Learning to Predict Events On-line: A Semi-Markov Model for Reinforcement Learning

François Rivest **Richard Kohar** **Najmatoullahi Amadou Boukary**
Department of Mathematics and Computer Science
Royal Military College of Canada
francois.rivest@rmc.ca *richard@math.kohar.ca* *amadoubo@rmc.ca*
francois.rivest@mail.mcgill.ca *kohar@rmc.ca*

Abstract

The ability to represent temporal information and to learn the timing of recurring, instantaneous events is central to the artificial intelligence problem, in particular for embodied agents such as robots. In this paper, we introduce a learning algorithm that provides a new way to learn the value function of a given policy in a partially observable semi-Markov domain, by predicting event timings on-line and in real-time with finite memory. The algorithm is based on a bounded accumulation process similar to drift-diffusion models (DDM) of decision making. It learns the relative event-rate of each input stimulus for each event type, and then uses each observation to accumulate evidence that a specific event is going to occur. We demonstrate how this algorithm could provide an alternative state-space representation in the form of an approximated semi-Markov model for reinforcement learning algorithms such as temporal-difference (TD) learning in a partially observable domain.

1 Introduction

The ability to efficiently use temporal information for predicting when an event is going to occur is central to the development of artificial intelligence [1]. We constantly make predictions about the future using evidences collected in our daily lives to help us make decisions about our schedules, our finances, and even in our simplest tasks. Moreover, event occurrence times are often not purely random. A school bus, a traffic light, the wait time at the restaurant, the time to prepare in the morning, or even the timing of notes in a musical piece — all of these are examples of situations with relatively stable schedules. Therefore, it is possible to learn the timing relationship of these kinds of events and their related time intervals. In reinforcement learning (RL), agents such as robots may find it beneficial to predict when they will receive a reward, like a power source, without having to learn the entire state space representation for a partially observable environment. While this would be useful in robotics, learning to predict event timing has received little attention by the machine learning and AI community [1].

Research have shown that animals can learn events timing quickly, within a few trials [2], while machine learning algorithms, such as recurrent neural networks, can takes thousands of trials or more to learn even the simplest tasks [3]. This fast learning ability by animals has recently been reproduced successfully using time-adaptive drift-diffusion models (TDDM) [4-6]. We introduce a new multivariate version of this algorithm for machine learning. For each specific event type, the algorithm accumulates bits of evidence about when that specific event is about to happen while learning the relative event-rate of each input stimulus. This information is then combined and used to produce, at each time step, an estimate of the remaining time until the next occurrence of that event. An important aspect of this algorithm is that it does not need to be told how much time is remaining; it uses its error when the event occurs to modify its internal event-rate parameters without requiring a precise history of the past inputs. This algorithm learns quickly by building an approximate semi-Markov model with minimal memory requirement. We further demonstrate how this algorithm can be used in conjunction to *temporal-difference* (TD) learning to learn a value function for a given policy in a partially observable environment.

This paper is organized as follows: Section 2 describes the new event prediction algorithm. Section 3 shows the results from training our algorithm to predict music notes. In Section 4, the new algorithm is used in conjunction with TD [7]. TD uses the state-duration estimate provided by our algorithm as a semi-Markov model to learn the value function of a given policy in a partially observable domain. Section 5 provides a brief conclusions and future work.

2 The Algorithm

A fast-learning univariate time-adaptive drift-diffusion model has recently been developed to explain an animal's rapid adaptation to change in the timing of events [4-6]. Similar to this model, our algorithm learns drift rates in the form of stimulus-event rates in a multivariate context. The point in time where the accumulation of drift reaches some arbitrary threshold corresponds to the time the model expects the event. When the event occurs, the algorithm adjusts the evidence-rate that each stimulus provides (*i.e.*, the weight associated to a specific input). Since the threshold is fixed, the difference between the current accumulation level and the threshold, divided by the current observation evidence rate gives an estimate of the remaining time (assuming everything else remains constant). We will state this formally below.

