Adaptive-Halting Policy Network for Early Classification

Thomas Hartvigsen
Worcester Polytechnic Institute
twhartvigsen@wpi.edu

Cansu Sen
Worcester Polytechnic Institute
csen@wpi.edu

Xiangnan Kong
Worcester Polytechnic Institute
xkong@wpi.edu

Elke Rundensteiner
Worcester Polytechnic Institute
rundenst@wpi.edu

ABSTRACT

Early classification of time series is the prediction of 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 must address this challenge to discover task-dependent trade-offs. We design an early classification model, called EARLIEST, which tackles this multi-objective optimization problem, jointly learning (1) to classify time series and (2) at which timestep to halt and generate this prediction. By learning the objectives together, we achieve a user-controlled balance between these contradictory goals while capturing their natural relationship. Our model consists of the novel pairing of a recurrent discriminator network with a stochastic policy network, with the latter learning a halting-policy as a reinforcement learning task. The learned policy interprets representations generated by the recurrent model and controls its dynamics, sequentially deciding whether or not to request observations from future timesteps. For a rich variety of datasets (four synthetic and three real-world), we demonstrate that EARLIEST consistently outperforms state-of-the-art alternatives in accuracy and earliness while discovering signal locations without supervision.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; **Supervised learning by classification**;

KEYWORDS

Recurrent Neural Network, Reinforcement Learning, Early Classification, Time Series Classification

ACM Reference Format:
Thomas Hartvigsen, Cansu Sen, Xiangnan Kong, and Elke Rundensteiner. 2019. Adaptive-Halting Policy Network for Early Classification. In *The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA.* ACM, Anchorage, AK, USA, August 4-8, 2019 , 10 pages. https://doi.org/10.1145/3292500.3330974

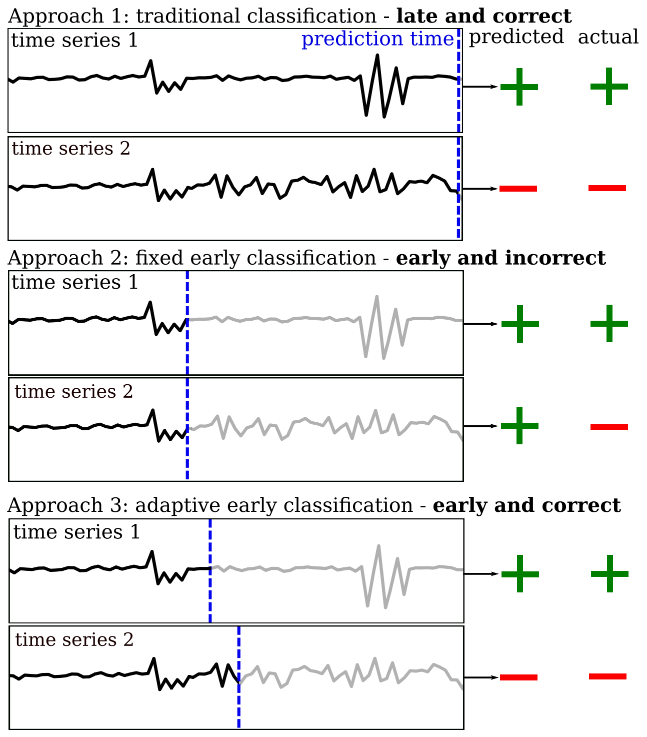

Approach 1: traditional classification - **late and correct**
time series 1
prediction time
predicted
actual
+
+
time series 2
−
−
Approach 2: fixed early classification - **early and incorrect**
time series 1
+
+
time series 2
+
−
Approach 3: adaptive early classification - **early and correct**
time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

Figure 1: Example of three approaches to 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.

1 INTRODUCTION

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

Motivating Examples. Figure 1 depicts an example of the Early Classification of Time Series (ECTS) problem where each time series contains unique signals indicating their respective class labels. Approach 1 illustrates the traditional classification scheme, predicting labels only after the entire time series has been observed. This results in a highly accurate classifier, as it has the opportunity to capture all signals at the cost of providing predictions at the *very end* (indicated by the dashed halting-line). Approach 2 refers to the strict early classification method, choosing a *fixed early timestep* at which to always stop and predict. In this approach, for some time series signals have not arrived yet, while for others decisions are postponed unnecessarily. Approach 3 shows the benefit of *adaptive-early* classification, selecting halting-points on a case-by-case basis (vertical line) thus allowing for early, yet accurate, predictions.

We note that an effective model for time series classification in time-sensitive domains should not only model discriminating signals, but also identify timesteps at which enough information has been observed to reliably (to the requested degree) predict a label. It must also be tunable based on the desired domain-specific emphasis on accuracy versus earliness.

State-of-the-Art. Recently, interest in ECTS has rapidly increased in the literature. Most existing methods exhaustively search among all possible subsequences to identify subsequences that imply class labels [8, 13, 19, 32, 33], called *shapelets* [35]. After shapelet identification, each shapelet candidate is compared to all possible subsequences of testing time series for classification. This strategy does not scale to the multivariate or long time series settings as the number of combinations of variables and subsequences increases exponentially [19]. Some works [23, 24, 31] instead search for time series prefixes, slightly limiting the search-space. However these works compare only prefixes of equal-length to one another, not making use of potential comparisons between different prefix lengths, and still not scaling to the multivariate setting. These search-based approaches are not only rigid, but also do not take advantage of recent success in parameterized sequence-modeling, a promising approach to long high-dimensional time-series. The aforementioned methods have two major limitations. First, they do not allow for direct tunability between accuracy and earliness. Second, they are not end-to-end: the exhaustive shapelet candidate extraction and early classification are independent. This leads to different parts of the models having distinct and contrasting goals (selecting a highly-predictive shapelet has nothing to do with earliness), missing their natural relationship.

