Early classification of temporal sequences with Deep Reinforcement Learning

Coralie Martinez

▶ **To cite this version:**

Coralie Martinez. Early classification of temporal sequences with Deep Reinforcement Learning. Signal and Image processing. Université Grenoble Alpes, 2019. English. NNT : 2019GREAT123 . tel-04276382

HAL Id: tel-04276382
https://theses.hal.science/tel-04276382v1

Submitted on 9 Nov 2023

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.

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **SIGNAL IMAGE PAROLE TELECOMS**

Arrêté ministériel : 25 mai 2016

Présentée par

Coralie MARTINEZ

Thèse dirigée par **Michèle ROMBAUT, UGA**
et codirigée par **Emmanuel RAMASSO, FEMTO-ST Institute**
préparée au sein du **Laboratoire Grenoble Images Parole Signal Automatique**
dans **l'École Doctorale Electronique, Electrotechnique, Automatique, Traitement du Signal (EEATS)**

Classification précoce de séquences temporelles par de l'apprentissage par renforcement profond

Early classification of temporal sequences with Deep Reinforcement Learning

Thèse soutenue à huit clos le **« 4 décembre 2019 »**,
devant le jury composé de :

Monsieur Germain FORESTIER
Professeur des Universités, Université de Haute-Alsace, Rapporteur
Monsieur Marc SEBBAN
Professeur des Universités, Université Jean Monnet, Rapporteur
Madame Latifa OUKHELLOU
Directrice de Recherche, Institut Français des Sciences et Technologies des Transports, de l'Aménagement et des Réseaux, Présidente du jury
Monsieur Christian WOLF
Maitre de conférences, Institut National des Sciences Appliquées de Lyon, Examinateur
Madame Michèle ROMBAUT
Professeur des Universités, Communauté Université Grenoble Alpes, Directrice de thèse
Monsieur Emmanuel RAMASSO
Maitre de conférences, FEMTO-ST Institute, Co-directeur de thèse
Monsieur Guillaume PERRIN
Lead Engineer, bioMérieux, Co-encadrant de thèse

Acknowledgements

First of all, I would like to thank all the people who, from near or far, through their advice or support, helped in the development of this thesis. I am grateful for the help and encouragement I have received over the past three years. Without it, the work would not have been the same, and the thought of getting up every morning to conduct this research study would have been less pleasant.

I especially thank my supervisors. Thank you Michèle for the kindness and availability you showed during this thesis. In particular, I appreciated your many advice and remarks on the writing of my various thesis documents. You always pointed out valuable improvements, which I hope have allowed me to improve my writing of scientific documents, and increase their understanding and pedagogy. I also appreciated your pointed questions about the method, which always led me to step back and get a clearer overview of the work.

Thank you Emmanuel for the energy, curiosity and encouragement that you have brought to this work. Coming to work with you in Besançon was always a source of inspiration and (re)motivation in my work. Your enthusiasm for the work carried out also allowed me to regain my confidence several times. I am grateful for that. Moreover, I want to thank you for sharing your scientific curiosity with me. You always asked precious questions and pointed to interesting articles that contributed to the work of this thesis.

Thank you Guillaume for your supervision, your advice and your practical help in the work done in thesis. You have always been there when needed and I enjoyed being able to count on you. In particular, I enjoyed analyzing the graphs with you, and racking our brains trying to understand what's going on and why the agent does what it does. These reflections greatly contributed to the work of this thesis and allowed to improve it. Thanks also for the jokes, which immediately relaxed the working atmosphere.

Thank you to all three of you for allowing me to conduct this research work, for giving me great confidence and freedom in my work. I'm delighted that I was able to learn with you by sharing our passion and knowledge.

And finally, I can not finish these acknowledgments without mentioning a few names. Thank you Meriem for being my neighbor office, my friend and for all the coffee/tea breaks shared with me. Thank you Magali and Nathalie for being my partners of the drawing club. Thank you Aurélien for the board games moments and the daily encouragement. Thank you Romain and Gael for climbing sessions, tips, and simply for those moments that allowed me to take a break on the work. Thanks to colleagues Maud, Pierre, Philippine, Magali and Meriem for your help, questions, time and encouragement! Your help has been precious and reassured me in my work. Finally, I thank my close relatives, friends and parents who encouraged me from the beginning and who are always present for me.

