Learned Legendre Predictor: Learning with Compressed Representations for Efficient Online Multistep Prediction

P. Michael Furlong[1], **Andreas Stöckel**[1], **Terrence C. Stewart**[2], **Chris Eliasmith**[1]

[1]Centre for Theoretical Neuroscience, University of Waterloo, Waterloo, ON, Canada
[2]National Research Council, University of Waterloo Collaboration Centre, Waterloo, ON, Canada

E-mail: michael.furlong@uwaterloo.ca

CTN-TR-20220814-001 Version 1 - 14 August 2022

Abstract. We describe the Learned Legendre Predictor (LLP), a new algorithm for predicting function values at all points within a prediction horizon. The algorithm distinguishes itself from prior work in this area by making continuous time predictions, and by learning online. While we implement the algorithm using the Legendre polynomials as a basis, we derive the algorithm for the class of generalized Fourier coefficiens. For the purposes of demonstrating basic functionality we test the algorithm on predicting observations from the Mackey-Glass function and the Lissajous function. In this pure prediction setting we demonstrate performance comparable to off-line trained baseline algorithms trained on a comparable volume of data.

1. Introduction

Multistep prediction of functions is the prediction of function values, $f(\cdot, t)$ over a set of future time points, $\subseteq [t, t + \theta_p]$. Multistep prediction has applications in forecasting including weather, control, and supply and demand modelling. Accurate predictions allow users to take appropriate, domain-specific actions.

In this paper, we develop the Learned Legendre Predictor (LLP) algorithm, a neural network learning algorithm that learns to make multistep predictions in an efficient online manner. The general approach is to store the history that is needed by the learning algorithm in a compressed format, and then directly use the compressed representation to perform the weight updates. We provide a derivation of this algorithm using generalized Fourier coefficients, but we instantiate the algorithm using Legendre polynomials as a compressed representation; rather than representing some sampling of the value over $[t, t + \theta_p]$, we instead encode predicted values as the coefficients of Legendre polynomials. Importantly, we use this same representation to encode not just the output prediction, but the history the prediction is based on, the history of the

activity of the neural network, and even the history of the output predictions themselves. These last two histories allow the network to compare its previous predictions to the current observed state, and determine what weight changes would have improved the prediction. Crucially, the use of Legendre polynomials allows us to define a learning rule where the learning happens at all points in time (in $[t, t + \theta_p]$) in parallel, using a single learning rule, working entirely in the compressed state space.

We demonstrate that this algorithm can learn online by training it to predict a number of delayed signals. For the purpose of demonstrating performance, we test the algorithm on the Mackey-Glass function and the Lissajous function. More rigorous testing of the algorithm's performance is left to future work.

In order to learn multistep predictive models online, one must keep a history of observations and predictions. In order to do so effectively we make use of the Legendre Delay Network (LDN) [Voelker et al., 2019], which can optimally delay observations from a linear system, and can be used an efficient memory of observations. An LDN is a linear system $\dot{\mathbf{m}}(t) = A\mathbf{m}(t) + B\mathbf{c}(t)$, where A and B are defined such that m is a q-dimensional vector that represents a sliding window of the past history of c using the Legendre polynomial basis space.

In the LLP, we use an LDN to collect the past history of any state information that our prediction should be based on. In the case of predicting the future state of a plant, this would include both the observed state of the plant and the control signal sent to it. This information, compressed into Legendre polynomial coefficients by the LDN, is then fed into a single-hidden-layer neural network. We use a Rectified Linear Unit for the hidden layer response function, but any non-linearity could be used. The input weights W and biases B are fixed, and the output weights D are initialized to zero and updated using our online learning rule. The network's outputs are the Legendre polynomial coefficients that are the network's current prediction of the future state of the plant.

To generate the input weights W and the biases B, we initialize them randomly so as to produce hidden layer activity where each neuron's maximum value is in the same range and the distribution of sparsity (*i.e.*, the proportion of the input space for which the neuron outputs a non-zero value) is uniform. This initialization has been used extensively in the Neural Engineering Framework (NEF) [Eliasmith and Anderson, 2003, Voelker et al., 2017] as a useful fixed representation across a wide variety of non-linear functions.

2. Prior Work

Multi-step, or multi-horizon, prediction breaks down broadly into two approaches: *iterative* (or *autoregressive*) and *direct* prediction methods [Chevillon, 2007]. The former iteratively applies one-step predictions to construct future trajectories, while the latter predict whole time series at once. In principle, iterative methods should be as good or better than direct methods, however, if the iterative step is inaccurately modelled,

then that error will be integrated, making a direct approach more favourable [Chevillon, 2007, Taieb and Atiya, 2015].

Autoregressive models are the prototypical multi-step prediction method [Winters, 1960, Box et al., 2015], however they have difficulty predicting non-linear signals. Non-linear autoregressive models exist, *e.g.* [Leontaritis and Billings, 1985a,b, Chen and Billings, 1989], and the adoption of deep learning methods has allowed exploration of a wide variety of techniques. Recent neural network–based iterative methods use recurrent neural networks, typically LSTMs, to summarize the historical observations that are used to predict future observations.

The deep learning approach of Rangapuram et al. [2018] explicitly learns state space models for probabilistic predictions about future states. Probabilistic prediction techniques include predicting the parameters of distributions functions [Wang et al., 2019, Salinas et al., 2020] or predicting a multinomial distribution over a quantized representation of the observation space [Wen et al., 2017, Fan et al., 2019, Orozco et al., 2018]. Quantized representations have the advantage of being distribution agnostic, and can readily represent multi-modal distributions, but cannot predict continuous states.

Lim et al. [2020] replace engineered representations in Bayesian filtering with learned components. This can mitigate the problems due to misspecified update models, while gaining the benefits of Bayesian filtering frameworks, potentially yielding better performance for iterative prediction.

Direct multi-step prediction does not feed predictions back to make future predictions, but makes all predictions at once. Wen et al. [2017] predict future observations using an LSTM, fed into a multilayer perceptron (MLP). Fan et al. [2019] combine recurrent networks and attention for direct prediction, and decode the future predictions from a bidirectional LSTM (BiLSTM).

