the computation of expectation values via Monte Carlo method, i.e. we generate a series of configurations $U^{(n)}$ according to the probability density $P[U] \propto \exp(-S_W[U])$, and approximate the expectation value by

$$\langle \mathcal{O} \rangle \approx \frac{1}{N_{\text{conf}}} \sum_{n=1}^{N_{\text{conf}}} \mathcal{O}[U^{(n)}]. \tag{3.1.22}$$

Gauge field configurations can be generated for example with the Metropolis algorithm [112], which is discussed in Appendix A for the SU(2) case, or using the heat bath method [113].

Physical observables are typically expressed in terms of (traced) Wilson loops. The simplest quantity is the Wilson action (3.1.15), which is a function of 1×1 loops. It is possible to improve the approximation of the continuum action by including larger loops [114–118]. Notably, the potential of a static quark pair can be determined from the expectation value of a Wilson loop with a large extent in the temporal direction [119,120].

A local observable that is typically studied is the topological charge density [121]. In the continuum, its definition is

$$q(x) = \frac{1}{32\pi^2} \epsilon_{\mu\nu\rho\sigma} \text{Tr}\left[F_{\mu\nu}(x) F_{\rho\sigma}(x)\right], \tag{3.1.23}$$

which on the lattice can be approximated by products of Wilson loops. The plaquette approximation of Eq. (3.1.23) reads

$$q_x^{\text{plaq}} = \frac{\epsilon_{\mu\nu\rho\sigma}}{32\pi^2} \text{Tr}\left[\frac{U_{x,\mu\nu} - U_{x,\mu\nu}^\dagger}{2i} \frac{U_{x,\rho\sigma} - U_{x,\rho\sigma}^\dagger}{2i}\right]. \tag{3.1.24}$$

The discretization errors of the plaquette approximation can be reduced with the inclusion of appropriate larger loops in the discretization of the field strength tensor [122]. Other observables, such as the energy momentum tensor [123], can be formulated as (non-linear) combinations of arbitrarily-sized Wilson loops.

There exist also non-local observables, which can be written in terms of Polyakov loops [124], that wrap around the periodic boundary of the lattice. The trace of the Polyakov loop can serve as an order parameter for quark confinement in a pure gauge theory at nonzero temperature, and correlators of the traced Polyakov loop are related to the static quark potential [125, 126]. Wilson loops and Polyakov loops are two topologically distinct objects, as can be seen for example by noticing that the former can be contracted to a single point, while the latter cannot.

In summary, all the aforementioned observables are in general non-linear functions either of Wilson or Polyakov loops. In the following section, we

will introduce new layers of artificial neural networks capable of expressing such functions while preserving gauge symmetry on the lattice.

In the next section, we will introduce new layers of artificial neural networks that can express such lattice gauge equivariant functions and respect gauge symmetry on the lattice.

3.2 Gauge equivariant layers

Lattice gauge equivariant convolutional neural networks (L-CNNs) can approximate a large class of gauge equivariant functions while preserving the symmetry properties of the underlying theory. Similar to how we introduced CNNs in the first chapter, we split L-CNNs into more elementary layers and discuss each of them in this section.

3.2.1 Input data

The input data of each layer is a tuple $(\mathcal{U}, \mathcal{W})$, consisting of non-locally transforming gauge link variables $\mathcal{U}$ and locally transforming objects $\mathcal{W}$. A natural choice for the first part of the tuple is the set of gauge link variables $\mathcal{U} = \{U_{x,\mu}\}$, whose transformation rule we remind to be Eq. (3.1.14). Concretely, we choose the fundamental representation of $\mathrm{SU}(N_c)$, in which links are treated as complex special unitary $N_c \times N_c$ matrices. The second part is a set of variables $\mathcal{W} = \{W_{x,i}\}$ with $W_{x,i} \in \mathbb{C}^{N_c \times N_c}$ and index $i \in \{1, 2, \ldots, N_{\mathrm{ch}}\}$, which can be interpreted as channels. The requirement for these input variables is to respect the following transformation rule:

$$T_\Omega W_{x,i} = \Omega_x W_{x,i} \Omega_x^\dagger. \tag{3.2.1}$$

We emphasize that these objects transform locally. A possible choice for $\mathcal{W}$ is the set of the Wilson loops $U_{x,\mu\nu}$ with positive orientation, i.e. $\mu < \nu$.

We specify that in practice we represent both $\mathcal{U}$ and $\mathcal{W}$ as $N \times N$ complex matrices, even though the links can be described with less degrees of freedom since they are constrained by the unitarity and specialness conditions.

3.2.2 Gauge equivariance

Having established what the input data for L-CNNs is, we can now define what we mean exactly by gauge equivariance and gauge invariance. A function f taking as input the tuple $(\mathcal{U}, \mathcal{W})$ is gauge equivariant (or covariant) if

$$f(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = T'_\Omega f(\mathcal{U}, \mathcal{W}), \tag{3.2.2}$$

for a generic $\Omega \in \mathrm{SU}(N_c)$, with $T'_\Omega f$ denoting the effect of the gauge transformation on the function f. As for the other definition, a function f of the

tuple $(\mathcal{U}, \mathcal{W})$ is gauge invariant if

$$f(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = f(\mathcal{U}, \mathcal{W}). \tag{3.2.3}$$

This can be viewed as a special case of equivariance with $T'_\Omega = \mathbf{1}$.

An L-CNN can be either an equivariant function as in Eq. (3.2.2) or an invariant function as in Eq. (3.2.3). This means that such networks are suited to approximate these types of functions, which notably include physical observables, that are gauge invariant.

3.2.3 Lattice gauge equivariant convolutions

A fundamental component of neural networks is represented by linear layers. A possible realization of such a layer for lattice gauge theory applications is

$$L_x(\mathcal{W}) = \sum_{y \in \Lambda} \omega_{x,y} W_y, \tag{3.2.4}$$

where $\omega_{x,y}$ are complex weights and x and y are lattice sites in Λ. No bias term has been included and the channel index in $\mathcal{W}$ has been dropped to simplify the notation.

This layer does not meet our requirements to be part of an L-CNN, as it does not respect gauge equivariance. In fact, objects that transform locally at different lattice sites are added up, therefore $L_x(\mathcal{W})$ does not transform according to Eq. (3.2.1) under arbitrary gauge transformations:

$$L_x(T_\Omega \mathcal{W}) = \sum_y \omega_{x,y} \, \Omega_y W_y \Omega_y^\dagger \neq \Omega_x L_x(\mathcal{W}) \Omega_x^\dagger.$$

In order to maintain a similar structure to a conventional linear layer and concurrently enabling the right transformation, we can modify Eq. (3.2.4) introducing the parallel transporter $U_{x \to y}$ in the following way:

$$L'_x(\mathcal{U}, \mathcal{W}) = \sum_y \omega_{x,y} \, U_{x \to y} W_y U_{x \to y}^\dagger. \tag{3.2.5}$$

The parallel transporter transforms according to

$$T_\Omega U_{x \to y} = \Omega_x U_{x \to y} \Omega_y^\dagger, \tag{3.2.6}$$

which guarantees that the definition of the linear layer in Eq. (3.2.5) is equivariant:

$$L'_x(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = \Omega_x L'_x(\mathcal{U}, \mathcal{W}) \Omega_x^\dagger = T_\Omega L'_x(\mathcal{U}, \mathcal{W}). \tag{3.2.7}$$

The parallel transporter can be built by taking the product of the sequence of links connecting x and y. However, no particular path has been

specified, which makes the gauge equivariant linear layer path dependent in general. A natural choice that can be made in the continuum is to set the path to be the geodesic (that is the shortest path) between x and y, reminiscent of the formulation of gauge equivariant neural networks on manifolds [79]. On the lattice, a unique geodesic cannot be introduced unless two points lie on the same lattice axis, therefore we restrict the path to straight lines along the lattice axes. This practical solution corresponds to imposing $\omega_{x,y} = 0$ if x and y cannot be linked moving along a single axis.

Lattice gauge theory is also symmetric under spacetime translations, since they leave the Wilson action (3.1.15) unaltered. Given the results of the study in Chapter 2, we equip the equivariant linear layers in Eq. (3.2.5) with translational equivariance by requiring the weights to satisfy translational invariance, i.e.

$$\omega_{x+s,y+s} = \omega_{x,y}, \tag{3.2.8}$$

where s denotes a translation on the lattice. The gauge equivariant linear layer becomes a gauge equivariant convolution. As a consequence of Eq. (3.2.8), convolutions are equivariant under translations. We define the shifted input for an arbitrary element and call t the element of the translation group corresponding to the shift vector s. Its action on links and loops is

$$tW_x \equiv W_{x+s}, \tag{3.2.9}$$
$$tU_{x,\mu} \equiv U_{x+s,\mu}. \tag{3.2.10}$$

Translational symmetry is preserved, as can be shown with the following:

$$
\begin{aligned}
L'_x(t\mathcal{U}, t\mathcal{W}) &= \sum_{y\in\Lambda} \omega_{x,y} U_{x+s\to y+s} W_{y+s} U^\dagger_{x+s\to y+s} \\
&= \sum_{y'\in\Lambda} \omega_{x+s,y'} U_{x+s\to y'} W_{y'} U^\dagger_{x+s\to y'} \\
&= L'_{x+s}(\mathcal{U}, \mathcal{W}) = tL'_x(\mathcal{U}, \mathcal{W}),
\end{aligned}
\tag{3.2.11}
$$

where we made use of Eq. (3.2.8) in the second equality. A linear layer with the property of translational equivariance is called convolution, therefore we define a gauge equivariant convolutional layer as

$$C_x(\mathcal{U}, \mathcal{W}) = \sum_s \omega_s\, U_{x\to x+s} W_{x+s} U^\dagger_{x\to x+s}, \tag{3.2.12}$$

where we adopted a more compact notation with weight parameters $\omega_s \in \mathbb{C}$. In this form, it is more manifest that this operation is translational equivariant.

For general applications, it is necessary to take into account a generalization of such a convolutional layer to an arbitrary number of input and

output channels. We name it lattice gauge equivariant convolution (L-Conv) and write it as

$$\mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}) = \sum_{j,\mu,k} \omega_{i,j,\mu,k} U_{x,k\cdot\mu} W_{x+k\cdot\mu,j} U_{x,k\cdot\mu}^\dagger, \tag{3.2.13}$$

where $\omega_{i,j,\mu,k}$ are the complex learnable parameters of the convolution, and the indices run in the following intervals: $1 \leq i \leq N_{\text{ch,out}}$, $1 \leq j \leq N_{\text{ch,in}}$, $0 \leq \mu \leq D$ and $-K \leq k \leq K$, with K being the kernel size. Note that with this choice the range of values for k is such that the convolution is symmetric around the center of the kernel. In our experiments, we restrict ourselves to positive shifts along the lattice axes, therefore $0 \leq k \leq K$.

The unique parallel transporters along the lattice axes read

$$U_{x,k\cdot\mu} = \begin{cases} U_{x,\mu} U_{x+\mu,\mu} \cdots U_{x+(k-1)\mu,\mu}, & k > 0, \\ U_{x,-\mu} U_{x-\mu,-\mu} \cdots U_{x-(k-1)\mu,-\mu}, & k < 0. \end{cases} \tag{3.2.14}$$

With the same arguments used for the equivariant linear layers, we can show that the convolution (3.2.13) is gauge equivariant

$$\mathcal{C}_{x,i}(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = \Omega_x \mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}) \Omega_x^\dagger = T_\Omega \mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}) \tag{3.2.15}$$

and transforms locally at x. Focusing on the effect on the data $(\mathcal{U}, \mathcal{W})$, the L-Conv operation modifies the $\mathcal{W}$ variables while the links remain the same:

$$\mathcal{C} : (\mathcal{U}, \mathcal{W}) \to (\mathcal{U}, \mathcal{W}') \tag{3.2.16}$$

with

$$\mathcal{W}' = \left\{ W'_{x,i} = \mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}), \, 1 \leq i \leq N_{\text{ch,out}}, \, x \in \Lambda \right\}. \tag{3.2.17}$$

We initially disregarded the bias term for the linear layer (3.2.4) for simplicity, but it is possible to include it to provide an even more general version of L-Conv. The addition of a bias $\omega_0 \in \mathbb{C}$ as in

$$\mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}) = \sum_{j,\mu,k} \omega_{i,j,\mu,k} U_{x,k\cdot\mu} W_{x+k\cdot\mu,j} U_{x,k\cdot\mu}^\dagger + \omega_{i,0}\mathbf{1} \tag{3.2.18}$$

is fully compatible with the equivariance properties of the L-Conv layer. It is also worth noticing that the L-Conv operation is a linear map in the second argument $\mathcal{W}$, i.e.

$$\mathcal{C}_{x,i}(\mathcal{U}, \lambda_1 \mathcal{W}_1 + \lambda_2 \mathcal{W}_2) = \lambda_1 \mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}_1) + \lambda_2 \mathcal{C}_{x,i}(\mathcal{U}, \mathcal{W}_2), \tag{3.2.19}$$

for arbitrary complex parameters λ_1 and λ_2, provided that no bias term is used, i.e. $\omega_0 = 0$.

To conclude, we mention that dilated convolutions [100] are covered by the definition (3.2.13), since these types of convolutions are a subset of standard convolutions.

3.2.4   Lattice gauge equivariant bilinear layers

A particularly relevant operation for our applications, especially in combination with L-Conv, is represented by bilinear functions. This infrequently-used layer can be adapted to satisfy gauge equivariance. A lattice gauge equivariant bilinear layer (L-Bilin) takes as input two tuples $(\mathcal{U}, \mathcal{W})$ and $(\mathcal{U}, \mathcal{W}')$ and combines them in the following way:

$$\mathcal{B}_{x,i}(\mathcal{U}, \mathcal{W}, \mathcal{W}') = \sum_{j,k} \alpha_{i,j,k} W_{x,j} W'_{x,k}. \qquad (3.2.20)$$

The complex parameters $\alpha_{i,j,k}$ are learnable and the indices take values as follows: $1 \leq i \leq N_{\text{out}}$, $1 \leq j \leq N_{\text{in},1}$ and $1 \leq k \leq N_{\text{in},2}$.

Eq. (3.2.20) features a product of locally transforming variables, guaranteeing that gauge equivariance (3.2.2) is respected:

$$\begin{aligned}
\mathcal{B}_{x,i}(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}, T_\Omega \mathcal{W}') &= \sum_{j,k} \alpha_{i,j,k} \left(\Omega_x W_{x,j} \Omega_x^\dagger \right) \left(\Omega_x W'_{x,k} \Omega_x^\dagger \right) \\
&= \Omega_x \sum_{j,k} \alpha_{i,j,k} W_{x,j} W'_{x,k} \Omega_x^\dagger \\
&= T_\Omega \mathcal{B}_{x,i}(\mathcal{U}, \mathcal{W}, \mathcal{W}'). \qquad (3.2.21)
\end{aligned}$$

The bilinear operation can be extended by allowing $\mathcal{W}$ and $\mathcal{W}'$ to also include the unit element $\mathbf{1}$ and the Hermitian conjugates of $\mathcal{W}$ and $\mathcal{W}'$, meaning that $\mathcal{W}$ is replaced by $\tilde{\mathcal{W}} = \{\tilde{W}_{x,l}\}$ with $\tilde{N}_{\text{ch}} = 2\,N_{\text{ch}} + 1$ channels:

$$\tilde{W}_{x,l} = \begin{cases} W_{x,l}, & 1 \leq l \leq N_{\text{ch}}, \\ W_{x,l-N_{\text{ch}}}^\dagger, & N_{\text{ch}} < l \leq 2N_{\text{ch}} \\ \mathbf{1}, & l = 2N_{\text{ch}} + 1, \end{cases} \qquad (3.2.22)$$

With this enlarged set, the L-Bilin layer can act as a residual module [127] and includes a bias term associated with the parameter $\alpha_{i,0,0}$.

As in the case of the L-Conv, the L-Bilin operation is seen as only modifying the $\mathcal{W}$ variables, while keeping the set of gauge links $\mathcal{U}$ unaltered:

$$\mathcal{B} : (\mathcal{U}, \mathcal{W}, \mathcal{W}') \to (\mathcal{U}, \mathcal{W}''), \qquad (3.2.23)$$

with

$$\mathcal{W}'' = \left\{ W''_{x,i} = \mathcal{B}_{x,i}(\mathcal{U}, \mathcal{W}, \mathcal{W}'),\, 1 \leq i \leq N_{\text{out}},\, x \in \Lambda \right\}. \qquad (3.2.24)$$

In practice, the two layers introduced so far, L-Conv and L-Bilin, are combined into a single module, called L-CB, in the implementation of L-CNNs. In said layer, the terms contributing to the convolution are first appropriately parallel transported,

$$W'_{x+k\mu,j} = U_{x,k\mu} W_{x+k\mu,j} U_{x,k\mu}^\dagger, \qquad (3.2.25)$$

and afterwards the outcome is multiplied by the local terms $W_{x,j}$ with the bilinear operation just described, such that the resulting locally transforming variables are

$$W_{x,i} \to \sum_{j,j',k,\mu} \alpha_{i,j,j',k,\mu} W_{x,j} W'_{x+k\mu,j'}. \tag{3.2.26}$$

The L-CB operation is equivalent to stacking an L-Conv and an L-Bilin layer, except that learnable weights are parameterized differently.

3.2.5 Gauge equivariant activation functions

Another fundamental building block of neural networks which has to be generalized to satisfy gauge equivariance are non-linear activation functions. Even though the L-Conv and the L-Bilin operations can lead to non-linear terms, more general functions can be expressed with the addition of a lattice gauge equivariant activation function (L-Act). We can define it as being applied to each lattice site via

$$a_{x,i}(\mathcal{U}, \mathcal{W}) = g_{x,i}(\mathcal{U}, \mathcal{W}) W_{x,i}, \tag{3.2.27}$$

where the function g respects

$$g_{x,i}(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = g_{x,i}(\mathcal{U}, \mathcal{W}), \tag{3.2.28}$$

hence is gauge invariant.

The function a is gauge equivariant:

$$\begin{aligned} a_{x,i}(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) &= \Omega_x g_{x,i}(\mathcal{U}, \mathcal{W}) W_{x,i} \Omega_x^\dagger \\ &= T_\Omega a_{x,i}(\mathcal{U}, \mathcal{W}). \end{aligned} \tag{3.2.29}$$

The expression in Eq. (3.2.27) is very general. Usually, activation functions only depend on local quantities, which here corresponds to $g_{x,i}$ depending solely on the set of local objects $\{W_{x,i}\}$. For example, a widely-employed activation function in traditional neural networks is the rectified linear unit (ReLU)

$$\text{ReLU}(x) = \theta(x)\, x \tag{3.2.30}$$

with $x \in \mathbb{R}$ and $\theta(x)$ denoting the Heaviside step function. An analogous function that respects gauge equivariance has the form

$$g_{x,i}(\mathcal{U}, \mathcal{W}) = \theta(\text{ReTr}\,[W_{x,i}]). \tag{3.2.31}$$

If we now take the real part of the trace of $a_{x,i}$ we obtain

$$\begin{aligned} \text{ReTr}\,[a_{x,i}(\mathcal{U}, \mathcal{W})] &= \theta(\text{ReTr}\,[W_{x,i}]) \text{ReTr}\,[W_{x,i}] \\ &= \text{ReLU}(\text{ReTr}\,[W_{x,i}]), \end{aligned} \tag{3.2.32}$$

which is the behavior we wanted to recreate. Like L-Conv and L-Bilin, L-Act only modifies $\mathcal{W}$ variables, i.e.

$$a : (\mathcal{U}, \mathcal{W}) \rightarrow (\mathcal{U}, \mathcal{W}'), \tag{3.2.33}$$

with

$$\mathcal{W}' = \left\{ W'_{x,i} = a_{x,i}(\mathcal{U}, \mathcal{W}), \, 1 \le i \le N_{\text{ch}}, x \in \Lambda \right\}. \tag{3.2.34}$$

The scalar function in Eq. (3.2.28) can also depend on variables evaluated at any lattice site and, moreover, on trainable parameters, such as in PReLU [52].

3.2.6   Exponentiation layers

The L-CNN layers that have been discussed up to this point only alter the locally transforming objects $\mathcal{W}$ and leave the gauge links $\mathcal{U}$ unchanged. A gauge equivariant operation

$$\mathcal{E} : (\mathcal{U}, \mathcal{W}) \rightarrow (\mathcal{U}', \mathcal{W}) \tag{3.2.35}$$

needs to modify the gauge links guaranteeing that they retain their gauge transformation behavior

$$T_\Omega U'_{x,\mu} = \Omega_x U'_{x,\mu} \Omega^\dagger_{x+\mu}, \tag{3.2.36}$$

and concurrently preserve the unitarity and determinant constraints

$$U'^\dagger_{x,\mu} U'_{x,\mu} = \mathbf{1}, \tag{3.2.37}$$

$$\det U'_{x,\mu} = 1, \tag{3.2.38}$$

for all the links in the set $\mathcal{U}' = \{U'_{x,\mu}\}$.

The function $\mathcal{E}$ satisfies these properties if it acts on the links according to

$$U'_{x,\mu} = \mathcal{E}_{x,\mu} U_{x,\mu}, \tag{3.2.39}$$

and $\mathcal{E}_{x,\mu}$ is an $\text{SU}(N_c)$ group element which transforms locally:

$$T_\Omega \mathcal{E}_{x,\mu} = \Omega_x \mathcal{E}_{x,\mu} \Omega^\dagger_x. \tag{3.2.40}$$

A possible way of constructing $\mathcal{E}_{x,\mu}$ as a function of $\mathcal{W}$ objects is by means of the exponential map (L-Exp)

$$\mathcal{E}_{x,\mu}(\mathcal{W}) = \exp\left(i \sum_i \beta_{\mu,i} \left[W_{x,i} \right]_{\text{ah}} \right), \tag{3.2.41}$$

where $\beta_{\mu,i}$ are real trainable weights with indices $0 \le \mu \le D$ and $1 \le i \le N_{\text{ch}}$. The notation $[W_{x,i}]_{\text{ah}}$ indicates the anti-Hermitian traceless part of locally transforming objects, which for a generic matrix X can be written as

$$[X]_{\text{ah}} = \frac{1}{2i}\left(X - X^\dagger\right) - \frac{1}{2iN_c}\mathbf{1}\operatorname{Tr}\left(X - X^\dagger\right). \qquad (3.2.42)$$

The L-Exp layer (3.2.41) has local gauge transformation behaviour:

$$
\begin{aligned}
\mathcal{E}_{x,\mu}(T_\Omega \mathcal{W}) &= \exp\left(i\sum_i \beta_{\mu,k}\Omega_x\left[W_{x,i}\right]_{\text{ah}}\Omega_x^\dagger\right) \\
&= \sum_{n=0}^\infty \frac{1}{n!}\left(i\sum_i \beta_{\mu,k}\Omega_x\left[W_{x,i}\right]_{\text{ah}}\Omega_x^\dagger\right)^n \\
&= \Omega_x \sum_{n=0}^\infty \frac{1}{n!}\left(i\sum_i \beta_{\mu,k}\left[W_{x,i}\right]_{\text{ah}}\right)^n \Omega_x^\dagger \\
&= \Omega_x \exp\left(i\sum_i \beta_{\mu,k}\left[W_{x,i}\right]_{\text{ah}}\right)\Omega_x^\dagger \\
&= T_\Omega \mathcal{E}_{x,\mu}(\mathcal{W}),
\end{aligned}
\qquad (3.2.43)
$$

where we used the series expansion of the exponential map and the unitarity of Ω_x. The L-Exp operation also conserves the unitarity and determinant constraints:

$$\left(\sum_i \beta_{\mu,i}\left[W_{x,i}\right]_{\text{ah}}\right)^\dagger = \sum_i \beta_{\mu,i}\left[W_{x,i}\right]_{\text{ah}}, \qquad (3.2.44)$$

$$\operatorname{Tr}\left[W_{x,i}\right]_{\text{ah}} = 0. \qquad (3.2.45)$$

The determinant constraint is a consequence of the renowned relationship between the trace and the determinant: $\det e^A = e^{\operatorname{Tr}A}$.

This layer therefore updates the link configuration in a way that is consistent with gauge symmetry.

