LieGG: Studying Learned Lie Group Generators

Anonymous Author(s)
Affiliation
Address
email

Abstract

Symmetries built into the network have appeared to be very beneficial for a wide range of tasks as it saves data to learn them. We depart from the position that when the symmetries are not built into a model a priori, it is advantageous for robust networks to learn the symmetries directly from the data to fit the task function. In this paper, we present a method to extract symmetries learned by the neural network and to evaluate the degree to which the network is invariant to them. With our method, we are able to explicitly retrieve learned invariances in a form of the generators of corresponding Lie-groups without prior knowledge of the symmetries in the data. We use the proposed method to study how symmetrical properties depend on the network's parameterization and configuration. We found that the ability of the network to learn symmetries generalizes over a wide range of architectures. However, the quality of the learned symmetries depends on the depth and number of parameters.

1 Introduction

Convolutional Neural Networks (CNNs) are efficient, for one because they can convert translation symmetry in the data into a built-in translation-equivariance property of the network without exhausting data to learn the equivariance. Group-equivariant networks generalize this property to rotation [1, 2, 3, 4], scale [5, 6, 7], and other symmetries defined by matrix groups [8]. Equipping a neural network with prior known symmetries has proved to be data-efficient.

Recent works [9, 10, 11] have demonstrated that hard coding the symmetry into the model does not always lead to a better generalization. Often, soft or learned equivariance is more advantageous mostly in terms of data efficiency and accuracy [12]. Moreover, architectures with no equivariance built-in, such as transformers [13] and multi-layer perceptron mixers [14], achieve a remarkable performance on a wide range of problems. Effectively, they learn their own geometrical priors without explicit symmetry constraints [15]. This raises the following questions to study in this paper. To what degree do neural networks learn symmetries? How accurately do learned symmetries reflect the true symmetries in the data? Can we support the capability of networks for learning symmetries? We present a method to study symmetries learned by networks to address the questions.

In the paper we depart from Lie group theory and develop a method that can retrieve symmetries learned from the data by any neural network. Our method makes the assumption that the model is differentiable, commonly met in neural networks. While previous works on analyzing symmetries in networks rely on empirical analysis of network representations [16] or on examination of a given set of transformations [17, 6, 18], our method outputs a generator of the corresponding Lie-group to quantitatively evaluate how sensitive the network is in the direction of the learned symmetry (Figure 1). We make the following contributions:

- From the perspective of Lie-groups, we propose a theory to study symmetrical properties of a trained neural network.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

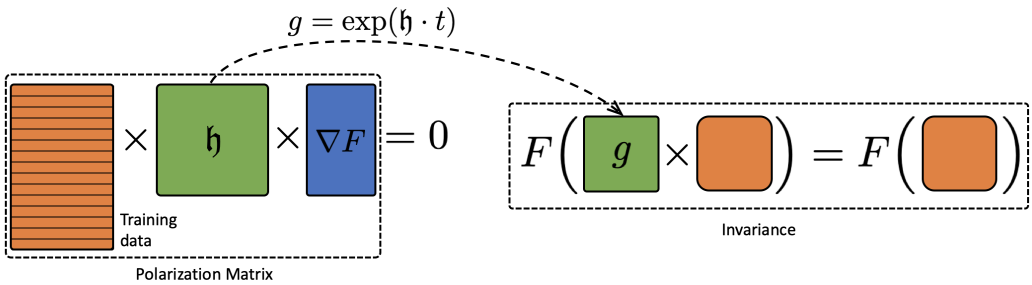

Figure 1: We solve a matrix equation to derive an infinitesimal generator from the model and its training data. Due to Lie algebra-Lie group correspondence, we can calculate the class of transformations the model is invariant to by exponentiating the generator.

- We derive an efficient implementation that allows improving the interpretability of the model by revealing symmetries it learned.
- With our method we conclude that models with more parameters and gradual fine-tuning learn more precise symmetry groups with a higher degree of invariances to them.

2 Related work

Equivariant and invariant networks The goal of equivariant and invariant networks is to build the symmetries into a network architecture as an inductive bias. Starting from the convolutional networks [19] that contain translation symmetry, the concept of equivariance was generalized to rotations [1, 2, 3, 4], permutations [20], scaling [5, 6, 21, 7, 22] and arbitrary matrix groups in [8]. Various methods have been proposed for learning symmetries directly from the training data [11, 23] among which a popular approach is to learn transformations by estimating the infinitesimal generators [24, 25, 26, 27, 28]. These papers focus on answering the question of how to build symmetries into the models by modifying the architecture. In our work, we focus on the reverse question, i.e. given a network with a fixed architecture, what symmetries does it learn from the data.

Symmetries in neural networks Another line of work is focused on interpreting symmetries in neural networks. In [16] Olah *et al.* take inspiration from biological circuit motifs [29] and study the equivariant patterns learned by a neural network on the image classification task. The authors demonstrate that a network learns rotation, hue, and scale symmetries when trained on ImageNet. In [17] Goodfellow *et al.* propose a number of empirical tests to study invariances to known transformations in the network, and demonstrate that auto-encoding architectures learn increasingly invariant features in deep layers of the network. Along the same lines, Lenc & Vedaldi [18] quantify invariances and equivariance in the layers of convolutional networks to a pre-defined set of transformations. In this paper, we also propose a method to study the symmetrical properties of neural networks. In contrast to existing works, our method does not require knowing the set of transformations beforehand, and it also provides theoretical guarantees of invariances from the perspective of the Lie groups theory.

