2018-11-11

Adaptively-Halting RNN for Tunable Early Classification of Time Series

Thomas Hartvigsen
Worcester Polytechnic Institute, twhartvigsen@wpi.edu

Adaptively-Halting RNN for Tunable Early Time Series Classification

by

Thomas Hartvigsen

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Data Science

November 2018

APPROVED:

Professor Elke A. Rundensteiner, Thesis Advisor

Assistant Professor Xiangnan Kong, Thesis Co-Advisor

Associate Professor Randy C. Paffenroth, Thesis Reader

Abstract

Early time series classification is the task of predicting the class label of a time series before it is observed in its entirety. In time-sensitive domains where information is collected over time it is worth sacrificing some classification accuracy in favor of earlier predictions, ideally early enough for actions to be taken. However, since accuracy and earliness are contradictory objectives, a solution to this problem must find a task-dependent trade-off.

There are two common state-of-the-art methods. The first involves an analyst selecting a timestep at which all predictions must be made. This does not capture earliness on a case-by-case basis, so if the selecting timestep is too early, all later signals are missed, and if a signal happens early, the classifier still waits to generate a prediction. The second method is the exhaustive search for signals, which encodes no timing information and is not scalable to high dimensions or long time series.

We design the first early classification model called EARLIEST to tackle this multi-objective optimization problem, jointly learning (1) to decide at which time step to halt and generate predictions and (2) how to classify the time series. Each of these is learned based on the task and data features. We achieve an analyst-controlled balance between the goals of earliness and accuracy by pairing a recurrent neural network that learns to classify time series as a supervised learning task with a stochastic controller network that learns a halting-policy as a reinforcement learning task. The halting-policy dictates sequential decisions, one per timestep, of whether or not to halt the recurrent neural network and classify the time series early. This pairing of networks optimizes a global objective function that incorporates both earliness and accuracy.

We validate our method via critical clinical prediction tasks in the MIMIC III database from the Beth Israel Deaconess Medical Center along with another publicly available time series classification dataset. We show that EARLIEST out-performs two state-of-the-art LSTM-based early classification methods. Additionally, we dig deeper into our model's performance using a synthetic dataset which shows that EARLIEST learns to halt when it observes signals without having explicit access to signal locations.

The contributions of this work are three-fold. First, our method is the first neural network-based solution to early classification of time series, bringing the recent successes of deep learning to this problem. Second, we present the first reinforcement-learning based solution to the unsupervised nature of early classification, learning the underlying distributions of signals without access to this information through trial and error. Third, we propose the first joint-optimization of earliness and accuracy, allowing learning of complex relationships between these contradictory goals.

Acknowledgements

I would like to thank my advisor Dr. Elke Rundensteiner, co-advisor Dr. Xiangnan Kong, and my research colleague Cansu Sen. Due to their mentorship and support, I have enjoyed learning about machine learning and data mining over this past year and improving my scientific practices. I thank the DSRG for feedback on my work and for providing a productive research community, in particular John Boaz Lee for his insights in tackling this problem. I would also like to thank the WPI Academic and Research Computing team, particularly Spencer Pruitt, for their hard work providing high performance computing resources to the WPI community. I would also like to thank the U.S. Department of Education for supporting my PhD studies via the grant P200A150306 on "GAANN Fellowships to Support Data-Driven Computing Research".

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Traditional time series classification assumes that a time series as a whole is considered before predicting its class label. In time-sensitive applications, however, it is essential that predictions are generated before the entire series has been observed. For example, in clinical diagnosis, it is often worth sacrificing some classification accuracy in favor of earlier predictions to give clinicians enough time to address infections as they evolve for the sake of the patient's health and to curb spread of infections. In these settings, an analyst must determine how much accuracy to sacrifice in favor of earliness, with the optimal trade-off depending on the task.

Figure 1.1 depicts an example of the early classification problem where each time series contains different signals, indicating their respective class labels. Case 1 shows the traditional classification scheme, predicting labels after observing all timesteps of a time series. This results in a highly accurate classifier that captures all signals but provides predictions at the very end (indicated by the dashed halting line). Case 2 shows strict early classifications, choosing a fixed timestep at which

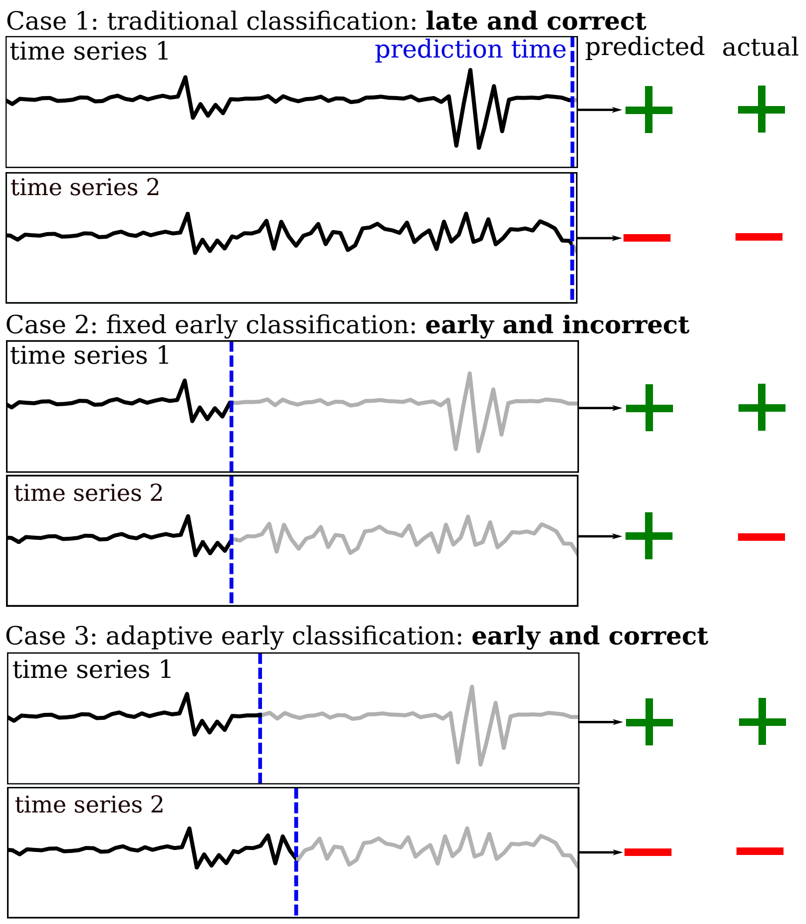

Case 1: traditional classification: **late and correct**
time series 1
prediction time
predicted
actual
+
+
time series 2
−
−
Case 2: fixed early classification: **early and incorrect**
time series 1
+
+
time series 2
+
−
Case 3: adaptive early classification: **early and correct**
time series 1
+
+
time series 2
−
−

Figure 1.1: Example of early classification of two time series. + and – denote class labels; vertical dashed lines indicate halting-points. Timesteps after halting-points in gray are not used for classification.

to always stop for each of the time series and make the prediction. In this case, predictions tend to be incorrect more often since signals may not have arrived yet. Case 3 shows adaptive early classification which selects a halting-point on a case-by-case basis (vertical line), allowing for both early and accurate predictions. In conclusion, an effective model for time series classification in time-sensitive domains should not only model discriminating signals, but also identify a timestep at which enough information has been observed to reliably (to the requested degree) predict a label. It must also be tunable between the domain-specific emphasis on accuracy versus earliness.