3.2.7 Trace layers

In the case an L-CNN is required to predict physical observables, it is convenient to build a layer that automatically integrates out the group structure and outputs a gauge invariant quantity. This can be realized by computing the trace of the $\mathcal{W}$ objects and neglecting the set of gauge links. The trace layer (Trace) is defined as

$$\mathcal{T}_{x,i}(\mathcal{U},\mathcal{W}) = \operatorname{Tr}\left[W_{x,i}\right], \qquad (3.2.46)$$

which is trivially gauge invariant

$$\mathcal{T}_{x,i}(T_\Omega \mathcal{U}, T_\Omega \mathcal{W}) = \mathrm{Tr}\left[\Omega_x W_{x,i} \Omega_x^\dagger\right]$$

$$= \mathcal{T}_{x,i}(\mathcal{U}, \mathcal{W}). \tag{3.2.47}$$

The output of Trace are $N_{\mathrm{lat}}N_{\mathrm{ch}}$ complex numbers, which correspond to $2N_{\mathrm{lat}}N_{\mathrm{ch}}$ real numbers:

$$\mathcal{T} : (\mathcal{U}, \mathcal{W}) \to \mathbf{y} \in \mathbb{R}^{2N_{\mathrm{lat}}N_{\mathrm{ch}}}, \tag{3.2.48}$$

which can be further processed by a CNN if the lattice structure is maintained, otherwise by a dense network, without ruining gauge symmetry.

3.2.8   Plaquette layers

In the tuple $(\mathcal{U}, \mathcal{W})$ introduced in Subsection 3.2.1, the link configuration $\mathcal{U}$ is the result of a Monte Carlo algorithm, as already mentioned, while the $\mathcal{W}$ variables need to be computed from the gauge links. The layers presented in this subsection and the next are used to calculate two common types of locally transforming objects. As opposed to the layers described so far, this type of operations is not characterized by trainable parameters and as such can be viewed as a preprocessing step.

Here, we introduce a layer

$$\mathcal{P} : (\mathcal{U}) \to (\mathcal{U}, \mathcal{W}), \tag{3.2.49}$$

called Plaq that computes the plaquettes (3.1.16) given the set of links $\mathcal{U}$ as follows

$$\mathcal{P}_{x,\mu\nu}(\mathcal{U}) = U_{x,\mu\nu} = U_{x,\mu}U_{x+\mu,\nu}U_{x+\nu,\mu}^\dagger U_{x,\nu}^\dagger, \tag{3.2.50}$$

thus yielding $\mathcal{W} = \{\mathcal{P}_{x,\mu\nu}\}$. In order to avoid redundancy, it is possible to restrict the calculation to plaquettes with positive orientation, i.e. $U_{x,\mu\nu}$ with $\mu < \nu$.

3.2.9   Polyakov layers

Another type of loop that has already been mentioned in Section 3.1 is the Polyakov loop, a Wilson loop that wraps around the lattice periodic boundary. Analogously to Plaq, we define the Poly layer

$$\mathcal{L} : (\mathcal{U}) \to (\mathcal{U}, \mathcal{W}), \tag{3.2.51}$$

as the function that computes Polyakov loops at each lattice site from the set of gauge links $\mathcal{U}$ with the following operation

$$\mathcal{L}_{x,\mu}(\mathcal{U}) = \prod_k U_{x+k\cdot\mu,\mu} = U_{x,\mu}U_{x+\mu,\mu}\ldots U_{x-\mu,\mu}. \tag{3.2.52}$$

The set of locally transforming objects is therefore $\mathcal{W} = \{\mathcal{L}_{x,\mu\nu}\}$. We point out that Polyakov loops transform locally at the starting site x because of the periodicity of the lattice. As shown in the next Section, plaquettes and Polyakov loops form a minimal basis to construct every possible Wilson loop on the lattice.

3.3 Lattice gauge equivariant convolutional neural networks

We list all the layers that have been introduced in the previous Section and can be combined using function composition to form a gauge equivariant neural network:

1. Gauge equivariant convolutional layer (L-Conv)

2. Gauge equivariant bilinear layer (L-Bilin)

3. Gauge equivariant activation function (L-Act)

4. Exponentiation layer (L-Exp)

5. Trace layer (Trace)

We also defined the preprocessing layers

1. Plaquette layer (Plaq)

2. Polyakov layer (Poly)

We now focus on how these layers can be assembled together into an architecture that is applicable to problems in lattice gauge theory.

A prototypical study is the prediction of physical observables, which are gauge invariant functions of Wilson loops, as discussed in section 3.1. L-CNNs should be able to tackle this task, which means that it should be possible for them to realize arbitrary Wilson loops by taking the set of gauge links $\mathcal{U}$ as input. This can be accomplished if we choose the initial $\mathcal{W}$ variables to be the set of plaquettes. Processing the initial tuple $(\mathcal{U}, \mathcal{W})$, L-CNN architectures are indeed able to grow Wilson loops of any shape and size. We illustrate the mechanism in 1+1D by considering a simple L-CNN architecture which computes a 2×1 Wilson loop from the set of gauge links $\mathcal{U}$:

$$\mathcal{U} \to \text{Plaq} \to \text{L-Conv} \to \text{L-Bilin} \to (\mathcal{U}, \mathcal{W}'). \tag{3.3.1}$$

With the first layer, the plaquettes $U_{x,01}$, the only ones available in 1+1D, are computed at each lattice size. The tuple $(\mathcal{U}, \mathcal{W} = \{U_{x,01}\})$ is passed to an L-Conv layer with $N_{\text{ch,in}} = 1$ input channels and $N_{\text{ch,out}} = 2$ output

channels. The learnable parameters can be adjusted to reproduce in one channel the original plaquettes $U_{x,01}$ and in the other the plaquettes parallel transported from x to $x + \hat{1}$. Then, the final bilinear layer combines the two channels into a single output channel multiplying the two plaquettes and giving as a result a 2×1 Wilson loop at each lattice site x.

The L-Conv+L-Bilin combination as detailed above can be used as a building block in deeper L-CNN architectures to compute Wilson loops of arbitrary size from elementary 1×1 loops. The number of times such a combination is repeated determines the maximum loop size that can be reached. For example, an L-Conv+L-Bilin combination with an appropriate kernel size in the convolution can double the area of a loop, such that two L-Conv+L-Bilin layers are required for 2×2 loops, three L-Conv+L-Bilin layers can realize 2×4 or 4×2 loops and four L-Conv+L-Bilin layers are necessary for 4×4 loops.

Although the method described above can be extended to the computation of arbitrarily large Wilson loops, not all loops can be realized this way. Polyakov loops, for example, are an exception, and the reason lies in their different topology: Wilson loops can be contracted to a single point, while Polyakov loops cannot. This issue can be conveniently circumvented by using as input $\mathcal{W}$ variables Polyakov loops, precomputed with the Poly layer.

In addition to taking care of the loop shape and size, L-CNNs are also able to provide non-linearity through the use of gauge equivariant activation functions in a similar fashion to how non-linear functions can be approximated in traditional neural networks. By analogy, the layer sequence L-Conv+L-Bilin+L-Act allows to express a wide range of gauge equivariant non-linear functions in lattice gauge theory.

An L-CNN architecture that can approximate a gauge invariant observable $\mathcal{O}[U]$ is given by

$$\mathcal{U} \to \text{Plaq} \to (\text{L-Conv} \to \text{L-Bilin} \to \text{L-Act})^n$$
$$\to \text{Trace} \to \text{CNN} \to \mathcal{O}[U], \tag{3.3.2}$$

where CNN indicates a traditional convolutional neural network and $(\dots)^n$ denotes the composition of n L-Conv+L-Bilin+L-Act blocks stacked one after the other. The Trace layer makes the data gauge invariant before it enters the CNN. If translational symmetry is relevant for the prediction of the observable, we recommend using a translationally equivariant CNN as described in Chapter 2, otherwise the CNN could even be replaced by a generic dense network without destroying gauge symmetry. An example of such an L-CNN architecture is shown in Fig. 3.1.

In contexts such as classical time evolution [128] or gradient flow [129], the set of gauge links $\mathcal{U}$ are required to undergo modifications, which is possible via the application of L-Exp. The $\mathcal{W}$ variable can be updated accordingly in the following Plaq or Poly layer.
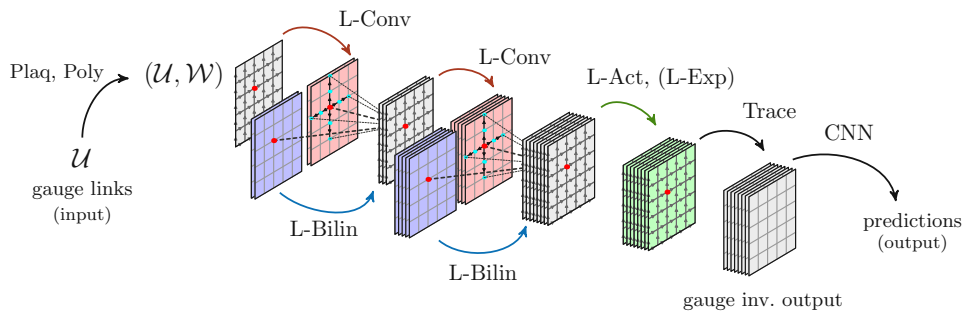
Figure 3.1: An example of L-CNN architecture. Using gauge links $\mathcal{U}$, locally transforming objects $\mathcal{W}$ are computed via the Plaq and Poly layers. The tuple $(\mathcal{U}, \mathcal{W})$ is the input of the L-CNN. A gauge-preserving convolutional operation is performed by L-Conv using parallel transporters to connect nearby $\mathcal{W}$ objects (green dots) along the coordinate axes to a specific lattice site (red dot). The output of such a layer (blue) is combined with the original one (red) by L-Bilin, which multiplies locally transforming objects in a gauge-equivariant manner. The stacked configuration sheets signal that L-Bilin can act on a preset number of channels. Non-linearity can be introduced via L-Act, while modifications of $\mathcal{U}$ are performed by L-Exp, both in a gauge-equivariant way (green layer). The group structure is integrated out with a Trace layer, and its gauge invariant output can be further processed by a traditional CNN. The image displays 1+1D configurations but there is no restriction to the number of dimensions. Image from [2].
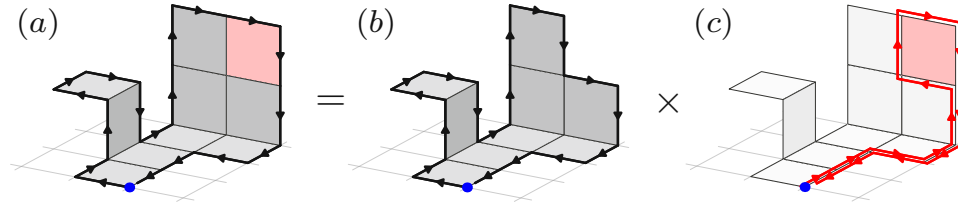
Figure 3.2: Sketch of the proof that L-CNNs can generate arbitrarily sized Wilson loops. An arbitrary Wilson loop (a) can be viewed as the boundary of a surface that can be tessellated into n tiles, each of which has a 1×1 unit lattice area. The initial point of the loop is indicated by a blue dot. This untraced Wilson loop can be decomposed into another one with $n - 1$ tiles (b) and the loop surrounding the removed tile connected to the starting point via an appropriate path (c). This path has to run along the loop (b) and can be generated by consecutive L-Conv layers. Image from [2].

We need to point out that depending on the problem, other gauge equivariant layers can be introduced. For example, in [89] and [90] the coupling layer of a normalizing flow is made gauge equivariant by imposing it to be dependent on gauge invariant objects, specifically traced plaquettes made up by links that are not being updated by said coupling layer. An adaptation of the L-Conv operation has also been used in [84] and [85] for the application of gauge equivariant neural networks to multigrid preconditioners for the Dirac operator [130], together with gauge equivariant pooling and unpooling layers that are combined with subsampling such that gauge symmetry is respected. A generalization of L-CNN layers (also including pooling operations) to preserve global symmetries as in the G-CNN construction [60] is given in [83].

3.4   Proof for generation of arbitrary Wilson loops

We provide a sketch of a proof by induction that L-CNNs can generate arbitrary Wilson loops, which is displayed in Fig. 3.2.

By virtue of the non-Abelian version of Stokes' theorem [131], it is possible to associate a contractible Wilson loop with its enclosing surface, which can be tessellated into area units, i.e. plaquettes. Such a surface is not unique in higher dimensions, hence a Wilson loop can be equivalently described by different tessellations. In order to be able to represent any Wilson loop, on topologies that are not simply connected it is necessary to include in the set of $\mathcal{W}$ loops that can not be contracted to a point, such as Polyakov loops.

An arbitrary Wilson loop can be decomposed into the same Wilson loop with a missing tile and another one, such that their product is the original

Wilson loop. By means of the iterated application of L-Convs, it is always possible to generate the path that connects the origin (marked with a blue dot in Fig. 3.2) with the removed plaquette, which can be computed via a Plaq layer. The Wilson loop obtained this way can be multiplied by the Wilson loop with the missing tile in a L-Bilin layer. This argument is valid for any starting Wilson loop larger than a plaquette. This guarantees that L-CNNs can in principle construct arbitrarily sized Wilson loops. In L-CNNs, generic Wilson loops are combined more efficiently by taking products of two larger sub-loops, instead of always multiplying with a loop of unit area.

The possibility of generating any closed loop enables the reconstruction of the complete gauge connection up to gauge transformations [132, 133]. With the usage of an L-Act layer that depends on the trace of loops, an L-CNN is therefore able in principle to approximate arbitrary non-linear functions of Wilson loops. Recalling the universality of CNNs [47, 48], L-CNNs can be seen as universal approximators of functions on the lattice.

3.5 Computational experiments

With an approach similar to the one in Chapter 2, we want to demonstrate the advantage of designing neural networks that preserve gauge symmetry to tackle problems in lattice gauge theory. Specifically, we compare the performance of L-CNNs with traditional CNN models on regression tasks consisting in the prediction of local gauge invariant observables $\mathcal{O}_x$. An observable that we consider is the real value of traced Wilson loops, given by

$$W_{x,\mu\nu}^{(m\times n)} = \frac{1}{N_c}\mathrm{ReTr}\left[U_{x,\mu\nu}^{(m\times n)}\right] \qquad (3.5.1)$$

where $U_{x,\mu\nu}^{(m\times n)}$ refers to an $m \times n$ Wilson loop in the $\mu\nu$ plane:

$$U_{x,\mu\nu}^{(m\times n)} = U_{x,m\cdot\mu}U_{x+m\cdot\mu,n\cdot\nu}U_{x+n\cdot\nu,m\cdot\mu}^{\dagger}U_{x,n\cdot\nu}^{\dagger}. \qquad (3.5.2)$$

The concatenation of gauge links $U_{x,n\cdot\mu}$ is expressed in Eq. (3.2.14). These observables are defined at every lattice site and are gauge invariant. The rationale for these experiments is that relevant observables in lattice gauge theory can be represented by non-linear functions of Wilson loops of arbitrary size.

Another observable that we focus on is the topological charge density q_x. The definition used here is the plaquette discretization given by Eq. (3.1.24).

3.5.1 Datasets

This section provides detailed information about the datasets to which the neural networks have access. The experiments are conducted on data generated in 1+1D and 3+1D with various lattice sizes and coupling constants

Table 3.1:   Datasets for regression tasks in 1+1D and 3+1D. The coupling constant β ranges from $\beta_{\min} = 0.1$ to $\beta_{\max} = 6.0$, with intermediate values separated by constant steps of $\Delta\beta = (\beta_{\max} - \beta_{\min})/N_\beta$, with $N_\beta = 10$. Table from [2].

1+1D			
	Training	Validation	Test
Lattice $N_t \times N_s$	8×8	8×8	8×8, 16×16, 32×32, 64×64
Examples	10^4	10^3	10^3 per lattice
Labels	$W_{x,01}^{(1\times1)}$, $W_{x,01}^{(1\times2)}$, $W_{x,01}^{(2\times2)}$, $W_{x,01}^{(4\times4)}$		
Coupling	$\beta \in \{0.1, \ldots, 6.0\}$		
3+1D			
	Training	Validation	Test
Lattice $N_t \times N_s^3$	4×8^3	4×8^3	4×8^3, 6×8^3, 6×12^3, 8×16^3
Examples	10^4	10^3	10^3 per lattice
Labels	$W_{x,12}^{(2\times2)}$, $W_{x,12}^{(4\times4)}$, q_x^{plaq}		
Coupling	$\beta \in \{0.1, \ldots, 6.0\}$		

β using an SU(2) Markov chain Monte Carlo (MCMC) code, detailed in Appendix A. The coupling β is related to the Yang-Mills coupling g in the continuum by

$$\frac{\beta}{2N_c} = \frac{1}{g^2}. \tag{3.5.3}$$

In order to achieve a higher computational efficiency, the set of 1×1 loops was precomputed and chosen as the locally transforming variables $\mathcal{W} = \{U_{x,\mu\nu}\}$ that is part of the input tuple $(\mathcal{U}, \mathcal{W})$ for the networks. In the following regression problems, such tuples have to be mapped to the following gauge invariant observables: $W_{x,\mu\nu}^{(1\times1)}$, $W_{x,\mu\nu}^{(1\times2)}$, $W_{x,\mu\nu}^{(2\times2)}$, $W_{x,\mu\nu}^{(4\times4)}$, and topological charge q_x^{plaq}. The indices $\mu\nu$ take specific values: in 1+1D all loops lie in the tx plane, therefore we compute $W_{x,01}^{(m\times n)}$. In 3+1D we choose the xy plane for all loops, thus $W_{x,12}^{(m\times n)}$. Furthermore, the definition of the topological charge in the form of Eq. (3.1.24) is only valid in 3+1D and is therefore only included in the 3+1D datasets.

In order to create the datasets, lattice configurations are randomly initialized and then undergo 2×10^3 sweeps such that the system reaches equilib-

rium. The tuple $(\mathcal{U}, \mathcal{W})$ is then saved together with the desired observables $\{\mathcal{O}_i\}$ every 10^2 sweeps. In both the 1+1D and the 3+1D case, the training set consists of 10^4 samples and the validation set of 10^3 samples, which are generated only for the smallest lattice size. The larger lattice sizes are used only for the test sets, while the 10 different values of the coupling constant β are employed both during training and testing. All details are reported in Table 3.1.

In the next two sections we discuss the hyperparameter choices made for the comparison between L-CNN architectures and traditional CNNs, which are used as baseline models that do not respect gauge symmetry by design.

3.5.2 L-CNN networks

The fully detailed composition of L-CNN architectures is contained in Tables B.1 and B.2 in Appendix B. In this section, we will go over their most relevant characteristics.

As mentioned in subsection 3.2.4, L-Conv and L-Bilin layers are combined into the L-CB operation. The architectures feature up to four L-CB layers stacked one after another, followed by a Trace layer and a single linear layer applied individually at each lattice site (corresponding to a 1×1 convolution [134]), with a total number of weights of varying from ≈ 100 to $\approx 40\,000$. More specifically, we employed only one L-CB layer for the smallest Wilson loop, $W^{(1\times1)}$. The prediction of larger Wilson loops requires an increasing number of L-CB operations. Given that an L-CB layer doubles the maximum possible loop area, the repetition of n L-CB layers that take plaquettes as initial input allows to predict loops of area 2^n. In order to generate loops with a size different from powers of 2, it is possible to store loops of different sizes in different channels. As an illustration, constructing 3×3 loops can be done by combining loops in the following sequence: $1 \times 1 \rightarrow (1 \times 1, 1 \times 2) \rightarrow (1 \times 1, 1 \times 2, 2 \times 2) \rightarrow (1 \times 3, 2 \times 3) \rightarrow 3 \times 3$, where $(\dots, \dots)$ indicates multiple channels. Compared to the general structure proposed in (3.3.2), L-Act layers are superfluous for the observables featured here. The L-CNN output is defined at every lattice site, allowing a pointwise comparison with the ground truth. We have observed that leaving out the lattice average, i.e. a GAP layer, leads to much easier convergence of the networks during training. The inclusion of a GAP makes training much harder, often leading to models that do not converge.

3.5.3 Baseline networks

The baseline models are CNNs that break gauge symmetry, whose details are displayed in Tables C.1, C.2 and C.3 in Appendix C. These networks consist of stacks of two-dimensional convolutional layers followed by an activation function, a GAP at the end of the convolutional part and a dense part leading

to the predictions. With the aim of comparing the architectures fairly, the number of trainable parameters of the CNNs is chosen similar to L-CNN architectures, such that their size and their ability to make predictions is in principle comparable. The amount of convolutions ranges from one or two up to six, and the number of trainable parameters varies from ≈ 300 to $\approx 100\,000$. Like the L-CNN networks, the CNN models are translationally equivariant, which is achieved by choosing a stride of one in all convolutions, as discussed extensively in Section 2.2. Also, periodic boundary conditions are satisfied imposing circular padding. This choice is justified by recalling that lattice gauge theory is symmetric under translations. We also note that both architecture types can be applied to different lattice sizes.

Another aspect we considered in order to ensure a fair comparison concerns the input of the CNNs. While in principle providing the networks only with gauge links $\mathcal{U}$ is sufficient, one might argue that L-CNNs are favored because they have also access to plaquettes in the input layer. For this reason, the input of the baseline networks also includes plaquettes. Empirically, the input tuple for these architectures that leads to the best performances is $(\mathcal{U}, \mathcal{W}, \mathcal{W}^\dagger)$, which is the one assumed to be used for all the experiments with baseline models.

We observed that CNNs without a GAP layer performed much worse, in contrast to the results found for L-CNNs. In order to enable a comparison, we take the lattice average over the final output layer of the L-CNN models.

3.5.4   Training details

For both architecture types, we reduce the impact of random initializations by training ten models with different initial random weights. In all tasks, we use the *AdamW* optimizer with no weight decay and early stopping. In 1+1D, the choices for the learning rate, the maximum number of epochs and the patience are chosen to be respectively 3×10^{-2}, 100 and 25, while for L-CNNs these number are different across the various tasks: for the prediction of 1×1 and 1×2 Wilson loops, the learning rate, the maximum number of epochs and the patience are respectively equal to 3×10^{-3}, 20 and 5; for 2×2 and 4×4, these hyperparameters are set to 1×10^{-3}, 100 and 25 respectively.

Given the higher computational cost in 3+1D, training is run only for five models of the architecture reported in table B.2. For the prediction of $W^{(2\times 2)}$ and $W^{(4\times 4)}$, the learning rate is set to 3×10^{-3}, while it is equal to 3×10^{-4} for the topological charge Q_P.

For all architectures in 1+1D, the batch size is always set to 50, and it is reduced to 10 in 3+1D because of memory limits.
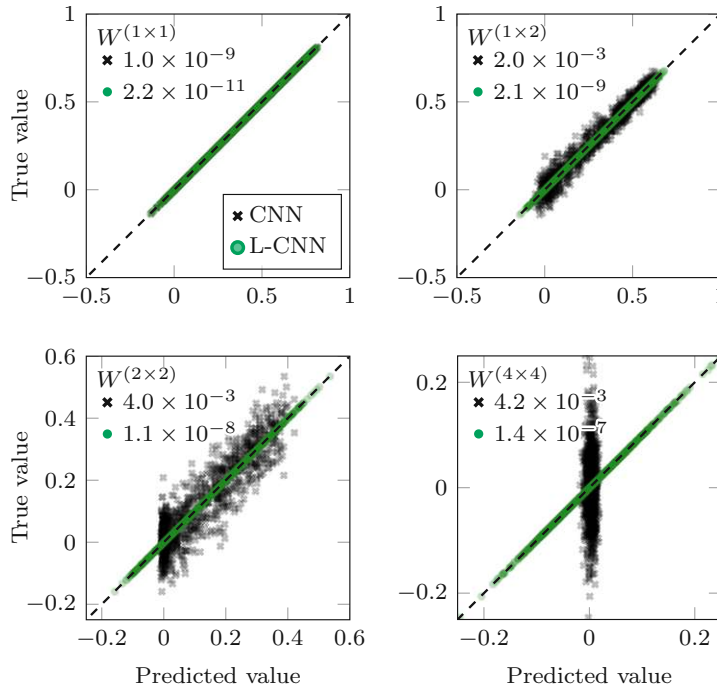
Figure 3.3: Scatter plot showing predictions versus ground truth of best L-CNN and CNN models for traced Wilson loops of various sizes tested on 8×8 lattices. The dashed line represents points where perfect predictions lie. The performance of the traditional CNN model strongly deteriorates as the loop size increases, to the point where predictions of 4×4 loops mostly lie close to the average of the training set, signaling that the CNNs did not learn any meaningful physical information. On the contrary, the L-CNN model scores a very low MSE (shown at the top left of each plot) for every loop size. Image from [2].

3.5.5 Traced Wilson loops in 1+1D

The first task we investigate is almost trivial: predicting the real value of the traced 1×1 Wilson loop as in Eq. (3.5.1). Since part of the input data consists of all possible 1×1 Wilson loops, the only required computations are first to perform the trace operation and then to take its real part.

