DIPLOMARBEIT

Group equivariant neural networks for a scalar field theory

ausgeführt am Institut für Theoretische Physik
der Technischen Universität Wien

unter Anleitung von
Privatdoz. Dipl.-Ing. Dr.techn. Andreas Ipp
Dipl.-Ing. Daniel Schuh, BSc

von

Lukas Einramhof
Mat.Nr.: 01525156

Abstract

Der Gebrauch von Machine Learning im Hochenergie-Physik Bereich ist in den letzten Jahren drastisch angestiegen. Eine mögliche Falle ist die Verwendung von Techniken aus dem Computer Vision Bereich, ohne die speziellen Eigenschaften physikalischer Systeme zu berücksichtigen. Convolutional Neural Networks (CNNs) können zum Beispiel verwendet werden um die Translationssymmetrien der studierten Systeme zu berücksichtigen. Wir wenden CNNs auf ein Skalarfeld an und berücksichtigen dabei die vorhandenen Symmetrien des Systems. In einer vorausgehenden Arbeit wurde bereits die Translationssymmetrie des untersuchten Systems in ein CNN eingebaut. In dieser Diplomarbeit untersuchen wir zusätzlich noch die Spiegelsymmetrie des physikalischen Systems und bauen diese in ein CNN ein. Hierzu verwenden wir die Methoden von Group equivariant CNNs (G-CNNs), allerdings in Bezug auf Spiegelungen statt Rotationen. Danach vergleichen wir dieses neue Netzwerk mit dem aus der vorgehenden Arbeit. Dabei zeigt sich, dass die Spiegelsymmetrie für die untersuchten Observablen gegenüber der Translationssymmetrie eine untergeordnete Rolle spielt.

Abstract

The use of machine learning algorithms in the high energy physics sector is ever increasing. A potential pitfall is using such methods coming from the computer vision sector without considering the special properties of physical systems. For example, Convolutional Neural Networks (CNNs) can be used to include the translational symmetry found in physical systems. In this work we study a scalar field model and use methods from Group equivariant CNNs (G-CNNs) in order to include the translation and reflection symmetry underlying the studied system in a CNN. In previous work a CNN with translational symmetry was already studied. We compare our new network with that of the previous paper. It turns out that the reflection symmetry plays a subordinate role compared to the translation symmetry for the studied observables.

Contents

Chapter 1

Introduction

Over the last years the computational cost of solving physical problems has been ever increasing. One promising avenue to tackle these problems are machine learning algorithms. As such, they have become more popular within the physics community. A special subset of these algorithms are Convolutional Neural Networks (CNNs). These are especially relevant to physical problems as they take the spatial arrangement of the problem into account. CNNs are mainly built from two ingredients: convolutional layers and pooling layers. Even though CNNs are based on the idea of translational symmetry, convolutional layers may break translational symmetry through border effects, as may pooling layers that shrink the image size. In [1] it was shown that creating a CNN that respects the translational symmetry of a physical system improves its ability to predict physical observables of the system.

In this thesis we will create a CNN that is equivariant under translations and reflections of the input and compare its ability to predict physical observables of a complex scalar field with the translationally equivariant network of [1]. For this we use the methods of Group equivariant CNNs [2] adapted to reflections instead of rotations.

To give an overview of this thesis: First we will discuss the complex scalar field, show its symmetry under translations and reflections, and how the symmetries apply to the discretized version in chapter 2. Then, we will examine CNNs more in depth and analyze how to create a network that is equivariant under the chosen symmetries in chapter 3. Next we will test the implemented network and compare it to the equivariant network of [1] in chapter 4. Lastly, in chapter 5 we draw conclusions and look at possible future avenues to explore. In the appendices we will show in detail how to discretize the continuous form of the action (Appendix A), how to convert the discretized action to the flux representation of the partition function (Appendix B), how the symmetries are translated over to the variables of the flux representation (Appendix C), and finally we demonstrate by example how the kernels of an equivariant convolutional layer need to transform (Appendix D).

Chapter 2

Symmetries of the action

In this thesis we try to predict observables of a complex scalar field ϕ. The associated action in D dimensions takes the form

$$S = S[\phi] = \int d^D x \mathcal{L} \tag{2.1}$$

with the Lagrangian

$$\mathcal{L} = \partial_\nu \phi^* \partial^\nu \phi - V(\phi^* \phi) \tag{2.2}$$

where V is a general potential function. In this paper the $(+, -, \ldots, -)$ version of the Minkowski metric is used. Applying the transformation $x_\nu \to x'_\nu = x_\nu + a_\nu, \forall \nu$ with a constant vector a_ν to the Lagrangian results in

$$\mathcal{L}' = \partial_{\nu'} \phi^*(x') \partial^{\nu'} \phi(x') - V(\phi^*(x')\phi(x')) = \partial_\nu \phi^*(x') \partial^\nu \phi(x') - V(\phi^*(x')\phi(x')) \tag{2.3}$$

as $\partial_{\nu'} = \partial_\nu$. Thus,

$$S = \int d^D x \mathcal{L}(x') = \int d^D x' \mathcal{L}(x') \tag{2.4}$$

and therefore, the action is invariant under translations.

Another important symmetry of the action in eq. (2.1) is a global $U(1)$ symmetry, i.e., transforming ϕ according to $\phi \to \phi' = e^{i\alpha}\phi$ leaves the action invariant. This then implies, by Noethers first theorem [3], the existence of a conserved four-current j^ν which allows the definition of a chemical potential μ. This chemical potential can then be included into the definition of the action via

$$S = \int dx_0 d^{D-1} x \left(|D_0 \phi|^2 - |\partial_i \phi|^2 - V(|\phi|^2) \right) \tag{2.5}$$

with $D_0 = \partial_0 - i\mu$ [4].

The last symmetry we will be discussing is the transformation $x_\nu \to x'_\nu = -x_\nu$, i.e., reflecting a subset of the basis vectors. To show that this transformation is indeed a symmetry of the action we will show it for each ν separately for $D = 2$. The original Lagrangian written out

then has the form

$$\mathcal{L} = \partial_0\phi^*\partial_0\phi - \partial_1\phi^*\partial_1\phi - V(|\phi|^2). \tag{2.6}$$

We write $\phi = \phi(x)$ and $\phi' = \phi(x')$. First we transform $x_0 \to x'_0 = -x_0$. Then, $\partial'_0 = -\partial_0$ and therefore,

$$\mathcal{L}' = (-\partial_0\phi'^*)(-\partial_0\phi') - \partial_1\phi'^*\partial_1\phi' - V(|\phi'|^2) = \partial_0\phi'^*\partial_0\phi' - \partial_1\phi'^*\partial_1\phi' - V(|\phi'|^2). \tag{2.7}$$

Inserting this into the action results in

$$\begin{aligned} S' = \int_{-\infty}^{\infty} dx_0 \int_{-\infty}^{\infty} dx_1 \big[&\partial_0\phi^*(-x_0, x_1)\partial_0\phi(-x_0, x_1) \\ &- \partial_1\phi^*(-x_0, x_1)\partial_1\phi(-x_0, x_1) - V(|\phi(-x_0, x_1)|^2)\big]. \end{aligned} \tag{2.8}$$

Now, substituting $x_0 = -x'_0$ results in $dx_0 = -dx'_0$ with integration bounds $x'^{down}_0 = +\infty, x'^{up}_0 = -\infty$. Switching the integration bounds cancels the extra minus sign from the integration measure. Thus, the action is invariant under a reflection in the x_0 direction. Doing the same for x_1 results again in invariance of the action. This can also be generalized to any number of spatial dimensions. Note that this invariance under reflections needs to hold in eq. (2.5) and therefore implies that μ has to switch signs under a reflection in x_0. To see this, we look only at the term $|D_0\phi|^2$. Applying the transformation $x_0 \to x'_0 = -x_0$ results in

$$\begin{aligned} |D'_0\phi'|^2 &= (\partial'_0 + i\mu')\phi'^*(\partial'_0 - i\mu')\phi' = (\partial_0 - i\mu')\phi'^*(\partial_0 + i\mu')\phi' \\ &= \partial_0\phi'^*\partial_0\phi' + \mu'^2|\phi'|^2 + i\mu'(\phi'^*\partial_0\phi' - \phi'\partial_0\phi'^*). \end{aligned} \tag{2.9}$$

Without the transformation this term has the form

$$\begin{aligned} |D_0\phi|^2 &= (\partial_0 + i\mu)\phi^*(\partial_0 - i\mu)\phi' = (\partial_0 + i\mu)\phi^*(\partial_0 - i\mu)\phi \\ &= \partial_0\phi^*\partial_0\phi + \mu^2|\phi|^2 - i\mu(\phi^*\partial_0\phi - \phi\partial_0\phi^*). \end{aligned} \tag{2.10}$$

Therefore, requiring $|D_0\phi'|^2 = |D'_0\phi'|^2$ leads to $\mu' = -\mu$. Under spatial reflections μ is invariant.

The above symmetries hold for any potential of the form $V(|\phi|^2)$. From now on we will be consider only $D = 2$ and $V(|\phi|^2) = m^2|\phi|^2 + \lambda|\phi|^4$ with mass m and coupling constant λ.

2.1 The discretized action

In order to apply a machine learning algorithm to this problem, the action must first be discretized. However, due to the oscillatory nature of the exponent in the partition function

$$Z = \int \mathcal{D}[\phi] e^{iS[\phi]}, \tag{2.11}$$

in practice, a euclidean action is used instead. This is achieved by applying a Wick rotation

$$x_0 \to i x_2, x_2 \in \mathbb{R}. \tag{2.12}$$

The euclidean action then takes the form

$$S_E = \int dx_1 dx_2 \big(|\partial_1 \phi|^2 + D_2' \phi^* D_2 \phi + m^2 |\phi|^2 + \lambda |\phi|^2\big) \tag{2.13}$$

with $D_2 = \partial_2 + \mu$ and $D_2' = \partial_2 - \mu$. Then, the partition function takes the form

$$Z = \int \mathcal{D}[\phi] e^{-S_E[\phi]} \tag{2.14}$$

with exponential damping instead of oscillations which makes numerical evaluations feasible.

After discretizing, the euclidean action takes the form as in eq. (A.8)

$$S_E = \sum_y \left(\eta |\phi_y|^2 + \lambda |\phi_y|^4 - \sum_{\omega \in \{x,t\}} \left(e^{\mu \delta_{\omega,t}} \phi_y^* \phi_{y+\hat{e}_\omega} + e^{-\mu \delta_{\omega,t}} \phi_y^* \phi_{y-\hat{e}_\omega} \right) \right), \tag{2.15}$$

where $\hat{e}_\omega$ is the unit vector in the $\omega \in \{x,t\}$ direction. For clarity we switched to calling points on the lattice y and use x, t as indices instead of $1, 2$ where x describes the spatial dimension and t the temporal dimension. The complex scalar field now lives on a discrete two-dimensional rectangular lattice. Periodic boundary conditions are enforced. The outer sum runs over all points on the lattice. The new parameter η has the form $\eta = 4 + m^2$ and we use $\phi_y := \phi(y)$ as a short-hand notation. For a detailed derivation see Appendix A.

From this discretized version we can then transform the partition function to a dual representation, called the flux representation. It describes the same physics as the discretized partition function from above, but is now a function of four new integer fields (flux variables) $k_{y,\omega} \in \mathbb{Z}$ and $l_{y,\omega} \in \mathbb{N}_0$. As derived in Appendix B the partition function in the flux representation has the form

$$Z = \sum_{\{k,l\}} \left[\left(\prod_{y,\omega} \frac{1}{(|k_{y,\omega}| + l_{y,\omega})! l_{y,\omega}!} \right) \left(\prod_y e^{\mu k_{y,t}} W(f_y) \right) \left(\prod_y \delta \left(\sum_\omega k_{y,\omega} - k_{y-\hat{\omega},\omega} \right) \right) \right] \tag{2.16}$$

where the outermost sum is shorthand for

$$\sum_{\{k,l\}} = \sum_{k_{y_1,x}} \sum_{k_{y_1,t}} \cdots \sum_{k_{y_N,x}} \sum_{k_{y_N,t}} \cdot \sum_{l_{y_1,x}} \sum_{l_{y_1,t}} \cdots \sum_{l_{y_N,x}} \sum_{l_{y_N,t}} \tag{2.17}$$

with $\{y_1, ..., y_N\}$ being the set of all lattice points. The function $W(f_y)$ is given by

$$W(f_y) = \int_0^\infty d\alpha \, e^{-\eta \alpha^2 - \lambda \alpha^4} \, \alpha^{f_y + 1} \tag{2.18}$$

and the field f_y is given by

$$f_y = \sum_{\omega} \left(2(l_{y,\omega} + l_{y-\hat{\omega},\omega}) + |k_{y,\omega}| + |k_{y-\hat{\omega},\omega}|\right). \tag{2.19}$$

2.2 Discretized symmetries

As was already shown in the beginning of this chapter, the partition function is invariant under translations and reflections. Now we show how these symmetries are carried over to the discrete flux representation of the partition function. Translations are now only possible in multiples of the lattice spacing, i.e.,

$$y \to y' = y + \alpha\hat{e}_x + \beta\hat{e}_t. \tag{2.20}$$

It is easy to see that this transformation will leave eq. (2.16) invariant as the products over all lattice points absorb any translation in the arguments. Thus, the transformation behavior of the flux variables reads

$$k_{y,\omega} \to k_{y+\alpha\hat{e}_x+\beta\hat{e}_t,\omega}, \tag{2.21}$$

and

$$l_{y,\omega} \to l_{y+\alpha\hat{e}_x+\beta\hat{e}_t,\omega}. \tag{2.22}$$

Similarly, a spatial reflection on the lattice will take the form

$$y = a\hat{e}_x + b\hat{e}_t \to y' = -a\hat{e}_x + b\hat{e}_t, \tag{2.23}$$

and a temporal reflection the form

$$y = a\hat{e}_x + b\hat{e}_t \to y' = a\hat{e}_x - b\hat{e}_t. \tag{2.24}$$

However, the transformation behavior of the flux variables for reflections is not as easy to see as for translations. The reason is that the flux variables, live as links between two lattice points, i.e., $k_{y,x}$ connects the lattice points y and $y + \hat{e}_x$. Then, when a reflection in the x direction is applied, k connects $y' = -a\hat{e}_x + b\hat{e}_t$ with $y' - \hat{e}_x$. Since this is now connected in the wrong direction, we have to redefine k as sitting on the site $y' - \hat{e}_x$ so it connects to y' in the correct direction. The problem is shown in Figure 2.1. This redefinition gives rise to an additional shift in some of the transformations.