1.2 State-of-the-Art

In the literature, early classification of time series (ECTS) has been studied extensively [1–8]. Most of these methods involve searching for sub-time series that imply a specific class label [1, 2, 5–7], called *shapelets* [9]. After extensive shapelet search, distances are computed from identified shapelets to subsequences in the time series. In this setting, all combinations of subsequence lengths must be considered. This search-based approach is expensive when considering multivariate time series since true exhaustive search requires the consideration of all combinations of subsequence lengths. For high dimensional data and long time series, this becomes an intractible problem [5].

Some works [3,4] instead search for time series prefixes, limiting the search-space. These works use many separate classifiers for each different prefix length – without, however, sharing parameters across these prefix lengths.

The aforementioned methods have two major limitations. First, they do not address tunability between accuracy and earliness. Second, they are not end-to-end: they first identify shapelet candidates, then seek early solutions, leading to objectives with different goals. This also limits tunability since while searching for shapelets candidates are not discovered with respect to earliness. These topics are further discussed in Chapter 2.

1.3 Problem Definition

The ECTS problem corresponds to the problem of selecting a timestep in a time series at which to predict a class label. Given a multivariate time series, the goal of ECTS is to both classify the time series and find a timestep at which to predict this classification. The selected timestep must be less than or equal to the full length of the time series. There are two objectives of ECTS: earliness and accuracy. Since they are contradictory goals, often times there must be a trade-off between these goals. It is particularly challenging to balance the number of observed timesteps with expected accuracy since these trade-offs are task-dependent and thus a good solution for one time series may be bad for another.

1.4 Challenges

Despite the importance of ECTS, many topics remain understudied. We summarize major challenges as follows:

- *Lack of supervision*: There are no labels indicating where signals occur within a time series; instead the complete time series is labeled by its class in most labeled data sets. Thus, quantifying whether or not a prediction *should* be made at a particular timestep is difficult. This is thus an inherently unsupervised problem within the otherwise supervised learning problem.

- *Multiple conflicting objectives*: Earliness and accuracy tend to contradict one-another. A maximally-early classifier may not have enough information to make accurate predictions, while a late classification causes unnecessary delay and misses precious opportunity to react. The balance is task-specific and optimal trade-offs depend on the task and domain.

- *Multivariate signals*: In multivariate time series, signals indicative of a particular class label may develop at vastly different times between variables, making the identification of halting points for the overall time series composed of all variables harder.

1.5 EARLIEST

In this paper, we propose a solution to the aforementioned challenges called **E**arly and **A**daptive **R**ecurrent **L**abel **EST**imator, or for short, EARLIEST. EARLIEST works by augmenting a recurrent neural network (RNN)-based *Discriminator* with a reinforcement learning-based stochastic *Controller* network and optimizing their cooperation. The discriminator predicts time series labels and the controller decides at each timestep whether to *stop and predict a label* or to *wait and request more data from the next timestep.* By rewarding the controller based on the success of the discriminator and tuning the penalization of late predictions, the controller learns a *halting policy* which directs the online halting-point selection. This results in a learned balance between *earliness* and *accuracy* according to requirements of the task at hand. In contrast to traditional ECTS methods, our proposed method supports flexible earliness-accuracy trade-offs per task, being optimized for both earliness and accuracy together in one end-to-end model. The resultant model is also applicable to a wide variety of time-sensitive classification tasks such as early video or text classification [10–12]. Empirical studies on real-world tasks demonstrate that our approach outperforms baseline methods while providing simple balancing of emphasis on opposing goals.

The main contributions of our work can be summarized as follows:

- We propose the first ECTS method that handles the unsupervised aspect of early classification by formulating halting-point selection as a reinforcement learning problem.

- We introduce the first neural-network ECTS method, which learns to combine any number of variables into low-dimensional representations which are then used to classify the time series.

- We design the first dual-optimization of the *earliness* and *accuracy* goals by combining them into one integrated objective function. This allows for an analyst to manually select a trade-off depending on the task via one hyperparameter.

- We apply our method to three real-world time-sensitive time series classification tasks. Results show that our method's performance significantly outperforms baselines in accuracy and earliness.

Chapter 2

Related Work

To the best of our knowledge, this is the first work supporting task-dependent tunability in ECTS, supporting both univariate and multivariate data. Our work is built upon ECTS methods, conditional computation in neural networks, recurrent neural networks, and Partially Observable Markov Decision Processes. We discuss each of them here.

2.1 Early Classification of Time Series

ECTS deals with predicting labels of time series before the time series is fully observed. Many works have been proposed based on updating traditional distance-based classifiers, such as kNN [1–3,6,7]. A well-known approach is to do a similarity search for shapelets, or sub time series indicative of a class [9], and then find their earliest occurrences. Typically, this involves extracting many sub-time series as shapelet candidates and pruning them based on their classification power. Then, a trade-off between accuracy and earliness could be simulated by lowering the support required to be a shapelet [1]. However, at test time, there is only a matching step as opposed to actually computing the risk associated with predicting at a particular timestep. Thus, these models are not inherently "time-aware", as shapelets do not capture the timing of an event. For example, the same signal may indicate different classes if it appears at different timesteps according to the class. Shapelet methods do not consider this to be discriminative. An additional issue with these methods is that the search space for shapelets increases exponentially with both the time series length and the number of variables [6].

Hence, prefix-based ECTS methods are another promising approach where sub-time series are extracted with the condition that they must begin at the first timestep [3, 4]. However, these existing methods unfortunately require a multitude of classifiers, one per time series length. Thus, they miss the potential to learn relationships between time series of similar lengths. Using one model for all time series lengths allows for the learning of more complicated relationships, potentially better-supporting time series with varying lengths and leaning upon the fact that sub-time series of different lengths can still be related to one another.

In these methods [1–4, 6, 7], feature extraction and prediction are entirely separate, and so the tasks are unaware of each other. Hence, they are not optimized together in one objective function, possibly missing out on natural connections between these two goals.

One method also studied ECTS as a multivariate marked point-process and tries to extract features related to different signals in multivariate time series [8] but does not address tunability. Beyond time series, some works have also studied early video classification [8, 11] and early text classification [10].

2.2 Conditional Computation

Conditional computation in neural networks deals with learning when to activate different parts of neural networks, depending on the input data [13]. This can reduce the extensive computation required to train a neural network since fewer computations need to be made per example [14]. Additionally, the depth of a neural network has a major impact on performance [15] but selecting the proper network complexity remains empirical and is often seen as more an art than a science. This motivates gating across network layers, allowing for direct information flow, such as in Highway Networks [16]. One work uses reinforcement learning strategies to learn a conditional computation policy, selectively turning on and off blocks of a neural network [17] but does not study early classification. Our model leverages the idea of selectively activating parts of a neural network and can be viewed as longitudinal conditional computation: learning when to activate sections of a network *in time*, or in other words, learning at each timestep whether or not to activate the neurons at the next timestep.

