Symbolic Processing for Trajectory Prediction

Alec J. Hoyland
Center for Systems Neuroscience
Boston University
Boston, MA 02215
ahoyland@bu.edu

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Proposal

The purpose of this project is to develop algorithms for trajectory prediction based on first-principles.

We explicate the core theory of trajectory prediction, and propose both biologically plausible and artificial network architectures for predicting a trajectory without any training data, or in-built physics models.

We build and evaluate a factorized neural relational inference machine, as well as a dense, two-layer neural network for ODE and SDE prediction. We also advance the theory on associative-memory machines for trajectory prediction. If I have time, I intend to also implement this theory as a computational model.

The goal of this project is to explore how these models handle prediction without pre-training, as well as to evaluate how well they can be used to elucidate basic principle of symbolic processing.

2 Introduction

Symbolic processing is an important feature of general intelligence, crucial for understanding underlying rules and latent features, to develop models of the world that can be used for prediction. In animals, simple prediction tasks such as object tracking and navigation, are necessary for survival. These tasks fall under the umbrella of symbolic processing. One of the most straightfoward symbolic processing tasks which can be abstracted to a computational model is trajectory prediction.

This task involves observing a system with a state that evolves in time for a few epochs, before making predictions of the system's future state based on those observations. Real-world applications involve pedestrian and vehicle tracking, robot navigation and agility, automated physics tracking, and aeronautics. In essence, it is a type of supervised learning problem, in which the prediction model is being updated at each time step.

The goal of this paper is to put forward several theories on how this can be practically done, with an emphasis on discovering fundamental mathematical relationships which could be performed by ensembles of neurons in the brain.

Preprint. Work in progress.

2.1 What is a trajectory?

For the purpose of this section, a *trajectory* is an ordered set of states which are evolved according to one or more rules. The purpose of symbolic trajectory prediction is, if given a few states of the trajectory, to intuit the latent rules and predict future states. In a real-world scenario, the agent predicts simultaneously to the evolution of the trajectory in time, meaning that more data are available to the agent each time step.

Mathematically, a trajectory is formally represented as a sequence of values $\left[f^k(x)\right]_{k\in\mathbb{N}}$, calculated by the iterated application of a mapping f to an element x of its source. It is analogous to the time-ordered set of states in a dynamical system (*e.g.* a Poincaré map) such as the function of position produced by integrating the equations of motion in Newtonian mechanics.

2.2 What is a feature?

A feature is any important defining characteristic of a snapshot of a trajectory. For example, if the trajectory represents the motion of a ball in flight across a 2-dimensional coordinate grid, the crucial features are the coordinates (x, y) parametrixed by the time-coordinate t. If the color of the ball mattered (*i.e.* isn't time-invariant), then it would also be a feature. For the purpose of this paper, the agent is assumed to be able to discern which features are important and which are not. From a biological perspective, we will take for granted the miraculous specificity of the visual system in mammals. In artificial intelligence, we will assume a sufficiently-advanced computer vision algorithm and peripherals. If, for example, three polygonal shapes were drawn on a blackboard and the agent were asked to determine the pattern, it is reasonable to neglect the minute deviations in the lines caused by the hand of an unskilled artist. Similarly, imperfections in printed symbols could be ignored as well. The best heuristic for determining what is a feature and what is not relies on comparing adjacent states of the trajectory. Major differences are features; minor differences can be neglected.

2.3 Representing a trajectory through vector transformations

One way to imagine a trajectory is to consider a series of points in the plane formed by the time-coordinate t and a function $f(t)$ which produces a vector output $\vec{x}$, where each element in $\vec{x}$ is the numerical value of a feature at time t. In this situation, the trajectory is a vector parametrized by the time-coordinate. To compare adjacent states, the first time-derivative can be taken. Since time is discrete, the derivative operator is best represented by the finite difference.

A finite difference is a mathematical expression of the form $f(x + b) - f(x - a)$. If the finite difference is divided by $b - a$, the different quotient is defined. The first forward finite difference is defined by

$$\mathrm{D}x_n = x_n - x_{n-1} \tag{1}$$

for variable x at discrete indices n and $n - 1$. Higher-order finite differences can also be defined, such as the second-order forward finite difference:

$$\mathrm{D}^2x_n = x_n - 2x_{n-1} + x_{n-2} \tag{2}$$

These finite differences are first-order correct in accuracy with uniform grid-spacing t_n. More accurate finite differences can be computed using more points. At minimum, approximating a numerical derivative of order k with the fewest points requires $k + 1$ points.

