An Online Infinite-step Prediction Algorithm Applied to Feedback Control with Delay

Anonymous Author(s)
Affiliation
Address
email

Abstract: We describe a new algorithm, Learned Legendre Predictor (LLP), for learning to predict future observations of non-linear systems. The algorithm distinguishes itself from prior work in this area by making continuous time predictions, and by learning online. The algorithm is applied to the problem of controlling a single-link pendulum with observation latency, generating a variation on the Smith predictor that is robust to some variation in dead time, reducing the need for precise latency estimation.

Keywords: Multi-step prediction, neural network, online dynamics learning

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 finance, weather, and supply and demand modelling. With accurate predictions then the appropriate domain-specific actions can be taken.

Similarly, control problems have a need to predict systems' states, to compensate for control loop latencies or to engage in model predictive control. The quality of these systems depends on the accuracy of their predictive models, which traditionally are either trained offline, or are engineered analytical models that are updated online. Offline training limits the adaptability of predictive systems, while rigid analytical models cannot adapt to unanticipated changes in the plant or environment.

In this paper, we develop the Learned Legendre Predictor (LLP) algorithm, a neural network learning algorithm that learns multistep predictions in an efficient online manner. The general approach is to store the past history that is needed by the learning algorithm in a compressed format, and then directly use the compressed representation to perform the weight updates. In particular, we use Legendre polynomials as a compressed representation; rather than representing some sampling of the value over $[t, t+\theta_p]$, we instead encode the coefficients of Legendre polynomials that produce the desired trajectory. Importantly, we use this same representation to encode the output prediction, the past history the prediction is based on, the past history of the activity of the neural network, and even the past history of the output predictions themselves. These last two histories are needed to 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 the utility of this algorithm in the context of a Smith predictor [1] controlling a pendulum operating with observation latency, τ_{obs}. Smith predictors compensate for latency by predicting the plant's state τ_{obs} seconds into the future, relative to the latest observation, $z(t)$. The quality of the Smith predictor is limited by the quality of the plant state prediction, $\hat{x}(t)$, and the accuracy of the observation latency estimate, $\hat{\tau}_{obs}$. LLP can address both these issues. Model quality is addressed by continually improving the plant model through online learning. LLP demonstrates a degree of robustness to latency estimation errors, as long as τ_{obs} is in the prediction window, rendering latency estimation less critical.

In order to learn dynamics 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; 2], which can optimally store a history of observations. An LDN is a linear system $\dot{m} = Am + Bc$, 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 that only has non-linearities in the hidden layer. We use a Rectified Linear Unit for the hidden layer, 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 networks' 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; 3, 4] as a useful fixed representation across a wide variety of non-linear functions. The only parameter for this process is r, the maximum magnitude of the input vector.

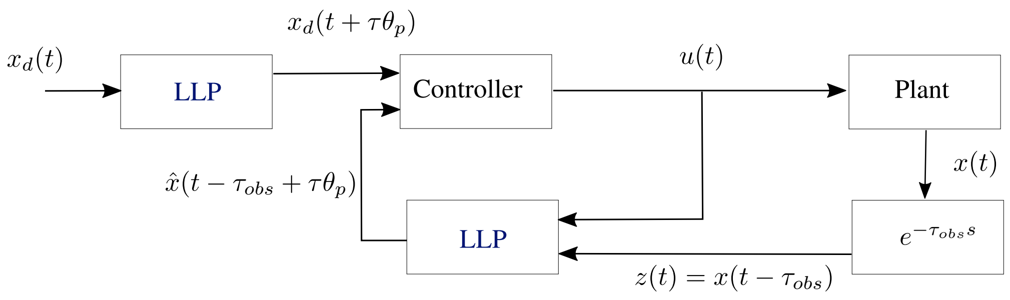

Figure 1: Our approach uses the LLP algorithm to predict the state of the plant in the Smith predictor framework. Another instance of LLP predicts the desired target trajectory to ensure that the observed state approximates the target trajectory, $z(t) \approx x_d(t)$, instead of the actual state approximating the target trajectory, $x(t) \approx x_d(t)$, which is the case for the normal Smith predictor.

