LIETRANSFORMER: EQUIVARIANT SELF-ATTENTION FOR LIE GROUPS

ECE 594N

George Hulsey

March 6, 2024

TABLE OF CONTENTS

1. Lie groups & homogeneous spaces
2. LieConv: group-equivariant convolutions
3. Review: multi-head attention
4. Self-attention for Lie groups
5. LieTransformer: particle dynamics

INTRODUCTION

We'll review "LieTransformer: Equivariant Self-Attention for Lie Group" **[Hutchinson et al, 2021]**.

The model is a transformer, using multi-head attention to determine the kernel of an equivariant convolution layer (LieConv, **[Finzi et al, 2020]**).

I will try to focus on how these concepts are applied to Lie groups.

In general, the method is a generalization of CNNs, but with local information (filters) captured (constructed) by the attention mechanism.

RELATED WORKS

Equivariant maps with/without lifting: Equivariant NNs without lifting need special parametrizations of kernels **[Cohen & Welling, 2017; Kondor et al., 2018, Fuchs et al., 2020]**. Non-lifting methods include the "steerable" CNNs we discussed earlier in the course.

Equivariant self-attention: equivariant convolutions are known to be the unique bounded form of equivariant linear layers **[Kondor & Trivedi, 2018; Cohen et al., 2019]**. The idea here is to get rid of convolutions as was done for image processing **[Parmar et al., 2019a; Dosovitsky et al., 2020]**.

[Romero & Cordonnier, 2021] consider a similar approach to this work but only for discrete groups and only for image processing tasks.

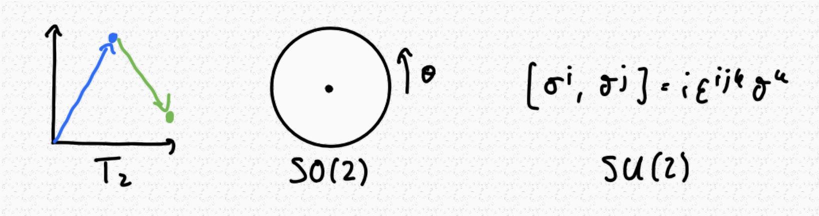
Lie groups & homogeneous spaces

GROUPS

Recall that a group G is a set of elements $\{g\}$ closed under composition and inverses.

Lie groups are continuous groups–they have infinitely many elements. Since they are continuous, we can view them as **manifolds**.

Familiar examples of Lie groups are $T_2 \cong \mathbb{R}^2$, $SO(2) \cong U(1) \cong S^1$, and $SU(2) \cong S^3$.



GROUP REPRESENTATIONS

Groups often act on objects. A **representation** of a group G is a map $\rho : G \to V$ for some vector space V. G acts on vectors $v \in V$ by matrices $\rho(g)$.

To be concrete, let $G = SO(2)$, the space of 2d rotations: elements $g_\theta \in G$ rotate by an angle θ. The following constitute representations of G:

1. $V = \mathbb{C} \quad \Longrightarrow \quad \rho(g_\theta) = e^{i\theta}$
2. $V = \mathbb{R}^2 \quad \Longrightarrow \quad \rho(g_\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$
3. $V = \mathbb{R}^3 \quad \Longrightarrow \quad \rho(g_\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 0 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$

GROUP EQUIVARIANCE

Let $f : S \to V$ map from some input space S to some feature space V; let G act on S, V by ρ_S, ρ_V. "Equivariance" means the following diagram commutes (you can go either way and end up in the same place!):

$$\begin{tikzcd}
\text{input space} & S \arrow[r, "f"] \arrow[d, "\rho_s"'] & V \arrow[d, "\rho_V"] & \text{feature space} \\
\text{transformed input} & \tilde{S} \arrow[r, "f"'] & \tilde{V} & \text{transformed features}
\end{tikzcd}$$

HOMOGENEOUS SPACES

We are given data in the form (x_i, f_i) for coordinates $x_i \in \mathcal{X}$ and features $f_i \in \mathcal{F}$. Here, we will assume that $\mathcal{X}$ is **homogeneous**.

Intuitively, homogeneous spaces “look the same everywhere”. Examples include $\mathbb{R}^n, S^n, SU(N)$, etc.