Problem Definition. The ECTS problem is to select a timestep in a time series at which enough information has been observed to predict a class label. It is challenging to balance the number of observed timesteps with the expected accuracy. This is because such multi-objective optimization problems involve task-dependent trade-offs. A good solution for one time series may be bad for another, requiring solutions to be highly adaptive and data-driven.

Challenges. Despite the importance of ECTS, several open challenges remain. We summarize three major challenges below:

- *Lack of supervision*: There are no labels indicating where signals occur within a time series; instead the complete time series is typically labeled by its class. Thus quantifying whether or not a prediction *should* be made at a particular timestep is difficult. In short, ECTS contains an inherently unsupervised sub-task within an 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 may cause unnecessary delay and miss precious opportunity to react. The balance is task-specific and an optimal trade-off depends on the particular task and domain. So far, no method allows for direct tuning between these two goals.
- *Multivariate signal evolution*: In multivariate time series, signals indicative of a particular class label may develop at vastly different times across variables, making the identification of one halting point per time series (composed of all variables) harder. There has been little work [8, 13, 19] in the multivariate setting of ECTS, each with limited scalability.

Proposed Method. In this paper, we propose a solution to the aforementioned open challenges called **E**arly and **A**daptive **R**ecurrent **L**abel **EST**imator, or short, EARLIEST. EARLIEST is a novel deep network composed of a recurrent neural network (RNN)-based *Discriminator* with a reinforcement learning-based stochastic *Controller* network. During classification, the recurrent model generates representations of time series one timestep at a time, capturing complex temporal dependencies. The controller interprets these in sequence, learning to parameterize a distribution from which decisions are sampled at each timestep, choosing whether to *stop and predict a label* or *wait and request more data*. Once the controller decides to halt, the discriminator interprets the sequential representation to classify the time series. By rewarding the controller based on the success of the discriminator and tuning the penalization of the controller for late predictions, the controller learns a *halting policy* which controls online halting-point selection. This results in a learned balance between *earliness* and *accuracy* depending on how much the controller is penalized. The size of the penalty is a parameter chosen by a decision-maker according to requirements of the task.

In contrast to traditional sequence-matching ECTS methods, our model-based approach supports flexible earliness-accuracy trade-offs per task using one integrated parameter, being optimized for earliness and accuracy together in one end-to-end model. The resultant solution corresponds to a general deep network model applicable to a rich variety of time-sensitive classification tasks, including video [20, 28] and text [15]. Empirical studies on real-world tasks demonstrate that our approach outperforms baseline methods while providing effective balancing between opposing goals.

Contributions. Our main contributions are summarized below:

- We design a novel ECTS method that handles the unsupervised subproblem of early classification by formulating halting-point selection as a reinforcement learning problem.
- We propose the first dual-optimization solution of the *earliness* and *accuracy* goals by combining them into the objective function for one model. This allows analysts to select a

trade-off depending on the task via only one hyperparameter, aiding interpretability for decision-making.

- We introduce a recurrent neural network-based ECTS method, learning low-dimensional representations of time series and leaning on recent successes in representation learning.
- We evaluate EARLIEST using four synthetic and three real-world time-sensitive classification tasks using public datasets. Results show that our method significantly outperforms state-of-the-art alternatives in both accuracy and earliness.

2 RELATED WORK

To the best of our knowledge, this is the first work supporting task-dependent tunability in ECTS through joint-optimization, supporting both univariate and multivariate data. Our work relates to ECTS methods and conditional computation in neural networks. We briefly discuss them both.

ECTS deals with predicting labels of time series before the time series is fully observed. Many works have been proposed based on modifying traditional distance-based classifiers through exhaustive search instead of parameterized inference [8–10, 13, 31–33]. A well-known approach is to do a similarity search for shapelets, or sub time series indicative of a class [33], then find their earliest occurrences in testing time series. Typically, this involves extracting all 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 qualify as a shapelet [33]. However, at test time, there is only sequence matching, so there is no clear risk computed with predicting at each timestep. Thus these models are not inherently "time-aware", as shapelets do not capture the timing of observations. For example, the same signal may appear at consistently different times in different classes. However, shapelet methods do not consider this to be discriminative. An additional issue with these methods is that the search space for shapelets increases with both the time series length and the number of variables [13]. Hence, prefix-based ECTS methods are another promising approach where sub-time series are extracted with the additional condition that they must begin at the first timestep [23, 24, 31]. However, these existing methods require a large group of independent classifiers, one per time series length. Thus they miss the potential to learn relationships between series of similar lengths. Using one model for all lengths allows for the learning of more complicated relationships.

In most of these methods, 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, missing out on natural connections between these two goals. Their exhaustive search instead of parameterized classification models also limits application in settings with long and/or high numbers of time series. Notably, [23] does joint-optimization, though through a massive collection of classifiers, missing relationships between prefixes of different lengths.

Conditional computation in neural networks deals with learning when to activate different subsets of neural networks, depending on input data [25]. This can reduce the extensive computation required to train a neural network since fewer computations need to be made per example [2]. Additionally, the depth of a neural network has a major impact on performance [5], but selecting the proper network complexity remains empirical and is often more art than science. 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 *in time.* There is one other halting RNN, built for text classification [15], which uses multiple loss functions without the full reinforcement learning setting. [6] uses reinforcement learning for ECTS but does not model temporal dynamics of the time series.