2 Prior Work

Multi-step prediction Multi-step, or multi-horizon, prediction breaks down broadly into two approaches: *iterative* and *direct* prediction methods [5]. 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, but there are conditions where direct methods are better, such as when the iterative step is inaccurately modelled [5, 6].

Autoregressive models are the prototypical multi-step prediction method [7, 8], however these models have difficulties predicting non-linear signals. A variety of non-linear autoregressive models exist [e.g., 9, 10, 11], but the adoption of deep learning methods has allowed exploration of other 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.

Rangapuram et al. [12] explicitly learn state space models for probabilistic predictions about future state. Probabilistic prediction techniques include predicting the parameters of distributions functions [13, 14] or predicting a multinomial distribution over a quantized representation of the observation space [15, 16, 17]. Quantized representations have the advantage of being distribution agnostic, and can readily represent multi-modal distributions, but lose the ability to predict continuous states.

Lim et al. [18] 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.

Unlike iterative predictions, direct multi-step prediction does not feed predictions back into the network to aid future predictions. Wen et al. [15] summarize observations with an LSTM network, and use the LSTM state to predict future observations with an multilayer perceptron (MLP). Fan et al. [16] combines recurrent networks and attention for direct prediction, and decode the future predictions from a bidirectional LSTM (BiLSTM).

Transformers [19] have had success in direct multi-step prediction [20], 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.* 21, 22], sparse attention mechanisms have improved memory requirements for multi-step time series prediction [23, 24].

Our algorithm distinguishes itself from this work in three ways. First, these prediction methods assume some discrete time steps, whereas 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 [25], 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 to being online, our algorithm is designed specifically as an online learning algorithm.

Control with latency Dead time compensation is a long-standing control systems problem [1]. The Smith predictor enables feedback control in the face of observation latency, provided the plant model and the dead time are well understood, [e.g., 26, 27]. Learned models have been used where explicit modelling is challenging, although they are typically generated offline from data [28] or calibration [29]. Smith predictors continue to be used in robotics because of their simplicity and effectiveness. Enabling online learning of the plant dynamics and enabling a tunable prediction window makes it easier to deploy this control paradigm.

3 Method

To demonstrate our on-line prediction algorithm we use it in a Smith predictor framework, controlling a single-link pendulum without gravity, in the presence unknown observation latency, τ_{obs}. LDNs, a key element of the algorithm are described in Section 3.1, the algorithm itself is described in Section 3.2, and the experimental setup in Section 3.3.

3.1 Legendre Polynomials and the Legendre Delay Network

In this work we use properties of Legendre polynomial representations of time varying-functions, and the Legendre Delay Network (LDN) [2] 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 $(n+1)P_{n+1}(t) = (2n+1)P_n(t) + nP_{n-1}(t)$. 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. 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 $dt = 1$.

3.2 Derivation of the Learning Algorithm

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, $c(t)$ over the time window $[t-\theta_h, t]$, where $\theta_h \in \mathbb{R}^+$, represented with an LDN[2] as $C(t)$. In our application, $c(t) = (z(t)^T, u(t)^T)^T$ is the concatenation of $z(t)$, the plant observations, and $u(t)$, the issued commands. The predicted observations, $\hat{z}([t, t+\theta_p])$, are represented as q_p^{th}-order Legendre coefficients, $\hat{Z}(t) \in \mathbb{R}^{m \times q_p}$. Specifically, we wish to learn a function $\hat{Z}(t) = g(C(t))$.

We choose $g(\cdot)$ to be a one layer neural network, $\hat{Z}(t) = Da(t)$, where $D \in \mathbb{R}^{m \times q_p \times N}$ is the decoding tensor of the hidden layer activity, $a(t)$, of a population of N neurons. The population activity is a non-linear function of the context, $a(t) = f(WC(t) - B)$, where W is the input connection weights, B is the bias, and $f(\cdot)$ is the ReLU nonlinearity. W and B are selected as described in Section 1. Next we give the derivation of the learning rule for the decoder weights, D.

