A New Timing Error Cost Function for Binary Time Series Prediction

François Rivest and Richard Kohar

***Abstract*—The ability to make predictions is central to the artificial intelligence problem. While machine learning algorithms have difficulty in learning to predict events with hundreds of time step dependencies, animals can learn event timing within tens of trials across a broad spectrum of time scales. This suggests strongly a need for new perspectives on the forecasting problem. This paper focuses on binary time series that can be predicted within some temporal precision. We demonstrate that the sum of squared errors (SSE) calculated at every time step is not appropriate for this problem. Next, we look at the advantages and shortcomings of using a dynamic time-warping (DTW) cost function. Then, we propose the squared timing error (STE) that uses DTW on the event space and applies SSE on the timing error instead of at each time step. We evaluate all three cost functions on different types of timing errors, such as phase-shift, warping, and missing events, on synthetic and real-world binary time series (heartbeats, finance, and music). The results show that STE provides more information about timing error, is differentiable, and can be computed on-line efficiently. Finally, we devise a gradient descent algorithm for STE on a simplified recurrent neural network. We then compare the performance of the STE-based algorithm to SSE and logit-based gradient descent algorithms on the same network architecture. The results on real-world binary time series show that the STE algorithm generally outperforms all the other cost functions considered.**

***Index Terms*—Squared timing error, Time series forecasting, Dynamic time warping, Recurrent neural network**

I. Introduction

THE ability to make predictions is central to the artificial intelligence problem [1]. We make predictions using evidence collected in our daily lives to help us make decisions about our schedules, our finances, and even in our simplest tasks. For example, crossing the street safely requires some form of time estimate for us to cross the street and avoid being struck by a car. The key to solve our prediction problems, even in the simple task of crossing the street, are timing mechanisms. They help us to predict the timing of upcoming events needed to synchronize strategies, decisions, and actions within our environment. Robots are also in need of such timing learning ability [2]. However, turning our continuous stream of observations into useful predictions that can be used to make or adjust our decisions in real time is a notoriously difficult computational problem [1].

Predicting the future can be seen as a time series forecasting problem. In this paradigm, one usually attempts to predict the upcoming observations for one or a few time steps ahead. The vast majority of machine learning algorithms try to minimize a cost function, such as the sum squared error or negative likelihood, using a gradient-based approach. The sum squared error is extended to time series forecasting by summing the squared error at each fixed-size time step over the entire sequence and making a gradient descent over that sum. However, the basic sum squared error cost may not perform well for many tasks since it only considers amplitude error without adequately taking timing error into account [3]. This is particularly important in time series recognition and clustering, where one would rather have a measure that is almost invariant under time shifting or warping [4]. The most common solution to this problem is dynamic time warping. Initially developed in speech recognition [3], dynamic time warping works by finding a time-warping function that once applied, would minimize the sum of squared error (or some other cost function) between two time series. While it has the advantage of being more time-shift and time-warp invariant, this method is computationally expensive ($\mathcal{O}(T^2 f^2)$, where T is the time window length in time units and f is the sampling frequency). Moreover, it does not provide a gradient to optimize the predictions of a forecasting system. Thus, dynamic time warping is more appealing to pattern recognition than to time series forecasting. Much work has been done on similarity measures in time series recognition and knowledge discovery (*e.g.*, [4], [5], [6], [7]), but they usually cannot use the temporal error directly in improving their predictions and often cannot be trained on-line in real-time.

Computationally, the forecasting problem becomes a matter of determining which subsequence of some finite past should be used to make a prediction. Even if we limit ourselves to a single stream of binary inputs, the number of possible subsequences over the past n time steps that need to be observed to make an appropriate prediction for the next time step only is on the powerset of n, which is superexponential. For example, with state of the art artificial neural networks, learning a simple association of a few time steps apart can take thousands of trials [8], and at a hundred time steps apart, it can take millions of trials and success is not guaranteed [9]. Moreover, increasing the sampling frequency escalates the problem's difficulty as much as increasing the interval length itself. Without any prior knowledge about the sequence structure, using artificial neural networks to predict a binary event is likely to be NP-complete [10]. This means that they may need a number of trials that is exponential with respect to the temporal distance, in time steps, between the necessary inputs and the corresponding prediction.

F. Rivest and R. Kohar are with the Department of Mathematics and Computer Science, Royal Military College of Canada, Kingston, ON, K7K 4B4, Canada. E-mail: francois.rivest@{mail.mcgill.ca, rmc.ca}, richard@math.kohar.ca.

F. Rivest is also with the Centre for Neuroscience Studies, Queen's University, Kingston, ON, Canada.

Despite those facts, learning timing in nature seems quite easy. Animals can learn stimulus-reward association in a few tens or hundreds of trials for intervals ranging from hundreds of milliseconds to many minutes [11]. In fact, as long as the ratio between the interstimulus interval (the time interval between the stimulus announcing the reward and the reward itself) and the intertrial interval (the time interval between two conditioning trials) is constant, the required number of trials (or paired observations) to conditioning is constant ($\mathcal{O}(1)$) [11]. Moreover, when a conditioned response appears, precise timing is already learned [12], [13]! Thus, evolution has found ways for animals to learn timing in a constant number of trials, independent of the time scale; this suggests that a different computational approach to the forecasting problem can be taken. Recent results using new learning rules focusing on timing, without minimizing the sum of squared error at every time step, have shown significant improvement in learning speed, approaching animal performance [14], [15], [16].

In this paper, we look at two standard cost functions and develop a new one with this perspective: trying to minimize the timing error for every event (learning *when*) rather than the output error for every time step (learning *what*). Our focus is on cost functions that can be used by on-line real-time learning systems to predict the timing of binary events or event onsets. First, we outline the failures of the sum squared error (SSE) as a cost function for learning to predict event timings, and then list the desirable properties that a good timing error cost function should have for this problem. Next, we evaluate how dynamic time warping (DTW) could be adapted for this purpose, reviewing its advantages and disadvantages. We then propose the squared timing errors (STE) as a new cost function and show how it implements the desired properties. The three cost functions—SSE, DTW, and STE—are then evaluated on a set of synthetic and real-world binary time series (heartbeats, stock prices, and music pieces) by injecting an increasing amount of temporal noise (such as time-shift, time-warp, or missing/extra events) to the binary time series. The results demonstrate that STE reports timing error accurately, more than SSE and DTW. STE seems, therefore, the best of the three cost functions having all the desired properties to learn event timing on-line. Finally, we devise an STE gradient descent algorithm for a simplified recurrent neural network and compare it to SSE and logit gradient descent on learning to predict the above real-world binary time series with the same network architecture. The STE gradient descent algorithm generally outperforms the other two algorithms under all cost function performance measures.

II. Binary Time Series Error

A. Sum Squared Error

The sum squared error (SSE) is one of the most widely used cost functions in machine learning and time series forecasting today. Many algorithms define a model (or forecaster) by minimizing this cost, or a related one, using a gradient descent algorithm or a similar optimization technique. There have been some successful results in training complex neural networks on difficult tasks with long-term relationships using this approach (for example, [9]). Yet, these artificial neural networks were unable to learn to predict even a short time interval in a reasonable number of training trials. For example, learning when a reward will occur under fixed-delay conditioning of a few time steps requires thousands of trials for those networks [8], while animals only require a few dozen trials to learn the same time interval [11]! One could argue that the error function does not cause the difficulty in learning for neural networks, but by the neural network recurrent architecture itself [17]. Nonetheless, we argue that the SSE itself overlooks a critical error component—the timing error.

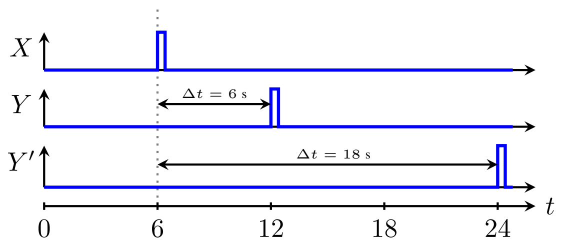

Fig. 1. Example showing that SSE does not provide information about the magnitude of the timing error. The three signals, X, Y and Y', only differ by their timing. While Y is three times closer to X than Y' in terms of timing, Y and Y' have the same SSE with X ($\text{SSE}(X,Y) = \text{SSE}(X,Y') = 2$).