Table 1: Basic Notation

Notation	Description
N	Number of time series in dataset.
M	Variables per time series.
L	Classes for prediction.
T	Number of possible timesteps.
$X_t^{(i)}$	Variables at timestep t for time series i.
$y^{(i)}$	True label for time series i.
$a_t^{(i)}$	Action at timestep t for time series i.
$p_t^{(i)}$	Prob. of halting at timestep t for time series i.
$S_t^{(i)}$	Learned representation for $X_{0,\cdots,t}^{(i)}$.
$\pi_\theta(\cdot)$	Policy, maps states to actions: $\pi_\theta(S_t) = a_t$.
$\tau^{(i)}$	Chosen halting-point for time series i.
	where $i = 1, \cdots, N$ and $t = 1, \cdots, T$.

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 $\hat{y}$ (*i.e.*, $f_\theta(X) \rightarrow \hat{y}$), ultimately generalizing to classify time series not observed during training. 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. Each time series is associated with exactly one label.

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, or $y^{(i)} = 1$, if an adverse event occurs. Otherwise, $X^{(i)}$ belongs to the control group and $y^{(i)} = 0$.

In this work, we model f_θ as a combination of neural networks. However, as opposed to using all T timesteps to generate this prediction, for each time series we seek a timestep $\tau \leq T$ that is both small enough to satisfy a preset requirement for earliness and large

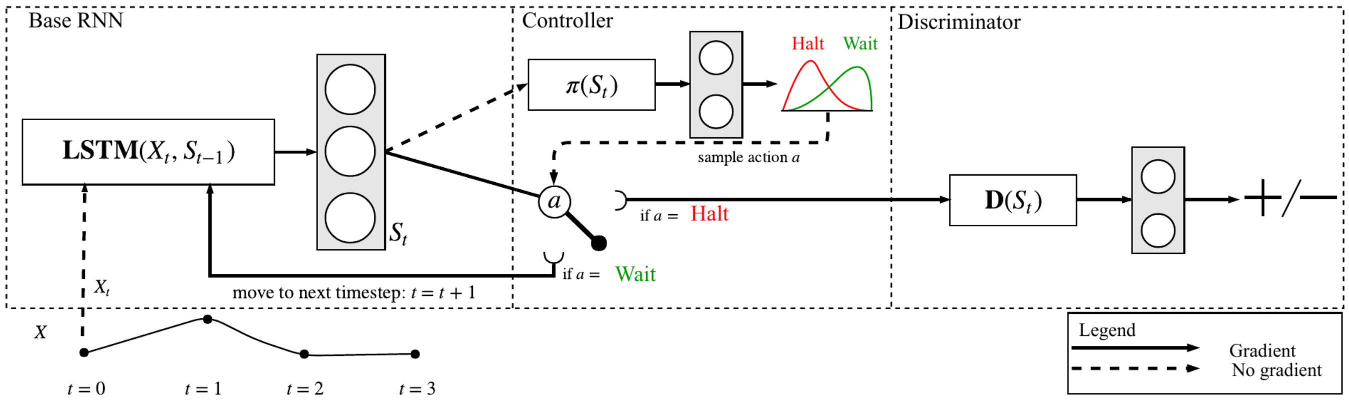


Figure 2: Overview of EARLIEST. Selected action a chooses whether or not to pass S_t to the *Discriminator* or back to the *Base RNN* to process the next timestep. Dashed lines indicate no gradient flow through these paths.

enough to satisfy a requirement for successful classification. We refer to the selected τ as the *halting-point*.

3.2 RNN & LSTM Background

RNNs have emerged as the state-of-the-art for many time series analysis models [12, 21] and other sequence modeling tasks such as sequence generation [11, 34]. Our proposed model builds on RNNs, which construct vector representations for real-valued sequences. 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 to classify the sequence. Empirically, RNNs with Long Short-Term Memory (LSTM) cells [14] are more effective than as they were originally proposed [7] as they preserve information over longer sequences.

3.3 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, generating low-dimensional vector represenations, (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 *Discriminator*. Once the *Controller* chooses to halt, the processing of the current time series is complete. An overview of the EARLIEST architecture is shown in Figure 2, where we display a rolled-up version of the RNN, showing the interaction between each network for each timestep.

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 affects the accuracy of the model depending on where signals are located.

3.3.1 Base Recurrent Neural Network. An RNN augmented with LSTM cells [14] 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*. 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}:

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

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

$$i_t = \sigma(W_i \cdot [S_{t-1}, X_t] + b_i) \tag{2}$$

The updated C_t is then computed as the gated combination of memory state C_{t-1} and current X_t using the *forget* and *input* gates:

$$C_t = f_t \odot C_{t-1} + i_t \odot \eta(W_c \cdot [S_{t-1}, X_t] + b_c) \tag{3}$$

Finally, state representation S_t is computed through an *output* gate shown in Equation 4 operating on a non-linear C_t in Equation 5.

$$o_t = \sigma(W_o \cdot [S_{t-1}, X_t] + b_o) \tag{4}$$

$$S_t = o_t \odot \eta(C_t) \tag{5}$$

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, σ is the sigmoid function, $\cdot$ is the dot product, and $\odot$ represents the hadamard product. 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, it is also possible to use alternative gating-mechanisms such as the Gated Recurrent Unit [4].

3.3.2 *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 arrive from a state, an action is sampled using a learned policy, and a reward is observed according to the selected action's quality. The objective is to optimize long-term rewards according to success of the *Discriminator*. The model is trained by 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. In our experiments, 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, a learned function maps S_t to p_t, where p_t is the probability of halting, computed as

$$\begin{aligned} p_t &= \sigma(W_{ha}S_t + b_{ha}) \\ &= \frac{e^{W_{ha}S_t + b_{ha}}}{e^{W_{ha}S_t + b_{ha}} + 1} \end{aligned} \tag{6}$$

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 $P(a_t = 1) = p_t$. In addition to hidden state S_t, we use current timestep t as additional context to the network when computing 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 $t = T$), t is considered to be the halting point τ. We use ε-greedy action selection to avoid abundant exploitation in the *Controller*: 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 7. During training, ε exponentially decreases from 1 to 0.

$$a_t = \begin{cases} a_t, & \text{with probability } 1 - \epsilon \\ \text{random action}, & \text{with probability } \epsilon \end{cases} \tag{7}$$

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 $R = \sum_{t=0}^{\tau} r_t$.