If one were training the data in batch mode, and had a sequence of observations $(c(t), z(t))$ for $t \in [0, T]$, then the update to the decoder weights at any point in time is:

$$\Delta D(t) = -\kappa a(t) \times \int_0^1 \left(\hat{z}(t+\tau\theta_p) - z(t+\tau\theta_p)\right) P_q(\tau) S d\tau \tag{1}$$

The terms $P_q(\tau)$ and $S = diag(\frac{1}{2i+1})\ i \in \{0, \ldots, q-1\}$ are necessary to convert the error between the predicted and observed state values into a Legendre coefficient representation. The corrected weight after that epoch is thus:

$$D(T) = D(0) + \int_0^{T-\theta_p} \Delta D(t) dt \tag{2}$$

which is effectively a double integral over the time we are making the prediction at, t, and how far into the future we are making a prediction, $\tau\theta_p$. Unfortunately, Eq. (1) is acausal, but this can be rectified by extracting the inner integral, re-writing the integration, and using the Legendre representation:

$$D(T) = D(0) + \int_0^1 \int_{\tau\theta_p}^{T-\tau\theta_p} -\kappa a(t) \times \left(\hat{Z}(t) P_{q_p}(\tau) - z(t+\tau\theta_p)\right) P_q(\tau) S_q^q dt d\tau \tag{3}$$

To figure out what the instantaneous update to the weight matrix would be, we take the derivative of $D(T)$ with respect to T. Using Leibniz' rule we find

$$\frac{d}{dT}[D(T)] = \int_0^1 -\kappa a(T-\tau\theta_p) \times \left(\hat{Z}(T-\tau\theta_p) P_{q_p}(\tau) - z(T-\tau\theta_p+\tau\theta_p)\right) P_q(\tau) S_q^q d\tau \tag{4}$$

We can rewrite the term $z(T-\tau\theta_p+\tau\theta_p)$ such that only the current observation, $z(T)$ is required. $\hat{Z}(T-\tau\theta_p)P_q(\tau)$ 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 Legendre coefficients, $\hat{Z}(t)$. We keep track of these values using two more LDNs, one for the neural population activity, $a(T-\tau\theta_p) = A(T)P_{q_a}(\tau)$, and one for the predicted Legendre coefficients, $\hat{Z}(T-\tau\theta_p) = M_{\hat{Z}}(T)P_{q_{\hat{Z}}}(\tau)$. 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:

$$\frac{d}{dt}\left[D(t)_N^{q_p m}\right] = \int_0^1 -\kappa A(t)_{N,q_a} P(\tau)^{q_a} \left(M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq} P(\tau)^{q_{\hat{Z}}} P(\tau)_{q_p} - z(t)_m\right) P(\tau)^q S_q^q d\tau \tag{5}$$

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. 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

$$\frac{d}{dt}\left[D(t)_N^{qm}\right] = -\kappa A(t)_{N,q_a}\Big(M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq_p}\int_0^1 P(\tau)^{q_a}P(\tau)^{q_{\hat{Z}}}P(\tau)_{q_p}P(\tau)^q S_q^q d\tau \\ -z(t)^m\int_0^1 P(\tau)^{q_a}P(\tau)^q S_q^q d\tau\Big) \tag{6}$$

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 $Q_{q_p}^{q_a q_{\hat{Z}} q} = \int_0^1 P(\tau)^{q_a}P(\tau)^{q_{\hat{Z}}}P(\tau)_{q_p}P^q(\tau)S_q^q d\tau$ and, because the Legendre polynomials scaled by S_q^q are orthonomal, meaning $\int_0^1 P(\tau)^{q_a}P(\tau)^q S_q^q d\tau$ evaluates to a $q_a \times q$ submatrix of the identity matrix, we can re-write the learning rule as:

$$\frac{d}{dt}\left[D(t)_N^{qm}\right] = -\kappa A(t)_{N,q_a}\left(M_{\hat{Z}}(t)_{q_{\hat{Z}}}^{mq_p}Q_{q_p}^{q_a q_{\hat{Z}} q} - z(t)^m I^{q_a q}\right) \tag{7}$$

The implementation of the LLP learning rule is given in Algorithm 1. Pseudocode for using LLP in the Smith predictor framework is available in supplementary material Appendix A.