In Appendix C the transformation behavior is derived in detail, here we will just state the results. For a reflection in the x direction the transformations are

$$\begin{aligned}
k_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{\hat{e}_x\to-\hat{e}_x} -k_{-(a+1)\hat{e}_x+b\hat{e}_t,x}, \\
k_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{\hat{e}_x\to-\hat{e}_x} k_{-a\hat{e}_x+b\hat{e}_t,t}, \\
l_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{\hat{e}_x\to-\hat{e}_x} l_{-(a+1)\hat{e}_x+b\hat{e}_t,x}, \\
l_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{\hat{e}_x\to-\hat{e}_x} l_{-a\hat{e}_x+b\hat{e}_t,t}
\end{aligned} \tag{2.25}$$

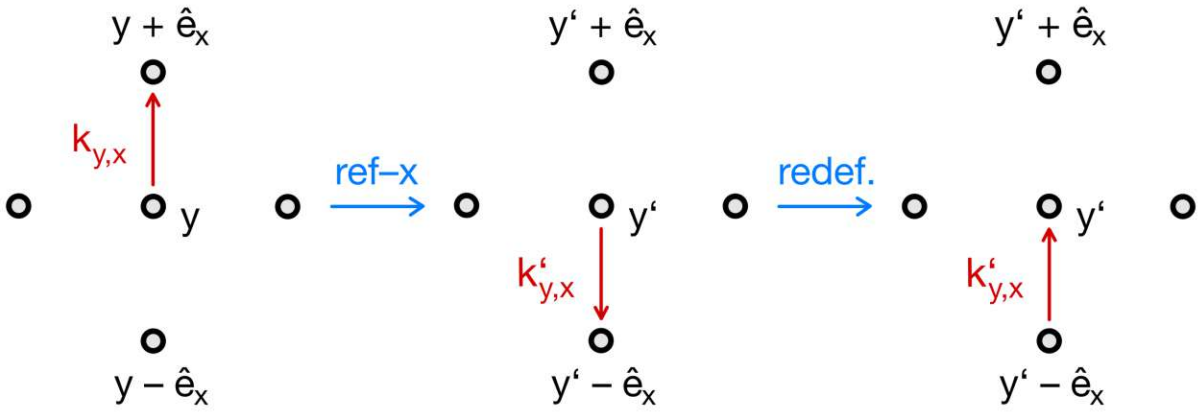

Figure 2.1: Behavior of the flux variables under reflections. $k_{y,x}$ connects the sites y and $y+\hat{e}_x$. When a reflection in the x direction is applied, $k_{y,x}$ has to also be redefined to sitting at the lower site $y' - \hat{e}_x$ in order to be defined the same way as before the reflection.

and for a reflection in the t direction the transformations are

$$\begin{aligned} k_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{\hat{e}_t\to-\hat{e}_t} k_{a\hat{e}_x-b\hat{e}_t,x}, \\ k_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{\hat{e}_t\to-\hat{e}_t} -k_{a\hat{e}_x-(b+1)\hat{e}_t,t}, \\ l_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{\hat{e}_t\to-\hat{e}_t} l_{a\hat{e}_x-b\hat{e}_t,x}, \\ l_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{\hat{e}_t\to-\hat{e}_t} l_{a\hat{e}_x-(b+1)\hat{e}_t,t}. \end{aligned} \tag{2.26}$$

Under these transformations the partition function is invariant.

Chapter 3

Equivariant structure of the neural network

The goal of this thesis is to predict different observables on the lattice with a neural network. Because of the two-dimensional nature of problem at hand, and therefore its similarity with an image, Convolutional Neural Networks (CNNs) are the natural choice to tackle this problem. The inputs will be the four integer fields $k_{y,\omega}$ and $l_{y,\omega}$ described in chapter 2. In [1] a CNN that incorporates the translational symmetry of the problem was created. The goal of this thesis is to include the reflection symmetry of the scalar field action as well.

3.1 General structure of CNNs

Most of the following discussion follows closely that of the Group equivariant CNNs (G-CNNs) from [2] just with reflections instead of rotations. Before we can discuss the structure of CNNs we first need to introduce some nomenclature. First, a feature map $f : \mathbb{Z}^2 \to \mathbb{R}$ is a real valued function on the two dimensional lattice $\mathbb{Z}^2$. A kernel (or filter) $\psi : \mathbb{Z}^2 \to \mathbb{R}$ is a real valued function on $\mathbb{Z}^2$ with finite support, i.e., only finitely many inputs return a nonzero output. We denote the space of all feature maps as $\mathcal{F} := \{f \mid f : \mathbb{Z}^2 \to \mathbb{R}\}$. Then, a convolution (or cross-correlation) is defined as

$$[f \star \psi](y) = \sum_{z \in \mathbb{Z}^2} f(z)\psi(z - y) = \sum_{z \in \mathbb{Z}^2} f(z + y)\psi(z). \tag{3.1}$$

The first component of a CNN is the convolutional layer. A convolutional layer $\Phi : \mathcal{F}^N \to \mathcal{F}^M$ is a function, which takes a set of N feature maps $G := (g_i)_{i=1}^N$ as input and has a set of $N \cdot M$ kernels $\Psi := \{\psi_j^i \mid i = 1, \ldots, N, j = 1, \ldots, M\}$ as parameters. Then, the output of $\Phi(G)$ is a set of M feature maps $H := (h_j)_{j=1}^M$ such that,

$$h_j(y) = \sum_i [g_i \star \psi_j^i](y), \forall j, y. \tag{3.2}$$

In practice, a feature map f lives on an $A \times B$ rectangular region $F \subset \mathbb{Z}^2$ that could for example be represented by a two-dimensional array in a computer. A kernel ψ is then defined on a $K \times L$ region Ψ, with $K \leq A$ and $L \leq B$. The convolution of these objects then has the form

$$[f \star \psi](y) = \sum_{z \in \Psi} f(z + y)\psi(z), \tag{3.3}$$

as in eq. (3.1), but only summing over the finite region Ψ. This can be visualized by laying the kernel over the feature map, multiplying overlapping elements and summing up all those products. For example, let f be a 3×3 feature map and ψ a 2×2 kernel. Then,

$$[f \star \psi] = \begin{array}{|c|c|c|}\hline 1&3&3\\\hline 4&2&5\\\hline 3&1&2\\\hline\end{array} \star \begin{array}{|c|c|}\hline 2&1\\\hline 0&1\\\hline\end{array} = \left\{ \begin{array}{l} \begin{array}{|c|c|c|}\hline \cellcolor{blue!10}1&\cellcolor{blue!10}3&3\\\hline \cellcolor{blue!10}4&\cellcolor{blue!10}2&5\\\hline \cellcolor{blue!10}3&\cellcolor{blue!10}1&2\\\hline\end{array} \star \begin{array}{|c|c|}\hline \cellcolor{blue!10}2&\cellcolor{blue!10}1\\\hline \cellcolor{blue!10}0&\cellcolor{blue!10}1\\\hline\end{array} = \begin{array}{|c|c|}\hline \cellcolor{blue!10}7&\\\hline &\\\hline\end{array} \\ \begin{array}{|c|c|c|}\hline 1&\cellcolor{blue!10}3&\cellcolor{blue!10}3\\\hline 4&\cellcolor{blue!10}2&\cellcolor{blue!10}5\\\hline 3&\cellcolor{blue!10}1&\cellcolor{blue!10}2\\\hline\end{array} \star \begin{array}{|c|c|}\hline \cellcolor{blue!10}2&\cellcolor{blue!10}1\\\hline \cellcolor{blue!10}0&\cellcolor{blue!10}1\\\hline\end{array} = \begin{array}{|c|c|}\hline 7&\cellcolor{blue!10}14\\\hline &\\\hline\end{array} \\ \begin{array}{|c|c|c|}\hline 1&3&3\\\hline \cellcolor{blue!10}4&\cellcolor{blue!10}2&\cellcolor{blue!10}5\\\hline \cellcolor{blue!10}3&\cellcolor{blue!10}1&\cellcolor{blue!10}2\\\hline\end{array} \star \begin{array}{|c|c|}\hline \cellcolor{blue!10}2&\cellcolor{blue!10}1\\\hline \cellcolor{blue!10}0&\cellcolor{blue!10}1\\\hline\end{array} = \begin{array}{|c|c|}\hline 7&14\\\hline \cellcolor{blue!10}11&\\\hline\end{array} \\ \begin{array}{|c|c|c|}\hline 1&3&3\\\hline 4&\cellcolor{blue!10}2&\cellcolor{blue!10}5\\\hline 3&\cellcolor{blue!10}1&\cellcolor{blue!10}2\\\hline\end{array} \star \begin{array}{|c|c|}\hline \cellcolor{blue!10}2&\cellcolor{blue!10}1\\\hline \cellcolor{blue!10}0&\cellcolor{blue!10}1\\\hline\end{array} = \begin{array}{|c|c|}\hline 7&14\\\hline 11&\cellcolor{blue!10}11\\\hline\end{array} \end{array} \right\} = \begin{array}{|c|c|}\hline 7&14\\\hline 11&11\\\hline\end{array}. \tag{3.4}$$

However, due to f being only defined on a finite region, problems occur at the boundary of F. This is easy to see in eq. (3.4) as there is no way to overlap the kernel with anything else, and as such the output of the convolution will be smaller. This problem can be handled in different ways. First, one can define the output of the convolution only for points y such that for all $z \in \Psi$, $y + z \in F$. In this case, the output feature map will be defined on a region that is smaller than F as seen in eq. (3.4). For example if Ψ is a 2×2 kernel as above and F is a $A \times B$ region, the output region will be of size $(A - 1) \times (B - 1)$. In general, if Ψ is defined on a $K \times L$ region the output will be of size $(A - K + 1) \times (B - L + 1)$.

The other possibility is to add information around the boundary of F to ensure that the output has the same size as the input. This is called padding the input and for a kernel of size $K \times L$ the padded input needs to be of size $(A + K - 1) \times (B + L - 1)$. Then, this padded input will be treated as above and results in an output with the same size as the input. There are multiple ways to pad an input. Two of the most common options are zero padding, where zeros are added around the original input, and circular padding, which creates periodic boundary

conditions. Take for example a feature map f of the form

$$\begin{bmatrix} f_{1,1} & \cdots & f_{1,B} \\ \vdots & \ddots & \vdots \\ f_{A,1} & \cdots & f_{A,B} \end{bmatrix}. \tag{3.5}$$

Then, equal zero padding of size one on all sides looks like

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & f_{1,1} & \cdots & f_{1,B} & 0 \\ 0 & \vdots & \ddots & \vdots & 0 \\ 0 & f_{A,1} & \cdots & f_{A,B} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.6}$$

and equal circular padding of size one looks like

$$\begin{bmatrix} f_{A,B} & f_{A,1} & \cdots & f_{A,B} & f_{A,1} \\ f_{1,B} & f_{1,1} & \cdots & f_{1,B} & f_{1,1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ f_{A,B} & f_{A,1} & \cdots & f_{A,B} & f_{A,1} \\ f_{1,B} & f_{1,1} & \cdots & f_{1,B} & f_{1,1}. \end{bmatrix}. \tag{3.7}$$

Equal padding on all sides is used when the kernel size is odd. Padding can also be added only on some sides, or with different sizes on different sides. For a kernel size of 4 one could for example add size two padding on the left and top sides, and size one padding on the other sides.

In general a convolutional layer can also include a bias term. That is, after the convolution, a bias is added to each output feature map. If a bias term $b = (b_j)_{j=1}^N$ is included, eq. (3.2) takes the form

$$g_j(y) = \sum_i [f_i \star \psi_j^i](y) + b_j, \forall j, y. \tag{3.8}$$

It is also possible to apply an activation function to the convolved feature map. An activation function is a function that is applied to each point of the feature map and can be nonlinear. Nonlinearities are necessary, because otherwise multiple convolutions without nonlinearities in between would collapse into a single convolution [5]. If we have a feature map $f : F \to \mathbb{R}$ with $F \subset \mathbb{Z}^2$, then we apply an activation function $\alpha : \mathbb{R} \to \mathbb{R}$ simply by composing it with f, i.e.,

$$(\alpha \circ f)(y) = \alpha(f(y)), \forall y \in F. \tag{3.9}$$

The second important layer of a CNN that we will use is a global pooling layer. Such an operation is only defined on feature maps with finite support. A global pooling layer $\Phi : \mathcal{F} \to \mathbb{R}$ takes a feature map and returns a single scalar value. Common variants of a global pooling

layer are global max pooling and global average pooling. Taking a feature map f defined on a rectangular region $F \subset \mathbb{Z}^2$ as an input, a global max pooling layer (GMP) is defined as

$$GMP(f) := \max_{y \in F} f(y), \tag{3.10}$$

and the global average pooling layer (GAP) as

$$GAP(f) := \frac{1}{|F|} \sum_{y \in F} f(y) \tag{3.11}$$

where $|F|$ is the number of points in F [6].

3.2 Group equivariant convolutional layers

First we will define equivariance. Let $\mathcal{G}$ be a group and L_g the action of the group element $g \in \mathcal{G}$ acting on functions $h : \mathbb{Z}^2 \to \mathbb{R}$, such that $[L_g h](y) = h(g^{-1}y)$. For example if $\mathcal{G}$ is the group of translations on $\mathbb{Z}^2$, then $[L_g h](y) = h(y - g)$. Thus, looking at the value of the new function at y is the same as looking at the value of the old function at the point $y - g$. Now let Φ be a convolutional layer and f a feature map. Then, Φ is called equivariant under the action of $\mathcal{G}$ if for all $g \in \mathcal{G}$ there exists a $g' \in \mathcal{G}$ such that

$$\Phi(L_g f) = L_{g'} \Phi(f). \tag{3.12}$$

If g' is the identity element for all $g \in \mathcal{G}$, i.e., $\Phi(L_g f) = \Phi(f)$, then Φ is called invariant under the group action of $\mathcal{G}$. As an example we again look at the group of translations on $\mathbb{Z}^2$. We take a convolutional layer that has only one input and one output and use eq. (3.1). Then,

$$\begin{aligned}
L_t \Phi(f) &= [L_t[f \star \psi]](y) \\
&= [f \star \psi](y - t) \\
&= \sum_{z \in \mathbb{Z}^2} f(z)\psi(z - (y - t)) \\
&= \sum_{z \in \mathbb{Z}^2} f(z)\psi((z + t) - y) \\
&= \sum_{z \in \mathbb{Z}^2} f(z - t)\psi(z - y) \\
&= \sum_{z \in \mathbb{Z}^2} [L_t f](z)\psi(z - y) \\
&= [[L_t f] \star \psi](y) \\
&= \Phi(L_t f)
\end{aligned} \tag{3.13}$$

and therefore a convolution is equivariant under translations. Additionally, $g = g' = t$. However, this only works when f is defined on all of $\mathbb{Z}^2$. If the feature map is only defined on a

finite region, the convolutional layer must employ circular padding, i.e., the feature map must have periodic boundary conditions.

