



HAL
open science

Adaptive early classification of temporal sequences using deep reinforcement learning

Coralie Martinez, Emmanuel Ramasso, Guillaume Perrin, Michèle Rombaut

► To cite this version:

Coralie Martinez, Emmanuel Ramasso, Guillaume Perrin, Michèle Rombaut. Adaptive early classification of temporal sequences using deep reinforcement learning. Knowledge-Based Systems, 2019, 10.1016/j.knosys.2019.105290 . hal-02401099v1

HAL Id: hal-02401099

<https://hal.science/hal-02401099v1>

Submitted on 9 Dec 2019 (v1), last revised 24 Nov 2020 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Adaptive early classification of temporal sequences using deep reinforcement learning

Coralie MARTINEZ^{a,*}, Emmanuel RAMASSO^b, Guillaume PERRIN^a,
Michèle ROMBAUT^c

^a*bioMérieux, Marcy l'Etoile, France*

^b*FEMTO-ST Institute, Univ. Bourgogne Franche-Comté, Besançon, France*

^c*Grenoble Institute of Engineering Univ. Grenoble Alpes, GIPSA-Lab, Grenoble, France*

Abstract

In this article, we address the problem of early classification on temporal sequences with adaptive prediction times. We frame early prediction as a sequential decision making problem and we define a partially observable Markov decision process (POMDP) fitting the competitive objectives of classification earliness and accuracy. We solve the POMDP by training an agent for early prediction with reinforcement learning. The agent learns to make adaptive decisions between classifying incomplete sequences now or delaying its prediction to gather more data points. We adapt an existing algorithm for batch and online learning of the agent's action value function with a deep neural network. We propose prioritized sampling, prioritized storing and a specific episode initialization to address the fact that the agent's memory is unbalanced due to (1): all but one of its actions terminate the process and thus (2): actions of classification are rarer than delay actions. In experiments, we compare two definitions of the POMDP based on delay reward shaping vs. reward discounting. We demonstrate that a static naive deep neural network trained to classify at static times is less efficient in terms of accuracy vs. speed than the equivalent network trained with adaptive decision making capabilities. Finally, we show improvements in accuracy induced by our specific adaptation to existing algorithm used in the online learning of the agent's action value function.

Keywords: early classification, adaptive prediction time, reinforcement learning, temporal sequences, Double DQN, trade-off between accuracy vs. speed

*Corresponding author

Email addresses: martinezcoralie.mc@gmail.com (Coralie MARTINEZ),
emmanuel.ramasso@univ-fcomte.fr (Emmanuel RAMASSO),
guillaume.perrin@biomerieux.com (Guillaume PERRIN),
Michele.Rombaut@gipsa-lab.grenoble-inp.fr (Michèle ROMBAUT)

1. Introduction

Early classification of temporal sequences with measurements collected dynamically over time is of prime importance in time-sensitive applications. When each measurement can be costly or when it is critical to act as early as possible, there is a need for methods to make fast online predictions. Taking into consideration that some incomplete sequences can be classified using fewer measurements than more complex ones, the method should make decisions with adaptive prediction time. It should adaptively decide to classify an incoming yet incomplete sequence now or to delay the prediction to gather more measurements. The method should balance its decision between two competitive objectives: classification earliness and accuracy.

1.1. Related work

As opposed to static data, temporal sequences are dynamic data that can be sequentially completed with new measurements. Classification on other types of dynamic data has been proposed by several authors which turned the problem of dynamic data classification as a sequential decision problem.

Formulated as "learning when to stop thinking and do something" in [1], this problem was tackled by reinforcement learning. The authors were interested in algorithms capable of returning a response at any time, which, when working on a problem, can be interrupted. In that case, an "any time algorithm" must return its current best answer. Their approach is policy-gradient-based and uses REINFORCE from [2]. After collecting an observation at a given step k , the stochastic policy determines whether the algorithm should continue to think with a given probability, or to act. In case it is interrupted, the policy thus allows to provide a confidence value in the prediction.

In [3], a Markov decision process (MDP) is formulated for the problem of text classification where it is not always necessary to read an entire document to classify its content. By reinforcement learning using approximate policy iteration, the authors propose a method that either continues reading a document sentence by sentence, or classifies it (using a support vector machine). Their method is shown to better accommodate to small training datasets than standard non-sequential classifiers.

The approach proposed in [3], working on a single feature (the sentence), was extended to multiple features by the same authors in [4]. The key idea is that some data points can easily be classified using few features while others would require more features to achieve an accurate classification. This can be of practical interest in various domains. In medicine for example, online symptom checking for disease diagnosis requires such an algorithm to find key positive symptoms. REFUEL algorithm proposed in [5] is a policy-based method using REINFORCE which encourages a reinforcement learning agent to discover positive symptoms more quickly. The authors incorporated a potential-based reward shaping in order to adapt the reward according to the observations collected by the agent before and after making an action.

The problem of costly feature acquisition in the medical domain was also
45 tackled in [6] who proposed to optimize the trade-off between classification ac-
curacy and the total feature cost using a deep reinforcement learning (DRL)
based on Double Deep-Q-Network (DDQN) algorithm from [7]. The authors
demonstrated the capability of their algorithm to solve binary classification
problems efficiently.

50 The trade-off between classification accuracy and the prediction time is also
of paramount importance in early classification applications. Also called early
prediction, this problem has been solved using sequential decision methods by
various non-DRL approaches in [8, 9, 10, 11, 12, 13].

We proposed in [14] a recent previous work on a DRL approach using online
55 Deep-Q-Network (DQN) algorithm for the early multi-class classification prob-
lem. Compared to standard approaches for early classification, this approach is
an end-to-end learning of both the features in the sequences and the decision
rules. The end-user thus does not need to perform feature engineering. The
simultaneous optimization of both classification accuracy and earliness relies on
60 a trade-off specified by the user in terms of a reward function dedicated to the
problem of early classification.

The framework for early prediction with reinforcement learning we proposed
in [14] makes the agent’s memory unbalanced. Indeed, after each acquisition of
new measurement, the agent can either predict a class label or wait for more
65 data. For a classification decision at time k , the agent collected k measurements
in the sequence and the memory has been filled with $k - 1$ delay actions against
one classification action. The delay action is over-represented. Moreover, since
most actions terminate the acquisition process, it is generally unlikely for the
agent to reach the end of a sequence. Early prediction times are over-represented
70 as well. The unbalanced memory in both prediction times and actions can lead
the agent to learn on sub-optimal experiences and disturb its overall training.