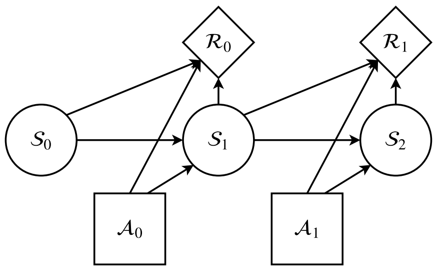
2.3 RNN & LSTM Background

RNNs have emerged as the state-of-the-art for many time series analysis models [18, 19] and other sequence modeling tasks such as sequence generation [20]. Our proposed model builds on RNNs since they have been shown to be powerful tools for constructing vector representations for real-valued sequences [21]. At each step of a sequence, a new representation is learned via a function of the previous representation and new data observed at the current step. The final vector, computed at the final step and modeling dynamics of the sequence, can then be used for prediction. Empirically, RNNs struggle to model long-term longitudinal dependencies due to the vanishing-gradient problem [22]. The Long Short-Term Memory (LSTM) cell [23] helps with this problem by augmenting the classic RNN with a memory vector that is persistent across timesteps, learning to remember and forget information longitudinally. Much of the success in RNNs has been found using LSTM cells, and thus our model uses this augmentation as well. However, it is also possible to easily swap in other memory cells, such as the Gated Recurrent Unit (GRU) [24].

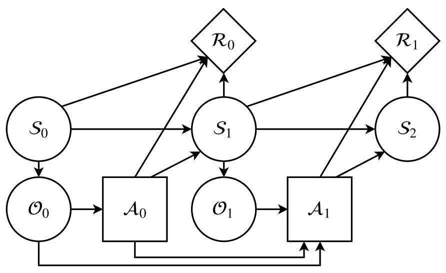
2.4 Markov Decision Processes

Reinforcement learning problems are a solution to problems that can be described using Markov Decision Processes (MDP), where an agent exists in an environment and sequentially takes actions that attempt to ultimately maximize some reward, which quantifies the agent's success with respect to a specific task. As the agent takes actions, the environment changes (*e.g.*, a chess move alters the current status of the board), and the status of the environment is referred to as the environment's current *state*. In simple examples, this state can be fully observed by the agent (*e.g.*, when learning to play chess, a chess-playing agent could see the entire board, thus having access to all possible available information). Shown in Figure 2.1a, the MDP begins with an initial state $\mathcal{S}_0$. The agent then selects an action $\mathcal{A}_0$ which prompts the transition to the next state $\mathcal{S}_1$. $\mathcal{S}_1$ then determines the observed reward $\mathcal{R}_1$, which quantifies the success of selection action $\mathcal{A}_0$. This process is repeated until a terminal state $\mathcal{S}_T$ is reached and the to solve the MDP is to learn a policy which maximizes the expected sum of rewards, $R = \sum_{i=0}^{T} \mathcal{R}_i$.

However, when scaling into more complicated settings it becomes impossible to fully observe all features of the environment. For example, in hospitals, doctors



(a) Markov Decision Process.



(b) Partially Observable Markov Decision Process.

Figure 2.1: MDP vs. POMDP

observe patients, but cannot understand every detail of a patient's health before they must take actions (*e.g.*, prescribe treatments). This motivates *partially-observable* MDP's (POMDP) where it is assumed that an MDP still describes the sequential decision-making problem, but the agent does not have access to the entire state of the environment. As shown in Figure 2.1b, PODMP shares many traits with MDP. However, in this setting, actions are selected after receiving observations from the state, assuming that there is an MDP at play but that the agent does not have full observation of the state. Additionally, in the partially-observable setting, the agent needs to remember the list of observations since its actions are based on this sequence. In our proposed method, the *controller network* solves a POMDP, thus learning a *halting-policy*.

Chapter 3

Methodology

3.1 Problem Formulation.

Given a set of labeled multivariate time series, $\mathcal{D} = \{(X, y)\}$ containing N time series instances and labels, consider the i^{th} instance

$$X^{(i)} = \begin{bmatrix} | & | & & | \\ x_1^{(i)} & x_2^{(i)} & \cdots & x_T^{(i)} \\ | & | & & | \end{bmatrix}$$

where $x_t^{(i)} \in \mathbb{R}^M$ contains the M variables recorded at time t. Henceforth, for ease-of-reading, we describe our method for one time series and omit index i when it is not ambiguous. The aim is to learn parameters θ of a function $f(\cdot)$, which maps a time series X to a label $\hat{y}$, as $f_\theta(X) \to \hat{y}$, ultimately predicting labels for each of the N instances. During the training process, the goal is to match predicted labels $\hat{y}$ to their corresponding true labels y where $y \in Y$ denotes the label associated with X and $Y = \{0, \cdots, L\}$, the set of possible class labels from a total of L classes.

In this work, we model f_θ as a combination of neural networks. However, as op-

Table 3.1: Basic Notation

Notation	Explanation
N	Number of time series instances.
M	Number of variables per time series.
L	Number of classes.
$T^{(i)}$	Number of timesteps for instance i.
$X_t^{(i)}$	Variables at timestep t for instance i.
$y^{(i)}$	True label for instance i.
$\tau^{(i)}$	Chosen halting-point for instance i.
$S_t^{(i)}$	Learned representation of $X_{0,\cdots,t}^{(i)}$.

posed to using all T timesteps to generate this prediction, we seek prefixes of length $\tau \leq T$ for each time series which is both small enough to satisfy the requirement for earliness and large enough to satisfy the requirement for successful classification accuracy. We refer to the selected τ as the *halting-point*.

As an example, for in-hospital adverse-event detection, a multivariate time series $X^{(i)}$ may contain a patient's vital signs recorded longitudinally throughout her stay. This instance is labeled positive, $y^{(i)} = 1$, indicating that the adverse event occurs. Otherwise, $X^{(i)}$ belongs to the control group and $y^{(i)} = 0$.

3.2 The Proposed Method.

The aims of our proposed adaptively-halting RNN, named EARLIEST, are twofold. First, to model multivariate time series for classification, and second, to select a *halting-point* at which enough timesteps have been observed to make a task-dependently adequate prediction. EARLIEST is a deep neural network consisting of three sub-networks: (1) a *Base RNN* which learns to model multivariate time series, (2) a *Discriminator Network*, or *Discriminator*, which learns to predict class labels based on the *Base RNN*'s model, and (3) a *Controller Network*, or *Controller*, which decides at each step whether or not to halt the *Base RNN* and activate the

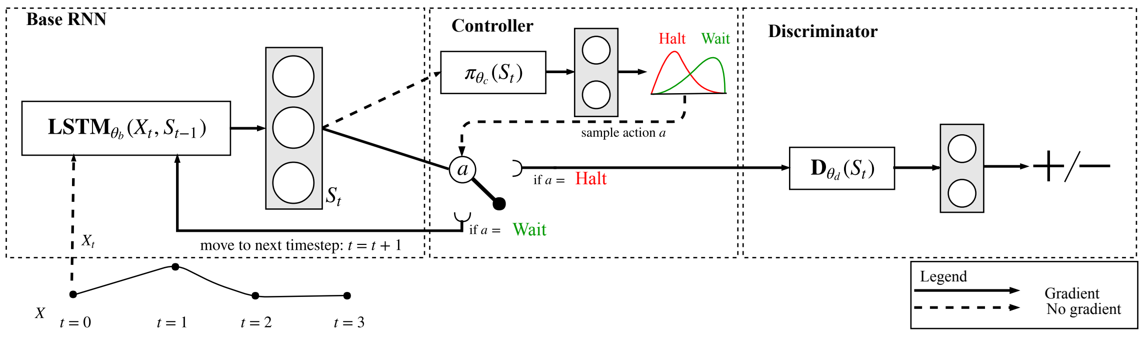


Figure 3.1: Overview of EARLIEST. Selected action a chooses whether or not to pass S_t to the *Discriminator* or back to the *Base RNN* to process the next timestep. Dashed lines indicate no gradient flow through these paths.

