A Extended derivations ans 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

$$\begin{aligned} g = P\exp\left[\int_\gamma t^i(s)L_i ds\right] &= \lim_{N\to\infty}\prod_{a=1}^{N}\left(I + \delta s \gamma'^i(s_a)L_i\right) \\ &= \int_0^{s_1} ds_0 \gamma'^i(s_0)L_i \int_0^{s_2} ds_1 \gamma'^j(s_1)L_j \cdots \int_0^1 ds_N \gamma'^k(s_N)L_k \end{aligned} \tag{24}$$

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

$$\begin{aligned} [\delta_\eta \star f](g) &= \int_G dv \delta_\eta(v) f(gv) = \int_{\|\epsilon\|<\eta} dv_\epsilon \delta_\eta(v_\epsilon) f(gv_\epsilon) \\ &= \int d\epsilon \delta_\eta(I + \epsilon^i L_i) f(g(I + \epsilon^i L_i)) \\ &= \int d\epsilon \delta_\eta(I + \epsilon^i L_i) f(g + \epsilon^i g L_i) \\ &= \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\left[f(g) + \epsilon^i g L_i \cdot \frac{d}{dg} f(g) + O(\epsilon^2)\right] \\ &\approx \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\left[I + \epsilon^i g L_i \cdot \frac{d}{dg}\right] f(g) = W^0\left[I + \bar{\epsilon}^i g L_i \cdot \frac{d}{dg}\right] f(g) \end{aligned} \tag{25}$$

where $d\epsilon$ is the integration measure on the Lie algebra induced by the Haar measure dv on G. 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

$$W^0 = c\int d\epsilon \delta_\eta(I + \epsilon^i L_i) \in \mathbb{R}^{m'} \otimes \mathbb{R}^m, \qquad \bar{\epsilon}^i = \int d\epsilon \delta_\eta(I + \epsilon^i L_i)\epsilon^i \in \mathbb{R}^m \otimes \mathbb{R}^m \tag{26}$$

with $\|\bar{\epsilon}\| < \eta$. 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

$$[f(gv_\epsilon)]^a = \sum_b f^b(g(\delta^a_b + [\epsilon^i]^a_b L_i)) \tag{27}$$

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 25 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

$$\begin{aligned} Q[f](g) &= W^0\left[I + \bar{\epsilon}^i g L_i \cdot \frac{d}{dg}\right] f(g) \\ &= [W^0]_b f^a\left(g\left(\delta^b_a + [\bar{\epsilon}^i]^b_a L_i\right)\right) + O(\bar{\epsilon}^2) \end{aligned} \tag{28}$$

Extended equivariance for L-conv From equation 28 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 $u \in G$

$$u \cdot Q[f](g) = W^0 f(u^{-1} g v_\epsilon) = W^0 [u \cdot f](g v_\epsilon) \tag{29}$$

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

$$\begin{aligned} Q_\sigma[f](g) &= \sigma(W f(g v_\epsilon) + b) \\ u \cdot Q_\sigma[f](g) &= Q_\sigma[f](u^{-1} g) = \sigma(W f(u^{-1} g v_\epsilon) + b) \\ &= \sigma(W [u \cdot f](g v_\epsilon) + b) = Q_\sigma[u \cdot f](g) \end{aligned} \tag{30}$$

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

We now show that G-conv equation 3 can be approximated by composing L-conv layers.

Universal approximation for kernels Using the same argument used for neural networks (Hornik et al., 1989; Cybenko, 1989), we may approximate any kernel $\kappa(v)$ as the sum of a number of kernels κ_k with support only on a small η neighborhood of $u_k \in G$ to arbitrary accuracy. The local kernels can be written as $\kappa_k(v) = c_k \delta_\eta(u_k^{-1} v)$, with $\delta_\eta(u)$ as in equation 25 and constants $c_k \in \mathbb{R}^{m'} \otimes \mathbb{R}^m$. Using this, G-conv equation 3 becomes

$$\begin{aligned} [\kappa \star f](g) &= \sum_k [\kappa_k \star f](g) = \sum_k c_k \int dv \delta_\eta(u_k^{-1} v) f(gv) \\ &= \sum_k c_k \int dv \delta_\eta(v) f(g u_k v) = \sum_k c_k [\delta_\eta \star f](g u_k). \end{aligned} \tag{31}$$

As we showed in equation 25, $[\delta_\eta \star f](g)$ is the definition of L-conv. Next, we need to show that $[\delta_\eta \star f](g u_k)$ can also be approximated with $[\delta_\eta \star f](g)$ and hence L-conv. For this we use $u_k = v_k(I + \epsilon_k^i L_i)$ to find $v_k \in G$ which are closer to I than u_k. Taylor expanding $F_\eta = \delta_\eta \star f$ in ϵ we obtain

$$\begin{aligned} F_\eta(g u_k) &= F_\eta\left(g v_k (I + \epsilon_k^i L_i)\right) = F_\eta(g v_k) + \epsilon_k^i u L_i \cdot \frac{dF_\eta(u)}{du}\bigg|_{u \to g v_k} + O(\epsilon^2) \\ [\kappa \star f](g) &= \sum_k c_k F_\eta(g u_k) = \sum_k \left[c_k + c_k \epsilon_k^i u L_i \cdot \frac{d}{du}\right] F_\eta(u)\bigg|_{u \to g v_k} \\ &= \sum_k \left[W_k^0 + W_k^i u L_i \cdot \frac{d}{du}\right] F_\eta(u)\bigg|_{u \to g v_k} = \sum_k Q_k[F_\eta](g v_k) \end{aligned} \tag{32}$$

Using equation 32 we can progressively remove the u_k as $F_\eta(g u_k) \approx Q_k^n[\dots[Q_k^1[F_\eta]]](g)$, i.e. an n layer L-conv. Thus, we conclude that any G-conv equation 3 can be approximated by multilayer L-conv.