Scatter plots comparing predictions to true values on 8×8 and 64×64 are shown in the top left plot of Figs. 3.3 and 3.4 respectively. The models chosen for these plots are the best models based on validation loss on 8×8 lattices. It is clear that both the L-CNN and the CNN models can perform the task of computing $W^{(1\times1)}$, although the L-CNN performs better by a few orders of magnitudes in terms of MSE. This should not come as a surprise, considering that the L-CNN architecture includes a Trace layer, whereas the
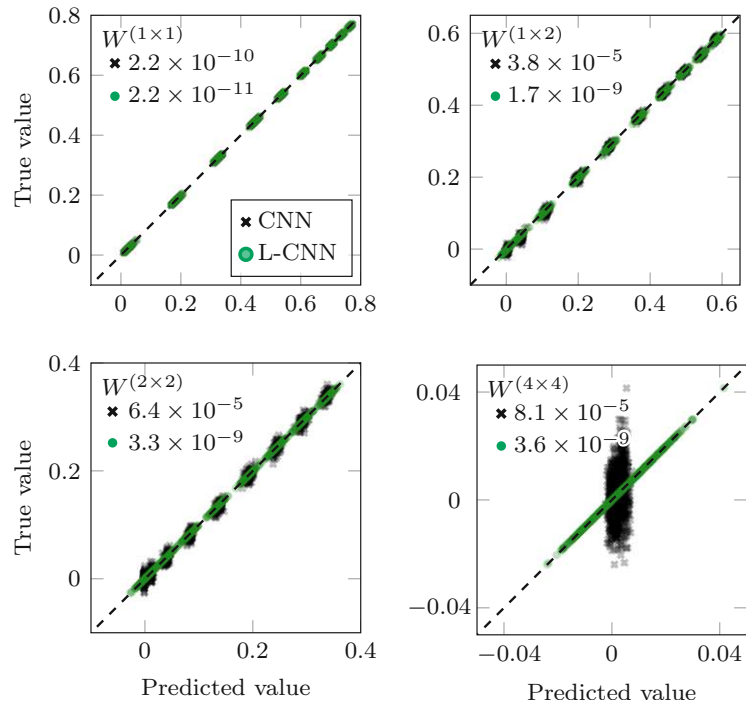
Figure 3.4: Scatter plot showing predictions versus ground truth of best L-CNN and CNN models for traced Wilson loops of various sizes tested on 64×64 lattices. This is the same comparison made in Fig. 3.3, but on the larger lattice size available in the test set. Similar conclusions to the previous scatter plot can be drawn. Each cluster visible in the plots is related to a specific value that the coupling constant β takes. Image from [2].

baseline model has to learn this operation during training. It should also be stressed that both architectures generalize well across different lattice sizes, even though training has only been performed on the smallest lattice size. As in Chapter 2, our models do not need to be retrained when tested on larger lattices, and this applies to all tasks analyzed in this section. More details about the results obtained by the different architectures in the prediction of $W^{(1 \times 1)}$ are provided in Appendix D in Table D.1.

Traced Wilson loops of sizes 1×2, 2×2 and 4×4 offer more sophisticated regression tasks. In this case, we try different L-CNN architectures that are small, medium and large in terms of number of parameters for each task (see Table B.1). Analogously to $W^{(1 \times 1)}$, we display a comparison for these larger loops in the top right and bottom plots of Figs. 3.3 and 3.4, where CNNs clearly do not solve adequately the more complicated task of computing larger Wilson loops. Conversely, all three L-CNN architectures perform well on the 1×2 and 2×2 tasks, and larger L-CNN models perform generally better in the 4×4 regression task. In some cases, the models do not converge during training, which shows why it is essential to repeat the training process multiple times with different random initializations of the weights. Another noteworthy observation is that baseline models struggle to predict negative values of traced Wilson loops, as can be seen in the top right and bottom left plots of Fig. 3.3 for $W^{(1 \times 2)}$ and $W^{(2 \times 2)}$. In the case of $W^{(4 \times 4)}$, the CNN seems to collapse and its predictions lie very close to the training set average. This shows that the baseline architecture is unable to learn any meaningful connection between the input and output data.

We report the performance of the different architectures used for the regression tasks with Wilson loops in Appendix D in Tables D.1, D.2, D.3 and D.4.

3.5.6 Traced Wilson loops in 3+1D

The Wilson loop regression task is further complicated by moving from 1+1D to 3+1D lattices. In the previous section we have already demonstrated that traditional CNNs are not able to satisfactorily learn even moderately sized Wilson loops in 1+1D. Furthermore, four-dimensional convolutional layers are currently not built-in modules in *PyTorch*, leading to a suboptimal formulation of traditional CNN architectures in 3+1D. We therefore focus only on testing L-CNN models in this section.

The regression task is very similar to the one in 1+1D: we train models to predict the value of traced 2×2 and 4×4 Wilson loops. One immediate complication that arises when going from 1+1D to 3+1D lattices is that memory requirements grow quickly. If we consider a $D + 1$-dimensional lattice with N_s lattice sites for each spatial coordinate and N_t lattice sites in the temporal one, the amount of real numbers in the input layer is (the

first summand coming from the links, the second from the loops)

$$N_{\text{input}} = 2\,N_c^2\,N_t\,N_s^D\left((D+1)+(D+1)\frac{D}{2}\right),\qquad (3.5.4)$$

and we see that a single sample of the $4 \cdot 8^3$ training set is larger by two orders of magnitude compared to a sample of the $8 \cdot 8$ training set. As a result, training a large L-CNN architecture takes longer, which is why the batch size, the total number of epochs and the number of differently initialized instances are reduced compared to the 1+1D case. Test results are summarized in Appendix D in Table D.5. Interestingly, smaller architectures seem to perform better in 3+1D compared to larger architectures. Overall, it is clear that L-CNNs are generally able to represent traced Wilson loops in higher dimensions. We also observe that successfully trained models generalize across different lattice sizes in 3+1D.

We conclude by mentioning that it may seem trivial for L-CNNs to solve regression tasks involving Wilson loops, since they are intrinsically constructed to form arbitrarily sized loops. As extensively discussed in the supplementary material of [2], the number of possible loops grows exponentially with their perimeter, therefore problems where the contribution from different Wilson loop shapes is unknown are challenging. An example is given by the so-called improved actions, which have been conceived to alleviate the influence of discretization effects through the inclusion of variously sized Wilson loops [135–137]. In particular, for fixed-point actions the renormalization group is used to remove lattice artifacts to obtain, for example, on-shell quantities [138]. The application of L-CNNs to a fixed-point action in SU(3) has improved the previously existing state-of-the-art results [86]. Another situation in which the contribution of various Wilson loops has to be determined is represented by trivializing maps, which map a theory to its strong-coupling limit, reducing critical slowing down. The approach suggested by Lüscher in [139] can be extended order by order identifying the relevant loops and computing their coefficient, an approach that can be systematized as proposed in [94]. In principle, L-CNNs can be used in this approach, as we will describe in Chapter 4.

3.5.7 Topological charge in 3+1D

The final observable we consider is the topological charge density q_x^{plaq} in the plaquette approximation (see Eq. (3.1.24)). Since the numerical values of this observable are rather small, it is reasonable to rescale the dataset by means of a constant factor, i.e.

$$\tilde{q}_x^{\text{plaq}} = Cq_x^{\text{plaq}},\qquad C>0,\qquad (3.5.5)$$

in order to facilitate training. For this experiment, we employ $C = 100$. In the following, Q_P will denote the topological charge, which is equal to the

sum over the lattice of the charge density

$$Q_P = \sum_{\mathbf{x}} q_{\mathbf{x}}^{\text{plaq}}. \tag{3.5.6}$$

The topological charge density is rather simple to describe in comparison to larger Wilson loops: it can be written as a linear combination of multiplications of two 1×1 Wilson loops. This regression task can be solved by relatively shallow L-CNN models. The architecture employed is reported in Table B.2, and its results are shown in Table D.5. Only one of the five randomly initialized models did not converge during training. In all other cases, we observed convergence to solutions which can predict the topological charge to a high degree of accuracy. These results are in line with the MSEs exhibited by L-CNN architectures in the two-dimensional tasks.

Having at our disposal the topological charge enables us to perform another more elaborated test than just a performance check with the MSE. A typical lattice study is the evolution of Q_P under Wilson flow or more generally gradient flow [121, 129]. In the continuum limit, the topological charge takes integer values due to restrictions that arise from the topology. We design a Wilson flow starting from configurations $U_{x,\mu}$ obtained from the MCMC simulation in Appendix A and then moderately smeared, such that the initial condition for the Wilson flow is

$$U_{\mathbf{x},\mu}(\tau = 0) = U_{\mathbf{x},\mu}, \tag{3.5.7}$$

with τ denoting the auxiliary Wilson flow time. The update procedure employed to realize Wilson flow is

$$U_{\mathbf{x},\mu}(\tau + \Delta\tau) = \exp\left(i\Delta\tau\,\omega_{\mathbf{x},\mu}(\tau)\right)U_{\mathbf{x},\mu}(\tau), \tag{3.5.8}$$

where ω is an algebra element given by

$$\omega_{\mathbf{x},\mu}(\tau) = -\sum_{|\nu|} [U_{\mathbf{x},\mu\nu}(\tau)]_{\text{ah}}, \tag{3.5.9}$$

with the sum $\sum_{|\nu|}$ running over the positive and the negative direction of the axis ν and the anti-Hermitian is computed as in Eq. (3.2.42). The steps $\Delta\tau$ are chosen to be small, leading to a gradual decrease of the Wilson action:

$$S_W[\mathcal{U}(\tau + \Delta\tau)] < S_W[\mathcal{U}(\tau)], \qquad \Delta\tau \ll 1. \tag{3.5.10}$$

Using $\Delta\tau = 0.005$ and training uniquely on the un-flowed configurations $U_{x,\mu}$ with a lattice size of 4×8^3, the L-CNN models reach a high degree of accuracy on 8×24^3 lattices, as can be seen in Fig. 3.5.

We notice that as the configurations are flowed the values of the topological charge get closer to integers, which is the expected behavior, whose details are related to the lattice size and the initial condition at $\tau = 0$.
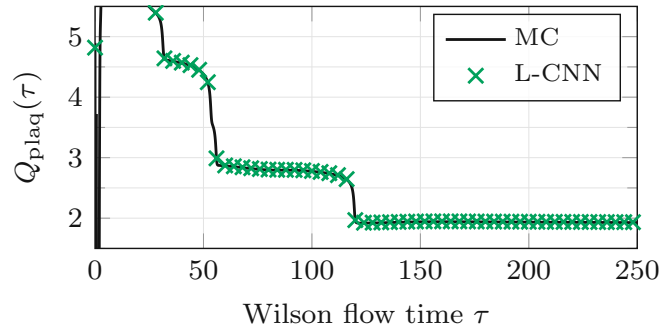
Figure 3.5: Predictions of the best L-CNN model for the topological charge on an 8×24^3 configuration at $2/g^2 = 0.2$ evolved with Wilson flow. MC indicates the Wilson flow obtained with Monte Carlo updates. Image from [2].

Chapter 4

Neural gradient flow

In this chapter, we design a method for generating configurations distributed according to a given action starting from a set of random configurations. We describe how the mapping between the initial and the final configurations can be approximated by solving a differential equation with neural networks, specifically with L-CNNs in the context of gauge theories. Since the initial configurations are random, the mapping amounts to a trivializing flow, as introduced by Lüscher in [139]. The advantage of this framework is that sampling new configurations is very cheap once the neural network has been successfully trained. The differential equation that is solved is equivalent to a gradient flow [121, 129] type of equation, which is why we call this technique neural gradient flow. We check the ability of a neural network to solve such an equation on a toy model consisting of single-link configurations in SU(2). We employ a simple network that mimics the behavior of L-CNNs and test it initially on the same interval of gradient flow time used during training, then we extend the test to much later times. Finally, we discuss the adjoint sensitivity method, which avoids memory saturation when performing backpropagation, and give our derivation in SU(N).

4.1 Adaptation of NODEs to lattice gauge theory

The starting point of this chapter is a technique that was proposed in [140]. It aims to solve the first order ordinary differential equation for the time-dependent vector $\mathbf{x}(t) \in \mathbb{R}^D$

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t), \theta, t), \qquad (4.1.1)$$

where the function $\mathbf{f}(\mathbf{x}(t), \theta, t)$ is not known a priori, hence its parameters θ have to be optimized to describe the dynamics of the vector. A neural network can be used to represent $\mathbf{f}$, which is why this setup takes the name neural ordinary differential equation (NODE). It is possible to interpret

NODEs as a special case of residual networks [127] in the limit of the discrete step between each layer going to zero, becoming the continuous time t in Eq. (4.1.1). Given an input state $\mathbf{x}_0 = \mathbf{x}(t_0)$ as initial condition at $t = t_0$, we can formally solve the NODE integrating Eq. (4.1.1):

$$\mathbf{x}(T) = \mathbf{x}_0 + \int_{t_0}^{T} \mathrm{d}t\, \mathbf{f}(\mathbf{x}(t), \theta, t). \tag{4.1.2}$$

Common ODE solvers such as Euler and Runge-Kutta (see e.g. Appendix A.7.1 of [141]) can be employed to get the prediction $\mathbf{x}(T)$ at the final time T. In order to be able to train the network, we can provide a dataset containing N_{samples} initial states $\mathbf{x}_0^i = \mathbf{x}^i(t_0)$ and target final states $\tilde{\mathbf{x}}_T^i$, then use the MSE (cf. with Eq. (2.4.5))

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{samples}}} \sum_{i=1}^{N_{\text{samples}}} (\tilde{\mathbf{x}}_T^i - \mathbf{x}^i(T))^2 \tag{4.1.3}$$

as loss function to optimize the weights θ. If other intermediate states of the actual dynamics $\tilde{\mathbf{x}}_j^i$ at time t_j are known, it is possible to include in the MSE also the discrepancies between them and the predictions the network yields, thus leading to

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{samples}}} \sum_{i=1}^{N_{\text{samples}}} \sum_{j=1}^{n} (\tilde{\mathbf{x}}_j^i - \mathbf{x}^i(t_j))^2, \tag{4.1.4}$$

with $t_n = T$. Resorting exclusively to the final state using the loss in Eq. (4.1.3) requires that the dataset provides sufficient information for a successful reconstruction of the underlying dynamics.

Another option is to treat Eq. (4.1.1) as a continuous normalizing flow [140], where the goal is the approximation of a probability distribution that can be achieved for example using the Kullback-Leibler (KL) divergence

$$\mathrm{KL}(q(\mathbf{x}) \,||\, p(\mathbf{x})) = \int \mathrm{d}\mathbf{x}\, p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \tag{4.1.5}$$

between the target distribution $p(\mathbf{x})$ and the probability distribution of the model $q(\mathbf{x})$.

As already mentioned in Chapter 3, in lattice gauge theory expectation values of observables are calculated via the path integral

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathrm{D}[U]\, \mathcal{O}(U) \mathrm{e}^{-S[U]}, \tag{4.1.6}$$

with Z being the partition function

$$Z = \int \mathrm{D}[U] \mathrm{e}^{-S[U]}. \tag{4.1.7}$$

If the links are redefined as $U = \mathcal{F}(V)$, the formula for the expectation value becomes

$$\langle \mathcal{O} \rangle = \frac{1}{Z} \int \mathrm{D}[V] \, \mathcal{O}(\mathcal{F}(V)) \mathrm{e}^{-S_{\mathcal{F}}[V]}, \qquad (4.1.8)$$

where the action $S_{\mathcal{F}}[V]$ incorporates the change of measure:

$$S_{\mathcal{F}}[V] = S(F(V)) - \log \det J(\mathcal{F}), \qquad (4.1.9)$$

with $J(\mathcal{F})$ indicating the Jacobian. The transformation $\mathcal{F}$ is said to be a trivializing map if $S_{\mathcal{F}}[V]$ is equal to a real-valued constant. In fact, the probability distribution for the redefined action is given by

$$p(V) = \frac{1}{Z} \mathrm{e}^{-S_{\mathcal{F}}[V]}, \qquad (4.1.10)$$

which is also constant, therefore it is possible to compute expectation values via (4.1.8) sampling from a uniform distribution. A model that approximates the transformation $\mathcal{F}(V)$ is characterized by a probability distribution $q(V)$ and can learn to reproduce a uniform distribution using the KL divergence (4.1.5), which is the approach used in [94] to find trivializing maps.

In order to find trivializing maps, Lüscher proposed to solve an equivariant gradient flow equation that can be written as

$$\frac{\mathrm{d}U_{x,\mu}(\tau)}{\mathrm{d}\tau} = iH_\mu[U_{x,\mu}(\tau), \tau] \, U_{x,\mu}(\tau), \qquad (4.1.11)$$

where $U_{x,\mu}$ is a configuration of $\mathrm{SU}(N_c)$ gauge links and τ is the gradient flow time. It is natural to parameterize H_μ as the gradient of the flow action $\tilde{S}$ (hence the name gradient flow), i.e.

$$iH_\mu[U_{x,\mu}(\tau), \tau] = -\partial_{x,\mu}\tilde{S}[U(\tau)], \qquad (4.1.12)$$

where we introduced the group derivative

$$\partial_{x,\mu} f(U) = t_a \frac{\mathrm{d}f(\mathrm{e}^{\epsilon t^a} U_{x,\mu})}{\mathrm{d}\epsilon}\bigg|_{\epsilon=0}, \qquad (4.1.13)$$

with t_a being the group generators. The action can be written as a linear combination of Wilson loops

$$\tilde{S}[U, \tau] = \sum_i c_i(\tau) W_i(U), \qquad (4.1.14)$$

where the coefficients c_i have to be determined. In [139], this action was expanded as

$$\tilde{S}[U, \tau] = \sum_{n=0}^{\infty} \tau^n \tilde{S}^{(n)} \qquad (4.1.15)$$

in the flow time $\tau \in [0, 1]$, and the leading order $\tilde{S}^{(0)}$ and its first correction $\tilde{S}^{(1)}$ were calculated analytically. In [94], using a predetermined set of loops, the coefficients c_i were machine-learned as a function of the flow time. The ability of L-CNNs to generate in principle any Wilson loop allows even more flexibility in tackling this problem. We therefore intend to solve the following flow equation

$$\frac{\mathrm{d}U_{x,\mu}(\tau)}{\mathrm{d}\tau} = i H_\mu[U_{x,\mu}(\tau), \theta, \tau]\, U_{x,\mu}(\tau), \tag{4.1.16}$$

where $H_\mu[U(\tau), \theta, \tau]$ is an L-CNN parametrized by the learnable parameters θ. We can show that H needs to be traceless and Hermitian. This derives from the requirement that the links remain in the group during the flow. The unitarity constraint

$$U(\tau)U^\dagger(\tau) = \mathbf{1} \tag{4.1.17}$$

needs to be preserved at every flow time τ. We dropped the indices x and μ for simplicity. Taking the derivative of both sides of the equation, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}\tau}(UU^\dagger) = \frac{\mathrm{d}U}{\mathrm{d}\tau}U^\dagger + U\frac{\mathrm{d}U^\dagger}{\mathrm{d}\tau} = 0, \tag{4.1.18}$$

where the dependence on time is implied. The Hermitian of Eq. (4.1.16) is

$$\frac{\mathrm{d}U^\dagger}{\mathrm{d}\tau} = -iU^\dagger H^\dagger. \tag{4.1.19}$$

When inserted in Eq. (4.1.18), we obtain

$$iHUU^\dagger - iUU^\dagger H^\dagger = 0, \tag{4.1.20}$$

which can be simplified using Eq. (4.1.17), bringing us to the Hermiticity of H, i.e.

$$H = H^\dagger. \tag{4.1.21}$$

The other property that has to be maintained under neural gradient flow is that the links must be represented by special matrices:

$$\det U(\tau) = 1. \tag{4.1.22}$$

We can recall Jacobi's formula, which relates the determinant of a square matrix $A(t)$ dependent on a real variable t with its trace as follows:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\det A(t)\right) = \det A(t)\,\mathrm{Tr}\left(A^{-1}(t)\frac{\mathrm{d}A(t)}{\mathrm{d}t}\right). \tag{4.1.23}$$

Jacobi's formula for the link variable $U(\tau)$ gives

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\left(\det U(\tau)\right) = \det U(\tau)\,\mathrm{Tr}\left(U^{-1}(\tau)\frac{\mathrm{d}U(\tau)}{\mathrm{d}\tau}\right). \tag{4.1.24}$$

Using the specialness of $U(\tau)$ (4.1.22) and Eq. (4.1.16), we find that

$$0 = 1 \cdot \text{Tr}\left(U^{-1}(iHU)\right), \tag{4.1.25}$$

where we imply the dependence on time. Since the trace is cyclic, U and U^{-1} simplify and we are left with the tracelessness of H:

$$\text{Tr}H = 0. \tag{4.1.26}$$

We can also derive the transformation property of H during gradient flow. Let us consider the formal solution of Eq. (4.1.16):

$$U_{x,\mu}(\tau) = U_{x,\mu}(\tau_0) + \int_{\tau_0}^{\tau} d\tau' \left(i\,H_\mu(\tau')\,U_{x,\mu}(\tau')\right). \tag{4.1.27}$$

We can apply a time-independent left transformation $\Omega_x \in \text{SU}(N_c)$ and a time-independent right transformation $\Omega_x^\dagger$ to both sides obtaining

$$\Omega_x U_{x,\mu}(\tau)\,\Omega_{x+\mu}^\dagger = \Omega_x U_{x,\mu}(\tau_0)\,\Omega_{x+\mu}^\dagger$$
$$+ i\int_{\tau_0}^{\tau} d\tau'\left(\Omega_x\,H_\mu(\tau')\,\Omega_x^\dagger\Omega_x\,U_{x,\mu}(\tau')\,\Omega_{x+\mu}^\dagger\right), \tag{4.1.28}$$

where we inserted the product $\Omega_x^\dagger\Omega_x = \mathbf{1}$ between H and U. This expression informs us that if we want the transformation property of the links in Eq. (3.1.14) to hold for every τ, H has to be a locally transforming object, meaning that it transforms according to

$$T_\Omega H = \Omega_x H\,\Omega_x^\dagger. \tag{4.1.29}$$

Given the properties just discussed, H can be modeled with an L-CNN, in particular by taking from the output tuple $(\mathcal{U}', \mathcal{W}')$ the objects of type $\mathcal{W}'$, since they follow the same gauge transformation of H, namely Eq. (3.2.1). The approach we propose to train such an L-CNN is to generate a dataset with initial gauge link configurations $U_{x,\mu,0}^i = U_{x,\mu}^i(\tau_0)$ and target output configurations $\tilde{U}_{x,\mu,T}^i$, and then make use of the loss function

$$\mathcal{L}(\theta) = \frac{1}{N_{\text{samples}}} \sum_{x,\mu,i} \|\tilde{U}_{x,\mu,T} - U_{x,\mu}^i(T)\|^2, \tag{4.1.30}$$

to minimize the distance between the target configurations and the predicted configurations $U_{x,\mu}^i(T)$. We use the Frobenius norm to compute matrix distances. For a matrix $A \in \mathbb{C}^{n\times n}$ with elements a_{lm} it is defined as

$$\|A\|^2 = \sum_{l,m=1}^{n} |a_{lm}|^2. \tag{4.1.31}$$

In order to perform the evolution of the links, we need to solve the integral in Eq. (4.1.27). Standard ODE integrators are not suited to compute trajectories on the $SU(N_c)$ manifold. Instead, we resort to the exponential map, which we apply iteratively according to

$$U^i_{x,\mu}(\tau_{j+1}) = \exp\left(iH_\mu[U^i(\tau_j), \theta, \tau_j]\Delta\tau\right) U^i_{x,\mu}(\tau_j). \tag{4.1.32}$$

Since H is traceless and Hermitian, we can view it as an $\mathfrak{su}(N_c)$ algebra element, which the exponential map projects onto $SU(N_c)$, thus guaranteeing that the links are flowed on the $SU(N_c)$ manifold.