Discriminator. As soon as the *Controller* chooses to halt, the processing of the current time series is complete. An overview of EARLIEST is shown in Figure 3.1.

The *Discriminator* is trained with respect to the classification task while the *Controller* is rewarded based on the success of the *Discriminator* and is punished based on how many steps it takes before deciding to halt. Thus, the *Controller* and *Discriminator* learn to cooperate to make correct predictions. To incorporate earliness, we add to the final objective function a loss term that competes with the *Controller*'s natural tendency to wait, thus balancing the trade-off between accuracy and earliness according to the scale of this loss term. The final output of EARLIEST is a label $\hat{y}$ which is generated at some *halting point* τ, where $\tau \leq T$. The tunability of the model dictates how much less τ is than T, which often affects the accuracy of the model, depending on where signals are located in the time series.

3.2.1 Base Recurrent Neural Network.

An RNN augmented with LSTM cells rests at the heart of EARLIEST, mapping variables observed at each timestep, X_t, to vector representations $S_t \in \mathbb{R}^k$ where k is the number of hidden dimensions, a tunable hyperparameter. Standard to RNN literature, we refer to the whole recurrent part of the network simply as an LSTM.

One vector S_t is created per timestep and is referred to as the *hidden state*. Theoretically, each vector S_t summarizes the time series dynamics present in $X_{\{0,\cdots,t\}}$. Since these vectors inform the other parts of the network, we refer to this recurrent component as the *Base RNN*.

The LSTM is a function which learns to represent time series data as vectors. Hidden state vector S_t is computed as a function of currently-observed data X_t and the previous hidden state S_{t-1}, hence the recurrent nature of the model. In an LSTM, the computation of S_t relies upon the computation of a cell memory state C_t, which is then used to compute hidden state S_t. The LSTM's success comes from learned gating mechanisms that curate information contained in vector C_t. To compute C_t, first a *forget* gate controls what information to remove from previous cell state C_{t-1}, where square brackets ([ ]) indicate a stacked column vector combining the contents of the brackets into one vector:

$$f_t = \sigma(W_f \cdot [S_{t-1}, X_t] + b_f) \tag{3.1}$$

An *input* gate controls new information added to C_t:

$$i_t = \sigma(W_i \cdot [S_{t-1}, X_t] + b_i) \tag{3.2}$$

C_t is then computed as the gated combination of the previous memory state C_{t-1} and the current input X_t using the *forget* and *input* gates, where $\odot$ indicates the hadamard product:

$$C_t = f_t \odot C_{t-1} + i_t \odot \eta(W_c \cdot [S_{t-1}, X_t] + b_c) \tag{3.3}$$

Finally, state representation S_t is computed through an *output* gate shown in Equation 3.4 operating on a non-linear C_t shown in Equation 3.5.

$$o_t = \sigma(W_o \cdot [S_{t-1}, X_t] + b_o) \tag{3.4}$$

$$S_t = o_t \odot \eta(C_t) \tag{3.5}$$

S_t is then used to inform decisions made by the *Controller*, generate classifications by the *Discriminator*, and compute the next hidden states S_{t+1} if the *Controller* so chooses. In these equations, W's and b's are learnable parameters, $\eta(\cdot)$ is the hyperbolic tangent function, and σ is the sigmoid function. For conciseness, we group these parameters into one large matrix θ_b. We denote this entire process as function LSTM$(\cdot)$ such that $\text{LSTM}_{\theta_b}(X_t, S_{t-1}) = S_t$. While we use LSTM cells in this work, it is also possible to use alternative gating-mechanisms, such as the Gated Recurrent Unit [24].

3.2.2 Controller Network.

The *Controller* is a reinforcement learning agent that decides whether or not to halt the *Base RNN* at each timestep, prompting the prediction of a label. To achieve this goal, the *Controller* solves a Partially-Observable Markov Decision Process (POMDP) where at each timestep observations from a state arrive, an action is sampled using a learned policy, and a reward is observed according to the quality of the selected action. Its objective is to optimize long-term rewards according to the success of the *Discriminator*, which we accomplish using gradient-based policy search.

State: At each timestep t, the state is the set of currently observed time series variables X_t, essentially a slice across all variables at timestep t. Taking advantage of the representational power of the *Base RNN*, the hidden state S_t is used as an observation from this state space. S_t informs the selection of an action by the *policy*.

Policy: Next, an action is selected by a stochastic policy $\pi_{\theta_c}(S_t) = a_t$, which treats input S_t as immutable data. We use a one-layer fully-connected neural network to approximate this function. Typical to reinforcement learning, we sample the action from a parameterized distribution. Thus, we learn a function mapping S_t to p_t, where p_t is the probability of halting, computed as

$$\begin{aligned} p_t &= \sigma(W_{ha}S_t + b_{ha}) \\ &= \frac{e^{W_{ha}S_t + b_{ha}}}{e^{W_{ha}S_t + b_{ha}} + 1} \end{aligned} \tag{3.6}$$

where W_{ha} and b_{ha} are learnable parameters for mapping hidden outputs to actions and σ is the sigmoid function, which ensures outputs between zero and one. p_t then parameterizes a Bernoulli distribution from which action a_t is sampled according to $P(a_t = 1) = p_t$.

Actions: Sampled action a_t dictates the proceedings of the *Base RNN* as follows: if $a_t = 0$, the *Controller* has selected WAIT. This prompts the Base RNN to move forward one timestep, the action-selection process beginning again with $\text{LSTM}(X_{t+1}) = S_{t+1}$. On the other hand, if $a_t = 1$, the *Controller* has selected HALT, at which point the *Discriminator* is activated to predict a label and processing of the current time series ends. Once the controller selects HALT (or if $t = T$), we consider t to be the halting point τ. To add exploration to the *Controller*, we use an ε-greedy approach: with probability ε, action a_t is replaced with a random action and exponentially decrease ε from 1 to 0 during training, as shown in Equation 3.7.

As the model trains, ε exponentially decreases from 1 to 0.

$$a_t = \begin{cases} a_t & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \tag{3.7}$$

Reward: To train the *Controller*, it must observe returns which qualify the parameters of the current policy. To encourage cooperation between the *Controller* and *Discriminator*, this return takes the form of a reward that quantifies the success of the *Discriminator*. Thus, when the *Discriminator* is correct, we set reward $r_t = 1$, and when it is incorrect, $r_t =$ -1. The objective of the *Controller* is to maximize total reward $R = \sum_{t=0}^{\tau} r_t$.