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 $\boldsymbol{x}^\rho = g_\sigma^\rho \boldsymbol{x}_0^\sigma$, we have

$$\frac{df(g\boldsymbol{x}_0)}{dg_\beta^\alpha} = \frac{d(g_\sigma^\rho \boldsymbol{x}_0^\sigma)}{dg_\beta^\alpha} \partial_\rho f(\boldsymbol{x}) = \boldsymbol{x}_0^\beta \partial_\alpha f(\boldsymbol{x}) \tag{33}$$

$$gL_i \cdot \frac{df}{dg} = [gL_i]_\beta^\alpha \boldsymbol{x}_0^\beta \partial_\alpha f(\boldsymbol{x}) = [gL_i \boldsymbol{x}_0] \cdot \nabla f(\boldsymbol{x}) = \hat{L}_i f(\boldsymbol{x}) \tag{34}$$

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 8, 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

$$Q[f](w^{-1}g\boldsymbol{x}_0) = W^0 \left[I + \bar{\epsilon}^i [\hat{L}_i]^\alpha [w^{-1}]^\beta_\alpha \frac{\partial}{\partial \boldsymbol{y}^\beta} \right] f(\boldsymbol{y})\big|_{\boldsymbol{y} \to w^{-1}\boldsymbol{x}} \tag{35}$$

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)

$$L = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}, \qquad g = \exp[\theta L] = \frac{1}{r}\begin{pmatrix} x & -y \\ y & x \end{pmatrix}, \qquad gL \cdot \frac{df}{dg} = (x\partial_y - y\partial_x) f. \tag{36}$$

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 36). $\mathbb{R}^2/0$ 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 34, $gL_2 \cdot df/dg = (x\partial_x + y\partial_y)f = r\partial_r f$, which is the scaling operation.

A.2.1 Rotation $SO(2)$

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

$$g = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, = \frac{1}{r}\begin{pmatrix} x & -y \\ y & x \end{pmatrix}, \quad L = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad gL = \frac{1}{r}\begin{pmatrix} -y & -x \\ x & -y \end{pmatrix} \tag{37}$$

$$gL\boldsymbol{x}_0 = \begin{pmatrix} -y \\ x \end{pmatrix} = \begin{pmatrix} -\sin\theta \\ \cos\theta \end{pmatrix} \tag{38}$$

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 10, we have

$$\frac{df(g\boldsymbol{x}_0)}{dg} = \frac{1}{r}\boldsymbol{x}_0^T \begin{pmatrix} \partial_x f & -\partial_y f \\ \partial_y f & \partial_x f \end{pmatrix} = \begin{pmatrix} \partial_x f & -\partial_y f \\ 0 & 0 \end{pmatrix}$$
$$gL \cdot \frac{df}{dg} = \mathrm{Tr}\begin{pmatrix} x\partial_y f - y\partial_x f & x\partial_x f + y\partial_y f \\ 0 & 0 \end{pmatrix} = (x\partial_y f - y\partial_x f) \tag{39}$$

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 $I = \int_G dg F(g)$ is G-invariant as for $w \in G$

$$w \cdot I = \int_G w \cdot F(g) dg = \int F(w^{-1}g) dg = \int_G F(g') d(wg') = \int_G F(g') dg' \tag{40}$$

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

$$I = \int_G F(g) dg = \int_H dh \int_{G/H} dg' F(g') = V_H \int_{G/H} dg' F(g') \tag{41}$$

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 $\Phi = (\phi_1, \dots, \phi_N)^T$ is an $(m + m_y) \times N$ matrix. Using equations 6 and 10, the MSE loss with parameters $W = \{W^0, \overline{\epsilon}\}$ becomes

$$\begin{aligned} I[\Phi; W] &= \int_G dg \mathcal{L}[\Phi; W] = \int_G dg \left\| W^0 \left[I + \overline{\epsilon}^i [\hat{L}_i]^\alpha \partial_\alpha \right] \Phi(g) \right\|^2 \\ &= 2 \int_G dg \left[\|W^0 \Phi\|^2 + \left\| W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi \right\|^2 + 2 \Phi^T W^{0T} W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi \right] \qquad (42) \\ &= \int_{\mathcal{S}} \frac{d^n x}{\left| \frac{\partial x}{\partial g} \right|} \left[\Phi^T \mathbf{m}_2 \Phi + \partial_\alpha \Phi^T \mathbf{h}^{\alpha\beta} \partial_\beta \Phi + [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T \mathbf{v}^i \Phi \right) \right] \qquad (43) \end{aligned}$$

