Automatic Symmetry Discovery with Lie Algebra Convolutional Network

Nima Dehmamy
Northwestern University
nimadt@bu.edu

Robin Walters
Northeastern University

Yanchen Liu
Northeastern University

Dashun Wang
Northwestern University

Rose Yu
University of California San Diego

Abstract

Existing equivariant neural networks require prior knowledge of the symmetry group and discretization for continuous groups. We propose to work with Lie algebras (infinitesimal generators) instead of Lie groups. Our model, the Lie algebra convolutional network (L-conv) can automatically discover symmetries and does not require discretization of the group. We show that L-conv can serve as a building block to construct any group equivariant feedforward architecture. We discuss how CNNs and Graph Convolutional Networks are related to and can be expressed as L-conv with appropriate groups. We also derive the MSE loss for a single L-conv layer and find a deep relation with Lagrangians used in physics, with some of the physics aiding in defining generalization and symmetries in the loss landscape. Conversely, L-conv could be used to propose more general equivariant ansätze for scientific machine learning.

Incorporating symmetries into a deep learning architecture can reduce sample complexity, improve generalization, while significantly decreasing the number of model parameters (Cohen et al., 2019b; Cohen & Welling, 2016b; Ravanbakhsh et al., 2017; Ravanbakhsh, 2020; Wang et al., 2020). For instance, Convolutional Neural Networks (CNN) (LeCun et al., 1989, 1998) implement translation symmetry through weight sharing. General principles for constructing symmetry-aware group equivariant neural networks were introduced in Cohen & Welling (2016b), Kondor & Trivedi (2018), and Cohen et al. (2019b).

However, most work on equivariant networks requires knowing the symmetry group *a priori*. A different equivariant model needs to be re-designed for each symmetry group. In practice, such knowledge of the symmetries may not be available. Constructing and selecting the right equivariant networks becomes quite tedious. Furthermore, many existing works are limited to *finite groups* such as permutations Hartford et al. (2018); Ravanbakhsh et al. (2017); Zaheer et al. (2017), 90 degree rotations Cohen et al. (2018) or dihedral groups D_N $E(2)$ Weiler & Cesa (2019).

For a continuous group, existing approaches either discretize the group Weiler et al. (2018a,b); Cohen & Welling (2016a), or use a truncated sum over irreducible representations (irreps) Weiler & Cesa (2019); Weiler et al. (2018a) via spherical harmonics in Worrall et al. (2017) or more general Clebsch-Gordon coefficients Kondor et al. (2018); Bogatskiy et al. (2020). These approaches are prone to approximation error. Recently, Finzi et al. (2020) propose to approximates the integral over the Lie group by Monte Carlo sampling. This approach requires implementing the matrix exponential and obtaining a local neighborhood for each point. Both parametrizing Lie groups for sampling and finding irreps are computationally expensive. Finzi et al. (2021) provide a general algorithm for constructing equivariant multi-layer perceptrons (MLP), but require explicit knowledge of the group to encode its irreps, and solving a set of constraints.

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

Learning symmetries from data has been studied in limited settings for commutative Lie groups as in Cohen & Welling (2014), 2D rotations and translations in Rao & Ruderman (1999), Sohl-Dickstein et al. (2010) or permutations (Anselmi et al., 2019). For full symmetry discovery, MSR (Zhou et al., 2020) uses meta-learning, which our approach can use as well. Yet, their weight-sharing scheme and the symmetry generators are very different from ours. Our approach can use much fewer parameters and has a direct interpretation using Lie algebras (SI B.3). (Benton et al., 2020) propose Augerino, uses data augmentation to learn equivariance. It also involves Lie algebras, but is restricted to a subgroup of 2D affine transformations and requires matrix logarithm and sampling (SI B.3). Our approach is more general and does not require these.

Contributions We provide a novel framework for equivariant architectures that can enable automatic discovery of symmetry from data. Our method stems from the fact that Lie groups can be constructed from a set of infinitesimal generators, called the Lie algebra. The Lie algebra has a finite basis, assuming the group is finite-dimensional. Working with the Lie algebra basis allows us to encode an infinite group without discretizing or summing over irreps. Additionally, all Lie algebras have the same general structure and hence can be implemented the same way.

We propose Lie Algebra Convolutional Network (**L-conv**), a novel architecture that can automatically discover symmetry from data. Our main contributions can be summarized as follows:

- We propose the Lie algebra convolutional network (**L-conv**), a building block for constructing group equivariant architectures.
- We show that multi-layer L-conv can approximate group convolutional layers, including CNN, and find graph convolutional networks to be a special case of L-conv.
- The Lie algebra basis in L-conv can be learned, enabling automatic symmetry discovery.
- We find that MSE loss function for L-conv generalizes important Lagrangians in physics.
- We find robustness and equivariance in the loss landscape can be expressed as Euler-Lagrange equations and Noether currents, transferring ideas from physics to ML.

A key feature of L-conv is its simplicity. In fact, we show that it can be implemented as a set of graph convolutional networks, with the Lie algebra generators replacing the graph adjacency matrix. Learning symmetries inherently requires learning more parameters than a pre-encoded symmetry. Thus, in problems where the desired symmetry group is known, this knowledge should be used as inductive bias to directly encode the symmetry (into L-conv or other architectures). Thus symmetry discovery capabilities of L-conv are particularly useful in cases where the symmetries are *unknown* and where we do not have good inductive bias.

1 Background

We now review the core concepts we build upon: equivariance, group convolution and Lie algebras.

Notations Unless explicitly stated, a in A^a is an index, not an exponent. We use the Einstein summation $A^a B_{ab} = \sum_a A^a B_{ab} = [AB]_b$, where a repeated upper and lower index are summed.

Equivariance Let $\mathcal{S}$ be a topological space on which a Lie group G (continuous group) acts from the left, meaning for all $\boldsymbol{x} \in \mathcal{S}$ and $g \in G$, $g\boldsymbol{x} \in \mathcal{S}$. We refer to $\mathcal{S}$ as the base space. Let $\mathcal{F}$, the "feature space", be the vector space $\mathcal{F} = \mathbb{R}^m$. Each data point is a feature map $f : \mathcal{S} \to \mathcal{F}$. For instance, in images $f(\boldsymbol{x}) \in \mathcal{F}$ are the colors at pixel $\boldsymbol{x} \in \mathcal{S}$. The action of G on the input of f induces an action on feature maps. For "scalar" features, for $u \in G$, the transformed features $u \cdot f$ are given by

$$u \cdot f(\boldsymbol{x}) = f(u^{-1}\boldsymbol{x}). \tag{1}$$

Denote the space of all functions from $\mathcal{S}$ to $\mathcal{F}$ by $\mathcal{F}^{\mathcal{S}}$, so that $f \in \mathcal{F}^{\mathcal{S}}$. Let F be a mapping to a new feature space $\mathcal{F}' = \mathbb{R}^{m'}$, meaning $F : \mathcal{F}^{\mathcal{S}} \to \mathcal{F}'^{\mathcal{S}}$. We say F is *equivariant* under G if G acts on $\mathcal{F}'$ and for $u \in G$

$$u \cdot (F(f)) = F(u \cdot f). \tag{2}$$

Group Convolution Kondor & Trivedi (2018) showed that F is a linear equivariant map if and only if it performs a group convolution (G-conv). To define G-conv, we first lift $\boldsymbol{x}$ to elements in G (Kondor & Trivedi, 2018). To do the lift, we pick an origin $\boldsymbol{x}_0 \in \mathcal{S}$ and replace each point $\boldsymbol{x} = g\boldsymbol{x}_0$

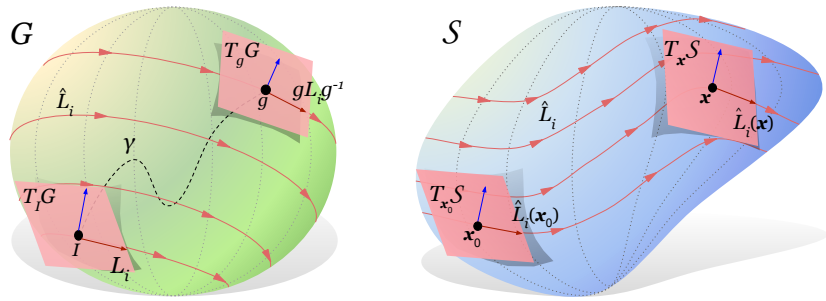

Figure 1: **Lie group and Lie algebra:** Illustration of the group manifold of a Lie group G (left). The Lie algebra $\mathfrak{g} = T_IG$ is the tangent space at the identity I. L_i are a basis for T_IG. If G is connected, $\forall g \in G$ there exist paths like γ from I to g and g can be written as a path-ordered integral $g = P\exp[\int_\gamma dt^i L_i]$. **Base space** Right is a schematic of the base space $\mathcal{S}$ as a manifold. The lift $\boldsymbol{x} = g\boldsymbol{x}_0$ takes $\boldsymbol{x} \in \mathcal{S}$ to $g \in G$, and maps the tangent spaces $T_{\boldsymbol{x}}\mathcal{S} \to T_gG$. Each Lie algebra basis $L_i \in \mathfrak{g} = T_IG$ generates a vector field $\hat{L}_i$ on the tangent bundle TG via the pushforward $\hat{L}_i(g) = gL_ig^{-1}$. Via the lift, L_i also generates a vector field $\hat{L}_i = \hat{L}_i^\alpha(\boldsymbol{x})\partial_\alpha = [gL_i\boldsymbol{x}_0]^\alpha\partial_\alpha$.

by g. We will often drop $\boldsymbol{x}_0$ for brevity and write $f(g) \equiv f(g\boldsymbol{x}_0)$. Let $\kappa : G \to \mathbb{R}^{m'} \otimes \mathbb{R}^m$ be a linear transformation from $\mathcal{F}$ to $\mathcal{F}'$. G-conv is defined as

$$[\kappa \star f](g) = \int_G \kappa(g^{-1}v)f(v)dv = \int_G \kappa(v)f(gv)dv, \tag{3}$$

where for brevity, we are denoting the Haar measure on G as $dv \equiv d\mu(v)$.

Equivariance of G-conv As shown in Kondor & Trivedi (2018), G-conv is equivariant. The equivariance of equation 3 can be readily checked using equation 1. For $w \in G$ we have

$$\begin{aligned}[\kappa \star w \cdot f](g) &= \int_G \kappa(v)w \cdot f(gv)dv = \int_G \kappa(v)f(w^{-1}gv)dv \\ &= [\kappa \star f](w^{-1}g) = w \cdot [\kappa \star f](g)\end{aligned} \tag{4}$$

Most existing literature on equivariance implements $\int_G$ by discretizing the group or summing over irreducible representations. We take a different approach and instead use the infinitesimal generators of the group. While a Lie group G is infinite, usually it can be generated using a small number of infinitesimal generator, comprising its "Lie algebra". We use the Lie algebra to introduce a building block from which any G-conv can be constructed. Next, we review core concepts about Lie algebras. Figure 1 shows an illustration of a Lie group, Lie algebra and the concept we discuss below.

Lie algebra Let G be a Lie group, which includes common continuous groups. Group elements $u \in G$ infinitesimally close to the identity element I can be written as $u \approx I + \epsilon^i L_i$ (note Einstein summation), where $L_i \in \mathfrak{g}$ with the Lie algebra $\mathfrak{g} = T_IG$ is the tangent space of G at the identity element. The Lie algebra has the property that it is closed under a Lie bracket $[\cdot,\cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$

$$[L_i, L_j] = c_{ij}{}^k L_k, \tag{5}$$

which is skew-symmetric and satisfies the Jacobi identity. Here the coefficients $c_{ij}{}^k \in \mathbb{R}$ or $\mathbb{C}$ are called the structure constants of the Lie algebra. For matrix representations of $\mathfrak{g}$, $[L_i, L_j] = L_iL_j - L_jL_i$ is the commutator. The L_i are called the infinitesimal generators of the Lie group.

Exponential map If the manifold of G is connected [1], an exponential map $\exp : \mathfrak{g} \to G$ can be defined such that $g = \exp[t^iL_i] \in G$. For matrix groups, if G is connected and compact, the matrix exponential is such a map and it is surjective. For most other groups (except $\mathrm{GL}_d(\mathbb{C})$ and nilpotent groups) it is not surjective. Nevertheless, for any connected group every $g \in G$ can be written as a product $g = \prod_a \exp[t_a^iL_i]$ (Hall, 2015). Making t_a^i infinitesimal steps $dt^i(s)$ tangent to a path γ from

[1] When G has multiple connected components, these results hold for the component containing I, and generalize easily for mutli-component groups such as $\mathbb{Z}_k \otimes G$ (Finzi et al., 2021).

I to g on G yields the surjective path-ordered exponential in physics, denoted as $g = P\exp[\int_\gamma dt^i L_i]$ (SI A, and see Time-ordering in Weinberg (1995, p143)).

Pushforward $L_i \in T_I G$ can be pushed forward to $\hat{L}_i(g) = gL_i g^{-1} \in T_g G$ to form a basis for $T_g G$, satisfying the same Lie algebra $[\hat{L}_i(g), \hat{L}_j(g)] = c_{ij}{}^k \hat{L}_k(g)$. The manifold of G together with the set of all $T_g G$ attached to each g forms the tangent bundle TG, a type of fiber bundle (Lee et al., 2009). $\hat{L}_i$ is a vector field on TG. The lift maps $\hat{L}_i$ to an equivalent vector field on $T\mathcal{S}$, which we will also denote by $\hat{L}_i$. Figure 1 illustrates the flow of these vector fields on TG and $T\mathcal{S}$.

2 Lie Algebra Convolutional Network

As mentioned above, the Lie algebra basis $L_i \in \mathfrak{g}$ can be used to construct the Lie group G using the exponential map. Similarly, we show that G-conv layers can be constructed using a more basic architecture utilizing the Lie algebra. The key idea is to approximate the kernel $\kappa(u)$ using localized kernels which can be constructed using the Lie algebra (Fig. 2). This is possible because the exponential map is a generalization of a Taylor expansion. The first step is to show that a G-conv whose kernel is concentrated near identity can be expanded in the Lie algebra.

Let $\delta_\eta(u) \in \mathbb{R}$ denote a normalized *localized kernel*, meaning $\int_G \delta_\eta(g)dg = 1$, and with support on a small neighborhood of size η near I (i.e., $\delta_\eta(I + \epsilon^i L_i) \to 0$ if $\|\epsilon\|^2 > \eta^2$). Let $\kappa_0(u) = W^0 \delta_\eta(u)$, where $W^0 \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$ are constants. The localized kernels κ_0 can be used to approximate G-conv. We first derive the expression for a G-conv whose kernel is κ_0.

Linear expansion of G-conv with localized kernel We can expand a G-conv whose kernel is $\kappa_0(u) = W^0 \delta_\eta(u)$ in the Lie algebra of G to linear order. With $v_\epsilon = I + \epsilon^i L_i$, we have (see SI A)

$$\begin{aligned} Q[f](g) = [\kappa_0 \star f](g) &= \int_G dv \kappa_0(v) f(gv) = \int_{\|\epsilon\|<\eta} dv_\epsilon \kappa_0(v_\epsilon) f(gv_\epsilon) \\ &= W^0 \int d\epsilon \delta_\eta(v_\epsilon) \left[f(g) + \epsilon^i g L_i \cdot \frac{d}{dg} f(g) + O(\epsilon^2) \right] \\ &= W^0 \left[I + \bar{\epsilon}^i g L_i \cdot \frac{d}{dg} \right] f(g) + O(\eta^2) \end{aligned} \tag{6}$$

where $W^0 \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$, and using $\int_G \delta_\eta(g)dg = \int d\epsilon \delta_\eta(v_\epsilon) = 1$ and $\epsilon^i \in \mathbb{R}^m \otimes \mathbb{R}^m$, we defined

$$\bar{\epsilon}^i = \int d\epsilon \delta_\eta(v_\epsilon) \epsilon^i \in \mathbb{R}^m \otimes \mathbb{R}^m. \tag{7}$$

Note that because $\|\epsilon\| < \eta$ we also have $\|\bar{\epsilon}\| < \eta$ (SI A, equation 24). Here $d\epsilon$ is the integration measure on the Lie algebra $\mathfrak{g} = T_I G$ induced by the Haar measure dv_ϵ on G.

Interpreting the derivatives In a matrix representation of G, we have $gL_i \cdot \frac{df}{dg} = [gL_i]^\beta_\alpha \frac{df}{dg^\beta_\alpha} = \mathrm{Tr}\left[[gL_i]^T \frac{df}{dg} \right]$. This can be written in terms of partial derivatives $\partial_\alpha f(\boldsymbol{x}) = \partial f / \partial \boldsymbol{x}^\alpha$ as follows. Using $\boldsymbol{x}^\rho = g^\rho_\sigma \boldsymbol{x}^\sigma_0$, we have $\frac{df(g\boldsymbol{x}_0)}{dg^\alpha_\beta} = \boldsymbol{x}^\beta_0 \partial_\alpha f(\boldsymbol{x})$, and so

$$\hat{L}_i f(\boldsymbol{x}) \equiv gL_i \cdot \frac{df}{dg} = [gL_i]^\alpha_\beta \boldsymbol{x}^\beta_0 \partial_\alpha f(\boldsymbol{x}) = [gL_i \boldsymbol{x}_0] \cdot \nabla f(\boldsymbol{x}) \tag{8}$$

Hence, for each L_i, the pushforward $gL_i g^{-1}$ generates a flow on $\mathcal{S}$ through the vector field $\hat{L}_i \equiv gL_i \cdot d/dg = [gL_i g^{-1} \boldsymbol{x}]^\alpha \partial_\alpha$ (Fig. 1).

Lie algebra convolutional (L-conv) layer Equation 6 states that for a kernel localized near identity the effect of the kernel can be summarized in W^0 and $\bar{\epsilon}^i \hat{L}_i$. It is important to note that we do not need to perform the integral over G explicitly anymore. Instead of working with a kernel κ_0, we only need to specify W^0 and $\bar{\epsilon}^i$. The construction in equation 6 turns out to be the basic building block of G-conv, as we show below. Hence, in general, we define the Lie algebra convolution (L-conv) as

$$\begin{aligned} Q[f](\boldsymbol{x}) &= W^0 \left[I + \bar{\epsilon}^i \hat{L}_i \right] f(\boldsymbol{x}) \\ &= W^0 \left[I + \bar{\epsilon}^i [gL_i \boldsymbol{x}_0]^\alpha \partial_\alpha \right] f(\boldsymbol{x}) \end{aligned} \tag{9}$$

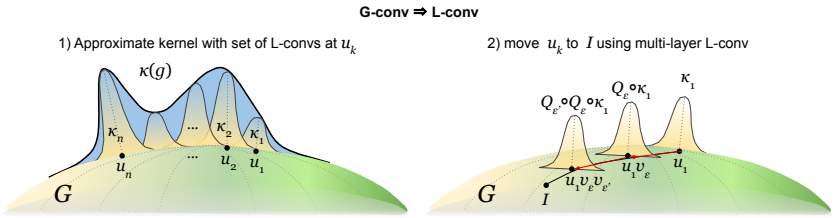

Figure 2: Sketch of the procedure for approximating G-conv using L-conv. First, the kernel is written as the sum of a number of localized kernels κ_k with support around u_k (left). Each of the κ_k is then moved toward identity by composing multiple L-conv layers $Q_{\epsilon'} \circ Q_\epsilon \ldots \kappa_k$ (right).