3.3.3 *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 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.

$$\begin{aligned} \text{P}(Y = i \mid S_t, W_{ho}, b_{ho}) &= \text{softmax}(W_{ho}S_t + b_{ho}) \\ &= \frac{e^{W_{ho}S_t + b_{ho}}}{\sum_j e^{W_{ho}S_t + b_{ho}}} \end{aligned} \tag{8}$$

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

$$\hat{y} = \arg\max_i \text{P}(Y = i \mid S_t, W_{ho}, b_{ho}) \tag{9}$$

3.3.4 *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 14, using stochastic gradient descent (SGD). The *Base RNN* and *Discriminator* are optimized together with respect to cross entropy loss shown in Equation 10 where θ_{bd} indicates parameters of the *Base RNN* and *Discriminator*.

$$J_{bd}(\theta_{bd}) = -(y\log(\hat{y}) + (1 - y)\log(1 - \hat{y})) \tag{10}$$

In contrast to the other networks, in training the *Controller* the goal is to find parameters θ_c that attain the highest expected return:

$$\theta_c^* = \arg\max_{\theta_c} \mathbb{E}[R] \tag{11}$$

Since the *Controller* involves sampling actions, back-propagation does not directly apply, mandating transformation from this raw form to a surrogate loss function [27]. This objective can thus be optimized using gradient descent by taking steps in the direction of $\mathbb{E}[R\nabla \log \pi(S_{0,\cdots,\tau}, a_{0,\ldots,\tau}, r_{0,\cdots,\tau})]$ [26]. The gradient can then be approximated as shown in Equation 12, which can then be minimized to update the parameters of the controller [30].

$$J_c(\theta_c) = -\mathbb{E}\left[R\sum_{t=0}^{\tau}\log\pi(a_t|S_t)\right] \tag{12}$$

However, minimizing $J_c(\theta_c)$ directly leads to gradient estimates that change dramatically across examples, resulting in high variance in policy updates since each example is treated as if in isolation [27]. To handle this, we add a baseline to $J_c(\theta_c)$, similar to [22], so that θ_c is updated based on *how much better the observed reward is* *than average*, resulting in

$$J_c(\theta_c) = -\mathbb{E}\left[\sum_{t=0}^{\tau} \log \pi(a_t|S_t)\left[\sum_{t'=t}^{\tau} \left(R - b_{t'})\right)\right]\right] \quad (13)$$

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.

3.3.5 *Balancing Earliness and Accuracy.* Up to this point, the *Controller*'s only objective is to maximize the performance of the *Discriminator.* To add earliness as a goal, we employ an additional loss term, shown as the last term of our final loss function $J(\theta)$ in Equation 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, controlling earliness directly.

$$J(\theta) = J_{bd}(\theta_{bd}) + \alpha J_c(\theta_c) + \lambda \sum_{t=0}^{\tau} -\log \pi(a_t = 1 \mid S_t) \quad (14)$$

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.

4 EXPERIMENTS

4.1 Datasets

We evaluate our method on four variants of a synthetic dataset and three real-world time-sensitive datasets from different domains.

`SimpleSignal`: The true locations of signals within time series are rarely known. To better understand both how EARLIEST performs in classification and in halting when it sees signals, we create a synthetic dataset. Here, we can record exactly where signals are located. For, each time series is initialized with a zero at each of the T timesteps. Then, for positive examples, we sample a location $t \in \{0, \ldots, T\}$ from a selected distribution and substitute a one at timestep t. Negative examples remain all zeros. The selection of the signal distribution aids us in studying how each method captures the true signal locations in a variety of settings. We use four signal distributions: *uniform, normal, right-skewed*, and *left-skewed.* In this setting, a successful model should halt when a signal is observed (a one) *and* generate the correct prediction. Given access to each full time series, a classifier should achieve perfect accuracy.

`Mortality`: This use case concerns predicting in-hospital mortality and is based on a real-world dataset composed of EHR records from intensive care unit patients in the Beth Israel Deaconess Medical Center (publicly-available MIMIC-III database [16]). These EHR records contain time series of vital signs and microbiology tests. For clinical classification tasks (*e.g.*, diagnosis), early predictions allow clinicians to take actions that directly benefit patient well-being. The task here is to predict which patients will perish during their stay given their multivariate time series of vital signs and lab tests. We extract patients with positive *Mortality Flags*, indicating their deaths and randomly draw an equal number of surviving patients from the rest of the database. This leads to health records from 11,508 patients. We use the five most frequently recorded vital signs as our variables. The fine-grained variables aren't often recorded at the same time, leading to sparse data. To handle this, we compute daily averages for each variable, fill missing values with variable-wise means, and use the first ten days for each series.

`Seizures` [1]: Seizure activity detection from EEG records, this EEG data set from 500 subjects, each of whom had their brain activities recorded via EEG, is used. There are a total of 11,800 178-timestep time series, and the task is to detect which of these time series contains evidence of epileptic seizure activity. Since there are only 2,300 cases of such activity, we down-sample an equal number from the negative class, resulting in a balanced dataset with 4,600 time series. Finally, we center the time series around zero and compute the mean value of every 10-timestep chunk, summarizing each 178 timestep series into 17 final timesteps.