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

```
1:  procedure LLP-INIT($m, \theta_p, q_p, N, r, \kappa, \theta_c, q_c, \theta_z, q_z, \Delta t$)
2:      $t \leftarrow 0$
3:      $\bar{A}_c, \bar{B}_c \leftarrow$ LDN($\theta_c, q_c, \Delta t$)                ▷ Initialize LDN matrices
4:      $\bar{A}_{\hat{Z}}, \bar{B}_{\hat{Z}} \leftarrow$ LDN($\theta_z, q_z, \Delta t$)
5:      $\bar{A}_a, \bar{B}_a \leftarrow$ LDN($\theta_z, q_z, \Delta t$)
6:      $C_0 \leftarrow 0^{m \times q_c}$                                  ▷ Initialize LDN state vectors
7:      $M_{\hat{Z},0} \leftarrow 0^{m \times q_z}$
8:      $A_0 \leftarrow 0^{N \times q_z}$
9:      $D_0 \leftarrow 0^{N \times m \times q_p}$                             ▷ Initialize Weight Matrix
10:     $a(0), W, B \leftarrow nef\_ensemble(N, r)$         ▷ Randomly initialize neural population.
11:     $Q \leftarrow \int_0^1 P(\tau)^{q_a}P(\tau)^{q_{\hat{z}}}P(\tau)_{q_p}P(\tau)^q S_q^q d\tau$   ▷ Precompute Q tensor
12: end procedure
13: procedure LLP-UPDATE($c(t), z(t)$)
14:     $C_t \leftarrow (I + \bar{A}_c)\, C_{t-1} + \bar{B}_c c(t)$                  ▷ Update LDN states.
15:     $M_{\hat{Z},t} \leftarrow (I + \bar{A}_{\hat{Z}})\, M_{\hat{Z},t-1} + \bar{B}_{\hat{Z}}\hat{Z}_t$
16:     $A_t \leftarrow (I + \bar{A}_a)A_{t-1} + \bar{B}_a a(t)$
17:     $D \leftarrow D - \kappa A_t \times \left(M_{\hat{Z},t}Q - z(t)I\right)$          ▷ Update weight matrix
18:     $a(t) \leftarrow f(WC_t + B)$                          ▷ Predict Legendre coefficients
19:     $\hat{Z}_t \leftarrow Da(t)$
20:     $t \leftarrow t + \Delta t$
21: end procedure
```

The update rule for the matrix D requires $mq_aq_pq_{\hat{Z}}^2 + Nmq_aq_{\hat{Z}}$ floating point operations for each update. Per the parameters in Table 1a, the update requires 110 kFLOP. Executing the update in 1msec would require computing power on the order of 110 MFLOPs, which is well within the realm of commodity hardware. This implies the algorithm could be run in real time on suitable hardware.

3.3 Experimental Setup

The plant to be controlled is a simple one-link pendulum of length $l = 0.1$m and mass $m = 1$kg operating without gravity, simulated using the PyKinSym software package [30]. The pendulum has hard stops at $\pm\pi$ rad. We tested control performance with target trajectories that were pure sinusoidal signals, $x_d(t) = \frac{\pi}{2}\sin(2\pi f t)$ with frequency values $f_{target} \in \{1, 5, 10\}$ Hz, as well as bandwidth-limited white noise, with cut-off frequencies of $f_{max} \in \{1, 5, 10\}$ Hz.

In the experiment we analyze the performance of three algorithms. The first algorithm (*PD*) is a PD controller, with the gains, K_P and K_D, optimized while controlling to a 5 Hz purely sinusoidal target trajectory without observation latency. The second algorithm (*Smith*) is an ideal Smith predictor that forward-simulates the exact dynamics of the simulated pendulum using the last θ_p seconds of command history to predict the state of the plant, compensating for the observation latency. The third algorithm (*LLP*) uses the LLP in a Smith predictor setup, with the prediction readout point always set to the end of the prediction window, $\theta_p = 0.02$ seconds. For the *LLP* and *Smith* controllers we used the same PD gains as the *PD* controller.

The LLP algorithm was implemented using the Nengo software framework [31], and the code for the LLP algorithm is available for use at XXX[1]. The software that produced the results reported in this paper is available at XXX[2].