Being an expansion of G-conv, L-conv inherits the equivariance of G-conv, as we show next.

Proposition 1 (Equivariance of L-conv)**.** *With assumptions above, L-conv is equivariant under G.*

Proof: First, note that the components of $\hat{L}_i$ transform as $[\hat{L}_i(v\boldsymbol{x})]^\alpha = [vgL_i\boldsymbol{x}_0]^\alpha = v^\alpha_\beta \hat{L}_i(\boldsymbol{x})^\beta$, while the partial transforms as $\partial/\partial[v\boldsymbol{x}]^\alpha = [v^{-1}]^\gamma_\alpha \partial_\gamma$. As a result in $\hat{L}_i = [gL_i\boldsymbol{x}_0]^\alpha \partial_\alpha$ all factors of v cancel, meaning for $v \in G$, $\hat{L}_i(v\boldsymbol{x}) = \hat{L}_i(\boldsymbol{x})$. This is because of the fact that $\hat{L}_i \in T\mathcal{S}$ is a vector field (i.e. 1-tensor) and, thus, invariant under change of basis. Plugging into equation 9, for $w \in G$

$$\begin{aligned} w \cdot Q[f](\boldsymbol{x}) &= Q[f](w^{-1}\boldsymbol{x}) = W^0 \left[I + \overline{\epsilon}^i \hat{L}_i(w^{-1}\boldsymbol{x})\right] f(w^{-1}\boldsymbol{x}) \\ &= W^0 \left[I + \overline{\epsilon}^i \hat{L}_i(g)\right] f(w^{-1}\boldsymbol{x}) = W^0 \left[I + \overline{\epsilon}^i \hat{L}_i(g)\right] w \cdot f(\boldsymbol{x}) = Q[w \cdot f](\boldsymbol{x}) \quad (10) \end{aligned}$$

which proves L-conv is equivariant. □

Examples Using equation 8 we can calculate the form of L-conv for specific groups (details in SI A.2). For **translations** $G = T_n = (\mathbb{R}^n, +)$ we find the generators become simple partial derivatives $\hat{L}_i = \partial_i$ (SI A.2.2), yielding $f(\boldsymbol{x}) + \epsilon^\alpha \partial_\alpha f(\boldsymbol{x})$. For **2D rotations** (SI A.2.1) we find $\hat{L} \equiv (x\partial_y - y\partial_x) = \partial_\theta$, which in physics is called the angular momentum operator about the z-axis in quantum mechanics and field theories. For rotations with scaling, **Rotation and scaling** $G = SO(2) \times \mathbb{R}^+$, we have two L_i, one $\hat{L}_\theta = \partial_\theta$ from $so(2)$ and a scaling with $L_r = I$, yielding $\hat{L}_r = x\partial_x + y\partial_y = r\partial_r$. Next, we discuss the form of L-conv on discrete data.

2.1 Approximating G-conv using L-conv

L-conv can be used as a basic building block which can be combined to make more G-conv with more general kernels. Figure 2 sketches the argument described here (see also SI A.1).

Theorem 1 (G-conv from L-convs)**.** *G-conv equation 3 can be approximated using L-conv layers.*

Proof: The procedure involves two steps, as illustrated in Fig. 2: 1) approximate the kernel using localized kernels as the δ_η in L-conv; 2) move the kernels towards identity using multiple L-conv layers. The following lemma outline the details. □

Lemma 1 (Approximating the kernel)**.** *Let the kernel $\kappa : G \to \mathcal{F}' \otimes \mathcal{F}$ with $\int_G \|\kappa(g)\|^2 dg < \infty$ be continuously differentiable with $\|d\kappa(g)/dg\|^2 < \xi^2$, and with compact support over $G_0 \subset G$. Let $\kappa_k(g) = c_k \delta_\eta(u_k^{-1} g)$ be a set of N kernels with support on an η neighborhood of $u_k \in G$. Then there exist $c_k \in \mathcal{F}' \otimes \mathcal{F}$ and $u_k \in G$ such that $\tilde{\kappa} = \sum_{k=1}^N \kappa_k$ approximates κ, meaning $\int_G \|\kappa(g) - \tilde{\kappa}(g)\|^2 dg < \zeta^2$ for arbitrary small $\zeta \in \mathbb{R}_+$.*

Proof: See SI A.1 for details. The intuition is similar to the universal approximation theorem for neural networks (Hornik et al., 1989; Cybenko, 1989), only generalized to a group manifold instead of $\mathbb{R}$. Let B_0 be the set of $v_\epsilon = I + \epsilon^i L_i \in \mathfrak{g}$, with $\|\epsilon\|^2 < \eta^2$. Choose a set of $u_k \in G$ such that the neighborhoods $B_k = u_k B_0 \subset G$ cover the support G_0 of κ. The bound $\|d\kappa(g)/dg\|^2 < \xi^2$ means that on small enough neighborhoods $B_k \subset G$, for any two $u, v \in B_k$ we have $\|\kappa(u) - \kappa(v)\|^2 \leq \eta^2\xi^2$, where $|G_0|$ is the volume of the support of κ. Hence, for $g \in B_k$, $\kappa(g)$ can be approximated with $\kappa_k(g) = \kappa(u_k)\delta_\eta(u_k^{-1} g)$, with normalized localized kernels $\delta_\eta(g)$, and any element $u_k \in$

B_k. We show that the approximation error of using $\tilde{\kappa} = \sum_k \kappa_k$ to approximate κ is bounded by $\int_G dg \|\kappa(g) - \tilde{\kappa}(g)\|^2 < |G_0|\eta^2\xi^2$. Any desired error bound ζ can then be attained by choosing small enough η for neighborhood sizes. □

Thus, we can approximate a large class of kernels as $\kappa(g) \approx \sum_k \kappa_k(g)$ where the local kernels $\kappa_k(g) = c_k \delta_\eta(u_k^{-1} g)$ have support only on an η neighborhood of $u_k \in G$. Here $c_k \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$ are constants and $\delta_\eta(u)$ is as in equation 6. Using this, G-conv equation 3 becomes

$$[\kappa \star f](g) = \sum_k c_k \int dv \delta_\eta(u_k^{-1} v) f(gv) = \sum_k c_k [\delta_\eta \star f](g u_k). \tag{11}$$

The kernels κ_k are localized around u_k, whereas in L-conv the kernel is around identity. We can compose L-conv layers to move κ_k from u_k to identity.

Lemma 2 (Moving kernels to identity)**.** *κ_k can be moved near identity using a multilayer L-conv.*

Proof: In equation 11, write $u_k = v_\epsilon u'_k$, with $v_\epsilon = I + \epsilon^i L_i \in \mathfrak{g}$. Using the definition equation 9 an L-conv layer $Q_\epsilon = I - \epsilon^i \hat{L}_i$ performs a first order Taylor expansion (SI A.1) and so $Q_\epsilon[\delta_\eta](u'^{-1}_k v) = \delta_\eta(u_k^{-1} v) + O(\epsilon^2)$. Thus, applying one L-conv layer moves the localized kernel along v_ϵ on G. Writing u_k as the product of a set of small group elements $u_k = \prod_{a=1}^p v_a$, with $v_a = I + \epsilon_a^i L_i \in \mathfrak{g}$. Defining L-conv layers $Q_a = I - \epsilon_a^i \hat{L}_i$, we can write

$$\kappa_k(g) \approx c_k Q_p \circ \cdots \circ Q_1 \circ \delta_\eta(g) \tag{12}$$

meaning κ_k localized around u_k can be written as a p layer L-conv acting on a kernel $\delta_\eta(g)$, localized around the identity of the group. With $\|\epsilon_a\| < \eta$, the error in u_k is $O(\eta^{p+1})$. □

Thus, we conclude that any G-conv equation 3 can be approximated by multilayer L-conv. Furthermore, for compact G, using the theorem in Kondor & Trivedi (2018), we can show that any equivariant feedforward neural network can be approximated using multilayer L-conv with nonlinearities.

Equivariance of nonlinearity Pointwise nonlinearities give equivariant maps between scalar feature maps. To see this, let $\sigma : \mathbb{R} \to \mathbb{R}$. We extend $\sigma : \mathcal{F} \to \mathcal{F}$ by applying σ component-wise. Let $f : \mathcal{S} \to \mathcal{F}$ be a scalar feature map (i.e., $g \cdot f(\boldsymbol{x}) = f(g^{-1}\boldsymbol{x})$). Then

$$g \cdot (\sigma \circ (f))(\boldsymbol{x}) = \sigma \circ (f)(g^{-1}\boldsymbol{x}) = \sigma \circ (g \cdot f)(\boldsymbol{x}).$$

Since the composition of equivariant maps is equivariant, given equivariant linear mapping $Q : \mathcal{F}^{\mathcal{S}} \to \mathcal{F}'^{\mathcal{S}}$ (i.e. $g \cdot Q[f] = Q[g \cdot f]$), the layer $f \mapsto \sigma \circ Q[f]$ is equivariant. Hence we have the corollary:

Corollary 1. *Assume G is compact and G acts on $\mathcal{S}$ transitively. Any equivariant feedforward neural network (FNN) can be approximated using multilayer L-conv with point-wise nonlinearities.*

Proof: A FNN is defined as $\sigma_p \circ F_p[\cdots[\sigma_1 \circ F_1[f]](\boldsymbol{x})$ where F_k are linear and σ_k are point-wise nonlinearities. By Theorem 1 of Kondor & Trivedi (2018), any linear equivariant layer is a G-conv, which by Theorem 1 can be approximated by multilayer L-conv. Therefore, multilayer L-conv with nonlinearity can approximate any equivariant FNN. □

Finally, note that to our knowledge it is not known whether *every equivariant function* can be approximated by equivariant FNN for a Lie group G. Hence, the corollary above is *not* a universal approximation theorem for equivariant scalar functions in terms of L-conv. However, it does show that multilayer L-conv is equally expressive as other equivariant networks. Next, we discuss implementation details.

3 Discretized space and implementation: the tensor notation

In many datasets, such as images, $f(\boldsymbol{x})$ is not given as continuous function, but rather as a discrete array, with $\mathcal{S} = \{\boldsymbol{x}_0, \ldots \boldsymbol{x}_{d-1}\}$ containing d points. Each $\boldsymbol{x}_\mu$ represents a coordinate in higher dimensional space, e.g. on a 10×10 image, $\boldsymbol{x}_0$ is $(x, y) = (0, 0)$ point and $\boldsymbol{x}_{99}$ is $(x, y) = (9, 9)$.

Feature maps and group action In the tensor notation, we encode $\boldsymbol{x}_\mu \in \mathcal{S}$ as the canonical basis (one-hot) vectors in $\boldsymbol{x}_\mu \in \mathbb{R}^d$ with $[\boldsymbol{x}_\mu]_\nu = \delta_{\mu\nu}$ (Kronecker delta), e.g. $\boldsymbol{x}_0 = (1, 0, \ldots, 0)$. The features become $\boldsymbol{f} \in \mathcal{F} = \mathbb{R}^d \otimes \mathbb{R}^m$, meaning $d \times m$ tensors, with $f(\boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T \boldsymbol{f} = \boldsymbol{f}_\mu$. Although

$\mathcal{S}$ is discrete, the group acting on $\mathcal{F}$ can be continuous (e.g. image rotations). Any $G \subseteq \mathrm{GL}_d(\mathbb{R})$ of the general linear group (invertible $d \times d$ matrices) acts on $\boldsymbol{x}_\mu \in \mathbb{R}^d$ and $\boldsymbol{f} \in \mathcal{F}$. We define $f(g \cdot \boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T g^T \boldsymbol{f}, \forall g \in G$, so that for $w \in G$ we have

$$w \cdot f(\boldsymbol{x}_\mu) = f(w^{-1} \cdot \boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T w^{-1T} \boldsymbol{f} = [w^{-1}\boldsymbol{x}_\mu]^T \boldsymbol{f} \tag{13}$$

Dropping the position $\boldsymbol{x}_\mu$, the transformed features are matrix product $w \cdot \boldsymbol{f} = w^{-1T}\boldsymbol{f}$. We can write G-conv in this notation (SI B). Similarly, we can rewrite L-conv equation 6 in the tensor notation. Defining $v_\epsilon = I + \overline{\epsilon}^i L_i$

$$\begin{aligned} Q[\boldsymbol{f}](g) &= W^0 f\left(g\left(I + \overline{\epsilon}^i L_i\right)\right) = \boldsymbol{x}_0^T \left(I + \overline{\epsilon}^i L_i\right)^T g^T \boldsymbol{f} W^{0T} \\ &= \left(\boldsymbol{x} + \overline{\epsilon}^i [g L_i \boldsymbol{x}_0]\right)^T \boldsymbol{f} W^{0T}. \end{aligned} \tag{14}$$

Here, $\hat{L}_i = g L_i \boldsymbol{x}_0$ is exactly the matrix analogue of pushforward vector field $\hat{L}_i$ in equation 8. The equivariance of L-conv in tensor notation is again evident from the $g^T \boldsymbol{f}$, resulting in

$$Q[w \cdot \boldsymbol{f}](g) = \boldsymbol{x}_0^T v_\epsilon^T g^T w^{-1T} \boldsymbol{f} W^{0T} = Q[\boldsymbol{f}](w^{-1}g) = w \cdot Q[\boldsymbol{f}](g) \tag{15}$$

Tensor L-conv layer implementation In conclusion, the discrete space L-conv equation 14 can be rewritten using the global Lie algebra basis $\hat{L}_i$

$$Q[\boldsymbol{f}] = \left(\boldsymbol{f} + \hat{L}_i \boldsymbol{f} \overline{\epsilon}^i\right) W^{0T}, \qquad Q[\boldsymbol{f}]^a_\mu = \boldsymbol{f}^b_\mu [W^{0T}]^a_b + [\hat{L}_i]^\nu_\mu \boldsymbol{f}^c_\nu \left[W^i\right]^a_c \tag{16}$$

Where $W^i = W^0 \overline{\epsilon}^i$, $W^0 \in \mathbb{R}^{m_{in}} \otimes \mathbb{R}^{m_{out}}$ and $\overline{\epsilon}^i \in \mathbb{R}^{m_{in}} \otimes \mathbb{R}^{m_{in}}$ are trainable weights. The $\hat{L}_i$ can be either inserted as inductive bias or they can be learned to discover symmetries.

To implement L-conv we we note that the structure of equation 16 is quite similar to a Graph Convolutional Network (GCN) (Kipf & Welling, 2016). For each i, the shared convolutional weights are $\overline{\epsilon}^i W^{0T}$ and the aggregation function of the GCN, a function of the graph adjacency matrix, is $\hat{L}_i$ in L-conv. Thus, L-conv can be implemented as GCN modules for each $\hat{L}_i$, plus a residual connection for the $\boldsymbol{f}W^{0T}$ term. Figure 3 shows the schematic of the structure of such an L-conv layer. In a naive implementation, $\hat{L}_i$ can be general $d \times d$ matrices. However, being vector fields generated by the Lie algebra, $\hat{L}_i$ has a more constrained structure which allows them to be encoded and learned using much fewer parameters than a $d \times d$ matrix. Specifically, encoding the topology of $\mathcal{S}$ as a graph (see SI B.1), the incidence matrix replaces partial derivatives (Schaub et al., 2020) in equation 8 and the L_i become weighting of the edges. This weighting is similar to Gauge Equivariant Mesh (GEM) CNN (Cohen et al., 2019a). Indeed, in L-conv the lift $\boldsymbol{x}_\mu = g_\mu \boldsymbol{x}_0$ fixes the gauge by mapping neighbors of $\boldsymbol{x}_0$ to neighbors of $\boldsymbol{x}_\mu$. Changing how the discrete $\mathcal{S}$ samples an underlying continuous space will change g_μ and hence the gauge, similar to GEM-CNN.

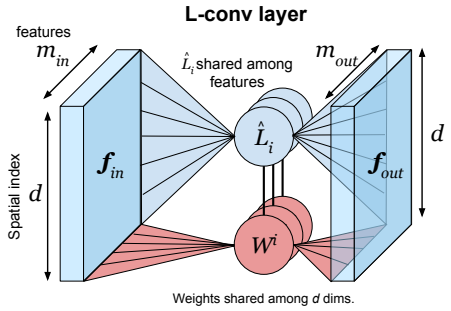

Figure 3: L-conv layer architecture. L_i only act on the d flattened spatial dimensions, and W^i only act on the m_{in} input features and returns m_{out} output features. For each i, L-conv is analogous to a GCN with d nodes and m_{in} features.

Choosing number of L_i Beside the width of W^0 and $\overline{\epsilon}^i$, the number n_L of L_i is a hyperparameter in L-conv. For instance, if $\mathcal{S}$ is a discretization of n dimensional space the symmetry group is likely $G \subset \mathrm{GL}_n(\mathbb{R}) \ltimes T_n$, with $n_L \sim O(n^2)$. Note that n_L is independent of the size d of the discretized space (e.g. number of pixels) and generally $n^2 \ll d$. Choosing n_L larger than the true number of L_i only results in an over-complete basis and shouldn't be a problem.

We conducted small controlled experiments to verify how multilayer L-conv approximates G-conv (SI C). We briefly discuss them here.

Learning symmetries using L-conv Rao & Ruderman (1999) introduced a basic version of L-conv and showed that it can learn 1D translation and 2D rotation. The learned L_i for 1D translation

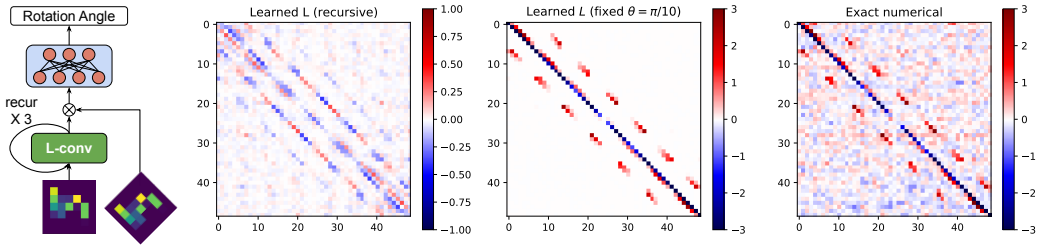

Figure 4: **Learning the infinitesimal generator of** $SO(2)$ Left shows the architecture for learning rotation angles between pairs of images (SI C.3). Next to it is the L learned using recursive L-conv in this experiment. Middle L is learned using a fixed small rotation angle $\theta = \pi/10$, and right shows L found using the numeric solution from the data.

reproduced finite translation well using $(I + \epsilon L)^N$, which is N recursive L-conv layers. Hence, their results prove that L-conv can be used to approximate CNN, as well as $SO(2)$ G-conv. We discuss the details of L-conv approximating CNN below in sec. 4. We also conducted more complex experiments using recursive L-conv to learn large rotation angle between two images (SI C). Figure 4 shows the learned L. Left shows $L \in so(2)$ learned using L-conv in 3 recursive layers to learn rotation angles between a pair of 7×7 random images $\boldsymbol{f}$ and $R(\theta)\boldsymbol{f}$ with $\theta \in [0, \pi/3)$. Middle and right of Fig. 4 are experiments with fixed small rotation angle $\theta = \pi/10$ (SI C.2). Middle is the L learned using L-conv and right is using the exact solution $R = (YX^T)(X^TX)^{-1}$. While the middle L is less noisy, it does not capture weights beyond first neighbors of each pixel. (also see SI C for a discussion on symmetry discovery literature.)