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

$$\mathbf{m}_2 = W^{0T} W^0, \qquad \mathbf{h}^{\alpha\beta}(\boldsymbol{x}) = \overline{\epsilon}^{iT} \mathbf{m}_2 \overline{\epsilon}^j [\hat{L}_i]^\alpha [\hat{L}_j]^\beta, \qquad \mathbf{v}^i = \mathbf{m}_2 \overline{\epsilon}^i. \tag{44}$$

From equation 42 to 43 we used the fact that W^0 and W^i do not depend on $\boldsymbol{x}$ (or g) to write

$$2 \Phi^T W^{0T} W^i [\hat{L}_i]^\alpha \partial_\alpha \Phi = [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T W^{0T} W^i \Phi \right) = [\hat{L}_i]^\alpha \partial_\alpha \left(\Phi^T \mathbf{m}_2 \overline{\epsilon}^i \Phi \right) \tag{45}$$

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 10, $[\hat{L}_i(v\boldsymbol{x})]^\alpha = v^\alpha_\beta \hat{L}_i(\boldsymbol{x})^\beta$ and

$$v \cdot \mathbf{h}^{\alpha\beta} = \mathbf{h}^{\alpha\beta}(v^{-1}\boldsymbol{x}) = [v^{-1}]^\alpha_\rho [v^{-1}]^\beta_\gamma \mathbf{h}^{\rho\gamma}(\boldsymbol{x}), \qquad (v \in G). \tag{46}$$

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

$$\mathbf{h}^{\alpha\beta}_{ab} = [\overline{\epsilon}^T \mathbf{m}_2 \overline{\epsilon}]_{ab} \otimes [\hat{L} \hat{L}^T]^{\alpha\beta} \tag{47}$$

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 38. Therefore, $R(\xi)^{-1}\hat{L}=R(\theta-\xi)=\hat{L}(R(\xi^{-1}\boldsymbol{x})$. Using equation 20 in equation 46, the transformed metric becomes

$$R(\xi)\cdot\mathbf{h}^{\alpha\beta}(R(\theta)\boldsymbol{x}_0)=\overline{\epsilon}^T\mathbf{m}_2\overline{\epsilon}\otimes[R(-\xi)\hat{L}]^\alpha[R(-\xi)\hat{L}]^\beta=\mathbf{h}^{\alpha\beta}(R(\theta-\xi)\boldsymbol{x}_0), \tag{48}$$

A.3.2 Third term as a boundary term

Since terms in equation 19 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 19. 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 $[\hat{L}_i]^\alpha=\delta_i^\alpha$ (SI A.2.2) and $|\partial g/\partial x|=1$ and $dg=d^nx$. Thus, the last term in equation 19 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 19 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 19 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 $[\hat{L}_i]^\alpha=\delta_i^\alpha$ (SI A.2.2), the last term in equation 19 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 19 generally becomes a vanishing boundary term and does not matter.

Next, the second term in equation 19 can be worked out as

$$\hat{L}_i^\alpha\partial_\alpha\phi^T\overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j\hat{L}_j^\beta\partial_\beta\phi=\partial_j\phi^T\overline{\epsilon}^{iT}\mathbf{m}_2\overline{\epsilon}^j\partial_i\phi=\partial_j\phi^T\mathbf{h}^{ji}\partial_i\phi \tag{49}$$

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 $[W^i]_b^a\sim\mathcal{N}(0,1)$ 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 19 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

$$d\Sigma_\beta[gL_\alpha\boldsymbol{x}_0]^\beta=d\Sigma^TgL_\alpha\boldsymbol{x}_0=[g^Td\Sigma]^TL_\alpha\boldsymbol{x}_0 \tag{50}$$

Since $g\in SO(n)$, $g^T=g^{-1}$ and $g^Td\Sigma(g\boldsymbol{x}_0)=d\Sigma(\boldsymbol{x}_0)=V_{n-1}\boldsymbol{x}_0$, 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

$$d\Sigma^TgL_i\boldsymbol{x}_0=\boldsymbol{x}_0^TL_i\boldsymbol{x}_0=0 \tag{51}$$

for all generators $L_i\in so(n)$ because $L_i=-L_i^T$ and hence diagonal entries like $\boldsymbol{x}_0^TL_i\boldsymbol{x}_0$ are zero. Only the scaling generator $L_0=I$ we have $\boldsymbol{x}_0^TL_0\boldsymbol{x}_0=1$. This means that the last term in equation 19 can be nonzero at the boundary only if $\Phi\mathbf{v}^i\Phi$ is in the radial direction, meaning $\overline{\epsilon}^0=0$, 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 Generalization error

Equivariant neural networks are hoped to be more robust than others. Equivariance should improve generalization. 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

$$\delta I[\phi; W^*] = \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^a} \delta\phi^a + \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^a)} \partial_\alpha(\delta\phi^a) \right] \tag{52}$$