Let $X = (x_1, x_2, \ldots, x_L)$ and $Y = (y_1, y_2, \ldots, y_L)$ be two binary time series with L data points each, such that $x_i, y_i \in \{0, 1\}$, and where the time interval between any two consecutive data points (*e.g.*, x_i and x_{i+1}) is constant. Then the sum of squared errors between the two series is

$$\text{SSE}(X,Y) = \sum_{i=1}^{L} (x_i - y_i)^2. \tag{1}$$

It is easy to show that some information is not directly accessible through the SSE, namely the timing error. For example, in Fig. 1, the timing difference between signals X and Y' is three times larger than between signals X and Y. Yet, both SSEs are exactly the same ($\text{SSE}(X,Y) = \text{SSE}(X,Y') = 2$).

To learn to predict an event coming a fixed number n of time steps after a given input (or stimulus), most forecasters would use a set of $D \geq n$ delay-lines to maintain instant access to a long enough history of recent past observations. These forecasters are limited to predictions that can be defined as a weighted sum or function of these D recent observations only. In the absence of sufficient delay-lines, a recurrent neural network could try to link the events together (the predictive input, the event itself, and its current prediction) through its architecture and the SSE gradient, but this is difficult to achieve [3], [8], [17]. A cost function using the timing error could directly provide the learner with some information about both, the relationships between an event (in X) and its prediction (in Y), and the size of the timing error.

B. Desirable Timing Cost Function Properties

A good timing error cost function should be reasonably:

1) monotonically increasing with phase shift (more phase shift should induce more error);

2) monotonically increasing with warping factor (more time compression or expansion, should induce more error); and
3) monotonically increasing with event rates difference (more missing or extra events, should induce more error).

For a good automatic forecasting system to learn on-line, such as for adaptive control of robots or in reinforcement learning, the following properties would be useful:

4) differentiability (to allow some form of cost minimization through gradient descent); and
5) on-line and real-time computability (to learn from every single event as they occur).

The SSE is clearly not proportional to amount of phase-shift or warping (Properties 1–2), as shown in Fig. 1. It does, however, increase as the number of events between the two sequences differs (Property 3). It can always be computed on-line (Property 5) and it can generate a gradient (Property 4) for a differentiable system that produces an output at every time step. But the gradient of the SSE has little timing error information and is therefore of limited use for our problem [3], [17].

C. Dynamic Time Warping

Dynamic time warping (DTW) is an approach that was first developed in speech recognition [3]. The idea was to eliminate the timing differences between two speech signals by stretching or warping their time axis to obtain the maximum coincidence. This can be computed efficiently using dynamic programming to find the warping path that minimizes the residual distance between the two speech sequences.

For our problem, we consider an observed binary time series X over a given time window of length L and the corresponding prediction binary time series Y over same the time window. We can then create an L-by-L matrix or grid W, where each grid point $p = (i, j)$ corresponds to a pair (x_i, y_j). A warping path $P = (p_1, p_2, \ldots, p_K)$ is a monotonic sequence of grid points traversing the grid such that $p_1 = (1, 1)$ and $p_K = (L, L)$. The dynamic time warping problem is to find the path P that minimizes the cost of matching the two signals X and Y using some cost function $\delta(\cdot, \cdot)$ (such as the SSE) for each point p_k on the path. This mathematical programming problem is

$$\mathrm{DTW}(X, Y) = \min_P \left\{ \sum_{k=1}^{K} \delta(p_k) \right\},$$

which can be solved in $\mathcal{O}(L^2)$. Note that in contrast to the basic SSE, the two signals can be of different lengths.

In short, dynamic time warping attempts to stretch or warp subsections of the signal to minimize SSE between the two time-realigned signals. But, this only gives a measure of the difference in amplitude between a signal and an optimally warped version of a second signal. It does not provide a measure of the timing error or of the amount of warping done. However, this can be solved by using the path length (*i.e.*, the amount of warping done), alone, or in addition to the remaining SSE, to get a combined measure of timing error and signal amplitude error.

1) Dynamic Time Warping and Timing Error: One can adapt the dynamic time warping method for binary streams of instantaneous events to find the realignment that requires the fewest unaligned events and the minimal amount of time shifting or warping. The optimal realignment of events minimizing the number m of missed events and the sum of all $K - L$ necessary time compression or expansion steps is

$$\mathrm{DTW}(X, Y) = \min_P \left\{ \sum_{k=1}^{K} (\delta(p_k) + w(p_{k-1}, p_k)) \right\}, \quad (2)$$

where $w(\cdot, \cdot)$ is the cost of moving in a particular direction in the grid, and $\delta(\cdot, \cdot)$ the cost of not matching an event. Let moving upward or rightward (warping) cost 1 and moving diagonally (no warping) cost 0. Then the matrix $W(i, j)$ of the minimal warping cost from $(1, 1)$ to (i, j) is

$$W(i,j) = \min \begin{cases} W(i-1,j) + 1 + \delta(x_i, y_j), \\ \quad \text{if we compress time;} \\ W(i,j-1) + 1 + \delta(x_i, y_j), \\ \quad \text{if we stretch time;} \\ W(i-1,j-1) + 0 + \delta(x_i, y_j), \\ \quad \text{otherwise,} \end{cases} \quad (3)$$

where

$$\delta(x_i, y_j) = \begin{cases} 0, & \text{if } x_i = y_j \text{ (matching events);} \\ L, & \text{if } x_i \neq y_j \text{ (unmatching events)} \end{cases} \quad (4)$$

with $W(i, 0) = W(0, j) = \infty$ and $W(0, 0) = 0$. Thus $\mathrm{DTW}(X, Y) = W(L, L) = \min_P\{2(K - L) + mL\}$.[1] Eq. (3) is a variation on the Levenshtein [18] edit distance. The high penalty L in Eq. (4) for not matching events forces the algorithm to prefer matching every possible event over any possible amount of warping. An example is shown in Fig. 2.

Everything else constant, shifting any number of events by some small enough amount such that the order of the events remains the same, will induce a proportional amount of error for realigning the events together. The same rule applies to a global shift of the time axis as long as all events are kept within the visible window. Similarly, any warping (without any changes in the number of events by collapsing events together), will induce an amount of error proportional to the total amount of phase-shifting required to realign them. Finally, if event times are kept constant, then the cost is proportional to the number of missing or extra events. Therefore, DTW as defined by Eqs. (2)–(4) satisfies Properties 1–3 for a substantial range of timing errors.

DTW's issues are with Properties 4 and 5. Because of the min operator and its {+1,0} warping costs of Eq. (3), DTW is not differentiable. Recent work has attempted to make it differentiable [19], but the derivative with respect to the prediction series does not include the temporal error, only the amplitude error given appropriate warping and shifting has been applied. Moreover, both signals need to be fully observed

[1]Note that $\mathrm{DTW}(X, Y) = \min_P\{2(K - L) + mL\}$ only if the sampling rate is sufficient to ensure that each event is separated by a non-event sample. Otherwise, consecutive events will join together and each additional event will pay only $+1$ compression/stretching penalty.

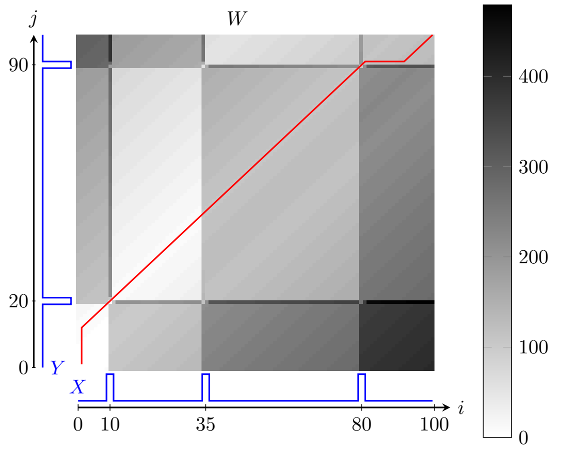

Fig. 2. Example of a dynamic time warping path between two signals X and Y. The first signal X has three events ($i \in \{10, 35, 80\}$). The signal Y corresponds to X right-shifted by 10 time steps without the middle event. The color intensity given by $W(i, j)$ increases as the associated cost of the minimal path from the origin to a given point (i, j) increases. Warping and phase shifting is given by the amount of horizontal and vertical movements (here 2×10). Missing event cost (here 100) occurs when an event in one stream has no corresponding event in the other stream (at $(35, 45)$). Thus, in this example DTW$(X, Y) = 120$.

before DTW can return the final cost, which precludes it from on-line learning as events are occurring. Finally, the complexity of DTW is $\mathcal{O}(L^2) = \mathcal{O}(T^2 f^2)$, which is computationally expensive for any fast sampling real-time system (*i.e.*, when f is large). In short, DTW lacks the desired properties for real-time on-line learning algorithms.