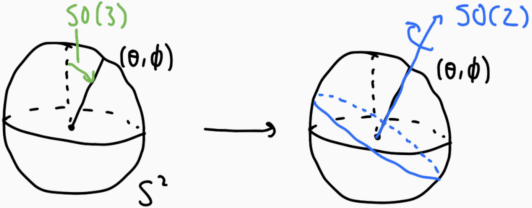
Formally, a space $\mathcal{X}$ is G-homogeneous if we can connect any two points by a group transformation:

$$\forall x, x' \in \mathcal{X} : \exists g \in G \quad \text{s.t.} \quad x' = gx \tag{1}$$

The space $\mathbb{R}^2$ is $T_2,\ \mathbb{R}^* \times SO(2)$ homogeneous: from the origin, any point x is a translation away. Any two points are connected by a rotation and a radial rescaling.

HOMOGENEOUS SPACES

A nice fact is that G-homogeneous spaces can be written as a quotient: $\mathcal{X} = G/H$ for some subgroup H. Example: the sphere S^2 is $SO(3)$-homogeneous: any two points are related by a 3d rotation **plus** an arbitrary 2d rotation about the axis of the point:



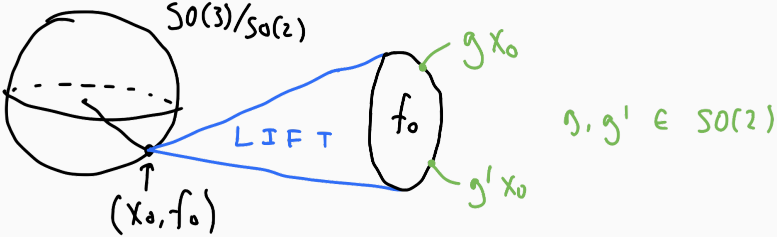
"$SO(2)$ stabilizes $(\theta,\phi) \in S^2$"

The conclusion is that $S^2 \cong SO(3)/SO(2)$. We call $SO(2)$ the **stabilizer** of $x \in S^2$.

THE LIFTING MAP

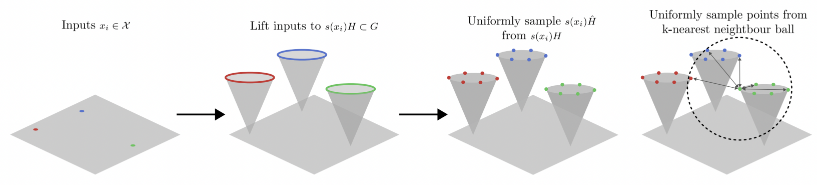
Let $\mathcal{X} = S^2$ and $f_{\mathcal{X}} : \mathcal{X} \to \mathcal{F}$ be our data (i.e. feature map). Though $f_{\mathcal{X}}$ is only defined on $SO(3)/SO(2)$, we can **lift** $f_{\mathcal{X}}$ to become a function $\mathcal{L}[f_{\mathcal{X}}]$ on $G = SO(3)$ simply by taking

$$(x_i, f_i) \longmapsto \{(gx_i, f_i) \mid g \in \mathsf{stab}(x_i) \cong SO(2)\} \tag{2}$$



The lifting map is intuitively saying: "$SO(3)/SO(2) \times SO(2) = SO(3)$"

LIFTING: VISUALIZED


Inputs $x_i \in \mathcal{X}$
Lift inputs to $s(x_i)H \subset G$
Uniformly sample $s(x_i)\hat{H}$ from $s(x_i)H$
Uniformly sample points from k-nearest neightbour ball

LieConv: group-equivariant convolutions

GROUP CONVOLUTIONS

We used the group properties of Lie groups to talk about homogeneous spaces. Now, we'll use their properties as manifolds to define **convolutions**.

Let $k : G \to \mathbb{R}$ be a function on a Lie group G; this will be our kernel, and let $f_G : G \to \mathcal{F}$ be a (lifted) feature map. Then we can define the G-**equivariant convolution**

$$(k \star f)(g) \equiv \int_G k(g'^{-1}g)f(g')dg' \tag{3}$$

Here we integrate over G with measure dg'. The measure is called the Haar measure, and every Lie group has one. This is equivariant "by construction".