Transformers [Vaswani et al., 2017] have had success in direct multi-step prediction [Lim et al., 2019], however, transformers have a memory complexity penalty. Due to attention mechanisms, the memory requirements of transformer models are functions of the input sequence length. Beyond mainline efforts to reduce complexity mechanisms, *e.g.* [Kitaev et al., 2019, Zaheer et al., 2021], sparse attention mechanisms have improved memory requirements for multi-step time series prediction [Li et al., 2019, Zhou et al., 2020].

Our algorithm distinguishes itself from past work in three ways. First, these prediction methods assume some discrete time steps, whereas the LLP makes predictions at all points in the window $[t, t+\theta_p]$. In principle, LLP should also handle missing data or irregular observations, as the update rule does not expect a particular update frequency, although demonstrating this is left to future work.

Second, where some of the above algorithms reduce complexity through Quantile Regression [Koenker and Bassett, 1978], we reduce complexity by constraining the smoothness of the predicted data. Limiting the degree of the Legendre polynomial representation limits how quickly the predicted signal can change, but it also reduces the computational complexity of the learning algorithm while still predicting continuous

states. Third, while the preceding algorithms *may* be readily converted into an online algorithm, the LLP is designed specifically as an online learning algorithm.

3. Method

To demonstrate our on-line prediction algorithm we use it to learn to predict functions in isolation, as described in Section 3.5. We briefly review Generalized Fourier Coefficients in Section 3.1. LDNs, a key element of the algorithm are described in Section 3.2, and the algorithm itself is described in Section 3.3. Readers seeking a more detailed discussion are referred to Stöckel [2021].

3.1. Generalized Fourier Coefficients

Fourier series are a representation of time-varying functions composed out of a summation of sinusoidal functions.

$$f(x) \approx \frac{a_0}{2} + \sum_{i=1}^{n} a_i \cos\left(\frac{2\pi}{P} nx\right) + b_i \sin\left(\frac{2\pi}{P} nx\right) \tag{1}$$

This concept can be generalized to orthogonal square-integrable function bases, provided a set of basis functions

$$\Phi = \{\phi_n : [a, b] \to \mathbb{Y}\}_{n=0}^{\infty} \tag{2}$$

where $[a, b]$ is the restricted domain of the basis functions[1] and $\mathbb{Y}$ is the set containing the outputs of the function, which may be the real ($\mathbb{R}$) or complex ($\mathbb{C}$) numbers. The bases do not need to be strictly orthogonal, but can be orthogonal with respect to a weighting function $w(\cdot)$:

$$\langle \phi_i, \phi_j \rangle_w = \int_a^b \phi_i(\tau)\phi_j(\tau)w(\tau)d\tau = 0 \quad \forall\, 0 \leq i, j \leq n, i \neq j \tag{3}$$

Using this set of basis functions we can approximate a function:

$$f(x) \approx \sum_{i=0}^{n} a_i \phi_n(x) \tag{4}$$

with the coefficients a_i defined as

$$a_i = \frac{\langle f, \phi_i \rangle_w}{\|\phi_i\|_w^2} \tag{5}$$

There are a number of different functions that can provide a basis for function approximation. However, in this work we will be focusing on the Legendre polynomials, as they have a demonstrated relationship with time cells [Voelker et al., 2019]. In this work we will restrict ourselves to approximating real-valued functions.

[1] The typical definition of Generalized Fourier series defines the basis functions to be defined on the domain $[a, b]$, but WLOG, we rescale it to the domain $[0, 1]$.

3.2. Legendre Polynomials and the Legendre Delay Network

The Legendre Delay Network [LDN; Voelker et al., 2019] is a recurrent neural network that was designed to produce as its output its input from θ seconds ago. The transfer function of such a network, in the Laplace domain, is $\mathrm{e}^{-s\theta}$. A perfect delay is not physically realizable. However, it can be approximated using as a rational function of two polynomials, which, in turn, can be implemented as a dynamical system.

In this work we use properties of Legendre polynomial representations of time varying-functions, and the Legendre Delay Network (LDN) [Voelker et al., 2019] to encode history. Legendre polynomials are orthogonal basis functions that can be used to represent functions over fixed input windows. We use the shifted Legendre basis polynomials, defined by the functions $P_0(t) = 1, P_1(t) = 2t - 1$, and the recursion $(n+1)P_{n+1}(t) = (2n+1)tP_n(t) - nP_{n-1}(t)$. The polynomials are defined over the domain $[0, 1]$, and the coefficients of the Legendre representation of a function $f(t)$ over a window $[t, t+\theta]$ are $a_n = \frac{2n+1}{2}\int_t^{t+\theta} f(\tau)P_n((\tau - t)/\theta)d\tau$. A representation using the first q polynomials is said to have an order of q. In this work we exploit the orthogonality of the Legendre polynomials, that is the $\int_0^1 P_i(t)P_j(t)dt = \frac{1}{2i+1}$ when $i = j$ and zero otherwise.

The LDN is a dynamic system that approximates the Legendre polynomial coefficients of an input signal over a sliding history window of length $\theta \in \mathbb{R}^+$. The coefficients are represented using the LDN's memory state, $m \in \mathbb{R}^q$, for an order q Legendre representation. The memory state m is updated according to $\dot{m}(t) = \mathbf{A}m(t) + \mathbf{B}u(t)$, where $u(t)$ is the input signal. To effect a Legendre basis, $\mathbf{A}$ and $\mathbf{B}$ are defined such that $A_{ij} = \frac{2i+1}{\theta}\begin{cases} -1 & i < j \\ (-1)^{i-j+1} & i \geq j \end{cases}$, and $B_i = \frac{(2i+1)(-1)^i}{\theta}$. The values of $\mathbf{A}$ and $\mathbf{B}$ are fixed once θ and q are selected. For discrete-time applications we approximate $\mathbf{A}$ and $\mathbf{B}$ with $\bar{\mathbf{A}} = e^{\mathbf{A}}$ and $\bar{\mathbf{B}} = A^{-1}(e^A - \mathbf{I})\mathbf{B}$, using a zero-order hold and $dt = 1$ [Stöckel, 2021, Chilkuri and Eliasmith, 2021].

Thus defined, we can use the LDN to represent the history of a variety of signals required for prediction. This includes recent observations of the system, contextually relevant predictors of those observations, and even the predictors past states when making a prediction. We choose the LDN a) because of its high-performance in representing time series and sequences [Voelker et al., 2019, Chilkuri and Eliasmith, 2021] and b) because LDNs have only two free parameters, the dimensionality of the representation, q, and the length of history window, θ.