Let $\{\boldsymbol{x}(t) = (x_1(t), \dots, x_N(t))\}$ be an N-dimensional *binary time series* representing the possible observable stimulus or state descriptions. Let $\{\boldsymbol{z}(t) = (z_1(t), \dots, z_M(t))\}$ be an M-dimensional *binary time series* representing the occurrence of events such that a sample $z_j(t) = 1$ only when an event of type j is occurring at time t. Given an input sequence $\{\boldsymbol{x}(t)\}$ of N-dimensional column vectors and an event sequence $\{\boldsymbol{z}(t)\}$ of M-dimensional column vectors, for each input-output pair (i, j), with $1 \leq i \leq N$ and $1 \leq j \leq M$, there is a weight (or evidence rate parameter) $w_{j,i}$ and an accumulator $\phi_{j,i}(t) = \phi_{j,i}(t-1) + w_{j,i}x_i(t)$. For each output stream j, the total accumulation at time t is given by $\Phi_j(t) = \sum_{i=1}^{N} \phi_{j,i}(t)$. The parameters $\boldsymbol{w}_j = [w_{j,1}, \dots, w_{j,N}]$ are trained such that $\Phi_j(t_{j,k}) = 1$ whenever $z(t_{j,k}) = 1$ (i.e. $t_{j,k}$ is the time of the k^{th} event of type j). Thus, the accumulator should reach its threshold (the value 1, in this case) exactly when the event occurs. If $\boldsymbol{x}(t)$ defines a state, then $\boldsymbol{w}_j$ should become the weight vector such that

$$\Phi_j(t_{j,k}) = \sum_{t=t_{j,k-1}}^{t_{j,k}} \boldsymbol{w}_j \cdot \boldsymbol{x}(t) = 1$$

Given that the state remains constant, the remaining time function can be constructed by rearranging the formula. The estimated remaining time is given by

$$\hat{y}_j(t) = \frac{1 - \Phi_j(t)}{\boldsymbol{w}_j \cdot \boldsymbol{x}(t)}$$

Assuming the accumulators in j are reset as soon as an event occurs on stream j, one could estimate the duration a given stimulus has been observed since the last event at time $t_{j,k}$ using the formula $a_{j,i}(t) = \phi_j(t)/w_{j,i}$. Given these durations in $\{\boldsymbol{a_j}(t) = [a_{j,1}(t), \dots, a_{j,N}(t)]\}$, the appropriate weight would be one such that

$$\Phi_j(t_{j,k}) = \boldsymbol{a}_j(t_{j,k}) \cdot \boldsymbol{x}(t_{j,k}) = 1$$

This constraint generates at time $t_{j,k}$ an $(N-1)$-dimensional hyperplane of possible solutions for $\boldsymbol{w}_j$. One can simply move $\boldsymbol{w}_j$ perpendicularly toward that hyperplane using

$$\boldsymbol{w}_j = \boldsymbol{w}_j - \alpha d \boldsymbol{a}_j(t_{j,k})$$

where

$$d = \frac{\boldsymbol{w}_j \cdot \boldsymbol{a}_j(t_{j,k}) - 1}{\boldsymbol{a}_j(t_{j,k}) \cdot \boldsymbol{a}_j^{\boldsymbol{T}}(t_{j,k})}$$

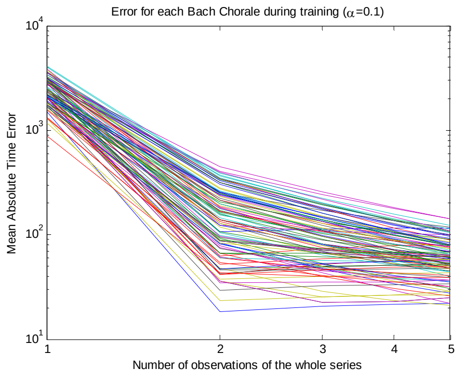
is the distance to the solution hyper plane, and $0 < \alpha < 1$ is a small learning rate. In short, whenever an event occurs, the system is making a correction to minimize its timing error.

This model has $\mathcal{O}(NM)$ parameters and requires $\mathcal{O}(NM)$ memory for the accumulation process. (With double precision, this can go for about a full day when sampling at 1000Hz. For an history of length T, the memory requirements becomes $\mathcal{O}(NM \log T)$ for numerical stability.) This model has a single hyperparameter α.