3.2.3 Discriminator Network.

The *Discriminator* generates a prediction $\hat{y}$ by first projecting the hidden state S_t into L-dimensional space using a fully-connected layer. Next, the resulting vector is normalized to sum to one via the softmax function and can be treated as probabilities. This computation is shown in Equation 3.8 where W_{ho} and b_{ho} are parameters for mapping the hidden state to the output space and are grouped into matrix θ_d.

$$\begin{aligned} \mathrm{P}(Y = i \mid S_t, W_{ho}, b_{ho}) &= \mathrm{softmax}(W_{ho}S_t + b_{ho}) \\ &= \frac{e^{W_{ho}S_t + b_{ho}}}{\sum_j e^{W_{ho}S_t + b_{ho}}} \end{aligned} \tag{3.8}$$

Since the output vector sums to one, predicted label $\hat{y}$ is simply the maximum probability:

$$\hat{y} = \arg\max_i \mathrm{P}(Y = i \mid S_t, W_{ho}, b_{ho}) \tag{3.9}$$

3.2.4 Training.

In the training phase, the goal is to iteratively update all learnable parameters of EARLIEST, minimizing errors made by the *Discriminator* and maximizing the rewards observed by the *Controller*. For readability, we gather all learnable parameters of EARLIEST into matrix θ. EARLIEST is optimized by minimizing one loss function $J(\theta)$, shown in Equation 3.14, using stochastic gradient descent (SGD). The *Base RNN* and *Discriminator* are optimized together with respect to cross entropy loss shown in Equation 3.10 where θ_{bd} indicates the parameters from the *Base RNN* and *Discriminator*.

$$J_{bd}(\theta_{bd}) = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y})) \tag{3.10}$$

In contrast to the *Base RNN* and Discriminator, the goal of the *Controller* is to find parameters θ_c that attain the highest expected return

$$\theta_c^* = \arg\max_{\theta_c} \mathbb{E}[R]. \tag{3.11}$$

Since the *Controller* involves sampling actions, back-propagation does not directly apply, mandating transformation from this raw form to a surrogate loss function [25]. This objective can thus be optimized using gradient descent by taking steps in the direction of $\mathbb{E}[R\nabla\log\pi(S_{0,\cdots,\tau}, a_{0,\dots,\tau}, r_{0,\cdots,\tau})]$ [26]. The gradient can be transformed into the loss function shown Equation 3.12 resulting in the REINFORCE algorithm [27].

$$J_c(\theta_c) = -\mathbb{E}\left[R\sum_{t=0}^{\tau}\log\pi(a_t|S_t)\right] \tag{3.12}$$

However, minimizing $J_c(\theta_c)$ directly leads to gradient estimates that change dramatically across examples, resulting in high-variance policy updates since each example is treated as if in isolation. To handle this, we add a baseline to $J_c(\theta_c)$, similar to [28], so that θ_c is updated based on *how much better than average* the observed reward is, resulting in

$$J_c(\theta_c) = -\mathbb{E}\Bigg[\sum_{t=0}^{\tau} \log \pi(a_t|S_t)\Bigg[\sum_{t'=t}^{T} \big(R - b_t)\big)\Bigg]\Bigg], \tag{3.13}$$

where b_t is predicted at each timestep. We learn this baseline by reducing the mean squared error between b_t and R, forcing b_t to approximate the mean R.

3.2.5 Balancing Earliness and Accuracy.

Up to this point, the *Controller*'s only objective is to maximize the performance of the *Discriminator*. To add earliness, we employ an additional loss term, shown as the final term of our final loss function $J(\theta)$ in Equation 3.14. This loss term encourages halting, depending on hyperparameter λ. When λ is large, to minimize the loss, the probability of selecting HALT must be large. Specifically, λ technically has an unlimited range of possible values, but below zero values will flip its effect (negative λ encourages WAIT) and at some large enough value, EARLIEST will be halting at the first time step for all time series and increasing λ any more is meaningless. Empirically, we explore $\lambda \in [0, 0.15]$, which captures this range in our experiments.

$$J(\theta) = J_{bd}(\theta_{bd}) + J_c(\theta_c) + \lambda \sum_{t=0}^{\tau} - \log \pi(a_t = 1 \mid S_t) \tag{3.14}$$

Thus, since minimizing the log probability corresponds to increasing the probability, by increasing λ, we effectively increase emphasis on HALT. On the other-hand, when λ is small or 0, it leaves the *Controller* free to exclusively maximize the performance of the *Discriminator*. We note that in some cases, this may not mean observing all timesteps. For example, if a time series is too long, the LSTM may have trouble remembering relevant information. Altogether, this loss term creates competition on the optimization of the *Controller*'s parameters as they are tugged in opposite directions, the force of the tugging being controlled by λ.

Chapter 4

Evaluation

4.1 Experimental Methodology

We evaluate our method on synthetic data and three real-world datasets.

`SimpleSignal`: As signals within time series are rarely labeled, we create a synthetic dataset to better evaluate how EARLIEST chooses halting points. Each time series is 10 timesteps long and is initialized with a 0 at every timestep. Then, for positive examples, we sample a location $t \in \{0, \dots, T\}$ from a selected distribution and substitute a 1 at timestep t. Negative examples remain 0's. The selection of this distribution allows us to test EARLIEST's signal-capturing performance in a variety of settings. We use four signal distributions in our experiments: uniform, normal, left-skewed, and right-skewed. Since these distributions result in drastically different signal locations, we can better understand if EARLIEST halts upon observing a signal. For instance, the right-skewed signal distribution allows us to test whether or not EARLIEST waits for long periods of time when it does not observe a signal. Ideally, EARLIEST matches these distributions without having direct access to this information.

Datasets `Mortality` and `MRSA` come from the publicly-available MIMIC III database [29] comprised of Electronic Health Records (EHR) collected from the Beth Israel Deaconess Medical Center in Boston, Massachusetts. Contained in these clinical records are time series of vital signs and microbiology tests. For each clinical task, early predictions allow clinicians to take actions that directly benefit patient well-being. For `Mortality` we extract patients with positive *Mortality Flags*, indicating that they perished during their stay. The task is to predict early whether or not a patient will die. `MRSA` is a prevalent in-hospital acquired infection. To extract patients who test positive for `MRSA`, we use positive microbiology tests for organism "80293". For both datasets, we ensure a balance between positive and negative classes by drawing negative examples randomly from the rest of the database. For each task, we use the five most frequently recorded vital signs as our variables. `Mortality` consists of 11,508 instances and `MRSA` consists of 2600 instances. Since the raw timestamps can be very fine-grained, leading to sparse data as variables aren't often recorded at the same time, we take hourly averages for each variable, fill missing values with variable-wise means, and finally use only the first 10-hours of recordings. Each of these tasks is of utmost clinical importance and the sooner a caretaker can be made aware of ailments as the develop, the better the outcomes for patients.