LIFTS & CONVOLUTIONS

This is why we need a lifting map: it's easy to define convolutions on groups G, but not as easy to define them on spaces $\mathcal{X} = G/H$. So we use the lifting map to take our features from being defined on $\mathcal{X}$ to being defined on G: we map $G/H \to G$ in the "obvious" way.

To be completely concrete with group convolutions, consider $G = \mathbb{R}$. Then

$$(k \star f)(g) = \int_G k(g'^{-1}g) f(g') dg' \tag{4}$$

$$= \int_{\mathbb{R}} k(x-t) f(t) dt \tag{5}$$

where $g' = t,\ g'^{-1} = -t, dg' = dt$, and we have a regular convolution.

LIECONV

The idea of LieConv is straightforward: we learn the convolution kernel/filter $k_\theta(\cdot)$ by a fully-connected NN with swish (smooth) activations (actually we learn $\log k_\theta$, a naturally linear object).

However, we usually want filters to be local. Thankfully, we have measures and metrics on Lie groups G so we can restrict the integral to a **neighborhood** $N(g)$ of $g \in G$:

$$\text{LieConv}_\theta[f](g) = \int_{N(g)} k_\theta(g'^{-1}g) f(g') dg' \tag{6}$$

In practice, we do these integrals by Monte Carlo sampling over a bunch of points.

LieCNNs

From here, we can define CNNs on arbitrary Lie groups. We learn the filters which extract local information from feature space. Standard CNNs are a special case where $G = \mathbb{R}^n$ (usually $n = 2, 3$).

However, we have seen that transformer architectures can integrate context in a different way from local filters in a convolution: via the attention mechanism.

We'll quickly review the attention mechanism and consider how it can be used to extract local context in a Lie group. Later, we'll compare the performance of LieConv architectures vs LieTransformer architectures on a 'particle physics' task.

Review: multi-head attention

ATTENTION IN A NUTSHELL

I'm not a computer scientist, so forgive me. Generally speaking, **attention** is a means of providing context (local information) to an input.

Really informally, attention is a way to learn a weighted version of your input values. Of course, this is quite similar to what a convolution does.

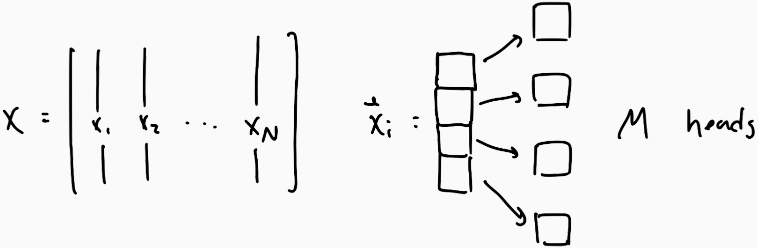
I'll give a very concise review of multihead self-attention (MSA) over the next few slides.

ATTENTION

Let our inputs be $x_i \in \mathbb{R}^D$, and let us have N inputs. We can place them in a matrix $X \in \mathbb{R}^{N \times D}$. MSA is:

$$MSA : X \to X_A \in \mathbb{R}^{N \times D} \tag{7}$$

where X_A is attentive? attended? at attention? :) First, we choose M heads so that M divides D; we split up the vector components:



ATTENTION (CONT.)

We will have three learnable weight matrices Q_m, K_m, V_m for attention head m which map $\mathbb{R}^D \to \mathbb{R}^{D/M}$. From Q_m, K_m we create a weight matrix W by:

$$W_m = \text{softmax}\left(XQ_m(XK_m)^T\right) \in \mathbb{R}^{N \times N} \tag{8}$$

Note that head m still depends on inputs from every part of X (and so attends to other heads). Finally, we use V_m to map back into $\mathbb{R}^{D/M}$. The output of attention head m is the function