Abstract

Early classification (EC) of time series is a recent research topic in the field of sequential data analysis [5, 20, 36, 107, 109, 113]. It consists in assigning a label to some data that is sequentially collected with new data points arriving over time, and the prediction of a label has to be made using as few data points as possible in the sequence. The EC problem is of paramount importance for supporting decision-makers in many real-world applications, ranging from process control to fraud detection. It is particularly interesting for applications concerned with the costs induced by the acquisition of data points, or for applications which seek for rapid label prediction in order to take early actions. This is for example the case in the field of health, where it is necessary to provide a medical diagnosis as soon as possible from the sequence of medical observations collected over time. Another example is predictive maintenance with the objective to anticipate the breakdown of a machine from its sensor signals.

In this doctoral work, we developed a new approach for this problem, based on the formulation of a sequential decision-making problem, that is the EC model has to decide between classifying an incomplete sequence or delaying the prediction to collect additional data points. Specifically, we described this problem as a Partially Observable Markov Decision Process noted EC-POMDP. The approach consists in training an EC agent with Deep Reinforcement Learning (DRL) in an environment characterized by the EC-POMDP. The main motivation for this approach was to offer an end-to-end model for EC which is able to simultaneously learn optimal patterns in the sequences for classification and optimal strategic decisions for the time of prediction. Also, the method allows to set the importance of time against accuracy of the classification in the definition of rewards, according to the application and its willingness to make this compromise.

In order to solve the EC-POMDP and model the policy of the EC agent, we applied an existing DRL algorithm, the Double Deep-Q-Network algorithm [100], whose general principle is to update the policy of the agent during training episodes, using a replay memory of past experiences. We showed that the application of the original algorithm to the EC problem lead to imbalanced memory issues which can weaken the training of the agent. Consequently, to cope with those issues and offer a more robust training of the agent, we adapted the algorithm to the EC-POMDP specificities and we introduced strategies of memory management and episode management. In experiments, we showed that these contributions improved the performance of the agent over the original algorithm, and that we were able to train an EC agent which compromised between speed and accuracy, on each sequence individually. We were also able to train EC agents on public datasets for which we have no expertise, showing that the method is applicable to various domains. Finally, we proposed some strategies to interpret the decisions of the agent, validate or reject them. In experiments, we showed how these solutions can help gain insight in the choice of action made by the agent.

Résumé étendu

La classification précoce (CP) des séquences temporelles est un sujet de recherche récent dans le domaine de l'analyse des séquences temporelles [5, 20, 36, 107, 109, 113]. Le problème consiste à assigner une étiquette de classe à des données acquises de manière séquentielle, avec de nouvelles observations arrivant au cours du temps. L'étiquette de classe doit être prédite le plus rapidement possible, c.-à-d. en utilisant un nombre minimal d'observations dans la séquence.

La CP est un problème d'une importance capitale dans de nombreuses applications du monde réel, allant du contrôle des processus à la détection des fraudes. Il est particulièrement intéressant pour les applications où l'acquisition des observations dans la séquence engendre un coût que l'on souhaiterait minimiser. Il est également adapté aux applications qui recherchent une prédiction rapide des étiquettes afin d'entreprendre des actions précoces. C'est par exemple le cas dans le domaine de la santé, où il est nécessaire de fournir un diagnostic médical dans les meilleurs délais à partir de la séquence des observations médicales collectées dans le temps. Un autre exemple est la maintenance prédictive où l'objectif est d'anticiper la panne d'une machine à partir des signaux enregistrés par ses capteurs.

Dans ce travail de doctorat, nous avons développé une nouvelle approche pour ce problème, basée sur la formulation d'un problème de prise de décision séquentielle. Nous définissons un classifieur précoce comme un modèle qui doit décider entre classer une séquence incomplète ou retarder la prédiction pour collecter des observations supplémentaires dans la séquence. Plus précisément, nous avons décrit ce problème comme un processus de décision de Markov partiellement observable noté CP-POMDP. L'approche consiste à former un agent classifieur précoce avec de l'apprentissage par renforcement profond dans un environnement caractérisé par le CP-POMDP. La principale motivation est de proposer un modèle bout-en-bout, c.-à-d. capable d'apprendre simultanément des descripteurs optimaux dans les séquences pour la classification et des décisions stratégiques optimales pour le moment de la prédiction. De plus, la méthode permet à l'utilisateur de régler l'importance du temps par rapport à la précision de la classification suivant son application et le compromis visé, dans la définition des récompenses.