`ItalyPowerDemand` [30] comes from the publicly-available *UCR Time Series Classification Archive* and is frequently used for classification problems. This dataset contains 1,096 time series with 24 timesteps each and 2 classes. Since the pre-selected training set is small and neural networks tend to succeed with larger training sets, we first combine the given training and testing sets, then shuffle the instances into 80% training, 10% validation, and 10% testing subsets. To ensure fairness in this paper, we train methods only on these splits.

4.2 Alternative Algorithms

We compare the performance of EARLIEST to the following algorithms.

- *LSTM-FH* [11, 31]. Fixed halting-point selection is common in time-sensitive classification tasks. It requires that an analyst pre-selects a timestep at which all classifications will be made. Since EARLIEST uses an LSTM, we use a fixed halting-point version of LSTM, referred to as *LSTM-FH*.

- *LSTM-s* [11]. Designed for early classification of video, *LSTM-s* can be applied to time series. It is similar to an LSTM version of the ECTS algorithm ECDIRE [4]. *LSTM-s* encourages early confidence in its predictions by penalizing the model when it becomes less confident. This method also uses fixed halting-points, classifying all time series at the same pre-selected timestep.

Other existing ECTS algorithms that support multivariate time series [6, 7] do not support multiple trade-offs so our model is not directly comparable to them.

4.3 Implementation Details

For all datasets, we use an 80% training, 10% validation, and 10% testing splits. For each dataset, we use the training set to tune the model's parameters and the validation set to evaluate the performance a particular hyperparameter setting (*e.g.*, nodes per layer or learning rate). The training and validation sets are used many times to tune hyperparameters, then the testing set is used once to compute the final accuracy. Using the validation set to pick hyperparameters, we learn 10-dimensional embeddings for time series variables, and sequences of 10-dimensional representations for each time series. All results shown are averages over 10 repetitions of the training process on random dataset splits. The model is optimized using RMSProp with a learning rate of 0.001.

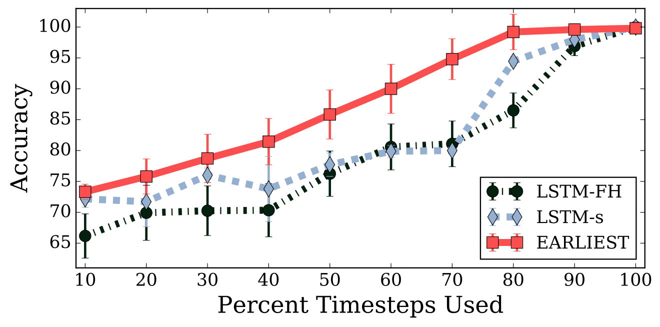
4.4 Experimental Results

4.4.1 Experiments on Synthetic Data

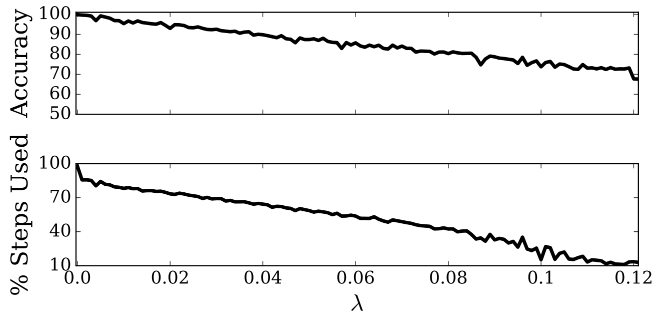
We first evaluate the performance of EARLIEST in a controllable setting where signal locations are known using `SimpleSignal`, our synthetic dataset described in Section 4.1. We evaluate EARLIEST in two ways: First, computing how early and accurate EARLIEST is compared to our baselines, driven by controlling λ, the earliness-accuracy trade-off hyperparameter. Second, how quickly EARLIEST halts when it observes signals, thus matching the true distribution of signal locations.

Accuracy and timing: EARLIEST should more accurately classify instances earlier than the baseline methods due to its adaptive-halting. In Figure 4.1, EARLIEST is run using the earliness hyperparameter $\lambda \in [0.0, 0.15]$, which empirically led to full coverage of all possible halting points. Any larger or smaller λ values do not change the results of these experiments. λ does not directly control accuracy or earliness, instead urging the optimization in one direction or the other. Thus, for each λ, EARLIEST stabilizes at some accuracy and distribution of halting points. From ten iterations of each λ, we extract the mean accuracies and halting-points (computed as the average percent of timesteps used, or $\frac{\tau}{T}$) with baseline predictions made at the same time. We see in Figure 4.1a that for nearly all halting-points, EARLIEST significantly outperforms the baselines. We report the average accuracy and percent timesteps used for each λ in Figure 4.1b, showing that λ allows for smooth coverage of all halting-points.

Signal-capturing: EARLIEST should halt when it sees a signal, and wait oth-



(a) Accuracy and timing

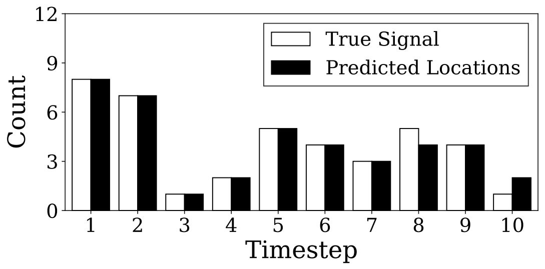


(b) λ coverage

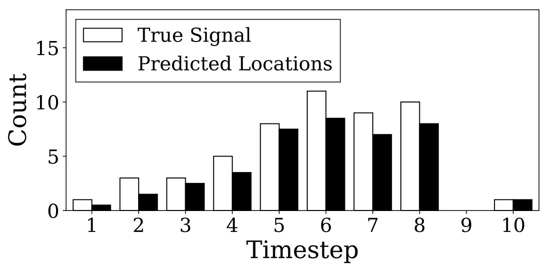
Figure 4.1: Accuracy and prediction times on synthetic data. (a) EARLIEST makes predictions more accurately and earlier than baselines across. (b) λ has control over halting at all timesteps.

erwise. To understand if this is the case, we compute the root mean squared error (RMSE) between EARLIEST's selected halting points and the true distribution of signals, thus quantifying how well EARLIEST halts when it sees a signal. In `SimpleSignal` we use four distributions of signals. For each distribution we expect that EARLIEST should halt when it observes a positive signal and otherwise wait until the end of the time series to classify negative instances. Next, in Figure 4.2, we show the raw timesteps where EARLIEST chooses to halt using $\lambda = 0.014$, a value which empirically performs well on all distributions. Shown halting-points are averages over ten random shuffles of the dataset.

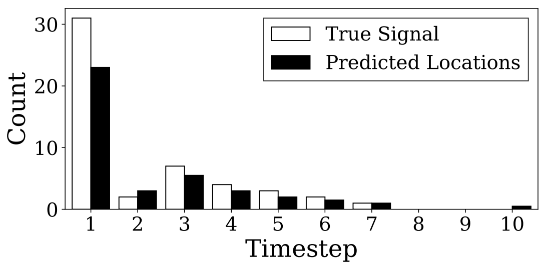
We first show results from sampling signal locations from a *uniform* distribution. Thus, when building `SimpleSignal`, each timestep is equally likely to be selected



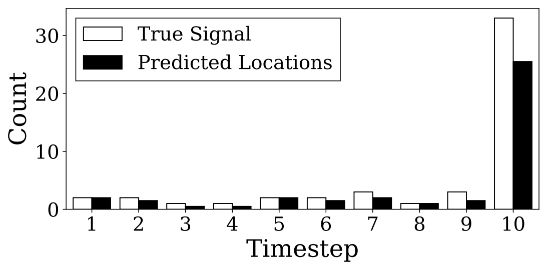
(a) Uniform signals.