D. Squared Timing Error

In this paper, we propose a new perspective on the forecasting problem: instead of examining the output error at every time step, we suggest focusing on the prediction's timing error for each event. Our objective is to develop a new error cost function that will help predict event timings while maintaining Properties 1–5. Therefore, it must also be more informative of the different types of timing errors such as phase-shift, warping, or missing events (Properties 1–3) than SSE. Moreover, unlike DTW, the cost function should be differentiable (Property 4) and be computable on-line in real-time (Property 5) so that an on-line optimization algorithm can learn from each observed event as it occurs.

Let C_X and C_Y be the number of events in signal X and Y respectively. From now on, we will denote a binary time series $X = (t_{X,1},\ t_{X,2}, \ldots,\ t_{X,C_X})$ as a strictly increasing sequence of times at which the events are occurring in X. That is, $t_{X,i}$ is the time of the i^{th} event in X such that $x_{t_{X,i}} = 1$.

1) Minimum Squared Timing error: We propose that the cost function of two binary signals X and Y be defined as the sum of the squared timing errors for pairs of event in X to event in Y that minimizes that sum while pairing every event; we will call this dynamic squared timing error (DSTE). This is similar to DTW, but instead of the path being along time steps, it is along events, and the cost of the path is determined by the squared timing difference of the matching events. We can define a $(C_X + 1)$ by $(C_Y + 1)$ matrix W_{STE} where each entry

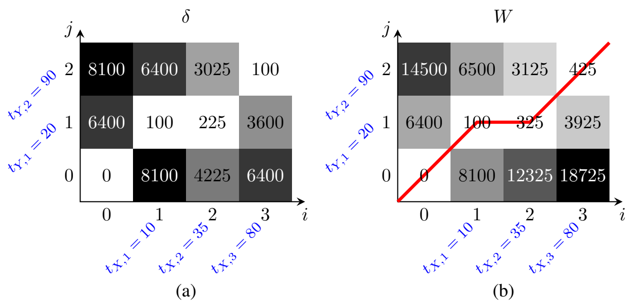

Fig. 3. Example of DSTE computations for the same signals X and Y from Fig. 2. In (a), it shows the individual costs $\delta_{\text{STE}}(\cdot)$ of each possible pairing of an event in X to an event in Y. In (b), it shows the cumulative minimal cost W_{STE} for the path from $(0, 0)$ to (C_X, C_Y). There is a missing event in Y, so DSTE pairs the first and second events of X to the first event of Y, while it pairs the last event of each signal together. The optimal path is $P = \{(0, 0), (1, 1), (2, 1), (3, 2)\}$ yielding DSTE$(X, Y) = 425$.

$p = (i, j)$ corresponds to a pair $(t_{X,i}, t_{Y,j})$. A monotonic path $P = (p_0, p_1, \ldots, p_K)$ is a sequence of grid points traversing the grid such that $p_0 = (0, 0)$ and $p_K = (C_X, C_Y)$ as shown in Fig. 3. Then, the problem is to find the path that minimizes the matching cost which is defined to be the squared timing error. This is given by

$$\text{DSTE}(X, Y) = \min_P \left\{ \sum_{k=1}^{K} \delta_{\text{STE}}(p_k) \right\}, \tag{5}$$

where

$$\delta_{\text{STE}}(i, j) = (t_{X,i} - t_{Y,j})^2 \tag{6}$$

is the cost of matching two events together, *i.e.*, traversing a particular grid point. In DSTE, the path can only go up ($\uparrow$), diagonally ($\nearrow$), or right ($\rightarrow$) by one grid point at a time, guaranteeing that every event is matched to at least one event in the other stream. Then the matrix $W_{\text{STE}}(i, j)$ of the minimal matching cost from $(0, 0)$ to (i, j) is given by

$$W_{\text{STE}}(i, j) = \min \left\{ \begin{array}{c} W_{\text{STE}}(i-1, j), \\ W_{\text{STE}}(i, j-1), \\ W_{\text{STE}}(i-1, j-1) \end{array} \right\} + \delta_{\text{STE}}(i, j) \tag{7}$$

where

$$\delta_{\text{STE}}(i, 0) = \max \left\{ (t_{X,i} - t_1)^2, (t_{X,i} - t_L)^2 \right\}, \tag{8}$$

$$\delta_{\text{STE}}(0, j) = \max \left\{ (t_1 - t_{Y,j})^2, (t_L - t_{Y,j})^2 \right\}, \tag{9}$$

t_1 and t_L are the times of the first and last time steps respectively, $W_{\text{STE}}(0, 0) = 0$, and DSTE$(X, Y) = W_{\text{STE}}(C_X, C_Y)$. This 0-based row and column ensures that if one signal has no event, then the events in the other signal are each binded to the furthest boundary to maintain monotonicity. If one event was to appear in the other signal at the furthest end, then it would be the new match, decreasing the error.

The difference with SSE is trivial: DSTE works on $t_{X,i}$ while SSE works on x_i. There are also two significant differences with DTW. First, the amount of shift needed to realign any event (the timing error) is squared in DSTE rather than linear. Second, extra events cost no more than the squared time distance to their nearest match in the other stream, as opposed to having a high fixed cost, as we defined in Eq. (4) for DTW. This implies that there could be a DSTE cost for extra or missing events, even if two streams have the same number of events (*i.e.*, when the path is not purely diagonal).

Clearly, DSTE, as defined by Eqs. (5)–(9), increases monotonically as events from two identical signals begin to differ by some local or global time-shift or time-warping (Properties 1 and 2). However, this monotonicity has a limited range. As soon as events from one stream are approaching different events from other streams, the total cost may begin to oscillate as events get linked differently. Therefore, one cannot prove Properties 1 and 2 for the general case where the order of all events (in the mix of both signals) start changing under time shift or warping. We argue in Section III that this range is acceptable for on-line learning. It is also easy to show that adding events to the signal that has the highest number of events (or similarly removing events from the signal that has the least) would increase the total cost (Property 3) since every event must be paired at least once.

Finally, since DSTE cost depends solely on $\delta(\cdot)_{\text{STE}}$ and has no warping penalties (unlike DTW's $+1$ in Eq. (3)), the result of its evaluation is differentiable with respect to the temporal error (Property 4) provided that the function predicting the events' time of occurrences is differentiable too. Although it requires the signals to be fully observed, DSTE's complexity is only $\mathcal{O}(C_X C_Y)$, which is much lower than DTW with $\mathcal{O}(T^2 f^2)$, and possibly SSE with $\mathcal{O}(Tf)$ provided that $C_X C_Y \ll Tf$.

2) Nearest Match Squared Timing Error: To further improve the on-line usability (Property 5) of DSTE, we propose an approximation to it when considering real-time learning. Simply take each event in one stream, find its nearest match (the event with the least amount of squared timing difference) in the other stream, and cumulate their squared distances. Proceed similarly from the other stream. This local STE (or LSTE) can thus be defined as

$$\text{LSTE}(X,Y) = \frac{1}{2}\sum_{i=1}^{C_X} \min_{j=1,..,C_Y} \{(t_{X,i}-t_{Y,j})^2\} + \frac{1}{2}\sum_{j=1}^{C_Y} \min_{i=1,..,C_X} \{(t_{Y,j}-t_{X,i})^2\}, \quad (10)$$

with

$$\text{LSTE}(X,\emptyset) = \frac{1}{2}\sum_{i=1}^{C_X} \max\left\{(t_{X,i}-t_1)^2, (t_{X,i}-t_L)^2\right\} \quad (11)$$

if one of the signals is empty (as in Eqs. (8)–(9)).

An example of LSTE (as defined by Eqs. (10)–(11)) can be calculated from Fig. 3. The minimum of every row in δ_{STE} is 100; the minimum of every column is 100, 225, and 100. So $\text{LSTE} = \frac{1}{2}(100+100) + \frac{1}{2}(100+225+100) = 312.5$.