L-conv can potentially replace other equivariant layers in an architectures. We conducted limited experiments for this on small image datasets (SI D). L-conv allows one to look for potential symmetries in data which may have been scrambled or harbors hidden symmetries.

4 Relation to other architectures

CNN This is a special case of expressing G-conv as L-conv when the group is continuous 1D translations. The arguments here generalize trivially to higher dimensions. Rao & Ruderman (1999, sec. 4) used the Shannon-Whittaker Interpolation (Whitaker, 1915) to define continuous translation on periodic 1D arrays as $\boldsymbol{f}'_\rho = g(z)^\nu_\rho \boldsymbol{f}_\nu$. Here $g(z)^\nu_\rho = \frac{1}{d}\sum_{p=-d/2}^{d/2} \cos\left(\frac{2\pi p}{d}(z + \rho - \nu)\right)$ approximates the shift operator for continuous z. These $g(z)$ form a 1D translation group G as $g(w)g(z) = g(w+z)$ with $g(0)^\nu_\rho = \delta^\nu_\rho$. For any $z = \mu \in \mathbb{Z}$, $g_\mu = g(z = \mu)$ are circulant matrices that shift by μ as $[g_\mu]^\rho_\nu = \delta^\rho_{\nu-\mu}$. Thus, a 1D CNN with kernel size k can be written suing g_μ as

$$F(\boldsymbol{f})^a_\nu = \sigma\left(\sum_{\mu=0}^{k} \boldsymbol{f}^c_{\nu-\mu}[W^\mu]^a_c + b^a\right) = \sigma\left(\sum_{\mu=0}^{k} [g_\mu \boldsymbol{f}]^c_\nu [W^\mu]^a_c + b^a\right) \tag{17}$$

where W, b are the filter weights and biases. g_μ can be approximated using the Lie algebra and written as multi-layer L-conv as in sec. 2.1. Using $g(0)^\nu_\rho \approx \delta(\rho - \nu)$, the single Lie algebra basis $[\hat{L}]_0 = \partial_z g(z)|_{z\to 0}$, acts as $\hat{L}f(z) \approx -\partial_z f(z)$ (because $\int \partial_z \delta(z-\nu) f(z) = -\partial_\nu f(\nu)$). Its components are $\hat{L}^\nu_\rho = L(\rho - \nu) = \sum_p \frac{2\pi p}{d^2} \sin\left(\frac{2\pi p}{d}(\rho - \nu)\right)$, which are also circulant due to the $(\rho - \nu)$ dependence. Hence, $[\hat{L}\boldsymbol{f}]_\rho = \sum_\nu L(\rho - \nu)\boldsymbol{f}_\nu = [L \star \boldsymbol{f}]_\nu$ is a convolution. Rao & Ruderman (1999) already showed that this $\hat{L}$ can reproduce finite discrete shifts g_μ used in CNN. They used a primitive version of L-conv with $g_\mu = (I + \epsilon\hat{L})^N$. Thus, L-conv can approximate 1D CNN. This result generalizes easily to higher dimensions.

Relation with Graph Convolutional Networks (GCN) Let $\boldsymbol{A}$ be the adjacency matrix of a graph. In equation 16 if $\hat{L}_i = h(\boldsymbol{A})$, such as $\hat{L}_i = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, we obtain a GCN (Kipf & Welling, 2016) ($\boldsymbol{D}_{\mu\nu} = \delta_{\mu\nu}\sum_\rho \boldsymbol{A}_{\mu\rho}$ being the degree matrix). So in the special case where all neighbors of each node $< \mu >$ have the same edge weight, meaning $[\hat{L}_i]^\nu_\mu = [\hat{L}_i]^\rho_\mu$, $\forall \nu, \rho \in < \mu >$, equation 6 is uniformly aggregating over neighbors and L-conv reduces to a GCN. Note that this similarity

is not just superficial. In GCN $h(\boldsymbol{A}) = \hat{L}$ is in fact a Lie algebra basis. When $\hat{L} = h(\boldsymbol{A})$, the vector field is the flow of isotropic diffusion $d\boldsymbol{f}/dt = h(\boldsymbol{A})\boldsymbol{f}$ from each node to its neighbors. This vector field defines one parameter Lie group with elements $g(t) = \exp[h(\boldsymbol{A})t]$. Hence, L-conv for flow groups with a single generator are GCN. These flow groups include Hamiltonian flows and other linear dynamical systems. The main difference between L-conv and GCN is that L-conv can assign a different weight to each neighbor of the same node. As mentioned earlier, this is similar to GEM-CNN (Cohen et al., 2019a) with a fixed gauge set by g_μ. Next, we discuss the mathematical properties of loss functions for L-conv.

5 Group invariant loss

Loss functions of equivariant architectures are rarely discussed. Yet, recent work by Kunin et al. (2020) showed existence of symmetry directions in the loss landscape. To understand how the symmetry generators in L-conv manifest themselves in the loss landscape, we work out the explicit example of a mean square error (MSE) loss. Because G is the symmetry group, f and $g \cdot f$ should result in the same optimal parameters. Hence, the minima of the loss function need to be *group invariant*. One way to satisfy this is for the loss itself to be group invariant, which can be constructed by integrating over G (global pooling (Bronstein et al., 2021)). A function $I = \int_G dg F(g)$ is G-invariant (SI A.3). We can also change the integration to $\int_{\mathcal{S}} d^n x$ by change of variable $dg/d\boldsymbol{x}$ (see SI A.3 for discussion on stabilizers).

MSE loss The MSE is given by $I = \sum_n \int_G dg \|Q[f_n](g)\|^2$, where f_n are data samples and $Q[f]$ is L-conv or another G-equivariant function. In supervised learning the input is a pair f_n, y_n. G can also act on the labels y_n. We assme that y_n are either also scalar features $y_n : \mathcal{S} \to \mathbb{R}^{m_y}$ with a group action $g \cdot y_n(\boldsymbol{x}) = y_n(g^{-1}\boldsymbol{x})$ (e.g. f_n and y_n are both images), or that y_n are categorical. In the latter case $g \cdot y_n = y_n$ because the only representations of a continuous G on a discrete set are constant. We can concatenate the inputs to $\phi_n \equiv [f_n|y_n]$ with a well-defined G action $g \cdot \phi_n = [g \cdot f_n | g \cdot y_n]$. The collection of combined inputs $\Phi = (\phi_1, \ldots, \phi_N)^T$ is an $(m + m_y) \times N$ matrix. Using equations 6 and 8, the MSE loss with parameters $W = \{W^0, \bar{\epsilon}\}$ becomes (SI A.3.1)

$$
\begin{aligned}
I[\Phi; W] &= \int_G dg \mathcal{L}[\Phi; W] = \int_G dg \left\| W^0 \left[I + \bar{\epsilon}^i [\hat{L}_i]^\alpha \partial_\alpha \right] \Phi(g) \right\|^2 \\
&= \int_{\mathcal{S}} \frac{d^n x}{\left| \frac{\partial x}{\partial g} \right|} \left[\Phi^T \mathbf{m}_2 \Phi + \partial_\alpha \Phi^T \mathbf{h}^{\alpha\beta} \partial_\beta \Phi + [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T \mathbf{v}^i \Phi \right) \right]
\end{aligned} \tag{18}
$$

where $\left| \frac{\partial x}{\partial g} \right|$ is the determinant of the Jacobian, $W^i = W^0 \bar{\epsilon}^i$ and

$$
\mathbf{m}_2 = W^{0T} W^0, \qquad \mathbf{h}^{\alpha\beta}(\boldsymbol{x}) = \bar{\epsilon}^{iT} \mathbf{m}_2 \bar{\epsilon}^j [\hat{L}_i]^\alpha [\hat{L}_j]^\beta, \qquad \mathbf{v}^i = \mathbf{m}_2 \bar{\epsilon}^i. \tag{19}
$$

Note that $\mathbf{h}$ has feature space indices via $[\bar{\epsilon}^{iT} \mathbf{m}_2 \bar{\epsilon}^j]_{ab}$, with index symmetry $\mathbf{h}^{\alpha\beta}_{ab} = \mathbf{h}^{\beta\alpha}_{ba}$. When $\mathcal{F} = \mathbb{R}$ (i.e. f is a 1D scalar), $\mathbf{h}^{\alpha\beta}$ becomes a a Riemannian metric for $\mathcal{S}$. In general $\mathbf{h}$ combines a 2-tensor $\mathbf{h}_{ab} = \mathbf{h}^{\alpha\beta}_{ab} \partial_\alpha \partial_\beta \in T\mathcal{S} \otimes T\mathcal{S}$ with an inner product $h^T \mathbf{h}^{\alpha\beta} f$ on the feature space $\mathcal{F}$.

A simpler version of the loss function in equation 18 is commonly used in field theories in physics (Polyakov, 2018). There too, the motivation is to preserve spatial symmetries for the metric $\mathbf{h}$. In equation 18, $\mathbf{h}$ transforms equivariantly as a 2-tensor $v \cdot \mathbf{h}^{\alpha\beta} = [v^{-1}]^\alpha_\rho [v^{-1}]^\beta_\gamma \mathbf{h}^{\rho\gamma}(\boldsymbol{x})$ for $v \in G$ (SI A.3). The last term in equation 18 vanishes for many groups (SI A.3) and it is also absent in physics. We discuss the implications of this relation between physics and equivariant neural networks below.

Relation to physics Although equation 18 was derived purely using ML and equivariance, physicists will recognize equation 18 as a generalization of the non-interacting field theory action (integral of a Lagrangian). In field theory, $\phi(x)$ (each input feature $[f|y]$) is called a field. $\mathbf{m}_2$ is the "mass matrix" whose eigenvalues are the squares of masses of the fields. The second term $\partial\phi^T \cdot \mathbf{h} \cdot \partial\phi = \partial^\mu \phi^a \partial_\mu \phi_a$ is the "kinetic" term. In physics, the loss in equation 18 is used in two ways. When the parameters $\mathbf{m}_2$ and $\mathbf{h}$ are known, a variational procedure known as the Euler-Lagrange equations is used to find optimal $\phi(\boldsymbol{x}) = \arg\min_\phi I[\phi]$. When $\mathbf{m}_2$ and $\mathbf{h}$ are not known, ϕ_n are observational data and physicists use equation 18 the same way as ML. In this case equation 18 is a variational ansatz used to identify parameters $\mathbf{m}_2$ and $\mathbf{h}$ such that the expected loss for

the observed ϕ_n is minimized. In physics the loss is minimized around real data points ϕ_n. This idea yields a natural way to probe robustness in equivariant neural networks, discussed next.

Robustness to noise and attacks Equivariant neural networks are expected to be more robust than others. To check this we can quantify how the network would perform for an input $\phi' = \phi + \delta\phi$ which adds a small perturbation $\delta\phi$ to a real data point ϕ. Note $\delta\phi$ here s a random noise, not an adversarial attack. Robustness to such perturbation would mean that, for optimal parameters W^* , the loss function would not change, i.e. $I[\phi'; W^*] = I[\phi; W^*]$, requiring I to be minimized around real data points ϕ. This can be cast as a variational equation $\delta I[\phi; W^*] = 0$, which yield the familiar Euler-Lagrange (EL) equation (SI A.4), widely used in physics and control theory. Thus, requiring good generalization, i.e. $\delta I[\phi; W^*]/\delta\phi = 0$ means for optimal parameters W^*, real data points ϕ satisfy the EL equations. Writing $I[\phi; W] = \int d^n x \mathcal{L}[\phi; W]$, we have

$$\text{Robustness to random noise} \Longleftrightarrow \text{EL:} \quad \frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} = 0 \tag{20}$$

where the partial derivative terms appear because of the L-conv layer. Note that equation 20 only needs to be satisfied for random noise and adversarial attacks may violate it. This could serve as a way to identify such attacks.

Conservation laws Conserved currents, via Noether's theorem provide a way to find hidden symmetries (see also Kunin et al. (2020)). The idea is that the equivariance condition equation 2 can be written for the integrand of the loss, $\mathcal{L}[\phi, W]$. If we write the equivariance equation for infinitesimal v_ϵ, we obtain a vector field which is divergence free. Since G is the symmetry of the system, transforming an input $\phi \to w \cdot \phi$ by $w \in G$ the integrand should change equivariantly, meaning $\mathcal{L}[w \cdot \phi] = w \cdot \mathcal{L}[\phi]$. When robustness error is minimized as in equation 20, an infinitesimal $w \approx I + \eta^i L_i$, with $\delta\phi = \epsilon^i \hat{L}_i \phi$, results in a conserved current (SI A.4)

$$\text{Noether current: } J^\alpha = \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b - \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\alpha} \delta \boldsymbol{x}^\alpha, \qquad \delta I[\phi; W^*] = 0 \quad \Rightarrow \partial_\alpha J^\alpha = 0 \tag{21}$$

J, called a Noether current, captures the change of the Lagrangian $\mathcal{L}$ along symmetry direction $\hat{L}_i$. The fact the divergence of J vanishes can provide an alternative means of discovering symmetry generators L_i by minimizing $\|\partial_\alpha J^\alpha\|$. For an equivariant neural net, we find that (SI A.5) the Noether current is the "stress-energy" tensor in physics (Landau, 2013). It is the Noether current associated with space (or space-time) variations $\delta \boldsymbol{x}$. It appears here because G acts on the space, as opposed to acting on feature dimensions. We will discuss the implication of this in the discussions.

6 Discussions

L-conv is essentially the infinitesimal version of G-conv. L-conv has a much simpler and more familiar structure than G-conv. L-conv layers do not require encoding irreps or discretizing the group, and can be combined to approximate any feedforward equivariant architecture. Additionally, L-conv's universal and simple structure allows us to discover symmetries from data. It is easy to implement, with a structure similar to GCN. But perhaps the most intriguing property of L-conv is its connection with symmetry encoding models in physics. This is significant for two reasons. First, methods for dealing with symmetry in physics acquire important meanings in ML, as our discussion on the loss function, robustness and conservation laws show. The second reason is for scientific machine learning. Physicists generally use simple polynomial forms for the Lagrangian (loss). These "perturbative" Lagrangian lead to divergences in quantum field theory and it is believed the true Lagrangian is more complicated. Hence, more complex L-conv based architectures can potentially provide more advanced ansatze for physics problems. Also, for machine learning, we discussed how Noether's theorem can provide a new way to discover symmetries. We also found that the current equivariance formulation is linked to the conservation of the so-called stress-energy tensor. In physics, there are many other types of conserved charges (including electric charge) which are not the stress- energy tensor because the underlying symmetry is not a spatial symmetry. This also reveals that the way equivariance is currently defined is much less general than how it is used in physics. This could lead to defining more general equivariant architectures in machine learning.

Acknowledgments and Disclosure of Funding

R. Walters is supported by a Postdoctoral Fellowship from the Roux Institute and NSF grants #2107256 and #2134178. This work was supported in part by the U. S. Army Research Office under Grant W911NF-20-1-0334, DOE ASCR 2493 and NSF Grant #2134274.

References

Anselmi, F., Evangelopoulos, G., Rosasco, L., and Poggio, T. Symmetry-adapted representation learning. *Pattern Recognition*, 86:201–208, 2019.

Benton, G., Finzi, M., Izmailov, P., and Wilson, A. G. Learning invariances in neural networks. *arXiv preprint arXiv:2010.11882*, 2020.

Bogatskiy, A., Anderson, B., Offermann, J., Roussi, M., Miller, D., and Kondor, R. Lorentz group equivariant neural network for particle physics. In *International Conference on Machine Learning*, pp. 992–1002. PMLR, 2020.

Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

Cohen, T. and Welling, M. Learning the irreducible representations of commutative lie groups. In *International Conference on Machine Learning*, pp. 1755–1763, 2014.

Cohen, T. and Welling, M. Group equivariant convolutional networks. In *International conference on machine learning*, pp. 2990–2999, 2016a.

Cohen, T., Weiler, M., Kicanaoglu, B., and Welling, M. Gauge equivariant convolutional networks and the icosahedral cnn. In *International Conference on Machine Learning*, pp. 1321–1330. PMLR, 2019a.

Cohen, T. S. and Welling, M. Steerable cnns. 2016b.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. Spherical cnns. In *International Conference on Learning Representations*, 2018.

Cohen, T. S., Geiger, M., and Weiler, M. A general theory of equivariant cnns on homogeneous spaces. In *Advances in Neural Information Processing Systems*, pp. 9142–9153, 2019b.

Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Finzi, M., Stanton, S., Izmailov, P., and Wilson, A. G. Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. *arXiv preprint arXiv:2002.12880*, 2020.

Finzi, M., Welling, M., and Wilson, A. G. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. *arXiv preprint arXiv:2104.09459*, 2021.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272. PMLR, 2017.

Hall, B. *Lie groups, Lie algebras, and representations: an elementary introduction*, volume 222. Springer, 2015.

Hartford, J., Graham, D., Leyton-Brown, K., and Ravanbakhsh, S. Deep models of interactions across sets. In *International Conference on Machine Learning*, pp. 1909–1918. PMLR, 2018.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Kondor, R. and Trivedi, S. On the generalization of equivariance and convolution in neural networks to the action of compact groups. *arXiv preprint arXiv:1802.03690*, 2018.

Kondor, R., Lin, Z., and Trivedi, S. Clebsch–gordan nets: a fully fourier space spherical convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 10117–10126, 2018.

Kunin, D., Sagastuy-Brena, J., Ganguli, S., Yamins, D. L., and Tanaka, H. Neural mechanics: Symmetry and broken conservation laws in deep learning dynamics. *arXiv preprint arXiv:2012.04728*, 2020.

Landau, L. D. *The classical theory of fields*, volume 2. Elsevier, 2013.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, J. M., Chow, B., Chu, S.-C., Glickenstein, D., Guenther, C., Isenberg, J., Ivey, T., Knopf, D., Lu, P., Luo, F., et al. Manifolds and differential geometry. *Topology*, 643:658, 2009.