Network configuration: width, depth, number of parameters Neural networks of various widths and depths have been studied through the lens of universal approximation theorem [30, 31], functional expressiveness [32] and by empirical analysis of learned representations [33]. These works focus on either characterizing the learning ability of the model or on interpreting the discrepancy between representations that models with different architectures learn. They, however, do not analyze how the symmetrical properties of a model depend on its width and depth.

Other works focus on analyzing the connection between a number of parameters and the generalization capability of networks [34, 35, 36, 37]. It is commonly argued that overparametrized architectures achieve better generalization bounds, i.e. provide smaller discrepancy between train and test performance and are thus preferred in many applications [37]. However, it is unclear if models with more parameters and hence with a better generalization capability also learn better symmetries. In our paper, we investigate this for a family of feed-forward networks.

Robust Learning Various methods have been proposed to train robust models on small datasets. While the naive training may lead to overfitting on the training subset [38], well-organized pretraining helps to mitigate this issue. Features extracted by a model pretrained on a bigger dataset often demonstrate more robust results [39]. While there are many useful techniques for training effective feature extractors [40, 41, 42], the analysis of such methods from the invariance point of view was not done before. We demonstrate that our method allows for better understanding and explaining what training regimes lead to more invariant, and thus more robust features.

3 Background

The focus of this paper is a *symmetry group*. Intuitively, a symmetry is a transformation of an object that leaves the object unchanged. The set of all such transformations of an object with composition as a binary operation forms a group. In this paper, the object of interest is a dataset, and we study what kind of transformations map one data sample to another and how neural networks preserve information about symmetries.

The theory of Lie groups is a sweet spot of mathematics that helps to formalize symmetries and provides practical methods for studying them. Formally, a Lie group is a group that has the structure of a differential manifold. The tangent space in the identity element of a Lie group forms a vector space called Lie algebra. A Lie algebra determines a Lie group up to an isomorphism for a simply connected Lie groups and, being a vector space, is a more convenient object to study than a Lie group. Thus, in order to recover a simply connected symmetry group, it is sufficient to understand its Lie algebra.

In this paper, we assume that a Lie group G is a matrix Lie group, i.e., a closed subgroup of the general linear group degree n, denoted as $GL(n)$ and is defined as the set of $n \times n$ invertible matrices. The correspondence between the Lie group G and its Lie algebra $\mathfrak{g}$ in this case is given by the exponential map:

$$\mathfrak{g} = \{\mathfrak{h} \in M(n) : \ e^{t \cdot \mathfrak{h}} \in G \ \forall t \in \mathbb{R}\}, \tag{1}$$

where $M(n)$ is the set of all $n \times n$ matrices. Each such element $\mathfrak{h}$ from Equation 1 is called an *infinitesimal generator* of the Lie group G.

We introduce **LieGG**, a method which extracts **Lie G**roup **G**enerators, i.e. finds the infinitesimal generators of the Lie algebra. To find the Lie algebra basis, we use the *discriminator* of the dataset, i.e. a function $f : \mathbb{R}^n \to \mathbb{R}$ such that $f(\mathbf{x}) = 0$ if and only if $\mathbf{x}$ is an element of the dataset $\mathcal{D}$ data. Having a discriminator, we use the following criterion for a symmetry group of algebraic equations:

Theorem 3.1 (Theorem 2.8 [43])**.** *Let G be a connected Lie group of linear transformations acting on an n-th dimensional manifold $\mathcal{X}$. Let $F : \mathcal{X} \to \mathbb{R}^l, \ l \leq n$, define a system of algebraic equations:*

$$F_\nu(\mathbf{x}) = 0, \ \nu = 1, \cdots, l,$$

and assume that the system is of maximal rank, meaning that the Jacobian matrix $\left(\frac{\partial F_\nu}{\partial x_k}\right)$ is of rank l for every solution $\mathbf{x}$ of the system. Then G is a symmetry group of the system if and only if

$$\sum_{i=1,j=1}^{i=n,j=l} \frac{\partial F_\nu}{\partial x_i} \cdot \mathfrak{h}_{ij} \cdot x_j = 0, \ \textit{whenever } F_\nu(\mathbf{x}) = 0,$$

for every infinitesimal generator $\mathfrak{h}$ of G, where $\mathfrak{h}_{ij}$ is an element of the matrix $\mathfrak{h}$ in the i-th row and j-th column.