3.3. Derivation of the Online Learning Algorithm using Generalized Fourier Coefficients

The objective of the learning algorithm is to predict, from a history of observations, future observations, $\hat{z}(t) \in \mathbb{R}^m$, over the time window $[t, t+\theta_p]$, where $\theta_p \in \mathbb{R}^+$. The history consists of context vectors, $c(t) = (z(t), u(t))$, over the time window $[t-\theta_h, t]$, consisting of plant observations, $z(t)$, and issued commands $u(t)$.

We wish to predict the future observations as the coefficients of a generalized Fourier series. Given a set of orthogonal basis functions, $\Phi = \{\phi_n : [0,1] \to \mathbb{R}\}_{n=0}^{\infty}$. We can represent the predicted function, $\hat{z}(\tau) = \sum_{n=0}^{q_p} \hat{Z}_n \phi_n(\tau)$ on the domain $[t, t+\theta_p]$, where

$$\hat{Z}_n = \frac{1}{\|\phi_n\|_w^2} \int_0^1 \phi_n(\tau) z\left(t + \tau\theta_p\right) w_n(\tau) d\tau$$

and $w(\tau)$ is the weighting function for the chosen function basis that ensures orthonormality.

We wish to predict from recent history the generalized Fourier coefficients, $\hat{Z}(t)_m^{q_p}$, written here as a $q_p \times m$ tensor using Einstein notation, based on some recent history, represented as the state of the context LDN, $C(t)_{q_h,d_c}$. We use a neural network with a single hidden layer to predict the state, such that $\hat{Z}(t)_m^{q_p} = D_m^{q_p N} a(t)_N$, where $a(t)_N$ is the activity of a population of N neurons, and $a(t)_N = f(C(t)_{q_h,d_c})$. The weights and biases for the neurons in $f(\cdot)$, are chosen randomly, using the techniques lade out in the Neural Engineering Framework, described above, and are held constant during learning. The only parameters to be learned are the decoding weights, $D_m^{q_p N}$. To learn these weights we employ the Delta rule [Widrow and Hoff, 1960].

If one were training the data in batch mode, operating for T seconds, and had a sequence of observations $\left\{(c(t_{\text{obs}})^T, z(t_{\text{obs}})^T)^T\right\}$ for $t_{\text{obs}} \in [0, T]$, then the update to the decoder weights at the time the prediction is made, $t_{\text{pred}} \in [0, T - \theta_p]$, is:

$$\Delta D(t_{\text{pred}})_m^{q_p N} = -\kappa a(t_{\text{pred}})^N \times \int_0^1 \left(\hat{z}\left(t_{\text{pred}} + \tau\theta_p\right)_m - z\left(t_{\text{pred}} + \tau\theta_p\right)_m\right) P(\tau)^{q_p} W(\tau)_{q_p}^{q_p} d\tau \tag{6}$$

Where $P(\tau)^{q_p} = (\phi_0(\tau), \ldots, \phi_{q_p}(\tau))^T$, is the vector of orthogonal bases[2], and $W(\tau)_{q_p}^{q_p} = \text{diag}\left(\frac{w(\tau)}{\|\phi_0\|_w^2}, \ldots, \frac{w(\tau)}{\|\phi_{q_p}\|_w^2}\right)$ is a diagonal matrix of the scale factors for computing the generalized Fourier coefficients. The updated weight matrix after the most recent observation is:

$$D(t_{\text{obs}}) = D(0) + \int_0^{t_{\text{obs}} - \theta_p} \Delta D(t) dt \tag{7}$$

which is a double integral over the time the prediction is made, t_{pred}, and how far into the future we are predicting observations, t_{obs}. Unfortunately, Eq. (6) is acausal with respect to the prediction time t_{pred}, but this can be rectified by extracting the inner integral, re-writing the integration, and using the Legendre representation:

$$D(t_{\text{obs}}) = D(0) + \int_0^1 \int_{\tau\theta_p}^{t_{\text{obs}} - \tau\theta_p} -\kappa a(t) \times \left(\hat{Z}(t) P(\tau)_{q_p} - z(t + \tau\theta_p)\right) P(\tau)_q W(\tau)_q^q dt d\tau \tag{8}$$

[2] This derivation is completed relying on the basis functions' orthogonality. However, when implemented using basis function generating LTI systems, some approximation error will be introducted. See §4.2 of [Stöckel, 2022].

To figure out what the instantaneous update to the weight matrix would be, we take the derivative of $D(t_{\text{obs}})$ with respect to t_{obs}. Using Leibniz's rule we find

$$\begin{aligned}\frac{d}{dT}\left[D(T)\right] = \int_0^1 &- \kappa a(T-\tau\theta_p)\times \\ &\left(\hat{Z}(T-\tau\theta_p)P(\tau)_{q_p} - z(T-\tau\theta_p+\tau\theta_p\right)P(\tau)_q W(\tau)_q^q d\tau\end{aligned} \tag{9}$$

We can rewrite the term $z(T-\tau\theta_p+\tau\theta_p)$ such that only the current observation, $z(T)$ is required. The term $\hat{Z}(T-\tau\theta_p)P(\tau)_q$ is the prediction made $\tau\theta_p$ seconds ago about an observation $\tau\theta_p$ seconds into the future. This formulation updates the decoder weights using only the current observation, $z(t)$, but it depends on the history of the neural population, $a(t)$ and the predicted generalized Fourier coefficients, $\hat{Z}(t)$. We can represent these histories using two more sets of generalized Fourier coefficients, one for the neural population activity, $a(T-\tau\theta_p) = A(T)_{q_a}^N P(\tau)_{q_a}$, and one for the predicted Legendre coefficients, $\hat{Z}(T-\tau\theta_p) = M_{\hat{Z}}(T)P(\tau)_{q_{\hat{Z}}}$. To tidy up notation, we rename the variable T as t. Because $M_{\hat{Z}}(t) \in \mathbb{R}^{q_{\hat{z}}\times q_p \times m}$ and $A(t) \in \mathbb{R}^{N\times q_a}$ are tensors, we also rewrite the equation using Einstein notation. This lets us re-write the learning rule as:

$$\begin{aligned}\frac{d}{dt}\left[D(t)_N^{q_p m}\right] = \int_0^1 &- \kappa A(t)_{N,q_a} P(\tau)^{q_a} \\ &\left(M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq} P(\tau)^{q_{\hat{z}}} P(\tau)_{q_p} - z(t)_m\right) P(\tau)^q S_q^q d\tau\end{aligned} \tag{10}$$