Polyakov, A. M. *Gauge fields and strings*. Routledge, 2018.

Rao, R. P. and Ruderman, D. L. Learning lie groups for invariant visual perception. In *Advances in neural information processing systems*, pp. 810–816, 1999.

Ravanbakhsh, S. Universal equivariant multilayer perceptrons. *arXiv preprint arXiv:2002.02912*, 2020.

Ravanbakhsh, S., Schneider, J., and Poczos, B. Equivariance through parameter-sharing. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2892–2901. JMLR. org, 2017.

Schaub, M. T., Benson, A. R., Horn, P., Lippner, G., and Jadbabaie, A. Random walks on simplicial complexes and the normalized hodge 1-laplacian. *SIAM Review*, 62(2):353–391, 2020.

Sohl-Dickstein, J., Wang, C. M., and Olshausen, B. A. An unsupervised algorithm for learning lie group transformations. *arXiv preprint arXiv:1001.1027*, 2010.

Wang, R., Walters, R., and Yu, R. Incorporating symmetry into deep dynamics models for improved generalization. In *International Conference on Learning Representations*, 2020.

Weiler, M. and Cesa, G. General e (2)-equivariant steerable cnns. In *Advances in Neural Information Processing Systems*, pp. 14334–14345, 2019.

Weiler, M., Geiger, M., Welling, M., Boomsma, W., and Cohen, T. S. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, pp. 10381–10392, 2018a.

Weiler, M., Hamprecht, F. A., and Storath, M. Learning steerable filters for rotation equivariant cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 849–858, 2018b.

Weinberg, S. *The quantum theory of fields*, volume 3. Cambridge university press, 1995.

Whitaker, E. On the functions which are represented by the expansion of interpolating theory. In *Proc. Roy. Soc. Edinburgh*, volume 35, pp. 181–194, 1915.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5028–5037, 2017.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Zhou, A., Knowles, T., and Finn, C. Meta-learning symmetries by reparameterization. *arXiv preprint arXiv:2007.02933*, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

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? [No]
   (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)? [Yes]
   (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? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (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]

Supplementary Information

A Extended derivations and proofs

Path-ordered Exponential Every element $g \in G$ can be written as a product $g = \prod_a \exp[t_a^i L_i]$ using the matrix exponential (Hall, 2015). This can be done using a path γ connecting I to g on the manifold of G. Here, t_a will be segments of the path γ which add up as vectors to connect I to g. This surjective map can be written as a "path-ordered" (or time-ordered in physics (Weinberg, 1995)) exponential (POE). In the simplest form, POE can be defined by breaking $u = \prod_a \exp[t_a^i L_i]$ down into infinitesimal steps of size $t_a = 1/N$ with $N \to \infty$. Choosing γ to be a differentiable path, we can replace the sum over segments $\sum_a t_a$ with an integral along the path $\sum_a t_a = \int_\gamma dst(s)ds$, where $t(s) = d\gamma/ds$ is the tangent vector to the path γ, where $s \in [0, 1]$ parametrizes γ The POE is then defined as the infinitesimal t_a limit of $g = \prod_a \exp[t_a^i L_i]$. This can be written as

$$
\begin{aligned}
g = P\exp\left[\int_\gamma t^i(s)L_i ds\right] &= \lim_{N\to\infty}\prod_{a=1}^{N}\left(I + \delta s \gamma'^i(s_a)L_i\right) \\
&= \int_0^{s_1} ds_0 \gamma'^i(s_0)L_i \int_0^{s_2} ds_1 \gamma'^j(s_1)L_j \cdots \int_0^1 ds_N \gamma'^k(s_N)L_k
\end{aligned} \tag{22}
$$

L-conv derivation Let us consider what happens if the kernel in G-conv equation 3 is localized near identity. Let $\kappa_I(u) = c\delta_\eta(u)$, with constants $c \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$ and kernel $\delta_\eta(u) \in \mathbb{R}$ which has support only on on an η neighborhood of identity, meaning $\delta_\eta(I + \epsilon^i L_i) \to 0$ if $|\epsilon| > \eta$. This allows us to expand G-conv in the Lie algebra of G to linear order. With $v_\epsilon = I + \epsilon^i L_i$, we have

$$
\begin{aligned}
[\delta_\eta \star f](g) &= \int_G dv \delta_\eta(v) f(gv) = \int_{\|\epsilon\|<\eta} dv_\epsilon \delta_\eta(v_\epsilon) f(gv_\epsilon) \\
&= \int d\epsilon \delta_\eta(I + \epsilon^i L_i) f(g(I + \epsilon^i L_i)) \\
&= \int d\epsilon \delta_\eta(I + \epsilon^i L_i) f(g + \epsilon^i g L_i) \\
&= \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\left[f(g) + \epsilon^i g L_i \cdot \frac{d}{dg} f(g) + O(\epsilon^2)\right] \\
&= \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\left[I + \epsilon^i g L_i \cdot \frac{d}{dg}\right] f(g) + O(\eta^2) \\
&= W^0\left[I + \bar{\epsilon}^i g L_i \cdot \frac{d}{dg}\right] f(g) + O(\eta^2)
\end{aligned} \tag{23}
$$

where $d\epsilon$ is the integration measure on the Lie algebra induced by the Haar measure dv on G. The $O(\eta^2)$ term arises from integrating the $O(\epsilon^2)$ terms. To see this, note that for an order p function $\phi(\epsilon)$, for $|\epsilon| < \eta$, $|\phi(\epsilon)| < \eta^p C$ (for some constant C). Substituting this bound into the integral over the kernel δ_η we get

$$
\begin{aligned}
&\left|\int d\epsilon \delta_\eta(I + \epsilon^i L_i)\phi(\epsilon)\right| \leq \int \left|d\epsilon \delta_\eta(I + \epsilon^i L_i)\phi(\epsilon)\right| \\
&< \int \left|d\epsilon \delta_\eta(I + \epsilon^i L_i)\eta^p C\right| \leq \eta^p C \int \left|d\epsilon \delta_\eta(I + \epsilon^i L_i)\right| \leq \eta^p C
\end{aligned} \tag{24}
$$

In matrix representations, $gL_i \cdot \frac{df}{dg} = [gL_i]^\beta_\alpha \frac{df}{dg^\beta_\alpha} = \mathrm{Tr}\left[[gL_i]^T \frac{df}{dg}\right]$. Note that in $g(I + \epsilon^i L_i)\boldsymbol{x}_0$, the $gL_i\boldsymbol{x}_0 = \hat{L}_i(g)\boldsymbol{x}$ come from the pushforward $\hat{L}_i(g) = gL_i g^{-1} \in T_g G$. Here

$$
W^0 = c\int d\epsilon \delta_\eta(I + \epsilon^i L_i) \in \mathbb{R}^{m'} \otimes \mathbb{R}^m, \qquad \bar{\epsilon}^i = \frac{\int d\epsilon \delta_\eta(I + \epsilon^i L_i)\epsilon^i}{\int d\epsilon \delta_\eta(I + \epsilon^i L_i)} \in \mathbb{R}^m \otimes \mathbb{R}^m \tag{25}
$$

with $\|\bar{\epsilon}\| < \eta$. When δ_η is normalized, meaning $\int_G \delta_\eta(g)dg = 1$, we have $W^0 = c$ and

$$\bar{\epsilon}^i = \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\epsilon^i$$

Note that with $f(g) \in \mathbb{R}^m$, each $\epsilon^i \in \mathbb{R}^m \otimes \mathbb{R}^m$ is a matrix. With indices, $f(gv_\epsilon)$ is given by

$$[f(gv_\epsilon)]^a = \sum_b f^b(g(\delta^a_b + [\epsilon^i]^a_b L_i)) \tag{26}$$

Similarly, the integration measure $d\epsilon$, which is induced by the Haar measure $dv_\epsilon \equiv d\mu(v_\epsilon)$, is a product $\int d\epsilon = \int |J| \prod d[\epsilon^i]^a_b$, with $J = \partial v_\epsilon / \partial \epsilon$ being the Jacobian.

Equation 23 is the core of the architecture we are proposing, the Lie algebra convolution or **L-conv**.

L-conv Layer In general, we define Lie algebra convolution (L-conv) as follows

$$\begin{aligned} Q[f](g) &= W^0 \left[I + \bar{\epsilon}^i g L_i \cdot \frac{d}{dg} \right] f(g) \\ &= [W^0]_b f^a \left(g \left(\delta^b_a + [\bar{\epsilon}^i]^b_a L_i \right) \right) + O(\bar{\epsilon}^2) \end{aligned} \tag{27}$$

Extended equivariance for L-conv From equation 27 we see that W^0 acts on the output feature indices. Notice that the equivariance of L-conv is due to the way $gv_\epsilon = g(I + \bar{\epsilon}^i L_i)$ appears in the argument, since for $u \in G$

$$u \cdot Q[f](g) = W^0 f(u^{-1} g v_\epsilon) = W^0 [u \cdot f](gv_\epsilon) \tag{28}$$

Because of this, replacing W^0 with a general neural network which acts on the feature indices separately will not affect equivariance. For instance, if we pass L-conv through a neural network to obtain a generalized L-conv Q_σ, we have

$$\begin{aligned} Q_\sigma[f](g) &= \sigma(W f(gv_\epsilon) + b) \\ u \cdot Q_\sigma[f](g) &= Q_\sigma[f](u^{-1}g) = \sigma(W f(u^{-1} g v_\epsilon) + b) \\ &= \sigma(W[u \cdot f](gv_\epsilon) + b) = Q_\sigma[u \cdot f](g) \end{aligned} \tag{29}$$

Thus, L-conv can be followed by any nonlinear neural network as long as it only acts on the feature indices (i.e. a in $f^a(g)$) and not on the spatial indices g in $f(g)$.

A.1 Approximating G-conv using L-conv

Lemma 1 (Approximating the kernel). *Let the kernel $\kappa : G \to \mathcal{F}' \otimes \mathcal{F}$ with $\int_G \kappa(g)dg < \infty$ be continuously differentiable with $\|d\kappa(g)/dg\|^2 < \xi^2$, and with compact support over $G_0 \subset G$. Let $\kappa_k(g) = c_k \delta_\eta(u_k^{-1} g)$ be a set of kernels with support on an η neighborhood of $I \in G$. Then, $\exists c_k \in \mathcal{F}' \otimes \mathcal{F}$ such that $\tilde{\kappa}$ approximates κ, meaning $\int_G \|\kappa(g) - \tilde{\kappa}(g)\|^2 dg < \zeta^2$ for arbitrary small $\zeta \in \mathbb{R}_+$.*

Proof: The intuition is similar to the universal approximation theorem for neural networks (Hornik et al., 1989; Cybenko, 1989), only generalized to a group manifold instead of $\mathbb{R}$. Let B_0 be the set of $v_\epsilon = I + \epsilon^i L_i \in \mathfrak{g}$, with $\|\epsilon\|^2 < \eta^2$. Choose a set of $u_k \in G$ such that the neighborhoods $B_k = u_k B_0 \subset G$, and such that $\bigcup_k B_k = G_0$. We can show that, for small enough η, the kernel does not change more than ξ over each B_k, allowing us to replace it with a constant localized kernel κ_k with support only on B_k. To see this, consider $g \in B_k$ and $v_\epsilon = I + \epsilon^i L_i \in \mathfrak{g}$, such that $v_\epsilon g \in B_k$. Let γ be a path connecting g to $v_\epsilon g$, via $u(s) = (I + s\epsilon^i L_i)g$, with $s \in [0, 1]$. Using the triangle inequality we have

$$\begin{aligned} \|\kappa(g) - \kappa(v_\epsilon g)\|^2 &= \left\| \int_\gamma ds \frac{d\kappa(u(s))}{ds} \right\|^2 \leq \int_\gamma ds \left\| \frac{d\kappa(u(s))}{ds} \right\|^2 \\ &< \int_0^1 ds \left\| \frac{dg}{ds} \right\|^2 \xi^2 = \|\epsilon\|^2 \xi^2 \end{aligned} \tag{30}$$

where $\|\epsilon\|^2 \leq \eta^2$ because $v_\epsilon g \in B_k$. This means that if we replace $\kappa(g)$ with $\kappa(u_k)$ for any $u_k \in B_k$, our error in approximating κ over B_k is less than $\eta^2\xi^2$. Setting $\kappa_k(g) = \kappa(u_k)\delta_\eta(u_k^{-1}g)$ with the localized kernels $\delta_\eta(g)$ being any continuously differentiable distribution such that $\int_{B_k} dg\delta_\eta(u_k^{-1}g) = 1$ on all B_k, we get

$$\int_G dg\|\kappa(g) - \tilde{\kappa}(g)\|^2 = \sum_k \int_{B_k} dg\|\kappa(g) - \kappa_k(g)\|^2 < \sum_k |B_k| \int_{B_k} dg\eta^2\xi^2 = |G_0|\eta^2\xi^2 \quad (31)$$

where $|G_0|$ is the volume of the support of κ □

Thus, we can approximate a large class of kernels as $\kappa(g) \approx \sum_k \kappa_k(g)$ where the local kernels $\kappa_k(g) = c_k\delta_\eta(u_k^{-1}g)$ have support only on an η neighborhood of $u_k \in G$. Here $c_k \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$ are constants and $\delta_\eta(u)$ is as in equation 6. Using this, G-conv equation 3 becomes

$$[\kappa \star f](g) = \sum_k c_k \int dv\delta_\eta(u_k^{-1}v)f(gv) = \sum_k c_k[\delta_\eta \star f](gu_k). \quad (32)$$

The kernels κ_k are localized around u_k, whereas in L-conv the kernel is around identity. We can compose L-conv layers to move κ_k from u_k to identity.

Lemma 2 (Moving kernels to identity). *The local kernel κ_k can be moved near identity using a multilayer L-conv.*

Proof: In equation 32, write $u_k = v_\epsilon u'_k$, with $v_\epsilon = I + \epsilon^i L_i \in \mathfrak{g}$. We have

$$\delta_\eta(u_k^{-1}v) = \left[I + \epsilon^i gL_i \cdot \frac{d}{dg}\right]\delta_\eta(g)\Big|_{g \to u'^{-1}_k v} + O(\epsilon^2) \approx Q_\epsilon[\delta_\eta](u'^{-1}_k v) \quad (33)$$

where $Q_\epsilon = I - \epsilon^i \hat{L}_i$ is an L-conv layer with $W^0 = I$. This means that a applying one L-conv layer with the parameters above moves the localized kernel along v_ϵ on G. Iterating this further, write u_k as the product of a set of small group elements $u_k = \prod_{a=1}^p v_a$, with $v_a = I + \epsilon_a^i L_i \in \mathfrak{g}$. Defining L-conv layers $Q_a = I - \epsilon_a^i \hat{L}_i$, we can write

$$\kappa_k(g) \approx c_k Q_p \circ \cdots \circ Q_1 \circ \delta_\eta(g) \quad (34)$$

meaning κ_k localized around u_k can be written as a p layer L-conv acting on a kernel $\delta_\eta(g)$, localized around the identity of the group. With $\|\epsilon_a\| < \eta$, the error in u_k is $O(\eta^{p+1})$. □

Thus, we conclude that any G-conv equation 3 can be approximated by multilayer L-conv. We can take this result even further, following the main theorem in Kondor & Trivedi (2018), and show that any feedforward equivariant neural network can be approximated using multilayer L-conv with nonlinearities.

Equivariance of nonlinearity Point wise nonlinearities will keep an equivariant scalar function equivariant. To see this, let $\sigma : \mathbb{R} \to \mathbb{R}$, $f : \mathcal{S} \to \mathcal{F}$ be a scalar (i.e., $g \cdot f(\boldsymbol{x}) = f(g^{-1}\boldsymbol{x})$) and $Q : \mathcal{F} \to \mathcal{F}'$ be equivariant (i.e. $g \cdot Q[f] = Q[g \cdot f]$). Then, $\sigma(Q[f](\boldsymbol{x}))$ is still a scalar and for $g \in G$ we have

$$g \cdot \sigma \circ Q[f](\boldsymbol{x}) = \sigma \circ Q[f](g^{-1}\boldsymbol{x}) = \sigma \circ g \cdot Q[f](\boldsymbol{x}) \quad (35)$$

meaning, point-wise nonlinearity σ is equivariant for scalar $Q[f]$. Hence we have the corollary:

Corollary 2 (Feedforward NN)**.** *Assume G is compact and G acts on $\mathcal{S}$ transitively. Any equivariant feedforward neural network (FNN) can be approximated using multilayer L-conv with point-wise nonlinearities.*

Proof: This follows from the main theorem in Kondor & Trivedi (2018) and the our Theorem 1. A FNN is defined as $\sigma_p \circ F_p(\cdots(\sigma_1 \circ (F_1(f)(\boldsymbol{x})$ where F_k are linear and σ_k are point-wise nonlinearities. Any linear equivariant layer has to be G-conv, which can be approximate by multilayer L-conv. Therefore, multilayer L-conv with nonlinearity can approximate any equivariant FNN. □

As a final note, we should point out that to our knowledge it is not known whether *every equivariant function* can be approximated by equivariant FNN. Hence, the corollary above is *not* a universal approximation theorem for equivariant scalar functions in terms of L-conv.

Since many datasets such as images deal with discretized spaces, we first need to derive how L-conv acts on such data, discussed next.