3 Predicting Note's Onsets in Music

We first demonstrate the algorithm's performance on a musical notes prediction task. The music dataset Bach Chorales [8] is composed of 100 chorales (musical pieces). Each piece of music is constituted of music notes (Pitch MIDI numbers from C4 (midi 60) to G5 (midi 79)) with their start times and durations. The input stream is a multivariate binary time series of 41 input at each time steps. The first one is a bias (always 1), the next 20 inputs correspond to when each notes are played (a value of 1 for each time step where the note should be audible, and 0 otherwise), and the last 20 inputs are the opposite (a value of 1 for each time step a note should not be audible, and 0 otherwise). The event streams to predict are the onsets and offsets of each individual note for a total of 40 outputs. In short, at each time step, the model is producing an estimated remaining time for each note's onset and for each note's offset. But the only signal the algorithm receives to learn is a binary stream containing a value of 1 at the time steps where these events occur, and 0 otherwise. Silence has been added at the both ends and a different model is learned for each piece.

Figure 1(a) shows the algorithm learning exponentially fast. Even with a single presentation of the data, we already see that the remaining time estimate is beginning to converge. A typical output for the fifth presentation of the time series is shown in Figure 1(b); the algorithm's prediction of the remaining time is mimicking the overall pattern of the actual remaining time. The residual error is due to the algorithms inability to distinguish between two different temporal patterns that precedes the same type of event (such as note X's onset).



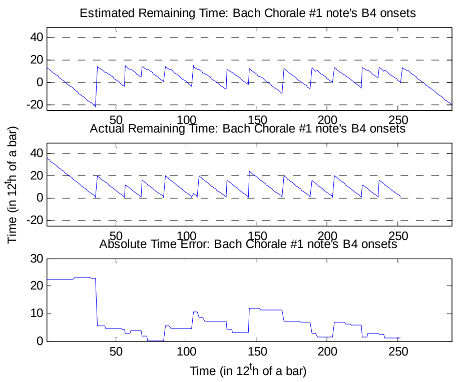


Figure 1: (a) The learning curves for each of the 100 Bach chorales in absolute timing error. (b) Algorithm's estimated remaining time, the actual remaining time, and the absolute timing error for note B4 onsets from Bach chorale #1 on the fifth presentation of the piece.

It is clear, that as it is, a single accumulator per event stream cannot solve every possible problem. Similar to the perceptron, it is possible to find a set of temporal patterns that have no common solution. Our goal is to develop a complete solution, allowing multiple units per event streams and multiple layers of units, similar in spirit to the original development of multilayer neural networks, but using this new type of accumulator unit and timing-based cost function.

4 Integration to Reinforcement Learning

In reinforcement learning, an agent learns to maximize the expected sum of discounted future rewards. For on-line reinforcement learning, solutions are often based on variations of the temporal difference (TD) learning algorithm [7] in which a real-valued function $V: S \to \mathbb{R}$ is learned for each state $s \in S$ and converging toward the expected sum of future rewards ($\sum_{k=1}^{\infty} \gamma^k r_{t+k}$ where $0 \leq \gamma \leq 1$ is a discounting factor) under a given action policy. With a learning rate $0 < \alpha < 1$, the TD learning rule is

$$V(s_t) \longleftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

TD can use the algorithm describe in this paper to learn a partial semi-Markov model of a partially observable environment onto which it can build its value function. For example, a robot in a given sensors state can learn to estimate when each individual sensor state is likely to change. The sensor that is expected to change first may be considered the most likely, and hence, the sensor leading to the most likely next sensors state. By defining $V(s_t)$ as the value of the sensors state s_t at the moment of exiting it for another state, one can estimate $V(t)$ by using the estimated expected remaining time in state s_t reported by our algorithm (at least 1 step). This is given by

$$V(t) = \gamma^{\max\left\{1, \min\limits_j \{\hat{y}_j(t)\}\right\}} V(s_t)$$

4.1 An agent in a new partially-observable environment

Consider a corridor in a square building with rewards at two opposite corners (Figure 2). The basic TD approach is to put some grid (say 15×15) and to assume the agent always knows its exact position (1 square ≡ 1 state). After sufficient learning, the value of each square (state) decreases as one moves away from the reward due to the discounting factor. But corridors could very well look the same. Let's assume that the agent only sees if there are walls one step to the left, in front, or to the right of itself, rather than knowing its exact position. While the grid-based environment itself is still Markovian, it is only partially observable with respect to the agent. It would take a lot of real time for a learning agent to develop a full model of the whole floor-plan without more *a priori* knowledge, because it would need to learn the length (how many steps or states) of each corridor.