4.2   Single SU(2) link toy model

We test the framework delineated in the previous section on a toy model consisting of one SU(2) link governed by the action

$$S[U] = -\mathrm{Re}\,\mathrm{Tr}(U^2). \tag{4.2.1}$$

The input configurations are randomly distributed in such a way that the SU(2) manifold is sufficiently well covered, as shown in the left plot of Fig. 4.1. In order to get the target configurations, we flow the initial conditions according to the action. This can be done introducing an infinitesimal variation of the link

$$\delta U = i\delta A\, U, \tag{4.2.2}$$

and calculating the corresponding variation of the action

$$\begin{aligned}
S[U+\delta U] &= -\mathrm{Re}\,\mathrm{Tr}((U+\delta U)^2) \\
&= -\mathrm{Re}\,\mathrm{Tr}(U^2 + \delta U\, U + U\delta U) + O(\delta U^2) \\
&= -\mathrm{Re}\,\mathrm{Tr}(U^2 + 2\delta U\, U) + O(\delta U^2) \\
&= S[U] + \delta S[U,\delta U] + O(\delta U^2),
\end{aligned} \tag{4.2.3}$$

which can be written explicitly as

$$\begin{aligned}
\delta S[U,\delta U] &= -2\mathrm{Re}\,\mathrm{Tr}(\delta U\, U) \\
&= -2\mathrm{Re}\,\mathrm{Tr}(i\delta A\, U^2).
\end{aligned} \tag{4.2.4}$$

Given the three Lie algebra generators $t_a = \sigma_a/2$ and Pauli matrices σ_a

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{4.2.5}$$

we can write $\delta A = \delta A_a\, t_a$ and rework the variation of the action as

$$\begin{aligned}
\delta S[U,\delta A] &= -2\mathrm{Re}\left[\delta A_a\,\mathrm{Tr}(it_a\, U^2)\right] \\
&= 2\delta A_a\,\mathrm{Im}\left[\mathrm{Tr}(t_a\, U^2)\right],
\end{aligned} \tag{4.2.6}$$

where repeated indices are intended to be summed over and we also used $\text{Re}(iw) = -\text{Im}w$ for a complex number w. We have thus obtained the variation of the action in terms of the infinitesimal coefficients δA_a

$$\frac{\delta S[U]}{\delta A_a} = 2\text{Im}\left[\text{Tr}(t_a\, U^2)\right]. \tag{4.2.7}$$

This term can be interpreted as the force, and, analogously to Lüscher's setup provided by Eqs. (4.1.11) and (4.1.12), we can write the flow as

$$\frac{\mathrm{d}U}{\mathrm{d}\tau} = it_a\frac{\delta S[U]}{\delta A_a}U(\tau). \tag{4.2.8}$$

This differential equation can be solved analogously to Eq. (4.1.32) as

$$U(\tau_{j+1}) = \exp\left(it_a\frac{\delta S[U]}{\delta A_a}\Delta\tau\right)U(\tau_j) \tag{4.2.9}$$

in the limit of $\Delta\tau \to 0$.

Summarizing, the input of the network is a set of approximately uniformly distributed SU(2) matrices U_0^i and the target is the set of flowed matrices $\tilde{U}_T^i$. The action (4.2.1) is characterized by two minima, i.e. $\pm\mathbf{1}$, which are the elements towards which Eq. (4.2.9) flows the initial links. More precisely, the link flows towards $+\mathbf{1}$, which we call north pole, if $\text{Tr}\,U > 0$, while if $\text{Tr}\,U < 0$ the dynamics pushes the link closer to the south pole $-\mathbf{1}$. Links with $\text{Tr}\,U = 0$ are unstable stationary states. L-CNNs can reconstruct the dynamics of the links via Eq. (4.1.32), and given its similarity to Eq. (4.2.9), it is clear that the objective of the network is to learn the force.

A visualization of a dataset SU(2) is not immediate, but we adopt a projection onto the three-dimensional unit sphere with the following steps. First, we rewrite the matrix U as a linear combination of the identity and the Pauli matrices:

$$U = u_0\mathbf{1} + i\sigma_a u_a. \tag{4.2.10}$$

Since the Pauli matrices are traceless and satisfy the anticommutation relation

$$\{\sigma_i, \sigma_j\} = 2\delta_{ij}\mathbf{1}, \tag{4.2.11}$$

the coefficients u_k, with $k \in \{0, 1, 2, 3\}$ can be found by

$$u_0 = \frac{1}{2}\text{Tr}\,(U), \qquad u_i = \frac{1}{2i}\text{Tr}\,(U\sigma_i), \quad i \in \{1, 2, 3\}. \tag{4.2.12}$$

The group SU(2) is characterized by three degrees of freedom, which suggests that these coefficients are subject to a constraint. Indeed, the matrix U can

be rewritten via the exponential map and its power series as

$$
\begin{aligned}
U &= \exp\left(i A_a t_a\right) \\
&= \sum_{n=0}^{\infty} \frac{(i A_a t_a)^n}{n!} \\
&= \sum_{n=0}^{\infty} \frac{(-1)^n (A_a t_a)^{2n}}{2n!} + i \sum_{n=0}^{\infty} \frac{(-1)^n (A_b t_b)^{2n}}{(2n+1)!} A_c t_c \\
&= \sum_{n=0}^{\infty} \frac{(-1)^n (\frac{A}{2})^{2n}}{2n!} \mathbf{1} + i \sum_{n=0}^{\infty} \frac{(-1)^n (\frac{A}{2})^{2n+1}}{(2n+1)!} e_a \sigma_a \\
&= \cos\left(\frac{A}{2}\right) \mathbf{1} + i \sin\left(\frac{A}{2}\right) e_a \sigma_a,
\end{aligned}
\tag{4.2.13}
$$

where we have introduced $A = \sqrt{A_a A_a}$ and $e_a = A_a/A$ (see derivation e.g. in Appendix C of [142]). Most of the generators were simplified by the property $4 t_a^2 = \mathbf{1}$. We can now equate this result with the coefficients above, obtaining

$$
u_0 = \cos\left(\frac{A}{2}\right), \qquad u_a = \sin\left(\frac{A}{2}\right) e_a, \quad a \in \{1,2,3\}.
\tag{4.2.14}
$$

Summing over the square of the coefficients, it follows the group constraint

$$
u_0^2 + u_a u_a = 1.
\tag{4.2.15}
$$

Moreover, we can see that each parameter is bounded:

$$
-1 \le u_k \le 1 \quad k \in \{0,1,2,3\}.
\tag{4.2.16}
$$

We now slightly modify the parametrization by normalizing u_0, u_1 and u_2:

$$
\tilde{u}_j = u_j / \sqrt{u_0^2 + u_1^2 + u_2^2}, \quad j \in \{0,1,2\}.
\tag{4.2.17}
$$

With this choice, the triple $\{\tilde{u}_0, \tilde{u}_1, \tilde{u}_2\}$ lies on the three-dimensional unit sphere, and the remaining parameter u_3 is associated with a color map, as shown in Fig. 4.1. In the left plot, a subset of the initial matrices is visualized, and on the right plot we see the outcome under the effect of Eq. (4.1.32). Choosing $\tilde{u}_0$ as the vertical axis, allows to identify $U = +\mathbf{1}$ with the north pole and $U = -\mathbf{1}$ with the south pole, while matrices for which $\operatorname{Tr} U = 0$ lie on the equator.

Since for this problem configurations contain only one link, it is not clear how to define locally transforming objects that represent the input of an L-CNN. For this reason, we design a network which guarantees that the flowed matrices stay on the SU(2) manifold. The first step to realize it is the separation of U into its real and imaginary parts, which represent the
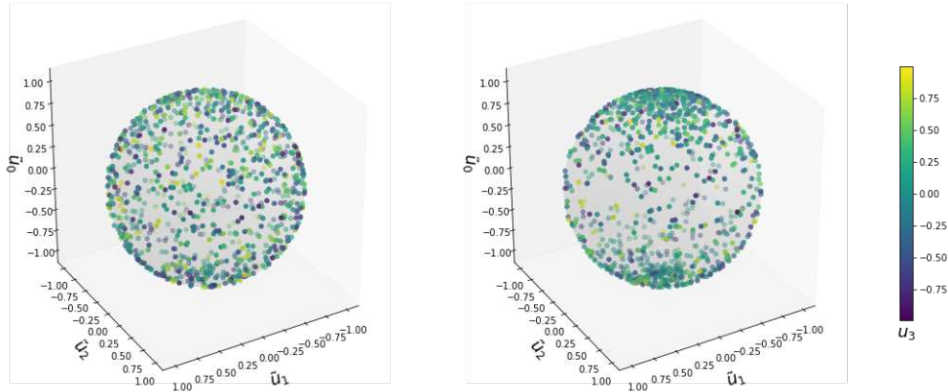
Figure 4.1: Projection on the 3D unit sphere of $1\,000$ matrices taken from the dataset. Left: visualization of the initial conditions U_0^i. Right: visualization of the target matrices $\tilde{U}_T^i$, resulting from the application of gradient flow to the initial conditions according to the action $S[U] = \operatorname{Re} \operatorname{Tr}(U^2)$ up to $T = 1$. Image from [3].

input of a dense network with real weights. Its output consists of eight real numbers that are assembled to form a complex matrix. This matrix is in general neither special nor unitary, so some kind of projection is needed. We make use of Eq. (3.2.42), which projects it onto the $\mathfrak{su}(2)$ Lie algebra. In order to get a group element, we finally apply the exponential map. This whole construction makes sure that the evolution stays on the SU(2) group. Let us point out that the action (4.2.1) is invariant under the transformation $U \to \Omega U \Omega^\dagger$ with $\Omega \in \mathrm{SU}(2)$, however this symmetry property is not implemented in the network just described and is ignored in the following for simplicity.

We provide now the details of our experiment. We choose the steps in gradient flow time to be sufficiently small for the discretized path (4.2.9) with the continuous flow (4.2.8), namely $\Delta\tau = 0.01$, and the final gradient flow time $T = 1$. The training set is made up by $50\,000$ samples and the batch size is 100. We train for 100 epochs using a learning rate of 10^{-3}. The best dense network found consists of four hidden layers with 16, 64, 32 and 16 nodes respectively, each with a learnable bias term and followed by a $\tanh(x)$ activation function, with the exception of the last linear layer. Once the learning process ends, a test on $4\,000$ samples is run within the same window of gradient flow time, e.g. $\tau \in [0, 1]$. Fig. 4.2 displays the outcome of this test. In the left plot, we show the projections on the three-dimensional unit sphere of the true trajectories obtained with Eq. (4.2.9), marked with blue, and of the predicted trajectories given by the NODE, in red. The two types of trajectories are visually indistinguishable, and indeed the MSE values reported as a function of flow time on the right plot

(b)   Loss as a function of flow time

(a)   True versus predicted trajectories
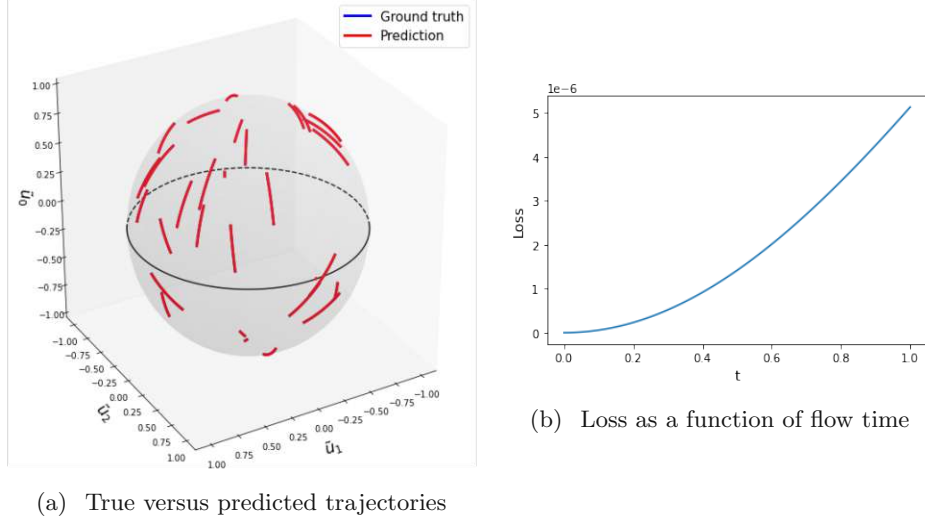
Figure 4.2: Test results. (a) Evolution of 30 samples projected on the 3D unit sphere. The true trajectories and the predicted ones are so close that the predictions cover the ground truth values. (b) Frobenius norm over the dataset as a function of flow time. Image from [3].

confirm how accurate the predictions are. On the other hand, the increase of the Frobenius norm raises the question of how the network performs if the final flow time is chosen outside the training interval. To answer this, an additional test is run on the same $4\,000$ samples flowed up to $\tau = 10$ without retraining the model. The results are shown in Fig. 4.3.

We observe a clear deterioration in the model's performance with a drop of roughly three orders of magnitude for the loss compared to the test run in the range $\tau \in [0, 1]$. A closer inspection of the test results leads us to the identification of two sources of the loss increase. One can be attributed to a specific sample, for which the predicted direction is wrong, as reported in Fig. 4.4. Provided that for $\tau \to \infty$ all matrices have flowed to one of the minima, the contribution of such mispredictions to the Frobenius norm $\mathcal{L}$ is given by

$$\lim_{\tau \to \infty} \Delta \mathcal{L} = \frac{N_{\text{mispred}}}{N_{\text{samples}}} \| \pm \mathbf{1} - (\mp \mathbf{1}) \|^2 = 8 \frac{N_{\text{mispred}}}{N_{\text{samples}}}. \tag{4.2.18}$$

In our case, $\Delta \mathcal{L}$ would be $2 \cdot 10^{-3}$, which is approximately the gain at $\tau = 10$ if we discard the problematic sample. The other significant contribution to the loss comes from samples for which the predictions overshoot the ground truth. The interpretation we give is that the learnt force is too high, which makes these samples flow faster than required towards a minimum. Both effects are associated with samples whose initial condition lies within a thin neighborhood of the equator, i.e. $\text{Tr}\, U \approx 0$. For these samples, the force at
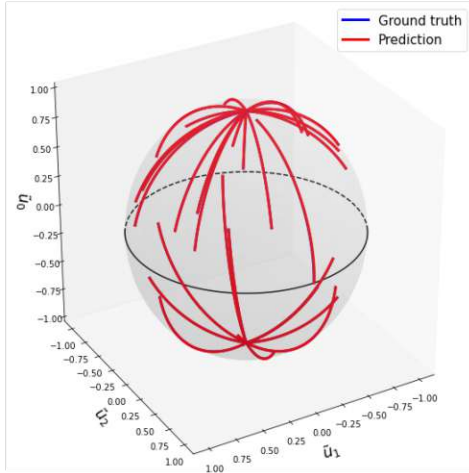
(a)   True versus predicted trajectories
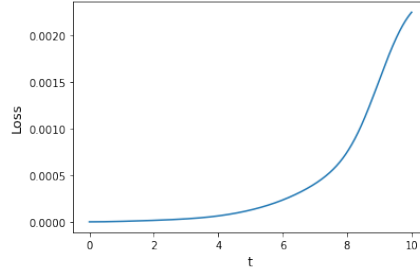
(b)   Loss as a function of flow time

Figure 4.3: Test results for final flow time $\tau = 10$ based on training up to $\tau = 1$. (a) Extrapolated evolution of 30 samples projected on the 3D unit sphere. The true and predicted trajectories lie too close to each other to be able to tell them apart. (b) Frobenius norm as a function of flow time. The loss increases significantly at larger times, by up to three orders of magnitude compared to the higher values reached in the original interval $\tau \in [0, 1]$. Image from [3].

the initial flow times can be very small, which is why the network struggles in determining the correct dynamics. Despite these flaws, the results are encouraging, considering that we employed a simple dense network which does not incorporate, for example, the symmetry of the dynamics with respect to the plane $\mathrm{Tr}\, U = 0$. More generally, the results can be improved with the use of a network that preserves the aforementioned symmetry of the system $U \to \Omega U \Omega^{\dagger}$ with $\Omega \in \mathrm{SU}(2)$.

4.3   Adjoint sensitivity method

A technical aspect of NODEs that still needs to be addressed is backpropagation. In machine learning contexts, it refers to the computation of the gradient of the loss function with respect to the model parameters. Backpropagation is a highly efficient implementation of the chain rule, but for NODEs it would require to store in memory the whole evolution of the system. To see this explicitly, we can consider the NODE for $x \in \mathbb{R}$

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x(t), \theta) \tag{4.3.1}$$

(b) Loss as a function of flow time

(a)   True versus predicted trajectory
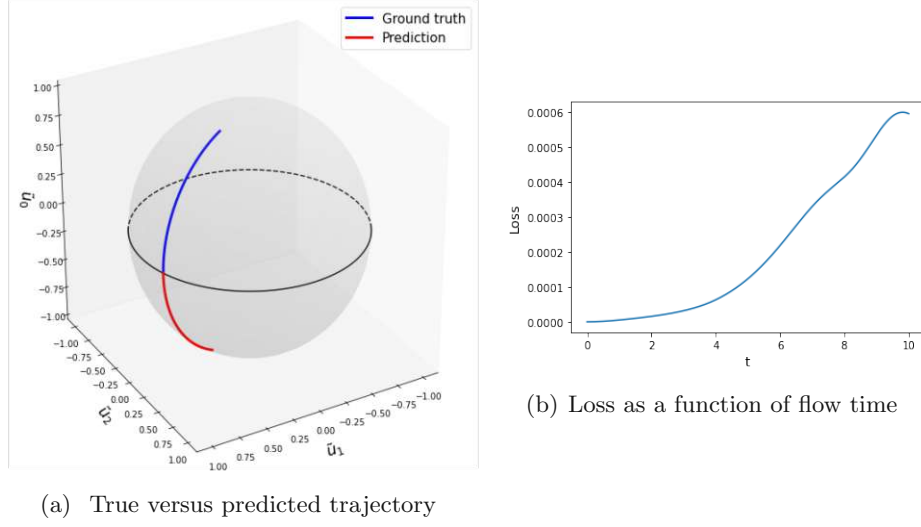
Figure 4.4: Mispredictions and their effect on the results. (a) We show that for one sample, the predicted trajectory goes in the opposite direction of the true trajectory, thus leading to an increase in the loss. (b) The Frobenius norm calculated on all other samples in the test set increases similarly to Fig. 4.3(b), but its value at the final flow time is smaller by about an order of magnitude.

with initial condition $x_0 = x(t_0)$. Formally, it is possible to solve it by

$$x(T) = x_0 + \int_{t_0}^{T} f(x(t), \theta) \, dt. \tag{4.3.2}$$

We can choose the loss function to be the MSE between the target final state $\tilde{x}_T$ and the predicted final state $x(T)$

$$\mathcal{L}(x(T)) = (\tilde{x}_T - x(T))^2. \tag{4.3.3}$$

The gradients are given by

$$\frac{d\mathcal{L}}{d\theta} = \frac{\partial \mathcal{L}}{\partial x(T)} \frac{dx(T)}{d\theta}, \tag{4.3.4}$$

but the term

$$\frac{dx(t)}{d\theta} = \int_{t_0}^{T} dt' \left(\frac{\partial f(x(t'), \theta)}{\partial \theta} + \frac{\partial f(x(t'), \theta)}{\partial x(t')} \frac{dx(t')}{d\theta} \right). \tag{4.3.5}$$

requires to store the whole evolution of the system in memory, which can quickly be saturated if the network's depth is increased. We have used the fact that $x(t) = x(t, \theta)$ for $t_0 < t < T$, therefore its derivative is

$$\frac{df}{d\theta} = \frac{\partial f}{\partial \theta} + \frac{\partial f}{\partial x} \frac{dx}{d\theta}. \tag{4.3.6}$$

In programs which incur memory saturation issues, it is common practice to trade off memory with computation. Here, the adjoint sensitivity method [140] serves the scope perfectly, as it computes the evolution backwards rather than saving it in memory. Historically, this method has encountered various applications ranging from meteorology [143] to geophysics [144].

We can formulate the adjoint sensitivity method as a constrained optimization problem using Lagrange multipliers [145]. We can rewrite Eq. (4.3.1) in the form of a constraint:

$$\dot{x} - f(x(t), \theta) = F(\dot{x}(t), x(t), \theta) = 0. \tag{4.3.7}$$

We introduced the notation $\dot{x}$ as an equivalent form of $\mathrm{d}x/\mathrm{d}t$. Our goal is to find $\arg\min_\theta \mathcal{L}(x(T))$ such that the constraint (4.3.7) remains satisfied. This can be fulfilled introducing the Lagrange multiplier $\lambda(t)$ and the modified loss function

$$\psi = \mathcal{L}(x(T)) - \int_{t_0}^{T} \lambda(t) F(\dot{x}(t), x(t), \theta) \mathrm{d}t. \tag{4.3.8}$$

Since $F = 0$, the gradients of the original loss function coincide with the gradients of the modified loss function:

$$\frac{\mathrm{d}\psi}{\mathrm{d}\theta} = \frac{\mathrm{d}\mathcal{L}(x(T))}{\mathrm{d}\theta}. \tag{4.3.9}$$

Therefore, we can compute the gradients $\mathrm{d}\mathcal{L}/\mathrm{d}\theta$ by taking the derivative of Eq. (4.3.8), obtaining

$$\frac{\mathrm{d}\psi}{\mathrm{d}\theta} = \frac{\partial\mathcal{L}}{\partial x(T)} \frac{\mathrm{d}x(T)}{\mathrm{d}\theta} - \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda F \mathrm{d}t. \tag{4.3.10}$$

We focus on the second term and rework it as follows:

$$
\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda F \, \mathrm{d}t &= \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda(\dot{x} - f) \, \mathrm{d}t \\
&= \frac{\mathrm{d}}{\mathrm{d}\theta} \left[\lambda(T)x(T) - \lambda(t_0)x(t_0) - \int_{t_0}^{T} (\dot{\lambda}x + \lambda f) \, \mathrm{d}t \right] \\
&= \lambda(T)\frac{\mathrm{d}x(T)}{\mathrm{d}\theta} - \int_{t_0}^{T} \left(\dot{\lambda} + \lambda\frac{\partial f}{\partial x} \right) \frac{\mathrm{d}x}{\mathrm{d}\theta} \, \mathrm{d}t - \int_{t_0}^{T} \lambda\frac{\partial f}{\partial\theta} \mathrm{d}t,
\end{aligned}
\tag{4.3.11}
$$

where we integrated by parts in the second equality and used Eq. (4.3.6) to get to the third equality. We stress that λ does not depend on the network's parameters θ and that

$$\frac{\mathrm{d}x(t_0)}{\mathrm{d}\theta} = \frac{\mathrm{d}x_0}{\mathrm{d}\theta} = 0. \tag{4.3.12}$$

The result for the gradients reads

$$\frac{\mathrm{d}\psi}{\mathrm{d}\theta} = \left(\frac{\partial \mathcal{L}}{\mathrm{d}x(T)} - \lambda(T)\right)\frac{\mathrm{d}x(T)}{\mathrm{d}\theta} + \int_{t_0}^{T}\left(\dot{\lambda} + \lambda\frac{\partial f}{\partial x}\right)\frac{\mathrm{d}x}{\mathrm{d}\theta}\,\mathrm{d}t + \int_{t_0}^{T}\lambda\frac{\partial f}{\partial \theta}\mathrm{d}t. \tag{4.3.13}$$

In order to get rid of the dependence on $\mathrm{d}x/\mathrm{d}\theta$, we impose the following equations

$$\begin{cases} \lambda(T) = \dfrac{\partial \mathcal{L}}{\partial x(T)}, & (4.3.14) \\[2ex] \dot{\lambda}(t) = -\lambda(t)\dfrac{\partial f(x(t),\theta)}{\partial x(t)}, & (4.3.15) \end{cases}$$

which define respectively the initial condition and the equation of motion for the adjoint λ. The remaining term in Eq. (4.3.13) yields the gradients we were looking for, i.e.

$$\frac{\mathrm{d}\mathcal{L}(x(T))}{\mathrm{d}\theta} = -\int_{T}^{t_0}\lambda(t)\frac{\partial f(x(t),\theta)}{\partial \theta}\mathrm{d}t, \tag{4.3.16}$$

without any dependence on $\mathrm{d}x/\mathrm{d}\theta$. We point out that in case the function f also depends explicitly on the flow time t, the derivation and the results would be identical.

In light of the above results, the implementation of the adjoint sensitivity method consists of the following steps. First, the NODE (4.3.1) is solved with the forward pass by using an ODE integrator such as the one given in Eq. (4.1.32). At $t = T$, we evaluate the initial condition of the adjoint in Eq. (4.3.14), $\partial \mathcal{L}/\partial x(T)$. With a standard ODE solver such as Runge-Kutta, we integrate Eqs. (4.3.1) and (4.3.15) backwards from T to t_0. In this way, we obtain the terms $\lambda(t)$ and $\partial f(x(t),\theta)/\partial \theta$ that contribute to the gradients in Eq. (4.3.16) without the need of using backpropagation and storing in memory the whole evolution of the system.