We will show that LSTE is a good approximation of DSTE by showing that $\frac{1}{2}\text{DSTE} \leq \text{LSTE} \leq \text{DSTE}$. DSTE has at least $\max\{C_X, C_Y\}$ costs to be added while LSTE has $C_X + C_Y$ terms. Without loss of generality, let $C_X \geq C_Y$. Then the C_X costs in the first summation of LSTE in Eq. (10) are the smallest values of each column in the δ_{STE} matrix. But DSTE has to go through every column, therefore, $\sum_{i=1}^{C_X} \min_{j=1,..,C_Y} \{(t_{X,i}-t_{Y,j})^2\} \leq \text{DSTE}$. Similarly, the remaining C_Y ($\leq C_X$) terms in the second LSTE summation are the smallest of each row, thus $\sum_{j=1}^{C_Y} \min_{i=1,..,C_X} \{(t_{Y,j}-t_{X,i})^2\} \leq \text{DSTE}$. But each LSTE summation is multiplied by $1/2$, proving that $\text{LSTE}(X,Y) \leq \text{DSTE}(X,Y)$. Now, LSTE has exactly $C_X + C_Y$ terms, which must form a connected path on the grid points in δ_{STE} from $(1,1)$ to (C_X, C_Y) because of the monotonicity of time indices. But DSTE is the shortest such path. Hence, it must be that $\frac{1}{2}\text{DSTE}(X,Y) \leq \text{LSTE}(X,Y)$. Therefore, $\frac{1}{2}\text{DSTE}(X,Y) \leq \text{LSTE}(X,Y) \leq \text{DSTE}$. The experiments provided in Section III will further demonstrate that LSTE is a good approximation of DSTE respecting Properties 1–3 for a similar range of timing differences.

Finally, the LSTE solves the DSTE on-line issue (Property 5). Indeed, it is built such that it computes the squared timing error as if each event was supposed to be aligned with its closest match in the other signal. As opposed to DSTE based on the global optimal matching, LSTE does not require a window larger than two events from each stream; hence, the memory requirement is independent of the number of events.

III. Experiments and Results

In Section III-A and III-B, we first compare the performance of the cost functions against Properties 1–3 to find which one is best at conveying timing error. To do this, let us assume that X is the target signal to learn, and Y is the prediction of a learning algorithm. To evaluate the monotonicity of the cost functions, different signals Y were artificially generated from target signal X using increasing level of a particular type of timing error. We evaluated five different types of noise: global phase shift, event-wise phase shift (temporal jerk), global (symmetric) time warping, local (asymmetric) time warping (or tempo variability), and missing (or failure to observe) events. In Section III-A, Xs are synthetic binary time series, while in Section III-B, Xs are coming from three sets of real-world binary time series. Finally, in Section III-C, we demonstrate Properties 4–5 by first developing an LSTE gradient descent algorithm for a simplified recurrent neural network, and then, by comparing its performance to SSE and logit gradient descent when learning on-line real-world binary time series using the same network architecture.

A. Experiments on Synthetic Data

In this first experiment, we evaluate the cost function properties for signals X generated from an approximated Poisson process. Synthetic binary time series X were generated as bit strings of fixed length, where 1 denotes the presence of an event, and 0 denotes its absence. For each experiment, $N = 100$ signals were generated using an almost geometric distribution for the interevent intervals. That is, at every time step a Bernoulli pseudo-random number generator produces an event with the probability λ, where λ represents the event rate in the signal. The experiments were replicated for three different values of $\lambda \in \{0.02, 0.10, 0.20\}$. In any signal ($X$ or Y), when two consecutive time steps contain an event in the generated signal, the second event is deleted to guarantee an event-free time step between each event. Similarly, the first and last time steps are always 0. Each signal evaluated is $L = 1000$ time steps long. These N samples represent arbitrary signals that could be partly predicted only given appropriate inputs. In these experiments, we are only interested in evaluating how each cost function behaves with respect to the size of the timing error produces by a hypothetical learning algorithm. To do that, we artificially inserted specific types and amount of temporal noise in a copy Y of a target signal X, where Y represents the imperfect prediction of such learning algorithm.

1) Global Phase Shift: To evaluate how each cost function behaves with respect to global phase shift error, we generated N random signals of length $2L$ as described above. Let $S = (0, s_2, s_3, \dots, s_{2L-1}, 0)$ be one of these randomly generated signals. Then let the signal X be defined as the first half of S such that $X = (0, s_2, \dots, s_{L-1}, 0)$. Then for a given shift of size $\tau \in \{0, 1, \dots, 200\}$ time steps, $Y_\tau = (0, s_{\tau+2}, s_{\tau+3}, \dots, s_{L+\tau-1}, 0)$. Constructing signals X and Y_τ this way ensures that the number or events within the visible signal window remains almost constant as the size of the phase-shift increases. Thus, time shifting remains the main variable in this experiment. Global phase shifting is symmetric for all cost functions, so a single direction is sufficient.

2) Local Phase Shift: To evaluate how each cost function behaves with respect to event-wise phase shift (or temporal noise), we generated N random signals of length L as before, to which we added an increasing amount of Gaussian noise to the timing of each event. Given a random signal S, the signal X is generated by zeroing the first and last $\max\{3\sigma\}$ time steps of S so that the noise is unlikely to throw an event outside the signal window. Then, for a given noise variance $\sigma \in \{0, 1, \dots, 20\}$, the signal Y_σ is created as follows: First, extract the timing $t_{X,i}$ of each event from X. Then, the new timings for those events is given by $t_{Y_\sigma,i} = t_{X,i} + \xi_{\sigma,i}$ where $\xi_{\sigma,i} \sim N(0, \sigma^2)$. Any event $t_{Y_\sigma,i} \notin (1, L)$ is discarded from Y_σ. Thus, the new signal Y_σ is made of 1s for the time steps $t_{Y_\sigma,i}$ and 0 everywhere else. Note that some event can collide, affecting slightly the number of events between signals, but the amount of event-wise phase shift remains the main variable.

3) Global (Symmetric) Time Warping: To evaluate how each cost function behaves with respect to time expansion (or compression), we generated N random signals of length L as before. Given a random signal, S, the signal X is generated by zeroing the first and last quarter of S so that the signal can be expanded from the center by up to a factor of 2 without throwing events outside the signal window. The signal is expanded about the center point ($c = L/2$) by an expansion factor $\omega \in \{1.00, 1.05, \dots, 2.00\}$. In this range, every event from the original signal X remains in the time window and thus, ensures warping is the main variable in this experiment. Thus, for an event $t_{X,i}$ and an expansion factor ω, the new timing of the event is given by $t_{Y_\omega,i} = (t_{X,i} - L/2) \cdot \omega + L/2$. Note that compression would simply be the inverse (taking a complete signal S and compressing it around c).

4) Local (Asymmetric) Time Warping: Another way time warping can be applied is by having some compression, and some expansion without significantly changing the overall signal length. Again, we generated N samples of length L. However, now, for each signal $X = S$, we used the center point c as an anchor point that moves to the left to compress the left part of the signal while expanding the right part. For an event $t_{X,i}$ on the left side of c and a compression ratio $\nu \in \{0.00, 0.05, \dots, 0.50\}$, the new timing of the event is given by $t_{Y_\nu,i} = t_{X,i} \cdot (1 - \nu)$. For an event on the right side, it is given by $t_{Y_\nu,i} = L/2 \cdot (1 - \nu) + (t_{X,i} - L/2) \cdot (1 + \nu)$. Local warping, as opposed to global warping, does not affect the signal overall event rate (but it may fuse few events as in the local time shift experiment).

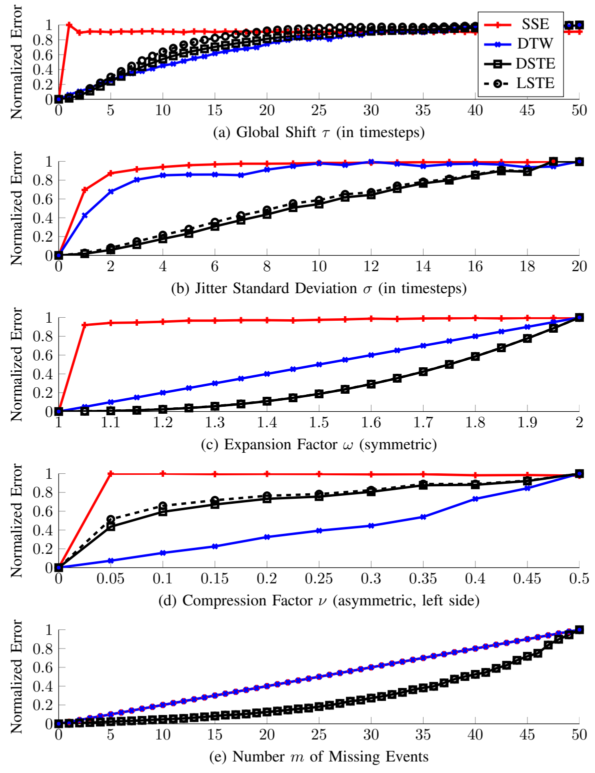

Fig. 4. Results for all 5 experiments on 100 synthetic signals with event rate $\lambda = 0.1$. Each curve is the average cost divided by the average cost maximum value. (a) Error with respect to the amount τ of global shift added to the original signals. (b) Error with respect to the amount of local jitter σ added to the original signals. (c) Error with respect to the amount of symmetric expansion ω added to the original signal. (d) Error with respect to the amount of asymmetric compression (warping) ν added to the original signals. (e) Error with respect to the number m of events removed from the original signals. While SSE conveys very little information about the amount of error, DTW and STEs are quite monotonic and increasing on average. Moreover, LSTE is an excellent approximation of DSTE.