(b) Normal signals.



(c) Left-skewed signals.

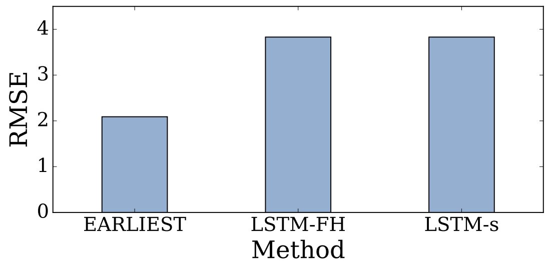


(d) Right-skewed signals.

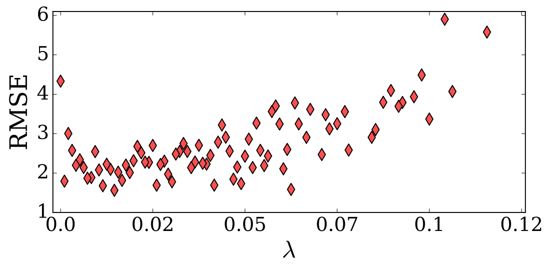
Figure 4.2: True Signal indicates where signals actually appear in the time series, *Predicted Locations* shows the halting-points selected by EARLIEST. $\lambda = .014$ for each setting.

while adding signals. Figure 4.2a that EARLIEST matches the underlying distribution of signals despite not having direct access to this information. Second, we sample signal locations from a *normal* distribution with a mean of 5.0 and a standard devion of 2.0, in order to keep the signals roughly in the center of the time series. We see in Figure 4.2b that EARLIEST matches the distribution, neither halting too early nor too late. In Figures 4.2c and 4.2d we see that the model captures the signals close to their true locations, waiting when it does not observe a signal. Thus, the model learns an effective halting-policy which can be expected to halt when it observes signals and wait otherwise.

We next compare signal-capture between EARLIEST and the alternative algorithms. In Figure 4.3a EARLIEST with $\lambda = .014$ dramatically outperforms *LSTM-FH* and *LSTM-s*, showing that EARLIEST is superior at halting when it observes signals. Additionally, we show how sensitive RMSE is to hyperparameter λ in Fig-



(a) Method comparison.



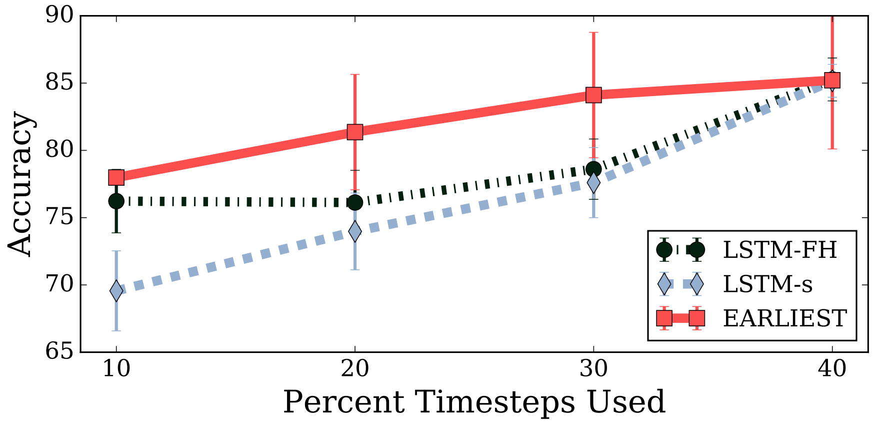
(b) RMSE sensitivity to λ.

Figure 4.3: EARLIEST's signal-capturing capabilities. (a) Comparing how well each method captures the true signal distribution. (b) Lower points indicate EARLIEST halting when it sees signals.

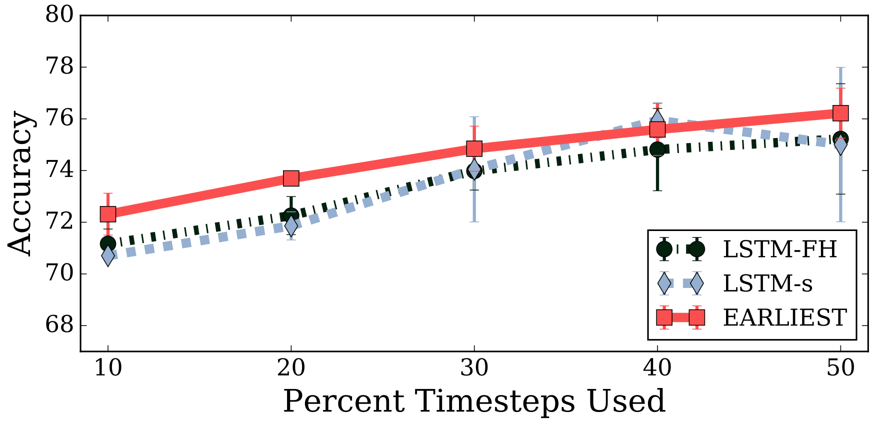
ure 4.3b. As expected, RMSE is poor with both low λ (emphasizing waiting) and high λ (emphasizing halting), and better in between. This indicates that λ controls how effectively EARLIEST halts and captures signals.

4.4.2 Experiments on Real-world Data.

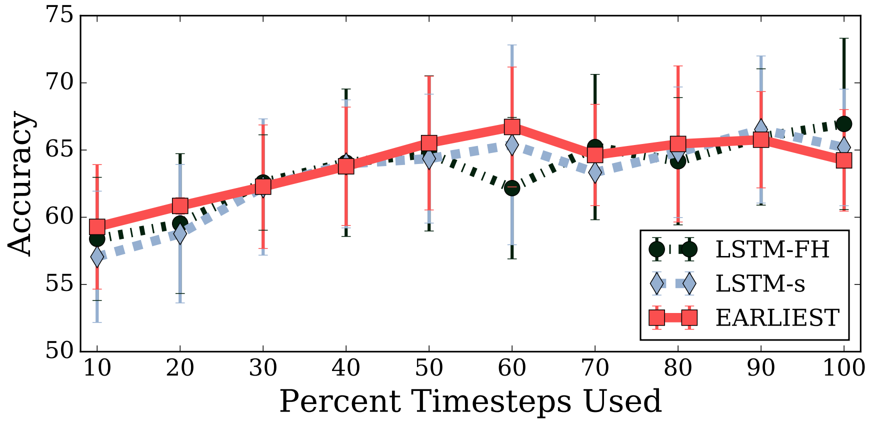
We next present results using real-world datasets `ItalyPowerDemand`, `Mortality`, and `MRSA`, explained in Section 4.1. We compare accuracies and average locations in Figure 4.4. Each point for EARLIEST represents averaged results from λ settings that lead to halting at each timestep. This makes the points comparable with the baseline metrics, but as shown in the synthetic examples, λ allows for a high range of potential average accuracies and average halting-points. In Figures 4.4a and 4.4b, we observe that EARLIEST outperforms baselines for the selected percent timesteps used. However, in Figure 4.4c, we see that EARLIEST falls back to the performance of the baselines. This implies that for some portions of the time series, there is no room for earliness, dependent on the data. From these experiments, we conclude that for many parameter settings EARLIEST has higher classification accuracy than the baselines while using fewer timesteps, and for all settings EARLIEST at least maintains the accuracy of the baselines. However, in general it is not guaranteed that EARLIEST should always win, since room for earliness depends on the underlying data and where signals appear.