Now we look at the transformations defined in eqs. (2.25) and (2.26). We will show the equivariance properties for the reflection in the x direction for $k_{y,x}$. Looking at the first line of eq. (2.25) the transformation for a feature map $K_x = (k_{y,x})_{y\in F}$ defined on a region $F \subset \mathbb{Z}^2$ can be seen as simply reflecting the feature map around the t axis, shifting it by a unit vector in the x direction, and then multiplying the new feature map with -1. We call this transformation $R_{k_x,x}$. Similar, in the first line of eq. (2.26) the transformation on K_x can be seen as simply reflecting the feature map around the x axis. We call this transformation $R_{k_x,t}$. Eq. (3.14) and (3.15) show both transformations. In a row the x dimension is held constant while for columns the t dimension is held constant.

$$K_x = \begin{bmatrix} f_{1,1} & \cdots & f_{1,M} \\ \vdots & \ddots & \vdots \\ f_{N-1,1} & \cdots & f_{N-1,M} \\ f_{N,1} & \cdots & f_{N,M} \end{bmatrix} \rightarrow L_{R_{k_x,x}} K_x = \begin{bmatrix} -f_{N-1,1} & \cdots & -f_{N-1,M} \\ \vdots & \ddots & \vdots \\ -f_{1,1} & \cdots & -f_{1,M} \\ -f_{N,1} & \cdots & -f_{N,M} \end{bmatrix} \tag{3.14}$$

$$K_x = \begin{bmatrix} f_{1,1} & \cdots & f_{1,M} \\ \vdots & \ddots & \vdots \\ f_{N,1} & \cdots & f_{N,M} \end{bmatrix} \rightarrow L_{R_{k_x,t}} K_x = \begin{bmatrix} f_{1,M} & \cdots & f_{1,1} \\ \vdots & \ddots & \vdots \\ f_{N,M} & \cdots & f_{N,1} \end{bmatrix} \tag{3.15}$$

It is easy to see that $(R_{k_x,\omega})^2 = \mathbb{1}$ and $R_{k_x,x}R_{k_x,t} = R_{k_x,t}R_{k_x,x}$. The symmetry group is then $\mathcal{R}_{k_x} = \{\mathbb{1}, R_{k_x,x}, R_{k_x,t}, R_{k_x,x}R_{k_x,t}\}$. Thus for any $r \in \mathcal{R}_{k_x}$,

$$\begin{aligned} [[L_r K_x] \star \psi](y) &= \sum_{z\in\mathbb{Z}^2} K_x(rz)\psi(z-y) \\ &= \sum_{z\in\mathbb{Z}^2} K_x(z)\psi(rz-y) \\ &= \sum_{z\in\mathbb{Z}^2} K_x(z)\psi(r(z-ry)) \\ &= \sum_{z\in\mathbb{Z}^2} K_x(z)[L_r\psi](z-ry) \\ &= [L_r[K_x \star [L_r\psi]]](y), \end{aligned} \tag{3.16}$$

and as such the convolution is not equivariant under this symmetry group. In the first step we made the change of variable $z \rightarrow rz$ and used that $r^{-1} = r$. For the other flux variables, the symmetry groups have the same structure just with slightly different transformations. Thus, for all flux variables the convolutions are not equivariant.

However, this can be circumvented by taking not just a single feature map as input, but instead a stack of all possible transformations applied to the feature map. Starting with a single feature

map, we take $(L_r K_x)_{r\in\mathcal{R}_{k_x}}$ as input and convolve it with a single kernel ψ to get a stack of feature maps $G_r := [[L_r K_x] \star \psi]$ as output. Then, applying a transformation $s \in R_{k_x}$ to the whole input stack we get

$$G'_r = [L_s[L_r K_x]] \star \psi = [[L_{sr} K_x]\psi] = G_{sr}. \tag{3.17}$$

For the transformation group $\mathcal{R}_{k_x}$ the stack $(G_{sr})_{r\in\mathcal{R}_{k_x}}$ is simply a permutation of the elements in $(G_r)_{r\in\mathcal{R}_{k_x}}$. Written out, the input stack has the form

$$(K_x^0 := K_x, K_x^1 := L_{R_{k_x,x}} K_x, K_x^2 := K_{R_{k_x,t}} K_x, K_x^3 := L_{R_{k_x,x} R_{k_x,t}} K_x). \tag{3.18}$$

Then, for example for $s = R_{k_x}$ we have

$$L_{R_{k_x,x}} K_x^0 = L_{R_{k_x,x}} K_x = K_x^1, \tag{3.19}$$
$$L_{R_{k_x,x}} K_x^1 = L_{R_{k_x,x}} L_{R_{k_x,x}} K_x = L_{(R_{k_x,x})^2} K_x = K_x = K_x^0, \tag{3.20}$$
$$L_{R_{k_x,x}} K_x^2 = L_{R_{k_x,x}} L_{R_{k_x,t}} K_x = L_{R_{k_x,x} R_{k_x,t}} K_x = K_x^3, \tag{3.21}$$
$$L_{R_{k_x,x}} K_x^3 = L_{R_{k_x,x}} L_{R_{k_x,x} R_{k_x,t}} K_x = L_{(R_{k_x,x})^2} L_{R_{k_x,t}} K_x = L_{R_{k_x,t}} K_x = K_x^2, \tag{3.22}$$

which is simply the permutation $K_x^0 \leftrightarrow K_x^1, K_x^2 \leftrightarrow K_x^3$. Since the outputs are all convolved with the same kernel the output stack is permuted the same way as the input stack. Thus, if we view the stacks as unordered sets and ignore the permutation we have

$$\{[[L_r K_x^i] \star \psi]\}_{i=0}^3 = \{L_r[K_x^i \star \psi]\}_{i=0}^3. \tag{3.23}$$

The calculation above works analogously for all the other flux variables and their respective transformation groups and thus, this concept is the basis on which the equivariant CNN was implemented. However, in the implementation of the network it is critical to keep track of the permutations.

3.3 Implementation of group equivariant convolutional layers

The input to the CNN will be a set of four feature maps, one for each of the flux variables. This immediately creates complications for the network structure. Adding up two feature maps of different flux variables creates a feature map with no well-defined symmetry transformations, as all flux variables transform differently. Thus, it is not possible to mix different flux variables. For this reason, we split the network into four distinct parts, one for each flux variable. As all four parts of the network behave similarly, just with different transformation behaviors, we will from now on only focus on the K_x part of the network.

We implement a network that is equivariant under translations and reflections. The trans-

lational part is handled by periodic boundary conditions that are implemented via circular padding. For the reflections we will proceed similarly to the previous section. The only difference to the above approach is, that we transform the kernels instead of the feature maps, as this is computationally more efficient. There are two cases to be considered for a convolutional layer. The first layer takes only a single feature map K_x as input, while all further layers take stacks of feature maps as input where each stack is equivariant under reflections.

To start we look at the first layer. For simplicity we assume that the layer has only one output stack $\mathcal{G} := (G_s)_{s \in R_{k_x}}$ and a single kernel ψ. As already stated above we will transform the kernel instead of the input K_x for computational efficiency. However, by transforming the kernel instead of the input, a transformation of the input does not simply lead to a permutation in the output stack. To see this let $r, s \in \mathcal{R}_{k_x}$. Then,

$$\begin{aligned}
[[L_r K_x] \star [L_s \psi]](y) &= \sum_{z \in \mathbb{Z}^2} K_x(rz)\psi(s(z-y)) \\
&= \sum_{z \in \mathbb{Z}^2} K_x(z)\psi(s(rz-y)) \\
&= \sum_{z \in \mathbb{Z}^2} K_x(z)\psi(sr(z-ry)) \\
&= [L_r[K_x \star [L_{rs}\psi]]](y), \qquad (3.24)
\end{aligned}$$

where we used the variable transformation $z \to rz$, redefined the sum from line one to line two, and used $r^{-1} = r$. If we look at the whole output stack $(G_s)_{s \in \mathcal{R}_{k_x}} := ([K_x \star [L_s\psi]])_{s \in \mathcal{R}_{k_x}}$, we can see that

$$[[L_r K_x] \star [L_s \psi]] = L_r G_{rs}. \qquad (3.25)$$

Thus, the output stack is not only permuted, but also has the same transformation L_r applied to each element of the stack. But this way, the stack is still clearly equivariant under the action of the group.

However, the action of the group $\mathcal{R}_{k_x}$ on the kernels is not the same as on the K_x. The problem is that shifting a kernel that is only defined on a compact region is not well-defined. Thus, the approach was to require the transformation behavior

$$[[L_r K_x] \star [L'_s \psi]] = [L_r[K_x \star [L'_{rs}\psi]]] \qquad (3.26)$$

for $r, s \in \mathcal{R}_{k_x}$, with some unknown transformation behavior $L'_s\psi$ of the kernel and derive from that the action of $\mathcal{R}_{k_x}$ on ψ. Then, from the full transformation behavior of $\mathcal{R}_{k_x}$ only the reflections of each transformation act on the kernel, i.e., if we write $\psi = \psi(y, z), (y, z) \in \Psi \subset \mathbb{Z}^2$, then

$$\psi^0(y,z) = L'_{\mathbb{1}}\psi(y,z) = \psi(y,z), \tag{3.27}$$
$$\psi^1(y,z) = L'_{R_{k_1,1}}\psi(y,z) = \psi(-y,z), \tag{3.28}$$
$$\psi^2(y,z) = L'_{R_{k_1,2}}\psi(y,z) = \psi(y,-z), \tag{3.29}$$
$$\psi^3(y,z) = L'_{R_{k_1,1}R_{k_1,2}}\psi(y,z) = \psi(-y,-z). \tag{3.30}$$

In Appendix D we work through an example that shows how this transformation of the kernels indeed leads to the transformation of the output stack as required in eq. (3.26).

Now for all further layers we have stacks of feature maps as input. The concept is the same as for the first layer. However, we also have to keep track of the permutations the stacks undergo if the input is transformed. Figure 3.1 (top-left) shows a network with a single input feature map and two layers each with a single output stack. The other parts of the figure show how the output stacks for both layers should transform if the input is transformed. In order for the network to be equivariant, all layers after the first have to have the same transformation property as the first output stack. Figure 3.1 shows, how for each element H_i^j of the output stack of the second layer, the kernels have to be permuted in order to achieve the same transformation behavior as the output stack of the first layer. Let K_x be the input of the network and ψ the kernel of the first input layer. Then, the output stack of the first layer is defined as

$$G_i = [K_x \star \psi^{i-1}], \tag{3.31}$$

where the superscript indices define the transformed kernels or feature maps as in eqs. (3.27)-(3.30) or (3.18) respectively. For the second layer we now need to take into account the permutation behavior of the output stack of the first layer. We define the first output H_1 of the second layer as

$$H_1 := \sum_j [G_j \star \phi_j], \tag{3.32}$$

with the set of kernels $(\phi_j)_{j=1}^4$. We also define the other outputs of the second layer as

$$H_i := \sum_j [G_j \star \sigma_{j,i}], \tag{3.33}$$

for $i \in \{2,3,4\}$. Now we need to find out how the kernels $\sigma_{j,i}$ relate to the ϕ_j. First we note that the H_i must be permuted in the same way as the G_i if the input is transformed, e.g., if $K_x^0 \to K_x^1$ then $H_1^0 \to H_2^1, H_2^0 \to H_1^1, H_3^0 \to H_4^1$, and $H_4^0 \to H_3^1$. As an example we look at the transformation $K_x^0 \to K_x^1$. Then, we have for the second output of the second layer

$$H_2' = \sum_j [G_j^1 \star \sigma_{j,2}] \overset{!}{=} H_1^1 = \sum_j [G_j^1 \star \phi_j^1]. \tag{3.34}$$

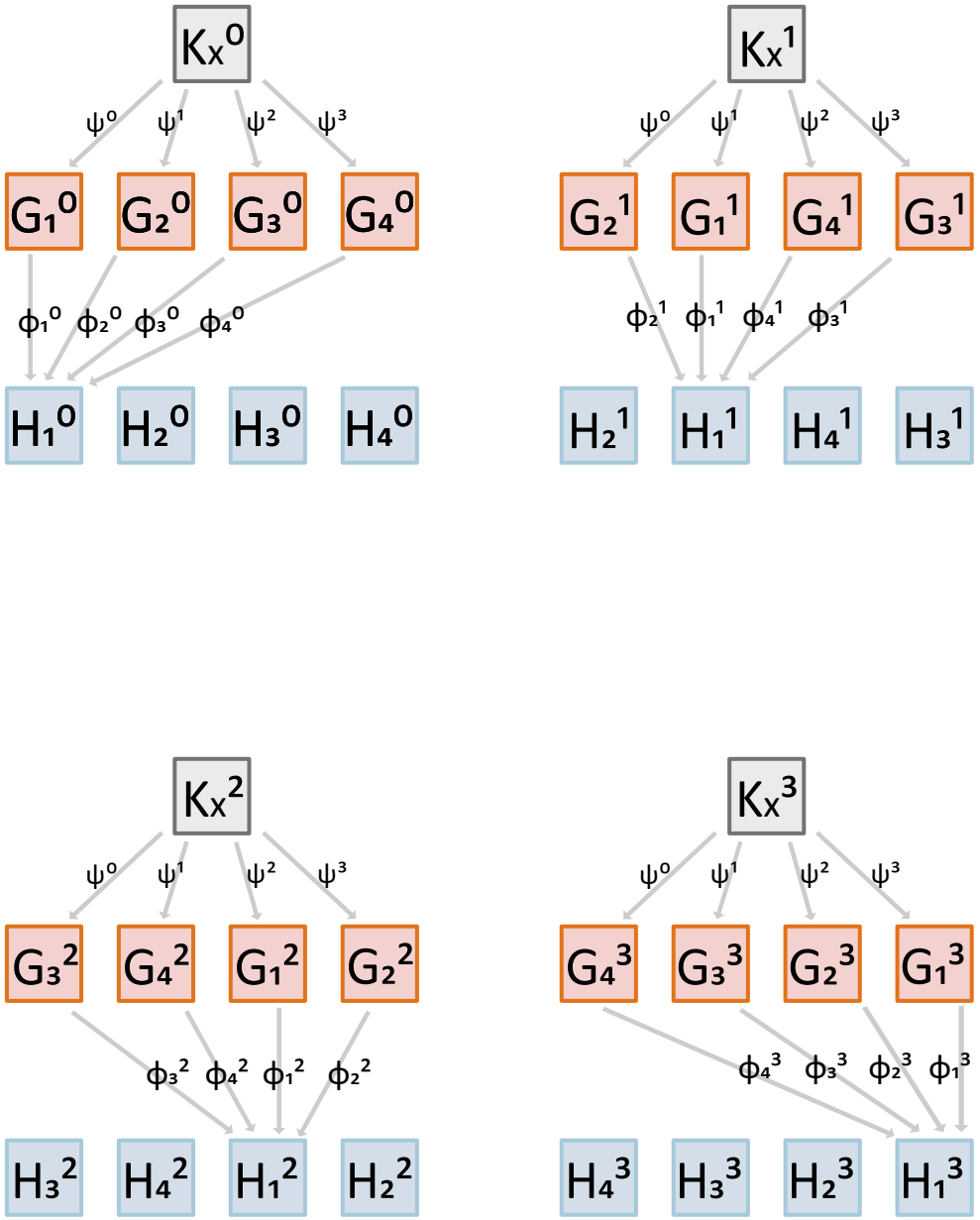

Figure 3.1: Network structure for a single input feature map and a single output stack for each layer. Top-left: The input is untransformed. The output of the second layer is defined as $H_1^0 = \sum_i [G_i^0 \star \phi_i^0]$ where the G_i^0 are the outputs of the first layer with K_x^0 as input. Others: These show how the kernels have to be permuted such that the output of the second layer has the same transformation property as the output of the first layer, e.g., if $K_x^0 \to K_x^1$ then $H_1^0 \to H_2^1, H_2^0 \to H_1^1, H_3^0 \to H_4^1$, and $H_4^0 \to H_3^1$. The superscript indices are shorthand for the transformed kernels or feature maps as defined in eqs. (3.27)-(3.30) or (3.18) respectively.