We assess the performance of the algorithms using the RMS control error, in radians, over the last two seconds of operation, by which time LLP should have learned how to control the plant. For each setting of the actual observation latency, τ_{obs}, we collect 30 observations. Latency was tested at 21 sample points evenly distributed across the time range $[0, 0.1]$ seconds, as well as additional sampling points at $\{0.002, 0.003, 0.004, 0.0125, 0.0175, 0.021, 0.023\}$ seconds, to gather finer detail at the transition from good to poor performance.

To determine the parameters for the LLP algorithm, we fixed the controller gains and used a genetic algorithm to identify the parameters that minimized the total integrated mean squared prediction error over 250 seconds of operations. The value of the parameters used in this experiment are given in Table 1. To determine the sensitivity of the algorithm to the different parameters, we conducted a sensitivity analysis, which identified q_p as the most influential parameter, followed by the parameters of the context LDN for the target and plant predictors, detailed in supplementary material Appendix D.

Parameter	Symbol	Value
Prediction window	θ_p	0.02 sec
Prediction Legendre order	q_p	10
Prediction # of neurons	N	1000
Neuron population radius	r	0.412
Prediction learning rate	κ	0.0014
Target history window	θ_c	0.029 sec
Target Legendre order	q_c	3
Plant history window	θ_z	0.029 sec
Plant Legendre order	q_z	3

(a) LLP Algorithm Parameters

Parameter	Symbol	Value
Proportional gain	K_P	448.83
Derivative gain	K_D	1.06
Simulation duration	T	250 sec
Prediction readout	τ	1
Pendulum length	l	0.1m
Pendulum mass	m	1 kg

(b) Controller and Simulation Parameters

Table 1: Parameter settings for the LLP algorithm, the feedback controller used in the Smith predictor, and the physical simulation. Neuron population radius is a concept used in the NEF.

Further, we have tested LLP in a pure prediction context on more complex datasets. As this work covers LLP in the context of a Smith predictor, and Smith predictors' performance depends on state prediction quality, the results in supplementary material Appendix E support the belief that LLP is applicable to controlling more complex plants.

4 Results

The left graph of Fig. 2 shows how the instantaneous error between the true (undelayed) state of the system and the predicted value of the state, $\|x(t) - \hat{x}(t - \tau_{obs} + \tau\theta_p)\|_2$, changes over the duration of one trial. As the trial continues the algorithm learns a model of the plant and prediction error decreases. The right graph of Fig. 2 shows the reduction in control error, $\|z(t) - x_d(t)\|_2$, which decreases with time and as the predicted state improves.

[1]URL removed for anonymous review

[2]URL removed for anonymous review

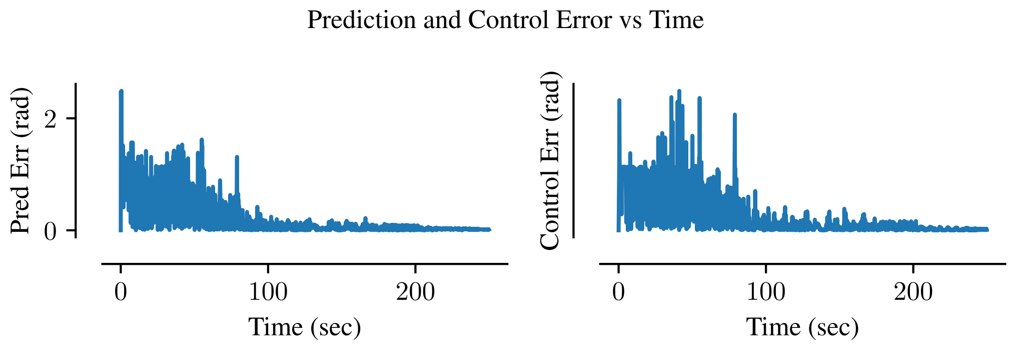

Figure 2: Prediction (left) and control (right) error for one trial with a 5Hz sinusoidal target trajectory, where $\tau_{obs} = 0.02$ seconds. As the system continues to learn the prediction error decreases, as does the control error. The two signals have a Pearson correlation coefficient of $r = 0.62$.

The LLP algorithm was able to compensate for the observation latency, as seen in Fig. 3. These graphs show the RMS control error computed over the last two seconds of a trial as the true delay, τ_{obs} is varied, while the prediction time point is kept constant. The left column shows the performance for a purely sinusoidal target function with frequency f_{target}, while the right column shows the performance for a bandwidth limited Gaussian random noise target with bandwidth limit f_{max}.