4.3.1   Calculation for U(1)

In principle, the adjoint sensitivity method just presented can be directly applied also in the context of groups, but the backward evolution of the system would not fulfill the group constraints in general, therefore we extend the derivation of the adjoint sensitivity method such that the constraints are respected by construction. We do it first for the U(1) group in order to develop some intuition for the calculation for $\mathrm{SU}(N_c)$. We can view a U(1) element as a complex variable z living on the unit circle in the complex plane, which in polar coordinates can be written as

$$z = \mathrm{e}^{i\varphi}, \qquad \varphi \in [0, 2\pi). \tag{4.3.17}$$

We define the complex conjugate of z:

$$\bar{z} = e^{-i\varphi}, \tag{4.3.18}$$

which acts as the inverse group element of z:

$$z\bar{z} = 1. \tag{4.3.19}$$

We aim to solve the following neural gradient flow equation:

$$\frac{\mathrm{d}z}{\mathrm{d}t} = ig(z(t), \bar{z}(t), \theta)\, z(t), \tag{4.3.20}$$

where $g \in \mathbb{R} \implies g = \bar{g}$, with initial condition $z_0 = z(t_0)$ and target final state $\tilde{z}_T$. This NODE is very similar to the one used in the real case, Eq. (4.3.1), except we have provided a more specific expression to the right hand side. This choice is justified because it guarantees that the constraint (4.3.19) is satisfied during the flow:

$$\frac{\mathrm{d}}{\mathrm{d}t}(z\bar{z}) = \dot{z}\bar{z} + z\dot{\bar{z}} = (igz)\bar{z} - z(i\bar{g}\bar{z}) = i(g - \bar{g}) = 0, \tag{4.3.21}$$

where the complex conjugate of Eq. (4.3.20) is

$$\frac{\mathrm{d}\bar{z}}{\mathrm{d}t} = -i\bar{g}\bar{z}. \tag{4.3.22}$$

The function g can be modeled by a neural network, whose input is the variable z. In complex analysis, though, g has to be considered as a function of $\bar{z}$ as well, which will be crucial when taking derivatives. The loss function is chosen to be the real-valued MSE

$$\mathcal{L}(z(T), \bar{z}(T)) = |z(T) - \tilde{z}_T|^2. \tag{4.3.23}$$

The gradients contain two terms due to the dependence on z and $\bar{z}$:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta} = \frac{\partial\mathcal{L}}{\partial z(T)}\frac{\mathrm{d}z(T)}{\mathrm{d}\theta} + \frac{\partial\mathcal{L}}{\partial \bar{z}(T)}\frac{\mathrm{d}\bar{z}(T)}{\mathrm{d}\theta}. \tag{4.3.24}$$

The partial derivatives with respect to z and $\bar{z}$ are called Wirtinger derivatives and are related to partial derivatives with respect to $u = \mathrm{Re}z$ and $v = \mathrm{Im}z$ as follows:

$$\begin{cases} \dfrac{\partial}{\partial z} = \dfrac{1}{2}\left(\dfrac{\partial}{\partial u} - i\dfrac{\partial}{\partial v}\right), & (4.3.25) \\[3mm] \dfrac{\partial}{\partial \bar{z}} = \dfrac{1}{2}\left(\dfrac{\partial}{\partial u} + i\dfrac{\partial}{\partial v}\right). & (4.3.26) \end{cases}$$

The NODE (4.3.20) is reworked to obtain the constraint

$$\dot{z} - igz = G = 0. \tag{4.3.27}$$

As discussed in the real case, what we are trying to solve is a constrained minimization problem. The Lagrange multiplier technique is adapted to the complex case by including the complex conjugate in the following way:

$$\psi = \mathcal{L}(z(T), \bar{z}(T)) - \int_{t_0}^{T} \left(\lambda G + \bar{\lambda}\bar{G} \right) \mathrm{d}t. \tag{4.3.28}$$

Because of the constraint (4.3.27), the gradients of ψ coincide with the gradients of $\mathcal{L}$ and are given by

$$\frac{\mathrm{d}\mathcal{L}(z(T), \bar{z}(T))}{\mathrm{d}\theta} = \frac{\mathrm{d}\psi}{\mathrm{d}\theta} = \frac{\partial\mathcal{L}}{\partial z(T)} \frac{\mathrm{d}z(T)}{\mathrm{d}\theta} + \frac{\partial\mathcal{L}}{\partial \bar{z}(T)} \frac{\mathrm{d}\bar{z}(T)}{\mathrm{d}\theta}$$
$$- \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \left(\lambda G + \bar{\lambda}\bar{G} \right) \mathrm{d}t. \tag{4.3.29}$$

We notice the following useful property, which is a consequence of Eq. (4.3.19):

$$\frac{\mathrm{d}}{\mathrm{d}\theta}(z\bar{z}) = 0 \implies z\frac{\mathrm{d}\bar{z}}{\mathrm{d}\theta} = -\bar{z}\frac{\mathrm{d}z}{\mathrm{d}\theta}. \tag{4.3.30}$$

A key consideration is that $z(t) = z(t, \theta)$, such that the chain rule leads to a result similar to Eq. (4.3.6):

$$\frac{\mathrm{d}g}{\mathrm{d}\theta} = \frac{\partial g}{\partial z}\frac{\mathrm{d}z}{\mathrm{d}\theta} + \frac{\partial g}{\partial \bar{z}}\frac{\mathrm{d}\bar{z}}{\mathrm{d}\theta} + \frac{\partial g}{\partial \theta}. \tag{4.3.31}$$

Elaborating the derivative of the integral in (4.3.29), we obtain

$$\frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda G \, \mathrm{d}t + \text{c.c.} = \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda(\dot{z} - igz) \, \mathrm{d}t + \text{c.c.}$$
$$= \frac{\mathrm{d}}{\mathrm{d}\theta} \left[\lambda(T)z(T) - \lambda(t_0)z(t_0) - \int_{t_0}^{T} (\dot{\lambda}z + i\lambda gz)\mathrm{d}t \right] + \text{c.c.}$$
$$= \lambda(T)\frac{\mathrm{d}z(T)}{\mathrm{d}\theta} - \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} (\dot{\lambda}z + i\lambda gz) \, \mathrm{d}t + \text{c.c.}, \tag{4.3.32}$$

where we performed an integration by parts and indicated with "c.c." the complex conjugate of the entire expressions. The derivative of the integrand in the last line yields

$$\frac{\mathrm{d}}{\mathrm{d}\theta}(\dot{\lambda}z + i\lambda gz) = \left[\dot{\lambda} + i\lambda\left(\frac{\partial g}{\partial z}z + g \right) \right]\frac{\mathrm{d}z}{\mathrm{d}\theta} + i\lambda\left(\frac{\partial g}{\partial \bar{z}}z\frac{\mathrm{d}\bar{z}}{\mathrm{d}\theta} + \frac{\partial g}{\partial \theta}z \right)$$
$$= \left[\dot{\lambda} + i\lambda\left(\frac{\partial g}{\partial z}z - \frac{\partial g}{\partial \bar{z}}\bar{z} + g \right) \right]\frac{\mathrm{d}z}{\mathrm{d}\theta} + i\lambda\frac{\partial g}{\partial \theta}z, \tag{4.3.33}$$

where we made use of Eq. (4.3.31) and exploited the property in Eq. (4.3.30) to transform the term with $\mathrm{d}z/\mathrm{d}\theta$ into a term with $\mathrm{d}\bar{z}/\mathrm{d}\theta$. This result can be substituted in Eq. (4.3.32), obtaining

$$
\frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \lambda G \, \mathrm{d}t + \text{c.c.} = \lambda(T)\frac{\mathrm{d}z(T)}{\mathrm{d}\theta}
$$
$$
- \int_{t_0}^{T} \left[\dot{\lambda} + i\lambda \left(\frac{\partial g}{\partial z}z - \frac{\partial g}{\partial \bar{z}}\bar{z} + g \right) \right] \frac{\mathrm{d}z}{\mathrm{d}\theta} \, \mathrm{d}t - i \int_{t_0}^{T} \lambda \frac{\partial g}{\partial \theta}z \, \mathrm{d}t + \text{c.c.} \quad (4.3.34)
$$

The gradients in Eq. (4.3.29) become

$$
\frac{\mathrm{d}\psi}{\mathrm{d}\theta} = \left(\frac{\partial \mathcal{L}}{\mathrm{d}z(T)} - \lambda(T) \right) \frac{\mathrm{d}z(T)}{\mathrm{d}\theta}
$$
$$
+ \int_{t_0}^{T} \left[\dot{\lambda} + i\lambda \left(\frac{\partial g}{\partial z}z - \frac{\partial g}{\partial \bar{z}}\bar{z} + g \right) \right] \frac{\mathrm{d}z}{\mathrm{d}\theta} \, \mathrm{d}t
$$
$$
+ i \int_{t_0}^{T} \lambda \frac{\partial g}{\partial \theta}z \, \mathrm{d}t + \text{c.c.} \quad (4.3.35)
$$

Similarly to the real case, we get rid of the dependence on $\mathrm{d}z/\mathrm{d}\theta$ by imposing

$$
\begin{cases}
\lambda(T) = \dfrac{\partial \mathcal{L}}{\partial z(T)}, & (4.3.36) \\[2mm]
\dot{\lambda} + i\lambda g = -i\lambda \left(\dfrac{\partial g}{\partial z}z - \dfrac{\partial g}{\partial \bar{z}}\bar{z} \right) = 2\lambda \operatorname{Im}\left(\dfrac{\partial g}{\partial z}z \right). & (4.3.37)
\end{cases}
$$

The first equation sets the initial condition for the adjoint, while the second is the ODE for the adjoint. With these choices, the corresponding complex conjugate also vanish, hence no term involving $\mathrm{d}\bar{z}/\mathrm{d}\theta$ survives. The remaining term in the last line of Eq. (4.3.35) and its complex conjugate corresponds to the gradients:

$$
\frac{\mathrm{d}\mathcal{L}(z(T), \bar{z}(T))}{\mathrm{d}\theta} = -i \int_{T}^{t_0} \frac{\partial g}{\partial \theta}(\lambda z - \bar{\lambda}\bar{z}) \, \mathrm{d}t = 2 \int_{T}^{t_0} \frac{\partial g}{\partial \theta} \operatorname{Im}(\lambda z) \, \mathrm{d}t. \quad (4.3.38)
$$

The gradients are manifestly real, as they are supposed to be. It is also possible to write a more compact solution by introducing the quantity $w = \lambda z$. Its time derivative is given by

$$
\dot{w} = \dot{\lambda}z + \lambda \dot{z} = \dot{\lambda}z + \lambda(igz) = z(\dot{\lambda} + i\lambda g). \quad (4.3.39)
$$

Substituting in Eq. (4.3.36), (4.3.37) and (4.3.38), the initial condition and the equation of motion for this modified adjoint are

$$
\begin{cases}
w(T) = \dfrac{\partial \mathcal{L}}{\partial z(T)}z(T), & (4.3.40) \\[2mm]
\dot{w} = 2w \operatorname{Im}\left(\dfrac{\partial g}{\partial z}z \right), & (4.3.41)
\end{cases}
$$
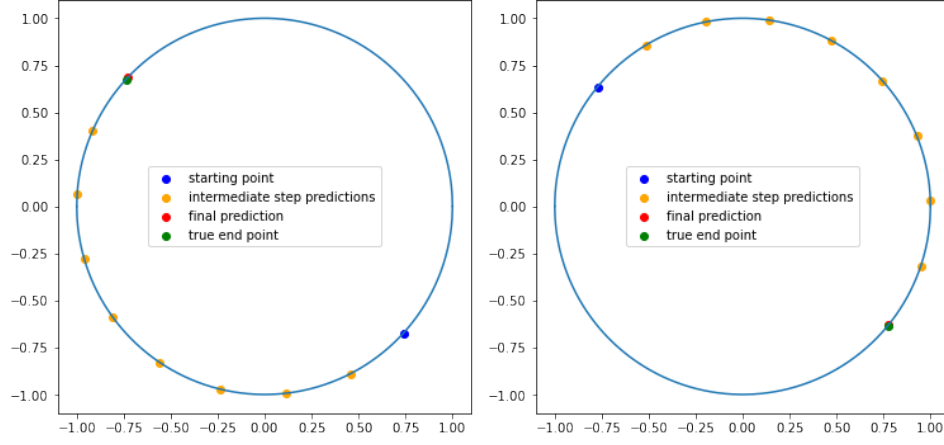
Figure 4.5: Solution of an NODE of a U(1) element with adjoint sensitivity method. We display the successful reconstruction of the time evolution of a complex variable constrained on the unit circle using the adjoint sensitivity method for two samples of our dataset. Since the starting point (blue) and the end point (green) are separated by a constant phase of π, the solution of the NODE is a complex variable rotating around the origin at fixed angular momentum. The intermediate flow times are equally spaced, which is why the predictions at such flow times (orange) are separated by constant rotations. The final prediction (red) lies close to the ground truth.

and the gradients become

$$\frac{\mathrm{d}\mathcal{L}(z(T), \bar{z}(T))}{\mathrm{d}\theta} = 2 \int_T^{t_0} \frac{\partial g}{\partial \theta} \operatorname{Im} w \, \mathrm{d}t. \tag{4.3.42}$$

The implementation of the adjoint sensitivity method for U(1) is almost identical to the case with real variables, except for the ODE solver. In order to stay in the group while integrating backwards, we use a solver that employs the exponential in a way similar to Eq. (4.1.32)

$$z(t_j) = \exp\left(-ig(z(t_{j+1}), \theta)\Delta t\right) z(t_{j+1}). \tag{4.3.43}$$

The exponent is purely imaginary, such that the exponential acts as a rotation of $z(t_j)$.

A test of this derivation is run on a simple situation in which each input z_0^i is separated by a constant phase $\Delta\varphi = \pi$ from the corresponding output $\tilde{z}_T^i$. In Fig. 4.5, we show the solution provided by a simple network. Specifically, the function g is modeled with a dense network with two hidden layers each featuring 20 nodes and followed by a tanh activation function. The solution of the NODE is a constant motion, such that z rotates at constant angular momentum around the origin. Given that the flow time steps Δt are

equal, the predictions at intermediate flow times have to be separated each by an approximately constant phase, as is the case in Fig. 4.5. We notice that the network is free to choose a clockwise or anticlockwise rotation for this problem. This ambiguity can be eliminated by providing intermediate values of the evolution and adding the MSE of the corresponding predictions to the loss function.

4.3.2 Calculation for $\mathrm{SU}(N_c)$

Let us now derive the adjoint sensitivity method in $\mathrm{SU}(N_c)$. The NODE we are dealing with is the neural gradient flow (4.1.16), where the indices x and μ will be dropped for simplicity:

$$\frac{\mathrm{d}U}{\mathrm{d}\tau} = iH(U(\tau), \bar{U}(\tau), \theta)\, U(\tau). \tag{4.3.44}$$

Notice how we define H as explicitly dependent on $\bar{U}$. This is because we will treat the elements of each matrix as independent complex numbers, in a way that resembles the U(1) case. The initial condition of the dynamics are $U_0 = U(\tau = 0)$, and the target final state $\tilde{U}_T$. We choose as loss function the Frobenius norm

$$\mathcal{L}(U(T), \bar{U}(T)) = \|U(T) - \tilde{U}_T\|^2, \tag{4.3.45}$$

defined by Eq. (4.1.31). The gradients depend on U and $\bar{U}$:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\theta} = \frac{\partial \mathcal{L}}{\partial U_{ij}(T)} \frac{\mathrm{d}U_{ij}(T)}{\mathrm{d}\theta} + \frac{\partial \mathcal{L}}{\partial \bar{U}_{ij}(T)} \frac{\mathrm{d}\bar{U}_{ij}(T)}{\mathrm{d}\theta}. \tag{4.3.46}$$

The partial derivatives can be defined analogously to the Wirtinger derivatives in Eqs. (4.3.25) and (4.3.26). We proceed as in the two cases previously examined, reworking the NODE in the form of the constraint

$$\dot{U} - H(U(t), \theta) = \mathcal{H}(\dot{U}(t), U(t), \theta) = 0. \tag{4.3.47}$$

In general, $\mathcal{H} \in \mathbb{C}^{N_c \times N_c}$. We introduce the Lagrange multiplier $\Lambda \in \mathbb{C}^{N_c \times N_c}$ and the modified loss function

$$\Psi = \mathcal{L}(U(T), \bar{U}(T)) - \int_{t_0}^{T} \mathrm{Tr}\left(\Lambda\mathcal{H} + \overline{\Lambda\mathcal{H}}\right) \mathrm{d}\tau$$

$$= \mathcal{L}(U(T), \bar{U}(T)) - \int_{t_0}^{T} \left(\Lambda_{ij}\mathcal{H}_{ji} + \bar{\Lambda}_{ij}\bar{\mathcal{H}}_{ji}\right) \mathrm{d}\tau. \tag{4.3.48}$$

The trace operation guarantees that the result of the integral is a scalar. The constraint (4.3.47) gives the following equalities for the gradients:

$$\frac{\mathrm{d}\mathcal{L}(U(T), \bar{U}(T))}{\mathrm{d}\theta} = \frac{\mathrm{d}\Psi}{\mathrm{d}\theta} = \frac{\partial \mathcal{L}}{\partial U(T)} \frac{\mathrm{d}U(T)}{\mathrm{d}\theta} + \frac{\partial \mathcal{L}}{\partial \bar{U}(T)} \frac{\mathrm{d}\bar{U}(T)}{\mathrm{d}\theta}$$

$$- \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \left(\Lambda\mathcal{H} + \bar{\Lambda}\bar{\mathcal{H}}\right) \mathrm{d}\tau. \tag{4.3.49}$$

Calculating the derivative of the integral in the last line of the above equation, we obtain

$$
\frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \mathrm{Tr}\,(\Lambda \mathcal{H})\,\mathrm{d}t = \frac{\mathrm{d}}{\mathrm{d}\theta} \int_{t_0}^{T} \mathrm{Tr}\left(\Lambda(\dot{U} - iHU)\right) \mathrm{d}\tau
$$

$$
= \frac{\mathrm{d}}{\mathrm{d}\theta}\left[\mathrm{Tr}\,(\Lambda(T)U(T)) - \mathrm{Tr}\,(\Lambda(t_0)U(t_0)) - \int_{t_0}^{T} \mathrm{Tr}\left(\dot{\Lambda}U + i\Lambda HU\right)\mathrm{d}\tau\right]
$$

$$
= \Lambda_{ji}(T)\frac{\mathrm{d}U_{ij}(T)}{\mathrm{d}\theta} - \frac{\mathrm{d}}{\mathrm{d}\theta}\int_{t_0}^{T}\mathrm{Tr}\left(\dot{\Lambda}U + i\Lambda HU\right)\mathrm{d}\tau, \tag{4.3.50}
$$

and a similar result for the complex conjugate. Noticing that a formula similar to Eq. (4.3.31) holds also for H, we can write

$$
\frac{\mathrm{d}}{\mathrm{d}\theta}(\dot{\Lambda}_{ij}U_{ji} + i\Lambda_{ik}H_{kj}U_{ji} + \dot{\bar{\Lambda}}_{ij}\bar{U}_{ji} - i\bar{\Lambda}_{ik}\bar{H}_{kj}\bar{U}_{ji})
$$

$$
= \dot{\Lambda}_{ij}\frac{\mathrm{d}U_{ji}}{\mathrm{d}\theta} + i\Lambda_{ik}\left(\frac{\partial H_{kj}}{\partial U_{lm}}\frac{\mathrm{d}U_{lm}}{\mathrm{d}\theta}U_{ji} + \frac{\partial H_{kj}}{\partial \bar{U}_{lm}}\frac{\mathrm{d}\bar{U}_{lm}}{\mathrm{d}\theta}U_{ji} + \frac{\partial H_{kj}}{\partial \theta}U_{ji} + H_{kj}\frac{\mathrm{d}U_{ji}}{\mathrm{d}\theta}\right)
$$

$$
+ \dot{\bar{\Lambda}}_{ij}\frac{\mathrm{d}\bar{U}_{ji}}{\mathrm{d}\theta} - i\bar{\Lambda}_{ik}\left(\frac{\partial \bar{H}_{kj}}{\partial U_{lm}}\frac{\mathrm{d}U_{lm}}{\mathrm{d}\theta}\bar{U}_{ji} + \frac{\partial \bar{H}_{kj}}{\partial \bar{U}_{lm}}\frac{\mathrm{d}\bar{U}_{lm}}{\mathrm{d}\theta}\bar{U}_{ji} + \frac{\partial \bar{H}_{kj}}{\partial \theta}\bar{U}_{ji} + \bar{H}_{kj}\frac{\mathrm{d}\bar{U}_{ji}}{\mathrm{d}\theta}\right)
$$

$$
= \left(\dot{\Lambda}_{ij} + i\Lambda_{ml}\frac{\partial H_{lk}}{\partial U_{ji}}U_{km} - i\bar{\Lambda}_{ml}\frac{\partial \bar{H}_{lk}}{\partial U_{ji}}\bar{U}_{km} + i\Lambda_{ik}H_{kj}\right)\frac{\mathrm{d}U_{ji}}{\mathrm{d}\theta} + i\Lambda_{ik}\frac{\partial H_{kj}}{\partial \theta}U_{ji}
$$

$$
+ \left(\dot{\bar{\Lambda}}_{ij} - i\bar{\Lambda}_{ml}\frac{\partial \bar{H}_{lk}}{\partial \bar{U}_{ji}}\bar{U}_{km} + i\Lambda_{ml}\frac{\partial H_{lk}}{\partial \bar{U}_{ji}}U_{km} - i\bar{\Lambda}_{ik}\bar{H}_{kj}\right)\frac{\mathrm{d}\bar{U}_{ji}}{\mathrm{d}\theta} - i\bar{\Lambda}_{ik}\frac{\partial \bar{H}_{kj}}{\partial \theta}\bar{U}_{ji}. \tag{4.3.51}
$$

In the third equality, we have repositioned some terms and changed indices in order to be able to factor out the derivatives of U and $\bar{U}$ with respect to the parameters θ. We can substitute this result in Eq. (4.3.50) and its complex conjugate, obtaining

$$
\frac{\mathrm{d}}{\mathrm{d}\theta}\int_{t_0}^{T}\mathrm{Tr}\,(\Lambda\mathcal{H})\,\mathrm{d}\tau + \text{c.c.} = \Lambda_{ji}(T)\frac{\mathrm{d}U_{ij}(T)}{\mathrm{d}\theta}
$$

$$
- \int_{t_0}^{T}\left[\dot{\Lambda}_{ij} + i\Lambda_{ik}H_{kj} - 2\,\mathrm{Im}\left(\frac{\partial H_{lk}}{\partial U_{ji}}U_{km}\Lambda_{ml}\right)\right]\frac{\mathrm{d}U_{ji}}{\mathrm{d}\theta}\,\mathrm{d}\tau
$$

$$
- i\int_{t_0}^{T}\mathrm{Tr}\left(\Lambda\frac{\partial H}{\partial \theta}U\right)\mathrm{d}\tau + \text{c.c.} \tag{4.3.52}
$$

Finally, we can rewrite the gradients in Eq. (4.3.49) as

$$
\frac{\mathrm{d}\Psi}{\mathrm{d}\theta} = \left(\frac{\partial \mathcal{L}}{\mathrm{d}U_{ij}(T)} - \Lambda_{ji}(T)\right)\frac{\mathrm{d}U_{ij}(T)}{\mathrm{d}\theta}
$$

$$
+ \int_{t_0}^{T}\left[\dot{\Lambda}_{ij} + i\Lambda_{ik}H_{kj} - 2\,\mathrm{Im}\left(\frac{\partial H_{lk}}{\partial U_{ji}}U_{km}\Lambda_{ml}\right)\right]\frac{\mathrm{d}U_{ji}}{\mathrm{d}\theta}\,\mathrm{d}\tau
$$

$$
+ i\int_{t_0}^{T}\mathrm{Tr}\left(\Lambda\frac{\partial H}{\partial \theta}U\right)\mathrm{d}\tau + \text{c.c.} \tag{4.3.53}
$$

We remove the dependence on $\mathrm{d}U/\mathrm{d}\theta$ and its complex conjugate imposing the following equations for the adjoint:

$$\begin{cases} \Lambda_{ij}(T) = \dfrac{\partial \mathcal{L}}{\partial U_{ji}(T)}, & (4.3.54) \\[3mm] \dot{\Lambda}_{ij} + i\Lambda_{ik}H_{kj} = 2\,\mathrm{Im}\left(\dfrac{\partial H_{lk}}{\partial U_{ji}}U_{km}\Lambda_{ml}\right), & (4.3.55) \end{cases}$$

which implies that the contribution of the complex conjugates vanishes. With the terms left, we can compute the gradients:

$$\frac{\mathrm{d}\mathcal{L}(U(T),\bar{U}(T))}{\mathrm{d}\theta} = 2\int_T^{t_0}\mathrm{Im}\,\mathrm{Tr}\left(\frac{\partial H}{\partial\theta}U\Lambda\right)\mathrm{d}\tau. \qquad (4.3.56)$$

In analogy to U(1), we can introduce the quantity $W = U\Lambda$. The initial condition and the equation of motion of such a combination is

$$\begin{cases} W_{ij}(T) = U_{ik}(T)\dfrac{\partial \mathcal{L}}{\partial U_{jk}(T)}, & (4.3.57) \\[3mm] \dot{W}_{ij} + i[W,H]_{ij} = 2U_{ik}\mathrm{Im}\left(\dfrac{\partial H_{lm}}{\partial U_{jk}}W_{ml}\right), & (4.3.58) \end{cases}$$

and the gradients are easily expressed as

$$\frac{\mathrm{d}\mathcal{L}(U(T),\bar{U}(T))}{\mathrm{d}\theta} = 2\int_T^{t_0}\mathrm{Im}\,\mathrm{Tr}\left(\frac{\partial H}{\partial\theta}W\right)\mathrm{d}\tau. \qquad (4.3.59)$$

The implementation of the above equations is structured similarly to the real case, the main difference being the solver employed for the backward integration: in $\mathrm{SU}(N_c)$, an appropriate choice is represented by Eq. (4.1.32), adapted to evolve backwards in time. We have checked that this method provides approximately the right gradients for the SU(2) toy model by comparing the gradients obtained via standard backpropagation with the ones given by the adjoint sensitivity method, with the highest relative errors of the order of 10^{-3}. Currently, the backward dynamics is integrated with a standard Runge-Kutta algorithm, but more precise results can be obtained with a Runge-Kutta method suited for Lie groups [146]. We finally report that the adjoint sensitivity method in $\mathrm{SU}(N_c)$ has also been solved with a different approach in [94].

