ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-TUPAB215

NOVEL NON-LINEAR PARTICLE TRACKING APPROACH EMPLOYING LIE ALGEBRAIC THEORY IN THE TensorFlow ENVIRONMENT

J. Frank, M. Arlandoo, P. Goslawski, J. Li, T. Mertens, M. Ries, L. Vera Ramirez
Helmholtz-Zentrum Berlin fuer Materialien und Energie GmbH (HZB), Berlin, Germany

Abstract

With this paper we present first results for encoding Lie transformations as computational graphs in Tensorflow that are used as layers in a neural network. By implementing a recursive differentiation scheme and employing Lie algebraic arguments we were able to reproduce the diagrams for well known lattice configurations. We track through simple optical lattices that are encountered as the main constituents of accelerators and demonstrate the flexibility and modularity our approach offers. The neural network can represent the optical lattice with predefined coefficients allowing for particle tracking for beam dynamics or can learn from experimental data to fine-tune beam optics.

INTRODUCTION AND MOTIVATION

Machine learning and deep learning approaches have become essential tools for analyzing big and complex structures. Recently, an algorithm was proposed to enable particle tracking and beam optics tuning in accelerator machines [1, 2]. The results seem promising, but we find it cumbersome to pre-calculate the weight matrices that are then implemented in a neural network (NN) for forward propagation. We present a novel approach where we encode Lie transformations as computational graphs and thus obtain a natural structure for a NN. Each Lie transformation propagates a charged particle through an accelerator element and is represent by a NN layer - what we denote as **Lie layers**.

Our approach presents the benefit that we only need to create the computational graph comprised of Lie layers and the succeeding evaluation takes place in the optimized TensorFlow (TF) environment [3] with its Keras interface [4]. In addition, we are able to input generic Lie maps for machine elements and train the network by feeding it real data. This allows us to extract dominant non-linear contributions from an existing accelerator.

LIE MAPS

For simplicity we assume Hamiltonians with no explicit dependence on the independent variable. For phase space coordinates $r \in \mathbb{R}^6$ Hamilton's equations of motion are the first order coupled differential equations that give the evolution of a physical system by

$$\dot{r} = -J\frac{\partial H}{\partial r}, \tag{1}$$

where

$$J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix} \tag{2}$$

is the symplectic matrix comprised of the 4 $n \times n$ blocks, I being the identity matrix. If we interpret the Poisson bracket as the symplectic 2-form in phase space, we can define the Lie operator as

$$\{H, \cdot\} =: H := \sum_i \frac{\partial H}{\partial x_i}\frac{\partial}{\partial p_i} - \frac{\partial H}{\partial p_i}\frac{\partial}{\partial x_i} \tag{3}$$

and formally represent the solution to the equations of motion Eq. (1) as Lie maps

$$r(s) = e^{-s:H:}r(0) = \sum_{k=0}^{\infty}\frac{(-s:H:)^k}{k!}r(0), \tag{4}$$

where $:H:^k r_0 = \{H, \{..., \{H, r_0\}\}\}$ are k-nested Poisson brackets. We refer the reader to [5, 6] for details.

LIE LAYER ARCHITECTURE

Let us denote each Lie map by $\mathcal{M}_i = e^{-s:H_i:}$, then the optical lattice can be represented by the 1-turn map

$$\mathcal{M} = \mathcal{M}_1 \circ \mathcal{M}_2 \circ ... \circ \mathcal{M}_n. \tag{5}$$

For an element of length L we create a computational graph able to evaluate the corresponding Lie operators Eq. (3) as part of a recursive differentiation scheme and hence, create a *Lie Transformation Layer* through the definition of Eq. (4). In particular we have conceived two implementations:

- `LieTransLayer` : a general implementation based on automatic differentiation (`tf.GradientTape()`) conceived for any function that can be represented with TF operations.

- `LieTransPolyLayer` : a specific implementation for polynomials optimizing the evaluation with exact differentiation. A polynomial with k terms in m variables is hence represented as a $k \times (m+1)$-matrix, allowing us to efficiently rewrite all the transformations needed for the Lie transformation (polynomial multiplication, partial derivation, simplification and evaluation) as tensor operations.