1.2. Contributions

The contributions we detail in the present paper are the following.

(1) We frame early prediction as a POMDP fitting the two competitive ob-
75 jectives of classification earliness and accuracy. We experimentally compare two
definitions of the POMDP based on delay reward shaping vs. reward discount-
ing.

(2) We solve the POMDP by training an agent with both online and batch
reinforcement learning. The latest is particularly interesting for (early) classifi-
80 cation of temporal sequences since datasets are generally finite, but the online
version of the algorithm can be useful to adapt the parameters of the agent’s
policy if more data are collected. We train the agent with DDQN algorithm
from [7] on which we introduce three modifications to cope with the aforemen-
tioned unbalanced memory issue. The modifications are the following: we make
85 use of an adapted prioritized sampling and prioritized storing when performing
experience replay and we simply redefine episode initialization.

We experimentally show that these modifications improve the agent’s training in terms of accuracy vs. speed and make the proposed algorithm more robust to hyper-parameters setting.

90 (3) In experiments, we demonstrate that static naive deep neural networks trained to classify at static times are less efficient in terms of accuracy vs. speed than equivalent networks trained with decision making capabilities on adaptive prediction times.

95 The rest of the paper is organized as follows. Section 2 gives background knowledge of reinforcement learning terminologies and algorithms. In Section 3, we define the problem of early classification. Sections 4 and 5 introduce the method by defining and solving a partially observable Markov decision process dedicated to early prediction. In Section 6, we carry out experimental evaluations on the method. Section 7 concludes the paper.

100 2. Background of deep reinforcement learning

2.1. Reinforcement learning

In reinforcement learning, the objective is to solve a decision making process characterized by an agent interacting in an unknown environment through trial and error. In each state s from the state space \mathcal{S} , the agent can pick some action a in the set of possible actions \mathcal{A} . The choice of action a is dictated by its policy π such that $a = \pi(s)$. As a response, the agent receives a reward $r = R(s, a)$ and moves toward next state $s' = T(s, a)$ with R the reward function from the environment and T its transition model. The interactions $\langle s, a, r, s' \rangle$ between the agent and the environment go on until the agent reaches a terminal state leading to the end of an episode.

110 At all timesteps $t \in \mathbb{N}^+$, the agent seeks to choose actions leading to maximal return defined as the sum of future discounted rewards $\sum_{k=0}^{\infty} \gamma^k r_{t+k}$. $\gamma \in [0, 1]$ is a discount factor valuing immediate rewards rather than future rewards. The optimal policy π^* leads to the maximal return.

State value. The value of a state $s \in \mathcal{S}$ is defined as the expectation of return the agent can hope to get starting from that particular state s and following its policy π .

$$V_{\pi}(s) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right]$$

Action value. The action value (or Q-value) of a state $s \in \mathcal{S}$ conditioned on an action a is defined as the expectation of return the agent can hope to get by picking action a in state s and then following its policy π .

$$Q_{\pi}(s, a) = \mathbb{E}_{\pi} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right]$$

Bellman equation allows to decompose the action value as the sum of immediate reward plus discounted action value of the following state.

$$Q_\pi(s, a) = \mathbb{E}_\pi[r_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) | s_t = s, a_t = a]$$

115 If the optimal action value function defined as $Q_*(s, a) = \max_\pi Q_\pi(s, a)$ is known, then an optimal policy can be inferred by acting greedily over the Q-function such that $\pi_*(s) = \arg \max_a Q_*(s, a)$.

To find an optimal policy, we can use two families of methods: policy-based methods approximate the policy function π directly while value-based methods 120 approximate the action value function $Q_\pi(s, a)$ and act greedily over it to derive the policy.

2.2. Deep-Q-Network

In [15], the authors seek to approximate the optimal action value function Q_* by a deep neural network $Q(s, a, \Theta)$ with parameters Θ . Through a gradient 125 descent on mini-batches of interactions $\{< s, a, r, s' >\}$ and using Bellman equation, the DQN algorithm minimizes the loss function from eq. 1 using two strategies:

$$L(\Theta) = (r + \gamma \arg \max_a Q(s', a, \Theta^-) - Q(s, a, \Theta))^2 \quad (1)$$

- Experience replay allows to sample mini-batches of past interactions $\{< s, a, r, s' >\}$ from a replay memory to perform stochastic gradient 130 descent. Samples within a batch are likely to come from independent or remote interactions further reducing correlations in the neural network updates than the original Q-learning algorithm.
- Q-learning targets are computed with a separate Q-network $Q(s, a, \Theta^-)$ whose parameters Θ^- are updated periodically to remove correlations and 135 improve convergence of the algorithm.

Double Deep-Q-Network. In order to overcome DQN overestimations of the action values, the authors in [7] introduce DDQN algorithm and modify the loss function to optimize in eq. 2.

$$L(\Theta) = (r + \gamma Q(s', \arg \max_a Q(s', a, \Theta), \Theta^-) - Q(s, a, \Theta))^2 \quad (2)$$

3. Problem definition

140 Let $X = (x_1, \dots, x_T) \in \mathbb{R}^{p \times T}$ be a temporal sequence with maximal length $T \in \mathbb{N}^+$. At each timestep $i \in [1, T]$, the observation x_i is a vector of $p \in \mathbb{N}^+$ features. When the temporal sequence is not fully acquired, we say that we observe a partial temporal sequence $X_{:t} = (x_1, \dots, x_t) \in \mathbb{R}^{p \times t}$ with $t \leq T$. We suppose we have a training dataset $\mathcal{D} = \{(X^j, l^j)\}_{j=1..n}$ with n pairs of complete 145 temporal sequences X and their associated class label $l \in \mathcal{L}$, with \mathcal{L} the set of labels.

Classification. A (static) classifier is a mathematical function f_{classif} mapping from a temporal sequence X to its class label l such that $f_{\text{classif}} : \{X\} \rightarrow \mathcal{L}$. The performance of static classifiers over a dataset \mathcal{D} is often measured through the accuracy score: $Acc = \sum_{j=1}^n \mathbb{1}(f_{\text{classif}}(X^j) = l^j)/n$.

Early classification. We define an early classifier as a mathematical function f_{early} mapping from temporal sequence X to class labels l and predicting the optimal earliest timestep $t^* \in [1, T]$ to perform classification, such that $f_{\text{early}} : \{X\} \rightarrow \mathcal{L} \times [1, T]$. The early classifier seeks to optimize the two competing scores of classification accuracy and earliness:

$$t^* = \arg \max_{t \in [1, T]} Acc(f_{\text{early}}(X_{:t}), l) + Earliness(t)$$

These two objectives are often competitive since for two timesteps $t_1, t_2 \in [1, T]$, an earlier timestep $t_1 < t_2$ gets a larger score of *Earliness* while its score of *Acc* can decrease due to the lack of information in X_{t_1} in comparison to X_{t_2} .