Where q_a is the dimensionality of the Legendre representation of the neural population activity, $a_{\hat{Z}}$ is the dimensionality of the Legendre representation of the history of predicted observations, and q_p is the dimensionality of Legendre representation of the predicted observations. The variable q is the dimensionality that the error signal is being projected into, and should be identical to q_p. However, since κ, $A(t)_N^{q_p m}$, $M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq}$, and $z(t)^m$ are not functions of τ, they can be taken out of the integration, leaving us with the expression

$$\begin{aligned}\frac{d}{dt}\left[D(t)_N^{qm}\right] = -\kappa A(t)_{N,q_a}\Bigg(&M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq_p}\int_0^1 P(\tau)^{q_a}P(\tau)^{q_{\hat{z}}}P(\tau)_{q_p}P(\tau)^q W(\tau)_q^q d\tau \\ &- z(t)^m \int_0^1 P(\tau)^{q_a}P(\tau)^q W(\tau)_q^q d\tau\Bigg)\end{aligned} \tag{11}$$

Fortunately, the terms under the integral signs are only dependent on the choice of the representation dimensions, and can be pre-computed before the algorithm is run. We denote the first term $Q_{q_p}^{q_a q_{\hat{Z}} q} = \int_0^1 P(\tau)^{q_a}P(\tau)^{q_{\hat{z}}}P(\tau)_{q_p}P(\tau)^q W(\tau)_q^q d\tau$ and the second term by $S = \int_0^1 P(\tau)^{q_a}P(\tau)^q W(\tau)_q^q d\tau$. Regardless of the choice of generalized Fourier basis functions, Q is a term that only depends on τ, and can be pre-computed. Should $P(\tau)^{q_a}$ and $P(\tau)^q$ correspond to the same orthogonal generalized Fourier basis functions, and because the integral is scaled by the matrix $W(\tau)_q^q$, S should evaluate to a $q_a \times q$

submatrix of the identity matrix. In our case, we use the Legendre Polynomials as the basis functions, and we re-write the learning rule as:

$$\frac{d}{dt}\left[D(t)_N^{qm}\right] = -\kappa A(t)_{N,q_a}\left(M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq_p} Q_{q_p}^{q_a q_{\hat{Z}} q} - z(t)^m I^{q_a q}\right) \tag{12}$$

The implementation of the LLP learning rule is given in Algorithm 1. The update rule for the matrix D requires $mq_aq_pq_{\hat{Z}}^2 + Nmq_aq_{\hat{Z}}$ floating point operations for each update.

Algorithm 1 Learning Legendre Prediction - online learning for multistep prediction.

procedure LLP-INIT($m, \theta_p, q_p, N, r, \kappa, \theta_c, q_c, q_z, \Delta t$)
  $t \leftarrow 0$
  $\bar{A}_c, \bar{B}_c \leftarrow \text{LDN}(\theta_c, q_c, \Delta t)$; $C_0 \leftarrow 0^{m\times q_c}$ ▷ Initialize LDN matrices and state vectors.
  $\bar{A}_{\hat{Z}}, \bar{B}_{\hat{Z}} \leftarrow \text{LDN}(\theta_p, q_z, \Delta t)$; $M_{\hat{Z},0} \leftarrow 0^{m\times q_z}$
  $\bar{A}_a, \bar{B}_a \leftarrow \text{LDN}(\theta_p, q_z, \Delta t)$; $A_0 \leftarrow 0^{N\times q_z}$
  $D \leftarrow 0^{N\times m\times q_p}$ ▷ Initialize Weight Matrix
  $a(0), W, B \leftarrow$ nef_ensemble(N, r) ▷ Randomly initialize neural population.
  $Q \leftarrow \int_0^1 P(\tau)^{q_a} P(\tau)^{q_{\hat{z}}} P(\tau)_{q_p} P(\tau)^q S_q^q d\tau$ ▷ Precompute Q tensor
end procedure
procedure LLP-UPDATE($c(t), z(t), \Delta t$)
  $C_t \leftarrow \left(I + \bar{A}_c\right) C_{t-1} + \bar{B}_c c(t)$ ▷ Update LDN states.
  $M_{\hat{Z},t} \leftarrow \left(I + \bar{A}_{\hat{Z}}\right) M_{\hat{Z},t-1} + \bar{B}_{\hat{Z}} \hat{Z}_t$
  $A_t \leftarrow (I + \bar{A}_a) A_{t-1} + \bar{B}_a a(t)$
  $D \leftarrow D - \kappa A_t \times \left(M_{\hat{Z},t} Q - z(t) I\right)$ ▷ Update weight matrix.
  $a(t) \leftarrow f(WC_t + B)$ ▷ Predict Legendre coefficients.
  $\hat{Z}_t \leftarrow Da(t)$
  $t \leftarrow t + \Delta t$
end procedure

3.4. Comparison to Naïve Implementation

An obvious implementation of an online learning rule would be to maintain a queue of the last $N_p = \theta_p \omega_z$ neural population activities and predicted outputs, where ω_z is the sampling frequency of the observations, and then update the decoder matrix for each new observation. Here we analyze the memory and time complexity of the LLP to determine at what point it is more efficient than the naïve implmentation.

In the naïve implementation we require a queue of length N_p to store copies of the neural network activities, $a(t)$ and the predicted outputs in Legendre space, $\hat{Z}(t)$, for each dimension, m. We also need to store the queue of observations, of m dimensions, of length N_p. This gives a total memory complexity of $N_p(N_a + m(q_p + 1))$.