Chapter 5

Conclusions

In this dissertation, we have shown the benefits of imposing global and local symmetries on neural network architectures in applications to lattice field theory. In particular, we have initially dealt with the application of neural networks to a physical system characterized by translational symmetry. The discussion has been extended to gauge symmetries, which led to the formulation of neural networks that preserve such symmetries on the lattice. These networks have then been shown to be applicable to the generation of field configurations. In the following, we will recapitulate each of these points in more detail.

First, we have compared the performance of three different CNN architectures on a 1+1D translationally symmetric complex scalar field theory with quartic interaction and a non-zero chemical potential. One architecture type was translationally equivariant (EQ), meaning that it respected the symmetry by construction, using convolutions and spatial pooling layer with a stride of one and a GAP after the convolutional part. Another type (ST) was characterized by symmetry-breaking layers, specifically spatial pooling layers with a stride larger than one. The last type (FL), in addition to the same non-equivariant layers of the ST type, featured a flattening step instead of a GAP, which completely destroys the symmetry. The first two architecture types could be applied to different input sizes without transfer learning, thanks to the use of GAP, while the flattening step confined the FL models to a fixed input size. FL has been a popular choice in computer vision and has been employed in physics too, despite its shortcomings.

The complex scalar field was mapped into two two-dimensional integer fields by means of a duality transformation, and the datasets were generated with the worm algorithm. In the first task, the networks were required to regress the values of two observables, the particle density n and the field average $|\phi|^2$, given the configurations of the two integer fields. In particular, for the training phase networks had access to data associated with a specific value of the chemical potential and only one lattice size, while testing was

conducted on a wide range of the chemical potential and on various lattice sizes to investigate the generalization capabilities of the architectures. The most promising architectures for each type have been selected with *optuna* to ensure a fair comparison. The EQ architecture outperforms the architecture of the other two types. Increasing the number of training samples steadily improved the results of the EQ type, whereas ST and FL remained almost unaffected. Remarkably, the results of the non-equivariant architectures were not ameliorated even by augmenting the data, in an attempt to prompt the networks to learn the symmetry during training. Equivariance proved to be relevant also when testing on configurations generated at chemical potentials higher than the one used for training, effectively exploring values of n and $|\phi|^2$ absent from the training phase. This suggested that EQ architectures are more resistant to generalization issues than the non-equivariant counterparts. The extension of the test to different lattice sizes between EQ and ST resulted again in favor of the former, additionally revealing that, for lattice sizes that are not a multiple of the stride used in the spatial pooling layers of ST, part of the input information is lost, further worsening the score of the ST architectures.

In the other two tasks, flux violations, which are associated with the presence of open worms, were artificially introduced in the configurations with a modification of the worm algorithm. The physical appeal of this study lies in its relationship with the calculation of n-point functions, and the interest from the machine learning point of view comes from the fact that networks containing only 1×1 convolutions cannot approximate the true function, since information from next neighbors is necessary to compute the flux. The generalization abilities included different values of the coupling constant and the mass, in addition to the chemical potential and the lattice size. The EQ and the ST architectures scored very similar results when asked to distinguish between configurations with and without open worms, while FL performed much worse. In the third task, the networks were required to count the number of open worms from 0 to 10 having been trained on configurations containing either 0 or 5 open worms. Also in this case, the equivariant architectures proved to be highly reliable, whereas the non-equivariant ones struggle to make good predictions, especially for values of open worms between 1 and 3. In all tasks, we found that the networks suggested by *optuna* featured a number of weights much smaller than in networks typically used in comparable studies in the literature.

As a result of these experiments, it is strongly advisable to employ translationally equivariant CNNs in problems symmetric under translations. A considerable performance boost is found when using GAP instead of a flattening step. An ideal scenario that is opened up by such a layer is to cheaply train a network on a small lattice size and apply it on larger ones without losing generalization abilities. The subsampling given by operations with a stride larger than one breaks translational symmetry and can be detrimental

to the performance, although it seems to be problem-dependent.

The observation that equivariant networks achieve higher accuracy compared to non-equivariant ones in the context of translational symmetry served as a motivation for embarking on the construction of equivariant networks in the context of lattice gauge theories. Conventional CNNs break gauge symmetry because no parallel transporters are used when adding up contributions from different lattice sites in convolutional layers. Hence, novel layers had to be designed. The input of a gauge-equivariant network is the link configuration and a set of conveniently precomputed locally transforming objects, such as the Wilson loops evaluated at each lattice site. A convolutional layer with appropriate parallel transport was constructed, and its output could be combined in a bilinear layer forming larger Wilson loops without breaking gauge symmetry. A gauge-equivariant activation function was introduced to provide non-linearity. An exponential layer represented a gauge-equivariant operation for modifying the links. After a trace operation, the resulting objects can be further processed by a CNN without spoiling gauge symmetry. A lattice gauge equivariant convolutional neural network consists of stacks of these layers and can in principle learn an arbitrarily sized Wilson loop. More generally, it can be viewed as a universal approximator of gauge equivariant or invariant functions on the lattice.

We experimented the potential of this novel type of network testing it on regressions of Wilson loops of various size, from the trivial 1×1 to the challenging 4×4, in 1+1D comparing the performance with that of traditional CNNs. As the loop size grew, not even the best CNN was able to make acceptable predictions, while the L-CNNs showed a robust performance, even when tested on lattice sizes larger than the one used for training. Additional experiments in 3+1D were conducted on 2×2 and 4×4 Wilson loops and on the topological charge, where L-CNNs showed again a remarkably good behavior. They were also able to reconstruct Wilson flow of the topological charge by being trained on uncooled samples alone. These networks are very flexible, not only because of the freedom of choosing the most appropriate sequence of layers, but also for the possibility of picking any locally transforming object as input, e.g. arbitrarily sized Wilson loops or Polyakov loops. Moreover, the code has been developed for SU(2) and extended to SU(3) in [86], and can easily be further extended to an arbitrary SU(N_c).

An application of these networks that we explored in this dissertation has been the generation of link configurations. The mechanism on which this proposal was based is the one of neural ordinary differential equations [140]. An adaptation of the original method to the SU(N_c) case led us to a gradient flow type of equation (see also [94]). The procedure is such that the network is provided with a dataset consisting of some initial configurations and their corresponding output ones. The starting configurations are chosen randomly, and are then evolved according to some action. After a successful training, the network is able to reconstruct the corresponding gradient flow.

We tested the neural gradient flow approach on an SU(2) toy model with a single link. In place of an L-CNN, we employed a dense network whose output was projected onto SU(2), in order to constrain the evolution on the group. Encouraging results were observed, with networks showing a good generalization to gradient flow times much larger with respect to the interval used for training. The memory saturation issues that can originate from backpropagating through the whole evolution of the system can be overcome by the adjoint sensitivity method. We thoroughly reported our derivation of such a method using Lagrange multipliers for real variables, then in U(1) and $SU(N_c)$. This method will enable the use of L-CNNs without stumbling into memory saturation problems even for small architectures, and will allow us to extend the toy model onto the lattice.

A long-term goal and the most natural continuation of the research conducted so far is the generation of configurations for relevant problems in lattice gauge theories. This may include the addition of fermions in the construction of L-CNNs, which still needs to be fully addressed. As discussed, L-CNNs have also been applied to other physical problems such as the fixed-point action [86], they have been used as preconditioners for the Dirac operator [84, 85], and have been extended to incorporate global symmetries [83]. Another exciting application is represented by the construction of a gauge-equivariant version of diffusion models [40] using the layers that have been designed for L-CNNs, again with the aim of generating gauge field configurations. The advantage of this approach is given by the inherent stochasticity of the diffusion models, which in principle allows the networks to explore topological sectors more efficiently than standard Monte Carlo algorithms. A very interesting possibility enabled by the gauge equivariance property of L-CNNs is to train these networks to identify an observable in a specific gauge and investigate the properties of such a quantity in other gauges. This can be very useful, for example, in the case of center vortices, which are usually found in the maximal center gauge. Analyzing their characteristics in other gauges can lead to a deeper understanding of the problem of quark confinement. In general, the possibility of customizing novel layers can inspire future works and the versatility of L-CNNs makes them a valuable tool that can find an application in several other lattice gauge theory problems.

Appendix A

Monte Carlo simulation of pure lattice gauge theory

This is a short review of how to use MCMC for the generation of gauge link configurations in a pure SU(2) gauge theory on the lattice, largely based on [110]).

The goal is the generation of a sequence of random configurations distributed according to the Wilson action $S_W[U]$ in Eq. (3.1.15), whose corresponding probability functional is

$$\rho[U] \propto e^{-S_W[U]}. \tag{A.1}$$

We propose the following random link updates:

$$U'_{x,\mu} = V U_{x,\mu}, \tag{A.2}$$

where V is a random SU(N_c) matrix that lies close to the identity. The Metropolis acceptance probability is

$$p[U, U'] = \min\left(1, e^{-S_W[U']+S_W[U]}\right). \tag{A.3}$$

In order to generate the matrices V, we first generate random color vectors $X^a = A\,\eta^a$ with amplitude $A > 0$ and $a \in \{1, 2, \ldots, N_c^2 - 1\}$, where η^a are sampled from a standard normal distribution. The color vectors are contracted with the generators of the group generators, which for SU(2) are the Pauli matrices (4.2.5), and the matrix exponential projects the result onto the gauge group, yielding

$$V = e^{i\sum_a T^a X^a} \in \mathrm{SU}(N_c). \tag{A.4}$$

In a single sweep, each link is updated according to (A.2) multiple times. In our implementation, we choose $A = 0.5$ and perform ten consecutive updates.

Appendix B

L-CNN architectures

Details of the composition of the L-CNN architectures employed Section 3.5 are summarized in Table B.1 and B.2 respectively for the 1+1D and 3+1D case.

The L-CB operation emerges from the combination of L-Conv and L-Bilin into a single layer, as explained in subsection 3.2.4. The notation L-CB(K, N_{in}, N_{out}) indicates that such a layer is characterized by a kernel size K in every direction, N_{in} input and N_{out} output channels. Applying the Trace to a configuration $(\mathcal{U}, \mathcal{W})$ with $N_{\mathcal{W}}$ channels for the $\mathcal{W}$ objects, we obtain $2\,N_{\mathcal{W}}$ real numbers per site. The doubled number of channels is a consequence of treating real and imaginary parts separately in the linear layers, indicated by Linear(N_{in}, N_{out}), where N_{in} is the number of input nodes and N_{out} the number of output nodes. Such linear layers act individually at every lattice site, leading to an output that is made up of observables defined over the whole lattice. Applying the same operation at each lattice site does not spoil the overall translational invariance of the networks. The output of the linear layers is not processed by an activation function. N_{param} indicates the number of trainable parameters.

Table B.1: L-CNN architectures for the prediction of $W^{(1\times1)}$, $W^{(1\times2)}$, $W^{(2\times2)}$ and $W^{(4\times4)}$ in 1+1D. Table from [2].

$W^{(1\times1)}$			
	Small		
	L-CB(1, 1, 1)		
	Trace		
	Linear(2, 1)		
N_{param}	12		
$W^{(1\times2)}$			
	Small	Medium	Large
	L-CB(2, 1, 2)	L-CB(3, 1, 4)	L-CB(4, 1, 8)
	Trace	Trace	Trace
	Linear(4, 1)	Linear(8, 1)	Linear(16, 1)
N_{param}	35	117	329
$W^{(2\times2)}$			
	Small	Medium	Large
	L-CB(2, 1, 2)	L-CB(3, 1, 4)	L-CB(4, 1, 8)
	L-CB(2, 2, 2)	L-CB(3, 4, 4)	L-CB(4, 8, 8)
	Trace	Trace	Trace
	Linear(4, 1)	Linear(8, 1)	Linear(16, 1)
N_{param}	125	1,305	13,521
$W^{(4\times4)}$			
	Small	Medium	Large
	L-CB(2, 1, 2)	L-CB(3, 1, 4)	L-CB(4, 1, 8)
	L-CB(2, 2, 2)	L-CB(3, 4, 4)	L-CB(4, 8, 8)
	L-CB(3, 2, 2)	L-CB(4, 4, 4)	L-CB(4, 8, 8)
	L-CB(3, 2, 2)	L-CB(4, 4, 4)	L-CB(4, 8, 8)
	Trace	Trace	Trace
	Linear(4, 1)	Linear(8, 1)	Linear(16, 1)
N_{param}	465	4,833	39,905

Table B.2: L-CNN architectures for the prediction of $W^{(2\times2)}$, $W^{(4\times4)}$ and q^{plaq} in 3+1D. Table from [2].

$W^{(2\times2)}$		
	Small	Medium
	L-CB(2, 6, 2)	L-CB(3, 6, 4)
	L-CB(2, 2, 2)	L-CB(3, 4, 4)
	Trace	Trace
	Linear(4, 1)	Linear(8, 1)
N_{param}	1,801	8,305
$W^{(4\times4)}$		
	Small	Medium
	L-CB(2, 6, 2)	L-CB(3, 6, 4)
	L-CB(2, 2, 2)	L-CB(3, 4, 4)
	L-CB(3, 2, 2)	L-CB(4, 4, 4)
	L-CB(3, 2, 2)	L-CB(4, 4, 4)
	Trace	Trace
	Linear(4, 1)	Linear(8, 1)
N_{param}	2,109	14,377
q^{plaq}		
	Small	
	L-CB(2, 6, 4)	
	Trace	
	Linear(8, 1)	
N_{param}	3,181	

Appendix C

Baseline networks

Here are reported the tables containing the information about the traditional CNN architectures used in Section 3.5 for the regressions of Wilson loops in 1+1D.

In order to find traditional architectures able to compete with L-CNNs, we tried to cover different ranges of some hyperparameters, specifically the network depth and width. We categorize the architectures into small, medium, large and wide, and manually select three representative architectures of comparable size for each category. These architectures consist of two-dimensional convolutional layers, which in the tables are denoted by $C2D(K, N_{\text{in}}, N_{\text{out}})$, with K indicating the size of the kernel $(K \times K)$, and N_{in} and N_{out} representing respectively the number of input and output channels. As previously introduced, the acronym GAP stands for global average pooling, which is used for all CNNs, in contrast to L-CNNs where predictions are made at each lattice site. In the dense part, linear layers use the same notation as for L-CNNs. We apply non-linear activation functions after every convolution and linear layer, except after the final output layer. Each architecture has been tried with four different activation functions, i.e. *LeakyReLU*, *ReLU*, *sigmoid* and *tanh*. We also list the number of learnable weights N_{param}.

Table C.1 lists the architectures employed for the prediction of $W^{(1\times1)}$ and $W^{(1\times2)}$, while Tables C.2 and C.3 report the architectures used for $W^{(2\times2)}$ and $W^{(4\times4)}$ respectively.

Table C.1: Baseline CNN architectures for the prediction of $W^{(1\times1)}$ and $W^{(1\times2)}$. Table adapted from [2].

$W^{(1\times1)}$, $W^{(1\times2)}$

Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 4)	C2D(2, N_{in}, 4)	C2D(1, N_{in}, 8)
	C2D(1, 4, 8)	C2D(2, 4, 4)	C2D(2, 8, 4)
	GAP	GAP	GAP
	Linear(8, 4)	Linear(4, 4)	Linear(4, 1)
	Linear(4, 1)	Linear(4, 1)	-
N_{param}	597	609	401
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 8)	C2D(2, N_{in}, 8)	C2D(3, N_{in}, 4)
	C2D(2, 8, 8)	C2D(2, 8, 8)	C2D(2, 4, 8)
	C2D(2, 8, 8)	-	-
	GAP	GAP	GAP
	Linear(8, 4)	Linear(8, 4)	Linear(8, 4)
	Linear(4, 1)	Linear(4, 1)	Linear(4, 1)
N_{param}	1,601	1,337	1,333
Large	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 16)	C2D(3, N_{in}, 16)	C2D(3, N_{in}, 16)
	C2D(2, 16, 16)	C2D(3, 16, 8)	C2D(1, 16, 8)
	C2D(2, 16, 16)	-	C2D(3, 8, 16)
	GAP	GAP	GAP
	Linear(16, 8)	Linear(8, 8)	Linear(16, 8)
	Linear(8, 1)	Linear(8, 1)	Linear(8, 1)
N_{param}	4,289	5,865	6,073
Wide	Architecture 1	Architecture 2	Architecture 3
	C2D(2, N_{in}, 128)	C2D(2, N_{in}, 256)	C2D(2, N_{in}, 512)
	-	C2D(3, 256, 32)	-
	GAP	GAP	GAP
	Linear(128, 1)	Linear(32, 1)	Linear(512, 64)
	-	-	Linear(64, 1)
N_{param}	16,641	106,817	98,945

Table C.2: Baseline CNN architectures for the prediction of $W^{(2\times2)}$. Table from [2].

$W^{(2\times2)}$			
Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 2)	C2D(2, 32, 4)
	C2D(2, 4, 4)	C2D(1, 2, 4)	C2D(2, 4, 2)
	GAP	GAP	GAP
	Linear(4, 4)	Linear(4, 1)	Linear(2, 1)
	Linear(4, 1)	-	
N_{param}	609	275	553
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 8)	C2D(3, 32, 4)
	C2D(2, 4, 8)	C2D(2, 8, 8)	C2D(2, 4, 8)
	C2D(2, 8, 8)	C2D(2, 8, 8)	C2D(3, 8, 8)
	C2D(2, 8, 8)	C2D(2, 8, 8)	C2D(2, 8, 8)
	GAP	GAP	GAP
	Linear(8, 16)	Linear(8, 8)	Linear(8, 4)
	Linear(16, 1)	Linear(8, 1)	Linear(4, 1)
N_{param}	1,341	1,905	2,181
Large	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 8)	C2D(2, 32, 8)	C2D(3, 32, 8)
	C2D(2, 8, 16)	C2D(2, 8, 16)	C2D(3, 8, 16)
	C2D(2, 16, 32)	C2D(2, 16, 32)	C2D(3, 16, 32)
	C2D(2, 32, 64)	C2D(2, 32, 64)	C2D(3, 32, 16)
	-	C2D(2, 64, 32)	-
	GAP	GAP	GAP
	Linear(64, 16)	Linear(32, 8)	Linear(16, 8)
	Linear(16, 1)	Linear(8, 1)	Linear(8, 1)
N_{param}	12,953	20,393	12,889

Table C.3: Baseline CNN architectures for the prediction of $W^{(4\times4)}$. Table from [2].

$W^{(4\times4)}$			
Small	Architecture 1	Architecture 2	Architecture 3
	C2D(2, 32, 4)	C2D(2, 32, 4)	C2D(2, 32, 4)
	C2D(2, 4, 4)	C2D(1, 4, 8)	C2D(2, 4, 2)
	GAP	GAP	GAP
	Linear(4, 4)	Linear(8, 4)	Linear(2, 1)
	Linear(4, 1)	Linear(4, 1)	-
N_{param}	609	597	553
Medium	Architecture 1	Architecture 2	Architecture 3
	C2D(3, 32, 16)	C2D(2, 32, 16)	C2D(3, 32, 8)
	C2D(1, 16, 8)	C2D(2, 16, 24)	C2D(2, 8, 16)
	C2D(3, 8, 16)	C2D(2, 24, 16)	C2D(1, 16, 32)
	-	-	C2D(2, 32, 16)
	-	-	C2D(2, 16, 8)
	GAP	GAP	GAP
	Linear(16, 8)	Linear(16, 8)	Linear(8, 8)
	Linear(8, 1)	Linear(8, 1)	Linear(8, 1)
N_{param}	6,073	5,321	6,049
Large	Architecture 1	Architecture 2	Architecture 3
	C2D(3, 32, 16)	C2D(2, 32, 16)	C2D(4, 32, 16)
	C2D(3, 16, 32)	C2D(2, 16, 32)	C2D(4, 16, 32)
	C2D(3, 32, 64)	C2D(2, 32, 64)	C2D(4, 32, 32)
	C2D(3, 64, 32)	C2D(2, 64, 64)	C2D(4, 32, 16)
	-	C2D(2, 64, 32)	-
	-	C2D(2, 32, 16)	-
	GAP	GAP	GAP
	Linear(32, 16)	Linear(16, 16)	Linear(16, 8)
	Linear(16, 1)	Linear(16, 8)	Linear(8, 8)
	-	Linear(8, 1)	Linear(8, 1)
N_{param}	46,769	39,553	41,273

Appendix D

Test results

The detailed results of the experiments in Section 3.5 are given in this Appendix. Tables D.1, D.2, D.3 and D.4 show the test results of both CNNs and L-CNNs in 1+1D for $W^{(1\times1)}$, $W^{(1\times2)}$, $W^{(2\times2)}$ and $W^{(4\times4)}$ respectively. Table D.5 displays the results of L-CNN architectures for $W^{(2\times2)}$, $W^{(4\times4)}$ and the topological charge.

In each table, we report the median MSE of the model ensemble for each architecture type and each lattice size. The architectures employed for these experiments can be found in Appendix B and C. The abbreviations are as follows: S stands for small, M for medium, L for large and W for wide. For CNN architectures, the number that follows such letters refers to whether architecture 1, 2 or 3 is used. For CNNs, we additionally specify the type of activation function used. In the first row, we also provide the variance of the observables in the test set. The values in boldface highlight which architecture type scored the lowest median MSE. The asterisk ($*$) signals the ensembles of CNNs and L-CNNs featuring the best individual models according to the validation loss, which are the models used to produce Figs. 3.3 and 3.4.

For $W^{(1\times1)}$ in 1+1D, the L-CNN architecture scores the best median MSE on 8×8 lattices, but interestingly the S1 type beats it when testing on larger lattices. For larger Wilson loops, the L-CNNs consistently outperform the traditional architectures. For $W^{(1\times2)}$, the best L-CNN architecture beats every traditional architecture by at least four orders of magnitude. We notice that baseline models considerably improve their median MSE as the lattice size grows, which can be ascribed to the use of a GAP in the final layer of the baseline architectures, effectively reducing fluctuations in the predictions. The improvement for L-CNNs is much milder. For $W^{(2\times2)}$ and $W^{(4\times4)}$, L-CNNs maintain a low median MSE, with the medium and large-sized architectures being favored over the small one, which is instead the better solution for smaller loops. In 3+1D, we did not train traditional CNNs for a direct comparison, nevertheless we observe that the behavior of

121

L-CNNs is similar to the 1+1D case, with comparable values of MSE.

Table D.1: Test results of L-CNN and CNN architectures (denoted as "Base") for the regression of $W^{(1\times1)}$ in 1+1D. The architectures used are displayed in Tables B.1 and C.1. Table from [2].