A.2 Example of continuous L-conv

The $gL_i \cdot df/dg$ in equation 6 can be written in terms of partial derivatives $\partial_\alpha f(\boldsymbol{x}) = \partial f/\partial \boldsymbol{x}^\alpha$. In general, using $\boldsymbol{x}^\rho = g^\rho_\sigma \boldsymbol{x}_0^\sigma$, we have

$$\frac{df(g\boldsymbol{x}_0)}{dg^\alpha_\beta} = \frac{d(g^\rho_\sigma \boldsymbol{x}_0^\sigma)}{dg^\alpha_\beta}\partial_\rho f(\boldsymbol{x}) = \boldsymbol{x}_0^\beta \partial_\alpha f(\boldsymbol{x}) \tag{36}$$

$$gL_i \cdot \frac{df}{dg} = [gL_i]^\alpha_\beta \boldsymbol{x}_0^\beta \partial_\alpha f(\boldsymbol{x}) = [gL_i\boldsymbol{x}_0]\cdot\nabla f(\boldsymbol{x}) = \hat{L}_i f(\boldsymbol{x}) \tag{37}$$

Hence, for each L_i, the pushforward gL_i generates a flow on $\mathcal{S}$ through the vector field $\hat{L}_i \equiv gL_i \cdot d/dg = [gL_i\boldsymbol{x}_0]^\alpha\partial_\alpha$ (Fig. 1). Being a vector field $\hat{L}_i \in T\mathcal{S}$ (i.e. 1-tensor), $\hat{L}_i$ is basis independent, meaning for $v \in G$, $\hat{L}_i(v\boldsymbol{x}) = \hat{L}_i$. Its components transform as $[\hat{L}_i(v\boldsymbol{x})]^\alpha = [vgL_i\boldsymbol{x}_0]^\alpha = v^\alpha_\beta \hat{L}_i(\boldsymbol{x})^\beta$, while the partial transforms as $\partial/\partial[v\boldsymbol{x}]^\alpha = [v^{-1}]^\gamma_\alpha \partial_\gamma$. Using this relation and Taylor expanding equation 10, we obtain a second form for the group action on L-conv. For $w \in G$, with $\boldsymbol{y} = w^{-1}\boldsymbol{x}$ we have

$$Q[f](w^{-1}g\boldsymbol{x}_0) = W^0\left[I + \bar{\epsilon}^i[\hat{L}_i]^\alpha[w^{-1}]^\beta_\alpha \frac{\partial}{\partial \boldsymbol{y}^\beta}\right] f(\boldsymbol{y})\big|_{\boldsymbol{y}\to w^{-1}\boldsymbol{x}} \tag{38}$$

1D Translation: Let $G = T_1 = (\mathbb{R}, +)$. A matrix representation for G is found by encoding x as a a 2D vector $(x, 1)$. The lift is given by $\boldsymbol{x}_0 = (0, 1)$ as the origin and $g = \begin{pmatrix} 1 & x \\ 0 & 1 \end{pmatrix}$. The Lie algebra basis is $L = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$. It is easy to check that $gg'\boldsymbol{x}_0 = (x + x', 1)$. We also find $gL = L$, meaning L looks the same in all T_gG. Close to identity $I = 0$, $v_\epsilon = I + \epsilon L = \epsilon$. We have $gv_\epsilon\boldsymbol{x}_0 = (g + \epsilon gL)x_0 = (x + \epsilon, 1)$. Thus, $f(g(I + \epsilon L)\boldsymbol{x}_0) \approx f(\boldsymbol{x}) + \epsilon df(\boldsymbol{x})/d\boldsymbol{x}$. This readily generalizes to nD translations T_n (SI A.2.2), yielding $f(\boldsymbol{x}) + \epsilon^\alpha\partial_\alpha f(\boldsymbol{x})$.

2D Rotation: Let $G = SO(2)$. The space which $SO(2)$ can lift is not the full $\mathbb{R}^2$, but a circle of fixed radius $r = \sqrt{x^2 + y^2}$. Hence we choose $\mathcal{S} = S^1$ embedded in $\mathbb{R}^2$, with $x = r\cos\theta$ and $y = r\sin\theta$. For the lift, we use the standard 2D representation. We have $\boldsymbol{x}_0 = (r, 0)$ and (see SI A.2.1)

$$L = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \qquad g = \exp[\theta L] = \frac{1}{r}\begin{pmatrix} x & -y \\ y & x \end{pmatrix}, \qquad gL \cdot \frac{df}{dg} = (x\partial_y - y\partial_x)\, f. \tag{39}$$

Physicists will recognize $\hat{L} \equiv (x\partial_y - y\partial_x) = \partial_\theta$ as the angular momentum operator in quantum mechanics and field theories, which generates rotations around the z axis.

Rotation and scaling Let $G = SO(2) \times \mathbb{R}^+$, where the $\mathbb{R}^+ = [0, \infty)$ is scaling. The infinitesimal generator for scaling is identity $L_2 = I$. This group is also Abelian, meaning $[L_2, L] = 0$ ($L \in so(2)$ equation 39). $\mathbb{R}^2/0$ can be lifted to G by choosing $\boldsymbol{x}_0 = (1, 0)$ in polar coordinates and $\boldsymbol{x} = g\boldsymbol{x}_0 = rL_2\exp[\theta L]\boldsymbol{x}_0$. We again have $gL \cdot df/dg = \partial_\theta f$. We also have $gL_2 = g$, so $gL_2\boldsymbol{x}_0 = (x, y)$ and from equation 37, $gL_2 \cdot df/dg = (x\partial_x + y\partial_y)f = r\partial_r f$, which is the scaling operation.

A.2.1 Rotation $SO(2)$

With $\boldsymbol{x}_0 = (r, 0)$

$$g = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, = \frac{1}{r}\begin{pmatrix} x & -y \\ y & x \end{pmatrix}, \quad L = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad gL = \frac{1}{r}\begin{pmatrix} -y & -x \\ x & -y \end{pmatrix} \tag{40}$$

$$gL\boldsymbol{x}_0 = \begin{pmatrix} -y \\ x \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix} \tag{41}$$

To calculate df/dg we note that even after the lift, the function f was defined on $\mathcal{S}$. So we must include the $\boldsymbol{x}_0$ in $f(g\boldsymbol{x}_0)$. Using equation 8, we have

$$\frac{df(g\boldsymbol{x}_0)}{dg} = \frac{1}{r}\boldsymbol{x}_0^T \begin{pmatrix} \partial_x f & -\partial_y f \\ \partial_y f & \partial_x f \end{pmatrix} = \begin{pmatrix} \partial_x f & -\partial_y f \\ 0 & 0 \end{pmatrix}$$
$$gL \cdot \frac{df}{dg} = \mathrm{Tr}\begin{pmatrix} x\partial_y f - y\partial_x f & x\partial_x f + y\partial_y f \\ 0 & 0 \end{pmatrix} = (x\partial_y f - y\partial_x f) \tag{42}$$

A.2.2 Translations T_n

Generalizing the T_1 case, we add a dummy dimension 0 and $\boldsymbol{x}_0 = (1, 0, \ldots, 0)$. The generators are $[L_i]^\nu_\mu = \delta_{i\mu}\delta^\nu_0$ and $g = I + x^i L_i$. Again, $gL_i = L_i + x^j L_j L_i = L_i$ as $L_j L_i = 0$ for all i, j. Hence, $[\hat{L}_i]^\alpha = [gL_i\boldsymbol{x}_0]^\alpha = \delta^\alpha_i$.

A.3 Group invariant loss

Because G is the symmetry group, f and $g \cdot f$ should result in the same optimal parameters. Hence, the minima of the loss function need to be *group invariant*. One way to satisfy this is for the loss itself to be group invariant, which can be constructed by integrating over G (global pooling (Bronstein et al., 2021)). A function $I = \int_G dgF(g)$ is G-invariant as for $w \in G$

$$w \cdot I = \int_G w \cdot F(g)dg = \int F(w^{-1}g)dg = \int_G F(g')d(wg') = \int_G F(g')dg' \tag{43}$$

where we used the invariance of the Haar measure $d(wg') = dg'$. We can change the integration to $\int_{\mathcal{S}} d^n x$ by change of variable $dg/d\boldsymbol{x}$. Since we need $\mathcal{S}$ to be lifted to G, the lift: $\mathcal{S} \to G$ is injective, the a map $G \to \mathcal{S}$ need not be. $\mathcal{S}$ is homeomorphic to G/H, where $H \subset G$ is the stabilizer of the origin, i.e. $h\boldsymbol{x}_0 = \boldsymbol{x}_0, \forall h \in H$. Since $F(g\boldsymbol{x}_0) = F(gh\boldsymbol{x}_0)$, we have

$$I = \int_G F(g)dg = \int_H dh \int_{G/H} dg' F(g') = V_H \int_{G/H} dg' F(g') \tag{44}$$

Since $G/H \sim \mathcal{S}$, the volume forms $dg' = V_H d^n x$ can be matched for some parametrization.

A.3.1 MSE Loss

The MSE is given by $I = \sum_n \int_G dg \|Q[f_n](g)\|^2$, where f_n are data samples and $Q[f]$ is L-conv or another G-equivariant function. In supervised learning the input is a pair f_n, y_n. G can also act on the labels y_n. We assme that y_n are either also scalar features $y_n : \mathcal{S} \to \mathbb{R}^{m_y}$ with a group action $g \cdot y_n(\boldsymbol{x}) = y_n(g^{-1}\boldsymbol{x})$ (e.g. f_n and y_n are both images), or that y_n are categorical. In the latter case $g \cdot y_n = y_n$ because the only representations of a continuous G on a discrete set are constant. We can concatenate the inputs to $\phi_n \equiv [f_n|y_n]$ with a well-defined G action $g \cdot \phi_n = [g \cdot f_n|g \cdot y_n]$. The collection of combined inputs $\Phi = (\phi_1, \ldots, \phi_N)^T$ is an $(m + m_y) \times N$ matrix. Using equations 6 and 8, the MSE loss with parameters $W = \{W^0, \overline{\epsilon}\}$ becomes

$$\begin{aligned} I[\Phi; W] &= \int_G dg\mathcal{L}[\Phi; W] = \int_G dg \left\| W^0 \left[I + \overline{\epsilon}^i [\hat{L}_i]^\alpha \partial_\alpha \right] \Phi(g) \right\|^2 \\ &= 2\int_G dg \left[\|W^0\Phi\|^2 + \left\| W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi \right\|^2 + 2\Phi^T W^{0T} W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi \right] \qquad (45) \\ &= \int_{\mathcal{S}} \frac{d^n x}{\left|\frac{\partial x}{\partial g}\right|} \left[\Phi^T \mathbf{m}_2 \Phi + \partial_\alpha \Phi^T \mathbf{h}^{\alpha\beta} \partial_\beta \Phi + [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T \mathbf{v}^i \Phi \right) \right] \qquad (46) \end{aligned}$$

where $\left|\frac{\partial x}{\partial g}\right|$ is the determinant of the Jacobian, $W^i = W^0 \overline{\epsilon}^i$ and

$$\mathbf{m}_2 = W^{0T} W^0, \qquad \mathbf{h}^{\alpha\beta}(\boldsymbol{x}) = \overline{\epsilon}^{iT} \mathbf{m}_2 \overline{\epsilon}^j [\hat{L}_i]^\alpha [\hat{L}_j]^\beta, \qquad \mathbf{v}^i = \mathbf{m}_2 \overline{\epsilon}^i. \tag{47}$$

From equation 45 to 46 we used the fact that W^0 and W^i do not depend on $\boldsymbol{x}$ (or g) to write

$$2\Phi^T W^{0T} W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi = [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T W^{0T} W^i \Phi \right) = [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T \mathbf{m}_2 \overline{\epsilon}^i \Phi \right) \tag{48}$$

Note that $\mathbf{h}$ has feature space indices via $[\overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j]_{ab}$, with index symmetry $\mathbf{h}^{\alpha\beta}_{ab} = \mathbf{h}^{\beta\alpha}_{ba}$. When $\mathcal{F} = \mathbb{R}$ (i.e. f is a 1D scalar), $\mathbf{h}^{\alpha\beta}$ becomes a a Riemannian metric for $\mathcal{S}$. In general $\mathbf{h}$ combines a 2-tensor $\mathbf{h}_{ab} = \mathbf{h}^{\alpha\beta}_{ab}\partial_\alpha\partial_\beta \in T\mathcal{S}\otimes T\mathcal{S}$ with an inner product $h^T\mathbf{h}^{\alpha\beta}f$ on the feature space $\mathcal{F}$. Hence $\mathbf{h} \in T\mathcal{S}\otimes T\mathcal{S}\otimes\mathcal{F}^*\otimes\mathcal{F}^*$ is a $(2,2)$-tensor, with $\mathcal{F}^*$ being the dual space of $\mathcal{F}$.

Loss invariant metric transformation The metric $\mathbf{h}$ transforms equivariantly as a 2-tensor. As discussed under equation 8, $[\hat{L}_i(v\boldsymbol{x})]^\alpha = v^\alpha_\beta\hat{L}_i(\boldsymbol{x})^\beta$ and

$$v\cdot\mathbf{h}^{\alpha\beta} = \mathbf{h}^{\alpha\beta}(v^{-1}\boldsymbol{x}) = [v^{-1}]^\alpha_\rho[v^{-1}]^\beta_\gamma\mathbf{h}^{\rho\gamma}(\boldsymbol{x}), \qquad (v\in G). \tag{49}$$

Note that $v\cdot\mathbf{m}_2 = \mathbf{m}_2$ since f_n and y_n are scalars. For example, let $G = SO(2)$ and $R(\xi)\in SO(2)$ be rotation by angle ξ. Since there is only one $L_i = L$, the metric factorizes to

$$\mathbf{h}^{\alpha\beta}_{ab} = [\overline{\epsilon}^T\mathbf{m}_2\overline{\epsilon}]_{ab}\otimes[\hat{L}\hat{L}^T]^{\alpha\beta} \tag{50}$$

To find $R(\xi)\cdot\mathbf{h}$ we only need to calculate $R(\xi)^{-1}\hat{L}$. With $g = R(\theta)$, we have $\hat{L}(\boldsymbol{x}) = R(\theta)L\boldsymbol{x}_0 = (-y,x) = r(-\sin\theta,\cos\theta)$ from equation 41. Therefore, $R(\xi)^{-1}\hat{L} = R(\theta-\xi) = \hat{L}(R(\xi^{-1}\boldsymbol{x})$. Using equation 19 in equation 49, the transformed metric becomes

$$R(\xi)\cdot\mathbf{h}^{\alpha\beta}(R(\theta)\boldsymbol{x}_0) = \overline{\epsilon}^T\mathbf{m}_2\overline{\epsilon}\otimes[R(-\xi)\hat{L}]^\alpha[R(-\xi)\hat{L}]^\beta = \mathbf{h}^{\alpha\beta}(R(\theta-\xi)\boldsymbol{x}_0), \tag{51}$$

A.3.2 Third term as a boundary term

Since terms in equation 18 are scalars, they can be evaluated in any basis. If $\mathcal{S}$ can be lifted to multiple Lie groups, either group can be used to evaluate equation 18. For example $\mathbb{R}^n/0$ can be lifted to both T_n and $SO(n)\times\mathbb{R}_+$. For the translation group $G = T_n$ we have $gL_i = L_i$ and $[\hat{L}_i]^\alpha = \delta^\alpha_i$ (SI A.2.2) and $|\partial g/\partial x| = 1$ and $dg = d^nx$. Thus, the last term in equation 18 simplifies to a complete divergence $\int d^nx\partial_i(\Phi^T\mathbf{v}^i\Phi)$. Using the generalized Stoke's theorem $\int_{\mathcal{S}} dw = \int_{\partial\mathcal{S}} w$, the last term in equation 18 becomes a boundary term. When $\mathcal{S}$ is non-compact, the last term is $I_\partial = \int_{\partial\mathcal{S}} d\Sigma_i\Phi^T\mathbf{v}^i\Phi$, where $d\Sigma_i$ is the normal times the volume form of the $(n-1)$D boundary $\partial\mathcal{S}$ and is in the radial direction (e.g. for $\mathcal{S} = \mathbb{R}^n$ the boundary is a hyper-sphere $\partial\mathcal{S} = S^{n-1}$). Generally we expect the features ϕ_n to be concentrated in a finite region of the space and that they go to zero as $r\to\infty$ (if they don't the loss term $\Phi^T\mathrm{m}_2\Phi$ will diverge). Thus, the last term in equation 18 generally becomes a vanishing boundary term and does not matter.

A.3.3 MSE Loss for translation group T_n

We have $gL_i = L_i$ and $[\hat{L}_i]^\alpha = \delta^\alpha_i$ (SI A.2.2), the last term in equation 18 becomes a complete divergence $I_\partial = \int d^nx\partial_i(\Phi^T\mathbf{v}^i\Phi)$. Using the generalized Stoke's theorem $\int_{\mathcal{S}} dw = \int_{\partial\mathcal{S}} w$, when $\mathcal{S}$ is non-compact, $I_\partial = \int_{\partial\mathcal{S}} d\Sigma_i\Phi^T\mathbf{v}^i\Phi$. Here $d\Sigma_i$ is the normal times the volume form of the $(n-1)$D boundary $\partial\mathcal{S}$ (e.g. for $\mathcal{S} = \mathbb{R}^n$ the boundary is a hyper-sphere $\partial\mathcal{S} = S^{n-1}$). Generally we expect the features ϕ_n to be concentrated in a finite region of the space and that they go to zero as $r\to\infty$ (if they don't the loss term $\Phi^T\mathrm{m}_2\Phi$ will diverge). Thus, the last term in equation 18 generally becomes a vanishing boundary term and does not matter.

Next, the second term in equation 18 can be worked out as

$$\hat{L}^\alpha_i\partial_\alpha\phi^T\overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j\hat{L}^\beta_j\partial_\beta\phi = \partial_j\phi^T\overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j\partial_i\phi = \partial_j\phi^T\mathbf{h}^{ji}\partial_i\phi \tag{52}$$

where $\mathbf{h}^{ji} = \overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j$ is a general, space-independent metric compatible with translation symmetry. When the weights $[W^i]^a_b \sim \mathcal{N}(0,1)$ are random Gaussian, we have $W^{jT}W^i \approx m^2\delta^{ij}$ and we recover the Euclidean metric. With the lst term vanishing, the loss function equation 18 has a striking resemblance to a Lagrangian used in physics, as we discuss next.

A.3.4 Boundary term with spherical symmetry

When $\mathcal{S}\sim\mathbb{R}^n$ and $G = T_n$, the third term becomes a boundary term. But we can also have $G = SO(n)\times\mathbb{R}_+$ (spherial symmetry and scaling). The boundary $\partial\mathcal{S}\sim S^{n-1}$, which has an $SO(n)$ symmetry. The normal $d\Sigma(\boldsymbol{x})$ is a vector pointing in the radial direction and g is the lift for $\boldsymbol{x}$. Since $g\in SO(n)$, we have

$$d\Sigma_\beta[gL_\alpha\boldsymbol{x}_0]^\beta = d\Sigma^T gL_\alpha\boldsymbol{x}_0 = [g^T d\Sigma]^T L_\alpha\boldsymbol{x}_0 \tag{53}$$

Since $g \in SO(n)$, $g^T = g^{-1}$ and $g^T d\Sigma(g\boldsymbol{x}_0) = d\Sigma(\boldsymbol{x}_0) = V_{n-1}\boldsymbol{x}_0$, meaning the normal vector is rotated back toward $\boldsymbol{x}_0$. Here V_{n-1} is the volume of the boundary S^{n-1}. Hence we have

$$d\Sigma^T g L_i \boldsymbol{x}_0 = \boldsymbol{x}_0^T L_i \boldsymbol{x}_0 = 0 \tag{54}$$

for all generators $L_i \in so(n)$ because $L_i = -L_i^T$ and hence diagonal entries like $\boldsymbol{x}_0^T L_i \boldsymbol{x}_0$ are zero. Only the scaling generator $L_0 = I$ we have $\boldsymbol{x}_0^T L_0 \boldsymbol{x}_0 = 1$. This means that the last term in equation 18 can be nonzero at the boundary only if $\Phi \mathbf{v}^i \Phi$ is in the radial direction, meaning $\bar{\epsilon}^0 = 0$, and Φ does not vanish at the boundary. However, a non-vanishing Φ at the boundary results in diverging loss unless the mass matrix $\mathbf{m}_2$ has eigenvalues equal to zero. This is what happens relativistic theories where light rays can have nonzero Φ at infinity because they are massless.