The dashed line in Fig. 3 is the performance of the Smith predictor using a perfect model of the system dynamics. While it is able to compensate when $\tau_{obs} = \theta_p$, it is not robust to incorrect estimates of observation latency. LLP demonstrates robustness to error in estimation of τ_{obs}, when $\tau_{obs} < \theta_p$. In all cases LLP has varying degrees of increased error as $\tau_{obs} \to 0$, a possible reason for this is discussed in Appendix B. In the case $\tau_{obs} > \theta_p$, the ideal Smith predictor has the lowest error, followed by LLP, although both algorithms' performance decays sharply when $\tau_{obs} > \theta_p$. The dotted lined is the performance of the PD controller, its performance deteriorates for all $\tau_{obs} > 0$.

To determine the effectiveness of the algorithm, we compared the performance of LLP and the ideal Smith predictor *with* observation latency $\tau_{obs} = \theta_p$ to the PD controller *without* observation latency. For a 1 Hz purely sinusoidal target signal the PD controller and the ideal Smith predictor were statistically significantly better than LLP, using a Bayesian hypothesis test with a 95% confidence level (difference of means 0.08 rad, HDI $[0.04, 0.11]$, respectively). However, for all other tested conditions, there behaviour of LLP was statistically indistinguishable from either the idealized Smith predictor or the PD controller without observation latency. The hypothesis test results are reported in supplementary material Appendix C

5 Discussion

There are three observations to draw from these results. First, in all cases using LLP improves the control error, compared to the uncompensated PD controller, when the latency is close to the prediction window, i.e. $\tau_{obs} = \theta_p$. Further, the comparison of LLP to the PD controller without latency supports the claim that LLP learns to predict the plant state online at a 95% confidence level.

Second, when the observation latency is greater than the prediction window, i.e. $\tau_{obs} > \theta_p$, both the LLP and analytical Smith predictors performance degrades. In this case the standard Smith predictor has better control error, but it is still on the order of 1 radian, which is not suitable for precision operations. Further, the performance of LLP decays in a fashion similar to that of the uncompensated PD controller. This is unsurprising, as in this case there remains observation latency beyond the prediction window.

Finally, when the true latency is less than the prediction point, i.e. $\tau_{obs} < \theta_p$, we see a degree of robustness without changing the prediction time point. The standard Smith predictor is fragile in the face of latency estimation errors, an intrinsic problem for models that anticipate specific observation latencies and that cannot adapt online. The adaptivity of LLP provides some robustness against latency estimation error.