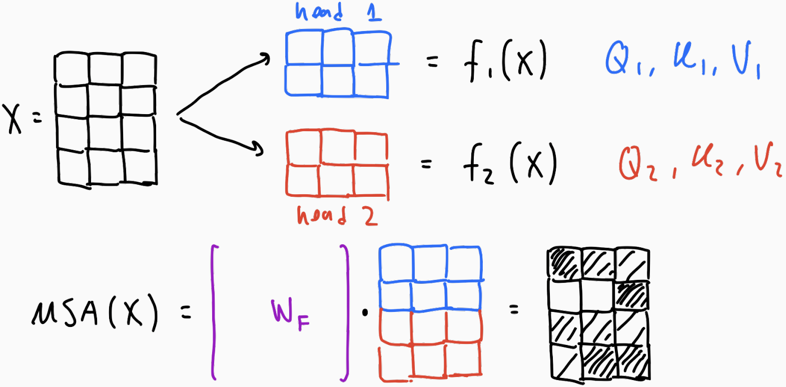
$$f_m(X) = W_m X V_m \in \mathbb{R}^{N \times D/M} \tag{9}$$

So far we have chosen a way to divide up the input vectors by component and learned a particular (nonlinear) mapping from the input data X to a set of M new matrices of shape $N \times D/M$ (so all $N \times D$ datapoints are accounted for).

ATTENTION (CONT.)

Given our data X we now have M (matrix) outputs $f_m(X)$. We combine these into an $N \times D$ matrix and multiply by a final $D \times D$ weight matrix W_F to get:

$$MSA(X) = [f_1(X)\ f_2(X)\ \cdots\ f_M(X)]W_F \in \mathbb{R}^{N \times D} \tag{10}$$



Self-attention for Lie groups

ATTENTION FOR GROUPS?

Self-attention allows us to learn a weighted version of the features by combining information about other features in the data.

In the context of Lie groups, attention will let us learn a **kernel** which we integrate against (lifted) features to obtain a weighted feature map.

We can have two types of attention for features $f(g)$ on a group G: **location attention** $A_L(g, g')$ and **content attention** $A_C(f(g), f(g'))$. There are many choices as to how to implement and combine these two.

CONTENT AND LOCATION ATTENTION

To be concrete, let's specify the content and location attention functions that are used in the **[Hutchinson et al, 2021]** default architecture. Let d be the feature dimension.

- Content attention: **dot product**

$$A_C(f(g), f(g')) = \frac{1}{\sqrt{d}}(Qf(g))^T Kf(g') \tag{11}$$

- Location attention: **MLP**

$$A_L(g, g') = \text{MLP}\left(\text{params}[\log(g^{-1}g)]\right) \tag{12}$$

where "params" extracts the parameters from the (Lie algebra) element $\log(g^{-1}g)$; in practice these are angles, distances, etc, and feeds it to an MLP of some depth (i.e. an MLP on the coordinates of the group element).

LIE SELF-ATTENTION

We begin with data (x_i, f_i) on a G-homogeneous space $\mathcal{X}$. First, we lift the features from $\mathcal{X} = G/H$ to G, giving us data $(g, f(g))$ on the group.
LieSelfAttention:
For g in the data,

- For g' in a neighborhood $N(g)$ of g,
  1. Compute content/location attention $A_C(f(g), f(g')), A_L(g, g')$
  2. Compute unnormalized weights:

$$\alpha_f(g, g') = A_C(f(g), f(g')) + A_L(g, g') \tag{13}$$

  (*we don't have to add them: can combine these arbitrarily*)
- Normalize the weights: $\{w_f(g, g')\}_{g' \in G} = \text{norm}\{(\alpha_f(g, g')\}_{g' \in G}$ (*default is constant; can use any choice of normalization*)
- Compute the output:

$$f_A(g) = \int_{N(g)} w_f(g, g') V f(g') dg' \tag{14}$$

ARCHITECTURE

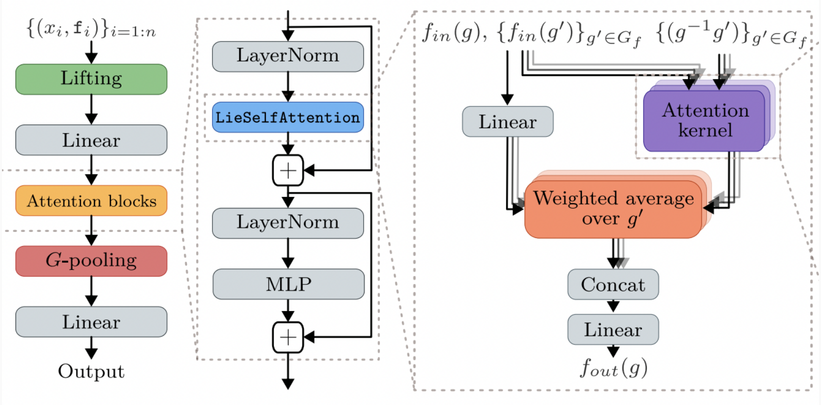


Figure 1: Architecture of the LieTransformer. We have described the lifting, Lie self-attention blocks. G-pooling is exactly analogous to spatial "maxpool" except in a neighborhood in the group.

ARCHITECTURE (CONT.)

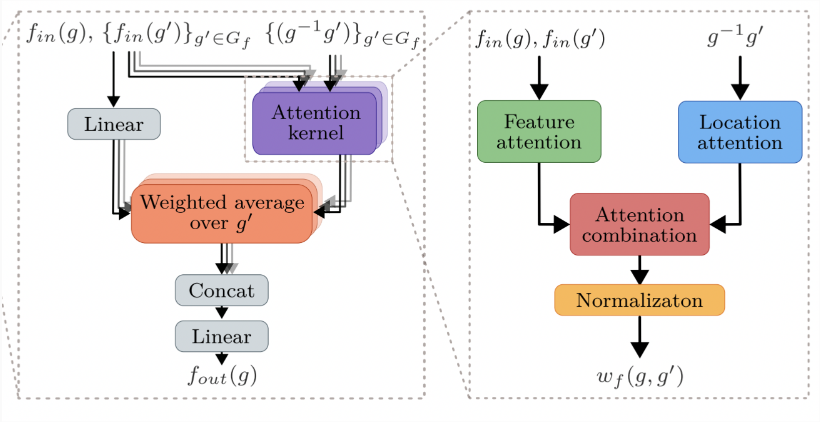


Figure 2: The attention block architecture.

COMMENTS

I have tried to describe the overall architecture. While many of these concepts seem abstract, they all have quite literal definitions. For any matrix group, the measures, maps, and representations have mundane definitions in terms of matrix elements.

Since Lie groups are smooth and infinite, many integrals over groups must be estimated by sampling from the Haar measure (Monte Carlo). Beyond this, samples are localized in neighborhoods.

The authors benchmark this architecture in a number of experiments. I'll focus on the most physics-y of these: particle dynamics.

LieTransformer: particle dynamics

PARTICLE PHYSICS

Everybody knows that particles are made of strings and connected by strings. Since string theory is too hard, we'll do **spring theory**.

The task is to predict the trajectories of N particles with random masses and connected by springs with random tensions. We are given the initial conditions (positions, momenta) of the particles as inputs.

You may know that physical law is invariant under many symmetries, and that this invariance leads to conserved quantities: energy, momentum, angular momentum, spin. This means that trajectories (dynamics) are **equivariant**: the initial conditions break the symmetry, but transform naturally under it.

LEARNING DYNAMICS

Hamiltonian dynamics are defined by first-order ODEs like:

$$\frac{dz}{dt} = F(z, t) = J\nabla_z H(z) \tag{15}$$

for some function $H(z)$, the Hamiltonian; here $z = (q, p)$. We can learn this two ways: learning the gradient $F_\theta(z, t)$ or learning the Hamiltonian $H_\theta(z)$.

People figured out that learning the Hamiltonian is much easier. It is conserved in time, meaning it is easier for models to extrapolate over more timesteps–by taking advantage of the time-translation symmetry.

Finally, we expect equivariant models to display almost exact conservation laws (as they both stem from symmetries). The authors measure this fact.