5) Missing (or Extra) Events: Finally, to evaluate how the cost function behaves with respect to missing (or extra) events, we generated N samples of length L. The signal $X = S$ is always the initial randomly generated signal. Signal Y_m is created by removing one, uniformly randomly selected event, from Y_{m-1} such that $Y_0 = X$ and $Y_m = Y_{m-1} \setminus \{t_{Y_{m-1},rnd}\}$. Thus, the difference between X and Y_m is always the absence of some events in Y_m with respect to X. If there are no more events in Y_m, then $Y_{m+1} = Y_m$. Again, by symmetry, the cost reports differences in the number of events, not only adding or only removing events.

6) Results: Results for all five experiments with $\lambda = 0.10$ are shown in Fig. 4; the results for $\lambda = 0.02$ and $\lambda = 0.20$ are similar. There are three main elements visible in these figures. First, as mentioned before, the SSE does not convey any information about the magnitude of the timing error. The SSE error is either 0, or close to 100% of its maximal value, except when removing events (Fig. 4(e)). Second, all cost functions except SSE are monotonically increasing on average as we increase the temporal error magnitude (noise parameter level). While DTW can support more global shift than DSTE and LSTE before reaching a plateau (Fig. 4(a)), DSTE and LSTE are better than DTW at reporting local shift (Fig. 4(b)). Third, the results show that the LSTE, which can be computed on-line using only the last and next event of both streams, is a good approximation of the optimal DSTE. Finally, it is also important to note the memory and time requirements of DTW which are in the order of $\mathcal{O}(L^2)$. In practice, DTW took approximately 60 times longer to compute than the three other cost functions in this experiment and that factor should increase with L. Overall, LSTE only requires a constant amount of memory and provides all the required properties for on-line real-time learning systems.

B. Experiments on Real Data

Although DSTE and LSTE cost functions seem to behave well with Poisson distributed events, in many situations of interest, the events are unlikely to follow this distribution. For example, notes onsets in a musical piece are usually not Poisson distributed. A note is more likely to fall on beats, half-beats, or quarter-beats, than anywhere else. Even timing error could be due to tempo variation, missing a note, or a slight deviation in its timing. Therefore, in this section, we run the same set of experiments as in section III-A, but on real-world time series from normal heartbeats, financial stock prices, and musical pieces, to ensure that DSTE and LSTE are still well-behaved. In general, for the global shift experiment, the original signals were expanded by $\max(\tau)$ zeros on each side to allow phase shift τ without losing any events. For local shift, they were expanded with $\max(3\sigma)$ zeros on each side for the same reason. For the global (symmetric warping), the signals were expanded with $L/2$ zeros on each side to allow expansion without losing events. In the missing events experiment, the maximum number of potentially missing events was related to the database maximum event rate. The maximum values for each of the five experiment parameters are in Table I.

TABLE I
MAXIMUM PARAMETERS VALUES FOR EACH DATASET.

Experiment's Parameter	Synthetic	Heart	Finance	Music
(a) Global Shift (τ)	200	20	50	50
(b) Standard Deviation (σ)	20	20	20	20
(c) Expansion Factor (ω)	2.00	2.00	2.00	2.00
(d) Compression Ratio (ν)	0.50	0.50	0.50	0.50
(e) Missing Events (m)	50	100	30	30

1) Heart Arrhythmia: Detection of heartbeat irregularity is essential in detecting arrhythmia. To do that, a good algorithm should be able to detect missing or extra beats as well as

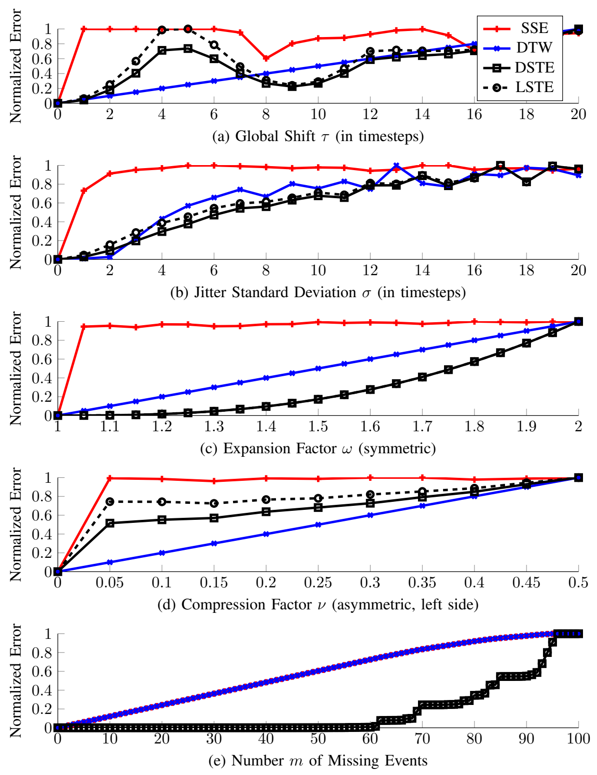

Fig. 5. Results for all five experiments on ten normal ECGs. Same format as in Fig. 4. Panel (a) shows that DSTE and LSTE are mostly cycle-invariant on this periodic dataset. Panels (c)-(e) results are similar to Fig. 4

time-shifted beats [20]. Therefore in this section, we run the same five experiments as in Section III-A on ECG digital records selected from the MIT-BIH Normal Sinus Rhythm Database (nsrdb) located on the Physionet server [21]. Each record was downloaded as a MAT file. The length of each ECG record is 60 seconds with 7680 samples, and the first channel of the file was imported into Matlab. Using Matlab's findpeaks function to find peaks with at least 0.5 mV or higher, we identify the R peaks as heartbeats. We excluded any record where the R peak did not cross the minimum height to ensure that the findpeaks function could perform accurately. A total of ten records[2] out of 18 were analyzed. The normal database had been downsampled 10-fold to 768 because of the computational time and memory requirements for DTW.

The results for all five experiments on the heartbeat data are shown in Fig. 5. There are three main elements visible in each of the subfigures: First, SSE normalized error is either 0 or close to 1, as we saw in Section III-A, except when removing events (Fig. 5(e)); Second, the results show that LSTE is a good approximation of the optimal DSTE; Lastly, the striking difference between the results on the synthetic and heartbeat data is the monotonicity, which is not always present in the heartbeat results.

In the global shift experiment, DTW is accurately reporting the amount of global shift (Fig. 5(a)). But, since heartbeats are almost periodic, DSTE and LSTE are only able to report phase

[2]The selected records were 16265m, 16272m, 16273m, 16420m, 16483m, 16539m, 16786m, 17052m, 17453m, and 18184m.

shift from within the current cycle. This is very reasonable for on-line periodic events. When introducing event-wise jitters, DSTE and LSTE appear more monotonic on average than DTW (Fig. 5(b)).

To better evaluate the monotonicity of each cost function we computed a simple monotonicity index for each signal X of each experiment by summing every decrease along the experiment parameter (τ, σ, ω, or ν), and dividing it by the averaged curve normalization factor (as used in Figs. 4–6). A negative value indicates that the cost function decreases when temporal noise is injected. An index near zero indicates that the cost function is almost monotonically increasing as temporal noise increases. As shown in Table II, DSTE outperforms DTW in each type of temporal noise (lower negative magnitude). Overall, LSTE provides the best set of properties for on-line real-time learning systems, particularly with its ability to be invariant to full cycle phase shift.