Example Let us illustrate the practical significance of the theorem with an example. Suppose data D lie on a sphere in $\mathbb{R}^n$:

$$D = \{\mathbf{x} \in \mathbb{R}^n : \ x_1^2 + \cdots + x_n^2 = 1\}.$$

The discriminator for this example is the function $f = x_1^2 + \cdots + x_n^2 - 1$. According to the theorem, in order to find Lie algebra of the symmetry group, we can find a basis of the solution of the following linear equations, where a_{ij} are sought variables:

$$\sum_{i,j} x_i \cdot a_{ij} \cdot x_j = 0,$$

when $x_1^2 + \cdots + x_n^2 = 1$. We assert, that the solutions are matrices A such that $A + A^T = 0$. Indeed, for $\mathbf{x}: \ x_i = 1, \ x_j = 0, \ j \neq i$, the condition means that $a_{ii} = 0$. For $\mathbf{x}: \ x_i = \frac{1}{\sqrt{2}}, \ x_j = \frac{1}{\sqrt{2}}, \ x_k = 0, \ k \neq i, j$, it follows, that $a_{ij} + a_{ji} = 0$. On the other hand if $A = -A^T$, then $\sum_{i \neq j} \left(x_i \cdot a_{ij} \cdot x_j + x_j \cdot a_{ji} \cdot x_i \right) = \sum_{i \neq j} \left(x_i \cdot a_{ij} \cdot x_j - x_i \cdot a_{ij} \cdot x_j \right) = 0$.

Note that the Lie algebra of the matrices

$$\{A : A + A^T = 0\}$$

corresponds to the Lie group of matrices

$$\{B : B \cdot B^T = E\}$$

which is a rotation group. Thus the symmetry of a sphere is a rotation group.

4 Method

To compute a Lie algebra given a discriminator function $f : \mathbb{R}^n \to \mathbb{R}$ we use Theorem 3.1 and solve the system of linear equations, where each equation corresponds to one element in the dataset:

$$\sum_{i=1,j=1}^{i=n,j=n} \frac{\partial f}{\partial x_i} \cdot \mathfrak{h}_{i,j} \cdot x_j = 0, \text{ for each element from the dataset.} \tag{2}$$

This is a system of linear equations with n^2 variables, and number of equations equal to the number of points in the data. We denote the matrix defined by equations 2 as $\mathcal{E}$ and call it the *network polarization matrix*. The problem of finding a basis of solutions of linear equations is a standard problem in numerical analysis, that can be solved using singular value decomposition (SVD). We remind that we can write $\mathcal{E}$ using SVD in the following way:

$$\mathcal{E} = U \Sigma V^*,$$

where U, V - are orthogonal matrixes and Σ is a diagonal matrix with decreasing elements on the diagonal. Thus, the columns in V corresponding to almost 0 singular values form a Lie algebra basis.

Thereby, in practice, the calculation of LieGG consists of 3 steps: 1.training a neural network; 2.calculation of the polarization matrix $\mathcal{E}$; and 3.calculation of singular vectors corresponding to almost zero singular values of the matrix $\mathcal{E}$.

4.1 (revised) Computation of Lie algebra of the group acting on $\mathbb{R}^2$.

Symmetries play an important role in computer vision problems, and we will describe LieGG for this case in more detail. In various computer vision problems, it is assumed that the symmetry group G changes images from the dataset by acting on $\mathbb{R}^2$. For example, convolutional networks [19] and recently proposed equivariant networks for rotations [3] assume that the group acts as a translation or rotation of the image. These neural networks perceive each image as a function f with non-zero values on some compact subset $\Omega \subset \mathbb{R}^2$, where the elements of the subset correspond to spatial coordinates. Thus, the space of images $\mathcal{X}$ is the space of all such functions:

$$\mathcal{X} \coloneqq \{f, \ f : \Omega \to \mathbb{R}\}.$$

A vision model $F : \mathcal{X} \to \mathbb{R}$ is a function on such a space of functions. We view an image is a function on a pixel grid $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$ of the domain Ω, i.e. the image can be identified with the point in $\mathbb{R}^n$, which encode the pixel values. In this case the model takes as input n real values $F(f(\mathbf{x}_1), \cdots, f(\mathbf{x}_n))$.

It is important to note that Theorem 3.1 assumes that the group G acts continuously on the data manifold $\mathcal{X}$. However, this assumption is violated for image data as an image is a non-continuous function of coordinates due to the discrete pixel grid. We overcome the discontinuity issue by preprocessing images with Gaussian smoothing.

To use LieGG for image data, we explicitly write the condition that the matrix $\mathfrak{h}$ is an infinitesimal generator for the image data $\mathcal{X}$ on the grid $(\mathbf{x}_1, \cdots, \mathbf{x}_n)$, where $\mathbf{x}_p[j]$ corresponds to the projection of $\mathbf{x}_p$ onto the j-th coordinate axis. The condition reads as follows:

$$F(f(e^{\mathfrak{h}\cdot t}\circ \mathbf{x}_1),\cdots,f(e^{\mathfrak{h}\cdot t}\circ \mathbf{x}_n))=0 \leftrightarrow \sum_{p=1,k=1,j=1}^{p=n,k=2,j=2} \frac{\partial F}{\partial f(\mathbf{x}_p)}\cdot\frac{\partial f(\mathbf{x}_p)}{\partial \mathbf{x}_p[k]}\cdot \mathfrak{h}_{k,j}\cdot \mathbf{x}_p[j]=0 \quad (3)$$

where $t \in \mathbb{R}$. The resulting condition is the equation in 4 variables $\mathfrak{h}_{k,j},\ k=1,2;\ j=1,2$. Thus, in this case, the polarization matrix has 4 columns. Thus, in equation 3 a space of potential image symmetries is reduced to 4-dimensional spatial subgroup. This is to provide a prior for the LieGG to retrieve meaningful symmetries from for images as the full-scale space of the potential symmetries is overwhelming for image data. This is a strong yet proven to useful prior for vision models [1, 44, 2, 3, 4].