Dans le but de résoudre le CP-POMDP et de trouver la politique optimale de l'agent, nous avons appliqué un algorithme existant d'apprentissage par renforcement profond de la littérature, l'algorithme Double Deep-Q-Network [100]. Le principe général de cet algorithme est de mettre à jour la politique de l'agent durant des épisodes d'entrainement entre l'agent et l'environnement, en utilisant une mémoire de rejeu dans laquelle sont stockées les interactions passées vécues par l'agent.

Nous avons montré que l'application de l'algorithme d'origine au problème de la CP mène à des déséquilibres dans la mémoire de rejeu de l'agent, causant une perte de qualité dans son entraînement. Par conséquent, pour résoudre les problèmes cités ci-avant et permettre un

entraînement plus robuste de l'agent, nous avons adapté l'algorithme en prenant en compte les spécificités du CP-POMDP et en introduisant des stratégies de gestion de la mémoire et des épisodes. Dans les expériences, nous avons montré que ces contributions améliorent les performances de l'agent. Nous avons également montré que cette méthode permet d'entrainer un agent classifieur précoce capable de faire un compromis entre la rapidité de ses prédictions et leur précision, en prenant des décisions individuellement sur chaque séquence. Nous avons démontré que la méthode est applicable à de nombreux domaines, en entrainant l'agent sur des jeux de données publics variés sur lesquels nous n'avions aucune expertise.

Enfin, nous avons proposé des stratégies pour l'interprétation des décisions prises par l'agent, mais aussi pour permettre de les valider ou de les rejeter. Dans les expériences, nous avons montré que ces solutions peuvent aider à mieux comprendre les choix d'actions de l'agent.

Contents

List of acronyms & abbreviations

Acronyms

CNN	Convolutional Neural Network
DDQN	Double Deep Q-Network
DNN	Deep Neural Network
DQN	Deep Q-Network
EC	Early classification
EC-POMDP	Partially Observable Markov Decision Process for Early Classification
HMM	Hidden Markov Model
MDP	Markov Decision Process
MP	Markov Process
MTS	Multivariate time series
NN	Neural Network
PAA	Piecewise Aggregate Approximation
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
SL	Supervised Learning
TS	Time series
UCR	University of California, Riverside
UTS	Univariate time series

Abbreviations

e.g.	exempli gratia ("for the sake of example")
i.e.	id est ("that is to say")

List of symbols

We use the following conventions. Bold uppercase letters represent matrices. Bold lowercase letters are used for vectors. Italic capital letters represent sets.

a	is an action.
a_d	is the delay action.
$\mathcal{A}$	is the set of actions.
$\mathcal{A}_c$	is the set of classification actions.
Acc	is the accuracy metric.
c_{l_j,l_i}	is the cost of predicting label l_i on a sample with true label l_j.
c_{n,l_j,l_i}	is the cost of predicting label l_i when the true label is l_j regarding the n-th individual of $\mathcal{D}$.
$C_{classif}$	is the cost of misclassification.
C_{time}	is the cost of time (or sequence acquisition cost).
C_{total}	is the total cost of time-sensitive classification.
$\mathcal{D}$	is the training dataset.
$\mathcal{D}_{sup}$	is the training dataset for Supervised Learning.
$\mathbf{e}$	is the vector of evidence.
ϵ	is the exploration rate.
$f_{classif}$	is a classifier.
f	is an end-to-end early classifier.
F	is the last convolutional layer in Q_Θ.
f_b	is the b-th convolution filter in F.
g_t	is the return from time step t.
γ	is the discount factor.

h is a set of hyper-parameters.
$\mathcal{H}$ is the combinatorial space of hyper-parameters.

K is the number of distinct labels.
κ is the penalty coefficient.

l is a label.
l_k is the k-th label.
l^n is the label of the n-th sample.
$\widehat{l_{pred}}$ is the predicted label.
$\mathcal{L}$ is the set of labels.
L is a loss function.
λ is the trade-off parameter balancing between classification quality and earliness.