4. Early classification as a Partially Observable Markov Decision Process

We defined an early classifier as a model mapping from temporal sequence X to class labels l and predicting the optimal earliest timestep $t^* \in [1, T]$ to perform classification. In real life applications, we do not observe the complete sequence X but rather sequentially collect new observations $x_i \in \mathbb{R}^p$ at each timestep $i \in [1, T]$. We focus on applications which do not seek to directly predict optimal timestep $t^* \in [1, T]$ for classification but rather decide online, at each time step t , to perform classification on the partial sequence $X_{:t}$ or to delay classification in order to get additional observations.

To move closer to this objective, we frame early classification as a sequential decision making problem represented by a POMDP. We define the POMDP by the tuple $\{\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma\}$ where \mathcal{S} is the state space, \mathcal{A} is the action space, T is the transition model, R is the reward function, \mathcal{O} is the observation space and γ is the discount factor. Each element of the tuple is introduced below.

Agent. The mathematical function for early classification that we seek to optimize becomes the policy of an agent which will interact and train within the POMDP.

States. \mathcal{S} is the state space. A state $s \in \mathcal{S}$ is characterized by the tuple $s = (X, l, t)$ with $(X, l) \in \mathcal{D}$ a pair of temporal sequence X and its associated label l from the training dataset and with $t \in [1, T]$ the number of time steps observed in the sequence. Since the objective is to predict class labels $l \in \mathcal{L}$ as early as possible, in real life applications we do not have access to the full state information. The class label and future observations are unknown and the Markov decision process is said to be partially observable. Such models assume that we cannot directly observe the underlying state but instead receive an incomplete or noisy observation of that state.

Observations. \mathcal{O} is the observation space. An observation o of a state $s = (X, l, t)$ is the partial sequence of data points from X collected until time t such that $o = X_{:t}$.

Actions. \mathcal{A} is the action space: $\mathcal{A} = \mathcal{A}_c \cup a_d$, with a_d the action of delaying the prediction and with \mathcal{A}_c the set of classification actions: $\mathcal{A}_c = \mathcal{L}$.

Dynamics. $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition model. In real life applications with early prediction objectives, the acquisition of observations is often costly and has to be shortened as much as possible. Once the system decides to perform classification, observations are no longer collected. The transition model T is defined by:

$$T((X, l, t), a) = \begin{cases} \text{terminal} & \text{if } \{a \in \mathcal{A}_c\} \cup \{a = a_d \cap t = T\} \\ (X, l, t + 1) & \text{if } a = a_d \end{cases}$$

Rewards. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function. Let $R(s, a)$ be the reward for taking action a in state s . Rewards should encode the objective we want the model to reach, specifically earliness and accuracy in the early classification problem.

We choose to reward the classification actions according to the accuracy of the predicted class label. When the predicted class label matches the reference label, we give a positive reward $R((X, l, t), a = l) = +1$. On the contrary when the predicted class label differs from the reference label, we give a negative reward $R((X, l, t), a \neq l) = -1$.

We point out that an objective can be encoded by several reward functions. For a same objective of fast prediction using as few features as possible, the agent is rewarded positively with a score +1 if the classification is correct in [5] while it receives null reward for correction classification and negative rewards for incorrect classifications in [6].

To encode the objective of earliness, the following strategies are possible:

- We could reward the agent based on classification actions only and use a discount factor $\gamma < 1$ to motivate the agent to get early rewards. The reward function is then defined by

$$R((X, l, t), a) \mapsto \begin{cases} +1 & \text{if } a \in \mathcal{A}_c \text{ and } a = l \\ -1 & \text{if } a \in \mathcal{A}_c \text{ and } a \neq l \\ 0 & \text{if } a = a_d \end{cases}$$

- Or we could shape the rewards for delay with a score depending on time. If the rewards for delay are given all at once at the time of classification, the agent will get sparse rewards which are often difficult to train on as explained in [16]. To avoid sparse rewards, the agent will be given negative rewards at each decision of delay instead of a single reward at the end of delay: $R((X, l, t), a_d) = -\lambda \times c(t)$ with $c : [0, T] \rightarrow \mathbb{R}^+$ the cost function of delaying the prediction at time t , a monotonic non-decreasing function

of time. $\lambda \in \mathbb{R}^+$ is a parameter setting the trade-off between the two objectives. The more important earliness is in comparison to accuracy, the larger λ should be. The will to compromise is application-dependent and the user can set λ to his preference.

We want the penalization for delay to take into account the amount of information the agent has collected so far. The idea is that the more observations and knowledge the agent has about the sequence, the worse it is to delay. We want a penalty increasing in time t , in the form of κ^t with $\kappa > 1$. We normalize the reward function for delay so that it is bounded independently of the sequence maximal length T .

The reward function is then defined by:

$$R((X, l, t), a) \mapsto \begin{cases} +1 & \text{if } a \in \mathcal{A}_c \text{ and } a = l \\ -1 & \text{if } a \in \mathcal{A}_c \text{ and } a \neq l \\ -\lambda * \kappa^t / (\kappa^T - 1) & \text{if } a = a_d \end{cases}$$

If available, including domain knowledge into the reward function can guide the agent towards a better or faster learning.

Discount factor. $\gamma \in [0, 1]$ is the discount factor. When $\gamma < 1$, rewards are discounted and more importance is given to immediate rewards. For episodic environments with short horizons, the cumulative reward is finite and γ can be set to 1. Environments for early classification have horizon of size T which is the maximal length of sequences.

4.1. POMDP models

We define two models of the POMDP, based on delay reward shaping vs. reward discounting:

- $M_{shaping} = \{\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma\}$ is a POMDP where delay actions are rewarded negatively over time with $R((X, l, t), a_d) = -\lambda * \kappa^t / (\kappa^T - 1), \forall t \in [1, T]$ and rewards are not discounted with $\gamma = 1$.
- $M_{discount} = \{\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma\}$ is a POMDP where rewards are discounted with $\gamma < 1$. The action of delay is not rewarded and the agent collects rewards (positive or negative) from classification actions only with $R((X, l, t), a_d) = 0, \forall t \in [1, T]$.