In practice, the calculation of LieGG for vision models consists of the following steps: (i) preprocessing images by doing Gaussian smoothing, (ii) training a neural network, (iii) computing polarization matrix using equation 3.

4.2 Symmetry bias and symmetry variance

The scope of LieGG is not limited to the study of symmetry, but also to the evaluation of the degree of neural network invariance. If we know the true symmetries, we can apply our method to analyze how close the extracted symmetries are to the true ones; and the degree of network invariance to these symmetries. The former we alias *symmetry bias*, and the later - *symmetry variance*. To understand this we show how we can estimate the degree of network invariance to the element g of a Lie group:

$$R(g) \coloneqq \mathbb{E}(F(g\cdot \mathbf{x})-F(\mathbf{x}))^2 \sim \frac{1}{|D|}\sum_{\mathbf{x}^i\in D}(F(g\cdot \mathbf{x}^i)-F(\mathbf{x}^i))^2 \coloneqq \tilde{R}(g).$$

Let $\mathfrak{h}$ is an element from Lie algebra, that corresponds to g, i.e.

$$\tilde{R}(g)=\frac{1}{|D|}\sum_{\mathbf{x}^i\in D}(F(\exp(\mathfrak{h}t)\cdot \mathbf{x}^i)-F(\mathbf{x}^i))^2=$$
$$\frac{1}{|D|}\sum_{\mathbf{x}^i\in D}(t\cdot\sum_{k,j}\frac{\partial F(\mathbf{x}^i)}{\partial x_k}\cdot \mathfrak{h}_{kj}\cdot \mathbf{x}^i_j+\mathcal{O}(t^2))^2=$$
$$\frac{1}{|D|}\sum_{\mathbf{x}^i\in D}t^2\cdot(\sum_{k,j}\frac{\partial F(\mathbf{x}^i)}{\partial x_k}\cdot \mathfrak{h}_{kj}\cdot \mathbf{x}^i_j)^2+\mathcal{O}(t^3)$$

We denote the matrix with the i-th row equal to $\frac{\partial F(\mathbf{x}^i)}{\partial x_k}\cdot \mathbf{x}^i_j$ as $\mathcal{E}$. Then

$$\tilde{R}(g)=\frac{1}{|D|}(t^2\cdot\|\mathcal{E}\cdot\mathfrak{h}\|_2^2+\mathcal{O}(t^3)),$$

where $\mathfrak{h}$ is a vector with coordinates equal to $\mathfrak{h}_{kj}$. To calculate the value of $\mathcal{E}\cdot\mathfrak{h}$ we use SVD of matrix $\mathcal{E}$. Lets $\mathcal{E}=U\Sigma V^*$, where U,V - are semi-unitary matrixes and Σ is a diagonal matrix, and $\mathfrak{h}=\sum_l h_l\mathbf{v}_i$. We can estimate $\tilde{R}(g)$ in the following way:

$$\tilde{R}(g)\sim\frac{1}{|D|}\|\mathcal{E}\cdot\mathfrak{h}\|_2^2=\frac{1}{|D|}\sum_l h_l^2(\sigma_l)^2, \quad (4)$$

where σ_l is the l-th singular value.

In particular, the equation 4 shows that singular values reflect the degree of network invariance to transformations corresponding to singular vectors. With this, we use the smallest singular value of the network polarization matrix to define the *symmetry variance* as $\mathcal{V}(\mathcal{E})=|\mathcal{E}|^{-1}\sigma_{min}(\mathcal{E})^2$, where $|\mathcal{E}|$ denotes number of rows in the network polarization matrix. Moreover, when the symmetries are known, the difference between the singular vectors and the generator of the known symmetry group equates to the *symmetry bias*, i.e. $\mathcal{B}_i(\mathcal{E},\mathcal{G})=||\tilde{v}_i(\mathcal{E})-\tilde{\mathcal{G}}||_F$, where $\tilde{v}_i(\mathcal{E})$ returns the i-th singular vector reshaped to a matrix, and $\tilde{\mathcal{G}}$ is a ground truth symmetry group generator, which is the closest to $\tilde{v}_i(\mathcal{E})$, e.g. the closest skew-symmetric matrix for $O(n)$.

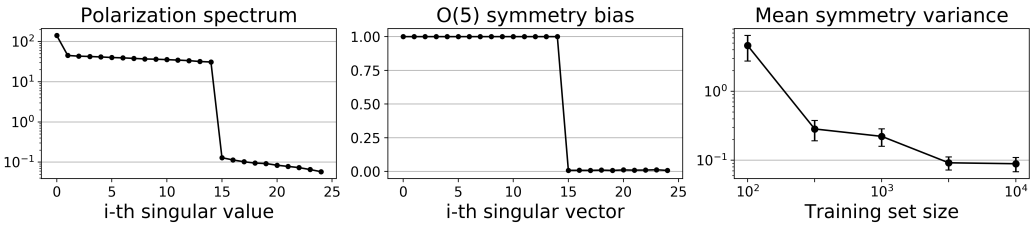

Figure 2: Singular values of the network polarization matrix and symmetry biases of the multi-layer perceptron fitted on $\mathrm{O}(5)$ invariant regression task. **(right)** Comparison of mean symmetry variance for different training set sizes.