	8×8	16×16	32×32	64×64
Variance	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.61×10^{-2}
L-CNN*	$\mathbf{2.19 \times 10^{-8}}$	2.19×10^{-8}	2.19×10^{-8}	2.19×10^{-8}
Base S1 (tanh)	8.82×10^{-6}	3.43×10^{-6}	1.98×10^{-6}	1.53×10^{-6}
Base S1 (sigm)	5.07×10^{-6}	1.60×10^{-6}	7.89×10^{-7}	5.93×10^{-7}
Base S1 (leaky)	2.26×10^{-8}	8.23×10^{-9}	4.55×10^{-9}	3.42×10^{-9}
Base S1 (relu)	3.14×10^{-8}	$\mathbf{6.99 \times 10^{-9}}$	$\mathbf{2.55 \times 10^{-9}}$	$\mathbf{1.20 \times 10^{-9}}$
Base S2 (tanh)	1.57×10^{-5}	4.87×10^{-6}	2.19×10^{-6}	1.37×10^{-6}
Base S2 (sigm)	4.74×10^{-6}	1.84×10^{-6}	9.49×10^{-7}	7.83×10^{-7}
Base S2 (leaky)*	2.67×10^{-7}	3.40×10^{-8}	1.47×10^{-8}	9.15×10^{-9}
Base S2 (relu)	4.82×10^{-7}	2.21×10^{-7}	6.37×10^{-8}	4.23×10^{-8}
Base S3 (tanh)	1.52×10^{-6}	5.78×10^{-7}	3.21×10^{-7}	2.48×10^{-7}
Base S3 (sigm)	1.40×10^{-6}	3.86×10^{-7}	1.27×10^{-7}	7.98×10^{-8}
Base S3 (leaky)	1.10×10^{-7}	3.78×10^{-8}	2.47×10^{-8}	1.83×10^{-8}
Base S3 (relu)	5.94×10^{-7}	2.25×10^{-7}	6.05×10^{-8}	4.95×10^{-8}
Base M1 (tanh)	1.39×10^{-5}	4.35×10^{-6}	1.62×10^{-6}	9.62×10^{-7}
Base M1 (sigm)	9.52×10^{-6}	3.76×10^{-6}	2.54×10^{-6}	2.28×10^{-6}
Base M1 (leaky)	2.41×10^{-7}	8.54×10^{-8}	4.93×10^{-8}	2.61×10^{-8}
Base M1 (relu)	5.22×10^{-6}	1.35×10^{-6}	6.63×10^{-7}	4.19×10^{-7}
Base M2 (tanh)	1.33×10^{-5}	4.27×10^{-6}	1.79×10^{-6}	1.08×10^{-6}
Base M2 (sigm)	3.86×10^{-6}	1.38×10^{-6}	7.22×10^{-7}	6.32×10^{-7}
Base M2 (leaky)	7.21×10^{-8}	1.83×10^{-8}	8.10×10^{-9}	4.81×10^{-9}
Base M2 (relu)	2.24×10^{-6}	7.41×10^{-7}	2.03×10^{-7}	1.25×10^{-7}
Base M3 (tanh)	7.21×10^{-6}	3.31×10^{-6}	2.46×10^{-6}	1.69×10^{-6}
Base M3 (sigm)	2.51×10^{-6}	9.12×10^{-7}	5.62×10^{-7}	5.02×10^{-7}
Base M3 (leaky)	1.17×10^{-7}	3.39×10^{-8}	1.23×10^{-8}	7.25×10^{-9}
Base M3 (relu)	6.51×10^{-7}	2.47×10^{-7}	4.64×10^{-8}	1.57×10^{-8}
Base L1 (tanh)	1.12×10^{-5}	3.71×10^{-6}	1.73×10^{-6}	1.04×10^{-6}
Base L1 (sigm)	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.61×10^{-2}
Base L1 (leaky)	7.83×10^{-7}	1.89×10^{-7}	7.43×10^{-8}	5.22×10^{-8}
Base L1 (relu)	4.12×10^{-6}	9.68×10^{-7}	2.97×10^{-7}	1.37×10^{-7}
Base L2 (tanh)	1.91×10^{-5}	6.74×10^{-6}	2.88×10^{-6}	2.11×10^{-6}
Base L2 (sigm)	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.61×10^{-2}
Base L2 (leaky)	3.00×10^{-6}	1.31×10^{-6}	9.96×10^{-7}	9.52×10^{-7}
Base L2 (relu)	3.21×10^{-6}	8.96×10^{-7}	2.86×10^{-7}	1.83×10^{-7}
Base L3 (tanh)	1.07×10^{-5}	3.38×10^{-6}	1.69×10^{-6}	1.35×10^{-6}
Base L3 (sigm)	2.34×10^{-5}	7.51×10^{-6}	2.90×10^{-6}	2.37×10^{-6}
Base L3 (leaky)	2.90×10^{-6}	7.75×10^{-7}	3.34×10^{-7}	1.73×10^{-7}
Base L3 (relu)	2.66×10^{-6}	8.01×10^{-7}	2.96×10^{-7}	1.41×10^{-7}
Base W1 (tanh)	7.26×10^{-7}	2.31×10^{-7}	7.88×10^{-8}	4.09×10^{-8}
Base W1 (sigm)	2.23×10^{-6}	6.94×10^{-7}	2.54×10^{-7}	1.58×10^{-7}
Base W1 (leaky)	8.99×10^{-7}	2.52×10^{-7}	8.42×10^{-8}	4.33×10^{-8}
Base W1 (relu)	1.68×10^{-6}	4.39×10^{-7}	1.50×10^{-7}	5.44×10^{-8}
Base W2 (tanh)	2.50×10^{-4}	6.50×10^{-5}	2.05×10^{-5}	1.05×10^{-5}
Base W2 (sigm)	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.61×10^{-2}
Base W2 (leaky)	6.97×10^{-5}	1.77×10^{-5}	4.83×10^{-6}	1.59×10^{-6}
Base W2 (relu)	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.61×10^{-2}
Base W3 (tanh)	2.70×10^{-5}	7.14×10^{-6}	3.41×10^{-6}	2.10×10^{-6}
Base W3 (sigm)	5.97×10^{-2}	5.69×10^{-2}	5.64×10^{-2}	5.62×10^{-2}
Base W3 (leaky)	5.03×10^{-5}	1.50×10^{-5}	7.16×10^{-6}	4.95×10^{-6}
Base W3 (relu)	3.25×10^{-6}	9.34×10^{-7}	3.44×10^{-7}	1.95×10^{-7}

Table D.2: Test results of L-CNN and CNN architectures (denoted as "Base") for the regression of $W^{(1\times2)}$ in 1+1D. The architectures used are reported in Tables B.1 and C.1. Table from [2].

	8×8	16×16	32×32	64×64
Variance	4.50×10^{-2}	4.16×10^{-2}	4.08×10^{-2}	4.07×10^{-2}
L-CNN S	$\mathbf{7.58 \times 10^{-9}}$	$\mathbf{7.15 \times 10^{-9}}$	$\mathbf{6.99 \times 10^{-9}}$	$\mathbf{6.97 \times 10^{-9}}$
L-CNN M	1.15×10^{-8}	1.10×10^{-8}	1.08×10^{-8}	1.08×10^{-8}
L-CNN L*	1.66×10^{-8}	1.60×10^{-8}	1.57×10^{-8}	1.57×10^{-8}
Base S1 (tanh)	2.34×10^{-3}	6.24×10^{-4}	1.63×10^{-4}	6.52×10^{-5}
Base S1 (sigm)	2.25×10^{-3}	5.96×10^{-4}	1.62×10^{-4}	6.29×10^{-5}
Base S1 (leaky)	2.20×10^{-3}	5.59×10^{-4}	1.45×10^{-4}	4.59×10^{-5}
Base S1 (relu)	2.17×10^{-3}	5.59×10^{-4}	1.50×10^{-4}	5.32×10^{-5}
Base S2 (tanh)	2.32×10^{-3}	6.05×10^{-4}	1.65×10^{-4}	7.00×10^{-5}
Base S2 (sigm)	2.23×10^{-3}	5.85×10^{-4}	1.52×10^{-4}	5.22×10^{-5}
Base S2 (leaky)	2.14×10^{-3}	5.55×10^{-4}	1.44×10^{-4}	5.33×10^{-5}
Base S2 (relu)	2.09×10^{-3}	5.31×10^{-4}	1.48×10^{-4}	5.38×10^{-5}
Base S3 (tanh)	2.19×10^{-3}	5.57×10^{-4}	1.54×10^{-4}	5.80×10^{-5}
Base S3 (sigm)*	2.07×10^{-3}	5.26×10^{-4}	1.32×10^{-4}	4.29×10^{-5}
Base S3 (leaky)	2.01×10^{-3}	5.08×10^{-4}	1.39×10^{-4}	4.73×10^{-5}
Base S3 (relu)	2.03×10^{-3}	5.14×10^{-4}	1.38×10^{-4}	4.81×10^{-5}
Base M1 (tanh)	2.51×10^{-3}	6.70×10^{-4}	1.78×10^{-4}	7.41×10^{-5}
Base M1 (sigm)	2.35×10^{-3}	6.21×10^{-4}	1.61×10^{-4}	6.22×10^{-5}
Base M1 (leaky)	2.18×10^{-3}	5.63×10^{-4}	1.46×10^{-4}	5.39×10^{-5}
Base M1 (relu)	2.29×10^{-3}	6.04×10^{-4}	1.77×10^{-4}	7.70×10^{-5}
Base M2 (tanh)	2.51×10^{-3}	6.84×10^{-4}	1.83×10^{-4}	7.66×10^{-5}
Base M2 (sigm)	2.31×10^{-3}	6.04×10^{-4}	1.53×10^{-4}	5.46×10^{-5}
Base M2 (leaky)	2.11×10^{-3}	5.33×10^{-4}	1.38×10^{-4}	4.87×10^{-5}
Base M2 (relu)	2.16×10^{-3}	5.52×10^{-4}	1.42×10^{-4}	5.36×10^{-5}
Base M3 (tanh)	2.89×10^{-3}	7.80×10^{-4}	2.24×10^{-4}	1.10×10^{-4}
Base M3 (sigm)	2.43×10^{-3}	6.39×10^{-4}	1.76×10^{-4}	6.92×10^{-5}
Base M3 (leaky)	2.31×10^{-3}	6.02×10^{-4}	1.53×10^{-4}	5.35×10^{-5}
Base M3 (relu)	2.43×10^{-3}	6.39×10^{-4}	1.77×10^{-4}	6.40×10^{-5}
Base L1 (tanh)	2.63×10^{-3}	7.16×10^{-4}	2.10×10^{-4}	9.13×10^{-5}
Base L1 (sigm)	2.38×10^{-3}	6.33×10^{-4}	1.76×10^{-4}	7.41×10^{-5}
Base L1 (leaky)	2.30×10^{-3}	5.93×10^{-4}	1.55×10^{-4}	5.49×10^{-5}
Base L1 (relu)	2.33×10^{-3}	6.23×10^{-4}	2.10×10^{-4}	1.14×10^{-4}
Base L2 (tanh)	2.83×10^{-3}	8.03×10^{-4}	2.60×10^{-4}	1.42×10^{-4}
Base L2 (sigm)	2.87×10^{-3}	7.49×10^{-4}	2.25×10^{-4}	1.12×10^{-4}
Base L2 (leaky)	2.41×10^{-3}	6.24×10^{-4}	1.72×10^{-4}	7.11×10^{-5}
Base L2 (relu)	2.51×10^{-3}	6.89×10^{-4}	2.32×10^{-4}	1.29×10^{-4}
Base L3 (tanh)	2.70×10^{-3}	7.38×10^{-4}	2.08×10^{-4}	9.14×10^{-5}
Base L3 (sigm)	2.53×10^{-3}	6.70×10^{-4}	1.87×10^{-4}	8.39×10^{-5}
Base L3 (leaky)	2.38×10^{-3}	6.55×10^{-4}	1.91×10^{-4}	8.43×10^{-5}
Base L3 (relu)	2.36×10^{-3}	6.38×10^{-4}	1.81×10^{-4}	7.93×10^{-5}
Base W1 (tanh)	3.73×10^{-3}	1.49×10^{-3}	8.38×10^{-4}	6.60×10^{-4}
Base W1 (sigm)	2.39×10^{-3}	6.35×10^{-4}	1.72×10^{-4}	6.35×10^{-5}
Base W1 (leaky)	2.14×10^{-3}	5.69×10^{-4}	1.78×10^{-4}	9.01×10^{-5}
Base W1 (relu)	2.14×10^{-3}	5.41×10^{-4}	1.55×10^{-4}	6.43×10^{-5}
Base W2 (tanh)	2.76×10^{-3}	7.59×10^{-4}	2.09×10^{-4}	9.09×10^{-5}
Base W2 (sigm)	4.50×10^{-2}	4.16×10^{-2}	4.09×10^{-2}	4.07×10^{-2}
Base W2 (leaky)	2.37×10^{-3}	6.75×10^{-4}	1.81×10^{-4}	7.37×10^{-5}
Base W2 (relu)	4.50×10^{-2}	4.16×10^{-2}	4.09×10^{-2}	4.07×10^{-2}
Base W3 (tanh)	2.81×10^{-3}	9.02×10^{-4}	4.35×10^{-4}	3.15×10^{-4}
Base W3 (sigm)	4.50×10^{-2}	4.16×10^{-2}	4.09×10^{-2}	4.07×10^{-2}
Base W3 (leaky)	2.51×10^{-3}	6.94×10^{-4}	2.41×10^{-4}	1.35×10^{-4}
Base W3 (relu)	2.12×10^{-3}	5.55×10^{-4}	1.78×10^{-4}	8.62×10^{-5}

Table D.3: Test results of L-CNN and CNN architectures (denoted as "Base") for the regression of $W^{(2\times2)}$ in 1+1D. The architectures used are reported in Tables B.1 and C.2. Table from [2].

	8×8	16×16	32×32	64×64
Variance	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}
L-CNN S	1.17×10^{-7}	6.91×10^{-8}	6.79×10^{-8}	6.77×10^{-8}
L-CNN M*	$\mathbf{3.24 \times 10^{-8}}$	$\mathbf{1.96 \times 10^{-8}}$	$\mathbf{1.68 \times 10^{-8}}$	$\mathbf{1.64 \times 10^{-8}}$
L-CNN L	6.67×10^{-8}	3.89×10^{-8}	3.18×10^{-8}	3.02×10^{-8}
Base S1 (tanh)	4.15×10^{-3}	1.10×10^{-3}	3.15×10^{-4}	1.27×10^{-4}
Base S1 (sigm)	3.88×10^{-3}	9.98×10^{-4}	2.81×10^{-4}	9.60×10^{-5}
Base S1 (leaky)	3.88×10^{-3}	1.01×10^{-3}	2.91×10^{-4}	1.11×10^{-4}
Base S1 (relu)	3.93×10^{-3}	1.01×10^{-3}	2.96×10^{-4}	1.09×10^{-4}
Base S2 (tanh)	3.80×10^{-3}	9.75×10^{-4}	2.82×10^{-4}	1.04×10^{-4}
Base S2 (sigm)	3.82×10^{-3}	1.00×10^{-3}	2.95×10^{-4}	1.12×10^{-4}
Base S2 (leaky)*	3.71×10^{-3}	9.54×10^{-4}	2.61×10^{-4}	8.63×10^{-5}
Base S2 (relu)	3.86×10^{-3}	9.89×10^{-4}	2.77×10^{-4}	1.00×10^{-4}
Base S3 (tanh)	4.15×10^{-3}	1.11×10^{-3}	3.20×10^{-4}	1.26×10^{-4}
Base S3 (sigm)	3.85×10^{-3}	9.74×10^{-4}	2.60×10^{-4}	8.31×10^{-5}
Base S3 (leaky)	3.89×10^{-3}	1.02×10^{-3}	2.93×10^{-4}	1.14×10^{-4}
Base S3 (relu)	3.86×10^{-3}	1.01×10^{-3}	2.86×10^{-4}	1.06×10^{-4}
Base M1 (tanh)	4.19×10^{-3}	1.08×10^{-3}	3.08×10^{-4}	1.21×10^{-4}
Base M1 (sigm)	3.98×10^{-3}	1.04×10^{-3}	2.93×10^{-4}	1.04×10^{-4}
Base M1 (leaky)	3.87×10^{-3}	9.96×10^{-4}	2.77×10^{-4}	1.02×10^{-4}
Base M1 (relu)	4.13×10^{-3}	1.11×10^{-3}	3.42×10^{-4}	1.66×10^{-4}
Base M2 (tanh)	4.20×10^{-3}	1.14×10^{-3}	3.61×10^{-4}	1.77×10^{-4}
Base M2 (sigm)	3.99×10^{-3}	1.06×10^{-3}	2.95×10^{-4}	1.21×10^{-4}
Base M2 (leaky)	3.94×10^{-3}	1.02×10^{-3}	2.83×10^{-4}	1.01×10^{-4}
Base M2 (relu)	4.18×10^{-3}	1.11×10^{-3}	3.51×10^{-4}	1.68×10^{-4}
Base M3 (tanh)	4.57×10^{-3}	1.19×10^{-3}	3.72×10^{-4}	1.82×10^{-4}
Base M3 (sigm)	4.26×10^{-3}	1.15×10^{-3}	3.62×10^{-4}	1.71×10^{-4}
Base M3 (leaky)	4.04×10^{-3}	1.05×10^{-3}	2.95×10^{-4}	1.12×10^{-4}
Base M3 (relu)	4.47×10^{-3}	1.21×10^{-3}	4.29×10^{-4}	2.30×10^{-4}
Base L1 (tanh)	4.26×10^{-3}	1.13×10^{-3}	3.21×10^{-4}	1.26×10^{-4}
Base L1 (sigm)	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}
Base L1 (leaky)	4.01×10^{-3}	1.07×10^{-3}	3.14×10^{-4}	1.25×10^{-4}
Base L1 (relu)	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}
Base L2 (tanh)	4.19×10^{-3}	1.12×10^{-3}	3.43×10^{-4}	1.54×10^{-4}
Base L2 (sigm)	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}
Base L2 (leaky)	4.07×10^{-3}	1.07×10^{-3}	3.32×10^{-4}	1.54×10^{-4}
Base L2 (relu)	4.26×10^{-3}	1.14×10^{-3}	3.50×10^{-4}	1.69×10^{-4}
Base L3 (tanh)	4.34×10^{-3}	1.17×10^{-3}	3.60×10^{-4}	1.70×10^{-4}
Base L3 (sigm)	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}
Base L3 (leaky)	4.07×10^{-3}	1.08×10^{-3}	3.18×10^{-4}	1.38×10^{-4}
Base L3 (relu)	1.96×10^{-2}	1.55×10^{-2}	1.47×10^{-2}	1.45×10^{-2}

Table D.4: Test results of L-CNN and CNN architectures (denoted as "Base") for the regression of $W^{(4\times4)}$ in 1+1D. The notation is the same as in Table D.1. The architectures used are reported in Tables B.1 and C.3. Table from [2].

	8×8	16×16	32×32	64×64
Variance	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.53×10^{-5}
L-CNN S*	3.34×10^{-7}	1.51×10^{-7}	1.17×10^{-7}	1.06×10^{-7}
L-CNN M	$\mathbf{2.06 \times 10^{-7}}$	$\mathbf{7.15 \times 10^{-8}}$	$\mathbf{4.00 \times 10^{-8}}$	$\mathbf{3.10 \times 10^{-8}}$
L-CNN L	2.82×10^{-7}	1.09×10^{-7}	6.11×10^{-8}	5.26×10^{-8}
Base S1 (tanh)	4.80×10^{-3}	1.15×10^{-3}	2.95×10^{-4}	8.52×10^{-5}
Base S1 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.88×10^{-4}	7.88×10^{-5}
Base S1 (leaky)	4.79×10^{-3}	1.13×10^{-3}	2.89×10^{-4}	7.88×10^{-5}
Base S1 (relu)*	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.53×10^{-5}
Base S2 (tanh)	4.80×10^{-3}	1.14×10^{-3}	2.95×10^{-4}	8.35×10^{-5}
Base S2 (sigm)	4.80×10^{-3}	1.13×10^{-3}	2.89×10^{-4}	7.97×10^{-5}
Base S2 (leaky)	4.79×10^{-3}	1.14×10^{-3}	2.92×10^{-4}	8.16×10^{-5}
Base S2 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.92×10^{-4}	8.09×10^{-5}
Base S3 (tanh)	4.80×10^{-3}	1.14×10^{-3}	2.92×10^{-4}	8.13×10^{-5}
Base S3 (sigm)	4.80×10^{-3}	1.14×10^{-3}	2.91×10^{-4}	8.03×10^{-5}
Base S3 (leaky)	4.80×10^{-3}	1.14×10^{-3}	2.94×10^{-4}	8.28×10^{-5}
Base S3 (relu)	4.80×10^{-3}	1.14×10^{-3}	2.92×10^{-4}	8.20×10^{-5}
Base M1 (tanh)	4.81×10^{-3}	1.15×10^{-3}	2.93×10^{-4}	8.22×10^{-5}
Base M1 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.95×10^{-4}	8.30×10^{-5}
Base M1 (leaky)	4.80×10^{-3}	1.14×10^{-3}	2.90×10^{-4}	8.02×10^{-5}
Base M1 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.94×10^{-4}	8.40×10^{-5}
Base M2 (tanh)	4.80×10^{-3}	1.14×10^{-3}	2.93×10^{-4}	8.15×10^{-5}
Base M2 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.96×10^{-4}	8.33×10^{-5}
Base M2 (leaky)	4.80×10^{-3}	1.14×10^{-3}	2.99×10^{-4}	8.76×10^{-5}
Base M2 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.54×10^{-5}
Base M3 (tanh)	4.79×10^{-3}	1.14×10^{-3}	2.94×10^{-4}	8.45×10^{-5}
Base M3 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.55×10^{-5}
Base M3 (leaky)	4.79×10^{-3}	1.14×10^{-3}	2.99×10^{-4}	8.68×10^{-5}
Base M3 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.53×10^{-5}
Base L1 (tanh)	4.83×10^{-3}	1.14×10^{-3}	2.99×10^{-4}	8.74×10^{-5}
Base L1 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.55×10^{-5}
Base L1 (leaky)	4.79×10^{-3}	1.13×10^{-3}	2.92×10^{-4}	8.16×10^{-5}
Base L1 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.54×10^{-5}
Base L2 (tanh)	4.80×10^{-3}	1.14×10^{-3}	2.95×10^{-4}	8.41×10^{-5}
Base L2 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.54×10^{-5}
Base L2 (leaky)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.54×10^{-5}
Base L2 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.55×10^{-5}
Base L3 (tanh)	4.81×10^{-3}	1.15×10^{-3}	3.00×10^{-4}	8.86×10^{-5}
Base L3 (sigm)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.55×10^{-5}
Base L3 (leaky)	4.79×10^{-3}	1.14×10^{-3}	2.89×10^{-4}	8.20×10^{-5}
Base L3 (relu)	4.79×10^{-3}	1.14×10^{-3}	2.97×10^{-4}	8.54×10^{-5}

Table D.5: Test results of L-CNN architectures for all regression tasks in 3+1D. Architecture details are provided in table B.2. Image from [2].

	4×8^3	6×8^3	6×12^3	8×16^3
$W^{(2\times2)}$				
Variance	7.03×10^{-2}	7.08×10^{-2}	7.05×10^{-2}	7.05×10^{-2}
L-CNN S	$\mathbf{1.64 \times 10^{-7}}$	$\mathbf{1.63 \times 10^{-7}}$	$\mathbf{1.63 \times 10^{-7}}$	1.63×10^{-7}
L-CNN M	9.16×10^{-7}	6.18×10^{-7}	2.17×10^{-7}	$\mathbf{1.30 \times 10^{-7}}$
$W^{(4\times4)}$				
Variance	2.00×10^{-2}	2.08×10^{-2}	2.04×10^{-2}	2.03×10^{-2}
L-CNN S	$\mathbf{3.77 \times 10^{-7}}$	$\mathbf{3.79 \times 10^{-7}}$	$\mathbf{3.74 \times 10^{-7}}$	$\mathbf{3.74 \times 10^{-7}}$
L-CNN M	8.26×10^{-7}	8.16×10^{-7}	7.99×10^{-7}	7.99×10^{-7}
Q_P				
Variance	2.91×10^{-7}	1.91×10^{-7}	6.27×10^{-8}	1.87×10^{-8}
L-CNN S	3.18×10^{-9}	3.17×10^{-9}	3.17×10^{-9}	3.17×10^{-9}

Bibliography

[1] S. Bulusu, M. Favoni, A. Ipp, D.I. Müller and D. Schuh, *Generalization capabilities of translationally equivariant neural networks*, *Phys. Rev. D* **104** (2021) 074504 [2103.14686].

[2] M. Favoni, A. Ipp, D.I. Müller and D. Schuh, *Lattice Gauge Equivariant Convolutional Neural Networks*, *Phys. Rev. Lett.* **128** (2022) 032003 [2012.12901].

[3] M. Favoni, A. Ipp and D.I. Müller, *Applications of Lattice Gauge Equivariant Neural Networks*, *EPJ Web Conf.* **274** (2022) 09001 [2212.00832].

[4] S. Weinberg, *A Model of Leptons*, *Phys. Rev. Lett.* **19** (1967) 1264.

[5] A. Salam, *Weak and Electromagnetic Interactions*, *Conf. Proc. C* **680519** (1968) 367.

[6] THE MUON $g - 2$ COLLABORATION collaboration, *Measurement of the Positive Muon Anomalous Magnetic Moment to 0.20 ppm*, *Phys. Rev. Lett.* **131** (2023) 161802.

[7] T. Aoyama et al., *The anomalous magnetic moment of the muon in the Standard Model*, *Phys. Rept.* **887** (2020) 1 [2006.04822].

[8] N. Metropolis and S. Ulam, *The Monte Carlo Method*, *Journal of the American Statistical Association* **44** (1949) 335.