For the LLP we require one LDN to keep track of the neural activities, which will require a memory of $N_a q_a$. We will also require an LDN to keep track of the predicted Legendre coefficients, $mq_p q_{\hat{Z}}$, where m is the dimensionality of the data, q_p is the number of Legendre coefficients used to represent the predictions, and $q_{\hat{Z}}$ is the dimensionality of the LDN keeping a history of prior predicted Legendre coefficients, $\hat{Z}(t)$. If we let $q = \max\{q_p, q_{\hat{Z}}, q_a, q\}$, the LLP has preferable memory properties when:

$$\begin{aligned}
&mq^2 + N_a q < N_p(N_a + m(q+1)) \\
&\implies mq^2 + N_a q - N_a N_p - mN_p q - mN_p < 0 \\
&\implies mq^2 + (N_a - mN_p)q - (N_a N_p + mN_p) < 0 \\
&\implies q^2 + (N_a/m - N_p)q - (N_a N_p/m + N_p) < 0
\end{aligned}$$

With equality being the threshold for dominance, we can consider the upper root of the quadratic, which occurs at:

$$\begin{aligned}
q &= \frac{-(N_a/m - N_p) + \sqrt{(N_a/m - N_p)^2 - 4(-N_a N_p/m - N_p)}}{2} \\
q &= \frac{N_p - N_a/m + \sqrt{(N_a/m + N_p)^2 + 4N_p}}{2}
\end{aligned}$$

We can look at the sensitivity of this criterion to the different model parameters, to determine the maximum q where the LLP has better memory characteristics. Fig. 1 shows a sensitivity of q to the parameters, m, N_p, and N_a. We can see that this term is domniated by the term N_p.

The time complexity of the naïve implementation is the number of samples times the cost of updating the weight matrix for each element in the history, $N_p \times mN_a q_p$. For the LLP algorithm, the update rule is: $\Delta D(t)^N_{q_r m} = -\kappa A^{N_a}_{q_a}(t)\left(M(t)^{q_p}_{mq} Q^{q_a q}_{q_p q_r} S^{q_r}_{q_r} - z_m(t)\delta^{q_a}_{q_r}\right)$. To compute this we need to compute two tensor products, $M(t)^{q_p}_{mq}$ with the pre-computed $q_a \times q \times q_r$ tensor, $Q^{q_a q}_{q_p q_r} S^{q_r}_{q_r}$, and then $A^{N_a}_{q_a}(t)$ with the $m \times q_a \times q_r q q_p$ output of the previous tensor product. This gives a computational complexity of the LLP update rule of $mq_p q_a q_r + mN_a q_q q_r$. Our algorithm is more efficient when $mq_p q_a q_r q + mN_a q_q q_r < N_p \times mN_a q_p$. If we let $q' = \max\{q, q_a, q_r, q_p\}$, then we can say that our algorithm is more computationally efficient when $q'^3/N_a + q' < N_p$, or when $q'^3/N_a + q' - N_p < 0$, the crossover point occurring when the expression is equal to zero. A plot of the maximum q' permitted as a function of N_a and N_p is given in Fig. 2.

3.5. Experimental Setup

We test the LLP first in isolation, predicting two different time series (Section 3.5.1).

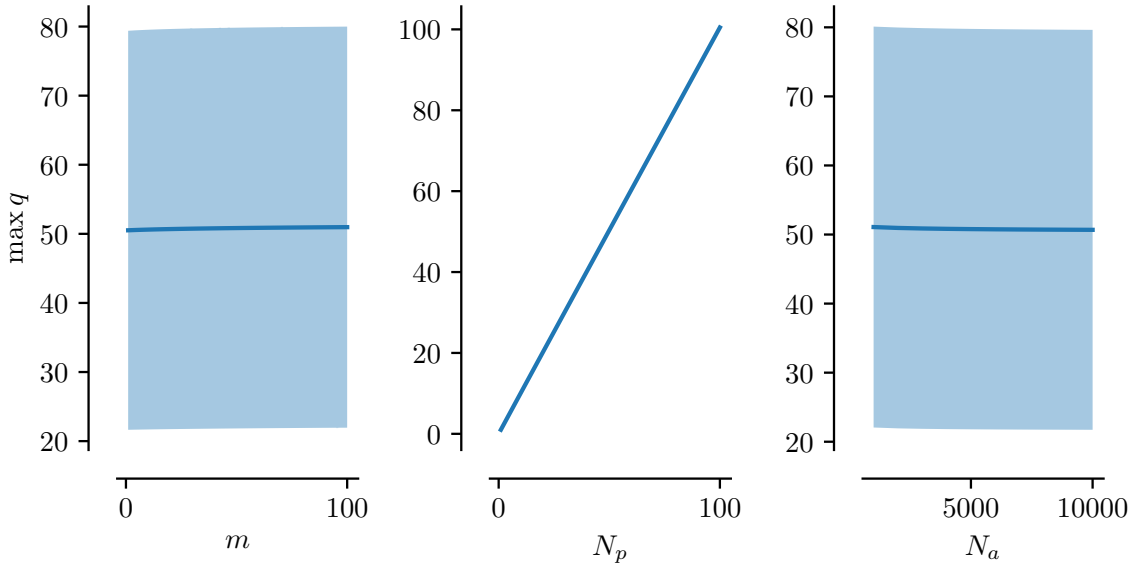

Figure 1: The sensitivity of the maximum q before LLP loses memory dominance over the naïve implementation. The plot shows the maximum value of q that can be used before the naïve implementation has better memory performance. The solid line is the average maximum q value and the shaded region is a 95% confidence interval. That threshold is dominated by the number of observations into the future that were being predicted, N_p.

3.5.1. Function Prediction To test the LLP in isolation, we use it to predict the Mackey-Glass equation[Mackey and Glass, 1977] and the Lissajous Curve[Bowditch, 1815]. The Mackey-Glass equations were developed to model the number of mature blood cells in blood, although through different parameter settings have been used to model a number of homeostatic processes. The function being predicted is a quantity $x(t)$, which is a function of the parameters β_0, γ, n, and τ, has an initial condition $x(0) = 0.1$, and is defined by Eq. (13).

$$x(t) = \frac{\beta_0 x(t-\tau)}{1 + x(t-\tau)^n} - \gamma x(t-\tau) \tag{13}$$

The Lissajous Curve is a system of parametric equations that can describe the motion of a compound pendulum. The motion is described by the system of parametric equations:

$$x_1(t) = A\sin(at + \delta) \tag{14}$$
$$x_2(t) = B\cos(bt) \tag{15}$$

In this experiment we set $A, B = 1$, and $a = 5$, $b = 4$, and $\delta = \frac{\pi}{4}$. For these two tests the LLP parameters are: the context history window, $\theta_c = 0.3$ sec; context Legendre order $q_c = 20$; learning rate $\kappa = 5 \times 10^{-5}$; readout point $\tau = 1$; neural population size $N = 2000$; prediction window length $\theta_p = 0.015$; and the prediction Legendre order