For instance, the paraxial approximation of a quadrupole will be expressed as a `LieTransLayer` as follows:

```
def H(x, px, y, py, z, pz,
        extra_vars, coefficients):
    k, beta = extra_vars
    return (px**2 + py**2 + k*(x**2 - y**2)
    + (pz/beta)**2)/2
layer = LieTransLayer(H, L = .27, N = 10,
    extra_vars = [.8, 1.])
```

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-TUPAB215

where $N = 10$ is the cutoff of the Lie map, `extra_vars` are used for element properties (e.g. magnet strengths) and `coefficients` can be a trainable variables. Since our layer implementations instantiate `tf.keras.layers.Layer`, they can be formally concatenated, evaluated in eager mode or pre-built as a computational graph, allowing variable updates and coefficient tuning with optimization methods. In every case we take advantage of TF performance, allowing us to evaluate Lie maps with high truncation orders in milliseconds.

We note that the cutoff produces only a symplectic N-jet, which leads to the violation of the symplectic condition in general. However, by choosing a large enough `N` and slicing each element

$$e^{-L:H_i:} = \left(e^{-\frac{L}{n}:H_i:}\right)^n = \underbrace{e^{-\frac{L}{n}:H_i:}\ldots e^{-\frac{L}{n}:H_i:}}_{\text{n-times}} \tag{6}$$

or by the "leap-frog" decomposition

$$e^{-L:H_i:} = e^{-\frac{L}{4}:H_i:}e^{-\frac{L}{2}:H_i:}e^{-\frac{L}{4}:H_i:} \tag{7}$$

we were able to reduce the error bellow machine precision.

Additionally, we are also able to have generic polynomials with tunable coefficients

$$\mathcal{M}_i = e^{-L:H_i:} \approx e^{:f:} \tag{8}$$

or in terms of Lie layer

```
x = LieTransLayer(f,N,L,coefficients=c)(x)
```

where `c` will represent the coefficients of the polynomial f. The TF framework will allow us to use real data from the accelerator to tune the coefficients and so pinpoint dominant non-linear effects in the optical lattice by order of non-linearity.

TRACKING THROUGH FODO

We will demonstrate the proposed approach on the optical lattice of a simple FODO cell

$$\mathcal{M}_{\texttt{FODO}} = \mathcal{M}_{\texttt{QF}} \circ \mathcal{M}_{\texttt{D}} \circ \mathcal{M}_{\texttt{QD}} \circ \mathcal{M}_{\texttt{D}} \circ \mathcal{M}_{\texttt{QF}}, \tag{9}$$

where `QF` denotes focusing quadrupole, `QD` defocusing quadrupole and `D` drift spaces. In particular, the FODO lattice is set with the following parameters:

- Focusing Quadrupole: $L = 0.27$, $k = 0.8$
- Defocusing Quadrupole: $L = 0.5$, $k = -0.57$
- Drift: $L = 2.48$
- $\beta = 1$

Figure 1 depicts phase space plots, where we tracked with our model and compare it to tracking with the exact solutions of the Lie maps. We use the full drift Hamiltonian and the paraxial approximation to represent quadrupoles as linear transfer maps.

In addition, TF allows us to calculate the Jacobian matrix M of our model as a computational graph so we can assess