EXPERIMENT DETAILS

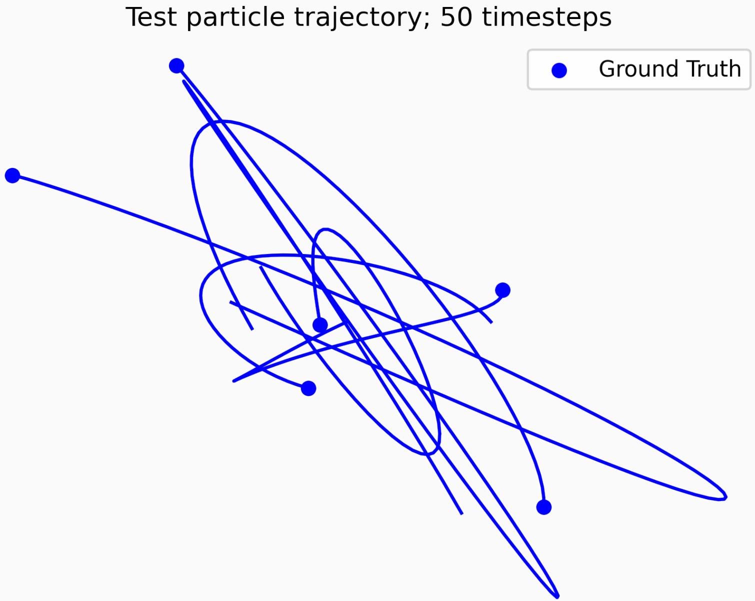
The task is to predict 6 particles over 50 timesteps with randomly selected masses and couplings.

We use 4 layers with hidden layer width $k = 384$ in the following.

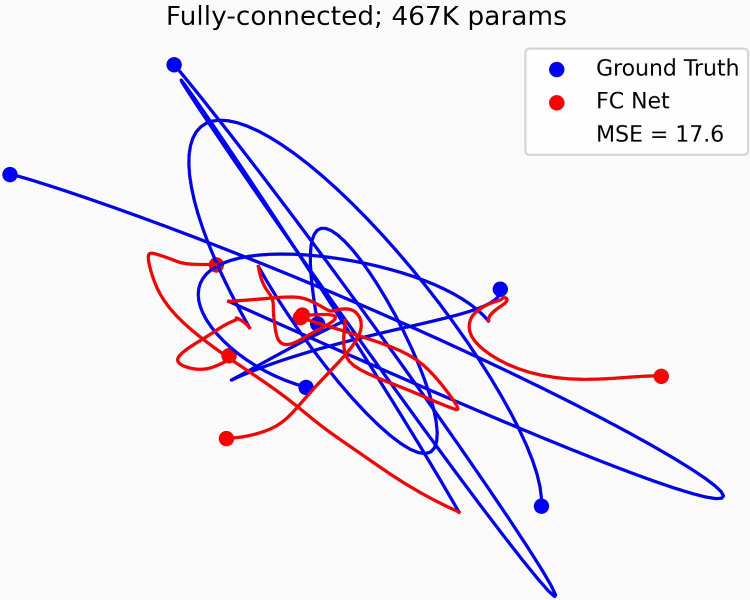
The models are tested on newly generated data after training on 3000 5-step trajectories with MSE loss.

We'll compare a number of architectures' performance on the task of trajectory prediction: fully-connected (FC), Hamiltonian fully-connected (HFC), LieConv CNN (both T_2 and $SO(2)$-equivariant), and finally LieTransformer.