Doing a partial integration on the second term, we get

$$\begin{aligned} \delta I[\phi; W^*] &= \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \right] \delta\phi^b + \int_{\mathcal{S}} d^n x \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b \right] \\ &= \int_{\mathcal{S}} d^n x \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \right] \delta\phi^b + \int_{\partial\mathcal{S}} d^{n-1} \Sigma_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b \right] \end{aligned} \tag{53}$$

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 53 is the classic Euler-Lagrange (EL) equation. Thus, requiring good generalization, i.e. $\delta I[\phi; W^*]/\delta\phi = 0$ means for optimal parameters W^*, the real data ϕ satisfies the EL equations

$$\text{Generalization Error Minimization} \iff \text{EL:} \quad \frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} = 0 \tag{54}$$

Applying this to the MSE loss equation 19, equation 54 becomes

$$\mathbf{m}_2 \phi - \partial_\alpha \left(|J| \mathbf{h}^{\alpha\beta} \partial_\beta \phi \right) - \partial_\alpha \left(|J| \mathbf{v}^i [\hat{L}_i]^\alpha \right) \phi = 0 \tag{55}$$

where $|J| = |\partial g / \partial x|$ is the determinant of the Jacobian. For the translation group, equation 55 becomes a Helmholtz equation

$$\mathbf{h}^{ij} \partial_i \partial_j \phi = \bar{\epsilon}^i \mathbf{m}_2 \bar{\epsilon}^j \partial_i \partial_j \phi = \mathbf{m}_2 \phi \tag{56}$$

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

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

$$\begin{aligned} w \cdot \phi(\boldsymbol{x}) &= \phi(w^{-1}\boldsymbol{x}) = \phi((I - \eta^i L_i)\boldsymbol{x}) = \phi(\boldsymbol{x}) - \eta^i [L_i \boldsymbol{x}]^\alpha \partial_\alpha \phi(\boldsymbol{x}) \\ &= \phi(\boldsymbol{x}) + \delta\boldsymbol{x}^\alpha \partial_\alpha \phi(\boldsymbol{x}) = \phi(\boldsymbol{x}) + \delta\phi(\boldsymbol{x}) \end{aligned} \tag{57}$$

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 $\mathcal{L}[w \cdot \phi]$.

$$\begin{aligned} \mathcal{L}[w \cdot \phi] &= \mathcal{L}[\phi(\boldsymbol{x}) + \delta\phi(\boldsymbol{x})] = \mathcal{L}[\phi] + \frac{\partial \mathcal{L}}{\partial \phi^b} \delta\phi^b + \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\partial_\alpha \phi^b \\ &= \mathcal{L}[\phi] + \left[\frac{\partial \mathcal{L}}{\partial \phi^b} - \partial_\alpha \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \right] \delta\phi^b + \partial_\alpha \left(\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b \right) \end{aligned} \tag{58}$$

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 55. This means that the first term in equation 58 vanishes. Setting the second term equal to $w \cdot \mathcal{L}$ we get

$$\mathcal{L}[w \cdot \phi] - w \cdot \mathcal{L}[\phi] = \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)} \delta\phi^b - \delta\boldsymbol{x}^\alpha \mathcal{L} \right] = 0 \tag{59}$$

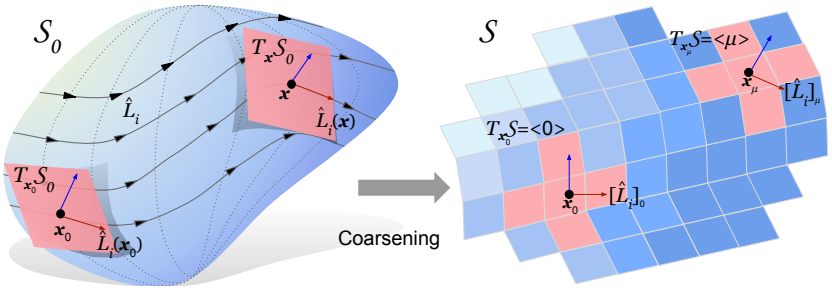

Figure 5: **Manifold vs. discretized Space** While real systems can have a continuous manifold $\mathcal{S}_0$ as their base space, often the data collected from is a discrete array $\mathcal{S}$. The discretization (coarsening) will induce some of the topology of $\mathcal{S}_0$ on $\mathcal{S}$ as 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.

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

$$\text{Noether current: } J^\alpha = \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b - \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}^\alpha}\delta\boldsymbol{x}^\alpha, \qquad \delta I[\phi; W^*] = 0 \quad \Rightarrow \partial_\alpha J^\alpha = 0 \tag{60}$$

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 57 we find

$$\partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\delta\phi^b - \delta\boldsymbol{x}^\alpha \mathcal{L} \right] = \delta\boldsymbol{x}^\beta \partial_\alpha \left[\frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\partial_\beta \phi^b - \delta^\alpha_\beta \mathcal{L} \right] = \delta\boldsymbol{x}^\beta \partial_\alpha T^\alpha_\beta. \tag{61}$$