4.2. Specificities of the POMDP models

All but one of the actions terminate the episode. As defined above, actions are either to predict a class label $l \in \mathcal{L}$ or to delay prediction: $\mathcal{A} = \mathcal{A}_c \cup a_d$. Since we terminate the acquisition of new observations once the classification is performed, all but one of the actions lead to a terminal state. The probability of reaching time t in an episode tends to zero as t increases:

$$P(s_t \neq \text{terminal}) = \underbrace{P(a_1 = a_d)}_{\leq 1} \underbrace{P(a_2 = a_d)}_{\leq 1} \dots \underbrace{P(a_{t-1} = a_d)}_{\leq 1} = \prod_{j=1}^{t-1} \underbrace{P(a_j = a_d)}_{\leq 1}$$

Actions of classification are the rarest. When the agent classifies at time t , the episode is composed of $t - 1$ actions of delay for one action of classification. This results in getting interactions that are mostly composed of delay action.

5. Learning the action value with a deep neural network

245 The action space being finite and small, we choose to learn the action value function and define the agent’s policy π by acting greedily over the action values. The observation space composed of temporal sequences is continuous and therefore the action value function cannot be represented by a finite table with action values on all pairs of observations and actions.

250 We approximate the action value function $Q(s, a)$ with a deep neural network $Q(o, a, \Theta)$ with parameters Θ defined over the state of observations \mathcal{O} . From the POMDP definition and by approximating the action value with a deep neural network, the method simultaneously learns optimal classification patterns in the sequences and optimal strategic decisions for the time of prediction. The
255 end-to-end learning capabilities of neural networks set the user free from a prior step of feature engineering and definition of prediction rules.

We train the neural network $Q(o, a, \Theta)$ with DDQN algorithm from [7] to find optimal parameters Θ .

5.1. Batch learning

260 Since many real-life applications of the early prediction problem come with a finite training dataset, their underlying POMDPs have a finite number of experiences to train on. Unlike video games traditionally used in DRL and for which the emulator can generate an infinite number of episodes, leading to simultaneous data collection and optimization, some applications cannot generate new
265 interactions with the environment along training. For those applications with a relatively small training dataset, a batch version of DDQN where the data collection is decoupled from training can be used. We present in Algorithm 1 the adaptation of DDQN to early classification in a batch mode. The idea is to first build an exhaustive replay memory with all possible interactions and
270 use prioritized sampling proposed in Section 5.1.1 to cleverly learn from it. The advantage of a batch version of the DDQN is to set the agent free from its traditional exploration-exploitation dilemma, leading to fewer hyper-parameters to tune.

5.1.1. Prioritized sampling

275 DDQN uses a stochastic gradient descent where a mini-batch of interactions is randomly selected from the replay memory to update the neural networks parameters and minimize the loss function from equation 2. A specificity of the POMDP for early classification is the over-representation of the delay action a_d compared to prediction actions \mathcal{A}_c . With DDQN uniform sampling in the
280 replay memory, batches of interactions will be highly unbalanced and the agent will hardly learn from prediction experiences.

We adapt DDQN with a simple strategy where a fraction of interactions within a mini-batch are forced to come from prediction actions and where the sampling is forced to be balanced among different class labels in order to be
 285 robust to unbalanced training datasets.

From a replay memory \mathcal{M} and for each class label $l \in \mathcal{L}$, we sample a random mini-batch of interactions $\{ \langle o, a, r, o' \rangle \} \sim \mathcal{M}$ such that the observation o is associated to a temporal sequence X of class label l , with fraction μ having $a \in \mathcal{A}_c$. $\mu \in [0, 1]$ is the sampling parameter.

Algorithm 1 DDQN algorithm applied to early classification in batch mode

Require: Environment described by a POMDP $\{\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma\}$ as defined in Section 4.1 and corresponding training dataset $\mathcal{D} = \{(X^j, l^j)\}_{j=1..n}$.

Sampling parameter $\mu \in [0, 1]$ and DDQN hyperparameters from [7].

Ensure: Action value function $Q(o, a, \Theta)$ with optimal weights Θ^*

Store all possible experiences in replay memory \mathcal{M} :

for $j = 1 \dots n$ **do**

 Sample a training pair $(X^j, l^j) \sim \mathcal{D}$.

for $t = 1 \dots T$ **do**

 Compute observation $o = X_{:t}^j$

for $a \in \mathcal{A}$ **do**

 Compute reward $r = R((X^j, l^j, t), a)$

 Compute next observation $o' = T((X^j, l^j, t), a)$.

 Store interaction $\langle o, a, r, o' \rangle$ into replay memory \mathcal{M} .

end for

end for

end for

Randomly initialize weights Θ . Set $\Theta^- = \Theta$.

for step = 1 ... M **do**

 Sample mini-batch of interactions $\{ \langle o, a, r, o' \rangle \} \sim \mathcal{M}$ using *prioritized sampling* from Section 5.1.1 with sampling parameter μ .

 Update weights Θ with gradient descent on loss function from eq. 2 computed on the mini-batch $\{ \langle o, a, r, o' \rangle \}$.

 Periodically update $\Theta^- = \Theta$

end for

290 **5.2. Online learning**

Solving early prediction with reinforcement learning can also be performed in an online mode with simultaneous data collection and optimization of the policy. To fit to the early prediction POMDP specificities, we propose in Algorithm 2 an adaptation of DDQN algorithm with a simple episode initialization strategy (Section 5.2.2), prioritized sampling (Section 5.1.1) and prioritized storing (Section 5.2.1).
 295

5.2.1. Prioritized storing

To avoid possible overwriting of the delay action a_d in the replay memory, we propose to allocate a fraction of the memory to prediction actions. With this strategy, delay actions will not be stored with the same importance than prediction actions and will be more often replaced.

5.2.2. Episode initialization

To answer our objective of fast decision making, the agent has little interest in postponing prediction and reaching the end of temporal sequences. Therefore a static episode initialization at time $t = 1$ would cause early prediction times to be over-represented in the replay memory. In Algorithm 2, we adapt DDQN with a specific episode initialization. We start an episode at random time in the temporal sequence to compel the agent to explore and train on all times of the sequence acquisition.

Algorithm 2 DDQN algorithm applied to early classification in online mode

Require: Environment described by a POMDP $\{\mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma\}$ as defined in Section 4.1 and corresponding training dataset $\mathcal{D} = \{(X^j, l^j)\}_{j=1..n}$.

Sampling parameter $\mu \in [0, 1]$ and DDQN hyperparameters from [7].