TABLE II
MONOTONICITY INDEX (MORE NEGATIVE IS WORST)

			Dataset			
	Experiment	Cost	Synthetic	Heart	Finance	Music
(a)	Global Shift	DTW	-0.9±0.5	+0.0±0.0	+0.0±0.0	+0.0±0.0
		DSTE	-0.2±0.1	-1.2±0.2	-0.0±0.0	-0.0±0.0
(b)	Local Shift	DTW	-2.4±0.6	-1.6±0.4	-4.4±1.2	-7.0±2.1
		DSTE	-0.8±0.3	-1.1±0.2	-1.5±0.7	-2.0±0.6
(c)	Symmetric Warping	DTW	+0.0±0.0	+0.0±0.0	+0.0±0.0	-0.0±0.2
		DSTE	+0.0±0.0	+0.0±0.0	+0.0±0.0	+0.0±0.0
(d)	Asymmetric Warping	DTW	-0.3±0.2	+0.0±0.0	-0.5±0.7	-0.1±0.2
		DSTE	-0.3±0.2	-0.0±0.1	-0.2±0.1	-0.0±0.0

2) Finance: In finance, predicting when a stock price will cross some threshold is often desirable. Closing prices of all NASDAQ-100 companies were selected, and the historical data from 1 April 2013 to 31 March 2014 (253 records per company) were downloaded in CSV format from [22]. For each stock, we created a binary time series using a 14-day moving average of closing price μ and threshold levels of $\mu + 0.7\sigma$, where σ is the moving 14-day sample standard deviation. An event, $x_t = 1$, was generated at every time step t where the stock price was crossing the threshold from below; otherwise, $x_t = 0$.

Results are showed in Fig. 6. Again, SSE performance is almost constant independently of the amount of temporal noise injected, except on the missing event experiments (Fig. 6). Moreover, the DTW performance seems less monotonic than DSTE and LSTE as temporal jitter is added (Fig. 6(b)) and confirmed by Table II, Panel (b). Overall, LSTE and DSTE remain the most well-behaved cost functions.

3) Music: The experiments were performed on 100 Bach Chorales from the UCI Machine Learning Repository [23]. Music is a domain where events (notes) are highly non-Poisson and in which timing is important. The LISP data was reformatted for Matlab. Each choral consists of approximately eight bars using notes from C4 to G5. Notes onsets were extracted and their time indices were computed assuming sampling at 1/16 of a beat so that all pieces were centered in a 500 time step long signal. The error cost between two versions of a piece (with or without timing noise) was considered to be the sum of the error cost for each note signal. In this

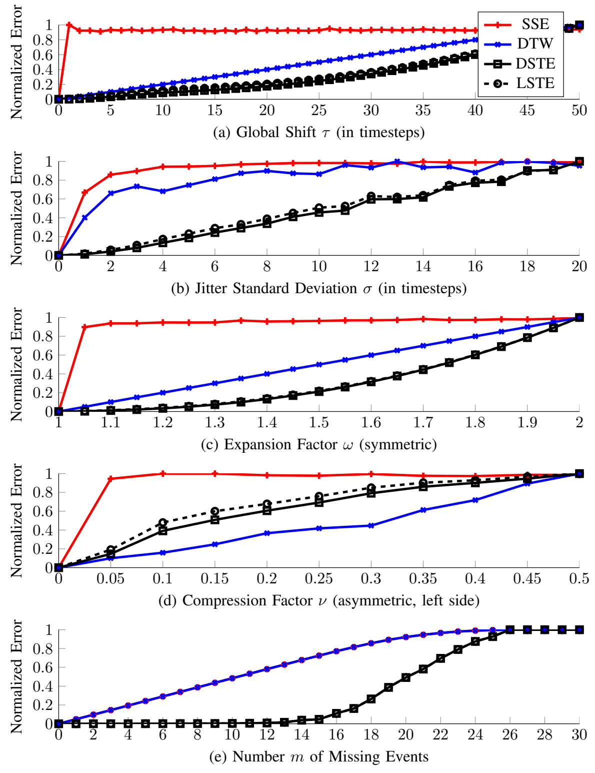

Fig. 6. Results for all five experiments on all NASDAQ-100 stocks from 1 April 2013 to 31 March 2014. Same format as in Fig. 4. DTW shows some issues on local shift (jitter), in panel (b), as it does in Fig. 5.

experiment, all cost functions are well-behaved on average, except SSE on global shift and DTW on jitter. The results are very similar to finance (Fig. 6) except that SSE on global shift shows small drops on each beat. Analysis of individual pieces also highlighted high non-monotonicity for DTW on jitter (see Table II, last column, panel (b)).

4) Summary: In short, SSE does not report the amount of timing error (Properties 1–2), while DTW does not always increase monotonically as the amount of temporal error increases (Property 1). Furthermore, DTW is computationally expensive, cannot be used to learn from temporal error on-line (Property 5), and is not differentiable (Property 4). In contrast, we showed that DSTE, and its on-line approximation, LSTE, are much better at reporting the amount of temporal noise. In the next section, we will show how to differentiate LSTE and show it can outperform SSE in on-line learning (the same can be applied to DSTE in batch learning).

C. Learning on Real Data

In order to compare LSTE's true potential as an on-line prediction learning cost function, we compare it to SSE (and its preferred variation for binary values, the logit loss-function) using a simplified recurrent neural network architecture for which we could derive a gradient for LSTE.

1) LSTE gradient descent: In this section, let $X = (t_{X,1},\, t_{X,2}, \ldots,\, t_{X,C_X})$ be the target binary time series and Y be the prediction time series represented as strictly increasing sequences of times at which the events are occurring. Let

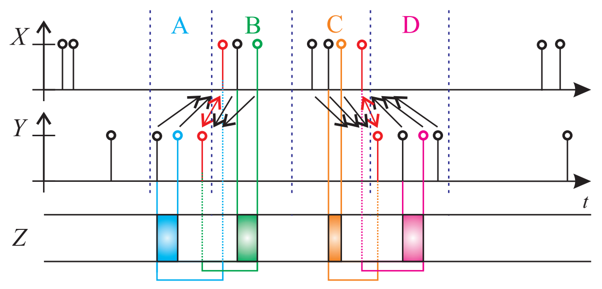

Fig. 7. The LSTE events can be categorized into four groups: A, B, C, D. An event can be observed (X) or predicted (Y), and is mapped to its closest neighbor (in red) in the other stream, which can either come before or after. The rectangles in Z represent the window of input signals (in Z) observed for a given event (in X) or prediction (in Y).

$Z = (\vec{z_1}, \vec{z_2}, \ldots, \vec{z_L})$ with $\vec{z_t} = (z_{1,t}, \ldots, z_{N,t})$ be the exogenous multivariate (N-dimensional) input time series which the model can observe to make event predictions.

Inspired by the time-adaptive drift-diffusion model of animal interval timing [14], [15], [16], we define a simple recurrent neural network for which we can define an on-line gradient for LSTE or apply a standard on-line SSE gradient descent algorithm.

Let $\phi_{n,t} = \sum_{\tau=t_{k-1}+1}^{t} w_n z_{n,\tau}$ be the nth evidence accumulator for the upcoming event, where $t \in [t_{k-1}, t_k]$, and t_{k-1} and t_k are the last and next observed event from any stream ($t_{k-1} < t_k \in X \cup Y$, as defined by event time sets) respectively. Let us say that an event is predicted whenever $\Phi_t = \sum_{n=1}^{N} \phi_{n,t}$ reaches some positive threshold (say 1, without loss of generality). If w_n is exactly the inverse of the accumulation of stimulus z_i on the time interval $[t_{k-1}+1, t_k]$, and both are external events ($t_{k-1}, t_k \in Y$), then Φ_t should reach 1 and predict an event at exactly the right time step. One may argue that if there is no input just before the event, then the prediction will be early, but this can be solved by adding a bias input $z_{0,t} = 1$ for all t. Note that this is not a unique solution. In fact, let $a_{n,t} = \phi_{n,t}/w_n$ and $\vec{a}_t = (a_{1,t}, \ldots, a_{N,t})$, then any $\vec{w} = (w_1, \ldots, w_N)$ such that $\vec{w}^\top \vec{a}_t = 1$ is a valid solution. Let the prediction system $y_t = f(\vec{z}_t, \vec{z}_{t-1}, \ldots; \vec{w})$ be such a system. We will show that whenever an event occurs, we can compute an approximate gradient of LSTE for the current pairing at the point of observation with respect to the weights $\vec{w}$ such that minimizing LSTE can be turned into a standard gradient descent algorithm for this system.