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

$$T^\alpha_\beta \equiv \frac{\partial \mathcal{L}}{\partial(\partial_\alpha \phi^b)}\partial_\beta \phi^b - \delta^\alpha_\beta \mathcal{L} = \partial_\rho \phi^T \left(\delta^\lambda_\beta \mathbf{h}^{\alpha\rho} - \delta^\alpha_\beta \mathbf{h}^{\rho\lambda} \right) \partial_\lambda \phi - \phi^T \mathbf{m}_2 \phi \tag{62}$$

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

$$w \cdot f(\boldsymbol{x}_\mu) = f(w^{-1} \cdot \boldsymbol{x}_\mu) = \boldsymbol{x}_\mu^T w^{-1T} \boldsymbol{f} = [w^{-1}\boldsymbol{x}]^T \boldsymbol{f} \tag{63}$$

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

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

$$[\kappa \star f](g\boldsymbol{x}_0) = \int_G \kappa(v) f(gv\boldsymbol{x}_0) dv = \boldsymbol{x}_0^T \int_G v^T g^T \boldsymbol{f} \kappa^T(v) dv \equiv \boldsymbol{x}_0^T [\boldsymbol{f} \star \kappa](g) \tag{64}$$

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 64 is readily checked with $w \in G$

$$w \cdot [\boldsymbol{f} \star \kappa](g) = [\boldsymbol{f} \star \kappa](w^{-1}g) = \int_G v^T g^T w^{-1T} \boldsymbol{f} \kappa^T(v) dv = [(w \cdot \boldsymbol{f}) \star \kappa](g) \tag{65}$$

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 $v_\epsilon = I + \overline{\epsilon}^i L_i$

$$\begin{aligned} Q[\boldsymbol{f}](g) &= W^0 f\left(g\left(I + \overline{\epsilon}^i L_i\right)\right) = \boldsymbol{x}_0^T \left(I + \overline{\epsilon}^i L_i\right)^T g^T \boldsymbol{f} W^{0T} \\ &= \left(\boldsymbol{x}_0 + \overline{\epsilon}^i [g L_i \boldsymbol{x}_0]\right)^T \boldsymbol{f} W^{0T}. \end{aligned} \tag{66}$$

Here, $\hat{L}_i = g L_i \boldsymbol{x}_0$ is eactly the tensor analogue of pushforward vector field $\hat{L}_i$ in equation 10. 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

$$Q[w \cdot \boldsymbol{f}](g) = \boldsymbol{x}_0^T v_\epsilon^T g^T w^{-1T} \boldsymbol{f} W^{0T} = Q[\boldsymbol{f}](w^{-1}g) = w \cdot Q[\boldsymbol{f}](g) \tag{67}$$

Next, we will discuss how to implement equation 66 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 66 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 66.

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 5.

Neighborhoods as discrete tangent bundle $\boldsymbol{B}$ is useful for extending differential geometry to graphs (Schaub et al., 2020). Define the neighborhood $< \mu >= \left\{\alpha \in \mathcal{E} \middle| \boldsymbol{B}_\alpha^\mu = -1\right\}$ 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

$$C \in < \mu >: \qquad C\boldsymbol{f} = \sum_{\alpha \in < \mu >} \tilde{C}^\alpha \boldsymbol{B}_\alpha^\nu \boldsymbol{f}_\nu \tag{68}$$

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 $[L_i\boldsymbol{x}_0]^T\boldsymbol{f} = [\hat{L}_i]_0^\rho \boldsymbol{f}_\rho = \sum_{\alpha\in<0>}[\hat{\ell}_i]_0^\alpha \boldsymbol{B}_\alpha^\rho \boldsymbol{f}_\rho$. The coefficients $[\hat{\ell}_i]_0^\alpha \in \mathbb{R}$ are in fact the discrete version of $\hat{L}_i$ components from equation 10. 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 $[\hat{L}_i]_\mu^\nu \in \mathbb{R}$ we have

$$[g_\mu(I+\eta L_i)\boldsymbol{x}_0]^T\boldsymbol{f} = \boldsymbol{f}_\mu + \eta \sum_{\alpha\in<\mu>} [\hat{\ell}_i]_\mu^\alpha \boldsymbol{B}_\alpha^\nu \boldsymbol{f}_\nu \tag{69}$$

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

$$\begin{aligned}[\hat{L}_i]_\mu^\nu = [\hat{\ell}_i]_\mu^\alpha \boldsymbol{B}_\alpha^\nu = [g_\mu L_i\boldsymbol{x}_0]^\nu &= \sum_\rho [g_\mu \boldsymbol{x}_\rho \boldsymbol{x}_\rho^T L_i \boldsymbol{x}_0]^T \boldsymbol{x}_\nu \\ &= \sum_{\rho\in<0>} [L_i]_0^\rho [\boldsymbol{x}_\nu^T g_\mu \boldsymbol{x}_\rho]^T = [L_i]_0^\rho [g_\mu]_\rho^\nu = [\hat{\ell}_i]_0^\alpha \boldsymbol{B}_\alpha^\rho [g_\mu]_\rho^\nu \end{aligned} \tag{70}$$

This $\hat{L}_i \equiv \hat{\ell}_i^\alpha \boldsymbol{B}_\alpha$ 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 10.