Ensure: Action value function $Q(o, a, \Theta)$ with optimal weights Θ^*

Randomly initialize weights Θ . Set $\Theta^- = \Theta$. Initialize replay memory \mathcal{M} .

for episode = 1 ... M **do**

Initialize episode observation o_t with *episode initialization* from Section 5.2.2

while episode not *terminated* **do**

The agent receives observation o_t and picks action $a_t = \arg \max_{a \in \mathcal{A}} Q(o_t, a, \Theta)$ with probability ϵ or random action with probability $1 - \epsilon$.

The environment computes reward $r_t = R((X, l, t), a_t)$ and next observation $o_{t+1} = T((X, l, t), a_t)$.

Store interaction $\langle o_t, a_t, r_t, o_{t+1} \rangle$ into replay memory \mathcal{M} according to *prioritized storing* from Section 5.2.1.

Sample mini-batch of interactions $\{\langle o, a, r, o' \rangle\} \sim \mathcal{M}$ according to *prioritized sampling* from Section 5.1.1 with sampling parameter μ .

Update weights Θ with gradient descent on loss function from eq. 2 computed on the mini-batch $\{\langle o, a, r, o' \rangle\}$.

Periodically update $\Theta^- = \Theta$

Increment time $t = t + 1$

end while

end for

310 6. Experimental evaluation

The experimental objectives are threefold: (1) We evaluate the effect of delay reward shaping vs. reward discounting in the definition of the POMDP.

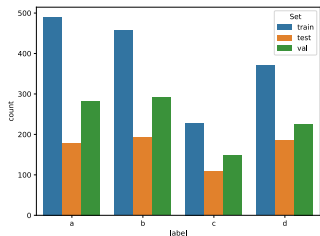


Figure 1: Distribution of class labels a , b , c , and d among the sets of train- validation and testing.

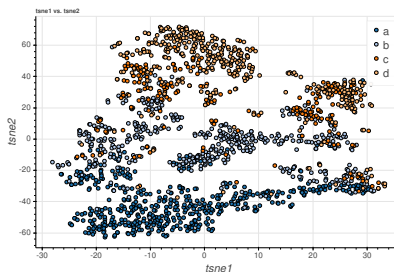


Figure 2: Two-dimensional t-SNE embedding of the temporal sequences from the training set.

(2) We compare early classifiers with adaptive prediction time capabilities to equivalent naive deep neural networks trained to classify at static times. (3) We assess performance gain brought by our specific adaptation to DDQN algorithm.

6.1. Dataset

Data. We conduct experimental evaluations on a dataset collected from a private project carried out by bioMérieux company. Data are multivariate time series derived from living organisms. The 3155 temporal sequences $X = (x_1, \dots, x_T)$ have length $T = 77$ and each data point $x_{i \in [1, T]}$ is a 5-dimensional array. With previous notations from Section 3, $X \in \mathbb{R}^{5 \times 77}$.

This real life example can be generalized to industrial problems with the same objective of early classification on multivariate or univariate temporal sequences. In previous work [14], we compared the reinforcement learning framework to state-of-the-art methods on the UCR archive from [17] which is widely used as benchmark for classification and clustering of time series.

Labels. Sequences are associated to class labels a , b , c , and d depicting four classes of living organisms. Figure 1 gives the distribution of the class labels among the training, validation and testing sets.

t-SNE projection. In Figure 2, we represent the training set with a two-dimensional t-SNE embedding of the (complete) temporal sequences using algorithm from [18]. We observe overlapping clusters of points from different class labels. Samples from class b and c are often mixed among the same clusters of points. This illustrates the complexity of the dataset in which sequences from different classes are very similar due to the biological variability in the dataset.

6.2. Evaluation pipeline

In Section 4, we framed early prediction as a sequential decision making problem defined by a POMDP. We proposed to solve the POMDP by training an agent with reinforcement learning in Section 5. In this section, we introduce metrics and procedures used to train the agent, select optimal policies and compare performance between trainings.

6.2.1. Hyper-parameter setting

In Section 5, the agent is defined by its policy whose model is a deep neural network $Q(o, a, \Theta)$ with weights Θ trained with DDQN algorithm. The deep
345 neural network training depends on a set of hyper-parameters to define. The combinatorial space of the hyper-parameters being too large, we cannot perform an exhaustive search.

To fine-tune the method, we randomly select a set of hyper-parameters in a restricted combinatorial space near optimal parameters presented in [15]. We
350 dedicate one agent per setting of hyper-parameters. Agents are trained separately between all settings.

6.2.2. Training procedure

When trained under supervision (for static classification or regression tasks), deep neural networks are updated until the loss function stops decreasing on
355 the validation set. The selection of the best deep neural network model is also straightforward: the selected model is the one with highest performance on the validation set. When trained with reinforcement, the loss function is based on an approximation of future cumulated rewards and is typically not used to stop the training procedure or to select optimal policies either.

360 Instead, for each hyper-parameter setting of the method, we independently train an agent for a fixed number of episodes in the environment, until it reaches 100000 updates of its deep neural network weights Θ . We simultaneously evaluate the agent of each setting on the validation set every 1000 updates of Θ . Figure 3 reports the evaluations performed during an agent’s training.

6.2.3. Evaluation metrics

365 *Accuracy.* We define the agent accuracy Acc on a dataset $\mathcal{D} = \{(X^j, l^j)\}_{j=1..n}$ as

$$Acc = \sum_{j=1}^n \mathbb{1}(f_{classif}(X^j) = l^j) / n$$

Time of prediction. The prediction time $t_{j,pred}$ of the agent on a sequence $(X^j, l^j) \in \mathcal{D}$ is defined as the earliest time-step for which the action value of a classification action outreaches the action value of delay, such that:

$$t_{j,pred} = \arg \min_{t \in [1, T]} \arg \max_{a \in \mathcal{A}} Q(X_{:t}^j, a) \in \mathcal{A}_c$$

The prediction time t_{pred} of the agent on a dataset \mathcal{D} is the mean of prediction times on all sequences from the dataset, such that:

$$t_{pred} = \sum_{j=1}^n t_{j,pred} / n$$

6.2.4. Optimal policy selection

In [15], the authors evaluate the agent policies over training and select the optimal policy as the one with the highest score of reward. In the special case of early prediction with two competitive objectives optimized one against the other, the optimal policy selection can be application-dependant.