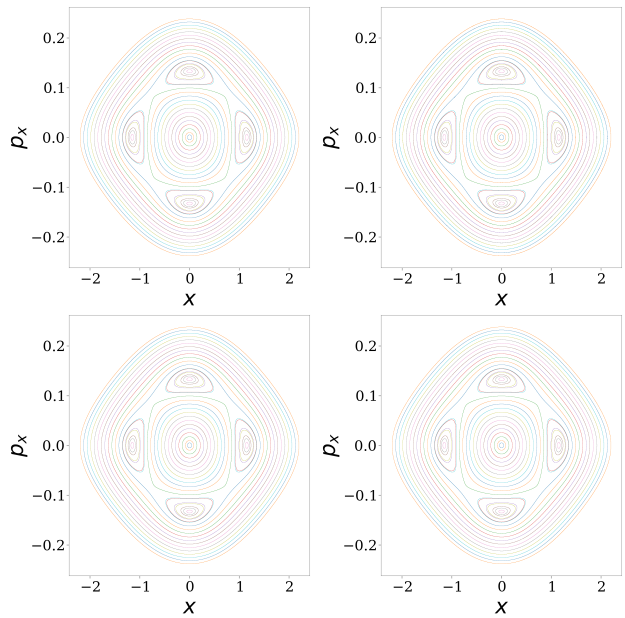

Figure 1: Phase space portraits for tracking 32 particles during 1 million turns through the FODO cell Eq. (9). **Upper left:** exact tracking with transfer maps. **Upper right:** concatenation of `LieTransLayers` with element slicing Eq. (6) ($n = 10$) and truncation order 4. **Lower left:** concatenation of `LieTransLayers` with leap-frog decomposition Eq. (7) and truncation order 8. **Lower right:** concatenation of `LieTransPolyLayers` with truncation order 12.

the symplectic condition through the *symplectic error* at a point x

$$\|M(x)^T J M(x) - J\|$$

where J is the symplectic matrix Eq. (2). The corresponding evaluation for several model configurations with a cloud of 1000 points sampled from $\mathcal{N}(0, 0.1)$ is shown in Table 1.

LEARNING FROM GENERATED DATA FOR FODO

As a further feature of our implementation, the functional concatenation of our Lie layers can be easily encapsulated and trained as a neural network. Notice that the resulting network will be an almost *deterministic* function, meaning that the only trainable parameters will be those that we set as free variables in the corresponding Hamiltonian definition.

As a proof of concept, we want the network structure to learn the coefficients of the second degree terms in the paraxial approximation of the quadrupole Hamiltonian

$$w_1 x^2 + w_2 p_x^2 + w_3 y^2 + w_4 p_y^2 + w_5 z^2 + w_6 p_z^2.$$

For this, we replace the two original instances of `H` function in the corresponding `LieTransLayer` representing `QF` and `QD` Hamiltonians with a parametrized version and compile the concatenation as a neural network, whose trainable weights will be the corresponding Hamiltonian coefficients for `QF` and `QD` . Figure 2 shows the learning process with a simple RMSE loss function calculating the deviation w.r.t. the training data. This training data is generated in real time

ISBN: 978-3-95450-214-1 ISSN: 2673-5490 doi:10.18429/JACoW-IPAC2021-TUPAB215

Table 1: Symplectic Error – FODO with Paraxial Approximation of the Quadrupoles

Order	Implementation	Slice Trick	Norm Avg.	Norm Std.	Norm Max.	Jacobian Graph Build
1	LieTransLayer	Leap frog	0.0341348812	0.062019799	17.733.960.929	0.306 secs
1	LieTransLayer		0.0908883922	0.1528337494	4.400.475.823	0.146 secs
1	LieTransLayer	n = 10	0.009324259	0.0181332906	0.5229723359	0.722 secs
1	LieTransPolyLayer		0.0908883922	0.1528337494	4.400.475.823	6.353 secs
4	LieTransLayer	Leap frog	1.17E-006	8.74E-008	3.14E-006	1.887 secs
4	LieTransLayer		7.23E-005	5.32E-006	0.0001921531	0.659 secs
4	LieTransLayer	n = 10	7.24E-010	5.45E-011	1.96E-009	6.112 secs
4	LieTransPolyLayer		7.23E-005	5.32E-006	0.0001921531	6.299 secs
8	LieTransLayer	Leap frog	5.82E-013	3.32E-014	9.73E-013	128.107 secs
8	LieTransLayer		5.96E-010	3.42E-011	9.89E-010	42.064 secs
8	LieTransLayer	n = 10	4.28E-015	1.51E-014	4.75E-013	423.84 secs
8	LieTransPolyLayer		5.96E-010	3.42E-011	9.89E-010	6.218 secs
12	LieTransPolyLayer		2.65E-015	1.05E-014	3.21E-013	6.329 secs
16	LieTransPolyLayer		1.98E-015	1.56E-014	4.93E-013	6.171 secs
20	LieTransPolyLayer		1.98E-015	1.56E-014	4.93E-013	6.367 secs

through the exact FODO tracking for particle clouds sampled from $\mathcal{U}(-0.04, 0.04)$.