(a) `ItalyPowerDemand`



(b) `Mortality`



(c) `MRSA`

Figure 4.4: EARLIEST's performance on real-world data. (a) and (b) show strong performance, (c) shows performance comparable to the baselines. Error bars are standard deviation over 10 experiment repetitions.

Chapter 5

Conclusions

5.1 Summary

In this work, we develop an adaptive model for the early classification of time series on a case-by-case basis that allows for the tuning between emphasis on earliness and accuracy. We demonstrate that reinforcement learning provides a useful framework for adding analyst-controlled tunability to contradictory goals while simultaneously tackling the unsupervised nature of early classification. It directly models multiple objectives of early classification, accuracy and earliness, allowing for their joint optimization despite being conflicting goals. This method is generally applicable to tasks where it may be beneficial to halt an RNN before it reaches the end of a sequence. Our experimental results using both synthetic and real-world datasets indicate that EARLIEST can effectively learn to halt when it observes a signal and wait otherwise, leading to fine-tuned and reactive case-by-case signal-capturing. EARLIEST effectively balances earliness and accuracy through one hyperparameter, allowing for analyst-controlled task-dependent solutions. When classifying a time series, our model learns representations of multivariate time series that can be jointly used to inform early-stopping decisions and classifications.

5.2 Future Work

This work is extensible in many directions. For instance, there are many early classification problems on sequential data that are not time series. For instance, early video classification [11] is an interesting direction, particularly because recurrent neural networks are used very frequently for such problems. Additionally, since EARLIEST is an augmented version of an RNN, EARLIEST can work as any RNN where it would be beneficial to stop the processing of an RNN early, for example generating text earlier during speech-to-text transcription.

The concept of adding a stochastic controller to an RNN also has many potential directions. In this work we give the controller access to breaking the recurrent loop of an RNN, but since RNN's have many moving parts, there are many controls which may be handed off to such a controller, allowing for complex case-by-case conditional computation in RNNs. For example, popular RNN architectures involve passing information directly through portions of large networks without non-linear transformations. However, this typically involves a preset plan for where information is passed. Instead, a controller could learn when and where to pass different information depending on the input data, allowing for more complex use of the recurrent neural network. As another example, there is work on changing the number of hidden layers at each timestep of an RNN. A stochastic controller could be an alternative method for making these decisions, avoiding issues mandating leaky gates and allowing for discrete decisions at each timestep.

Bibliography

[1] Z. Xing, J. Pei, P. S. Yu, and K. Wang, "Extracting interpretable features for early classification on time series," in *SDM*, pp. 247–258, 2011.

[2] Z. Xing, J. Pei, and P. S. Yu, "Early classification on time series," *Knowledge and Information Systems*, vol. 31, no. 1, pp. 105–127, 2012.

[3] Z. Xing, J. Pei, and P. Yu, "Early prediction on time series: A nearest neighbor approach," in *IJCAI*, pp. 1297–1302, 2009.

[4] U. Mori, A. Mendiburu, E. Keogh, and J. A. Lozano, "Reliable early classification of time series based on discriminating the classes over time," *Data Mining and Knowledge Discovery*, vol. 31, no. 1, pp. 233–263, 2017.

[5] Y.-F. Lin, H.-H. Chen, V. S. Tseng, and J. Pei, "Reliable early classification on multivariate time series with numerical and categorical attributes," in *PAKDD*, pp. 199–211, 2015.

[6] G. He, Y. Duan, R. Peng, X. Jing, T. Qian, and L. Wang, "Early classification on multivariate time series," *Neurocomputing*, vol. 149, pp. 777–787, 2015.

[7] M. F. Ghalwash and Z. Obradovic, "Early classification of multivariate temporal observations by extraction of interpretable shapelets," *BMC Bioinformatics*, vol. 13, no. 1, p. 195, 2012.

[8] K. Li, S. Li, and Y. Fu, "Early classification of ongoing observation," in *ICDM*, pp. 310–319, IEEE, 2014.

[9] L. Ye and E. Keogh, "Time series shapelets: a new primitive for data mining," in *ACM SIGKDD*, pp. 947–956, 2009.

[10] Z. Huang, Z. Ye, S. Li, and R. Pan, "Length adaptive recurrent model for text classification," in *ACM CIKM*, pp. 1019–1027, 2017.

[11] S. Ma, L. Sigal, and S. Sclaroff, "Learning activity progression in lstms for activity detection and early detection," in *CVPR*, pp. 1942–1950, IEEE, 2016.

[12] M. Weber, M. Liwicki, D. Stricker, C. Scholzel, and S. Uchida, "Lstm-based early recognition of motion patterns," in *ICPR*, pp. 3552–3557, IEEE, 2014.

[13] J. Schmidhuber, “Self-delimiting neural networks,” arXiv, 2012.

[14] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” arXiv, 2013.

[15] D. Cireşan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” arXiv, 2012.

[16] R. K. Srivastava, K. Greff, and J. Schmidhuber, “Highway networks,” arXiv, 2015.

[17] E. Bengio, P.-L. Bacon, J. Pineau, and D. Precup, “Conditional computation in neural networks for faster models,” *arXiv preprint arXiv:1511.06297*, arXiv, 2015.

[18] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *IEEE ICASSP*, pp. 6645–6649, 2013.

[19] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, “Recurrent neural network based language model,” in *ISCA*, 2010.

[20] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *ICML*, pp. 2048–2057, 2015.

[21] J. L. Elman, “Finding structure in time,” *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[22] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[23] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” arXiv, 2014.

[25] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in *NIPS*, pp. 1057–1063, 2000.

[26] J. Schulman, N. Heess, T. Weber, and P. Abbeel, “Gradient estimation using stochastic computation graphs,” in *NIPS*, pp. 3528–3536, 2015.

[27] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[28] V. Mnih, N. Heess, A. Graves, and K. Kavukcuoglu, “Recurrent models of visual attention,” in *NIPS*, pp. 2204–2212, 2014.

[29] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, “Mimic-iii, a freely accessible critical care database,” *Scientific Data*, vol. 3, 2016.

[30] E. Keogh, L. Wei, X. Xi, S. Lonardi, J. Shieh, and S. Sirowy, “Intelligent icons: Integrating lite-weight data mining and visualization into gui operating systems,” in *IEEE ICDM*, pp. 912–916, 2006.

[31] J. Wiens, E. Horvitz, and J. V. Guttag, “Patient risk stratification for hospital-associated c. diff as a time-series classification task,” in *NIPS*, pp. 467–475, 2012.