5 Experiments

5.1 Implementation details

We use a multi-layer perceptron with linear layers followed by Swish activation [45]. For the regression experiment, we use the MLP with 3 layers and set the hidden dimension to 32. For the classification experiment we use the MLP with a classification head attached, and train various configurations varying a number of parameters, depth, and width. We train all of the models with Adam optimizer [46] setting the learning rate set to 10^{-3}. The results for all of the experiments are averaged over 8 common fixed random seeds.

5.2 Retrieving symmetries from a synthetic regression

Firstly, we perform the experiment on a synthetic problem to test the ability of our method to accurately retrieve the symmetry learned by the network. To do so, we adopt the invariant regression task from [8] with clean $\mathrm{O}(5)$ symmetry built-in. The regression function $f : \mathbb{R}^{2\times 5} \to \mathbb{R}$ is given by:

$$f(\mathbf{x}_1, \mathbf{x}_2) = \sin(\|\mathbf{x}_1\|_2) - 0.5 \cdot \|\mathbf{x}_2\|_2^3 + \frac{\mathbf{x}_1 \cdot \mathbf{x}_2^T}{\|\mathbf{x}_1\|_2 \cdot \|\mathbf{x}_2\|_2} \tag{5}$$

With this, we construct the training dataset to fit the regression function with the multi-layer perceptron. We note that the MLP is an unconstrained feed-forward neural network with no a priori knowledge of symmetries in the data. Once the model is converged, we apply our method to analyze how close are the retrieved symmetries to the clean $\mathrm{O}(5)$ group (symmetry bias), and how invariant is the network to these symmetries (symmetry variance). We also study how the ability of the network to learn symmetries depends on a number of available training samples. We report the results in Figure 2.

As can be seen from Figure 2a, the spectrum of the network polarization matrix is zeroing, indicating that the model becomes invariant to the transformations encoded by the corresponding singular vectors, and these singular vectors are the generators of the rotation group as indicated by zero $\mathrm{O}(5)$ symmetry bias (Figure 2b). We also observed that the network can accurately learn the symmetry group generators even on a small training set size. However, a larger training set size results in the network having smaller symmetry variance, i.e. being more invariant. For additional validation, we used the LieGG to extract symmetries from the randomly initialized $\mathrm{O}(5)$ invariant EMLP [8]. For the EMLP, the LieGG recorded zero symmetry variance and zero symmetry bias with respect to the $\mathrm{O}(5)$ group without any model pre-training.

This experiment demonstrates that on a synthetic task, our method can accurately retrieve the symmetries learned or built-in. When the symmetries are learned, the larger training dataset facilitates learning invariances for the model.

5.3 Retrieving symmetries from rotation MNIST

In this experiment, we test the ability of our method to retrieve learned symmetries on a more challenging rotation MNIST dataset [47]. We train the feed-forward network to output the class of

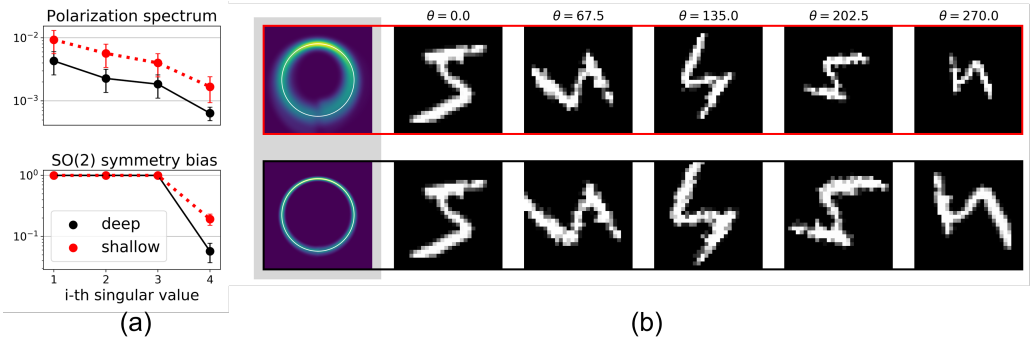

Figure 3: **(a)** Symmetrical properties of the shallow and deep networks: network polarization spectrum, and SO(2) symmetry bias. **(b)** Shaded: kernel density of learned 2D rotations on a perfect circle. Non shaded: visualizing learned Lie algebras for various rotation angles (in columns). Top row: shallow network. Bottom row: deep network.

the digit invariant to 2D in-plane rotations. The symmetry group that the model should learn is thus $\mathrm{SO}(2)$. We note that different from the synthetic dataset, the symmetries in the rotation MNIST are much noisier since the rotations are coarsened by the projection onto a pixel grid. In addition, the network has a classification error, hence it serves as only an approximate data discriminator function. In this scenario, we test how well our method can retrieve the noisy symmetries.