`TwitterBuzz` [17]: To predict buzz events on Twitter, we work with this data set of 77-dimensional labeled time series indicating whether or not a spike in tweets on a particular topic is observed. Starting with over 140,000 timesteps, we compute the mean of every five steps, center the time series around zero, and break the resulting 28,000 timesteps into 2,800 length-ten sequences. We then extract the 1,271 time series containing any buzz events and balance the dataset by randomly selecting an equal number of no-buzz time series, resulting in a balanced dataset of 2,542 time series.

4.2 Compared Methods

We compare EARLIEST's performance to the following algorithms:

- *LSTM-Fixed* [20, 29]. 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-Fixed.*
- *LSTM-s* [20]. Designed for early classification of video, *LSTM-s* can be directly applied to time series. *LSTM-s* encourages early confidence in its predictions by penalizing the model when it becomes less confident similar to an LSTM version of ECTS algorithm ECDIRE [24]. Similar to LSTM-Fixed, this method also uses a fixed pre-selected halting-point.
- *LSTM-Confidence.* Classifiers based on a softmax output assign a probability to each class [3]. For this baseline, we set a threshold α for the minimum confidence of a probabilistic prediction. Once the network surpasses this confidence (*i.e.*, $\max \hat{y} > \alpha$), the model halts and predicts the most likely class. This model adaptively-halts per time series, but since α is not included in the loss function, parameters are not updated with respect to the goal of earliness.

Since other multivariate ECTS algorithms [8, 13] do not directly support multiple trade-offs, our model is not directly comparable.

4.3 Implementation Details

For all datasets, we use an 80% training, 10% validation, and 10% testing split. We use the training set to learn model parameters and the validation set to evaluate the performance of a particular hyperparameter setting (*e.g.*, *nodes-per-layer* or *learning rate*). The training and validation sets are used multiple times to tune hyperparameters, then the testing set is used once to compute the final accuracy of each model. We use a two-layer BaseRNN for these experiments, first learning 10-dimensional embeddings for time series variables, and second learning sequences of 10-dimensional representations, one per time step. We repeat this setup five times and compute averages over these five settings to compute final results. The model is optimized using Adam [18] with a learning rate of $1e^{-4}$. All models are implemented using PyTorch with the code available at `https://github.com/thartvigsen/EARLIEST`.

4.4 Experimental Results

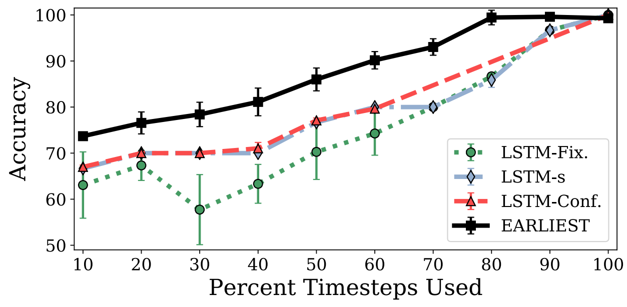
4.4.1 Experiments on Synthetic Data. We first evaluate the performance of EARLIEST in a controllable setting where signal locations are known using the synthetic dataset `SimpleSignal` described in Section 4.1. We evaluate EARLIEST in two ways: by determining how early and accurate EARLIEST is compared to our baselines by controlling the earliness-accuracy trade-off hyperparameter λ; and second, how effectively 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 adaptive-halting. In Figure 3, EARLIEST is run for $\lambda \in [0.0, 0.15]$. 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. 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 3a that for nearly all halting-points, EARLIEST significantly outperforms the baselines, indicating higher accuracy using on average the same information. The only overlap is when all models have observed the entire time series, leading to perfect classification accuracy. Additionally, we report the sensitivity of EARLIEST to parameter λ in Figure 3b. This shows the average accuracy and percent timesteps used for each λ. The smooth down-ward trends indicate that as λ is increased, there is a smooth coverage of all possible halting-points in these time series.

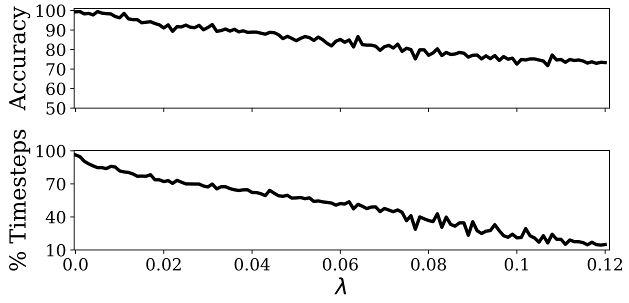
Signal-capturing: Next, EARLIEST should halt when it sees a signal, and wait otherwise. To understand if this is the case, we compute the root mean squared error (RMSE) between EARLIEST's selected halting points and the true distribution of signals, thus quantifying how well EARLIEST halts when it sees a signal.

We use four distributions of positive labels to generate four versions of the `SimpleSignal` data set. 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. We deem the final timestep to be the true signal location for negative examples so we compare positive predicted locations to their known signal locations, measuring true-positive signal-capture.

We show EARLIEST's signal-capturing on `SimpleSignal` with *uniform*, *normal*, *left-skewed*, and *right-skewed* signal distributions in Figure 4, where λ is fixed to be .014, the best performing λ on the *uniform* signal distribution. Additionally, this setting empirically tended to result in a wide variance in predicted locations. We show

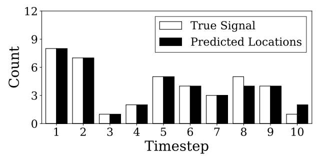


(a) Accuracy and timing

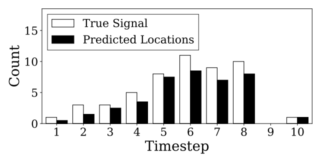


(b) λ coverage in EARLIEST

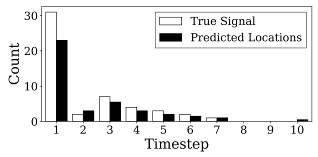
Figure 3: Accuracy and prediction times on synthetic data. (a) EARLIEST makes predictions more accurately and earlier than baselines. (b) λ-tuning leads to smooth halting at each timestep.