A.4 Robustness to random noise

Equivariant neural networks are hoped to be more robust than others. One way to check this is to see how the network would perform for an input $\phi' = \phi + \delta\phi$ which adds a small perturbation $\delta\phi$ to a real data point ϕ. Robustness to such perturbation would mean that, for optimal parameters W^*, the loss function would not change, i.e. $I[\phi'; W^*] = I[\phi; W^*]$. This can be cast as a variational equation, requiring I to be minimized around real data points ϕ. Writing $I[\phi; W] = \int d^n x \mathcal{L}[\phi; W]$, we have

$$\delta I[\phi; W^*] = \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^a} \delta\phi^a + \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^a)} \partial_\alpha(\delta\phi^a) \right] \tag{55}$$

Doing a partial integration on the second term, we get

$$\begin{aligned} \delta I[\phi; W^*] &= \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \right] \delta\phi^b + \int_{\mathcal{S}} d^n x \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b \right] \\ &= \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \right] \delta\phi^b + \int_{\partial\mathcal{S}} d^{n-1}\Sigma_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b \right] \end{aligned} \tag{56}$$

where we used the Stoke's theorem again to change the last term to a boundary integral. We will discuss this term further below, but since features ϕ have to finite in extent, $\phi(\boldsymbol{x}) \to 0$ as $|\boldsymbol{x}| \to \infty$, and the boundary term vanishes. We will return to the boundary integral below. The first term in equation 56 is the classic Euler-Lagrange (EL) equation. Thus, requiring robustness, i.e. $\delta I[\phi; W^*]/\delta\phi = 0$ means for optimal parameters W^*, the real data ϕ satisfies the EL equations

$$\text{Robustness to random noise} \iff \text{EL:} \quad \frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} = 0 \tag{57}$$

Applying this to the MSE loss equation 18, equation 57 becomes

$$\mathbf{m}_2 \phi - \partial_\alpha \left(|J| \mathbf{h}^{\alpha\beta} \partial_\beta \phi \right) - \partial_\alpha \left(|J| \mathbf{v}^i [\hat{L}_i]^\alpha \right) \phi = 0 \tag{58}$$

where $|J| = |\partial g / \partial x|$ is the determinant of the Jacobian. For the translation group, equation 58 becomes a Helmholtz equation

$$\mathbf{h}^{ij} \partial_i \partial_j \phi = \bar{\epsilon}^i \mathbf{m}_2 \bar{\epsilon}^j \partial_i \partial_j \phi = \mathbf{m}_2 \phi \tag{59}$$

where $\mathbf{h}^{ij} \partial_i \partial_j = \nabla^2$ is the Laplace-Beltrami operator with $\mathbf{h}$ as the metric.

A.5 Conservation laws

The equivariance condition equation 2 can be written for the integrand of the loss $\mathcal{L}[\phi, W]$. Since G is the symmetry of the system, transforming an input $\phi \to w \cdot \phi$ by $w \in G$ the integrand changes equivariantly as $\mathcal{L}[w \cdot \phi] = w \cdot \mathcal{L}[\phi]$. Now, let w be an infinitesimal $w \approx I + \eta^i L_i$. The action $w \cdot \phi$ can be written as a Taylor expansion, similar to the one in L-conv, yielding

$$\begin{aligned} w \cdot \phi(\boldsymbol{x}) &= \phi(w^{-1}\boldsymbol{x}) = \phi((I - \eta^i L_i)\boldsymbol{x}) = \phi(\boldsymbol{x}) - \eta^i [L_i \boldsymbol{x}]^\alpha \partial_\alpha \phi(\boldsymbol{x}) \\ &= \phi(\boldsymbol{x}) + \delta\boldsymbol{x}^\alpha \partial_\alpha \phi(\boldsymbol{x}) = \phi(\boldsymbol{x}) + \delta\phi(\boldsymbol{x}) \end{aligned} \tag{60}$$

with $\delta \boldsymbol{x}^\alpha = -\eta^i [L_i \boldsymbol{x}]^\alpha$ and $\delta\phi = \delta \boldsymbol{x}^\alpha \partial_\alpha \phi$. Similarly, we have $w \cdot \mathcal{L} = \mathcal{L} + \delta \boldsymbol{x}^\alpha \partial_\alpha \mathcal{L}$. Next, we can use the chain rule to calculate $\mathcal{L}[w \cdot \phi]$.

$$\begin{aligned}\mathcal{L}[w \cdot \phi] = \mathcal{L}[\phi(\boldsymbol{x}) + \delta\phi(\boldsymbol{x})] &= \mathcal{L}[\phi] + \frac{\partial \mathcal{L}}{\partial \phi^b}\delta\phi^b + \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\partial_\alpha\phi^b \\ &= \mathcal{L}[\phi] + \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\right]\delta\phi^b + \partial_\alpha\left(\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b\right) \end{aligned} \tag{61}$$

where we used the fact that $\delta \boldsymbol{x} = \eta^i L_i \boldsymbol{x}$ can vary independently from $\boldsymbol{x}$ (because of η^i), and so $\delta\partial_\alpha\phi^b = \partial_\alpha \delta\phi^b$. The same way, $\delta \boldsymbol{x}^\alpha \partial_\alpha \mathcal{L} = \partial_\alpha(\delta \boldsymbol{x}^\alpha \mathcal{L})$. Now, if ϕ are the real data and the parameters in $\mathcal{L}$ minimize generalization error, then $\mathcal{L}$ satisfies equation 58. This means that the first term in equation 61 vanishes. Setting the second term equal to $w \cdot \mathcal{L}$ we get

$$\mathcal{L}[w \cdot \phi] - w \cdot \mathcal{L}[\phi] = \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b - \delta \boldsymbol{x}^\alpha \mathcal{L}\right] = 0 \tag{62}$$

Thus, the terms in the brackets are divergence free. These terms are called a Noether conserved current J^α. In summary

$$\text{Noether current: } J^\alpha = \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b - \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\alpha}\delta \boldsymbol{x}^\alpha, \qquad \delta I[\phi; W^*] = 0 \quad \Rightarrow \partial_\alpha J^\alpha = 0 \tag{63}$$

J captures the change of the Lagrangian $\mathcal{L}$ along symmetry direction $\hat{L}_i$. Plugging $\delta\phi = \delta \boldsymbol{x}^\alpha \partial_\alpha \phi$from equation 60 we find

$$\partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b - \delta \boldsymbol{x}^\alpha \mathcal{L}\right] = \delta \boldsymbol{x}^\beta \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\partial_\beta \phi^b - \delta^\alpha_\beta \mathcal{L}\right] = \delta \boldsymbol{x}^\beta \partial_\alpha T^\alpha_\beta. \tag{64}$$

T^α_β is known as the stress-energy tensor in physics (Landau, 2013). It is the Noether current associated with space (or space-time) variations $\delta \boldsymbol{x}$. It appears here because G acts on the space, as opposed to acting on feature dimensions. For the MSE loss we have

$$T^\alpha_\beta \equiv \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\partial_\beta \phi^b - \delta^\alpha_\beta \mathcal{L} = \partial_\rho \phi^T \left(\delta^\lambda_\beta \mathbf{h}^{\alpha\rho} - \delta^\alpha_\beta \mathbf{h}^{\rho\lambda}\right)\partial_\lambda \phi - \phi^T \mathbf{m}_2 \phi \tag{65}$$

It would be interesting to see if the conserved currents can be used in practice as an alternative way for identifying or discovering symmetries.

B Tensor notation details

If the dataset being analyzed is in the form of $f(\boldsymbol{x})$ for some sample of points $\boldsymbol{x}$, together with derivatives $\nabla f(\boldsymbol{x})$, we can use the L-conv formulation above. However, in many datasets, such as images, $f(\boldsymbol{x})$ is given as a finite dimensional array or tensor, with $\boldsymbol{x}$ taking values over a grid. Even though the space $\mathcal{S}$ is now discrete, the group which acts on it can still be continuous (e.g. image rotations). Let $\mathcal{S} = \{\boldsymbol{x}_0, \ldots \boldsymbol{x}_{d-1}\}$ contain d points. Each $\boldsymbol{x}_\mu$ represents a coordinate in higher dimensional grid. For instance, on a 10×10 image, $\boldsymbol{x}_0$ is $(x, y) = (0, 0)$ point and $\boldsymbol{x}_{99}$ is $(x, y) = (9, 9)$.

Feature maps To define features $f(\boldsymbol{x}_\mu) \in \mathbb{R}^m$ for $\boldsymbol{x}_\mu \in \mathcal{S}$, we embed $\boldsymbol{x}_\mu \in \mathbb{R}^d$ and encode them as the canonical basis (one-hot) vectors with components $[\boldsymbol{x}_\mu]^\nu = \delta^\nu_\mu$ (Kronecker delta), e.g. $\boldsymbol{x}_0 = (1, 0, \ldots, 0)$. The feature space becomes $\mathcal{F} = \mathbb{R}^d \otimes \mathbb{R}^m$, meaning feature maps $\boldsymbol{f} \in \mathcal{F}$ are $d \times m$ tensors, with $f(\boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T \boldsymbol{f} = \boldsymbol{f}_\mu$.

Group action Any subgroup $G \subseteq \mathrm{GL}_d(\mathbb{R})$ of the general linear group (invertible $d \times d$ matrices) acts on $\mathbb{R}^d$ and $\mathcal{F}$. Since $\boldsymbol{x}_\mu \in \mathbb{R}^d$, $g \in G$ also naturally act on $\boldsymbol{x}_\mu$. The resulting $\boldsymbol{y} = g\boldsymbol{x}_\mu$ is a linear combination $\boldsymbol{y} = c^\nu \boldsymbol{x}_\nu$ of elements of the discrete $\mathcal{S}$, not a single element. The action of G on $\boldsymbol{f}$ and $\boldsymbol{x}$, can be defined in multiple equivalent ways. We define $f(g \cdot \boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T g^T \boldsymbol{f}, \forall g \in G$. For $w \in G$ we have

$$w \cdot f(\boldsymbol{x}_\mu) = f(w^{-1} \cdot \boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T w^{-1T} \boldsymbol{f} = [w^{-1}\boldsymbol{x}]^T \boldsymbol{f} \tag{66}$$

Dropping the position $\boldsymbol{x}_\mu$, the transformed features are matrix product $w \cdot \boldsymbol{f} = w^{-1T}\boldsymbol{f}$.

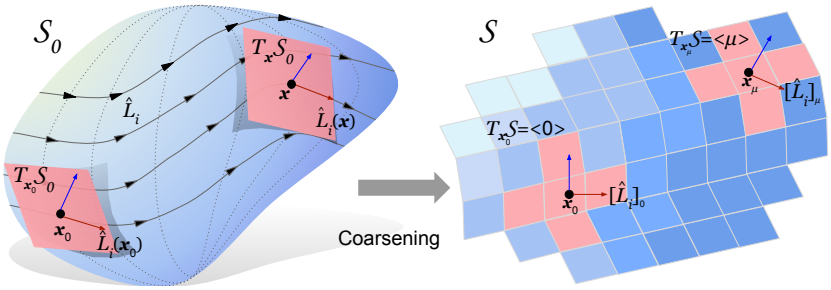

Figure 5: **Manifold vs. discretized Space** While real systems can have a continuous manifold $\mathcal{S}_0$ as their base space, often the data collected from is a discrete array $\mathcal{S}$. The discretization (coarsening) will induce some of the topology of $\mathcal{S}_0$ on $\mathcal{S}$ as a graph. Graph neighborhoods $< \mu >$ on the discrete $\mathcal{S}$ represent tangent spaces $T_{\boldsymbol{x}_\mu}\mathcal{S}$ and approximate $T_{\boldsymbol{x}}\mathcal{S}_0$. The lift takes $\boldsymbol{x} \in \mathcal{S}_0$ to $g \in G$, and maps the tangent spaces $T_{\boldsymbol{x}}\mathcal{S}_0 \to T_g G$. Each Lie algebra basis $L_i \in \mathfrak{g} = T_I G$ generates a vector field on the tangent bundle TG via the pushforward as $L^{(g)} = gL_i g^{-1}$. Due to the lift, each L_i also generates a vector field $\hat{L}_i^\alpha(\boldsymbol{x})\partial_\alpha = [gL_i\boldsymbol{x}_0]^\alpha \partial_\alpha$, with $\boldsymbol{x} = g\boldsymbol{x}_0$. Analogously, on $\mathcal{S}$ we get a vector field $[\hat{L}_i]_\mu = g_\mu L_i \boldsymbol{x}_0$ on $T\mathcal{S}$, with $\boldsymbol{x}_\mu = g_\mu \boldsymbol{x}_0$. Note that depicting $\mathcal{S}$ and $\mathcal{S}_0$ as 2D is only for convenience. They may have any dimensions.

G-conv and L-conv in tensor notation Writing G-conv equation 3 in the tensor notation we have

$$[\kappa \star f](g\boldsymbol{x}_0) = \int_G \kappa(v) f(gv\boldsymbol{x}_0) dv = \boldsymbol{x}_0^T \int_G v^T g^T \boldsymbol{f} \kappa^T(v) dv \equiv \boldsymbol{x}_0^T [\boldsymbol{f} \star \kappa](g) \tag{67}$$

where we moved $\kappa^T(v) \in \mathbb{R}^m \otimes \mathbb{R}^{m'}$ to the right of $\boldsymbol{f}$ because it acts as a matrix on the output index of $\boldsymbol{f}$. The equivariance of equation 67 is readily checked with $w \in G$

$$w \cdot [\boldsymbol{f} \star \kappa](g) = [\boldsymbol{f} \star \kappa](w^{-1}g) = \int_G v^T g^T w^{-1T} \boldsymbol{f} \kappa^T(v) dv = [(w \cdot \boldsymbol{f}) \star \kappa](g) \tag{68}$$

where we used $[w^{-1}g]^T \boldsymbol{f} = g^T w^{-1T} \boldsymbol{f}$. Similarly, we can rewrite L-conv equation 6 in the tensor notation. Defining $v_\epsilon = I + \bar{\epsilon}^i L_i$

$$\begin{aligned} Q[\boldsymbol{f}](g) &= W^0 f\left(g\left(I + \bar{\epsilon}^i L_i\right)\right) = \boldsymbol{x}_0^T \left(I + \bar{\epsilon}^i L_i\right)^T g^T \boldsymbol{f} W^{0T} \\ &= \left(\boldsymbol{x}_0 + \bar{\epsilon}^i [gL_i \boldsymbol{x}_0]\right)^T \boldsymbol{f} W^{0T}. \end{aligned} \tag{69}$$

Here, $\hat{L}_i = gL_i\boldsymbol{x}_0$ is eactly the tensor analogue of pushforward vector field $\hat{L}_i$ in equation 8. We will make this analogy more precise below. The equivariance of L-conv in tensor notation is again evident from the $g^T \boldsymbol{f}$, resulting in

$$Q[w \cdot \boldsymbol{f}](g) = \boldsymbol{x}_0^T v_\epsilon^T g^T w^{-1T} \boldsymbol{f} W^{0T} = Q[\boldsymbol{f}](w^{-1}g) = w \cdot Q[\boldsymbol{f}](g) \tag{70}$$

Next, we will discuss how to implement equation 69 in practice and how to learn symmetries with L-conv. We will also discuss the relation between L-conv and other neural architectures.

B.1 Constraints from topology on tensor L-conv

To implement equation 69 we need to specify the lift and the form of L_i. We will now discuss the mathematical details leading to and easy to implement form of equation 69.

Topology Although he discrete space $\mathcal{S}$ is a set of points, in many cases it has a topology. For instance, $\mathcal{S}$ can be a discretization of a manifold $\mathcal{S}_0$, or vertices on a lattice or a general graph. We encode this topology in an undirected graph (i.e. 1-simplex) with vertex set $\mathcal{S}$ edge set $\mathcal{E}$. Instead of the commonly used graph adjacency matrix $\boldsymbol{A}$, we will use the incidence matrix $\boldsymbol{B} : \mathcal{S} \times \mathcal{E} \to \{0, 1, -1\}$. $\boldsymbol{B}_\alpha^\mu = 1$ or -1 if edge α starts or ends at node μ, respectively, and $\boldsymbol{B}_\alpha^\mu = 0$ otherwise (undirected graphs have pairs of incoming and outgoing edges). Similar to the continuous case we will denote the topological space $(\mathcal{S}, \mathcal{E}, \boldsymbol{B})$ simply by $\mathcal{S}$.

Figure 5 summarizes some of the aspects of the discretization as well as analogies between $\mathcal{S}_0$ and $\mathcal{S}$. Technically, the group G_0 acting on $\mathcal{S}_0$ and G acting on $\mathcal{S}$ are different. But we can find a group G which closely approximates G_0 (see SI B.2). For instance, Rao & Ruderman (1999) used Shannon-Whittaker interpolation theorem (Whitaker, 1915) to define continuous 1D translation and 2D rotation groups on discrete data. We return to this when expressing CNN as L-conv in 4.