As shown in Fig. 7, there are four possible cases when correcting an event, each leading to a slightly different gradient, depending on the temporal window of input over Z that generated the accumulation present in ϕ_{n,t_k} at the time t_k of the event. Let t_k be the kth event of some group (with t_0 being the last event of the previous group) and t^* be its corresponding target under the LSTE. That is, t_k is an event iterated by one of the LSTE's sums in Eq. (10), and t^* is its corresponding event returned by the $\min$ function. The input of f for the evaluation of the gradient is therefore $(\vec{z}_{t_{k-1}+1}, \ldots, \vec{z}_{t_k})$. Thus, we need to find the gradient of the LSTE term with respect to $\vec{w}$ when Φ is evaluated at the end of this interval. To do that, we will project the current weight vector $\vec{w}$ on the hyperplane of solutions for the equation

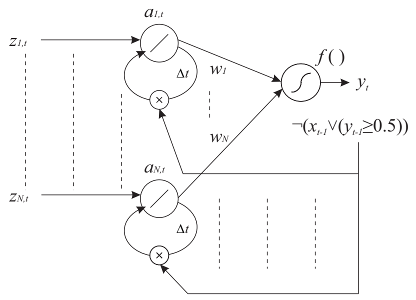

Fig. 8. The LSTE prediction system seen as a special LSTM network.

$\lambda \vec{w}^{*\top} \vec{a}_{t_k} = 1$ where

$$\lambda = \begin{cases} \text{not applicable}, & \text{if } t_k = t^* \text{ (no error);} \\ \frac{(t_k - t_{k-1}+1)}{(t^* - t_{k-1}+1)}, & \text{if } t_k < t^* \text{ and } t_k \in Y \text{ (group A);} \\ \frac{(t_k - t_{k-1}+1)}{(t^* - t_{k-1})}, & \text{if } t_k > t^* \text{ and } t_k \in Y \text{ (group D);} \\ 1, & \text{if } t_k \in X \text{ (groups B and C).} \end{cases} \tag{12}$$

When t_k is a positive prediction in Y (groups A and D), then λ is the ratio between the current length of the interval leading to the prediction t_k by f, and the target length of the interval leading up to event t^*; otherwise (groups B and C), $\lambda = 1$. The solution $\vec{w}^*$ closest to $\vec{w}$ is given by $\vec{w}^* = \vec{w} - \lambda d \vec{a}_{t_k}$ where $d = (\lambda \vec{w}^{*\top} \vec{a}_{t_k} - 1)/\|\lambda \vec{a}_{t_k}\|_2^2$. We can then estimate a gradient $\Delta \vec{w} = \vec{w} - \vec{w}^* = \lambda d \vec{a}_{t_k}$ resulting in the learning rule

$$\vec{w} \leftarrow \vec{w} - \alpha \Delta \vec{w} = \vec{w} - \alpha \frac{(\vec{w}^{*\top} \vec{a}_{t_k} - 1/\lambda)}{\|\vec{a}_{t_k}\|_2^2} \vec{a}_{t_k} \tag{13}$$

to be applied anytime a prediction or an event occurs, and where $0 < \alpha < 1$ is a small learning rate. To guarantee that the weights are pushed enough for Φ_t to reach threshold on time, rules B and C use a slightly modified version of d subtracting $(2f)^{-2}$ from the denominator.

Given Eq. (13), it is also straightforward to devise a batch learning algorithm for DSTE. First compute DSTE, then compute the gradient estimate $\Delta \vec{w}$ for each cost term selected by DSTE (each δ_{STE} grid point), apply the updates, and repeat until convergence.

2) SSE and logit gradient descent: We can rewrite the above model as a special instance of a long short-term memory network (LSTM) [24] with fewer connections as shown in Fig. 8. Each input z_i is connected to a linear recurrent unit (the LSTM memory cell) with a fixed weight of 1. Reusing the time steps notation $X = (x_1, x_2, \ldots, x_L)$ for X and Y, the recurrence weight is determined by the output of another unit (the LSTM forget gate) which is 0 whenever $x_{t-1} = 1$ or $y_{t-1} \geq .5$ and is 1 otherwise. The memory cells are then connected to the output unit through their respective weight w_i. The output unit computes the weighted sum and processes it through a logistic sigmoid function $f(\text{net}) = 1/(1 + e^{-\text{net}})$. This output function will allow the network to learn to output values near 0 and 1 and for the cost function (SSE at each time step) to be differentiable with respect to the network weights. An output $y_t = f(\vec{w}^\top \vec{a}_t)$ will be considered an event prediction if $y_t \geq .5$. This is the same architecture as above with the same degrees of freedom, except for the soft threshold at the output. As an LSTM, the network is trained using gradient descent on the output squared error $(y_t - x_t)^2$ applied at each time step (SSE) with respect to each weight producing the learning rule

$$\vec{w} \leftarrow \vec{w} - \alpha(y_t - x_t) f(\vec{w}^\top \vec{a}_t)(1 - f(\vec{w}^\top \vec{a}_t))\vec{a}_t. \quad (14)$$

The output of $f(\cdot)$ can also be interpreted as the probability $y_t = p(x_t = 1 \mid \vec{a}_t)$ of an event at time step t, and in such case, the negative log likelihood (or cross-entropy) is considered the preferable cost function for such regression (called the logit loss function). We will also test this learning rule which has the following gradient descent update rule

$$\vec{w} \leftarrow \vec{w} - \alpha(y_t - x_t)\vec{a}_t. \quad (15)$$

Note that these two learning rules, unlike the previous one, applies one correction at every time step instead of at every event or prediction.

3) Datasets: The three datasets from Section III-B are used with the following adaptations. For the heartbeat dataset, the input of the prediction system is merely a bias. For finance, stocks are grouped into 11 categories labeled by NASDAQ. The input consists of a bias, whether or not each stock of the category is above its moving average, and whether or not each stock is below its moving average. The outputs to predict are, for each stock in the category, whether or not the stock price crosses the threshold, as in the previous section. Finally, for music, the binary time series are not centered in a longer stream. Instead, they are presented as is but repeated five times in a row to allow the system to learn the piece on-line from hearing it multiple times. The inputs are the bias, whether or not each note is heard, and whether or not each note is not heard, for a total of 41 inputs. Similarly, the system predicts both notes onset and offset for a total of 40 outputs.

4) Method: For each signal, such as a musical piece or a group of stocks, the weights for ten networks are initialized randomly using a $\mathcal{N}(0, \sqrt{N})$ distribution. Each network is then copied, for each algorithm and learning rate (α) pair with $\alpha \in \{0.5, 0.1, 0.05, 0.01, 0.005, 0.001\}$. The first 60% of the signal is used for training. The next 20% of the signal is used for validation to determine the best learning rate for each algorithm and task pair using the algorithm cost function (Eq. (13) for LSTE, Eq. (14) for SSE, and Eq. (15) for logit). Finally the last 20% of the signal is used for testing the performance measure under SSE, DTW, DSTE, and LSTE (the logit loss function cannot be applied to LSTE predictions since it requires probabilities while the LSTE network generates only 0s and 1s). Since this is on-line learning, the learning rate $\alpha > 0$ is constant throughout the whole signal processing.

5) Results: Table III shows the performance of each learning algorithm (LSTE gradient descent, SSE gradient descent, and logit gradient descent) as measured under each cost function discussed in Section II of the paper under each dataset. Strikingly, except the SSE algorithm's SSE test performance on finance, the LSTE gradient descent algorithm outperforms the SSE and logit gradient descent under all these cost functions. That is, the LSTE gradient descent gets better LSTE, DSTE, DTW, and SSE performance than the SSE and logit-based algorithms.

TABLE III
RESULTS OF THE THREE LEARNING ALGORITHMS (LOWER IS BETTER)

Optimizing	Cost Function Test Performance			
Heart	SSE	DTW	DSTE($\times 10^2$)	LSTE($\times 10^2$)
LSTE (α=0.500)	14±4	1740±551	285±324	143±162
Logit (α=0.500)	26±2	2664±379	2379±331	1190±165
SSE (α=0.500)	23±3	2664±379	2379±331	1190±165
Finance	SSE	DTW	DSTE($\times 10^2$)	LSTE($\times 10^2$)
LSTE (α=0.500)	7±2	136±49	11±7	6±4
Logit (α=0.001)	9±3	384±470	26±33	13±17
SSE (α=0.500)	4±1	183±38	53±15	26±7
Music	SSE	DTW	DSTE($\times 10^2$)	LSTE($\times 10^2$)
LSTE (α=0.500)	6±2	281±148	38±32	21±18
Logit (α=0.500)	39±13	2558±1721	652±713	326±357
SSE (α=0.500)	13±8	3231±2456	3225±3465	1613±1733

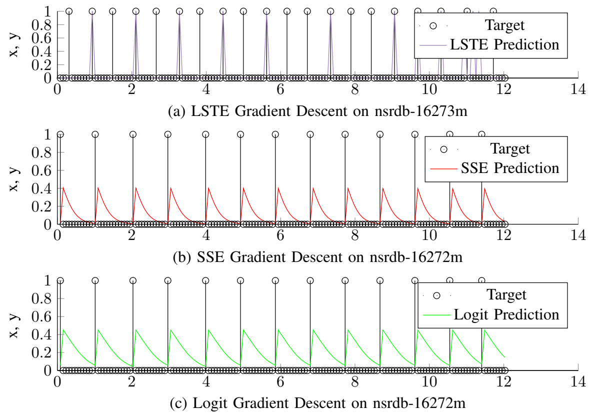

Fig. 9. Best predictions learned on a normal ECGs. Best network trained predictions on the test portion of (a) LSTE on nsrdb-16273m, (b) SSE on nsrdb-16272m, and (c) Logit on nsrdb-16272m. While the LSTE trained network can make predictions at right time steps or in their vicinity (a), even the best SSE (b) and logit (c) trained networks have purely reactive behavior attempting to predict events only after they occurred, and not successfully reaching the event prediction threshold of $y_t \geq 0.5$.