$\mathcal{M}$ is the replay memory.
$\tilde{\mathcal{M}}$ is the exhaustive replay memory.
M is the number of training episodes in DDQN algorithm in online learning.
M_a is the Class Activation Map for action a.
μ is the prioritized sampling parameter.

N is the number of training samples.

o is an observation.
$\mathcal{O}$ is the set of observations.

π is the policy.
π^* is the optimal policy.
π_Θ is the policy associated to Q_Θ.
P is the number of features observed in a sequence $\mathbf{X}$.
P is the state transition model.

Q is the action value function (or Q-function).
Q^* is the optimal action value function.
Q_Θ is the Deep Neural Network for the Q-function with parameters Θ.

r is a scalar reward.

R	is the reward function.
R_d	is the reward function for delay.
$R_{d,shape}$	is the reward function for delay with reward shaping.
$R_{d,discount}$	is the reward function for delay with reward discounting.
R_c	is the reward function for classification.
$R_{c,ins}$	is the reward function for cost-insensitive classification.
$R_{c,sen}$	is the reward function for cost-sensitive classification.
$R_{c,ed-sen}$	is the reward function for example-dependent cost-sensitive classification.
ρ	is the prioritized storing parameter.
s	is a state.
$\mathcal{S}$	is the state space.
t	is a time step.
t_{pred}	is the time of prediction.
T	is the maximal length of a sequence $\mathbf{X}$.
Θ	is the Deep Neural Network parameters.
u	is the DNN uncertainty.
U	is the number of parameters updates in DDQN algorithm in batch learning.
V	is the value function.
$\mathbf{x}_t$	is the vector of data point from $\mathbf{X}$ collected at time step t.
$\mathbf{x}_t^p$	is the p-th feature from $\mathbf{X}$ collected at time step t.
$\mathbf{X}$	is a complete temporal sequence.
$\mathbf{X}^n$	is the complete temporal sequence of the n-th individual.
$\mathbf{X}_{:t}$	is a prefix of temporal sequence $\mathbf{X}$ acquired until time step t.
y	is an action label (defined for Supervised Learning).
$y_{:t}^n$	is the optimal action to take at time step t on the prefix $\mathbf{X}_{:t}^n$.
Ψ	is the emission observation probabilities.

List of Figures

List of Tables

Prelude

The thesis topic is about *early classification of temporal sequences with reinforcement learning methods*. This chapter is dedicated to those of you who frowned excessively or tilted their head from one side while reading the thesis title. We want to demonstrate that the thesis topic is easy to understand. It is neither barbaric nor wacky. It relates to something humans experience everyday, without realizing it. Before we go through a long journey of thoughts, experiments and analysis together, let's demystify the thesis' main components: *temporal sequences*, *classification*, *early classification* and *reinforcement learning*. In the following, we explain and illustrate each one of these components so that you can relate to the thesis topic. For this purpose let us make a few demonstrations.

You daily process temporal sequences.

A temporal sequence is a *sequence of information collected over time*. Temporal sequences are everywhere, recorded or simply processed as they go. We daily collect and process information over time. This information can be of various type. For example, it can be sequences of sounds — when we listen to people, to music, to the news. It can be sequences of words — when we read a book. It can be sequences of images — when we watch a movie or when sitting in a train and watching the landscape passing by. It can be sequences of numbers — when we look at the evolution of the score in a basketball game.

What makes a temporal sequence different from other data is that the *order* of the information makes sense and is critical for our overall understanding. Mixing the items in your shopping list won't have the same impact as mixing the words in a book.

You daily classify temporal sequences.

Temporal sequences can be associated to some *labels*. For example, a music can be associated to its title, its singers, its time, its style. A piece of news can relate to education, politics, media, public health or society. The evolution of the score in a basketball game leads to a winner. To *classify* a temporal sequence means to assign a label. So the last time you listened to a song and *recognized the band*, you classified a temporal sequence.

You daily early classify temporal sequences.

To *early* classify a temporal sequence means to classify *before* you get the full content of the sequence. You early classify temporal sequences when you associate a label to an *incomplete*

data. The last time you listened to a song and recognized the band *before the end of the song*, you *early* classified a temporal sequence.

You daily learn by reinforcement.