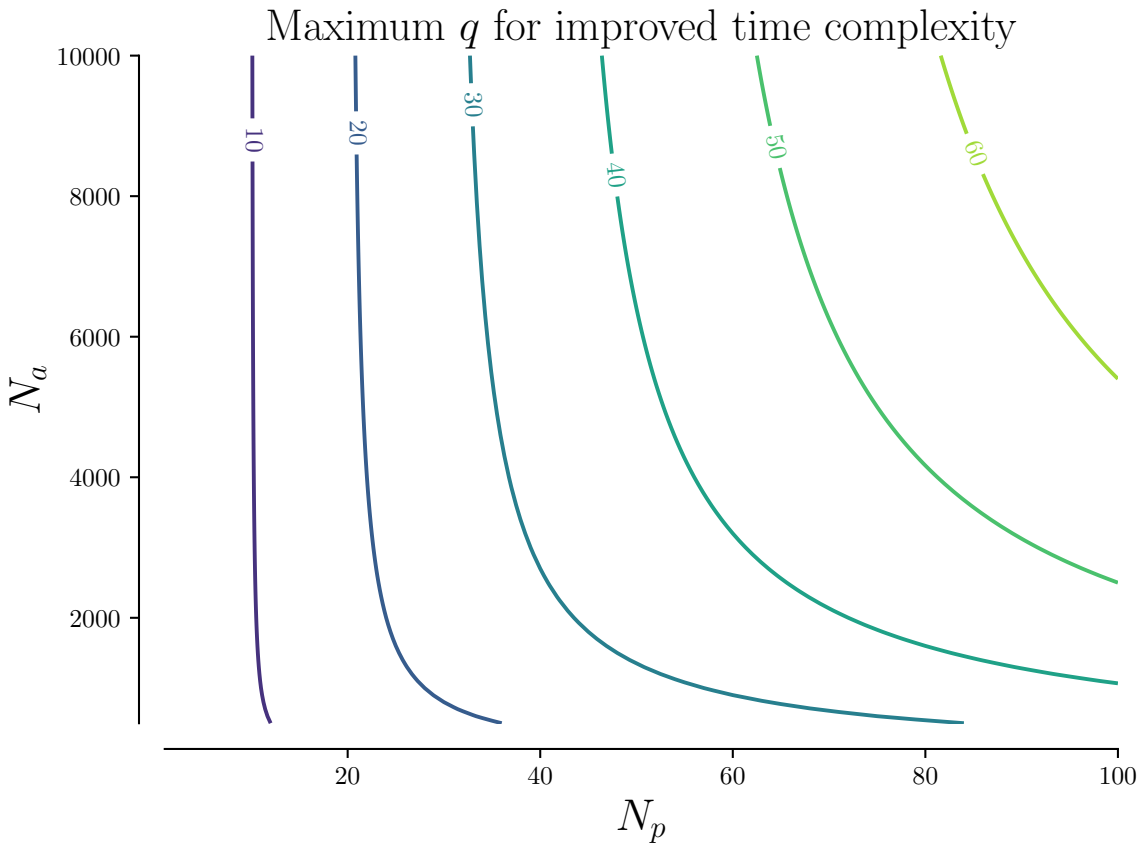

Figure 2: A contour plot of the maximum q before LLP loses time complexity dominance over the naïve implementation. The numbers on the contour lines show the maximum value of q for different values of the number of samples, N_p, and the size of the neural network, N_a. By inspection, we see the algorithm is more sensitive to the number of prediction samples than it is to the size of the prediction network.

$q_p = 20$. We also set the Legendre order for the LDNs that track the network history – $a(t)$ and $\hat{Z}(t)$ – to be equal to the prediction Legendre order, $q_p = q_z$.

3.6. Baseline Algorithms

We compared the LLP against a model trained offline, using $428k$ samples of the Mackey-Glass and Lissajous datasets, with a $70, 20, 10\%$ training, validation, and test split, for one epoch. This split and limited epochs ensures the offline models are trained on as much data as the LLP is exposed to during operations. As a baseline, we use an LSTM model with 32 Hidden Units, a dense layer of 2000 ReLU neurons, and then a dense output layer of $\frac{\theta_p}{\Delta t} \times m$ neurons. The network was trained with a batch size of 128, once using the Adam optimizer, and again using Stochastic Gradient Descent with a learning rate of 5×10^{-5}, the same learning rate the LLP used.

We also compared this to an off-line equivalent of the lds algorithm. We collected training data, and extracted the Legendre coefficients of the context window, $C =$

$\text{leg}(\mathbf{x}(t-\theta_c),\ldots,\mathbf{x}(t))$, and for the prediction window, $Z = \text{leg}(\mathbf{x}(t),\ldots,\mathbf{x}(t+\theta_p))$. We then trained a single hidden layer neural network with ReLU neurons, to predict the future state, Z from the context, C. We then compute the Mean Absolute Error of the predicted value, $\hat{Z}P(1)$.

4. Results

The results of using LLP in isolation are shown for the Mackey-Glass dataset in Fig. 3 and the Lissajous curve in Fig. 4. In both graphs, the solid blue line is the average over $N = 30$ trials (shaded regions represents a 95% confidence interval) of the sliding window average (window size of $10k$ timesteps) of the LLP's prediction error for the 15 time steps past the current observation. All networks were tested on the functions starting from the same starting point, with variability due to weight initialization. The horizontal dashed lines represent the testing error of the offline-trained LSTM baseline models, for $N = 30$ randomly initialized networks, shaded regions represent a 95% confidence interval.

For both the Mackey-Glass and the Lissajous functions, the LLP outperforms the LSTM-SGD model. However, while the performance of LLP approaches the performance of LSTM using the Adam optimizer, it does not achieve equivalence in the case of the Mackey-Glass dataset.