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 $\boldsymbol{f}'_\mu = g(z)_\mu^\nu \boldsymbol{f}_\nu$, where $g(z)_\mu^\nu = \frac{1}{d}\sum_{p=-d/2}^{d/2} \cos\left(\frac{2\pi p}{d}(z+\mu-\nu)\right)$ 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.

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 17 (equation 70) to ensure that there is an underlying shared generator $\hat{\ell}_i$ for all μ which is pushed forward using g_μ, shared

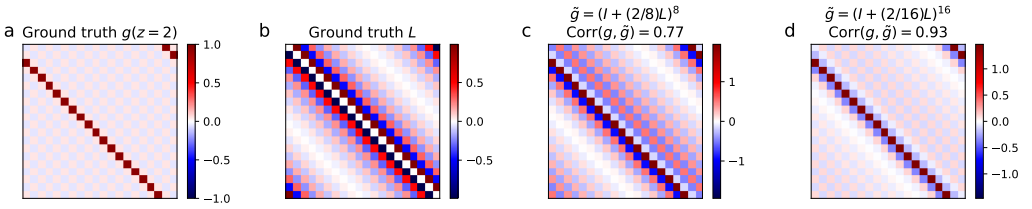

Figure 6: 1D Translation: Using the Shannon-Whittaker Interpolation (SWI) one can generate continuous shifts on discrete data. These include integer shifts (a, ground truth). (SWI) also yields an infinitesimal generator L for shifts (b). This L can be used to approximate finite shifts using $\tilde{g}_n(z)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

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 17, 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.

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 $\boldsymbol{f}'_\rho = g(z)_\rho^\nu \boldsymbol{f}_\nu$. Here $g(z)_\rho^\nu = \frac{1}{d}\sum_{p=-d/2}^{d/2} \cos\left(\frac{2\pi p}{d}(z + \rho - \nu)\right)$ 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 $g(0)_\rho^\nu = \delta_\rho^\nu$. For any $z = \mu \in \mathbb{Z}$, $g_\mu = g(z = \mu)$ are circulant matrices that shift by μ as $[g_\mu]_\nu^\rho = \delta_{\nu-\mu}^\rho$. g_μ can be approximated using the Lie algebra and written as multi-layer L-conv as in sec. 3.1. Using $g(0)_\rho^\nu \approx \delta(\rho - \nu)$, 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 $\hat{L}_\rho^\nu = L(\rho - \nu) = \sum_p \frac{2\pi p}{d^2} \sin\left(\frac{2\pi p}{d}(\rho - \nu)\right)$, 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.

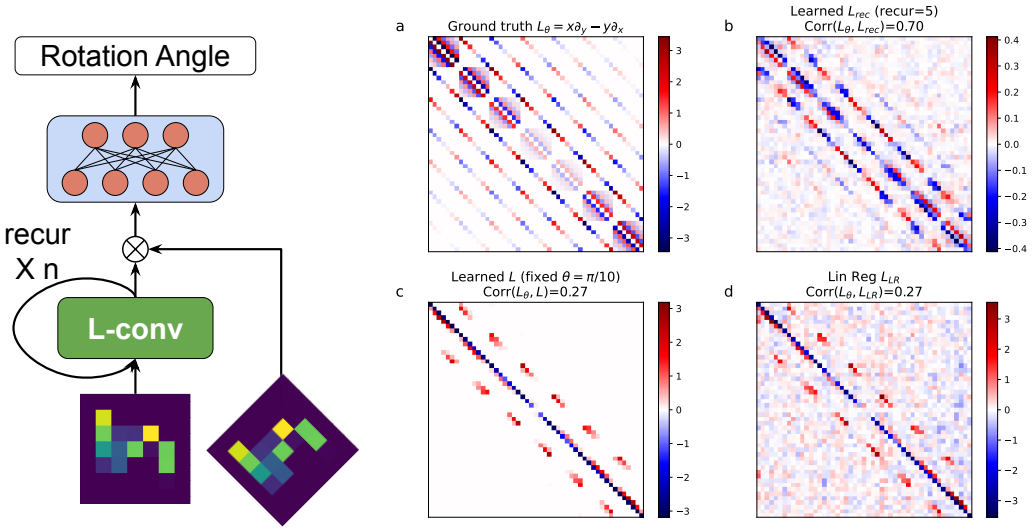

Figure 7: **Learning the infinitesimal generator of** $SO(2)$ Left shows the architecture for learning rotation angles between pairs of images. (a) shows the rotation generator calculated analytically using Shannon-Whittaker interpolation. (b) is the $\hat{L}$ learned using recursive L-conv learning 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.

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 $\text{Corr}(g, \tilde{g}) = \text{Tr}\left[g^T\tilde{g}\right]/(\|g\|\|\tilde{g}\|)$ 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 $\hat{L}$ 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 $X = (\boldsymbol{f}_1, \ldots \boldsymbol{f}_N)$ and $Y = (\boldsymbol{f}'_1, \ldots \boldsymbol{f}'_N)$ be the matrix of all inputs and outputs, respectively. The rotation equation is $Y^T = RX^T$. Thus, the rotation matrix is given by $R = (Y^TX)(X^TX)^{-1}$. 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.