We experiment with two architectures: a shallow 2-layer perceptron and a deeper 6-layer network with 4 times number of parameters. Once, the networks' performance plateaus on the validation split, we terminate the training, and apply our method to retrieve learned symmetries. We plot the spectrum of the network polarization matrix, $\mathrm{SO}(2)$ symmetry bias and visualize learned symmetries for shallow and deep models in Figure 3.

From Figure 3a we observe that for both shallow and deep models, our method can robustly retrieve the rotation symmetry as indicates by zeroing symmetry bias with respect to $\mathrm{SO}(2)$ group, given the 4-dimensional spatial subgroup prior. However, we noticed that the rotation learned by the shallow model is in general noisier than the symmetry learned by the deeper networks as indicated by a higher symmetry bias. We visualize it in Figure 3b by sampling learned Lie algebras and applying them to transform an input image. We observe that the symmetry learned by the shallow model diverges from a clean rotation and also contains scaling and shearing. We also see, that on rotation-MNIST, the spectrum of the network polarization matrix is zeroing not as fast as in the synthetic case. It indicates that the models, although do learn the rotation symmetry, do not become fully invariant to it.

This experiment demonstrates that our method can robustly retrieve noisy symmetries from the data even when an imprecise discriminator function is used.

5.4 Symmetries in networks with different configurations

Observing the differences in symmetrical properties for different network configurations in Experiment 5.3, we further conduct the study on how symmetry variance and symmetry bias depend on width, depth, and a number of parameters in the network. In this experiment, we are interested to know if for some architecture it is easier to learn symmetries from the data.

To obtain different configurations, we sample a number of parameters from a range of $[40000, 200000]$ with a step of 20000, and vary the depth of the network from 1 to 5 hidden layers. We train the networks until convergence on the validation split and apply our method to study how well they learn the symmetries. We also record the accuracy of the models to correlate it with their symmetry variance and symmetry bias. We report results in Figure 4.

From Figure 4a we can see that we make the following conclusions on how the configurations of the network influence its symmetrical properties:

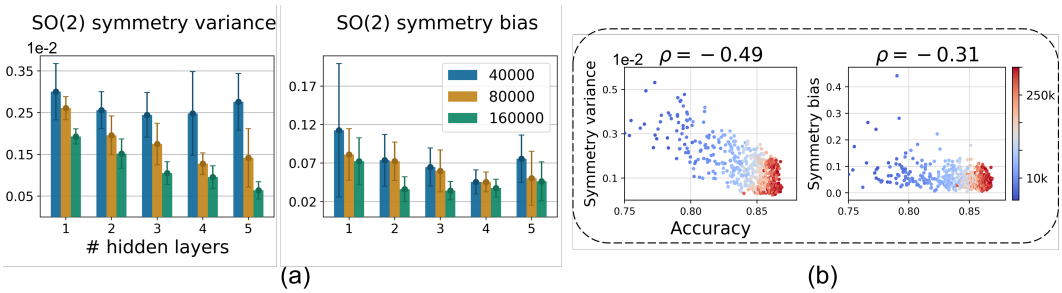

Figure 4: **(a)** SO(2) symmetry variance and bias for the networks with different width, depth and number of parameters. **(b)** Correlation plot of symmetry variance and bias versus a test accuracy of the model. Each point corresponds to a network. The color indicates a number of parameters.

Number of parameters: we observe that the networks with more parameters learn to become more invariant than their shallow counterparts as indicated by consistently smaller symmetry variance. Also, we see that the networks with fewer number of parameters generally learn less precise symmetries as shown by higher symmetry bias.

Depth / width: our results indicate that the sufficiently parameterized deep networks are better able to learn invariances than the wide ones with the same amount of parameters, i.e. deeper the network is, more invariant it becomes in the direction of the learned symmetries. At the same time, this does not hold for the quality of learned symmetry groups as indicated by fluctuating symmetry bias, when depth > 3. We see that the network with balanced width and depth achieve smaller symmetry bias than the deeper ones.

Symmetry vs. accuracy: we also test how the symmetry variance and bias correlates with the performance of the model. As we can see from Figure 4b, the symmetry variance and bias have moderate correlation coefficients $\rho_{\text{var}} = -0.49$ ($p\text{-value} = 10^{-35}$) and $\rho_{\text{bias}} = -0.31$ ($p\text{-value} = 10^{-14}$) respectively. This trend indicates that more accurate models generally also tend to have lower symmetry variance and bias. However such correlation is not strict, and it is possible to have a network, which does not learn the most precise symmetries from the training data, but still perform well on the test.

5.5 Tuning the layer-wise symmetries in deep networks

In the previous experiment we analyzed how the symmetrical properties of the networks depend on the configuration of the model. Now, we ask if by modifying training regime, we can facilitate learning the symmetries for the network. In this experiment we test two known approaches to improve the training: (i) pre-training with fine-tuning, and (ii) layer-wise training approach [40].