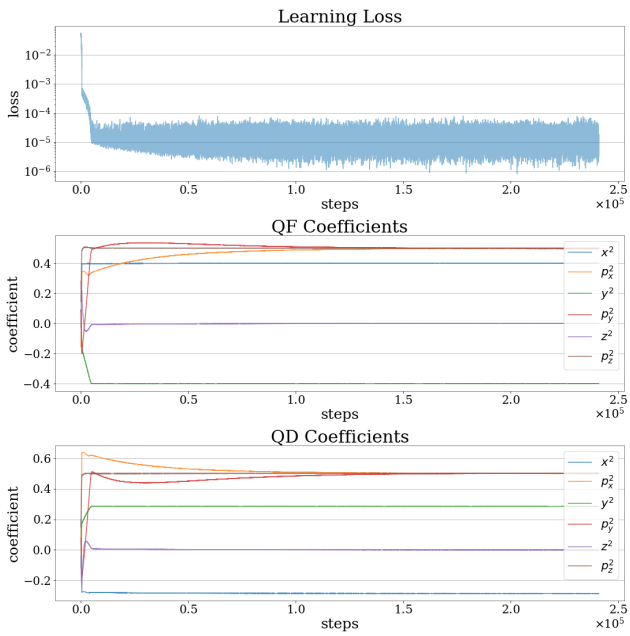

Figure 2: Learning process for the QF and QD Hamiltonian coefficients in the FODO lattice. LieTransLayer with truncation order 12, RMSE as loss function, Adam as optimizer, ca. 12 ms per learning step.

Notice that the loss function compares the tracked particles only *at the end* of the FODO lattice. It means that our model is able to properly capture the intermediate quadrupole behaviour and converge to the theoretical QF and QD Hamiltonian coefficients w.r.t. our FODO parameters

$$\texttt{QF}: 0.4x^2 - 0.4y^2 + 0.5p_x^2 + 0.5p_y^2 + 0.5p_z^2$$
$$\texttt{QD}: -0.285x^2 + 0.285y^2 + 0.5p_x^2 + 0.5p_y^2 + 0.5p_z^2$$

just observing the final state of the particle cloud after the five FODO elements.

CONCLUSION AND OUTLOOK

In this article we have presented a novel approach of encoding Lie transformations as computational graphs in TensorFlow. On the example of the FODO cell we showed the flexibility and modularity of our approach. The results seem promising albeit we slightly violate the symplectic condition. The authors are currently investigating possible approaches for endowing the Lie layers with an inherent symplectic structure. Lastly, we also demonstrate the benefit of extracting Hamiltonian coefficients from data. We hope this will allow us to determine dominant non-linear terms for actual storage rings and thus provide an essential tool in non-linear beam dynamics.

REFERENCES

[1] S. N. Andrianov *et al.*, "A role of symbolic computations in beam physics", in *Int. Workshop on Computer Algebra in Scientific Computing (CASC 2010)*, Tsakhkadzor, Armenia, 2010, pp. 19-30. doi:10.1007/978-3-642-15274-0_3

[2] A. Ivanov *et al.*, "Physics-based deep neural networks for beam dynamics in charged particle accelerator", *Phys. Rev. Accel. Beams*, vol. 23, no. 7, p. 074601, Jul. 2020. doi:10.1103/PhysRevAccelBeams.23.074601

[3] TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, https://www.tensorflow.org/

[4] Keras, https://keras.io

[5] A.J. Dragt *et al.*, *Lie methods for nonlinear dynamics with applications to accelerator physics*, 2011.

[6] A.J. Dragt, "Lectures on nonlinear orbit dynamics", *AIP Conference Proceedings*, vol. 87, pp. 147-313, 1982. doi:10.1063/1.33615