Now, we will show that by using our algorithm, an agent can try to learn a partial semi-Markov model to predict when the state will change under a fixed policy. Assuming the agent follows a policy such that as it follows the wall, it will always keep the wall to its right. Clearly, it will face a wall in front of it every 12 steps (including turning), and it will face a reward every 24 steps. Let the binary input vector be defined by those 4 sensors (wall left, wall front, reward front, wall right) and their negations (thus, the input space has 8 dimensions). Then, we can create one forecaster for each of the 8 inputs. Note that from the 2^4 = 16 possible states, only 3 will occur in the current set-up: along a wall facing nothing, facing a reward, or in a corner facing a wall. As shown in Figure 2, within five rounds, the forecaster learns to predict exactly when it will reach the next corner, and exactly when it will reach the next reward (all $\alpha = 0.5$).

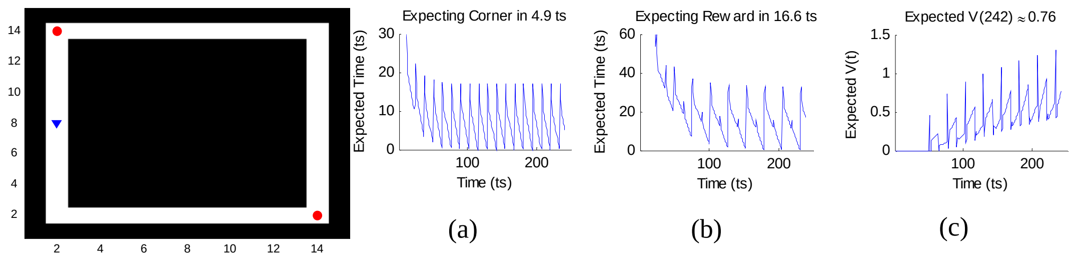


Figure 2: Artificial floor plan for the 15 × 15 world with rewards in opposite corners. The triangle is the agent heading downward. As one can see, after only 242 time steps (or about 5½ round trips), the time forecaster is already predicting the corner 5 time steps ahead (a) and the reward 17 time steps ahead (b) within time step precision from its current position for an expected reward value of .76 (c). Only in the corners are the estimates clearly wrong because the sensory input is different than in the corridor. (Video in supplemental material.)

In a second experiment, we used a more complex cross-shaped corridor environment using the same sensory inputs. The agent has therefore only 5 states: 4-way intersection, 3-way intersection, corner with reward, corner without reward, and corridor. The agent's policy is to move forward while in a corridor, and to select a new corridor randomly when reaching an intersection. This introduces stochasticity in the policy from which the model is learning. Nevertheless, the agent quickly learns the state changes (Figure 3(a)) and reward regularities (Figure 3(b)) allowing it to learn an approximate sum of future reward value for each of these 5 states.

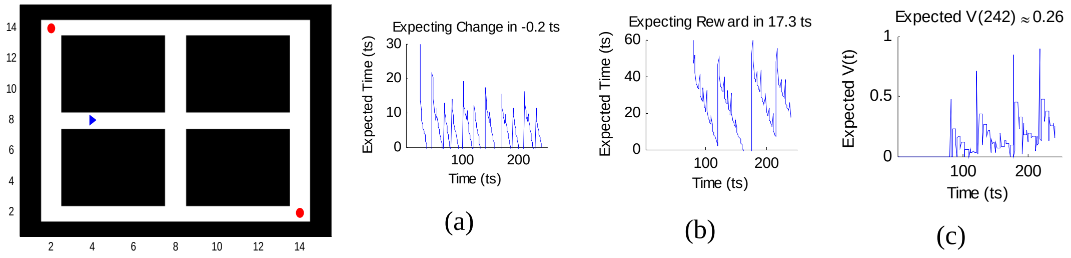


Figure 3: More complex cross-shaped artificial floor plan for the 15 × 15 world with rewards in opposite corners. The triangle is the agent heading rightward. After about 100 time steps, the time forecaster predicts the state change times (which appends each time the agent reaches a corner or an intersection) (a) and reward (b) with regularities. The expected value is plotted in (c) (Video in supplemental material.)