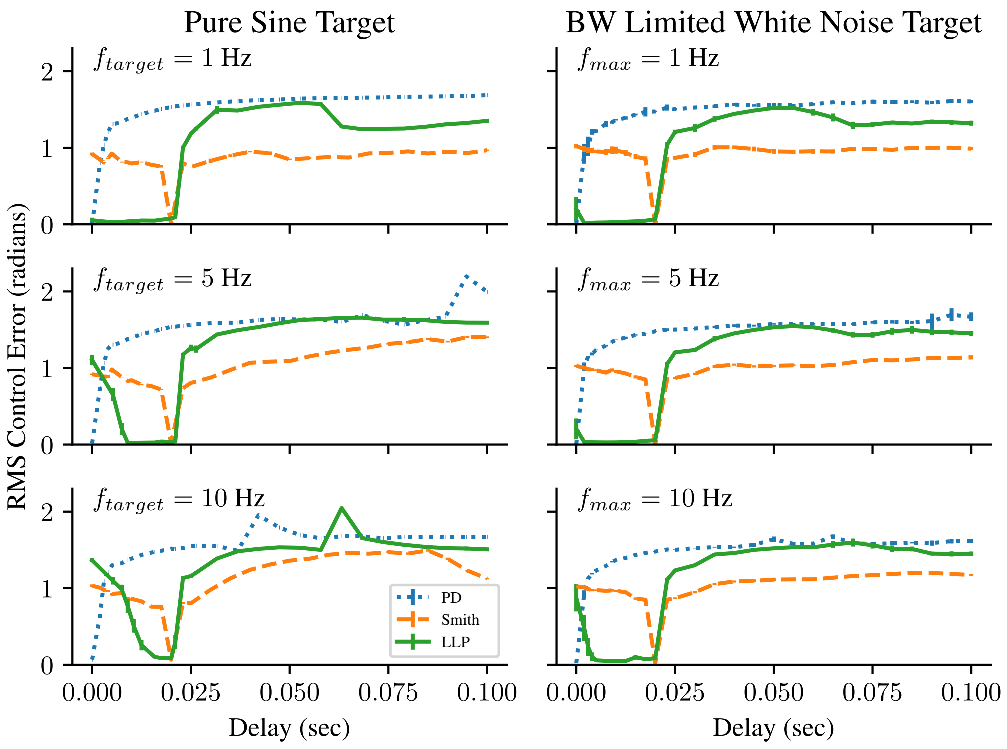

Figure 3: Graph of RMS control error vs observation latency for the last two seconds of operation. Error bars represent a 95% HDI, $N = 30$ trials. The left column figures show system performance with a purely sinusoidal desired trajectory with frequencies 1, 5, and 10 Hz. Right column figures show system performance for a bandwidth limited Gaussian noise desired trajectory. In all cases the control command is generated by predicting $\theta_p = 0.02$ seconds into the future, while the true latency, τ_{obs} is varied from 0 to 0.1 seconds. The dotted line shows the standard PD controller with no deadtime compensation (PD). The dashed line is the Smith predictor using perfect system dynamics model (Smith). The solid line is the Smith predictor with the LLP prediction. Both Smith and LLP compensate when $\tau_{obs} = \theta_p$, but LLP The predictive model is able to improve performance, compared to the uncompensated PD controller, when the delay equals the prediction time point, and shows some degree of target signal-dependent resilience when $\tau_{obs} < \theta_p$.

There are a few options to attempt to mitigate improperly specified latencies: change the PD gains, as one might do to compensate for latency without a predictor; change the controller to be predictive instead of feedback; and change the readout time of the predictor model. The last two options both require identification of the observation latency - predictive controllers need a "now" to start controlling from, and improving the control error through changing the readout time would require setting $\tau\theta_p \approx \tau_{obs}$. While our system does require some calibration of observation latency, it does reduce the need to specify the plant dynamics. In future work we will explore methods for inferring the observation latency from performance.

6 Conclusion

We have demonstrated an online algorithm for learning not just multi-step prediction of continuous states, but prediction over the continuous time. We have used this prediction system in a Smith predictor framework to control a system with observation latency. In the context of control in the presence of latency, our system enabled a Smith predictor to operate without a well-specified model, where previous approaches required precise models, either engineered or learned off-line. Because the algorithm is on-line, it adapts to changes in the mechanism being controlled. Further LLP demonstrated some robustness to uncertainty in the observation latency.

References

[1] O. J. Smith. A controller to overcome dead time. *ISA J.*, 6:28–33, 1959.

[2] 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.

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

[4] 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.

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

[6] 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.

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

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

[9] 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, 1985.

[10] 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, 1985.

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

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

[26] A. Alvarez-Aguirre. Predictor based control strategy for wheeled mobile robots subject to transport delay. *Ch*, 3:33–59, 2010.

[27] M. Bowthorpe, M. Tavakoli, H. Becher, and R. Howe. Smith predictor-based robot control for ultrasound-guided teleoperated beating-heart surgery. *IEEE Journal of biomedical and Health Informatics*, 18(1):157–166, 2013.

[28] S. Behnke, A. Egorova, A. Gloye, R. Rojas, and M. Simon. Predicting away robot control latency. In *Robot Soccer World Cup*, pages 712–719. Springer, 2003.

[29] T. T. Andersen, H. B. Amor, N. A. Andersen, and O. Ravn. Measuring and modelling delays in robot manipulators for temporally precise control using machine learning. In *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*, pages 168–175. IEEE, 2015.

[30] A. Stöckel. Pykinsim - python kinematic chain simulator. `https://github.com/astoeckel/pykinsim`, 2020.

[31] T. Bekolay, J. Bergstra, E. Hunsberger, T. DeWolf, T. C. Stewart, D. Rasmussen, X. Choo, A. Voelker, and C. Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7:48, 2014.