Thus,

$$\sigma_{1,2} = \phi_2^1, \sigma_{2,2} = \phi_1^1, \sigma_{3,2} = \phi_4^1, \sigma_{4,2} = \phi_3^1. \tag{3.35}$$

Doing the same for the other output results in

$$\sigma_{1,3} = \phi_3^2, \sigma_{2,3} = \phi_4^2, \sigma_{3,3} = \phi_1^2, \sigma_{4,3} = \phi_2^2 \tag{3.36}$$

and

$$\sigma_{1,4} = \phi_4^3, \sigma_{2,4} = \phi_3^3, \sigma_{3,4} = \phi_2^3, \sigma_{4,4} = \phi_1^3 \tag{3.37}$$

as seen in Figure 3.1.

Lastly, we discuss how to extend the above behavior to more than a single output stack. For the first layer this is trivial. Let K_x be the input, N the number of output stacks, $(\psi_i)_{i=1}^N$, and $(\mathcal{G}_i)_{i=1}^N$ the output stacks with $\mathcal{G}_i = (G_{i,j})_{j=1}^4$. Then,

$$G_{i,j} := [K_x \star \psi_i^{j-1}]. \tag{3.38}$$

For all further layers we again have to keep track of the permutations of the stacks. We take for some layer after the first $(\mathcal{G}_i)_{i=1}^N$ as input. Since we have $4N$ feature maps in total as input, we also need $4N$ kernels for each output. If we have M outputs, we have $4NM$ kernels $(\phi_{i,j,k}), i \in \{1, \dots, N\}, j \in \{1, \dots, 4\}, k \in \{1, \dots, M\}$. Defining the output stacks as $(\mathcal{H}_i)_{i=1}^M$ and $\mathcal{H}_i := (H_{i,j})_{j=1}^4$, we have

$$H_{k,1} := \sum_{i,j} [G_{i,j} \star \phi_{i,j,k}^0], \tag{3.39}$$

for the first element of each output stack, where the superscript index refers to the transformed kernels as in eqs. (3.27)-(3.30). For the other elements we permute the kernels and get

$$H_{k,2} := \sum_i \left([G_{i,1} \star \phi_{i,2,k}^1] + [G_{i,2} \star \phi_{i,1,k}^1] + [G_{i,3} \star \phi_{i,4,k}^1] + [G_{i,4} \star \phi_{i,3,k}^1] \right) \tag{3.40}$$

$$H_{k,3} := \sum_i \left([G_{i,1} \star \phi_{i,3,k}^2] + [G_{i,2} \star \phi_{i,4,k}^2] + [G_{i,3} \star \phi_{i,1,k}^2] + [G_{i,4} \star \phi_{i,2,k}^2] \right) \tag{3.41}$$

$$H_{k,4} := \sum_i \left([G_{i,1} \star \phi_{i,4,k}^3] + [G_{i,2} \star \phi_{i,3,k}^3] + [G_{i,3} \star \phi_{i,2,k}^3] + [G_{i,4} \star \phi_{i,1,k}^3] \right). \tag{3.42}$$

The structure of these $H_{k,i}$ can now be continued for an arbitrary number of layers.

The above calculations have been done without including a possible bias term as described in eq. (3.8). For the $l_{y,\omega}$ feature maps, adding a bias does not interfere with the equivariance if we use the same value for the whole stack. For the $k_{y,\omega}$ feature maps however, adding a bias term interferes with the equivariance of the transformation. The problem stems from the multiplication with -1, where it does matter if it is multiplied before or after adding the bias. This problem does not occur for the $l_{y,\omega}$ feature maps. Thus, we add a bias term only to the $l_{y,\omega}$ outputs of each convolutional layer.

To add an activation function, as described in eq. (3.9), after a convolutional layer we also have to check if it is compatible with the transformation behavior. Since the function is applied pointwise the reflection and shift do not add any requirement to the activation function. However, for the $k_{y,\omega}$ feature maps the multiplication with -1 requires the activation function to be an odd function. Thus, if we choose to apply an activation function after a convolutional layer, we have to use an odd function.

In the end, to get an output that can either be directly used to predict observables or that can be fed into a dense network we need to apply a global pooling layer to the output of the last convolutional layer. In order to have an equivariant output we need to adjust the global pooling defined in eqs. (3.10), (3.11) slightly. Instead of pooling over only a single feature map, we need to pool over the whole output stack and take the absolute value of the result. If we have a set of M output stacks $(\mathcal{H}_i)_{i=1}^{M}$ and each stack is defined as $\mathcal{H}_i := (H_{i,j})_{j=1}^{4}$, then the global average pooling (GAP) of $\mathcal{H}_i$ will take the form

$$GAP(\mathcal{H}_i) := \left| \frac{1}{4|F|} \sum_{j=1}^{4} \sum_{y \in F} H_{i,j}(y) \right|. \tag{3.43}$$

Therefore, the output of the global pooling layer is invariant under the chosen transformations and can be freely combined in the dense linear layers.

Chapter 4

Predicting observables

We will now define an equivariant network using the components described in the previous chapter. The goal is to compare the resulting network to the networks of [1]. Thus, we will use the same input data. In [1] the particle density n and the lattice averaged squared absolute value of the field $|\phi|^2$ were used. They are given by the equations

$$n = \frac{T}{V}\frac{\partial \ln Z}{\partial \mu} = \frac{1}{N_x N_t}\sum_y k_{y,t} \tag{4.1}$$

and

$$|\phi|^2 = -\frac{T}{V}\frac{\partial \ln Z}{\partial \eta} = \frac{1}{N_x N_t}\sum_y \frac{W(f_y+2)}{W(f_y)}. \tag{4.2}$$

We however use the absolute value of the particle density $|n|$ instead of n since the output of the global pooling layer is invariant under all transformations while n changes the sign. From now on we will refer to the absolute value of the particle density by n.

N_x, N_t are the lattice dimensions, and the functions $W(\cdot)$ and f_y are given by eqs. (2.18) and (2.19) respectively. Both n and $|\phi|^2$ depend on the physical parameters λ, η, μ. For this task their values are set to $\lambda = 1, \eta = 4 + m^2 = 4.01, \mu = 1.05$ as in [7] and [1].

4.1 Architecture optimization

As input to the network we have four feature maps, one for each flux variable. Those inputs are fed through a number of convolutional layers as defined in chapter 3, with a bias term only for the $l_{y,\omega}$ part of the network. After each convolutional layer, we apply a *Tanh* activation function, as it is an odd function. Then, the output of the last convolutional layer is fed through a global average pooling layer after the *Tanh* activation function. After the global pooling, the output is invariant under the transformations of the flux variables, and thus it can either be used directly as an output or it can be fed into a dense linear network. The linear network consists of a number of fully connected layers, with a *LeakyReLU* activation function after

each layer but the last. The last layer must have two outputs that are used to predict the two observables n and $|\phi|^2$. If no linear network is added, the last convolutional layer must have two output stacks and no activation function must be applied afterwards. The result of the global pooling layer can then be used again to predict the two observables.

With this general structure, we use the optimization framework *optuna* [8] to find network architectures with low validation loss. We use the same setup as [1]. The validation loss averaged over three different parameter-initializations is used as a metric for the performance of the chosen architecture. The best performing architecture from the defined search space is then retrained ten times to account for fluctuations in the validation loss. This architecture can then be compared to the best performing equivariant network of [1].

After performing an initial parameter search, the search space of Table 4.1 was chosen to start the optimization procedure with the *optuna* framework. The initial parameter search was done with the best performing network of [1] in mind. Compared to [1] the number of channels in Table 4.1 is divided by four, as a channel here contains a feature map for each of the flux variables. Kernel sizes larger than (2×2) were already excluded in the initial parameter search. The single linear layer is already fully defined by the number of output channels of the global pooling layer.

	conv	lin	kernel size	channels
search space	$[2, 4]$	1	$\{(1 \times 1), (2 \times 2)\}$	$\{10, 13, 16, 19, 22\}$

Table 4.1: Search space for the optimization procedure. This table lists the number of allowed convolutional layers (conv), the number of allowed linear layers (lin), the different allowed kernel sizes for each convolutional layer (kernel size), and the allowed number of channels for each convolutional layer (channels).

The optimization procedure was done separately for different sizes of the training data set, as different architectures might perform differently for differently sized training sets. Indeed, for smaller training sets the best performing architecture had only two convolutional layers, while for larger sizes it had three convolutional layers. The best performing architectures found by the optimization procedure can be found in Table 4.2.

4.2 Training and testing

For the training process, the *AMSGrad* [9] version of the *AdamW* [10] optimizer with vanishing weight decay was used. As a loss function the mean squared error (MSE) was used, where the total loss was the arithmetic mean of the individual losses for each of the two observables. We use the same data sets as in [1]. The training set contains a total of 20000 samples. Each sample contains a feature map for each of the four flux variables of size 60×4, with the first being the size in the temporal dimension and the second in the spatial dimension. The validation set contains 2000 samples of the same shape. The network was trained on random

subsets of the training set with varying sizes. The validation set was chosen to be 10% of the chosen training set size. As training set sizes we chose $\{50, 200, 2000, 20000\}$ to find the best performing architectures with the *optuna* framework. The best performing architectures for each training set size can be found in Table 4.2.

# training samples	arc_{50}	arc_{200}
	Conv$(1 \times 1, 1, 13, 1)$ *Tanh* Conv$(1 \times 1, 13, 13, 4)$ *Tanh* GlobalAvgPool Linear$(52, 2)$	Conv$(2 \times 2, 1, 19, 1)$ *Tanh* Conv$(1 \times 1, 19, 19, 4)$ *Tanh* GlobalAvgPool Linear$(76, 2)$
trainable parameters	2914	6310
# training samples	arc_{2000}	arc_{20000}
	Conv$(2 \times 2, 1, 10, 1)$ *Tanh* Conv$(1 \times 1, 10, 13, 4)$ *Tanh* Conv$(1 \times 1, 13, 10, 4)$ *Tanh* GlobalAvgPool Linear$(40, 2)$	Conv$(2 \times 2, 1, 10, 1)$ *Tanh* Conv$(2 \times 2, 10, 10, 4)$ *Tanh* Conv$(1 \times 1, 10, 19, 4)$ *Tanh* GlobalAvgPool Linear$(76, 2)$
trainable parameters	4468	9832

Table 4.2: Best architectures for different numbers of training samples predicting the two observables n and $|\phi|^2$ found by the optimization framework *optuna*. The description of the convolutional layer has the form Conv(*kernel size, input channels, output channels, input stack size*). Each input and output channel contains a set of four feature maps, one for each of the flux variables. Input stack size is 1 for the first convolutional layer and 4 for all others. The linear layer has the form Linear(*input channels, output channels*). Since the linear layer can combine outputs of the global pooling layer from different flux variables the number of input channels is four times larger than the number output channels of the last convolutional layer.

The best performing architectures as well as the three best performing architectures of [1] were then retrained 10 times on each of the four training set sizes. The three architectures of [1] are a translationally equivariant architecture (TE), a strided architecture (ST), and a flattened architecture (FL). All three are listed in Table 4.3. For more details see [1].

The testing was then done in two steps. First all networks were tested on test samples with the same parameters as the training and validation set and 60×4 feature maps. The networks were then compared by their test losses. In the second step the trained networks were tested on differently sized feature maps. This is possible since the convolutional layers all preserve the size of the feature maps and the global pooling layer reduces each feature map to a single value independent of size, the arc_i (as in Table 4.2) and TE network structure supports arbitrary input sizes. For this we used feature maps of size 50×2 and 100×5. The ST and FL

TE	***ST***	***FL***
Conv$(1 \times 1, 4, 64)$	Conv$(1 \times 1, 4, 80)$	Conv$(1 \times 1, 4, 64)$
LeakyReLU	*LeakyReLU*	*LeakyReLU*
Conv$(1 \times 1, 64, 48)$	Conv$(1 \times 1, 80, 80)$	Conv$(2 \times 2, 64, 80)$
LeakyReLU	*LeakyReLU*	*LeakyReLU*
Conv$(1 \times 1, 48, 80)$	Conv$(1 \times 1, 80, 48)$	AvgPool$(2 \times 2, 2)$
LeakyReLU	*LeakyReLU*	Conv$(1 \times 1, 80, 48)$
Conv$(2 \times 2, 80, 80)$	AvgPool$(2 \times 2, 2)$	*LeakyReLU*
LeakyReLU	Conv$(2 \times 2, 48, 80)$	Conv$(2 \times 2, 48, 64)$
GlobalAvgPool	*LeakyReLU*	*LeakyReLU*
Linear$(80, 2)$	GlobalAvgPool	AvgPool$(2 \times 2, 2)$
	Linear$(80, 2)$	Conv$(1 \times 1, 64, 24)$
		Flatten
		Linear$(360, 24)$
		LeakyReLU
		Linear$(24, 2)$
33202	26370	47394

Table 4.3: The three best performing architectures (*TE*, *ST*, *FL*) from [1] predicting the two observables n and $|\phi|^2$. The description of the convolutional layer has the form Conv(*kernel size, input channels, output channels*). Each input and output channel contains a single feature map as different flux variables can be mixed in the convolutional layers in this network structure. The linear layer has the form Linear(*input channels, output channels*). The average pooling layer has the form Avg(*kernel size, stride*). A $(2N, 2M)$ feature map will have size (N, M) after the average pooling layer. The last row shows the trainable parameters of each architecture.

architectures were only compared on the original 60×4 lattice size.