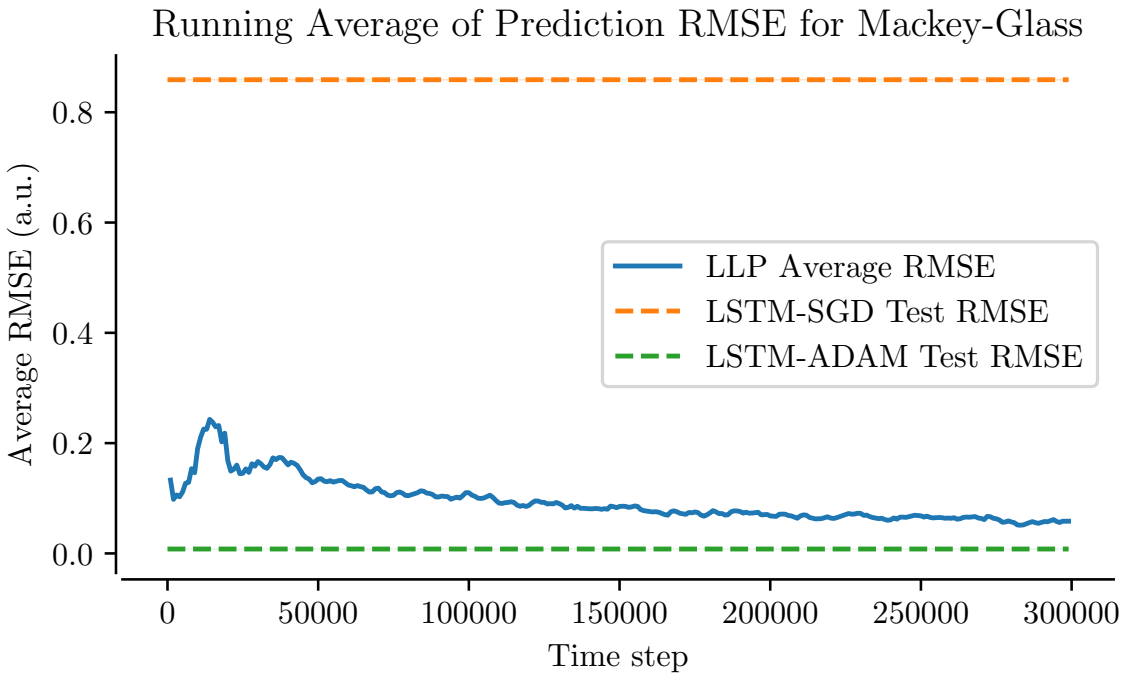

Figure 3: The prediction error of LLP vs time step. The dashed horizontal line gives the test error of an LSTM model trained on the Mackey-Glass dataset.

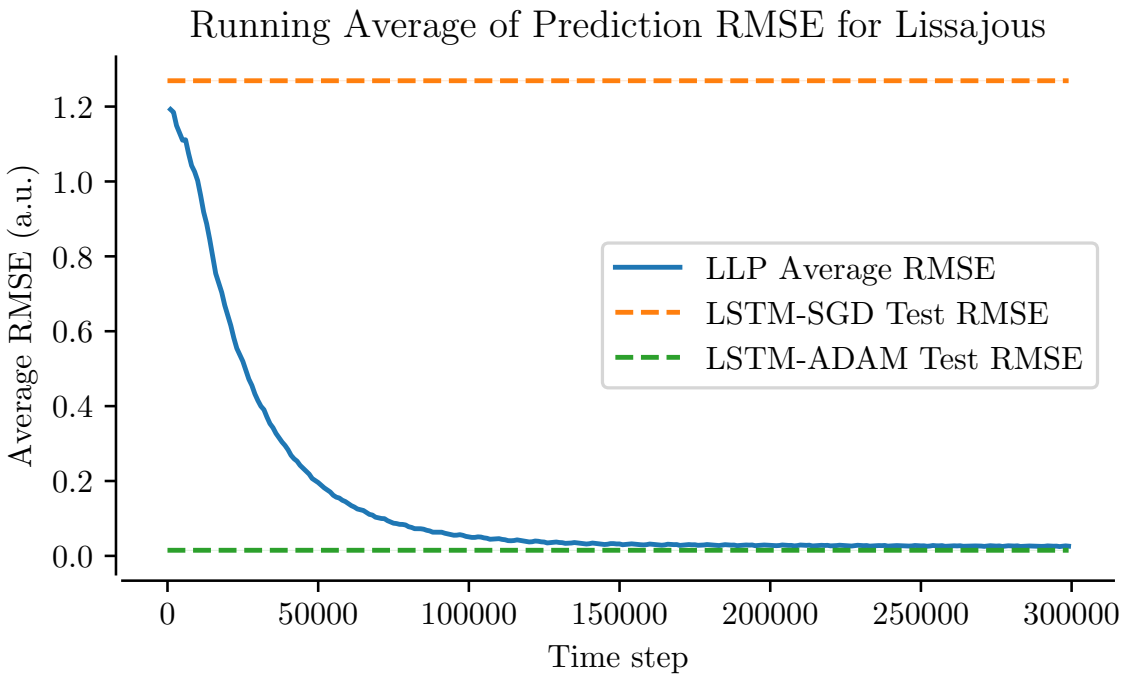

Figure 4: The prediction error of LLP vs time step. The dashed horizontal line gives the test error of an LSTM model trained on the Lissajous function using SGD and Adam optimizers.

5. Discussion

The results of these experiments demonstrate the ability of the algorithm to learn to predict functions, online. When predicting the Mackey-Glass and Lissajous functions the LLP learns online to predict the functions with an RMSE error that is better than that of the LSTM-SGD model. These algorithms had the same learning rate, and so the LSTM-SGD model may be disadvantaged by having more free parameters to optimize, compared to the LLP. The performance of the LLP is inferior to the LSTM-Adam model, although the performance of the LLP does tend towards that of the Adam-optimized model. This motivates complementing the LLP learning rule with adaptive learning rates and momentum. In either case, we may conclude that LLP is learning to predict the function it is trained on, and that its performance is comparable to models trained using standard offline methods, when exposed to similar volumes of data.

5.1. Selecting Algorithm Parameters

We do not address the question of how to select the parameters, the q's and θ's for the different LDNs, as well as the dimensionality of the prediction network's hidden layer, N_a. If the problem's sampling rate, r, and the prediction window, θ_p, are fixed, and we assume that the sampling rate is the Nyquist sampling rate, then we can set the order Legendre of prediction coefficients, q_p, and context networks, q_c, q_a, and $q_{\hat{Z}}$, to be less

than $r\theta_p$, to achieve an acceptable level of reconstruction error. The order, q, can be determined based on the frequency content of the signal, see Figure 12 of Stöckel [2021]. The choice of prediction window, θ_p, constrains the θ. parameters of the LDNs that store the prediction network activity, $a(t)$, and the coefficients of the predicted signal, $\hat{Z}$. If one can otherwise bound the Nyquist frequency of the system, then the context and prediction Legendre dimensionality can be selected, given a sampling frequency. The choice of N_a is hyperparameter to be optimized, as is the dimensionality of the LDNs that store the hidden layer activity, $a(t)$, and predictions, $\hat{Z}$. An interesting avenue of future work is to modify the system structure online, either by expanding the structure (larger q, larger N_a) to improve prediction performance, or pruning the network to improve efficiency.