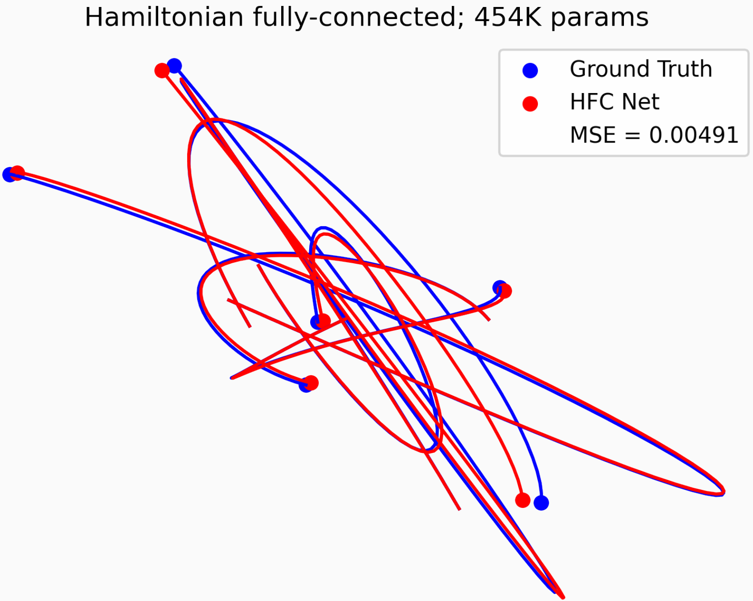
SIMULATED TRAJECTORY



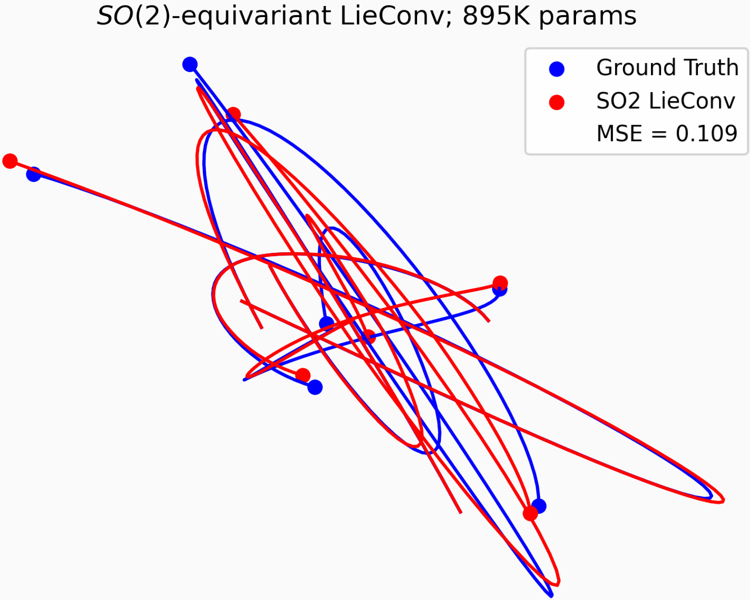
FULLY-CONNECTED



HAMILTONIAN FULLY-CONNECTED

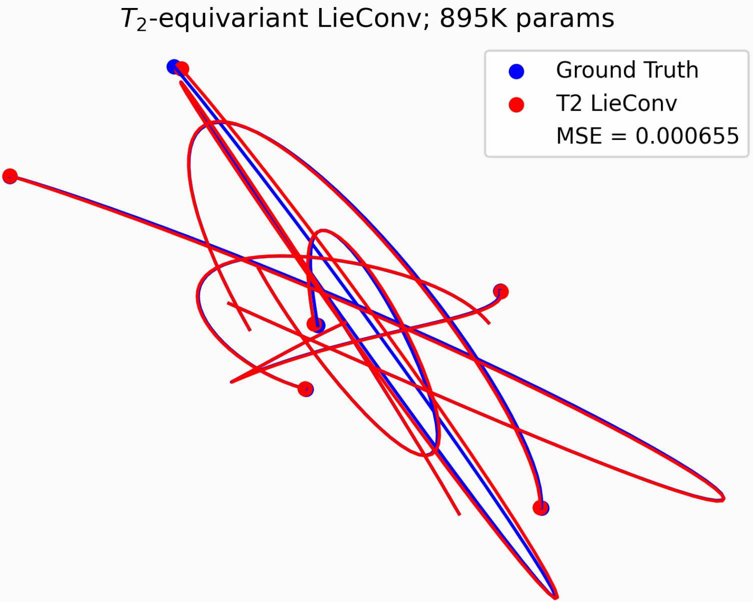


LIECONV (1)