To learn by reinforcement means to take some *actions* in an uncertain environment, to *observe* the consequences of your actions and hopefully to *learn* from it. Humans experience reinforcement learning everyday. They seek to behave optimally so that they get the most out of a situation. Their choice of action is often dictated by past experiences. For example, when you learned to ride a bike, you experienced a lot of *feedback* and *rewards* from your decisions. You felt excited when you stayed on the bike for long distances or when you rode towards your target destination, leading to large rewards. You felt frustrated or even hurt when you fell from the bike, leading to poor rewards. All the attempts made you learn the right way to hold your bike, the right speed to put, the right orientation for the handlebar, etc. with always the same natural objective in mind: to get the most out of the situation, which here can mean holding for the longest time without falling and reaching your target destination.

The last and all-in-one argument.

This chapter aimed at explaining each one of the thesis' main components. But actually, there is no need to, because you already are familiar with temporal sequences, classification, early classification and reinforcement learning. And we could have done an all-in-one demonstration. Whether you once played to the Miming Game, to the Pictionary game, or made a sports bet, you used reinforcement learning to early classify temporal sequences during these times. Putting a *label* on your friend's drawing or imitation, predicting the winner: these all are *classification* tasks. Remember, to *classify* means to assign a label to a data. What about the data you used? All the predictions you made were based on *incoming evidence* brought by your friend and by your favorite sports team. At Pictionary, it was the sequence of movements of your friend leading to the progressive elaboration of the drawing. For the sports bet, it was a series of match results. All these data are *temporal sequences*. And the *decisions* you made, the precise time at which you finally took your chance, arise from a succession of past experiences. You might have experienced games when you hurried too much and made bad guesses. Or at the opposite you might have experienced games when you had the right answer in mind but took too long and missed the win for a few seconds. These past experiences made you learn to *compromise* between speed and confidence.

What about the thesis objectives?

The thesis objective is to design algorithms capable of early classifying temporal sequences. Such algorithms have to be able to analyze and find meaningful information in sequences even if they are not complete. They have to be able to associate some labels to the sequences. They have to determine the labels as quickly as possible, using the fewest number of observations. They have to compromise between delaying their predictions to be more confident, or rushing to make fast guesses. Given some sets of examples to learn from, the algorithm has to be able to specialize in all applications and classification tasks.

Chapter 1

Introduction

This thesis investigates the use of Reinforcement Learning to solve the problem of early classification (EC) of temporal sequences. It was conducted through a CIFRE agreement signed between GIPSA-lab, FEMTO-ST and the industrial partner bioMérieux.

1.1 The importance of sequential data analysis

Over the past few years, more and more data have been produced and stored everyday in all business domains. Extracting knowledge from data has thus become one of the hottest research topic of the century. Data can come with different formats. It can be available all at once, as a set of independent data points, each one being attached to quantitative or qualitative features. Or data can be a set of ordered data points, and the order plays an important role. **These are referred to as *sequential data* and are at the core of this thesis.** A sequence is defined as a succession of data points appearing in a specific order. Contrary to other data types, *sequences have complex temporal dependencies and the position of a data point in the sequence matters.* Stock market, sensor data, speech signals, video, text, genes – all are sequential data.

1.1.1 Sequential data taxonomy

In the literature, several appellations are employed and the taxonomy depends on the type, dimension, and indexing of data points in the sequence [108].

According to the indexing. In many application fields, data arrive sequentially over time. The sequences are so called *temporal sequences* or *time series*. The features are indexed by time and the time index can relate to minutes, hours, years, etc. Well-known examples of time series are records of air pollution, electricity consumption, or sales statistics which are often monitored over time. Other application fields collect sequential data that are not indexed by time, such as texts and genomic data which are respectively a succession of words and nucleotides.

According to the dimension. When a single value is collected at each time step (or at each index), the sequence is qualified as *univariate*. Otherwise, when the sequence records a vector of values at each time step, it is *multivariate*.

According to the type. Depending on the type of data points, the sequence is described as *categorical* or *symbolic* when they take value in a finite set (e.g. a DNA sequence with

nucleotide as data points). It is qualified as *continuous* or *numeric* when they are real values.

As an example, a text is an ordered set of words – it is an univariate symbolic sequence. A video an ordered set of frames, each frame being a set of numerical pixels with values between 0 and 255 – it is a multivariate time series.

In this doctoral work, we mainly focus on multivariate time series, but we seek to develop a method that can be generalized to other types of sequential data.