To get a sense of how LSTE gradient descent can outperform SSE gradient under SSE cost, we looked for the best prediction network based on its the training cost function. For example, for SSE in the heartbeats dataset, nsrdb-16272m had the SSE trained network with the lowest SSE test performance. The resulting prediction signals are plotted on the test portion of their target signal in Figs. 9–10 for the finance and heartbeat datasets respectively. As expected, the LSTE gradient descent is producing predictions at, or in the temporal vicinity, of target events, sometimes totally missing an event (Figs. 9–10, panel (a)). In contrast, the SSE and logit gradient descent show purely reactive behavior. That is, following an event, their weights are highly increased, inducing a response on the next time steps. But then, the absence of matching events pulls back down the weights, reducing the network output, despites the increasing values in the accumulators $\vec{a}_t$. (Remember that the event stream y_t is not part of the networks input stream $\vec{z}_t$.) This seems less of a problem for the logit gradient descent compared to the SSE (see Fig. 9, panel (c)) as the cost function considers y_t as a probability of an event $p(x_t = 1 \mid \vec{a}_t)$ at time step t, but still shows a purely reactive behavior. Therefore, the LSTE gradient descent algorithm seems to be the most well-behaved cost function and outperforms standard SSE and logit

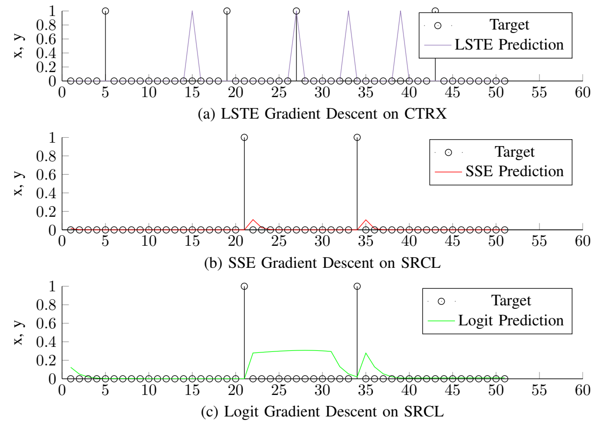

Fig. 10. Best predictions learned on a NASDAQ-100 stock from 16 January 2014 to 31 March 2014. Best network trained predictions on the test portion of (a) LSTE on CTRX, (b) SSE on SRCL, and (c) Logit on SRCL. The results are similar to Fig. 9.

gradient based descent algorithms when learning to predict binary events.

IV. Conclusion

In this paper, we revisit SSE and the reasons for not selecting it as a cost function for binary time-series forecasting, especially when timing is critical [3]; SSE does not convey timing error magnitude well. We then showed that while DTW is a better approach to recognize time series, it is not suitable for on-line and real-time learning of binary time series prediction because it needs to know the whole signal, it has a high $\mathcal{O}(T^2 f^2)$ computational cost, and its warping cost is not differentiable. Inspired by recent animal behavioral models [14], [15], [16], we developed the dynamic squared timing error (DSTE) and its on-line approximation, the local squared timing error (LSTE), to be used when a cost function having Properties 1–5 is required, such as in robotics [2].

In the first set of experiment, we demonstrated that the local minimum squared timing error (LSTE) is an excellent approximation to the dynamic minimum squared timing error (DSTE) and that both behave better than SSE and DTW under five types of timing noise in both artificial and real binary streams of events. We have also shown that unlike DTW, LSTE is differentiable and can be computed quickly on-line. In the second set of experiments, we derived an on-line gradient descent algorithm for LSTE. This algorithm generally outperformed SSE and the logit gradient descent on the same network architecture under all cost functions evaluated. That is, the new LSTE algorithm reached lower SSE and DTW costs than logit and SSE training itself. Moreover, this new algorithm can be directly extended to batch or off-line learning using either LSTE or DSTE. This shows that LSTE has the five properties outlined in Section II-B and that these properties are indeed desirable cost function properties for on-line learning of timing.

Future research should investigate how to generalize LSTE and DSTE to more complex neural networks as well as how to extend it to continuous input (or even output) signals.

Acknowledgment

This research was supported by a start-up fund and an ARP grant from RMC and the Canadian Defence Academy to Dr. Rivest. The authors also thank Nathan Chalmers for his contribution in preliminary work and Rick J. Kohar for his comments on an early draft.

References

[1] J. Schmidhuber, "2006: Celebrating 75 years of AI—History and Outlook: The Next 25 Years," in *50 Years of AI*. Springer, 2007, pp. 29–41.
[2] M. Maniadakis and P. Trahanias, "Temporal cognition: a key ingredient of intelligent systems," *Frontiers in Neurorobotics*, vol. 5, p. 2, 2011.
[3] F. Itakura, "Minimum prediction residual principle applied to speech recognition," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 23, no. 1, pp. 67–72, 1975.
[4] C.-S. Perng, H. Wang, S. R. Zhang, and D. S. Parker, "Landmarks: a new model for similarity-based pattern querying in time series databases," in *Proceedings of the 16th International Conference on Data Engineering*. IEEE, 2000, pp. 33–42.
[5] D. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *AAAI Workshop on Knowledge Discovery in Databases*, 1994, pp. 229–248.
[6] J. A. Ward, P. Lukowicz, and H. W. Gellersen, "Performance metrics for activity recognition," *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 1, pp. 6:1–6:23, Jan. 2011.
[7] J. Frank, S. Mannor, J. Pineau, and D. Precup, "Time series analysis using geometric template matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 3, pp. 740–754, 2013.
[8] F. Rivest, J. F. Kalaska, and Y. Bengio, "Alternative time representation in dopamine models," *Journal of Computational Neuroscience*, vol. 28, no. 1, pp. 107–130, 2010.
[9] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2002.
[10] A. L. Blum and R. L. Rivest, "Training a 3-node neutral network is NP-complete," *Neural Networks*, vol. 5, pp. 117–127, 1992.
[11] C. R. Gallistel and J. Gibbon, "Time, rate, and conditioning," *Psychol. Rev.*, vol. 107, no. 2, pp. 289–344, 2000.
[12] P. D. Balsam, M. R. Drew, and C. Yang, "Timing at the start of associative learning," *Learning and Motivation*, vol. 33, no. 1, pp. 141–155, 2002.
[13] F. Balci, C. R. Gallistel, B. D. Allen, K. M. Frank, J. M. Gibson, and D. Brunner, "Acquisition of peak responding: what is learned?" *Behavioural Processes*, vol. 80, no. 1, pp. 67–75, 2009.
[14] F. Rivest and Y. Bengio, "Adaptive drift-diffusion process to learn time intervals," 2011, arXiv:1103.2382.
[15] P. Simen, F. Balci, L. de Souza, J. D. Cohen, and P. Holmes, "A model of interval timing by neural integration," *Journal of Neuroscience*, vol. 31, no. 25, pp. 9238–9253, 2011.
[16] A. Luzardo, E. A. Ludvig, and F. Rivest, "An adaptive drift-diffusion model of interval timing dynamics," *Behavioural Processes*, vol. 95, pp. 90–99, 2013.
[17] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, 1994.
[18] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics–Doklady*, vol. 10, no. 8, pp. 707–710, 1966.
[19] M. Cuturi and M. Blondel, "Soft-DTW: a differentiable loss function for time-series," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 894–903.
[20] L. Citi, E. Brown, and R. Barbieri, "A point process local likelihood algorithm for robust and automated heart beat detection and correction," in *Computing in Cardiology, 2011*, 2011, pp. 293–296.
[21] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, "Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.
[22] "NASDAQ stock market," May 2014, www.nasdaq.com.
[23] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml
[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, 1997.