5.2. Representing Multivariate Data

We have not explicitly addressed the problem of representing multivariate data, *i.e.* $m > 1$. A multidimensional LDN is essentially equivalent to maintaining one LDN per dimension, in the sense that there is no cross-talk between the input dimensions. There is a question of how to most efficiently represent multivariate data, especially when the desire is to represent and predict systems where there is higher-order (non-linear) interactions between dimensions, like in the double pendulum. There is an open question of whether this is the most efficient way to represent multivariate data.

For some desired prediction accuracy, there should be an LDN dimensionality that captures sufficient information to reconstruct the signal. Appealing to the universal approximation theorem, then given that LDN representation and some single hidden layer neural network, we should be able to predict the system to some desired accuracy.

Take, for example, the system $\dot{x_1} = x_1(t) + x_1(t)x_2(t) + x_2(t)$. If $x_1(t)$ and $x_2(t)$ have bandwidth B, then $\dot{x}$ has a bandwidth $2B$. If we were to represent the term $x_1(t)x_2(t)$ separately, we would need twice the LDN dimensionality to achieve the Nyquist sampling frequency compared to just x_1 and x_2 alone. If the product term is not represented separately, then the prediction network must perform the multiplication, making the prediction a harder problem. So there must be some trade-off between the complexity of the LDN representation and the complexity of the prediction network.

If we were to add second-order orthogonal polynomials to the basis set, we would make the problem easier for the prediction network, but at the cost of increasing the complexity of the recurrent context network.

Additionally, we may have some pre-processing stage where variables can be mixed into signals that can be represented independently. It may be more efficient to pass observable variables through a transformation, *e.g.* ICA, or the hidden state of some neural network, before passing it into the context LDN. What the right architecture is for this network requires additional exploration, although we observe that in the non-neural case it should be easy to propagate a gradient back through an LDN to learning a pre-network encoding stage.

6. CONCLUSIONS

We have described an online algorithm for learning multi-step prediction of continuous states, over a continuous time window. We have used this algorithm as an isolated predictor with observation latency. Future work involves using this algorithm to predict systems for control, both feedback and model predictive applications.

References

N. Bowditch. On the motion of a pendulum suspended from two points. *American Academy of Arts and Sciences, Boston. Memoirs of the American Academy of Arts and Sciences (1785-1902)*, 3(2):413, 1815.

G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

S. Chen and S. A. Billings. Representations of non-linear systems: the narmax model. *International journal of control*, 49(3):1013–1032, 1989.

G. Chevillon. Direct multi-step estimation and forecasting. *Journal of Economic Surveys*, 21(4):746–785, 2007.

N. R. Chilkuri and C. Eliasmith. Parallelizing legendre memory unit training. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 1898–1907. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/chilkuri21a.html`.

C. Eliasmith and C. H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

C. Fan, Y. Zhang, Y. Pan, X. Li, C. Zhang, R. Yuan, D. Wu, W. Wang, J. Pei, and H. Huang. Multi-horizon time series forecasting with temporal attention learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2527–2535, 2019.

N. Kitaev, L. Kaiser, and A. Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2019.

R. Koenker and G. Bassett. Regression quantiles. volume 46, 1978.

I. Leontaritis and S. A. Billings. Input-output parametric models for non-linear systems part i: deterministic non-linear systems. *International journal of control*, 41(2):303–328, 1985a.

I. Leontaritis and S. A. Billings. Input-output parametric models for non-linear systems part ii: stochastic non-linear systems. *International journal of control*, 41(2):329–344, 1985b.

S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, 32:5243–5253, 2019.

B. Lim, S. O. Arik, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint arXiv:1912.09363*, 2019.

B. Lim, S. Zohren, and S. Roberts. Recurrent neural filters: Learning independent bayesian filtering steps for time series prediction. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

M. C. Mackey and L. Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.

B. P. Orozco, G. Abbati, and S. Roberts. Mordred: Memory-based ordinal regression deep neural networks for time series forecasting. *arXiv preprint arXiv:1803.09704*, 2018.

S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31:7785–7794, 2018.

D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

A. Stöckel. Discrete function bases and convolutional neural networks. *arXiv preprint arXiv:2103.05609*, 2021.

A. Stöckel. *Harnessing Neural Dynamics as a Computational Resource*. Phd thesis, University of Waterloo, Waterloo, ON, 2022. URL `http://hdl.handle.net/10012/17850`.

S. B. Taieb and A. F. Atiya. A bias and variance analysis for multistep-ahead time series forecasting. *IEEE transactions on neural networks and learning systems*, 27(1): 62–76, 2015.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 6000–6010, 2017.

A. R. Voelker, J. Gosmann, and T. C. Stewart. Efficiently sampling vectors and coordinates from the n-sphere and n-ball. Technical report, Centre for Theoretical Neuroscience, Waterloo, ON, 01 2017. URL `https://www.researchgate.net/publication/312056739_Efficiently_sampling_vectors_and_coordinates_from_the_n-sphere_and_n-ball`.

A. R. Voelker, I. Kajić, and C. Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. *33rd Conference on Neural Information Processing Systems*, 2019.

Y. Wang, A. Smola, D. Maddix, J. Gasthaus, D. Foster, and T. Januschowski. Deep factors for forecasting. In *International Conference on Machine Learning*, pages 6607–6617. PMLR, 2019.

R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. A multi-horizon quantile recurrent forecaster. *NIPS 2017 Time Series Workshop*, 2017.

B. Widrow and M. E. Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

P. R. Winters. Forecasting sales by exponentially weighted moving averages. *Management science*, 6(3):324–342, 1960.

M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed. Big bird: Transformers for longer sequences, 2021.

H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *arXiv preprint arXiv:2012.07436*, 2020.