Lastly, we also used *optuna* to find the best performing architectures for smaller training set sizes. This was done to see what happens to the test loss for smaller training set sizes. The chosen training set sizes were $\{1, 2, 5, 10, 12, 14, 16, 18, 20, 30, 40\}$. The validation set size was chosen to be 5 for all training set sizes. The best performing architectures were similar for all training set sizes. We chose just one to compare. The chosen architecture, arc_{min}, can be found in Table 4.4.

arc_{min}	layers	trainable parameters
	Conv$(1 \times 1, 1, 19, 1)$ *Tanh* Conv$(1 \times 1, 19, 19, 4)$ *Tanh* GlobalAvgPool Linear$(76, 2)$	6082

Table 4.4: Best architecture for the training set sizes $\{1, 2, 5, 10, 12, 14, 16, 18, 20, 30, 40\}$ predicting the two observables n and $|\phi|^2$ found by the optimization framework *optuna*. The description of each layer follows that of Table 4.2

4.3 Results

First we compare the four best performing architectures for each of the training set sizes in Figure 4.1. For shorthand we call the architectures as in Table 4.2. In the top plot of Figure 4.1 we can see that arc_{20000} performs the worst on the whole 60×4 test set for all training set sizes. The other architectures perform similarly to each other for all training set sizes. The overall test loss on the whole 60×4 test set lies between 10^{-2} and 10^{-5}. For all architectures the test loss decreases slightly with increasing training set size.

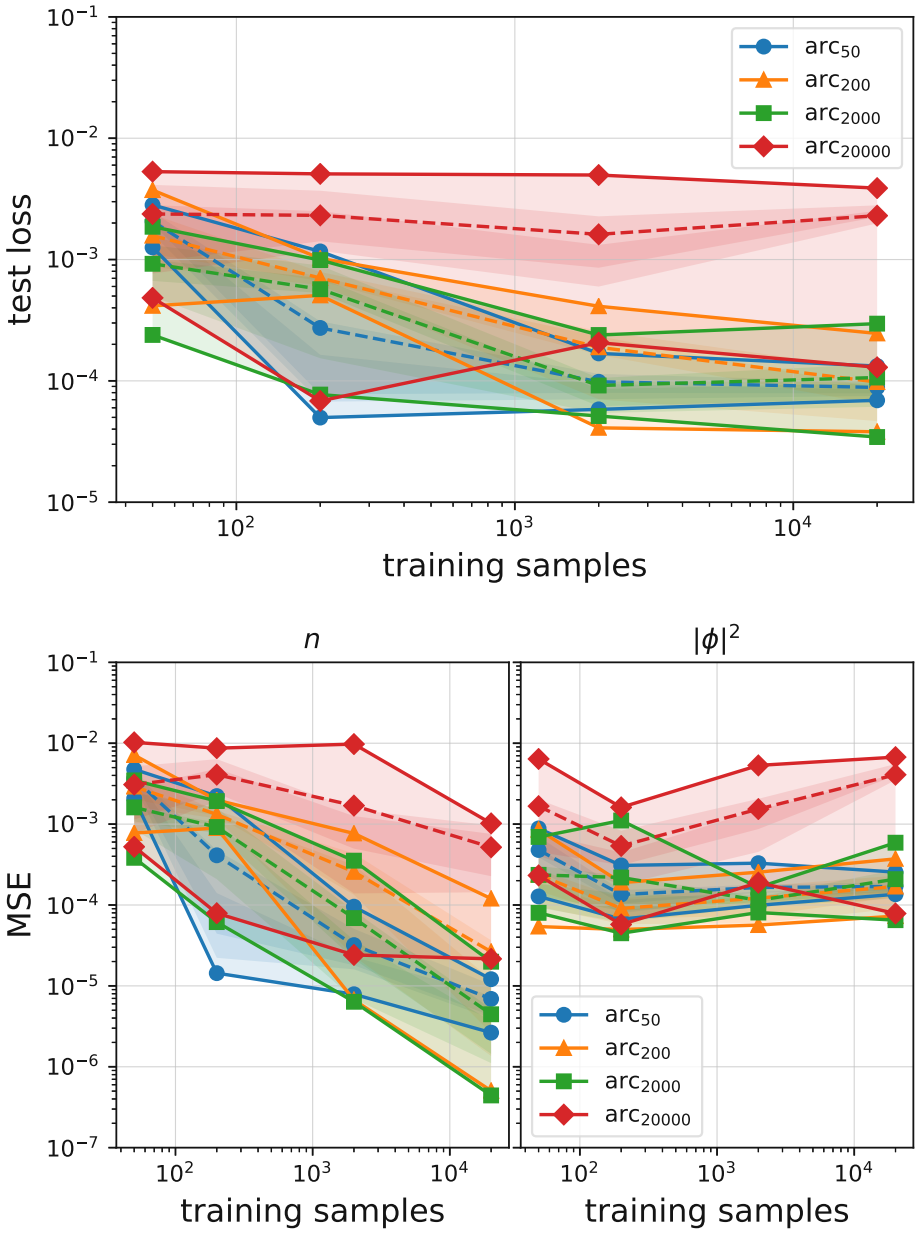

Figure 4.1: Test loss (top) and MSEs for each observable, n and $|\phi|^2$, (bottom) for each of the four best performing architectures of Table 4.2 over training set size on the whole test set of 60×4 lattices. For each architecture the top and bottom (solid) lines represent the worst and best losses and the middle (dashed) line the arithmetic mean over all ten reinitializations of the network parameters. The darker shaded areas represent the 20% quantiles. Only the symbols on the lines represent actual measurements, the lines are only for better visualization.

When looking at the MSE for each observable separately (see Figure 4.1 bottom) we can see that all architectures are far better at predicting n than at predicting $|\phi|^2$. Furthermore, the MSE for n decreases with increasing training set size by multiple orders of magnitude, while for $|\phi|^2$ the MSE is approximately independent of the training set size.

Figure 4.2 shows the test loss for each of the four architectures on the different lattice sizes 50×2, 60×4, and 100×5. The training set size was chosen to be 20000 samples. Again arc_{20000} performs the worst while the other three architectures perform similar to each other. arc_{50} performs similar on all three lattice sizes. The other architectures each have similar performance on the 60×4 and 100×5 lattices while they perform worse on the 50×2 lattice. Another thing to note is that the test losses for all reinitializations of arc_{50} are spread far less than for all the other architectures.

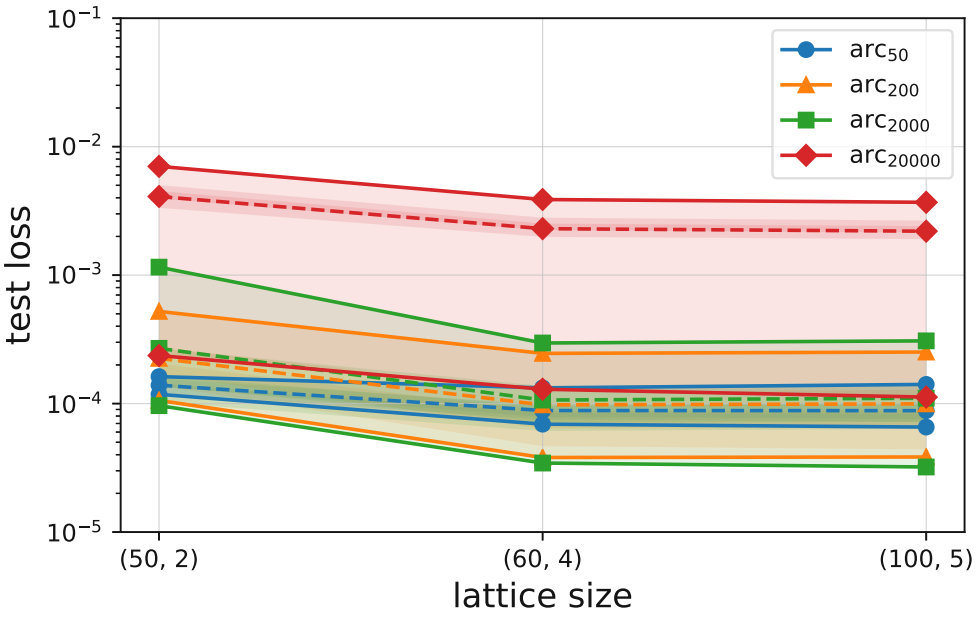

Figure 4.2: Test loss for each of the four best performing architectures of Table 4.2 over three different lattice sizes $\{50\times2, 60\times4, 100\times5\}$. The training set size was chosen to be 20000 samples. Lines and shaded areas defined as in Figure 4.1.

Next we compare the architectures found in this thesis with the *TE*, *ST*, and *FL* architectures of [1]. In order not to overfill the plots with information, we will only compare arc_{50} with the best performing architectures of [1]. Since arc_{20000} performs worse and the other two architectures perform similarly well compared to arc_{50}, no information is lost by this.

In Figure 4.3 we compare arc_{50} with the *TE*, *ST*, and *FL* architecture of [1]. In the top plot we see the overall test losses of all architectures over the different training set sizes. The *TE* architecture performs far better than arc_{50}. For the largest training set size the test losses are separated by three orders of magnitude. Furthermore, the test loss of the *TE* decreases more with increasing training set size than to the test loss of arc_{50}. The test loss of *ST* is comparable to arc_{50}, while the *FL* architecture performs worse.

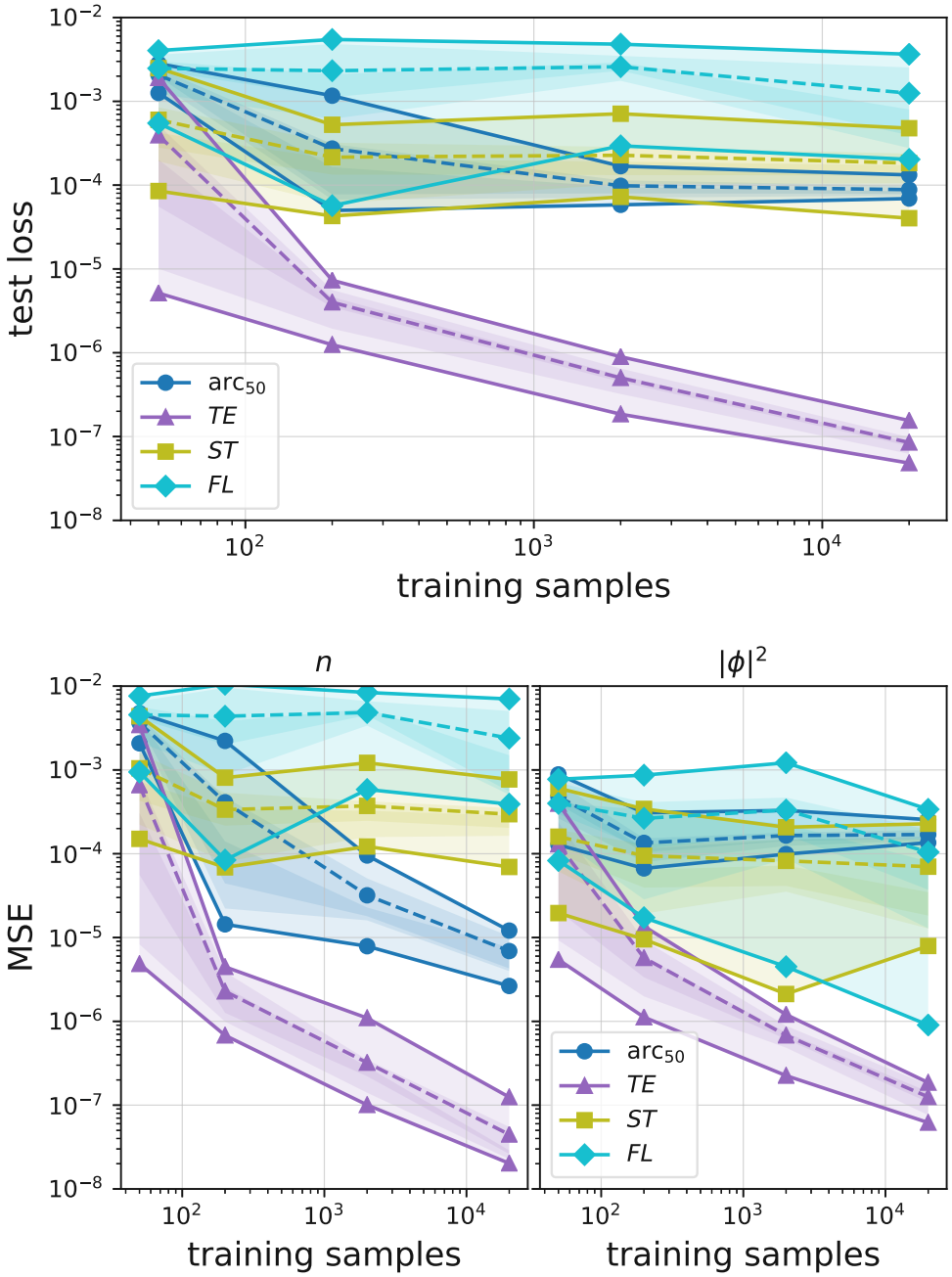

Figure 4.3: Test loss (top) and MSEs for each observable, n and $|\phi|^2$, (bottom) for arc_{50} and the *TE*, *ST*, and *FL* architectures of [1] over training set size on the whole test set of 60×4 lattices. Lines and shaded areas defined as in Figure 4.1.

When looking at the MSEs for each observable separately (see Figure 4.3 bottom), we see that for arc_{50} the MSE of $|\phi|^2$ is the main contribution to the test loss. However, for the best *TE* architecture both MSEs are roughly of the same size. Additionally, we see that arc_{50} is able to predict n better than *ST* and *FL*. For $|\phi|^2$ the three architectures perform similarly.

Figure 4.4 shows the test loss of arc_{50} and *TE* over the different lattice sizes 50×2, 60×4, and 100×5. All reinitializations of the training process are done with a training set size of 20000 samples. Both networks can be seen to generalize to other lattice sizes. However, the test loss

for arc_{50} is roughly three orders of magnitude larger on all lattice sizes compared to the test loss of the best *TE* architecture.

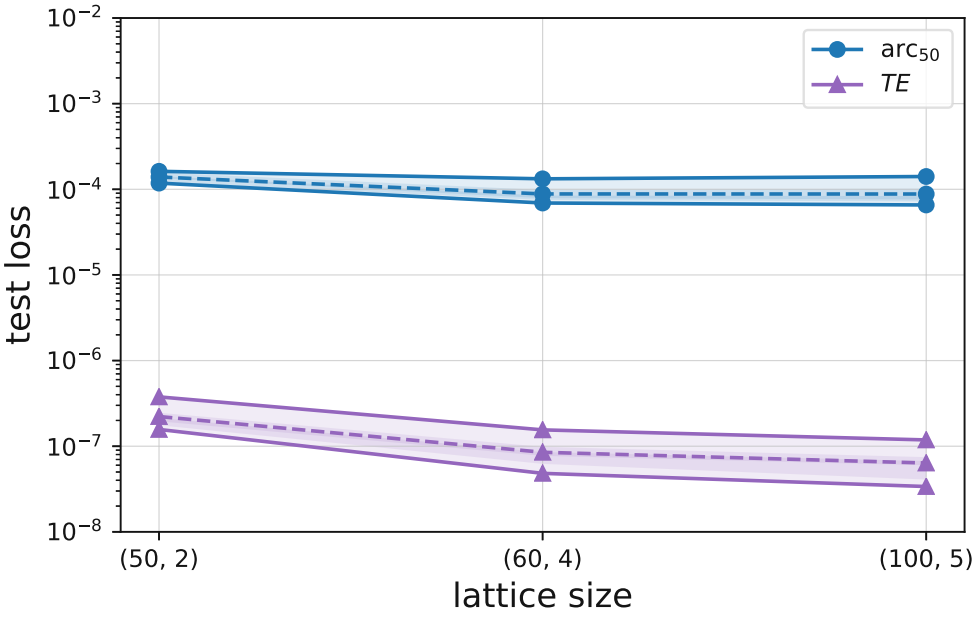

Figure 4.4: Test loss for arc_{50} and the *TE* architecture of [1] over three different lattice sizes $\{50 \times 2, 60 \times 4, 100 \times 5\}$. The training set size was chosen to be 20000 samples. Lines and shaded areas defined as in Figure 4.1.