To simulate the pre-training a part of the network weights, we take randomly initialized 7 layers perceptron (mother network), and extract sub-network that consist of the first 3 layers. We then attach the classification head to this sub-network and train it independently from the mother network on rotation-MNIST dataset. After that, we re-inject the learned weights into the mother network and retrain the model. For fine-tuning, we train all of the weights jointly. For layer-wise training, we freeze the pre-trained part and only train the remaining layers. We then apply our method to evaluate the symmetrical properties of the models. We plot the symmetry variance and symmetry bias after each layer of the network in Figure 5.

Final and pen-ultimate layer symmetries: from Figure 5 we observe that for different training regimes, the network learns approximately the same symmetries at the same degree of invariance in a final layer. Surprisingly, we also noticed that in a pen-ultimate layer, the symmetries for all of the training regimes come very close as measured by symmetry variance and bias. Since the last layers are directly responsible for providing outputs for the task [1], we hypothesize that *there exist preferred*

[1] pen-ultimate layer prepares representations to be linearly separated by final classification layer [48]

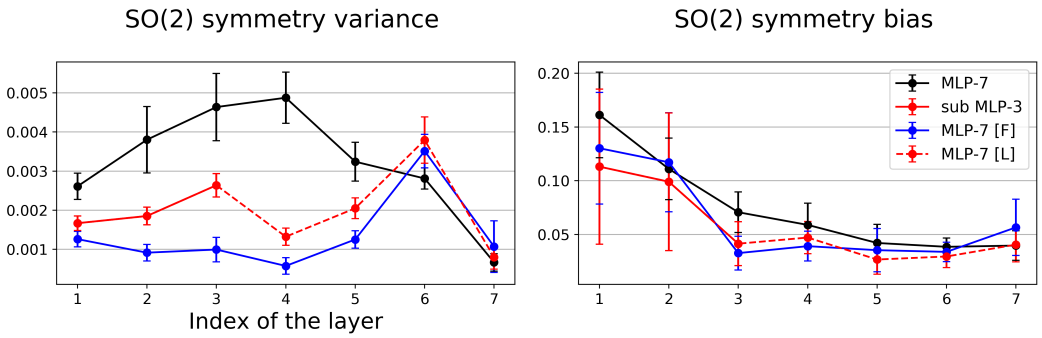

Figure 5: SO(2) symmetry variance and bias measured layer-wise for the network trained on the rotation-MNIST dataset. Lines of different colors correspond to the different training regimes of the model. Training scenarios are: **MLP-7** for 7-layer perceptron, **sub MLP-2** for 3-layer sub-network, **MLP-7[F]** for joint fine-tuning, **MLP-7[L]** for layer-wise training.

optimal symmetry level for the whole network that depends on the architecture and data, but not from the training regime of the model.

Intermediate symmetries: on the other hand, we observed that the *symmetries in the intermediates layers of the network do depend on the training regime*. Firstly, the pre-trained sub-network learns more accurate symmetry with a higher level of invariance compared to when it is trained as a part of the deeper model. We attribute it to the fact that symmetries are defined on the level of the labels, hence the invariances learned by the model in the final layer should be kept at a minimum. Secondly, after the fine-tuning or layer-wise training, symmetries in the intermediate layers of the network are both more precise and utilized at a higher degree of invariance as indicated by smaller symmetry bias and variance respectively. It indicates that *we can facilitate learning intermediate invariances in the network* by using pre-training followed by fine-tuning or layer-wise learning approach. We assume this insight can be useful for transfer learning when it is important to transfer as much inductive bias as possible into the other models or to other tasks.

6 Conclusion

Summary We present the LieGG - an approach to retrieve the symmetries learned by a neural network. Conveniently, our method extracts the symmetries directly in a form of corresponding Lie group generators. This allows to explicitly obtain the symmetries learned by the model without specifying any set of transformations of interest beforehand. The LieGG also evaluates how the model is invariant in the direction of a learned symmetry by computing the symmetry variance. We make use of our approach by studying how the quality of the symmetry learned by the network depends on its configuration and a training regime. With this, by improving an interpretability of the symmetry learning, we can also identify good practices that improve the symmetries in a model.

Limitations and Future Work The proposed method can facilitate the interpretability of neural networks and help to design efficient ways of incorporating the symmetries into the model. However, some limitations remain. In particular, the LieGG is currently limited to the instances of GL connected group, which hinders the analysis of complex symmetries, e.g. where the transformation can not be described as a group action. It is an intriguing direction for future research to extend the mechanism of retrieving the symmetries to more complex transformations. Nonetheless, the LieGG provides an important building block on the way to increasing the interpretability of the symmetries learned by the neural networks. Also, in our analysis, we focus on a specific family of models, i.e. feed-forward multi-layer perceptions trained on the well-controlled datasets. We assume that the further large-scale analysis of the LieGG in complex scenarios and with more advanced models (e.g. graph networks, and self-attention transformers) can provide a more general understanding of the symmetry learning in deep networks.

References

[1] Taco Cohen and Max Welling. Group equivariant convolutional networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2990–2999, New York, New York, USA, 20–22 Jun 2016. PMLR.

[2] Daniel E. Worrall, Stephan J. Garbin, Daniyar Turmukhambetov, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7168–7177, 2017.