1.1.2 Application fields

Finance, medicine, statistics, econometric, seismology, meteorology, geophysics — all business domains are concerned. Traders daily process financial time series such as stock prices. Cardiologists handle electrocardiograms which are recordings of the electrical activity of the heart over time [40]. Network security engineers analyze sequences of log files from the machines. Moreover, over the last years, streaming data has been gaining an increasing attention both in industry and in academia due to its wide range of applications (e.g. transactions monitoring for fraud detection, sensor measurements for predictive maintenance, social networks feeds for corporate reputation management, mobile applications for geolocation systems, IoT for health monitoring, etc.), and these data are a form of never-ending sequences.

In this doctoral work, we aim to develop a methodology that is not restricted to one application in particular, but rather which can easily be transposed to various applications involving temporal sequences.

1.1.3 Some challenges of sequential data analysis

Because temporal sequences are widely produced in all business domains, sequential data analysis has been largely studied over the past decades [2, 30, 54, 56]. A wide range of methods were proposed in order to extract knowledge from sequences, either for exploratory or predictive approaches such as classification, forecasting, regression or clustering. Despite many years of research on this topic, sequential data analysis still remains a highly topical subject with a lot of challenges – from the representation of these high-dimensional data into smaller representation space (in order to ease data visualization or model learning), to time-sensitive classification, also called *early classification*.

Regardless of the objective, sequences are challenging data for which many questions can arise. First, sequential data are generally collected dynamically over time, with new data points completing the sequence progressively. Knowing that sequential data can therefore be of variable length, and even incomplete, the question is how to handle this *dynamic temporal dimension*.

Second, sequential data are *high-dimensional* data [30]. Each successive data point can be considered as an additional dimension in the representation space. Consequently, a sequence easily becomes very large as the number of time step (or more generally index) increases. Being able to extract meaningful information from these high-dimensional data, also called *feature extraction*, is fundamental in many Machine Learning algorithms. Some of them indeed necessitate to transform the data into a more representative and smaller space.

Finally and as presented before, the data points in sequential data are *highly correlated* due to *temporal dependencies*. This particularity requires specific methods of features extraction that can learn these dependencies in the data points. As a result, many studies in the literature address the problem of temporal sequence analysis and features selection, such as k-gram for DNA and protein sequences [108], shapelets for time series [36, 107, 110, 113], Fourier transform for periodic signals [3], etc.

1.2 A rising need for early classification of temporal sequences

In many applications, sequences can be associated to a label. For example, electrocardiograms can come from a healthy or sick patient. A sequence of log files can correspond to a regular normal use of the system or to a cyber attack. The objective then becomes to model the relationship between the data points evolution over time and its label, known as *sequence classification problem*. The model mapping from the sequence to the label is called a *classifier*. A popular example in the literature is the classification of Electroencephalograph (EEG) signals in order to identify mental states of the patients [63].

For decades, researchers were interested in conventional sequence classification for which the classifier is given the entire sequence before predicting the label. From the 2000s, new motivations appeared. Temporal sequences are high dimensional data with a lot of data points. Generally, data points arrive sequentially over time and their acquisition can be costly. Using fewer data points during the analysis could reduce the *cost of acquisition*. Moreover, it is critical for some applications to infer labels as early as possible so that *early actions* can be initiated. Such *time sensitive applications* are for example related to medical diagnosis for which early treatments have to be adopted [51, 80], disaster prediction to anticipate security measures, intrusion detection to protect against computer attacks, etc.

Generally, motivation for EC lies in the fact that it is not always required to observe an entire sequence in order to predict its label. The meaningful information can be contained early in the sequence and additional data points are valueless for classification. As an example, the authors in [40] argue that "*newborn infants who had abrupt clinical deterioration as a result of sepsis and sepsis-like illness had abnormal HRC and SNAP that worsened over 24 hours before the clinical suspicion of sepsis. A strategy for monitoring these parameters in infants at risk for sepsis and sepsis-like illness might lead to earlier diagnosis and more effective therapy*".

As a conclusion, EC 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, and this is the topic research that will be addressed in this thesis.**

The EC problem is challenging. First, it differs from conventional sequence classification in its necessity to both predict a label and choose the length of sequence that is needed to perform classification. Second, it faces a trade-off. The more data points are used for classification, the more accurate it can be. Nevertheless, the acquisition is more expensive and the prediction is delayed. Consequently EC is an *optimization problem* with *conflicting objectives*. **In this doctoral work, we aim at proposing a method for EC that can handle this trade-off and adjust the relative importance of classification accuracy vs. speed,** independently for each application.