Neighborhoods as discrete tangent bundle $\boldsymbol{B}$ is useful for extending differential geometry to graphs (Schaub et al., 2020). Define the neighborhood $<\mu>=\left\{\alpha\in\mathcal{E}\middle|\boldsymbol{B}^{\mu}_{\alpha}=-1\right\}$ of $\boldsymbol{x}_\mu$ as the set of outgoing edges. $<\mu>$ can be identified with $T_{\boldsymbol{x}_\mu}\mathcal{S}$ as for $\alpha\in<\mu>$, $\boldsymbol{B}_\alpha\boldsymbol{f}=\boldsymbol{f}_\mu-\boldsymbol{f}_\nu\sim\partial_\alpha\boldsymbol{f}$, where μ and ν are the endpoints of edge α. This relation becomes exact when $\mathcal{S}$ is an nD square lattice with infinitesimal lattice spacing. The set of all neighborhoods is $\boldsymbol{B}$ itself and encodes the approximate tangent bundle $T\mathcal{S}$. For some operator $C:\mathcal{S}\otimes\mathcal{F}\to\mathcal{F}$ acting on $\boldsymbol{f}$ we will say $C\in<\mu>$ if its action remains within vertices connected to μ, meaning

$$C\in<\mu>:\qquad C\boldsymbol{f}=\sum_{\alpha\in<\mu>}\tilde{C}^{\alpha}\boldsymbol{B}^{\nu}_{\alpha}\boldsymbol{f}_\nu \tag{71}$$

Lift and group action Lie algebra elements L_i by definition take the origin $\boldsymbol{x}_0$ to points close to it. Thus, for small enough η, $(I+\eta L_i)\boldsymbol{x}_0\in<0>$ and so $[L_i\boldsymbol{x}_0]^T\boldsymbol{f}=[\hat{L}_i]^{\rho}_0\boldsymbol{f}_\rho=\sum_{\alpha\in<0>}[\hat{\ell}_i]^{\alpha}_0\boldsymbol{B}^{\rho}_{\alpha}\boldsymbol{f}_\rho$. The coefficients $[\hat{\ell}_i]^{\alpha}_0\in\mathbb{R}$ are in fact the discrete version of $\hat{L}_i$ components from equation 8. For the pushforward gL_ig^{-1}, we define the lift via $\boldsymbol{x}_\mu=g_\mu\boldsymbol{x}_0$. We require the G-action to preserve the topology of $\mathcal{S}$, meaning points which are close remain close after the G-action. As a result, $<\mu>$ can be reached by pushing forward elements in $<0>$. Thus, for each i, $\exists\eta\ll 1$ such that $g_\mu(I+\eta L_i)\boldsymbol{x}_0\in<\mu>$, meaning for a set of coefficients $[\hat{L}_i]^{\nu}_{\mu}\in\mathbb{R}$ we have

$$[g_\mu(I+\eta L_i)\boldsymbol{x}_0]^T\boldsymbol{f}=\boldsymbol{f}_\mu+\eta\sum_{\alpha\in<\mu>}[\hat{\ell}_i]^{\alpha}_{\mu}\boldsymbol{B}^{\nu}_{\alpha}\boldsymbol{f}_\nu \tag{72}$$

where $\boldsymbol{f}_\mu=\boldsymbol{x}_\mu^T\boldsymbol{f}$. Acting with $[g_\mu L_i\boldsymbol{x}_0]^T\boldsymbol{x}_\nu$ and inserting $I=\sum_\rho\boldsymbol{x}_\rho\boldsymbol{x}_\rho^T$ we have

$$\begin{aligned}[\hat{L}_i]^{\nu}_{\mu}=[\hat{\ell}_i]^{\alpha}_{\mu}\boldsymbol{B}^{\nu}_{\alpha}&=[g_\mu L_i\boldsymbol{x}_0]^{\nu}=\sum_{\rho}[g_\mu\boldsymbol{x}_\rho\boldsymbol{x}_\rho^T L_i\boldsymbol{x}_0]^T\boldsymbol{x}_\nu\\&=\sum_{\rho\in<0>}[L_i]^{\rho}_0[\boldsymbol{x}_\nu^T g_\mu\boldsymbol{x}_\rho]^T=[L_i]^{\rho}_0[g_\mu]^{\nu}_{\rho}=[\hat{\ell}_i]^{\alpha}_0\boldsymbol{B}^{\rho}_{\alpha}[g_\mu]^{\nu}_{\rho}\end{aligned} \tag{73}$$

This $\hat{L}_i\equiv\hat{\ell}^{\alpha}_i\boldsymbol{B}_\alpha$ is the discrete $\mathcal{S}$ version of the vector field $\hat{L}_i(\boldsymbol{x})=[gL_i\boldsymbol{x}_0]^{\alpha}\partial_\alpha$ in equation 8.

B.2 Approximating a symmetry and discretization error

Discretization error While systems such as crystalline solids are discrete in nature, many other datasets such as images result from discretization of continuous data. The discretization (or "coarsening" (Bronstein et al., 2021)) will modify the groups that can act on the space. For example, first rotating a shape by $SO(2)$ then taking a picture is different from first taking a picture then rotating the picture (i.e. group action and discretization do not commute). Nevertheless, in most cases in physics and machine learning the symmetry group G_0 of the space before discretization has a small Lie algebra dimension n, (e.g. $SO(3), SE(3), SO(3,1)$ etc). Usually the resolution of the discretization is $d\gg n$. In this case, there always exist some $G\subseteq\mathrm{GL}_d(\mathbb{R})$ which approximates G_0 reasonably well. The approximation means $\forall g_0\in G_0,\exists g\in G$ such that the error $\mathcal{L}_G=\|g_0\cdot f(\boldsymbol{x}_\mu)-\boldsymbol{x}_\mu^T g\boldsymbol{f}\|^2<\eta^2$ where η depends on the resolution of the discretization. Minimizing the error $\mathcal{L}_G$ can be the process of identifying the G which best approximates G_0. We will denote this approximate similarity as $G\simeq G_0$. For example, Rao & Ruderman (1999) used the Shannon-Whittaker Interpolation theorem (Whitaker, 1915) to translate discrete 1D signals (features) by arbitrary, continuous amounts. In this case the transformed features are $\boldsymbol{f}'_\mu=g(z)^{\nu}_{\mu}\boldsymbol{f}_\nu$, where $g(z)^{\nu}_{\mu}=\frac{1}{d}\sum_{p=-d/2}^{d/2}\cos\left(\frac{2\pi p}{d}(z+\mu-\nu)\right)$ approximates the shift operator for continuous z. The $g(z)$ form a group because $g(w)g(z)=g(w+z)$, which is a representation for periodic 1D shifts. Rao & Ruderman (1999) also use a 2D version of the interpolation theorem to approximate $SO(2)$. In practice, we can assume the true symmetry to be G, as we only have access to the discretized data and can't measure G_0 directly.

B.3 Comparison with other symmetry discovery methods

Meta-learning Symmetries by Reparameterization Recently Zhou et al. (2020) also introduced an architecture which can learn equivariances from data. We would like to highlight the differences between their approach and ours, specifically Proposition 1 in Zhou et al. (2020). Assuming a discrete group $G = \{g_1, \ldots, g_n\}$, they decompose the weights $W \in \mathbb{R}^{s\times s}$ of a fully-connected layer, acting on $\boldsymbol{x} \in \mathbb{R}^s$ as $\mathrm{vec}(W) = U^G v$ where $U^G \in \mathbb{R}^{s\times s}$ are the "symmetry matrices" and $v \in \mathbb{R}^s$ are the "filter weights". Then they use meta-learning to learn U^G and during the main training keep U^G fixed and only learn v. We may compare MSR to our approach by setting $d = s$. First, note that although the dimensionality of $U \in \mathbb{R}^{nd\times d}$ seems similar to our $L \in \mathbb{R}^{n\times d\times d}$, the L_i are n matrices of shape $d \times d$, whereas U has shape $(nd) \times d$ with many more parameters than L. Also, the weights of L-conv $W \in \mathbb{R}^{n\times m_l\times m_{l-1}}$, with m_l being the number of channels, are generally much fewer than MSR filters $v \in \mathbb{R}^d$. Finally, the way in which Uv acts on data is different from L-conv, as the dimensions reveal. The prohibitively high dimensionality of U requires MSR to adopt a sparse-coding scheme, mainly Kronecker decomposition. Though not necessary, we too choose to use a sparse format for L_i, finding that very low-rank L_i often perform best. A Kronecker decomposition may bias the structure of U^G as it introduces a block structure into it.

Augerino In a concurrent work, (Benton et al., 2020) propose Augerino, a method to learn equivariance with neural networks, but restricted to a subgroup of the augmentation transformations. Augerino learns which subset of the augmentations improved the prediction. This is done by writing The data augmentation is written as $g_\epsilon = \exp\left(\sum_i \epsilon_i \theta_i L_i\right)$ (equation (9) in Benton et al. (2020)), with randomly sampled $\epsilon_i \in [-1, 1]$. θ_i are trainable weights which determine which L_i helped with the learning task. Furthermore, Their Lie algebra is fixed to affine transformations in 2D (translations, rotations, scaling and shearing). Our approach is more general. We learn the L_i directly without restricting to known symmetries. Additionally, we do not use the exponential map or matrix logarithm, hence, our method is easy to implement. Lastly, Augerino uses sampling to effectively cover the space of group transformations. Since we work with the Lie algebra rather the group itself, we do not require sampling.

C Experiments

We conduct a set of experiments to see how well L-conv can extract infinitesimal generators.

Nonlinear activation As noted in Weiler et al. (2018a), an arbitrary nonlinear activation σ may not keep the architecture equivariant under G. However, as we showed in SI A (**Extended equivariance for L-conv**), the feature dimensions can pass through any nonlinear neural network without affecting the equivariance of L-conv. This means that the weights of the nonlinear layer should act only on $\mathcal{F}$ and not $\mathcal{S}$.

Implementation The basic way to implement L-conv is as multiple parallel GCN units with aggregation function $f(A)$ being (propagation rule) being $\hat{L}_i$. We do not use Deep Graph Library (DGL) or other libraries, as we want to make $\hat{L}_i$ learnable for discovering symmetries. A more detailed way to implement L-conv is to encode $\hat{L}_i$ in the form of equation 73 to ensure that there is an underlying shared generator $\hat{\ell}_i$ for all μ which is pushed forward using g_μ, shared for all i. To implement L-conv this way, we need the lift g_μ and the edge weights $w_i^\alpha = [\hat{\ell}_i]_0^\alpha$. With known symmetries, the learnable parameters are W^0 and $\bar{\epsilon}^i$. When the topology and hence $\boldsymbol{B}$ is known, we can encode it into the geometry and only learn the edge weights $[\hat{\ell}_i]_\mu^\alpha$, similar to edge features in message passing neural networks (MPNN) (Gilmer et al., 2017). In general each $\hat{\ell}_i$ has $|\mathcal{E}|$ (i.e. number of edges) components. We can further reduce these using equation 73, where instead of n matrices $\hat{\ell}_i$, we learn one g_μ shared for all, and a small set of elements $[\hat{\ell}_i]_0^\alpha$. This is easiest when the graph is a regular lattice and each vertex has the same number of neighbors. When the topology of the underlying space is not known (e.g. point cloud or scrambled coordinates), we can learn $\hat{L}_i$ as $d \times d$ matrices. We do this for the scrambled image tests, where we encode $\hat{L}_i$ as low-rank matrices.

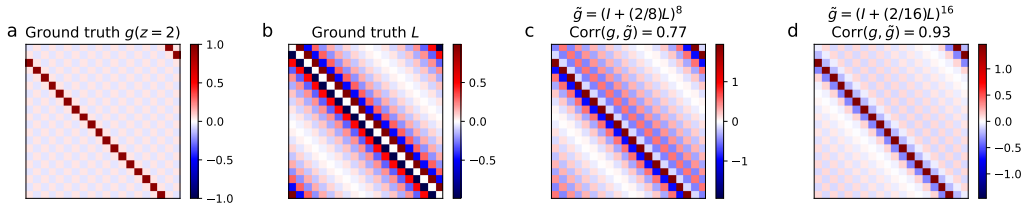

Figure 6: 1D Translation: Using the Shannon-Whittaker Interpolation (SWI) one can generate continuous shifts on discrete data. These include integer shifts (a, ground truth). (SWI) also yields an infinitesimal generator L for shifts (b). This L can be used to approximate finite shifts using $\tilde{g}_n(z) = (I + z/nL)^n$, with $n \to \infty$ yielding $\exp[zL]$. (c) and (d) show the approximation of a shift by two pixels using $n = 8$ and $n = 16$.

Symmetry Discovery Literature In addition to simplifying the construction of equivariant architectures, our method can also learn the symmetry generators from data. Learning symmetries from data has been studied before, but mostly in restricted settings. Examples include commutative Lie groups as in Cohen & Welling (2014), 2D rotations and translations in Rao & Ruderman (1999), Sohl-Dickstein et al. (2010) or permutations (Anselmi et al., 2019). Zhou et al. (2020) uses meta-learning to automatically learn symmetries in the data. Yet their weight-sharing scheme and the encoding of the symmetry generators is very different from ours. (Benton et al., 2020) propose Augerino, a method to learn equivariance with neural networks, but restricted to a subgroup of the augmentation transformations. Their Lie algebra is fixed to affine transformations in 2D (translations, rotations, scaling and shearing). Our approach is more general. We learn the L_i directly without restricting to known symmetries. Additionally, we do not use the exponential map or matrix logarithm, hence, our method is easy to implement. Lastly, Augerino uses sampling to effectively cover the space of group transformations. Since we work with the Lie algebra rather the group itself, we do not require sampling.

C.1 Approximating 1D CNN

As discussed in the text, Rao & Ruderman (1999, sec. 4) used the Shannon-Whittaker Interpolation (SWI) (Whitaker, 1915) to define continuous translation on periodic 1D arrays as $\boldsymbol{f}'_\rho = g(z)^\nu_\rho \boldsymbol{f}_\nu$. Here $g(z)^\nu_\rho = \frac{1}{d}\sum_{p=-d/2}^{d/2} \cos\left(\frac{2\pi p}{d}(z+\rho-\nu)\right)$ approximates the shift operator for continuous z. These $g(z)$ form a 1D translation group G as $g(w)g(z) = g(w+z)$ with $g(0)^\nu_\rho = \delta^\nu_\rho$. For any $z = \mu \in \mathbb{Z}$, $g_\mu = g(z=\mu)$ are circulant matrices that shift by μ as $[g_\mu]^\rho_\nu = \delta^\rho_{\nu-\mu}$. g_μ can be approximated using the Lie algebra and written as multi-layer L-conv as in sec. 2.1. Using $g(0)^\nu_\rho \approx \delta(\rho-\nu)$, the single Lie algebra basis $[\hat{L}]_0 = \partial_z g(z)|_{z\to 0}$, acts as $\hat{L}f(z) \approx -\partial_z f(z)$ (because $\int \partial_z \delta(z-\nu) f(z) = -\partial_\nu f(\nu)$). Its components are $\hat{L}^\nu_\rho = L(\rho-\nu) = \sum_p \frac{2\pi p}{d^2}\sin\left(\frac{2\pi p}{d}(\rho-\nu)\right)$, which are also circulant due to the $(\rho-\nu)$ dependence. Hence, $[\hat{L}\boldsymbol{f}]_\rho = \sum_\nu L(\rho-\nu)\boldsymbol{f}_\nu = [L \star \boldsymbol{f}]_\nu$ is a convolution. Rao & Ruderman (1999) already showed that this $\hat{L}$ can reproduce finite discrete shifts g_μ used in CNN. They used a primitive version of L-conv with $g_\mu = (I + \epsilon\hat{L})^N$. Thus, L-conv can approximate 1D CNN. This result generalizes easily to higher dimensions.

Figure 6 shows how this approximation works. (b) shows the analytical form of $\hat{L}$. (c) and (d) show two approximations of $g_2 = g(z=2)$, shift by two pixels, using $\tilde{g}(z) = (I + z/n\hat{L})^n$ with $n = 8$ and $n = 16$. We can evaluate the quality of these approximations using their cosine correlation defined as $\mathrm{Corr}(g, \tilde{g}) = \mathrm{Tr}\left[g^T\tilde{g}\right]/(\|g\|\|\tilde{g}\|)$ where $\|A\| = \sqrt{\sum_{i,j}(A^i_j)^2}$ is the Frobenius L_2 norm. (c) shows 0.77 correlation and (d) has 0.93.

C.2 Extracting 2D rotation generator for fixed small rotations

Ground truth We can use the same SWI 1D translation generators discussed above for CNN as ∂_x and ∂_y to construct the rotation generator $L_\theta = x\partial_y - y\partial_x$ (Fig. 7, a). We will use cosine correlation with this L_θ to evaluate the quality of the learned $\hat{L}$. As we will find below, the best outcome is from

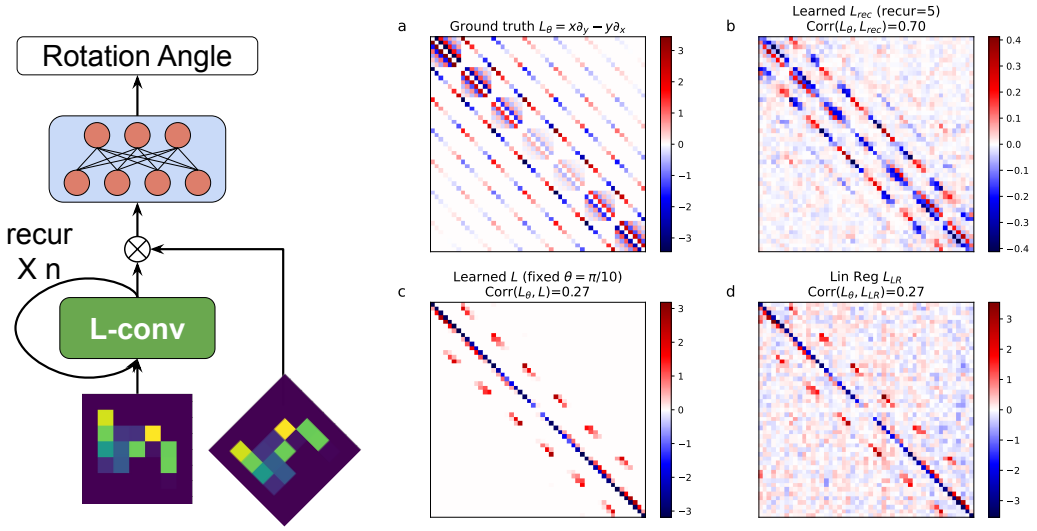

Figure 7: **Learning the infinitesimal generator of** $SO(2)$ Left shows the architecture for learning rotation angles between pairs of images. (a) shows the rotation generator calculated analytically using Shannon-Whittaker interpolation. (b) is the $\hat{L}$ learned using recursive L-conv learning rotation angle between a pair of images. (c) is an L learned using a fixed small rotation angle $\theta = \pi/10$, and (d) shows $\hat{L}$ found using the numeric linear regression solution from the fixed angle data. (b) has the highest cosine correlation (0.70) with the ground truth, compared to 0.27 for $\hat{L}$ extracted using small angles.

$\hat{L}$ learned using a recursive L-conv learning the angle of rotation between a pair of images (Fig. 7, b) with 0.70 correlation.

Using fixed small angle In the first experiment we try to learn a small rotation with angle $\theta = \pi/10$ using a single layer L-conv (Fig. 7,c). This experiment was already done in (Rao & Ruderman, 1999). The input is a random 7×7 image $\boldsymbol{f}$ with pixels chosen in $[-.5, 0.5)$. The output $\boldsymbol{f}'$ is the same image rotated by θ using pytorch affine transform. Our training set contains 50,000 images., the test set was 10,000 images, batch size was 64. The code was implemented in pytorch and we used the Adam optimizer with learning rate 10^{-2}. The experiments were run for 20 epochs. This problem is simply a linear regression with $\boldsymbol{f}' = R\boldsymbol{f} = (I + \epsilon\hat{L})\boldsymbol{f}$. L-conv solves it using SGD and finds $\epsilon\hat{L}$. This problem can also be solved exactly using the solution to linear regression. Let $X = (\boldsymbol{f}_1, \dots \boldsymbol{f}_N)$ and $Y = (\boldsymbol{f}'_1, \dots \boldsymbol{f}'_N)$ be the matrix of all inputs and outputs, respectively. The rotation equation is $Y^T = RX^T$. Thus, the rotation matrix is given by $R = (Y^T X)(X^T X)^{-1}$. Figure 7 (c, d) shows the results of this experiment. The L found using L-conv with SGD is much cleaner than the numerical linear regression solution $L_{LR} = (R - I)/\theta$. The loss becomes extremely small both on training and test data.