[3] Maurice Weiler and Gabriele Cesa. General E(2)-Equivariant Steerable CNNs. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[4] Erik Jenner and Maurice Weiler. Steerable partial differential operators for equivariant neural networks. In *International Conference on Learning Representations*, 2022.

[5] Ivan Sosnovik, Michał Szmaja, and Arnold Smeulders. Scale-equivariant steerable networks. In *International Conference on Learning Representations*, 2020.

[6] Ivan Sosnovik, Artem Moskalev, and Arnold Smeulders. Disco: accurate discrete scale convolutions. *arXiv preprint arXiv:2106.02733*, 2021.

[7] Erik J Bekkers. B-spline {cnn}s on lie groups. In *International Conference on Learning Representations*, 2020.

[8] Marc Finzi, Max Welling, and Andrew Gordon Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. *Arxiv*, 2021.

[9] Marc Finzi, Gregory Benton, and Andrew G Wilson. Residual pathway priors for soft equivariance constraints. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[10] Naman Khetan, Tushar Arora, Samee Ur Rehman, and Deepak K. Gupta. Implicit equivariance in convolutional networks, 2021.

[11] Allan Zhou, Tom Knowles, and Chelsea Finn. Meta-learning symmetries by reparameterization, 2020.

[12] Rui Wang, Robin Walters, and Rose Yu. Approximately equivariant networks for imperfectly symmetric dynamics. *arXiv preprint arXiv:2201.11969*, 2022.

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[14] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.

[15] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *International Conference on Learning Representations*, 2020.

[16] Chris Olah, Nick Cammarata, Chelsea Voss, Ludwig Schubert, and Gabriel Goh. Naturally occurring equivariance in neural networks. *Distill*, 2020. https://distill.pub/2020/circuits/equivariance.

[17] Ian Goodfellow, Honglak Lee, Quoc Le, Andrew Saxe, and Andrew Ng. Measuring invariances in deep networks. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

[18] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence, 2014.

[19] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, Contour and Grouping in Computer Vision*, pages 319–345. Springer Verlag, 1999. International Workshop on Shape, Contour and Grouping in Computer Vision ; Conference date: 26-05-1998 Through 29-05-1998.

[20] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets, 2017.

[21] Ivan Sosnovik, Artem Moskalev, and Arnold Smeulders. How to transform kernels for scale-convolutions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1092–1097, 2021.

[22] Daniel Worrall and Max Welling. Deep scale-spaces: Equivariance over scale. *Advances in Neural Information Processing Systems*, 32, 2019.

[23] Fabio Anselmi, Georgios Evangelopoulos, Lorenzo Rosasco, and Tomaso Poggio. Symmetry-adapted representation learning. *Pattern Recognition*, 86:201–208, 2019.

[24] Jascha Sohl-Dickstein, Ching Ming Wang, and Bruno A Olshausen. An unsupervised algorithm for learning lie group transformations. *arXiv preprint arXiv:1001.1027*, 2010.

[25] Nima Dehmamy, Robin Walters, Yanchen Liu, Dashun Wang, and Rose Yu. Automatic symmetry discovery with lie algebra convolutional network. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[26] Marissa Connor and Christopher Rozell. Representing closed transformation paths in encoded network latent space. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3666–3675, 2020.

[27] Rajesh Rao and Daniel Ruderman. Learning lie groups for invariant visual perception. *Advances in neural information processing systems*, 11, 1998.

[28] Benjamin Culpepper and Bruno Olshausen. Learning transport operators for image manifolds. *Advances in neural information processing systems*, 22, 2009.

[29] Uri Alon. *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman and Hall/CRC, 2006.

[30] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. *Advances in neural information processing systems*, 30, 2017.

[31] Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. In *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 2306–2327. PMLR, 09–12 Jul 2020.

[32] Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2847–2854. PMLR, 06–11 Aug 2017.

[33] Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations*, 2021.

[34] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers, 2018.

[35] Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

[36] Sattar Vakili, Michael Bromberg, Jezabel Garcia, Da-shan Shiu, and Alberto Bernacchia. Uniform generalization bounds for overparameterized neural networks, 2021.

[37] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[38] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[39] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *International Conference on Machine Learning*, pages 2712–2721. PMLR, 2019.

[40] Y. Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, and U. Montreal. Greedy layer-wise training of deep networks. volume 19, 01 2007.

[41] Barret Zoph, Golnaz Ghiasi, Tsung-Yi Lin, Yin Cui, Hanxiao Liu, Ekin Dogus Cubuk, and Quoc Le. Rethinking pre-training and self-training. *Advances in neural information processing systems*, 33:3833–3845, 2020.

[42] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4918–4927, 2019.

[43] Peter J Olver. *Applications of Lie groups to differential equations*, volume 107. Springer Science & Business Media, 2000.

[44] J-H. Jacobsen, B. de Brabandere, and A. W. M. Smeulders. Dynamic steerable blocks in deep residual networks. In *British Machine Vision Conference*, 2017.

[45] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

[46] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[47] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 473–480, New York, NY, USA, 2007. Association for Computing Machinery.

[48] Weinan E and Stephan Wojtowytsch. On the emergence of simplex symmetry in the final and penultimate layers of neural network classifiers, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]