[9] A. Sokal, *Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms*, Springer US, Boston, MA (1997), 10.1007/978-1-4899-0319-8_6.

[10] G. Aarts, *Introductory lectures on lattice QCD at nonzero baryon number*, *Journal of Physics: Conference Series* **706** (2016) 022004.

[11] C. Gattringer and K. Langfeld, *Approaches to the sign problem in lattice field theory*, *Int. J. Mod. Phys. A* **31** (2016) 1643007 [1603.09517].

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma et al., *ImageNet Large Scale Visual Recognition Challenge*, *International Journal of Computer Vision (IJCV)* **115** (2015) 211 [1409.0575].

[13] OpenAI, *GPT-4 Technical Report*, 2303.08774.

[14] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez et al., *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, *Science* **362** (2018) 1140.

[15] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger et al., *Highly accurate protein structure prediction with AlphaFold*, *Nature* **596** (2021) 583.

[16] K.A. Tran, O. Kondrashova, A. Bradley, E.D. Williams, J.V. Pearson and N. Waddell, *Deep learning in cancer diagnosis, prognosis and treatment selection*, *Genome Medicine* **13** (2021) 152.

[17] S.R. Dubey, S.K. Singh and B.B. Chaudhuri, *Activation functions in deep learning: A comprehensive survey and benchmark*, *Neurocomputing* **503** (2022) 92.

[18] F. Rosenblatt, *The perceptron - A perceiving and recognizing automaton*, Tech. Rep. 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York (January, 1957).

[19] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.*, *Psychological Review* **65** (1958) 386.

[20] Y. LeCun, Y. Bengio and G. Hinton, *Deep learning*, *Nature* **521** (2015) 436.

[21] S. Ruder, *An overview of gradient descent optimization algorithms*, 1609.04747.

[22] X. Ying, *An Overview of Overfitting and its Solutions*, *Journal of Physics: Conference Series* **1168** (2019) 022022.

[23] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, *Mathematics of Control, Signals and Systems* **2** (1989) 303.

[24] Z. Lu, H. Pu, F. Wang, Z. Hu and L. Wang, *The Expressive Power of Neural Networks: A View from the Width*, 1709.02540.

[25] J. Carrasquilla and R.G. Melko, *Machine learning phases of matter*, *Nature Physics* **13** (2017) 431–434.

[26] K. Zhou, G. Endrődi, L.-G. Pang and H. Stöcker, *Regressive and generative neural networks for scalar field theory*, *Phys. Rev. D* **100** (2019) 011501(R) [`1810.12879`].

[27] L. Wang, Y. Jiang, L. He and K. Zhou, *Continuous-mixture Autoregressive Networks for efficient variational calculation of many-body systems*, `2005.04857`.

[28] D. Bachtis, G. Aarts and B. Lucini, *Mapping distinct phase transitions to a neural network*, *Phys. Rev. E* **102** (2020) 053306 [`2007.00355`].

[29] S. Blücher, L. Kades, J.M. Pawlowski, N. Strodthoff and J.M. Urban, *Towards novel insights in lattice field theory with explainable machine learning*, *Physical Review D* **101** (2020) [`2003.01504`].

[30] P.E. Shanahan, D. Trewartha and W. Detmold, *Machine learning action parameters in lattice quantum chromodynamics*, *Phys. Rev. D* **97** (2018) 094506 [`1801.05784`].

[31] K.A. Nicoli, C.J. Anders, L. Funcke, T. Hartung, K. Jansen, P. Kessel et al., *Estimation of Thermodynamic Observables in Lattice Field Theories with Deep Generative Models*, *Phys. Rev. Lett.* **126** (2021) 032001 [`2007.07115`].

[32] A. Alexandru, G. Basar, P.F. Bedaque and N.C. Warrington, *Complex paths around the sign problem*, *Rev. Mod. Phys.* **94** (2022) 015006 [`2007.05436`].

[33] D. Alvestad, R. Larsen and A. Rothkopf, *Towards learning optimized kernels for complex Langevin*, *JHEP* **04** (2023) 057 [`2211.15625`].

[34] D. Alvestad, A. Rothkopf and D. Sexty, *Lattice real-time simulations with learned optimal kernels*, *Phys. Rev. D* **109** (2024) L031502 [`2310.08053`].

[35] G. Kanwar, A. Lovato, N. Rocco and M. Wagman, *Mitigating Green's function Monte Carlo signal-to-noise problems using contour deformations*, `2304.03229`.

[36] H.-Y. Hu, S.-H. Li, L. Wang and Y.-Z. You, *Machine Learning Holographic Mapping by Neural Network Renormalization Group*, *Phys. Rev. Res.* **2** (2020) 023369 [`1903.00804`].

[37] D. Bachtis, G. Aarts, F. Di Renzo and B. Lucini, *Inverse Renormalization Group in Quantum Field Theory*, *Phys. Rev. Lett.* **128** (2022) 081603 [`2107.00466`].

[38] D. Bachtis, G. Aarts and B. Lucini, *Adding machine learning within Hamiltonians: Renormalization group transformations, symmetry breaking and restoration, Phys. Rev. Res.* **3** (2021) 013134 [2010.00054].

[39] J.M. Pawlowski and J.M. Urban, *Reducing Autocorrelation Times in Lattice Simulations with Generative Adversarial Networks, Mach. Learn. Sci. Tech.* **1** (2020) 045011 [1811.03533].

[40] L. Wang, G. Aarts and K. Zhou, *Generative Diffusion Models for Lattice Field Theory*, in *37th Conference on Neural Information Processing Systems*, 11, 2023 [2311.03578].

[41] H. Wu, J. Köhler and F. Noe, *Stochastic Normalizing Flows*, in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan and H. Lin, eds., vol. 33, pp. 5933–5944, Curran Associates, Inc., 2020 [2002.06707].

[42] M. Caselle, E. Cellini, A. Nada and M. Panero, *Stochastic normalizing flows as non-equilibrium transformations, JHEP* **2022** (2022) 15 [2201.08862].

[43] C. Bonanno, A. Nada and D. Vadacchino, *Mitigating topological freezing using out-of-equilibrium simulations*, 2402.06561.

[44] M. Caselle, E. Cellini and A. Nada, *Sampling the lattice Nambu-Goto string using Continuous Normalizing Flows, JHEP* **02** (2024) 048 [2307.01107].

[45] E. Noether, *Invariante Variationsprobleme, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* **1918** (1918) 235.

[46] B. Jähne, *Digital Image Processing, 5th revised and extended edition*, Berlin: Springer-Verlag (2002), 10.1088/0957-0233/13/9/711.

[47] D. Yarotsky, *Universal Approximations of Invariant Maps by Neural Networks, Constructive Approximation* **55** (2022) 407.

[48] D.-X. Zhou, *Universality of deep convolutional neural networks, Applied and Computational Harmonic Analysis* **48** (2020) 787.

[49] K. Fukushima, *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position, Biological Cybernetics* **36** (1980) 193.

[50] A. Krizhevsky, I. Sutskever and G.E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks, Commun. ACM* **60** (2017) 84–90.

[51] M. Lin, Q. Chen and S. Yan, *Network in network*, 1312.4400.

[52] K. He, X. Zhang, S. Ren and J. Sun, *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, in *2015 IEEE International Conference on Computer Vision (ICCV)*, (Los Alamitos, CA, USA), pp. 1026–1034, IEEE Computer Society, dec, 2015, 10.1109/ICCV.2015.123.

[53] S.J. Wetzel and M. Scherzer, *Machine learning of explicit order parameters: From the Ising model to SU(2) lattice gauge theory*, *Physical Review B* **96** (2017) [1705.05582].

[54] K. Padavala, A. Singh and J. Kundu, *Machine Learned Phase Transitions in a System of Anisotropic Particles on a Square Lattice*, 2102.03006.

[55] Y. Wang, Z. Cao and A.B. Farimani, *Deep Reinforcement Learning Optimizes Graphene Nanopores for Efficient Desalination*, 2101.07399.

[56] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang and L. Yang, *Physics-informed machine learning*, *Nature Reviews Physics* **3** (2021) 422.

[57] S. Greydanus, M. Dzamba and J. Yosinski, *Hamiltonian Neural Networks*, in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds., vol. 32, Curran Associates, Inc., 2019 [1906.01563].

[58] M.D. Cranmer, S. Greydanus, S. Hoyer, P.W. Battaglia, D.N. Spergel and S. Ho, *Lagrangian Neural Networks*, 2003.04630.

[59] E.H. Müller, *Exact conservation laws for neural network integrators of dynamical systems*, *Journal of Computational Physics* **488** (2023) 112234.

[60] T.S. Cohen and M. Welling, *Group Equivariant Convolutional Networks*, in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48, pp. 2990–2999, JMLR, Jun, 2016 [1602.07576].

[61] T.S. Cohen and M. Welling, *Steerable CNNs*, in *International Conference on Learning Representations (ICLR)*, Apr, 2017 [1612.08498].

[62] D.E. Worrall, S.J. Garbin, D. Turmukhambetov and G.J. Brostow, *Harmonic Networks: Deep Translation and Rotation Equivariance*, in

*IEEE Conference on Computer Vision and Pattern Recognition
(CVPR)*, pp. 7168–7177, Jul, 2017, 10.1109/CVPR.2017.758
[1612.04642].

[63] D. Worrall and G. Brostow, *CubeNet: Equivariance to 3D Rotation
and Translation*, in *Proceedings of the European Conference on
Computer Vision (ECCV)*, pp. 567–584, Sep, 2018,
10.1007/978-3-030-01228-1_35 [1804.04458].

[64] A.S. Ecker, F.H. Sinz, E. Froudarakis, P.G. Fahey, S.A. Cadena,
E.Y. Walker et al., *A rotation-equivariant convolutional neural
network model of primary visual cortex*, in *International Conference
on Learning Representations (ICLR)*, May, 2019 [1809.10504].

[65] B.S. Veeling, J. Linmans, J. Winkens, T. Cohen and M. Welling,
Rotation Equivariant CNNs for Digital Pathology, in *Medical Image
Computing and Computer-Assisted Intervention (MICCAI)*,
pp. 210–218, Sep, 2018, 10.1007/978-3-030-00934-2_24 [1806.03962].

[66] M.W. Lafarge, E.J. Bekkers, J.P.W. Pluim, R. Duits and M. Veta,
*Roto-Translation Equivariant Convolutional Networks: Application
to Histopathology Image Analysis*, *Medical Image Analysis* **68** (2021)
[2002.08725].

[67] S. Pang, A. Du, M.A. Orgun, Y. Wang, Q. Sheng, S. Wang et al.,
*Beyond CNNs: Exploiting Further Inherent Symmetries in Medical
Images for Segmentation*, 2005.03924.

[68] A.M.M. Scaife and F. Porter, *Fanaroff-Riley classification of radio
galaxies using group-equivariant convolutional neural networks*,
Monthly Notices of the Royal Astronomical Society **503** (2021)
2369–2379 [2102.08252].

[69] R. Kondor and S. Trivedi, *On the Generalization of Equivariance
and Convolution in Neural Networks to the Action of Compact
Groups*, 1802.03690.

[70] M.C. Cheng, V. Anagiannis, M. Weiler, P. de Haan, T.S. Cohen and
M. Welling, *Covariance in Physics and Convolutional Neural
Networks*, 1906.02481.

[71] C. Esteves, *Theoretical Aspects of Group Equivariant Neural
Networks*, 2004.05154.

[72] M. Rath and A.P. Condurache, *Boosting Deep Neural Networks with
Geometrical Prior Knowledge: A Survey*, 2006.16867.

[73] J.E. Gerken, J. Aronsson, O. Carlsson, H. Linander, F. Ohlsson, C. Petersson et al., *Geometric Deep Learning and Equivariant Neural Networks*, 2105.13926.

[74] E. Celledoni, M.J. Ehrhardt, C. Etmann, B. Owren, C.-B. Schönlieb and F. Sherry, *Equivariant neural networks for inverse problems*, *Inverse Problems* **37** (2021) 085006.

[75] J. Aronsson, *Homogeneous vector bundles and G-equivariant convolutional neural networks*, *Sampling Theory, Signal Processing, and Data Analysis* **20** (2022) 10.

[76] M. Zhdanov, N. Hoffmann and G. Cesa, *Implicit Convolutional Kernels for Steerable CNNs*, in *Advances in Neural Information Processing Systems*, A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt and S. Levine, eds., vol. 36, pp. 17395–17407, Curran Associates, Inc., 2023 [2212.06096].

[77] M.S. Hossain, S. Roy, K. Asad, A. Momen, A.A. Ali, M.A. Amin et al., *Morphological classification of Radio Galaxies using Semi-Supervised Group Equivariant CNNs*, *Procedia Computer Science* **222** (2023) 601.

[78] T. Edixhoven, A. Lengyel and J. van Gemert, *Using and Abusing Equivariance*, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 119–128, 2023, 10.1109/ICCVW60793.2023.00019.

[79] T.S. Cohen, M. Weiler, B. Kicanaoglu and M. Welling, *Gauge Equivariant Convolutional Networks and the Icosahedral CNN*, 1902.04615.

[80] D. Luo, G. Carleo, B.K. Clark and J. Stokes, *Gauge equivariant neural networks for quantum lattice gauge theories*, 2012.05232.

[81] M. Finzi, S. Stanton, P. Izmailov and A.G. Wilson, *Generalizing Convolutional Neural Networks for Equivariance to Lie Groups on Arbitrary Continuous Data*, 2002.12880.

[82] Y. Nagai and A. Tomiya, *Gauge covariant neural network for 4 dimensional non-abelian gauge theory*, 2103.11965.

[83] J. Aronsson, D.I. Müller and D. Schuh, *Geometrical aspects of lattice gauge equivariant convolutional neural networks*, 2303.11448.

[84] C. Lehner and T. Wettig, *Gauge-equivariant neural networks as preconditioners in lattice QCD*, *Phys. Rev. D* **108** (2023) 034503 [2302.05419].

[85] C. Lehner and T. Wettig, *Gauge-equivariant pooling layers for preconditioners in lattice QCD*, 2304.10438.

[86] K. Holland, A. Ipp, D.I. Müller and U. Wenger, *Machine learning a fixed point action for SU(3) gauge theory with a gauge equivariant convolutional neural network*, 2401.06481.

[87] Y. Nagai and A. Tomiya, *Self-learning Monte Carlo with equivariant Transformer*, 2306.11527.

[88] A. Tomiya and Y. Nagai, *Equivariant transformer is all you need*, *PoS* **LATTICE2023** (2024) 001 [2310.13222].

[89] G. Kanwar, M.S. Albergo, D. Boyda, K. Cranmer, D.C. Hackett, S. Racanière et al., *Equivariant Flow-Based Sampling for Lattice Gauge Theory*, *Physical Review Letters* **125** (2020) [2003.06413].

[90] D. Boyda, G. Kanwar, S. Racanière, D.J. Rezende, M.S. Albergo, K. Cranmer et al., *Sampling using $SU(N)$ gauge equivariant flows*, *Phys. Rev. D* **103** (2021) 074504 [2008.05456].

[91] M.S. Albergo, G. Kanwar, S. Racanière, D.J. Rezende, J.M. Urban, D. Boyda et al., *Flow-based sampling for fermionic lattice field theories*, *Phys. Rev. D* **104** (2021) 114507 [2106.05934].

[92] R. Abbott, M.S. Albergo, D. Boyda, K. Cranmer, D.C. Hackett, G. Kanwar et al., *Gauge-equivariant flow models for sampling in lattice field theories with pseudofermions*, *Phys. Rev. D* **106** (2022) 074506.

[93] K.A. Nicoli, C.J. Anders, T. Hartung, K. Jansen, P. Kessel and S. Nakajima, *Detecting and mitigating mode-collapse for flow-based sampling of lattice field theories*, *Phys. Rev. D* **108** (2023) 114501 [2302.14082].

[94] S. Bacchio, P. Kessel, S. Schaefer and L. Vaitl, *Learning trivializing gradient flows for lattice gauge theories*, *Phys. Rev. D* **107** (2023) L051504.

[95] T. Brauner, *Remarks on relativistic scalar models with chemical potential*, 2009.08895.

[96] C. Gattringer and T. Kloiber, *Lattice study of the Silver Blaze phenomenon for a charged scalar ϕ^4 field*, *Nuclear Physics B* **869** (2013) 56–73 [1206.2954].

[97] R. Angulu, J.R. Tapamo and A.O. Adewumi, *Age estimation via face images: a survey*, *EURASIP Journal on Image and Video Processing* **2018** (2018) 42.

[98] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 8026–8037, Dec, 2019 [1912.01703].

[99] S. Bozinovski, *Reminder of the First Paper on Transfer Learning in Neural Networks, 1976*, *Informatica (Slovenia)* **44** (2020) 291.

[100] F. Yu and V. Koltun, *Multi-Scale Context Aggregation by Dilated Convolutions*, in *ICLR*, 2016 [1511.07122].

[101] N. Prokof'ev and B. Svistunov, *Worm Algorithms for Classical Statistical Models*, *Physical Review Letters* **87** (2001) [cond-mat/0103146].

[102] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama, *Optuna: A Next-generation Hyperparameter Optimization Framework*, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 2623–2631, Association for Computing Machinery, Jul, 2019, 10.1145/3292500.3330701 [1907.10902].

[103] S.J. Reddi, S. Kale and S. Kumar, *On the Convergence of Adam and Beyond*, in *International Conference on Learning Representations (ICLR)*, May, 2018 [1904.09237].

[104] I. Loshchilov and F. Hutter, *Fixing Weight Decay Regularization in Adam*, in *International Conference on Learning Representations (ICLR)*, May, 2019 [1711.05101].

[105] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press (2016). http://www.deeplearningbook.org.

[106] T.D. Cohen, *Functional Integrals for QCD at Nonzero Chemical Potential and Zero Density*, *Phys. Rev. Lett.* **91** (2003) 222001.

[107] C. Gattringer and T. Kloiber, *Spectroscopy in finite density lattice field theory: An exploratory study in the relativistic Bose gas*, *Physics Letters B* **720** (2013) 210–214 [1212.3770].

[108] T. Rindlisbacher, O. Åkerlund and P. de Forcrand, *Sampling of general correlators in worm-algorithm based simulations*, *Nuclear Physics B* **909** (2016) 542–583 [1602.09017].

[109] G. Gao, J. Gao, Q. Liu, Q. Wang and Y. Wang, *CNN-based Density Estimation and Crowd Counting: A Survey*, 2003.12783.

[110] C. Gattringer and C.B. Lang, *Quantum Chromodynamics on the Lattice*, Springer Berlin Heidelberg (2010), 10.1007/978-3-642-01850-3.

[111] J. Smit, *Introduction to Quantum Fields on a Lattice*, Cambridge University Press (Sept., 2002), 10.1017/cbo9780511583971.

[112] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller, *Equation of State Calculations by Fast Computing Machines*, *The Journal of Chemical Physics* **21** (1953) 1087.

[113] M. Creutz, *Monte Carlo study of quantized SU(2) gauge theory*, *Phys. Rev. D* **21** (1980) 2308.

[114] F. Niedermayer, *Improved lattice actions*, *Nucl. Phys. B Proc. Suppl.* **53** (1997) 56 [hep-lat/9608097].

[115] Y. Iwasaki, *Renormalization Group Analysis of Lattice Theories and Improved Lattice Action: Two-Dimensional Nonlinear O(N) Sigma Model*, *Nucl. Phys. B* **258** (1985) 141.

[116] G.D. Moore, *Improved Hamiltonian for Minkowski Yang-Mills theory*, *Nucl. Phys. B* **480** (1996) 689 [hep-lat/9605001].

[117] J.F. Lagaë and D.K. Sinclair, *Improved staggered quark actions with reduced flavor symmetry violations for lattice QCD*, *Phys. Rev. D* **59** (1998) 014511 [hep-lat/9806014].

[118] A. Ipp and D. Müller, *Implicit schemes for real-time lattice gauge theory*, *Eur. Phys. J.* **C78** (2018) 884 [1804.01995].

[119] K.G. Wilson, *Confinement of quarks*, *Phys. Rev. D* **10** (1974) 2445.

[120] G.S. Bali, *QCD forces and heavy quark bound states*, *Phys. Rept.* **343** (2001) 1 [hep-ph/0001312].

[121] C. Alexandrou, A. Athenodorou, K. Cichy, A. Dromard, E. Garcia-Ramos, K. Jansen et al., *Comparison of topological charge definitions in Lattice QCD*, *Eur. Phys. J. C* **80** (2020) 424 [1708.00696].

[122] S.O. Bilson-Thompson, D.B. Leinweber and A.G. Williams, *Highly improved lattice field strength tensor*, *Annals Phys.* **304** (2003) 1 [hep-lat/0203008].

[123] S. Caracciolo, G. Curci, P. Menotti and A. Pelissetto, *The Energy Momentum Tensor for Lattice Gauge Theories*, *Annals Phys.* **197** (1990) 119.

[124] A.M. Polyakov, *Thermal Properties of Gauge Fields and Quark Liberation*, *Phys. Lett. B* **72** (1978) 477.

[125] J. Kuti, J. Polonyi and K. Szlachanyi, *Monte Carlo Study of SU(2) Gauge Theory at Finite Temperature*, *Phys. Lett. B* **98** (1981) 199.

[126] L.D. McLerran and B. Svetitsky, *Quark Liberation at High Temperature: A Monte Carlo Study of SU(2) Gauge Theory*, *Phys. Rev. D* **24** (1981) 450.

[127] K. He, X. Zhang, S. Ren and J. Sun, *Deep Residual Learning for Image Recognition*, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016, 10.1109/CVPR.2016.90 [1512.03385].

[128] J. Ambjørn, T. Askgaard, H. Porter and M. Shaposhnikov, *Sphaleron transitions and baryon asymmetry: A Numerical real time analysis*, *Nucl. Phys. B* **353** (1991) 346.

[129] M. Lüscher, *Properties and uses of the Wilson flow in lattice QCD*, *JHEP* **08** (2010) 071 [1006.4518].

[130] J. Brannick, R.C. Brower, M.A. Clark, J.C. Osborn and C. Rebbi, *Adaptive Multigrid Algorithm for Lattice QCD*, *Phys. Rev. Lett.* **100** (2008) 041601.

[131] P.M. Fishbane, S. Gasiorowicz and P. Kaus, *Stokes' Theorems for Nonabelian Fields*, *Phys. Rev. D* **24** (1981) 2324.

[132] R. Giles, *The Reconstruction of Gauge Potentials From Wilson Loops*, *Phys. Rev. D* **24** (1981) 2160.

[133] R. Loll, *Yang-Mills theory without Mandelstam constraints*, *Nucl. Phys. B* **400** (1993) 126.

[134] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov et al., *Going deeper with convolutions*, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015, 10.1109/CVPR.2015.7298594.

[135] K. Symanzik, *Continuum Limit and Improved Action in Lattice Theories. 1. Principles and ϕ^4 Theory*, *Nucl. Phys. B* **226** (1983) 187.

[136] K. Symanzik, *Continuum Limit and Improved Action in Lattice Theories. 2. O(N) Nonlinear Sigma Model in Perturbation Theory*, *Nucl. Phys. B* **226** (1983) 205.

[137] M. Lüscher and P. Weisz, *On-shell improved lattice gauge theories*, *Commun. Math. Phys.* **98** (1985) 433.

[138] P. Hasenfratz and F. Niedermayer, *Perfect lattice action for asymptotically free theories*, *Nucl. Phys. B* **414** (1994) 785 [`hep-lat/9308004`].

[139] M. Lüscher, *Trivializing maps, the Wilson flow and the HMC algorithm*, *Commun. Math. Phys.* **293** (2010) 899 [`0907.5491`].

[140] R.T.Q. Chen, Y. Rubanova, J. Bettencourt and D.K. Duvenaud, *Neural Ordinary Differential Equations*, in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, eds., vol. 31, Curran Associates, Inc., 2018 [`1806.07366`].

[141] J. Thijssen, *Numerical methods*, Cambridge University Press (2007), 10.1017/CBO9781139171397.

[142] D. Müller, *Simulations of the Glasma in 3+1D*, Ph.D. thesis, Vienna, Tech. U., 2019. `1904.04267`.

[143] R.M. Errico, *What Is an Adjoint Model?*, *Bulletin of the American Meteorological Society* **78** (1997) 2577 .

[144] R.-E. Plessix, *A review of the adjoint-state method for computing the gradient of a functional with geophysical applications*, *Geophysical Journal International* **167** (2006) 495.

[145] Y. Cao, S. Li and L. Petzold, *Adjoint sensitivity analysis for differential-algebraic equations: algorithms and software*, *Journal of Computational and Applied Mathematics* **149** (2002) 171.

[146] H. Munthe-Kaas, *Runge-Kutta methods on Lie groups*, *BIT Numerical Mathematics* **38** (1998) 92.