Lastly, Figure 4.5 shows the test losses and MSEs for both observables of arc_{min} for the training set sizes $\{1, 2, 5, 10, 12, 14, 16, 18, 20, 30, 40\}$. We can see in the top plot that the test loss stays roughly around 10^{-3} for most training set sizes and only starts to rapidly increase for fewer than 5 training samples. The MSEs for both observables behave similarly with the MSE for $|\phi|^2$ being smaller than the one for n.

4.4 Discussion of the results

We have seen that the newly found networks do not perform as well as the *TE* architecture. However, when compared with the *ST* and *FL* architectures arc_{50} performs similarly well and also outperforms both when only looking at the observable n. In general, all four newly found architectures are able to predict n better than $|\phi|^2$. Due to their transformation behaviour, all four flux variables have to stay separated throughout the convolutional part of the network. Thus, no architecture can learn local influences the different flux variables have on each other. The only part where information between them can be shared is in the linear part of the network after the global pooling layer. This could explain why in Figure 4.1 the MSE for n is far lower than the one for $|\phi|^2$. Looking at eqs. (4.1) and (4.2), n depends only on the flux variable $k_{y,t}$ while $|\phi|^2$ depends on all four flux variables via the function f_y.

Reference [1] already showed that there exist symmetries in the physical system at hand, which can improve the predictive power of CNNs. The problem with the reflection symmetry chosen

in this thesis is that it leads to a general architecture that is too restrictive. Thus, it cannot compete with the TE architecture. However, arc_{50} still performs similarly compared to ST and FL and is better at predicting n than both of those architectures.

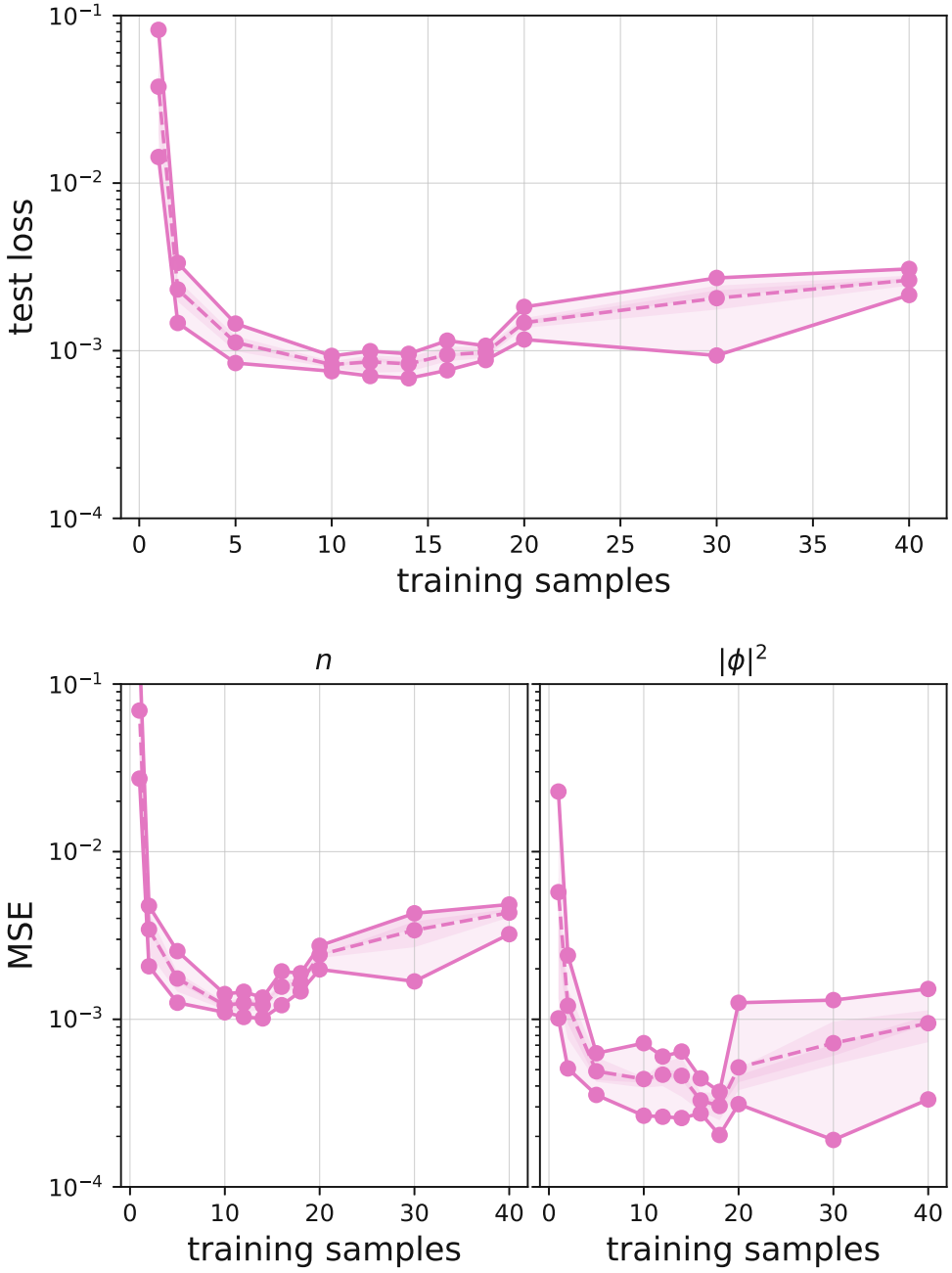

Figure 4.5: Test loss (top) and MSEs for each observable, n and $|\phi|^2$, (bottom) for the arch_{min} of Table 4.4 over training set size on the whole test set of 60×4 lattices. Lines and shaded areas defined as in Figure 4.1.

Another important point is that compared to *TE*, *ST*, and *FL*, the architectures from this thesis have far fewer trainable parameters. Especially arc_{50} has roughly a tenth of the trainable parameters compared to the smallest network of [1]. The network structure includes roughly four times the convolutions, since each feature map is convolved with the same kernel in all four transformations. Still, this effect is more than compensated by the lower amount of parameters. Therefore, if one uses only small training sets and a larger test loss is acceptable, arc_{50} could be a good architecture choice.

Chapter 5

Conclusion

In this thesis we looked at a $1+1$ dimensional complex scalar field on a lattice and used Convolutional Neural Networks (CNNs) to predict two observables of the system. We imposed equivariance under reflections of the lattice on a CNN to see how this would affect the predictive power of the network. A previous paper ([1]) already created a translationally equivariant (TE) network and showed that this performed better than non-TE networks. To compare the newly found network architectures to those of [1], we tested them on the same regression task that was chosen there. For this purpose we built networks that predict two physical observables of the complex scalar field - the particle density n and the mean squared absolute value of the field $|\phi|^2$. As input for the networks we used the four integer valued flux variables, which are found by transforming the discretized scalar field action to the flux representation.

We then looked at how networks perform for different sizes of the training set. We chose four different sizes: $\{50, 200, 2000, 20000\}$. For each of the training set sizes we then looked for the best performing architecture. For this purpose we used the optimization framework *optuna*. This left us with four different networks to compare. Except for the architecture found for the 20000 training set size, all networks performed in a similar way. However, when compared to the best performing TE architecture of [1], the networks found in this thesis performed far worse. Especially for the larger training set size the difference was significant. For the 20000 training samples there were three orders of magnitude between the architectures of this thesis and the TE architecture of the previous paper.

One thing to note is that the newly found architectures here were better at predicting n than at predicting $|\phi|^2$. This can be explained by the network structure, where different flux variables must stay separated throughout the convolutional part of the networks. Thus, networks could not learn the influences different flux variables have on each other locally. Since n only depends on a single flux variable while $|\phi|^2$ depends on all four, this explains why the networks are better at predicting n than $|\phi|^2$.

All networks were trained on the same lattice size. Therefore, we also looked at how the

architectures perform on different lattice sizes they were not trained on. Here we found that for all architectures from this thesis the test loss for other lattice sizes was comparable to the one the networks were trained on. However, while the networks generalized well to other lattices, they also performed worse than the *TE* architecture there.

Lastly, we also looked at how networks would perform with smaller training set sizes. For this purpose we looked at training sets with fewer than 50 samples. We again used the *optuna* framework to find the best performing architecture. Here we found that for most training set sizes the test loss was comparable or only a bit worse than for the larger training set sizes.

In total, the inclusion of the reflection symmetry did not produce networks with better predictive power than the *TE* architecture. However, for other physical systems the reflection symmetry might not impose such drastic restrictions on the general structure of the networks. In our case the problem was that all inputs transformed differently. Including other symmetries, where every input transforms the same, could yield better results.

Appendix A

Discretizing the euclidean action

We start with the euclidean action of eq. (2.13)

$$S_E = \int dy_x dy_t \big(\partial_x \phi^* \partial_x \phi + D_t' \phi^* D_t \phi + m^2 |\phi|^2 + \lambda |\phi|^2\big), \tag{A.1}$$

where we write points on the lattice as y and the indices of the dimensions as x, t instead of $1, 2$. x describes the spatial dimension and t the temporal dimension. Before discretizing we introduce some notation. Continuous space-time turns into a two-dimensional rectangular lattice with finite spacing between lattice points y and periodic boundary conditions enforced. The lattice spacing is set to unity in both dimensions. From this it follows that all dimensionful quantities are understood to be given in appropriate units of the lattice spacing. The field ϕ is now defined only on the discrete points of the lattice. To reflect this, the new notation $\phi(y) \to \phi_y$ is used. The discrete derivative then becomes

$$\partial_i \phi(y) \to \frac{1}{a}(\phi_{y+\hat{e}_x} - \phi_y) = \phi_{y+\hat{e}_x} - \phi_y, \tag{A.2}$$

where $y + \hat{e}_\omega$ represents the point which is reached by moving one unit of the basis vector $\hat{e}_\omega$ into the ω direction. The equality holds since the lattice spacing $a = 1$. The integral simply becomes a sum over all lattice points

$$\int dy_x dy_t \to \sum_y, \tag{A.3}$$

where $y = y_x \hat{e}_x + y_t \hat{e}_t$. Lastly, the covariant derivative D_t becomes

$$\begin{aligned} D_t \phi &= (\partial_t + \mu)\phi = \partial_t \phi - (e^{\mu y_t} \partial_t e^{-\mu y_t})\phi \\ &\to (\phi_{y+\hat{e}_t} - \phi_y) - e^{\mu y_t}(e^{-\mu(y_t+1)} - e^{-\mu y_t})\phi_y = \phi_{y+\hat{e}_t} - e^{-\mu}\phi_y. \end{aligned} \tag{A.4}$$

Similarly for D'_t we get

$$\begin{aligned} D'_t\phi^* &= (\partial_t - \mu)\phi^* = \partial_t\phi^* - (e^{-\mu y_t}\partial_t e^{\mu y_t})\phi^* \\ &\to \phi^*_{y+\hat{e}_t} - e^{\mu}\phi^*_y. \end{aligned} \tag{A.5}$$

Using all definitions, the discretized euclidean action becomes

$$\begin{aligned} S_E = &\sum_y \big((\phi^*_{y+\hat{e}_x} - \phi^*_y)(\phi_{y+\hat{e}_x} - \phi_y) \\ &+ (\phi^*_{y+\hat{e}_t} - e^{\mu}\phi^*_y)(\phi_{y+\hat{e}_t} - e^{-\mu}\phi_y) + m^2|\phi_y|^2 + \lambda|\phi_y|^4\big) \\ = &\sum_y \big(|\phi_{y+\hat{e}_x}|^2 + |\phi_y|^2 - \phi^*_{y+\hat{e}_x}\phi_y - \phi^*_y\phi_{y+\hat{e}_x} \\ &+ |\phi_{y+\hat{e}_t}|^2 + |\phi_y|^2 - e^{-\mu}\phi^*_{y+\hat{e}_t}\phi_y - e^{\mu}\phi^*_y\phi_{y+\hat{e}_t} + m^2|\phi_y|^2 + \lambda|\phi_y|^4\big). \end{aligned} \tag{A.6}$$

Since $\sum_y |\phi_{y+\hat{e}_\omega}|^2 = \sum_y |\phi_y|^2$ for $\omega \in \{x, t\}$ we get

$$S_E = \sum_y \left(\eta|\phi_y|^2 + \lambda|\phi_y|^4 - \sum_\omega \left(e^{-\mu\delta_{\omega,t}}\phi^*_{y+\hat{e}_\omega}\phi_y + e^{\mu\delta_{\omega,t}}\phi^*_y\phi_{y+\hat{e}_\omega}\right)\right), \tag{A.7}$$

or equivalently

$$S_E = \sum_y \left(\eta|\phi_y|^2 + \lambda|\phi_y|^4 - \sum_\omega \left(e^{\mu\delta_{\omega,t}}\phi^*_y\phi_{y+\hat{e}_\omega} + e^{-\mu\delta_{\omega,t}}\phi^*_y\phi_{y-\hat{e}_\omega}\right)\right), \tag{A.8}$$

where $\eta = 4 + m^2$ and in the $e^{-\mu\delta_{\omega,t}}\phi^*_{y+\hat{e}_\omega}\phi_y$ of eq. A.7 term we substituted $y + \hat{e}_\omega \to y$.