If two states of the trajectory x_n and x_{n-1} are known, the next point can be approximated as:

$$x_{n+1} \approx \frac{1}{\Delta}\mathrm{D}x_n + x_n = \frac{x_n - x_{n-1}}{\Delta} + x_n \tag{3}$$

where $\Delta = t_n - t_{n-1}$ is the difference in the time-coordinate between adjacent indices. If higher-order finite differences are used, the estimate will be more accurate, up to the derivative order equal to the order of the polynomial representation of the function. This requires knowing more past states.

Since any well-behaved function can be approximated by its (truncated) Taylor series, the discrete Taylor series generalization, given by Einar Hille, can approximate a well-behaved function using finite differences [1]. For $t > 0$

$$f(a+t) = \lim_{\Delta \to 0^+} \sum_{n=0}^{\infty} \frac{t^n}{n! \Delta^n} \mathrm{D}^n f(a) \tag{4}$$

This equation describes how if a function is known to be evaluatable at a: $f(a)$, and finite differences can be taken, then the function can be approximated at any future point $a + t$.

The vector transformation formulation is satisfactory for predicting trajectories based on linear rules. It is mathematically equivalent to least-squares fitting, and therefore can be represented in matrix form. This results in a linear transformation called the predictor:

$$\mathrm{P}(a) = \left[\frac{1}{n!} \mathrm{D}^n f(a) \right]_{n \in \mathbb{N}} \tag{5}$$

The dot product of the predictor can be taken with $\left[\frac{t^n}{\Delta^n} \right]_{n \in \mathbb{N}}$ to find the predicted trajectory $f(a + t)$ for $t > 0$.

2.3.1 Example: one-dimensional trajectory

Here we consider a trajectory produced by a linear combination in a polynomial basis. The goal is to predict the one-dimensional trajectory

$$f(t) = -40 + 100t - 10t^2 \tag{6}$$

For $\Delta = 0.01$, the first and second finite differences are computed at $f(2\Delta)$. By the Hille series (4), we can compute

$$f(2\Delta + t) = f(0) + \frac{t}{\Delta} \mathrm{D} f(2\Delta) + \frac{t^2}{2\Delta^2} \mathrm{D} f(2\Delta) \tag{7}$$

which is equivalent to the Taylor series

$$f(t) = x_0 + v_0 + \frac{1}{2} a_0 t^2 \tag{8}$$

where x_0, v_0, and a_0 are parameters fit by the model. Any extra parameters, for example, for a 3rd order polynomial fit are zero, and thus omitted.

ADD CODE & FIGURE

2.3.2 Example: multi-dimensional trajectory

Here we consider a multi-dimensional example. The latent rule is at time $t_n \in \mathbb{N}$, we draw a square, where the top-left vertex is at the origin, with a side length of $s = t_n + 1$.

Treating the observation mechanism (*e.g.* the visual cortex) as an oracle, we identify eight features: the four coordinate-pairs which note the vertices of the square. One vertex (top left) never changes position – it is fixed at the origin. The top right vertex proceeds according to the rule $f_{tr}(t_n) = (0, t_n + 1)$. The bottom left vertex proceeds according to $f_{bl}(t_n) = (-1 - t, 0)$. Correspondingly, the bottom right vertex evolves according to $f_{br}(t_n) = (-1 - t, -1 - t)$.

Thus, each coordinate pair evolves either linearly or doesn't change (is constant). Finite difference approximations can be taken for each coordinate independently, and the rule can be determined after two steps.

ADD CODE & FIGURE

2.4 Representing a trajectory through function transformations

The idea of mapping vectors across a time-coordinate can be expanded to a larger space of problems. In this generalization, the latent rule involves defining a series of functions ordered in time: $[f_n(\mathcal{S})]_{n\in\mathbb{N}}$.

Each function acts on a surface $\mathcal{S}$, the specifics of which should not matter for the general formalism (*p.c.* Marc Howard).