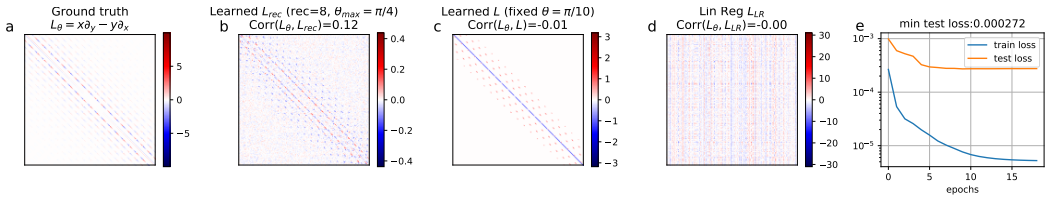

Figure 8: Learning $\hat{L}$ via larger rotation angles for larger images. This time the correlation with ground truth L_θ is much less, but accuracy is still very good.

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 $\overline{\epsilon}$ 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.

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.

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.

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

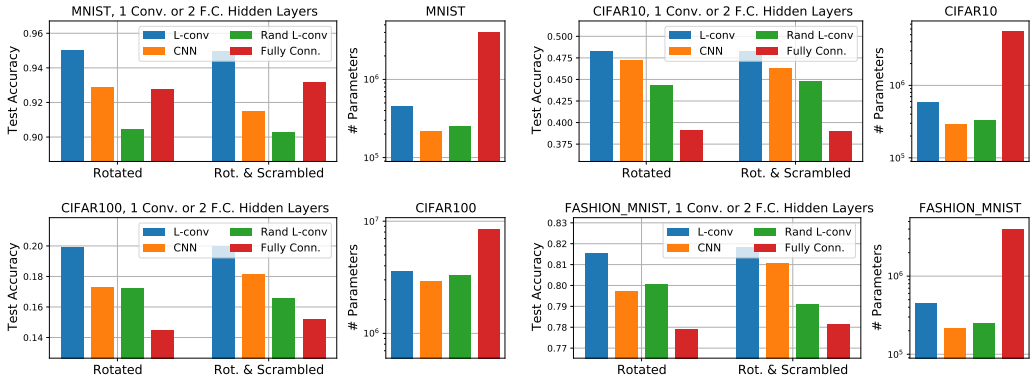

Figure 9: Results on four datasets with two variant: "Rotated" and "Rotated and scrambled". In all cases L-conv performs best. On MNIST, FC and CNN come close, but using 5x more parameters.

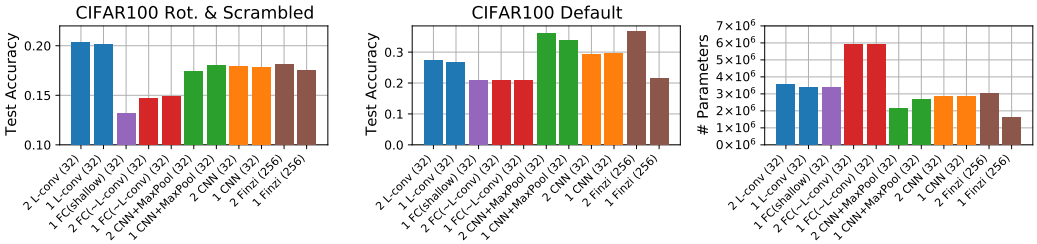

Figure 10: Comparison of one and two layer performance of L-conv (blue), CNN without pooling (orange), CNN with Maxpooling after each layer (green), fully connected (FC) with structure similar to L-conv (red), Lie-Conv (Finzi) Finzi et al. (2020) (brown), and shallow FC, which has a single hidden layer 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.

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.

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.1.1 Comparison with related models

Comparison with Meta-learning Symmetries by Reparameterization (MSR) 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.

Contrast with Augerino In a concurrent work, (Benton et al., 2020) propose Augerino, a method to learn invariances with neural networks. Augerino uses data augmentation to transform the input data, which means it is restricting the group to be a subgroup of the augmentation transformations. 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. However, in Augerino, L_i are fixed *a priori* to be the six generators of affine transformations in 2D (translations, rotations, scaling and shearing). In contrast, our approach is more general. We learn the generators L_i directly without restricting them to be a known set of generators. Additionally, we do not use the exponential map, hence, implementing L-conv is very straightforward.