Appendix B

The flux representation of the partition function

This calculation closely follows that of [11]. To get the flux representation of the partition function we start with the discretized action of eq. (A.7)

$$S_E = \sum_y \left(\eta |\phi_y|^2 + \lambda |\phi_y|^4 - \sum_\omega \left(e^{-\mu \delta_{\omega,t}} \phi^*_{y+\hat{e}_\omega} \phi_y + e^{\mu \delta_{\omega,t}} \phi^*_y \phi_{y+\hat{e}_\omega} \right) \right). \tag{B.1}$$

The partition function can be written as

$$Z = \int \mathcal{D}[\phi] e^{-S_E}, \tag{B.2}$$

with

$$\int \mathcal{D}[\phi] := \prod_y \int_{\mathbb{C}} \frac{d\phi_y}{2\pi}. \tag{B.3}$$

Using eq. (B.1),

$$e^{-S_E} = \left(\prod_y e^{-\eta |\phi_y|^2} \right) \left(\prod_y e^{-\lambda |\phi_y|^4} \right) \left(\prod_{y,\omega} e^{e^{\mu \delta_{\omega,t}} \phi^*_y \phi_{y+\hat{\omega}}} e^{e^{-\mu \delta_{\omega,t}} \phi_y \phi^*_{y+\hat{\omega}}} \right), \tag{B.4}$$

where we can expand the two exponentials in the last bracket as

$$e^{e^{\mu \delta_{\omega,t}} \phi^*_y \phi_{y+\hat{\omega}}} = \sum_{n_{y,\omega}=0}^{\infty} \frac{1}{n_{y,\omega}!} \left(e^{\mu \delta_{\omega,t}} \phi^*_y \phi_{y+\hat{\omega}} \right)^{n_{y,\omega}} \tag{B.5}$$

and

$$e^{e^{-\mu \delta_{\omega,t}} \phi_y \phi^*_{y+\hat{\omega}}} = \sum_{\bar{n}_{y,\omega}=0}^{\infty} \frac{1}{\bar{n}_{y,\omega}!} \left(e^{-\mu \delta_{\omega,t}} \phi_y \phi^*_{y+\hat{\omega}} \right)^{\bar{n}_{y,\omega}}. \tag{B.6}$$

Using this we get

$$\prod_{y,\omega} e^{e^{\mu\delta_{\omega,t}}\phi_y^*\phi_{y+\hat{\omega}}} e^{e^{-\mu\delta_{\omega,t}}\phi_y\phi_{y+\hat{\omega}}^*} = \prod_{y,\omega}\sum_{n_{y,\omega}}\sum_{\bar{n}_{y,\omega}} \frac{1}{n_{y,\omega}!\bar{n}_{y,\omega}!} e^{\mu\delta_{\omega,t}(n_{y,\omega}-\bar{n}_{y,\omega})} \left(\phi_y^*\phi_{y+\hat{\omega}}\right)^{n_{y,\omega}} \left(\phi_y\phi_{y+\hat{\omega}}^*\right)^{\bar{n}_{y,\omega}}. \tag{B.7}$$

To transform this equation we use that

$$\begin{aligned}\prod_{i=1}^{N}\sum_{a_i}\sum_{b_i} f(a_i)g(b_i) &= \sum_{a_1}\sum_{b_1} f(a_1)g(b_1)\cdots\sum_{a_N}\sum_{b_N} f(a_N)g(b_N)\\ &= \sum_{a_1}\sum_{b_1}\cdots\sum_{a_N}\sum_{b_N}\left(f(a_1)g(b_1)\cdots f(a_N)g(b_N)\right)\\ &=: \sum_{\{a,b\}}\prod_i f(a_i)g(b_i)\end{aligned} \tag{B.8}$$

for arbitrary functions f, g of the summation variables and the fact that

$$\prod_{\omega}\prod_{y}(\phi_{y+\hat{\omega}})^{n_{y,\omega}} = \prod_{\omega}\prod_{y-\hat{\omega}}(\phi_y)^{n_{y-\hat{\omega},\omega}} = \prod_{\omega}\prod_{y}(\phi_y)^{n_{y-\hat{\omega},\omega}}. \tag{B.9}$$

Now using eqs. (B.8) and (B.9) to transform eq. (B.7) gives the following

$$\begin{aligned}\prod_{y,\omega} e^{e^{\mu\delta_{\omega,t}}\phi_y^*\phi_{y+\hat{\omega}}} e^{e^{-\mu\delta_{\omega,t}}\phi_y\phi_{y+\hat{\omega}}^*} &= \sum_{\{n,\bar{n}\}}\left(\prod_x e^{\mu(n_{y,t}-\bar{n}_{y,t})}\right)\left(\prod_{y,\omega}\frac{1}{n_{y,\omega}!\bar{n}_{y,\omega}!}\right)\\ &\quad\times\left(\prod_y \phi_y^{(\bar{n}_{y,x}+\bar{n}_{y,t}+n_{y-\hat{e}_x,x}+n_{y-\hat{e}_t,t})}\right)\\ &\quad\times\left(\prod_y \phi_y^{*(n_{y,x}+n_{y,t}+\bar{n}_{y-\hat{e}_x,x}+\bar{n}_{y-\hat{e}_t,t})}\right)\\ &= \sum_{\{n,\bar{n}\}}\left(\prod_y e^{\mu(n_{y,t}-\bar{n}_{y,t})}\right)\left(\prod_{y,\omega}\frac{1}{n_{y,\omega}!\bar{n}_{y,\omega}!}\right)\\ &\quad\times\left(\prod_y \phi_y^{\sum_\omega(\bar{n}_{y,\omega}+n_{y-\hat{e}_\omega,\omega})}\phi_y^{*\sum_\omega(n_{y,\omega}+\bar{n}_{y-\hat{e}_\omega,\omega})}\right).\end{aligned} \tag{B.10}$$

Inserting everything into the partition function (eq. (B.2)) leads to

$$\begin{aligned}Z = \sum_{\{n,\bar{n}\}}\Bigg[&\left(\prod_y e^{\mu(n_{y,t}-\bar{n}_{y,t})}\right)\left(\prod_{y,\omega}\frac{1}{n_{y,\omega}!\bar{n}_{y,\omega}!}\right)\\ &\times\prod_y\int_{\mathbb{C}}\frac{d\phi_y}{2\pi}e^{-\eta|\phi_y|^2-\lambda|\phi_y|^4}\phi_y^{\sum_\omega(\bar{n}_{y,\omega}+n_{y-\hat{e}_\omega,\omega})}\phi_y^{*\sum_\omega(n_{y,\omega}+\bar{n}_{y-\hat{e}_\omega,\omega})}\Bigg].\end{aligned} \tag{B.11}$$

Substituting $\int_{\mathbb{C}} d\phi_y \to \int_0^\infty \int_0^{2\pi} r_y dr_y d\varphi_y$ with $\phi_y = r_y e^{i\varphi_y}$ gives

$$\begin{aligned} Z = \sum_{\{n,\bar{n}\}} \Bigg[&\Big(\prod_y e^{\mu(n_{y,t}-\bar{n}_{y,t})}\Big)\Big(\prod_{y,\omega} \frac{1}{n_{y,\omega}!\bar{n}_{y,\omega}!}\Big) \\ &\times \prod_y \int_0^\infty dr_y r_y \, e^{-\eta r_y^2 - \lambda r_y^4} \, r_y^{\sum_\omega (\bar{n}_{y,\omega}+\bar{n}_{y-\hat{e}_\omega,\omega}+n_{y,\omega}+n_{y-\hat{e}_\omega,\omega})} \\ &\times \prod_y \int_0^{2\pi} \frac{d\varphi_y}{2\pi} e^{-i\varphi_y \sum_\omega (n_{y,\omega}+\bar{n}_{y-\hat{e}_\omega,\omega}-\bar{n}_{y,\omega}-n_{y-\hat{e}_\omega,\omega})} \Bigg]. \end{aligned} \tag{B.12}$$

Since r_y and φ_y are just integration variables, we can drop the index. The y dependence that matters occurs in the exponents only. To get the final form of this equation we define

$$k_{y,\omega} := n_{y,\omega} - \bar{n}_{y,\omega}, \tag{B.13}$$

$$2l_{y,\omega} + |k_{y,\omega}| := n_{y,\omega} + \bar{n}_{y,\omega}, \tag{B.14}$$

$$f_y := \sum_\omega (\bar{n}_{y,\omega} + \bar{n}_{y-\hat{e}_\omega,\omega} + n_{y,\omega} + n_{y-\hat{e}_\omega,\omega}) = \sum_\omega (2(l_{y,\omega} + l_{y-\hat{e}_\omega,\omega}) + |k_{y,\omega}| + |k_{y-\hat{e}_\omega,\omega}|), \tag{B.15}$$

$$W(f_y) := \int_0^\infty dr (r \, e^{-\eta r^2 - \lambda r^4} \, r^{f_y}) = \int_0^\infty dr \, e^{-\eta r^2 - \lambda r^4} \, r^{f_y+1}. \tag{B.16}$$

Lastly, the sum in the exponent in the last line of eq. (B.12) is always an integer. We consider the following two cases. First, if

$$\sum_\omega (n_{y,\omega} + \bar{n}_{y-\hat{\omega},\omega} - \bar{n}_{y,\omega} - n_{y-\hat{e}_\omega,\omega}) = 0, \tag{B.17}$$

the integral becomes

$$\int_0^{2\pi} \frac{d\varphi}{2\pi} = 1. \tag{B.18}$$

Else, if

$$\sum_\omega (n_{y,\omega} + \bar{n}_{y-\hat{\omega},\omega} - \bar{n}_{y,\omega} - n_{y-\hat{e}_\omega,\omega}) = n \neq 0, \tag{B.19}$$

the integral results in

$$\int_0^{2\pi} \frac{d\varphi}{2\pi} e^{in\varphi} = 0. \tag{B.20}$$

Therefore, we can put the two cases together into a Kronecker delta and get

$$\begin{aligned} \int_0^{2\pi} \frac{d\varphi}{2\pi} e^{-i\varphi \sum_\omega (n_{y,\omega}+\bar{n}_{y-\hat{\omega},\omega}-\bar{n}_{y,\omega}-n_{y-\hat{e}_\omega,\omega})} &= \delta\Big(\sum_\omega (n_{y,\omega} + \bar{n}_{y-\hat{e}_\omega,\omega} - \bar{n}_{y,\omega} - n_{y-\hat{e}_\omega,\omega})\Big) \\ &= \delta\Big(\sum_\omega k_{y,\omega} - k_{y-\hat{e}_\omega,\omega}\Big). \end{aligned} \tag{B.21}$$

Altogether, in the flux representation the partition function takes the form

$$Z = \sum_{\{k,l\}} \Big[\Big(\prod_{y,\omega} \frac{1}{(|k_{y,\omega}| + l_{y,\omega})! l_{y,\omega}!} \Big) \Big(\prod_{y} e^{\mu k_{y,t}} W(f_y) \Big) \Big(\prod_{y} \delta \Big(\sum_{\omega} k_{y,\omega} - k_{y-\hat{e}_\omega,\omega} \Big) \Big) \Big], \quad \text{(B.22)}$$

with the flux variables $k_{y,\omega} \in \mathbb{Z}$ and $l_{y,\omega} \in \mathbb{N}_0$.

Appendix C

Reflections of the lattice in the flux representation

Taking $\{\hat{e}_x, \hat{e}_t\}$ as the basis of the two-dimensional lattice of the problem we can write any point as $y = a\hat{e}_x + b\hat{e}_t$. Thus, a reflection in the x direction can be written as $y = a\hat{e}_x + b\hat{e}_t \rightarrow y' = -a\hat{e}_x + b\hat{e}_t$. Similarly, a reflection in the t direction can be written as $y = a\hat{e}_x + b\hat{e}_t \rightarrow y' = a\hat{e}_x - b\hat{e}_t$.

To see how such a reflection transforms the flux variables $k_{y,\omega}, l_{y,\omega}$, we start with the flux representation of the partition function as described in Appendix B

$$Z = \sum_{\{k,l\}} \Big[\Big(\prod_{y,\omega} \frac{1}{(|k_{y,\omega}| + l_{y,\omega})! l_{y,\omega}!} \Big) \Big(\prod_{y} e^{\mu k_{y,t}} W(f_y) \Big) \Big(\prod_{y} \delta \Big(\sum_{\omega} k_{y,\omega} - k_{y-\hat{e}_\omega,\omega} \Big) \Big) \Big]. \tag{C.1}$$

As was already shown in chapter 2 the action is invariant under reflections and as such, also the partition function is invariant. We will start by showing how a reflection affects the delta-function in eq. (C.1) and afterwards demonstrate that under the transformation one obtains, the partition function as a whole is invariant. To ensure that the partition function stays invariant, any term of the form

$$\delta \Big(\sum_{\omega} k_{y,\omega} - k_{y-\hat{e}_\omega,\omega} \Big), \tag{C.2}$$

for some y should be mapped to a term of the same form of another point (possibly the same) on the lattice. Additionally this mapping must be bijective to ensure

$$\prod_{y} \delta \Big(\sum_{\omega} k_{y,\omega} - k_{y-\hat{e}_\omega,\omega} \Big) \tag{C.3}$$

stays invariant.

First we start with a reflection in the x direction. For some a, b the term in the delta function

reads

$$k_{a\hat{e}_x+b\hat{e}_t,x} - k_{(a-1)\hat{e}_x+b\hat{e}_t,x} + k_{a\hat{e}_x+b\hat{e}_t,t} - k_{a\hat{e}_x+(b-1)\hat{e}_t,t}. \tag{C.4}$$

Reflecting this term in the x direction leads to

$$-k_{-a\hat{e}_x+b\hat{e}_t,x} + k_{-(a-1)\hat{e}_x+b\hat{e}_t,x} + k_{-a\hat{e}_x+b\hat{e}_t,t} - k_{-a\hat{e}_x+(b-1)\hat{e}_t,t}. \tag{C.5}$$

The term at $-a\hat{e}_x + b\hat{e}_x$ before the reflection is given by

$$k_{-a\hat{e}_x+b\hat{e}_t,x} - k_{-(a+1)\hat{e}_x+b\hat{e}_t,x} + k_{-a\hat{e}_x+b\hat{e}_t,t} - k_{-a\hat{e}_x+(b-1)\hat{e}_t,t}. \tag{C.6}$$

Comparing the two equations we see that the last two terms in eq. (C.5) and (C.6) are equal. Thus, the proposed transformation for $k_{y,t}$ is

$$k_{a\hat{e}_x+b\hat{e}_t,t} \to k_{-a\hat{e}_x+b\hat{e}_t,t}. \tag{C.7}$$

For $k_{y,x}$ we notice that a shift of -1 in the x direction together with an additional minus sign in eq. (C.5) gives the same $k_{y,x}$ terms as in eq. (C.6). Therefore, the proposed transformation for $k_{y,x}$ reads

$$k_{a\hat{e}_x+b\hat{e}_t,x} \to -k_{-(a+1)\hat{e}_x+b\hat{e}_t,x}. \tag{C.8}$$

Similarly, one can show that for a reflection in the t direction the transformations read

$$k_{a\hat{e}_x+b\hat{e}_t,x} \to k_{a\hat{e}_x-b\hat{e}_t,x} \tag{C.9}$$

and

$$k_{a\hat{e}_x+b\hat{e}_t,t} \to -k_{a\hat{e}_x-(b+1)\hat{e}_t,t}. \tag{C.10}$$

Now we look at the term $\prod_y e^{\mu k_{y,t}}$ from eq. (C.1). Since, for a reflection in the t direction $\mu \to -\mu$, as shown in chapter 2,

$$\prod_{a,b} e^{\mu k_{a\hat{e}_x+b\hat{e}_t,t}} \to \prod_{a,b} e^{-\mu(-k_{a\hat{e}_x-(b+1)\hat{e}_t,t})} = \prod_{a,(b-1)} e^{\mu k_{a\hat{e}_x-b\hat{e}_t,t}} = \prod_{a,b} e^{\mu k_{a\hat{e}_x-b\hat{e}_t,t}} \tag{C.11}$$

and therefore, this term is left invariant. Similarly, for a reflection in the x direction μ does not change its sign. And since $k_{y,t}$ also does not change sign the term is invariant as well.