In the field of functional calculus, arithmetic can be defined on functions. The "sum" of two functions is the convolution $*$, and the "difference" of two functions is the cross-correlation $\star$.

If the rule can be modeled as a first-order transformation in function space, the evolution rule is:

$$f_{n+1}(t) = (f_n(t) \star f_{n-1}(t)) * f_n(t) \tag{9}$$

This formalism is an extension of the vector representation formalism (2.3), since convolution and cross-correlation are defined for all well-behaved functions.

2.4.1 Example: rate coding in biological neurons

Consider a biological neuron, where the firing rate is represented by $r(t)$, which receives an input $f(t)$ which is related to the firing rate by the following differential equation:

$$\frac{\mathrm{d}r}{\mathrm{d}t} = -kr + f(t) \tag{10}$$

where k is a parameter which describes how quickly the firing rate decays. Solving for $r(t)$ yields:

$$r(t) = F(t) \equiv \int_0^\infty f(t)e^{-kt}\mathrm{d}t \tag{11}$$

which is the Laplace transform of $f(t)$ [2, 3]. This formulation, in which biological neurons record temporal information in their rate of firing rate decay, has broad implications for trajectory prediction, since convolution and cross-correlation in the time domain are multiplication in the Laplace domain:

$$F_{n+1}(t) = F_n(t) \cdot \bar{F}^*_{n-1}(t) \cdot F_n(t) \tag{12}$$

where f^* is the complex conjugate of f. This provides a convenient mathematical framework for exploring spatiotemporal navigation in a systems neuroscience context, however in practice, the theory is difficult to implement in quantitative models [4].

One could conceive of a network where each neuron receives an input, and decays down to some

2.5 Universal approximation with neural networks

The problem of predicting a trajectory is equivalent to that of fitting a function to the trajectory. The function is then a predictor of the future state of the process underlying the trajectory. This allows artificial neural networks (ANNs) to perform regression analysis on a trajectory, to fit a model to the trajectory data. Since the data are available only in real-time, prediction and training occur in an interleaved manner. Real-time machine learning is an active field of study, with the NSF offering up 10,000,000 USD in grant funding in 2019 to this field.

Neural networks are nonlinear prediction engines, which exploit the Stone-Weierstrass theorem [5] result that any continuous function on a closed interval can be uniformly approximated by a polynomial function, and the universal approximation theorem [6, 7], which proves that arbitrarily wide feedforward neural networks are universal approximators for Lebesgue measure zero functions. **(author?)** [8] shows that arbitrarily deep feedforward rectifying linear unit (ReLU) networks can approximate with arbitrary precision any Lebesgue measure zero function.

This is an extremely powerful result, which shows that artificial neural networks of sufficient size can approximate a large class of functions.

3 Related Works

4 Methods

5 Results

6 Discussion

7 Conclusion

References

[1] Einar Hille and Ralph S. Phillips. *Functional Analysis and Semi-Groups*. American Mathematical Society Colloquium Publications, Vol. 31. American Mathematical Society, Providence, R. I., 1957.

[2] Karthik H. Shankar and Marc W. Howard. A scale-invariant internal representation of time. *Neural Computation*, 24(1):134–193, January 2012.

[3] Zoran Tiganj, Karthik H Shankar, and Marc W Howard. Encoding the Laplace transform of stimulus history using mechanisms for persistent firing. *BMC Neuroscience*, 14(Suppl 1):P356, July 2013.

[4] Albert Tsao, Jørgen Sugar, Li Lu, Cheng Wang, James J. Knierim, May-Britt Moser, and Edvard I. Moser. Integrating time from experience in the lateral entorhinal cortex. *Nature*, 561(7721):57–62, September 2018.

[5] Louis de Branges. The Stone-Weierstrass theorem. *Proceedings of the American Mathematical Society*, 10(5):822–824, 1959.

[6] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, December 1989.

[7] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The Expressive Power of Neural Networks: A View from the Width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc., 2017.

[8] Boris Hanin and Mark Sellke. Approximating Continuous Functions by ReLU Nets of Minimal Width. *arXiv:1710.11278 [cs, math, stat]*, March 2018.