1.3 General objectives of the thesis

In this doctoral work, we aim at solving the EC problem and it is assumed that a training dataset is available, with a set of complete sequences and their label. The objective is to propose an early classifier capable of performing classification using as few data points in a sequence as possible. In the remainder of the document, we refer to an *early classifier* as a solution to the EC problem.

We focus on proposing an *end-to-end* early classifier, i.e. a single model that can directly map an incoming incomplete sequence to the decision to predict, and in particular to a classification label. We therefore discard the strategies which decompose feature extraction from classification and prediction decision, as traditionally addressed in methods from the literature [7, 8, 20, 38, 39, 41, 70, 71, 79, 107, 110].

Industrial application The thesis is related to an industrial application which involves some specificities regarding the classification problem:

- Data are multivariate time series,
- There are more than two labels and they are ordered,
- Labels are unequally represented in the training set,
- Classification errors carry some costs, depending on the true and predicted label. The costs can vary between the samples, depending on additional information on the data.

While the objective is to solve the general problem of EC of temporal sequences, we will develop a method that can address the industrial application specificities.

1.4 Research hypothesis

In this doctoral work, we address the EC problem as a *decision to classify* an incomplete sequence or *a decision to postpone* the prediction. Consequently, **we address EC from a *sequential decision-making* point of view**. The early classifier becomes a model which receives and analyses incoming (yet incomplete) sequences, and sequentially takes some actions: either to classify or to wait for additional data points. Moreover, we frame EC as an *optimization problem* between two competitive objectives: to classify using fewer data points and to make accurate predictions. An early classifier has to make a decision *compromising* between the classification *accuracy* and its *earliness*, and the decision has to be adapted to each sample individually.

We base this doctoral work on the research hypothesis that the EC problem formalized as a sequential decision-making problem can be solved with Deep Reinforcement Learning, a popular discipline for solving complex decision-making problems such as playing video games [42, 44, 47, 68, 73, 100], robot navigation [11], medical diagnosis [80], and autonomous vehicle driving [91].

1.5 Plan of the manuscript

The plan of the manuscript is the following:

- In Chap. 2, we first introduce temporal sequences and specifically multivariate time series. We then define classification and specifically multi-class, imbalanced, ordinal and cost-sensitive classification. Finally, we define the objectives of EC, introduce its trade-off and formalize an optimization problem. Throughout the chapter, we propose a literature review on how existing methods address the data and the problem.

- In Chap. 3, we frame EC as a sequential decision-making problem. We then argue why Supervised Learning is not appropriate for the problem and we show that the latter can be addressed with Reinforcement Learning. Specifically, we define a Partially Observable Markov Decision Process (POMDP) fitting the competitive objectives of classification earliness and accuracy. We propose a strategy to address cost-sensitive learning and to set the trade-off of classification accuracy vs. speed through rewards definition.

- In Chap. 4, we solve the POMDP by training an early classifier agent, an end-to-end Reinforcement Learning agent. The agent has to learn to make adaptive decisions between classifying incomplete sequences now or delaying its prediction to gather more data points. It is trained with a value-based approach, the Double Deep-Q-Network (DDQN) algorithm [100], which aims at approximating the agent's behavior function (its policy) with a Deep Neural Network (DNN) combined with Q-learning [104] and the use of a replay memory. We evaluate the method on a time series benchmark [21] and we show some illustrations on the problem of imbalanced replay memory when training the agent.

- In Chap. 5, we adapt the existing DDQN algorithm to train the agent with both online and batch Reinforcement Learning. To that end, we introduce three strategies in relation to a more robust replay memory management. Specifically, we make use of an adapted *prioritized sampling* and *prioritized storing* when performing experience replay and we redefine *episode initialization*. We then conduct experiments on an industrial dataset composed of multivariate time series. In experiments, we also compare how a static naive DNN trained to classify at static times performs in terms of accuracy vs. speed compared to the equivalent network trained with adaptive decision-making capabilities.