4.2 Discussion

The critical assumption for TD algorithms is that the environment has to be Markovian; that is, the probability of the next state or reward must be fully dependent on, and only on, the last observed state and performed action. But, this assumption is rarely true in reality. Among the alternatives, one could use a partially observable Markov model (POMDP), or predictive state representations (PSR). The POMDP assumes that the world is Markovian, but that the state is not fully observable, and hence must be inferred from current and past observations. The PSR, on the other hand, is attempting to build a set of tests from past observations to predict the observations for the next few time steps. Both approaches are equivalent to some degree [9] and both are based on precise time step history. Here instead, we are developing a partial semi-Markov model allowing a state to have a distribution of exit time (or duration). We also showed one way such model could be used in conjunction with TD to learn a value function for a given policy. There is plenty of work is left to be done along that line.

The algorithm described here is not building a complete semi-Markov model, and still has very limited representational power. For example, it still has a large error when it is in the corner, since the time estimate is made assuming the input stays constant; this explains why when facing the corner, the remaining time is largely over estimated in Figure 2. A similar issue would occurs if that floor-plan was rectangular, the model would keep oscillating its corner prediction, estimating the long corridor as a short one (after the short corridor forecaster updated) and the short corridor as a long one. Another issue is that this simple system is event-order agnostic. As mention in Section 3, we are working on a more complete learning system, but its potential looks promising.

5 Conclusion and future work

In this paper we introduced a novel algorithm that learns to predict when upcoming events will occur based upon the recent time-adaptive drift-diffusion models of animal learning [4-6]. When applied to learn musical data, the algorithm learned event timings within a few trials, orders of magnitudes faster than state-of-the-art online recurrent network [3]. In fact, most of the learning occurs on the first presentation of the time series, as required for real-time on-line reinforcement learning. Furthermore, we showed how this fast learning algorithm (or potentially semi-Markov model) can provide an alternative state-space representation for RL algorithms such as TD in partially observable domains. Here, after only 5½ round trips, the agent had learned when to expect the rewards generating a value function for its semi-Markov representation.

While the algorithm's performance in terms of error and representational power, is still very limited, its ability to learn fast, and its fundamental roots from computational models of animal learning, suggests that it leads to a new class of learning algorithms. In decision making, inputs are not restricted to be binary, thus, the algorithm should be extendable to real-valued inputs. By adding more accumulators per event types, this should also allow fuller semi-Markov representations.

Acknowledgements

This work was supported by an ARP research grant from the Royal Military College of Canada and a CD-ARP research grant from the Canadian Defence Academy to FR. This manuscript also profited from the comments of anonymous reviewers.

References

[1] Maniadakis, M., & Trahanias, P. (2011). Temporal cognition: a key ingredient of intelligent systems. *Frontiers in neurorobotics, 5*, 2. doi: 10.3389/fnbot.2011.00002

[2] Balsam, P. D., Drew, M. R., & Yang, C. (2002). Timing at the start of associative learning. *Learning and Motivation, 33*(1), 141-155.

[3] Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2002). Learning Precise Timing with LSTM Recurrent Networks. *Journal of Machine Learning Research, 3*, 115-143.

[4] Rivest, F., & Bengio, Y. (2011). *Adaptive Drift-Diffusion Process to Learn Time Intervals*. arXiv:1103.2382v1 (1103.2382v1). Cornell University Librairy.

[5] Simen, P., Balci, F., de Souza, L., Cohen, J. D., & Holmes, P. (2011). A model of interval timing by neural integration. *J Neurosci, 31*(25), 9238-9253. doi: 10.1523/JNEUROSCI.3121-10.2011

[6] Luzardo, A., Ludvig, E. A., & Rivest, F. (2013). An adaptive drift-diffusion model of interval timing dynamics. *Behavioural Processes, 95*, 90-99. doi: 10.1016/j.beproc.2013.02.003

[7] Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press.

[8] Bache, K., & Lichman, M. (2013). UCI Machine Learning Repository

[9] Littman, M., Sutton, R. S., & Singh, S. (2002). *Predictive representations of state*. Paper presented at the Neural Information Processing Systems, Cambridge, MA.