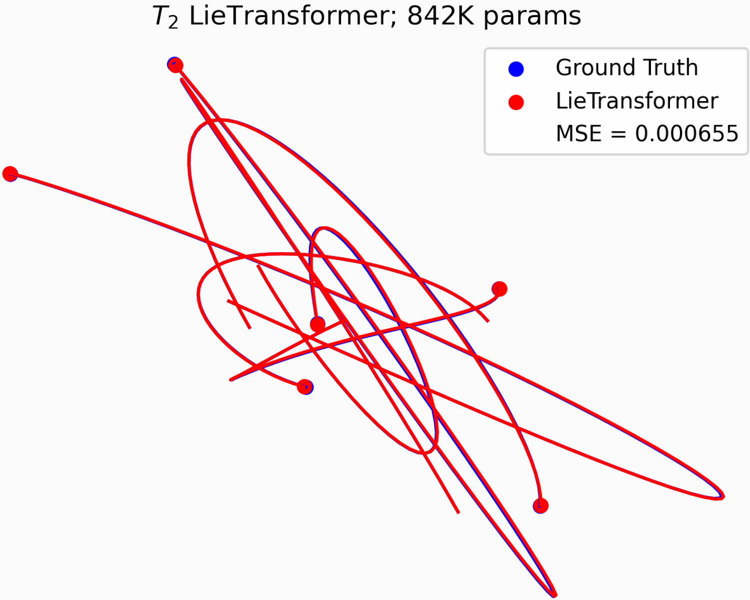


$SO(2)$-equivariant LieConv; 895K params
● Ground Truth
● SO2 LieConv
MSE = 0.109

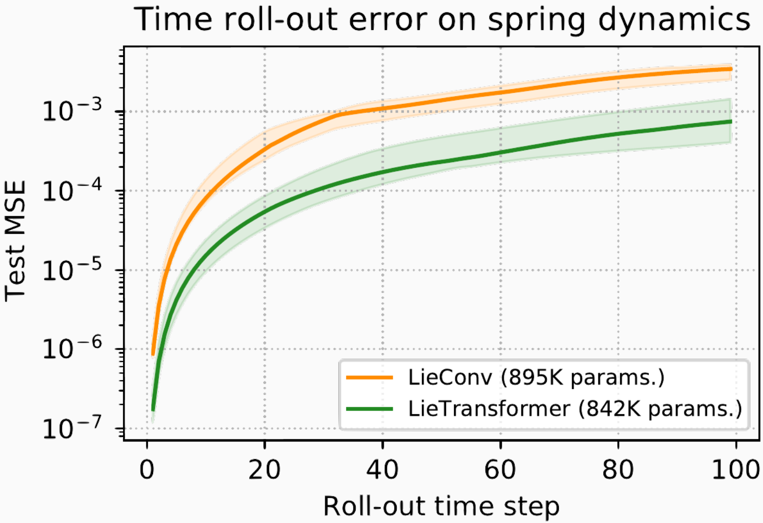
LIECONV (2)


T_2-equivariant LieConv; 895K params
● Ground Truth
● T2 LieConv
MSE = 0.000655

LIETRANSFORMER



LIECONV VS LIETRANSFORMER



DATA EFFICIENCY

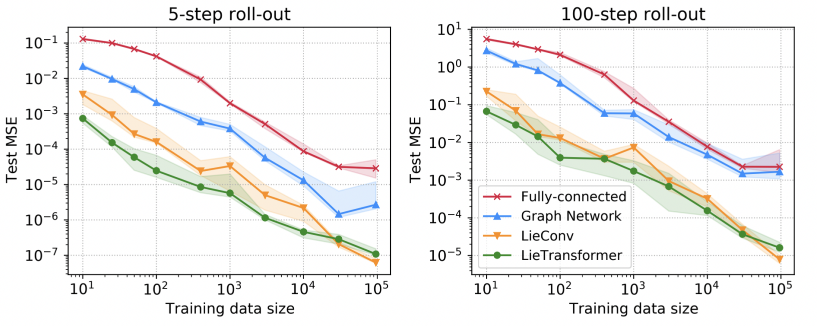


Figure 3: Comparison of data efficiency on the particle dynamics task for multiple architectures **[Hutchinson et al, 2021]**.

CONCLUSION

LieTransformer outperforms LieConv by an order of magnitude on the particle dynamics task while using fewer parameters.

LieTransformer integrates symmetries and contextual information by the attention mechanism.

LieTransformer architectures can be applied to vision tasks, physical simulation, and more.

Thanks for your **attention**!