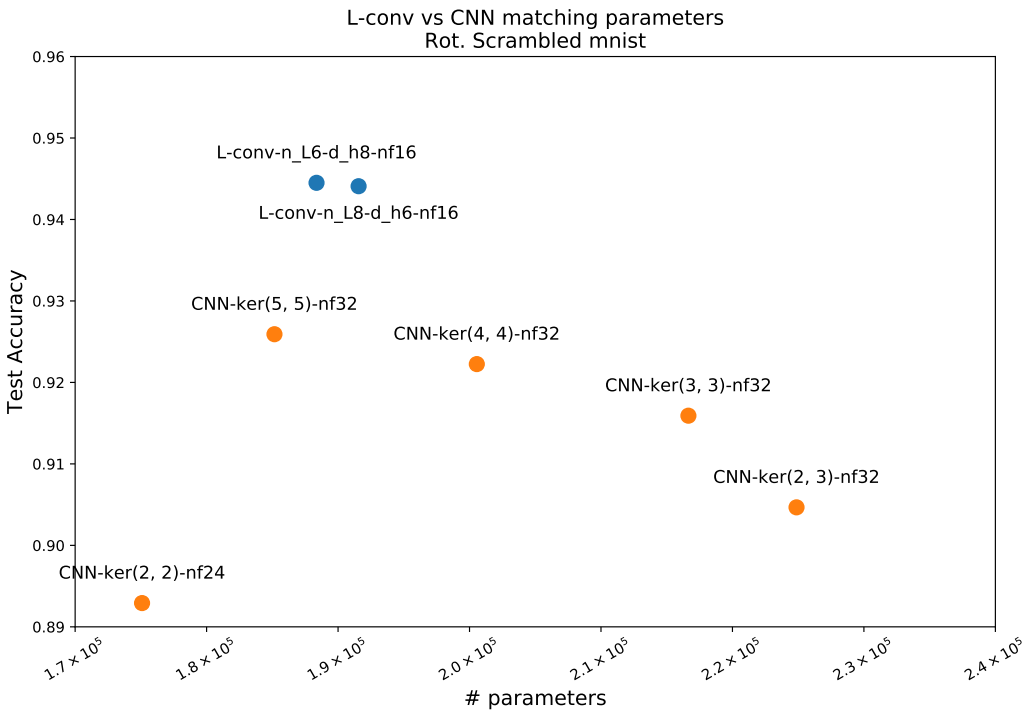

Figure 11: Matching number of parameters in CNN and L-conv, we observe that L-conv still performs better on Rotated and Scrambled MNIST.

Lastly, Augerino uses sampling to effectively cover the space of group transformations. Since the sum over Lie algebra generators is tractable, we do not need to use sampling.

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.

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.

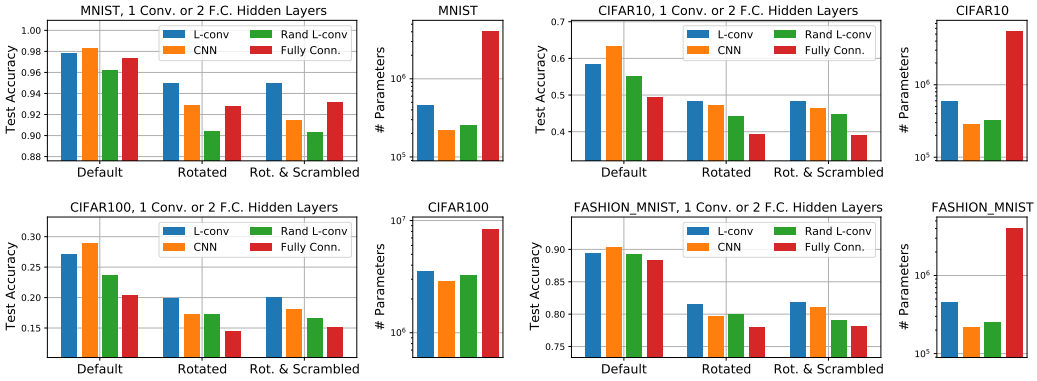

Figure 12: Test results on four datasets with three variant: "Default" (unmodified dataset), "Rotated" and "Rotated and scrambled". On the Default dataset, CNN performs best, but L-conv is always the second best. For Rotated and Rot. & Scrambled, in all cases L-conv performed best. In MNIST, FC and CNN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

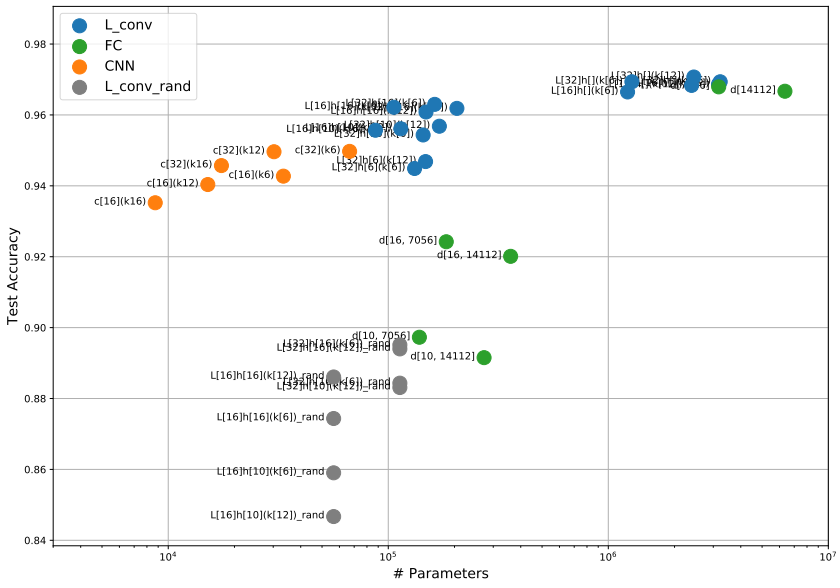

Figure 13: Training low-rank L-conv layer during training.