Among all trainings, each one being dedicated to a set of hyper-parameters, we select the policy with highest Acc on the validation set for several ranges of t_{pred} (as illustrated in Figures 4, 6 and 7 where the top-5 optimal policies are represented). We then have as many optimal policy candidates as ranges of t_{pred} considered. Among all candidates, we can then choose the optimal policy as the one satisfying the most our will to compromise between accuracy and speed. The optimal policy reflects the best performance achieve by the method during its fine-tuning.

6.2.5. Training evaluation

Best performance. To assess an agent best performance during its training, we compute max Acc , as illustrated in Figure 3.

Mean performance. To globally assess an agent performance over its entire training, we compute mean Acc and mean t_{pred} over all the agent’s evaluations, that is to say on the 100 policies that were evaluated every 1000 updates of Θ , as illustrated in Figure 3. A large score of mean Acc means that the agent was globally highly accurate all along its training.

Stability. We measure the stability of a training through the variation in Acc and t_{pred} with the standard deviation metric (stdev), as illustrated in Figure 3. A high score of stdev Acc means that the policies evaluated along training were not equally accurate and very unstable.

6.2.6. Methods comparison

Best performance. When comparing several methods, we seek to identify which one gave the best results. Thus we compare the optimal policies results between each method, as illustrated in Figures 4, 6 and 7.

Robustness. We are also interested in assessing the robustness of each method regarding the hyper-parameter setting. We compare each method through the distribution of max Acc , mean Acc , stdev Acc , mean t_{pred} and stdev t_{pred} computed on each training. For each metric, we report the p-values of Mann-Whitney rank statistical tests on the null hypothesis that the two versions are equivalent.

6.3. Experimental comparison between two models of POMDP: reward discounting and delay reward shaping

We carry out an experiment to assess the impact of delay reward shaping vs. rewards discounting in the definition of the POMDP. We compare the two POMDP models $M_{discount}$ and $M_{shaping}$ from Section 4.

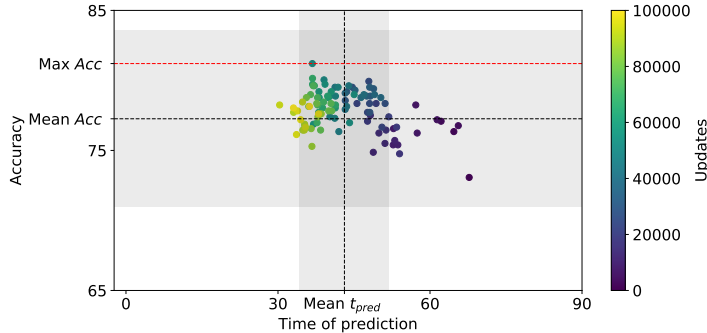


Figure 3: An agent training with 100000 updates of its deep neural network parameters Θ . The agent’s policy is evaluated every 1000 updates on the validation set. Policies performances are represented with dot points in terms of Acc vs. t_{pred} . Dots points are colored according to the updates. The black vertical line (resp. band) gives the agent’s mean (resp. stdev) t_{pred} over training. The black horizontal line (resp. band) gives the agent’s mean (resp. stdev) Acc over training. The red horizontal line gives the agent’s maximal Acc over training.

Experimental setting. We solve each POMDP with DDQN algorithm adapted with prioritized sampling, introduced in Algorithm 1. We perform 50 trainings on each POMDP model (Section 6.2.2) by varying the deep neural network architecture and respective specific hyper-parameters. We vary $\gamma \in [0.3, 1]$ for $M_{discount}$, $\lambda \in \{0.05, 0.1, 0.25, 0.5, 1, 2\}$ and $\kappa \in \{1.06, 1.09, 1.1, 1.2\}$ for $M_{shaping}$. Other shared DDQN hyper-parameters are fine-tuned (Section 6.2.1).

Experimental comparison. To evaluate if both POMDP models achieve comparable best classification accuracy under different trade-offs, we report in Figure 4 the top-5 optimal policies within ranges of prediction times (Section 6.2.4). Experiments show that $M_{shaping}$ results in top-5 policies with higher Acc than $M_{discount}$ under different trade-off of t_{pred} .

We compare the robustness between the two POMDP models by computing metrics from Section 6.2.6 which are shown in Figure 5 and statistically compared in Table 1. Tests allow to reject the null hypothesis that both POMDP models achieve comparable max Acc along training. Figure 5 shows that $M_{shaping}$ reaches higher max Acc . Also, tests on the stdev Acc and stdev t_{pred} lead to the conclusion that $M_{shaping}$ is more variable than $M_{discount}$ during its fine-tuning.

6.4. Experimental comparison between early classifier and naive static classifiers

We seek to experimentally measure the added value of our method for early classification in comparison to static classification. To perform the evaluation, we deactivate the decision making capability of our algorithm, ie. the reinforcement learning part, and train the equivalent naive deep neural network to classify at a list of predefined (static) time steps.

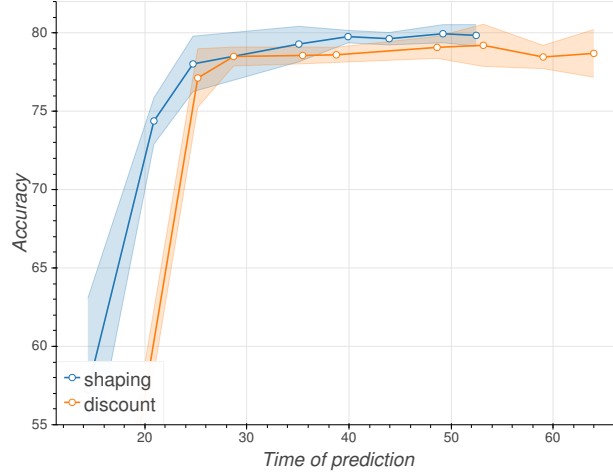


Figure 4: Top-5 policies from $M_{shaping}$ and $M_{discount}$. We select the top-5 policies in Acc on the validation set for several ranges of t_{pred} . We evaluate those policies on the test set. The full line represents mean Acc and the band is the stdev Acc .

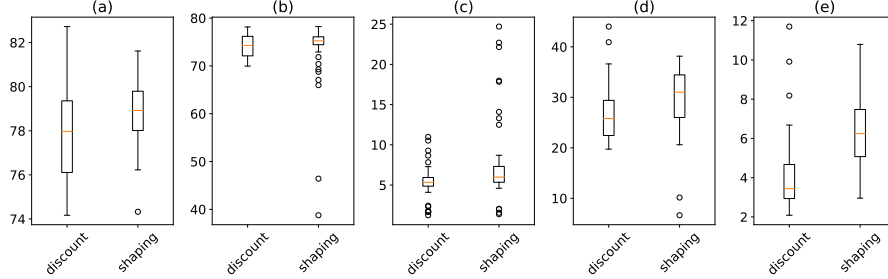


Figure 5: Performance metrics on $M_{shaping}$ and $M_{discount}$ on the validation set. (a) Max Acc . (b) Mean Acc . (c) Stdev Acc . (d) Mean t_{pred} . (e) Stdev t_{pred} .