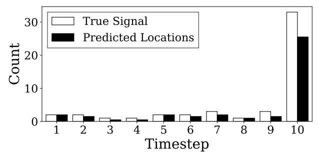
(a) Uniform signals.



(b) Normal signals.



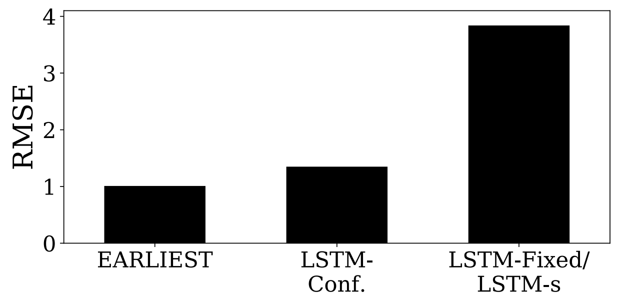
(c) Left-skewed signals.



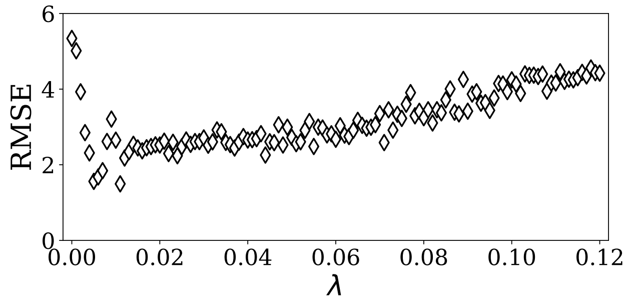
(d) Right-skewed signals.

Figure 4: *True Signals* indicates where signals actually appear in the time series, *Predicted Locations* shows the true-positive halting-points selected by EARLIEST.

the true-positive predictions for each distribution, and the halting-points are averages over all experiment repetitions. In the *Uniform* setting, signals are equally likely to appear at any timestep. In Figure 4a we see that the bars match, indicating that EARLIEST does capture signal locations despite having no access to this information. We see a similar trend for the *Normal* signal distribution ($\mu = 6.0$,



(a) Baseline comparison.



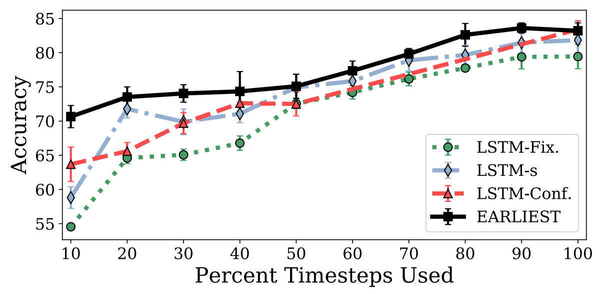
(b) EARLIEST signal-capture sensitivity to λ.

Figure 5: Signal-capturing capabilities on synthetic data. RMSE compares the predicted locations of positive examples with the true signal locations. (a) Minimum RMSE between predicted and actual locations for each model. (b) Parameter analysis for EARLIEST.

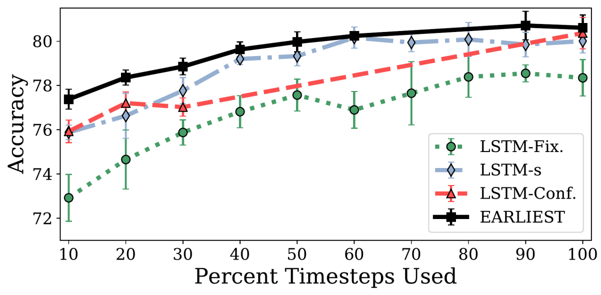
$\sigma = 2.0$) in Figure 4b, though the signal capture is not as exact. The *left-skewed* distribution tests whether or not EARLIEST halts when observing consistently-early signals. In Figure 4c we see this is the tendency of the model, though EARLIEST waits until the end to make one prediction once, missing the signal location. Using the *right-skewed* distribution we test whether or not EARLIEST can wait for long periods of time if it does not observe any signals. In Figure 4d we show this is in fact the case, and the distributions match quite well. These results demonstrate that EARLIEST is capable of flexibly capturing signal locations, halting when signals are observed.

We next compare EARLIEST's signal capture to that of the baselines using the uniform distribution of signal locations. For a fair comparison, we compare the best average performance of each method. In Figure 5a we show that EARLIEST with $\lambda = .014$ dramatically outperforms *LSTM-FH* and *LSTM-s* and is slightly superior to *LSTM-Confidence*, demonstrating that EARLIEST is better at halting when it observes signals. We show the effect of parameter λ on RMSE in Figure 5b. As expected, RMSE is poor with both low λ (emphasizing waiting) and high λ (emphasizing halting), and better in between. This indicates that beyond controlling accuracy, λ also controls how effectively EARLIEST halts and captures signals.

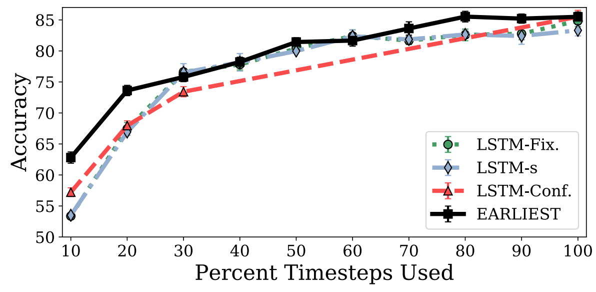
4.4.2 Experiments on Real-world Data. We next present results using real-world datasets `Morality`, `Seizures`, and `TwitterBuzz`. We compare accuracies and average locations in Figure 6. Each point for EARLIEST represents averaged results from λ settings that lead to average halting at each timestep. Overall, we see that EARLIEST consistently performs equal to or better than the compared algorithms at all possible halting points. Optimal halting-points are unknown for these datasets, so we compare accuracy and earliness.