C.3 Learning rotation angle

In this experiment we have a pair of input images $(\boldsymbol{f}_n, R(\theta_n)\boldsymbol{f}_n)$, with $R(\theta) \in SO(2)$ (approximating 2D rotations). The two inputs differ by a finite rotation with angle $\theta_n \in [0, \pi/8)$. Th task is to learn the rotation angle θ_n. For this task we use a recursive L-conv. We set $W^0 = I$. The L-conv weight $\bar{\epsilon}$ is $m \times m$. To be able to encode multiple angles, we set $m = 10$ and feed 10 copies of the $\boldsymbol{f}_n$ as input $\boldsymbol{h}_0 = [\boldsymbol{f}_n] \times 10$. We pass this through the same L-conv layer $t = 3$ times as $\boldsymbol{h}_i = Q[\boldsymbol{h}_{i-1}]$. In the final layer, we first take the dot product of the final output $\boldsymbol{h}_t$ with the rotated input $\boldsymbol{y}_n = R(\theta)\boldsymbol{f}_n$ to obtain $\boldsymbol{g} = \tanh\left({\boldsymbol{y}_n}^T \boldsymbol{h}_t\right)$. We then pass the output $\boldsymbol{g} \in \mathbb{R}^m$ through a fully-connected (FC) layer with 5 nodes and tanh activation, and finally through a linear FC layer with one output to obtain the angle. The batch size was 16, Adam optimizer, learning rate 10^{-3}, rest were default.

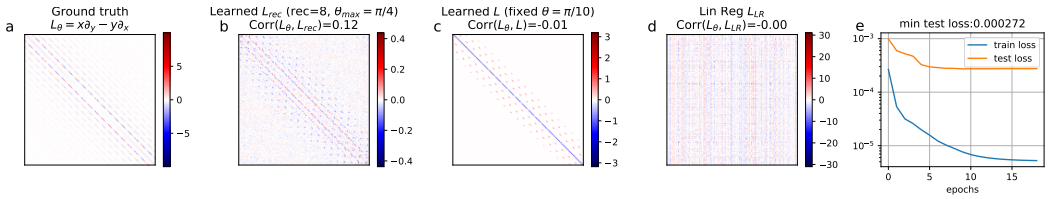

Figure 8: Learning $\hat{L}$ via larger rotation angles for larger images. This time the correlation with ground truth L_θ is much less, but accuracy is still very good.

Despite being a much harder task than fitting a fixed angle rotation, the learned $\hat{L}$ of this experiment has the highest (0.70) cosine correlation with the ground truth L_θ (Fig. 7, b). Even though the architecture is rather complicated and L-conv is followed by two MLP layers, the $\hat{L}$ in L-conv learns the infinitesimal generator of rotations very well. We also conducted experiments with larger random images (20×20) and larger angles of rotation $\theta_n \in [0, \pi/4)$ (Fig. 8). While the accuracy of learning the angles is still pretty good (Fig. 8, e,test loss 2.7×10^{-4}) the larger angles result in less correlation between the learned $\hat{L}$ and the ground truth L_θ (Fig. 8, b, correlation 0.12 with 8 times recurrence). The learned $\hat{L}$ is closer to a finite angle rotation. This may be because the with small number of recurrences the network found small but finite rotations approximate larger rotations better than using a true infinitesimal generator.

D Experiments on Images

To understand precisely how L-conv performs in comparison with CNN and other baselines, we conduct a set of carefully designed experiments. Defining pooling for L-conv merits more research. Without pooling, we cannot use L-conv in state-of-the-art models for problems such as image classification. Therefore, we use the simplest possible models in our experiments: one or two L-conv, or CNN, or FC layers, followed by a classification layer. We do not use any other operations such as dropout or batch normalization in any of the experiments.

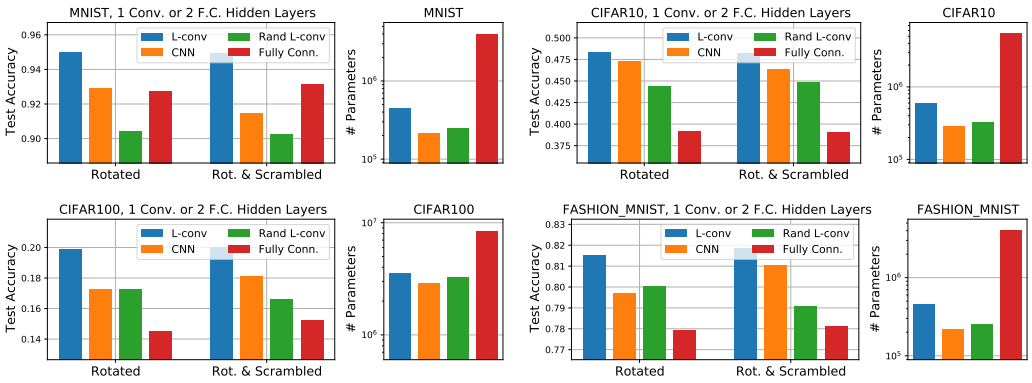

Figure 9: Results on four datasets with two variant: "Rotated" and "Rotated and scrambled". In all cases L-conv performs best. On MNIST, FC and CNN come close, but using 5x more parameters.

Test Datasets We use four datasets: MNIST, CIFAR10, CIFAR100, and FashionMNIST. To test the efficiency of L-conv in dealing with hidden or unfamiliar symmetries, we conducted our tests on two modified versions of each dataset: 1) **Rotated:** each image rotated by a random angle (no augmentation); 2) **Rotated and Scrambled:** random rotations are followed by a fixed random permutation (same for all images) of pixels. We used a 80-20 training test split on 60,000 MNIST and FashionMNIST, and on 50,000 CIFAR10 and CIFAR100 images. Scrambling destroys the correlations existing between values of neighboring pixels, removing the locality of features in images. As a result, CNN need to encode more patterns, as each image patch has a different correlation pattern.

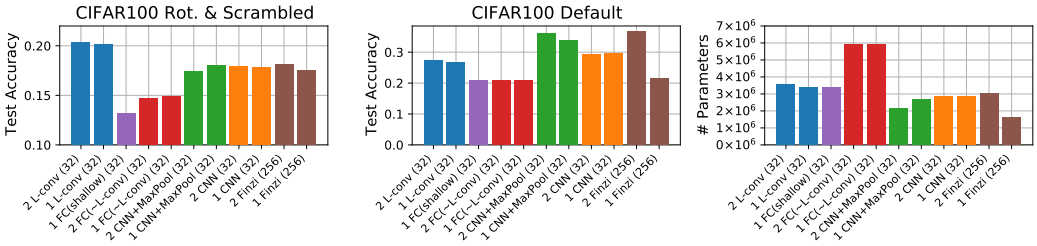

Figure 10: Comparison of one and two layer performance of L-conv (blue), CNN without pooling (orange), CNN with Maxpooling after each layer (green), fully connected (FC) with structure similar to L-conv (red), Lie-Conv (Finzi) Finzi et al. (2020) (brown), and shallow FC, which has a single hidden layer with width such that the total number of parameters matches L-conv (purple). The labels indicate number of layers, layer architecture and number of filters (e.g. "2 L-conv (32)" means two layers of L-conv with 32 filters followed by one classification layer). Left and middle plots show test accuracies on CIFAR100 with rotated and scrambled images, and on the original CIFAR100 dataset, respectively. The plot on the right shows the number of parameters in each model, which is the same for the two datasets.

Test Model Architectures We conduct controlled experiments, with one (Fig. 9) or two (Fig. 10) hidden layers being either L-conv or a baseline, followed by a classification layer. For CNN, L-conv and L-conv with random L_i, we used $n_f = m_l = 32$ for number of output filters (i.e. output dimension of W^i). For CNN we used 3×3 kernels and equivalently used $n_L = 9$ for the number of L_i in L-conv and random L-conv. We also used "LieConv" Finzi et al. (2020) as a baseline (Fig. 10, brown). We used the default $k = 256$ in LieConv, which yields comparable number of parameters to our other models. For the symmetry group in LieConv we used $SE(3)$. We also used the default ResNet architecture provided by Finzi et al. (2020) for both the one and two layer experiments. We turned off batch normalization, consistent with other experiments. We encode L_i as sparse matrices $L_i = U_i V_i$ with hidden dimension $d_h = 16$ in Fig. 9 and $d_h = 8$ in Fig. 10, showing that very sparse L_i can perform well. The weights W^i are each $m_l \times m_{l+1}$ dimensional. The output of the L-conv layer is $d \times m_{l+1}$. As mentioned above, we use two FC baselines. The FC in Fig. 9 and FC($\sim$L-conv) in Fig. 10 mimic L-conv, but lacks weight-sharing. The FC weights are $W = ZV$ with V being $(n_L d_h) \times d$ and Z being $(m_{l+1} \times d) \times d_h$. For "FC (shallow)" in Fig. 10, we have one wide hidden layer with $u = n_{L-conv}/(mdc)$, where n_{L-conv} is the total number of parameters in the L-conv model, m and c the input and output channels, and d is the input dimension. We experimented with encoding L_i as multi-layer perceptrons, but found that a single hidden layer with linear activation works best. We also conduct tests with two layers of L-conv, CNN and FC (Fig. 10), with each L-conv, CNN and FC layer as descried above, except that we reduced the hidden dimension in L_i to $d_h = 8$.

Baselines We compare L-conv against four baselines: CNN, random L_i, fully connected (FC) and LieConv. Using CNN or $SE(3)$ LieConv on scrambled images amounts to using poor inductive bias in designing the architecture. Similarly, random, untrained L_i is like using bad inductive biases. Testing on random L_i serves to verify that L-conv's performance is not due to the structure of the architecture, and that the L_i in L-conv really learn patterns in the data. Finally, to verify that the higher parameter count in L-conv is not responsible for the high performance, we construct two kinds of FC models. The first type ("Fully Conn." in Fig. 9 and "FC ($\sim$ L-conv)" in Fig. 10) is a multilayer FC network with the same input ($d \times m_0$), hidden ($k \times n_L$ for low-rank L_i) and output ($d \times m_1$) dimensions as L-conv, but lacking the weight-sharing, leading to much larger number parameters than L-conv. The second type ("FC (shallow)" in Fig. 10) consists of a single hidden layer with a width such that the total number of model parameters match L-conv.

Results Fig. 9 shows the results for single layer experiments. On all four datasets both in the rotated and the rotated and scrambled case L-conv performed considerably better than CNN and the baselines. Compared to CNN, L-conv naturally requires extra parameters to encode L_i, but low-rank encoding with rank $d_h \ll d$ only requires $O(d_h d)$ parameters, which can be negligible compared to FC layers. We observe that FC layers consistently perform worse than L-conv, despite having much more parameters than L-conv. We also find that not training the L_i ("Rand L-conv") leads to significant performance drop. We ran tests on the unmodified images as well (Supp. Fig 12), where CNN performed best, but L-conv trails closely behind CNN.

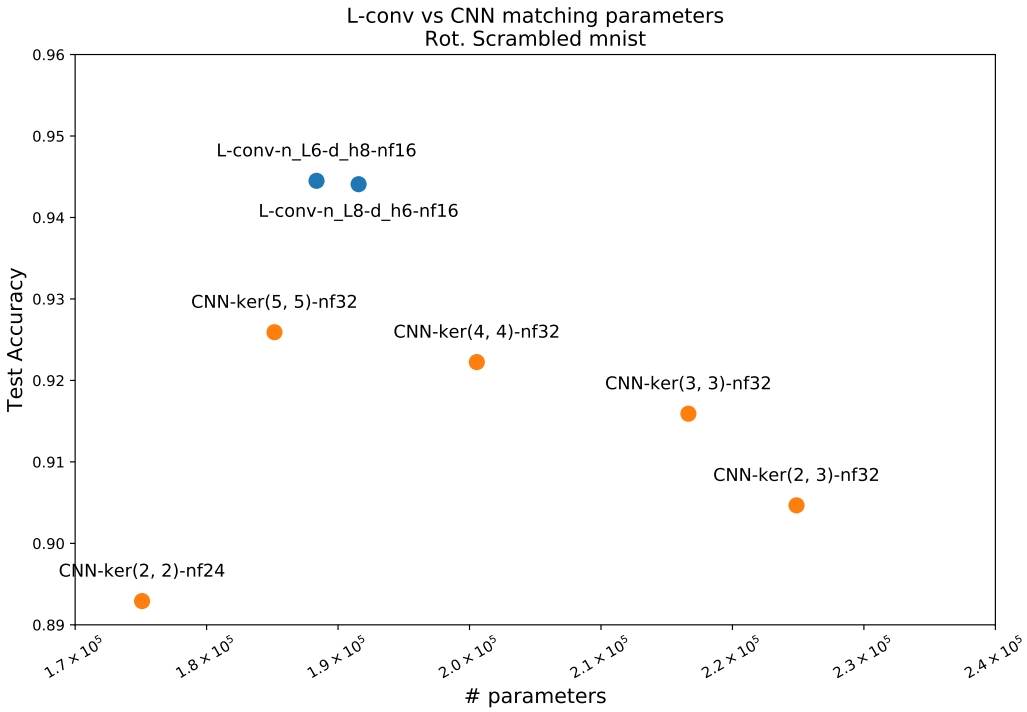

Figure 11: Matching number of parameters in CNN and L-conv, we observe that L-conv still performs better on Rotated and Scrambled MNIST.

Additional experiments testing the effect of number of layers, number of parameters and pooling are shown in Fig. 10. On CIFAR100, we find that both FC configurations, FC($\sim$L-conv) and FC(shallow) consistently perform worse than L-conv, evidence that L-conv's performance is *not* due to its extra parameters. L-conv outperforms all other tested models on rotated and scrambled CIFAR100, including LieConv. Without pooling, we observe that both L-conv and CNN do not benefit from adding a second layer. On the default CIFAR100 dataset, one and two layer CNN with max-pooling perform significantly better than L-Conv. Two Layer $SE(3)$ LieConv (labelled "2 Finzi (256)") performs best on default CIFAR100, but not on the scrambled and rotated version. This is expected, as the symmetries of the latter are masked by the scrambling. This is where the benefit of our model becomes evident, namely cases where the data may have hidden or unfamiliar symmetries. We also verified that the higher performance of L-conv compared to CNN is not due to higher number number of parameters (Appendix D.2)

D.1 Details of experiments

Hardware and Implementation We implemented L-conv in Keras and Tensorflow 2.2 and ran our tests on a system with a 6 core Intel Core i7 CPU, 32GB RAM, and NVIDIA Quadro P6000 (24GB RAM) GPU. The L-conv layer did not require significantly more resources than CNN and ran only slightly slower.

D.2 Additional Experiments

Matching number of parameters in CNN To verify that the difference in the number of parameters between CNN and L-conv was not responsible for the improved performance, we ran experiment where we allowed the kernel-size of L-conv and CNN to differ and tried to match the number of parameters between the two. Fig. 11 shows that on rotated and scrambled MNIST L-conv still performs better than CNN even after the latter has been allowed to have the same or more number of parameters than L-conv.

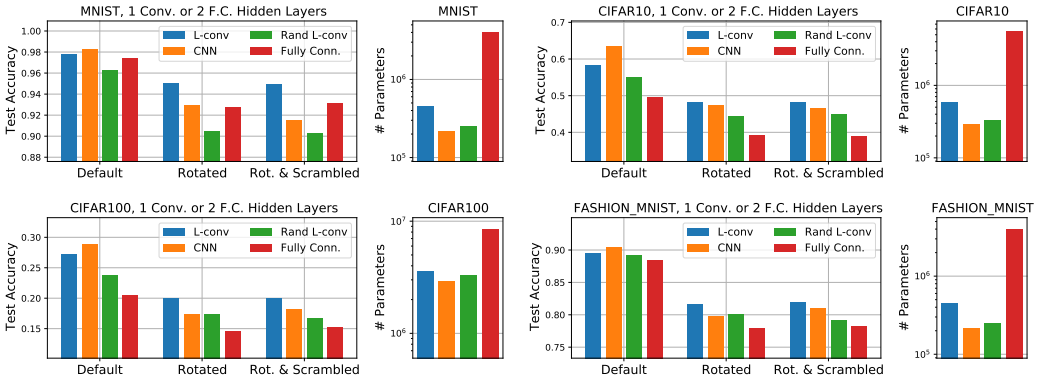

Figure 12: Test results on four datasets with three variant: "Default" (unmodified dataset), "Rotated" and "Rotated and scrambled". On the Default dataset, CNN performs best, but L-conv is always the second best. For Rotated and Rot. & Scrambled, in all cases L-conv performed best. In MNIST, FC and CNN layers come close, but using 5x more parameters.

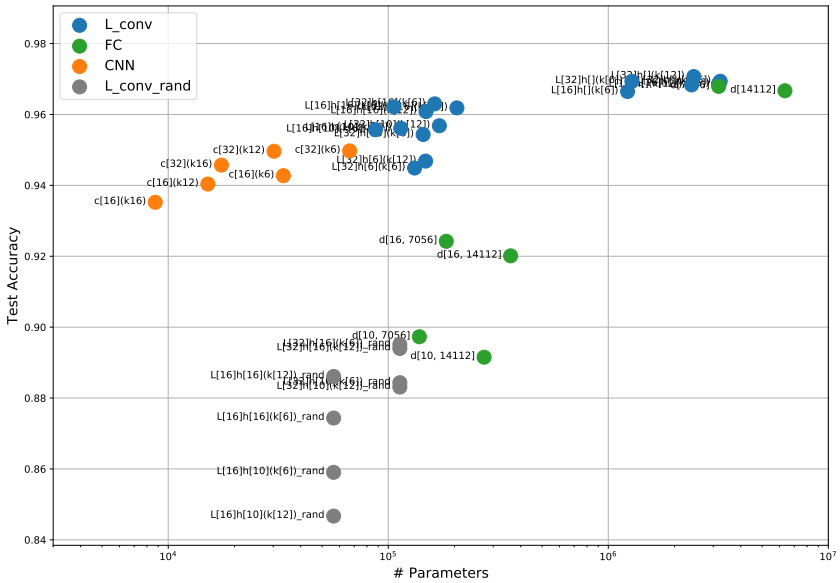

Figure 13: Training low-rank L-conv layer during training.

In Figure 13 we compare the performance of a single layer of L-conv on a classification task on scrambled rotated MNIST, where pixels have been permuted randomly and images have been rotated between -90 to $+90$ degrees. The models consisted of a final classification layer preceded by either one L-conv (blue), or one CNN (orange), or multiple fully-connected (FC, green) layers with similar number of neurons as the L-conv, but without weight sharing. We see that most L-conv configurations had the highest performance without a too many trainable parameters. Note that, parameters in FC layers are much higher than comparable L-conv, but yield worse results. The dots are labeled to show the configurations, with $L[32]h[6](k[6])$ meaning $k = 6$ as number of L_i, 32 output filters, and $h = 6$ hidden dimensions for low-rank encoding of L_i. The y-axis shows the test accuracy and the x-axis the number of trainable parameters. The grey lines show the performance of L-conv with fixed random L_i, but trainable shared wights, showing that indeed the learned L_i improve the performance quite significantly.