Performance			Stability	
Max Acc	Mean Acc	Mean t_{pred}	Std Acc	Std t_{pred}
0.0228	0.1962	0.0018	0.0016	1.3162e-8

Table 1: p-values of Mann-Whitney rank tests on the null hypothesis that $M_{shaping}$ and $M_{discount}$ have comparable metric score for each metric from Figure 5.

430 *Experimental setting.*

Early classifier We use experiments from 6.3 on $M_{shaping}$ solved with Algorithm 1 to obtain early classifiers enhanced with decision making capabilities.

Static classifier For regular time steps $t \in [1, T]$, we train equivalent deep neural networks to map between the partial temporal sequences and the class labels. We use the training pairs from dataset $\mathcal{D} = \{(X^j, l^j)\}_{j=1..n}$ and we train deep neural networks as a mathematical function $f_{classif}$ such that $f_{classif} : \{X:t\} \rightarrow \mathcal{L}$. For each regular time step $t \in [1, T]$, the deep neural networks are trained separately until the loss function stops decreasing on the validation set (Section 6.2.2).

440 The neural networks used for both static classification and the agent’s policy are similar except from the output layer. The output layer of the agent’s policy is linear and has an additional neuron for the delay action compared to the static classifier which has as many neurons as class labels and a softmax activation.

Experimental comparison. In Figure 6, we report top-5 policies performance for different ranges of t_{pred} (Section 6.2.4). Both static deep neural network and early classifier have poor *Acc* in early times ($t_{pred} < 20$) due to lack of information in the partial temporal sequences.

Then the early classifier provides top-5 policies with higher *Acc* than static classifiers. The improvement in *Acc* for equivalent t_{pred} is due to the capability of the agent to adapt its classification individually on each temporal sequence. The agent can choose to quickly classify sequences that can easily been categorized or to require more observations on sequences lacking discriminant patterns. The early classifier’s will to individually compromise makes the classification more efficient than static networks using the same amount of observations in all sequences independently of their complexity.

455 Interestingly, we cannot evaluate the early classifier in late prediction times ($t_{pred} > 55$). To reach its objective of fast decision making, the agent did not choose to classify at the end of the sequences and it always provided fastest policies.

460 6.5. Online learning: Experimental evaluation of prioritized sampling, prioritized storing and episode initialization in DDQN algorithm

We carry out an experiment to assess the impact of prioritized sampling (Section 5.1.1), prioritized storing (Section 5.2.1) and episode initialization (sec 5.2.2) when training early classifiers with DDQN algorithm in an online fashion.

465 We consider four versions of DDQN to solve $M_{shaping}$:

- *DDQN-baseline* refers to original DDQN algorithm [7].
- *DDQN-ps* refers to DDQN with prioritized sampling and prioritized storing proposed in sections 5.1.1 and 5.1.1.

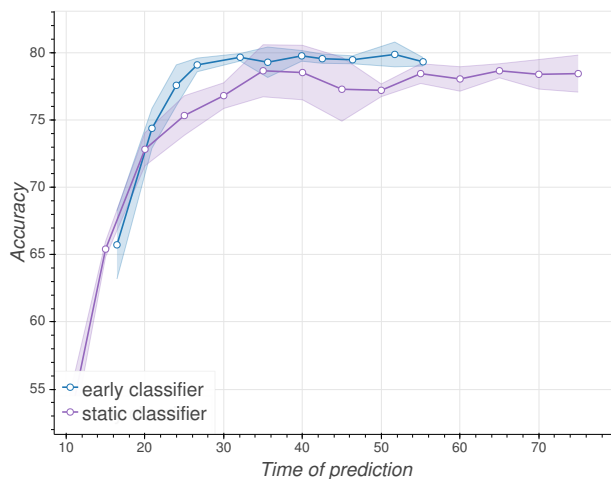


Figure 6: Top-5 policies from $M_{shaping}$ and top-5 static deep neural network classifiers. We select the top-5 policies and classifiers in Acc on the validation set for several ranges of t_{pred} . We evaluate those policies and classifiers on the test set. The full line represents mean Acc and the band is the $stdev Acc$.

- $DDQN-ei$ refers to DDQN with specific episode initialization proposed in Section 5.2.2.
- $DDQN-ps-ei$ refers to DDQN with simultaneously prioritized sampling, prioritized storing and episode initialization as synthesized in Algorithm 2.

Experimental setting. All shared DDQN hyper-parameters are first manually fine-tuned (Section 6.2.1). On each version of DDQN algorithm, we perform 100 trainings (Section 6.2.2). We vary rewards for correct classification $R((X, l, t), a = l) \in \{0, +1\}$ in order to obtain policies with slow decision making and to be able to compare the four versions of DDQN in late prediction times.

Experimental comparison. Top-5 policies (Section 6.2.4) on all four versions of DDQN algorithm are shown in Figure 7. $DDQN-baseline$ top-5 policies are globally the least accurate under all trade-offs of t_{pred} . Top-5 policies with highest Acc for different trade-off of t_{pred} are produced by $DDQN-ei$ and $DDQN-ps-ei$.

The distributions of performance metrics from Section 6.2.5 are shown in Figure 8 and statistically compared in Table 2. Tests show that both $DDQN-ei$ and $DDQN-ps-ei$ improve max Acc and shorten mean t_{pred} over $DDQN-baseline$. Also, mean Acc is significantly higher over trainings from $DDQN-ps$ and $DDQN-ps-ei$ compared to the baseline which leads to the conclusion that those improvements over DDQN are more robust to initial hyper-parameter setting. Policies from $DDQN-ps$ are significantly less variable in terms of accuracy compared to $DDQN-baseline$.