To see how $l_{y,\omega}$ needs to transform we look at $\prod_y W(f_y)$. Again, we start with a reflection in the x direction. Before the transformation $f_{a\hat{e}_x+b\hat{e}_t}$ reads

$$\begin{aligned} f_{a\hat{e}_x+b\hat{e}_t} = {} & |k_{a\hat{e}_x+b\hat{e}_t,x}| + |k_{(a-1)\hat{e}_x+b\hat{e}_t,x}| + |k_{a\hat{e}_x+b\hat{e}_t,t}| + |k_{a\hat{e}_x+(b-1)\hat{e}_t,t}| + \\ & + 2(l_{a\hat{e}_x+b\hat{e}_t,x} + l_{(a-1)\hat{e}_x+b\hat{e}_t,x} + l_{a\hat{e}_x+b\hat{e}_t,t} + l_{a\hat{e}_x+(b-1)\hat{e}_t,t}). \end{aligned} \tag{C.12}$$

Applying the reflection in the x direction gives

$$\begin{aligned} f^{ref_x}_{a\hat{e}_x+b\hat{e}_t} = &| - k_{-(a+1)\hat{e}_x+b\hat{e}_t,x}| + | - k_{-a\hat{e}_x+b\hat{e}_t,x}| + |k_{-a\hat{e}_x+b\hat{e}_t,t}| + |k_{-a\hat{e}_x+(b-1)\hat{e}_t,t}|+ \\ &+ 2(l_{-a\hat{e}_x+b\hat{e}_t,x} + l_{-(a-1)\hat{e}_x+b\hat{e}_t,x} + l_{-a\hat{e}_x+b\hat{e}_t,t} + l_{-a\hat{e}_x+(b-1)\hat{e}_t,t}). \end{aligned} \tag{C.13}$$

Comparing this to $f_{-a\hat{e}_x+b\hat{e}_t}$ before the reflection

$$\begin{aligned} f_{-a\hat{e}_x+b\hat{e}_t} = &| - k_{-a\hat{e}_x+b\hat{e}_t,x}| + | - k_{-(a+1)\hat{e}_x+b\hat{e}_t,x}| + |k_{-a\hat{e}_x+b\hat{e}_t,t}| + |k_{-a\hat{e}_x+(b-1)\hat{e}_t,t}|+ \\ &+ 2(l_{-a\hat{e}_x+b\hat{e}_t,x} + l_{-(a+1)\hat{e}_x+b\hat{e}_t,x} + l_{-a\hat{e}_x+b\hat{e}_t,t} + l_{-a\hat{e}_x+(b-1)\hat{e}_t,t}), \end{aligned} \tag{C.14}$$

all the $k_{y,\omega}$ terms can be matched, as can the last two terms of the $l_{y,t}$. For the other two $l_{y,x}$ terms, a shift of -1 in the x direction is needed to match them up. Thus, the transformations for $l_{y,\omega}$ read

$$l_{a\hat{e}_x+b\hat{e}_t,x} \to l_{-(a+1)\hat{e}_x+b\hat{e}_t,x} \tag{C.15}$$

and

$$l_{a\hat{e}_x+b\hat{e}_t,t} \to l_{-a\hat{e}_x+b\hat{e}_t,t}. \tag{C.16}$$

Similarly, for the reflection in t direction we get

$$l_{a\hat{e}_x+b\hat{e}_t,x} \to l_{a\hat{e}_x-b\hat{e}_t,x} \tag{C.17}$$

and

$$l_{a\hat{e}_x+b\hat{e}_t,t} \to l_{a\hat{e}_x-(b+1)\hat{e}_t,t}. \tag{C.18}$$

The only term left to check is

$$\prod_{y,\omega} \frac{1}{(|k_{y,\omega}| + l_{y,\omega})! l_{y,\omega}!}. \tag{C.19}$$

However, since the absolute value cancels the minus sign for the $k_{y,\omega}$ and the product is still over every point on the lattice, the invariance of this term is easy to see.

To summarize, the complete transformation behavior for the flux variables is

$$\begin{aligned} k_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{x\to -x} -k_{-(a+1)\hat{e}_x+b\hat{e}_t,x}, \\ k_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{x\to -x} k_{-a\hat{e}_x+b\hat{e}_t,t}, \\ l_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{x\to -x} l_{-(a+1)\hat{e}_x+b\hat{e}_t,x}, \\ l_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{x\to -x} l_{-a\hat{e}_x+b\hat{e}_t,t}, \end{aligned} \tag{C.20}$$

for a reflection in the x direction and

$$
\begin{aligned}
k_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{t\to -t} k_{a\hat{e}_x-b\hat{e}_t,x}, \\
k_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{t\to -t} -k_{a\hat{e}_x-(b+1)\hat{e}_t,t}, \\
l_{a\hat{e}_x+b\hat{e}_t,x} &\xrightarrow{t\to -t} l_{a\hat{e}_x-b\hat{e}_t,x}, \\
l_{a\hat{e}_x+b\hat{e}_t,t} &\xrightarrow{t\to -t} l_{a\hat{e}_x-(b+1)\hat{e}_t,t},
\end{aligned}
\tag{C.21}
$$

for the t direction.

Appendix D

Kernel transformation example

We will show a simple example of a convolutional layer with a single feature map f as input, a single kernel ψ, and an output stack $G = (g_i)_{i=1}^4$. We will see that a transformation of the kernels according to eqs. (3.27)-(3.30) will create the expected behavior. Let

$$f = \begin{bmatrix} 4 & 2 & 3 \\ 5 & 7 & 2 \\ 1 & 6 & 8 \end{bmatrix}, \tag{D.1}$$

$$\psi = \begin{bmatrix} 5 & 1 & 8 \\ 4 & 8 & 3 \\ 2 & 5 & 7 \end{bmatrix}. \tag{D.2}$$

Applying the transformations of eqs. (3.27)-(3.30) to ψ results in the stack of kernels

$$\psi^0 = \begin{bmatrix} 5 & 1 & 8 \\ 4 & 8 & 3 \\ 2 & 5 & 7 \end{bmatrix}, \psi^1 = \begin{bmatrix} 2 & 5 & 7 \\ 4 & 8 & 3 \\ 5 & 1 & 8 \end{bmatrix}, \psi^2 = \begin{bmatrix} 8 & 1 & 5 \\ 3 & 8 & 4 \\ 7 & 5 & 2 \end{bmatrix}, \psi^3 = \begin{bmatrix} 7 & 5 & 2 \\ 3 & 8 & 4 \\ 8 & 1 & 5 \end{bmatrix}. \tag{D.3}$$

Using circular padding on f creates the larger feature map

$$f' = \begin{bmatrix} 8 & 1 & 6 & 8 & 1 \\ 3 & 4 & 2 & 3 & 4 \\ 2 & 5 & 7 & 2 & 5 \\ 8 & 1 & 6 & 8 & 1 \\ 3 & 4 & 2 & 3 & 4 \end{bmatrix}. \tag{D.4}$$

Then, applying the convolutions to f' the output stack takes the form

$$\begin{aligned} g_0 &= \begin{bmatrix} 217 & 175 & 149 \\ 167 & 216 & 163 \\ 169 & 163 & 215 \end{bmatrix}, g_1 = \begin{bmatrix} 184 & 177 & 180 \\ 198 & 196 & 152 \\ 171 & 181 & 195 \end{bmatrix}, \\ g_2 &= \begin{bmatrix} 197 & 168 & 176 \\ 185 & 181 & 180 \\ 157 & 184 & 206 \end{bmatrix}, g_3 = \begin{bmatrix} 178 & 150 & 213 \\ 214 & 177 & 155 \\ 147 & 206 & 194 \end{bmatrix}. \end{aligned} \tag{D.5}$$

Now we apply the transformation $R_{k_x,x}$ to f, add circular padding, apply the same convolutions as above, and compare the results with eq. (D.5). $R_{k_x,x}$ flips the feature map vertically, moves each horizontal line up by one, and then multiplies the resulting feature map by -1. The result is

$$f^* := L_{R_{k_x,x}} f = \begin{bmatrix} -5 & -7 & -2 \\ -4 & -2 & -3 \\ -1 & -6 & -8 \end{bmatrix}. \tag{D.6}$$

First adding circular padding and then applying the convolutions results in the output

$$\begin{aligned} g_0^* &= \begin{bmatrix} -198 & -196 & -152 \\ -184 & -177 & -180 \\ -171 & -181 & -195 \end{bmatrix}, g_1^* = \begin{bmatrix} -167 & -216 & -163 \\ -217 & -175 & -149 \\ -169 & -163 & -215 \end{bmatrix}, \\ g_2^* &= \begin{bmatrix} -214 & -177 & -155 \\ -178 & -150 & -213 \\ -147 & -206 & -194 \end{bmatrix}, g_3^* = \begin{bmatrix} -185 & -181 & -180 \\ -197 & -168 & -176 \\ -157 & -184 & -206 \end{bmatrix}. \end{aligned} \tag{D.7}$$

Now, if we compare eq. (D.5) with eq. (D.7) we can see that

$$g_0^* = L_{R_{k_x,x}} g_1, \tag{D.8}$$
$$g_1^* = L_{R_{k_x,x}} g_0, \tag{D.9}$$
$$g_2^* = L_{R_{k_x,x}} g_3, \tag{D.10}$$
$$g_3^* = L_{R_{k_x,x}} g_2, \tag{D.11}$$

which is the desired result.

Since shifting the kernel makes no sense, the other option would have been to also include the negative sign of the transformations. As we will see, this results in

$$g_0^* = -L_{R_{k_x,x}} g_1, \tag{D.12}$$
$$g_1^* = -L_{R_{k_x,x}} g_0, \tag{D.13}$$
$$g_2^* = -L_{R_{k_x,x}} g_3, \tag{D.14}$$
$$g_3^* = -L_{R_{k_x,x}} g_2, \tag{D.15}$$

where we have extra negative signs. These transformations are not included in the group $\mathcal{R}_{k_x}$ and thus do not result in equivariant behavior.

If we include the negative sign in the transformation of the kernels, we would get a different stack of kernels

$$\psi^0 = \begin{bmatrix} 5 & 1 & 8 \\ 4 & 8 & 3 \\ 2 & 5 & 7 \end{bmatrix}, \psi^1 = \begin{bmatrix} -2 & -5 & -7 \\ -4 & -8 & -3 \\ -5 & -1 & -8 \end{bmatrix}, \psi^2 = \begin{bmatrix} 8 & 1 & 5 \\ 3 & 8 & 4 \\ 7 & 5 & 2 \end{bmatrix}, \psi^3 = \begin{bmatrix} -7 & -5 & -2 \\ -3 & -8 & -4 \\ -8 & -1 & -5 \end{bmatrix}. \tag{D.16}$$

Using these kernels for the convolution then gives the output

$$\begin{aligned} g^0 &= \begin{bmatrix} 217 & 175 & 149 \\ 167 & 216 & 163 \\ 169 & 163 & 215 \end{bmatrix}, g^1 = \begin{bmatrix} -184 & -177 & -180 \\ -198 & -196 & -152 \\ -171 & -181 & -195 \end{bmatrix}, \\ g^2 &= \begin{bmatrix} 197 & 168 & 176 \\ 185 & 181 & 180 \\ 157 & 184 & 206 \end{bmatrix}, g^3 = \begin{bmatrix} -178 & -150 & -213 \\ -214 & -177 & -155 \\ -147 & -206 & -194 \end{bmatrix}, \end{aligned} \tag{D.17}$$

for the untransformed input and

$$\begin{aligned} g_0^* &= \begin{bmatrix} -198 & -196 & -152 \\ -184 & -177 & -180 \\ -171 & -181 & -195 \end{bmatrix}, g_0^* = \begin{bmatrix} 167 & 216 & 163 \\ 217 & 175 & 149 \\ 169 & 163 & 215 \end{bmatrix}, \\ g_2^* &= \begin{bmatrix} -214 & -177 & -155 \\ -178 & -150 & -213 \\ -147 & -206 & -194 \end{bmatrix}, g_3^* = \begin{bmatrix} 185 & 181 & 180 \\ 197 & 168 & 176 \\ 157 & 184 & 206 \end{bmatrix}. \end{aligned} \tag{D.18}$$

If we compare these two outputs we get

$$g_0^* = -L_{R_{k_x},x} g_1, \tag{D.19}$$
$$g_1^* = -L_{R_{k_x},x} g_0, \tag{D.20}$$
$$g_2^* = -L_{R_{k_x},x} g_3, \tag{D.21}$$
$$g_3^* = -L_{R_{k_x},x} g_2, \tag{D.22}$$

which, as discussed above, does not result in an equivariant transformation. Thus, (D.8)-(D.11) show the only equivariant transformation behavior and therefore, we transform the kernels according to eqs. (3.27)-(3.30).

Bibliography

[1] S. Bulusu, M. Favoni, A. Ipp, D. I. Müller, and D. Schuh. Generalization capabilities of translationally equivariant neural networks. *Physical Review D* 104.7 (2021), p. 074504. DOI: 10.1103/PhysRevD.104.074504.

[2] T. Cohen and M. Welling. Group equivariant convolutional networks. *International conference on machine learning*. PMLR. 2016, pp. 2990–2999. DOI: 10.48550/arXiv.2009.08895.

[3] E. Noether. Invariante Variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse* 1918 (1918), pp. 235–257. URL: http://eudml.org/doc/59024.

[4] T. Brauner. Remarks on relativistic scalar models with chemical potential. *arXiv* (2020). DOI: 10.48550/arXiv.2009.08895.

[5] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv* (2018). DOI: 10.48550/arXiv.1811.03378.

[6] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv* (2013). DOI: 10.48550/arXiv.1312.4400.

[7] K. Zhou, G. Endrődi, L.-G. Pang, and H. Stöcker. Regressive and generative neural networks for scalar field theory. *Physical Review D* 100.1 (2019), p. 011501. DOI: 10.1103/PhysRevD.100.011501.

[8] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631. DOI: 10.1145/3292500.3330701.

[9] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. *arXiv* (2019). DOI: 10.48550/arXiv.1904.09237.

[10] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv* (2017). DOI: 10.48550/arXiv.1711.05101.

[11] C. Gattringer and T. Kloiber. Lattice study of the Silver Blaze phenomenon for a charged scalar ϕ^4 field. *Nuclear Physics B* 869.1 (2013), pp. 56–73. DOI: 10.1016/j.nuclphysb.2012.12.005.