(a) `TwitterBuzz`



(b) `Mortality`



(c) `Seizures`

Figure 6: EARLIEST's performance on real-world data. EARLIEST consistently has equal or better accuracy than the compared methods given on average the same information. Error bars are standard deviation over five repetitions.

For `TwitterBuzz`, we observe in Figure 6a that EARLIEST outperforms the baselines at nearly all timesteps. In the $20 - 60\%$ range, LSTM-s performs equally well. EARLIEST shows an average increase of 2.81% accuracy over the best among the baselines at each timestep with a maximum of up to 12.88%. This indicates that some timesteps significantly benefit from adaptive-halting.

For `Mortality`, we see in Figure 6b that EARLIEST consistently has a higher accuracy than the compared methods. This results in a more modest average increase of .96% over the best baselines with a maximum improvement of 1.91%. Interestingly, despite fine-grained search, no λ led to EARLIEST halting in the $70 - 80\%$ range, possibly due to underlying time series dynamics.

Finally, for Seizures, in Figure 6c we again see a similar trend, the largest difference being that the most improvement over the compared methods occurs early in the time series. This may indicate that signals in this dataset appear early, and after a certain point each model has observed nearly all useful information. In these experiments, EARLIEST shows a mean of 2.61% improvement over the best baselines with a max of 9.77%.

From our experiments on real-world datasets, we conclude that for many parameter settings EARLIEST has higher accuracy than the baselines while using on average fewer timesteps. For all settings, EARLIEST performs equally or better than the baselines. LSTM-s is competitive in many settings, though this method suffers from its requirement for a preset fixed halting-point. For LSTM-Confidence, the resulting halting-points are erratic since the confidence-threshold is set externally to the model. We conclude that benefits of adaptive-halting are also strongly-dependent on the timing of signals. We suspect that the most benefit may be seen with uniformly-distributed and pronounced signals.

5 CONCLUSION

In this work, we have developed EARLIEST, an adaptive model for the early classification of time series. Our neural network-based approach tackles the unsupervised nature of early classification through reinforcement learning. EARLIEST directly models the multiple objectives of early classification, accuracy and earliness, allowing for their joint optimization despite conflicting tendencies. During classification, our model learns representations of multivariate time series that are then used to both inform early-halting decisions and to predict labels. Our experimental results for both synthetic and real-world datasets demonstrate that EARLIEST effectively learns to halt when it observes a signal and wait otherwise, leading to fine-tuned reactive case-by-case signal-capture. EARLIEST effectively balances earliness and accuracy via one hyperparameter, allowing for analyst-controlled task-dependent solutions.

6 ACKNOWLEDGEMENTS

This research was supported by the U.S. Dept. of Education grant P200A150306, Worcester Polytechnic Institute through the Arvid Anderson Fellowship, and the National Science Foundation through grants IIS-1718310, IIS-1815866, CNS-1852498, and CNS-1560229. We also thank the Data Science Research group at WPI.