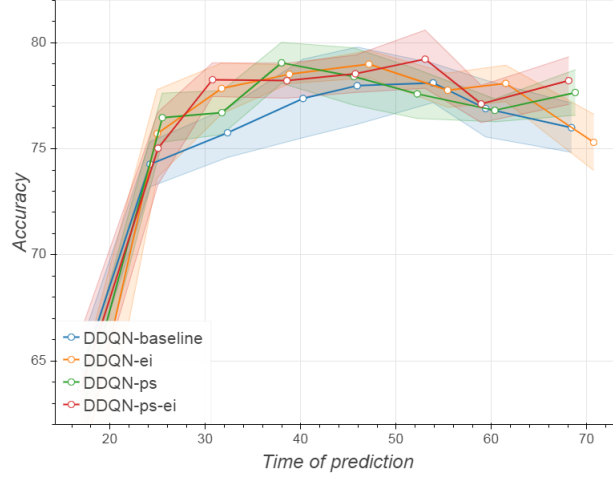


Figure 7: Top-5 policies from *DDQN-baseline*, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei* evaluated on the test set. The full line represents mean accuracy and the band is the accuracy standard deviation.

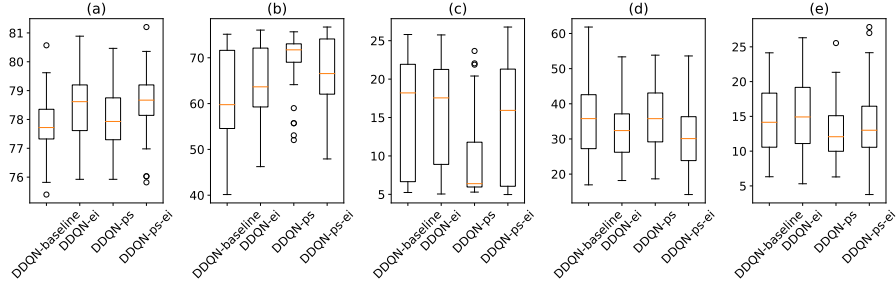


Figure 8: Performance metrics on *DDQN-baseline*, *DDQN-ps*, *DDQN-ei* and *DDQN-ps-ei* on the validation set. (a) Max Acc. (b) Mean Acc. (c) Stdev Acc. (d) Mean t_{pred} . (e) Stdev t_{pred} .

Methods	Performance			Stability	
	Max Acc	Mean Acc	Mean t_{pred}	Stdev Acc	Stdev t_{pred}
<i>DDQN-baseline</i> vs. <i>DDQN-ei</i>	0.0023	0.1467	0.0430	0.8227	0.6270
<i>DDQN-baseline</i> vs. <i>DDQN-ps</i>	0.2464	0.0001	0.8067	1.7212e-5	0.1090
<i>DDQN-baseline</i> vs. <i>DDQN-ps-ei</i>	0.0001	0.0036	0.0286	0.2263	0.5418

Table 2: p-values of Mann-Whitney rank tests on the null hypothesis that *DDQN-baseline* have a metric score comparable to *DDQN-ps* and *DDQN-ps-ei* for each metric from Figure 8.

7. Conclusion

We defined a POMDP to train an agent for early prediction with reinforcement learning. We modeled the agent’s policy by a deep neural network and we adapted the DDQN algorithm in order to address the specificities of the POMDP that could lead to unbalanced memory of the agent if applied without modifications. The validity of the method was shown experimentally on a complex multi-class classification problem on a dataset of multivariate temporal sequences with natural variability. We experimentally demonstrated that:

- Shaping the environment reward signal for delay leads to higher accuracy at all prediction times than sparse discounted rewards.
- Improvements to DDQN online algorithm such as prioritized sampling, prioritized storing and specific episode initialization increase the classification accuracy of the agent while boosting the rapidity of its decision making.
- The method empirically results in an agent with adaptive fast-classification capabilities which achieves higher accuracy performance than an equivalent neural network trained for static classification.

References

- [1] B. Póczos, Y. Abbasi-Yadkori, C. Szepesvári, R. Greiner, N. Sturtevant, Learning when to stop thinking and do something!, in: Proceedings of the 26th Annual International Conference on Machine Learning, ACM, 2009, pp. 825–832.
- [2] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine learning* 8 (3-4) (1992) 229–256.
- [3] G. Dulac-Arnold, L. Denoyer, P. Gallinari, Text classification: A sequential reading approach, in: European Conference on Information Retrieval, Springer, 2011, pp. 411–423.
- [4] G. Dulac-Arnold, L. Denoyer, P. Preux, P. Gallinari, Datum-wise classification: a sequential approach to sparsity, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2011, pp. 375–390.
- [5] Y.-S. Peng, K.-F. Tang, H.-T. Lin, E. Chang, Refuel: Exploring sparse features in deep reinforcement learning for fast disease diagnosis, in: Advances in Neural Information Processing Systems, 2018, pp. 7333–7342.
- [6] J. Janisch, T. Pevný, V. Lisý, Classification with costly features using deep reinforcement learning, in: AAAI Conference on Artificial Intelligence, 2019.

- 530 [7] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.
- [8] Z. Xing, J. Pei, S. Y. Philip, Early prediction on time series: a nearest neighbor approach, in: Twenty-First International Joint Conference on Artificial Intelligence, 2009.
- 535 [9] Z. Xing, J. Pei, P. S. Yu, K. Wang, Extracting interpretable features for early classification on time series, in: Proceedings of the 2011 SIAM International Conference on Data Mining, SIAM, 2011, pp. 247–258.
- [10] G. He, Y. Duan, R. Peng, X. Jing, T. Qian, L. Wang, Early classification on multivariate time series, *Neurocomputing* 149 (2015) 777–787.
- 540 [11] A. Dachraoui, A. Bondu, A. Cornuéjols, Early classification of time series as a non myopic sequential decision making problem, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2015, pp. 433–447.
- [12] W. Wang, C. Chen, W. Wang, P. Rai, L. Carin, Earliness-aware deep convolutional networks for early time series classification, arXiv preprint arXiv:1611.04578.
- 545 [13] T. Santos, R. Kern, A literature survey of early time series classification and deep learning., in: Sami@ iknow, 2016.
- [14] C. Martinez, G. Perrin, E. Ramasso, M. Rombaut, A deep reinforcement learning approach for early classification of time series, in: 2018 26th European Signal Processing Conference (EUSIPCO), IEEE, 2018, pp. 2030–2034.
- 550 [15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, *Nature* 518 (7540) (2015) 529.
- 555 [16] R. S. Sutton, A. G. Barto, et al., Introduction to reinforcement learning, Vol. 135, MIT press Cambridge, 1998.
- [17] H. A. Dau, A. Bagnall, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, E. Keogh, The ucr time series archive, arXiv preprint arXiv:1810.07758.
- 560 [18] L. v. d. Maaten, G. Hinton, Visualizing data using t-sne, *Journal of machine learning research* 9 (Nov) (2008) 2579–2605.