REFERENCES

[1] Ralph G Andrzejak, Klaus Lehnertz, Florian Mormann, Christoph Rieke, Peter David, and Christian E Elger. 2001. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E* 64, 6 (2001), 061907.
[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. arXiv, 2013. Estimating or propagating gradients through stochastic neurons for conditional computation.
[3] John S Bridle. 1990. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *NeurIPS*.
[4] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*. 1724–1734.
[5] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. arXiv, 2012. Multi-column deep neural networks for image classification.
[6] Martinez Coralie, Guillaume Perrin, E Ramasso, and Rombaut Michèlle. 2018. A deep reinforcement learning approach for earlyclassification of time series. In *EUSIPCO*.
[7] Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14, 2 (1990), 179–211.
[8] Mohamed F Ghalwash and Zoran Obradovic. 2012. Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC Bioinformatics* 13, 1 (2012), 195.
[9] Mohamed F Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. 2013. Extraction of interpretable multivariate patterns for early diagnostics. In *IEEE ICDM*. 201–210.
[10] Mohamed F Ghalwash, Vladan Radosavljevic, and Zoran Obradovic. 2014. Utilizing temporal patterns for estimating uncertainty in interpretable early decision making. In *ACM SIGKDD*. 402–411.
[11] Alex Graves. arXiv, 2013. Generating sequences with recurrent neural networks.
[12] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *IEEE ICASSP*. 6645–6649.
[13] Guoliang He, Yong Duan, Rong Peng, Xiaoyuan Jing, Tieyun Qian, and Lingling Wang. 2015. Early classification on multivariate time series. *Neurocomputing* 149 (2015), 777–787.
[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[15] Zhengjie Huang, Zi Ye, Shuangyin Li, and Rong Pan. 2017. Length Adaptive Recurrent Model for Text Classification. In *ACM CIKM*. 1019–1027.
[16] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. MIMIC-III, a freely accessible critical care database. *Scientific Data* 3 (2016).
[17] François Kawala, Ahlame Douzal-Chouakria, Eric Gaussier, and Eustache Dimert. 2013. Prédictions d'activité dans les réseaux sociaux en ligne. In *4ième conférence sur les modèles et l'analyse des réseaux: Approches mathématiques et informatiques*. 16.
[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *ICLR*.
[19] Yu-Feng Lin, Hsuan-Hsu Chen, Vincent S Tseng, and Jian Pei. 2015. Reliable early classification on multivariate time series with numerical and categorical attributes. In *PAKDD*. 199–211.
[20] Shugao Ma, Leonid Sigal, and Stan Sclaroff. 2016. Learning activity progression in lstms for activity detection and early detection. In *IEEE CVPR*. 1942–1950.
[21] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *ISCA*.
[22] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent models of visual attention. In *NeurIPS*. 2204–2212.
[23] Usue Mori, Alexander Mendiburu, Sanjoy Dasgupta, and Jose A Lozano. 2018. Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE transactions on neural networks and learning systems* 29, 10 (2018), 4569 – 4578.
[24] Usue Mori, Alexander Mendiburu, Eamonn Keogh, and Jose A Lozano. 2017. Reliable early classification of time series based on discriminating the classes over time. *Data Mining and Knowledge Discovery* 31, 1 (2017), 233–263.
[25] Jürgen Schmidhuber. arXiv, 2012. Self-delimiting neural networks.
[26] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. 2015. Gradient estimation using stochastic computation graphs. In *NeurIPS*. 3528–3536.
[27] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*. 1057–1063.
[28] Markus Weber, Marcus Liwicki, Didier Stricker, Christopher Scholzel, and Seiichi Uchida. 2014. Lstm-based early recognition of motion patterns. In *ICPR*. IEEE, 3552–3557.
[29] Jenna Wiens, Eric Horvitz, and John V Guttag. 2012. Patient risk stratification for hospital-associated c. diff as a time-series classification task. In *NeurIPS*. 467–475.
[30] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3-4 (1992), 229–256.
[31] Zhengzheng Xing, Jian Pei, and Phillip Yu. 2009. Early Prediction on Time Series: A Nearest Neighbor Approach. In *IJCAI*. 1297–1302.
[32] Zhengzheng Xing, Jian Pei, and Philip S Yu. 2012. Early classification on time series. *Knowledge and Information Systems* 31, 1 (2012), 105–127.
[33] Zhengzheng Xing, Jian Pei, Philip S Yu, and Ke Wang. 2011. Extracting interpretable features for early classification on time series. In *SDM*. 247–258.
[34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*. 2048–2057.
[35] Lexiang Ye and Eamonn Keogh. 2009. Time series shapelets: a new primitive for data mining. In *ACM SIGKDD*. 947–956.

APPENDIX

Here we describe the details of implementing EARLIEST to aid in reproducibility.

6.1 Forward pass pseudo-code for inference using EARLIEST

The pseudo-code for the forward pass through EARLIEST is shown below, where *BaseRNN()*, *Controller()*, *Discriminator()*, and *Baseline()* are each independently defined networks, but their parameters are all updated together with respect to the final loss. Our models are implemented in PyTorch 1.0 so the pseudo-code roughly emulates PyTorch code.

```
def forward(time_series):
    t = 0
    action = 0
    while action != 1:
        x = time_series[t]
        state = BaseRNN.forward(x, state)
        state_t = concatenate(state, t)
        halt_prob = Controller.forward(state_t)
        action = Bernoulli.sample(halt_prob)
        log_prob = Bernoulli.log_prob(action)
        baseline = Baseline.forward(state_t)
        t += 1
    prediction = Discriminator.forward(state)
    halting_point = t
    return prediction, halting_point
```

6.2 Computing the baseline

The baseline network, mentioned in Sections 3.3.2 and 3.3.4, plays a key role in optimizing policy gradient-based reinforcement learning methods: variance reduction. A standard approach to variance reduction is to add a *baseline*, which is a predicted value that is subtracted from the raw reward, smoothing the reward function and therefore reducing the variance in the observed error signals.

In our implementation, we use a one-layer baseline network with a ReLU nonlinearity (Equation 15) that observes the same information as the controller (hidden state and temporal information) and outputs one real value b_t at each timestep t.

$$b_t = \max(0, W_b[S_t, t] + b_b) \tag{15}$$

We then train this baseline network to approximate the mean of the observed rewards by minimizing the mean squared error between the predicted value b_t and the raw reward R.

6.3 Cohort Extraction in MIMIC III

As described in Section 4.1, the MIMIC III database consists of over 58,000 intensive care unit (ICU) stays. As part of our experiments we perform a common machine learning for healthcare task: *mortality prediction*, or predicting which patients will survive their stays in the ICU of the Beth Israel Deaconess Medical Center in Boston, MA. We begin by selecting all patients who perish while in the ICU (5,754), indicated by *Hospital_Expire_Flag* = 1 in the `ADMISSIONS` table of MIMIC-III. We then randomly sample the same number of surviving patients, totaling 11,508 time series with balanced labels.

Each stay consists of a multivariate time series where each variable is a vital sign, microbiology test, laboratory result, etc. Considering all possible variables that may be observed for a patient leads to an incredibly high-dimensional setting, motivating the need for feature selection. Therefore, we examine a set of commonly-recorded tests and vital signs from tables `MICROBIOLOGYEVENTS`, `CHARTEVENTS`, and `LABEVENTS`, pick the five most frequent variables in this cohort (which as expected end up all being from `CHARTEVENTS`), listed in Table 2 along with the `ITEMID`'s, which identify measurements for their extraction. Since the timestamps across variables are often misaligned, we take hourly averages, impute missing values with variable-wise means, and center each variable around zero before classifying the time series.

Table 2: Variables from the `CHARTEVENTS` table in MIMIC-III

Variable	ITEMID
Systolic Arterial Blood Pressure	51
Motor Response	454
Non-invasive Blood Pressure Mean	456
Temperature (F)	678
Non-invasive Blood Pressure Alarm [Low]	5817