- In Chap. 6, we propose tools to interpret the decisions made by the agent, i.e. its policy. First, we apply the Class Activation Map method [116] to highlight the data points in a test sequence that contribute the most to the agent's predictions. We then provide visualizations on the predictions made by the agent, on test sequences, and show some of their dynamics. Finally and as a perspective for future work, we apply the method introduced in [89] to measure the uncertainty of a DNN prediction in the context of a classification problem, and we adapt it to the EC problem.

- In Chap. 7, we conclude on the doctoral work and give some perspectives for future studies.

Chapter 2

Analysis of the early classification problem of multivariate time series

This chapter aims to define the EC problem raised by this doctoral work and to present main notations. Literature related to the problem is presented throughout the chapter. A major focus of this doctoral work involves studying input data described by multivariate time series (MTS) generated dynamically, which means that input data are progressively enriched by additional multivariate data points. Sec. 2.1 defines the data to be processed. Additionally, this work raises a classification problem described in Sec. 2.2 and which is:

- multi-class (i.e. there are two or more labels),
- class-imbalanced (i.e. some labels in the training set are under-represented),
- ordinal (i.e. labels are ordered),
- example-dependent cost-sensitive (i.e. misclassification involves some costs that depend on the true label, the predicted label and additional information on the data) and
- time-sensitive (i.e. collecting data points in the time series is costly and therefore the amount of acquisitions required to make a prediction has to be minimized).

The classification problem being time-sensitive, the final objective of this work is to early classify input time series (TS) with as few data points as possible while ensuring the quality of the classification. The objectives of EC are presented in Sec. 2.3 which introduces the competitive costs of early and accurate classification, and describes the optimization problem to be solved. In Sec. 2.4, we review related work from the literature on the EC problem on TS.

This chapter focuses on the definition of the EC problem applied to MTS which will be at the core of this thesis. However, we specify that one of the general objectives of the thesis is to be able to apply the developed method on other type of sequential data and on which we might not have expertise.

2.1 Data definition

The problem of the thesis involves temporal sequences as input data. A temporal sequence is a vector of data points indexed by time. It can be of various type depending on the nature of the data (see Chap. 1). In the following of this section, we specifically define MTS which are the type of temporal sequences at the core of the industrial problem. We nevertheless specify that, in order to be able to generalize the method to sequences of images in a future

industrial study, **we will propose in Chap. 3 a method that can be easily generalized to various types of temporal sequences.**

2.1.1 Multivariate time series

Definition 2.1.1 (Time series)**.** Let $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ be a time series with maximal length $T \in \mathbb{N}^+$. At each time step $t \in [1, T]$, the data point $\mathbf{x}_t$ in a sequence $\mathbf{X}$ is a vector of $P \in \mathbb{N}^+$ features, such that $\mathbf{x}_t = (x_t^1 \cdots x_t^P)^\intercal$, with $a^\intercal$ being the transpose of a vector a.

TS are a certain type of data with one or several *time-dependent* features. Contrary to basic machine learning problems on static data, TS are a particular type of data having specific characteristics that need to be taken into consideration. First, TS are data for which the features are observed over time. The *temporal order* of the data points is of prime importance. Data points in these dynamic data are no longer independent: each data point in a TS is dependent on previous data points. Second, when analyzed in its raw representation, that is to say in time domain, each data point in the sequence can be considered as one dimension of the representation space. This often leads to very *large representation* space.

When $P = 1$ the TS is *univariate* (UTS) and there is a single time-dependant feature. When $P > 1$, it is *multivariate* and there are more than one time-dependant features. The P features are observed simultaneously at each time step and they are not necessarily independent. Therefore, MTS are more complex data than UTS because they involve two types of relationships: the temporal relationship between successive data points and the relationship between the different features.

We focus on MTS $\mathbf{X} \in \mathbb{R}^{P \times T}$ such that $P > 1$:

$$\mathbf{X} = \begin{pmatrix} x_1^1 & \cdots & x_T^1 \\ \vdots & \ddots & \vdots \\ x_1^P & \cdots & x_T^P \end{pmatrix} \tag{2.1}$$

Because data from the industrial application are measured on a biological process that evolves over time, we focus on MTS that are non-stationary (see Def. 2.1.2).

Definition 2.1.2 (Stationary and non-stationary time series)**.** A stationary time series is a time series which has statistical properties such as the mean, variance and auto-correlation all constant over time. A non-stationary time series is one for which statistical properties change over time.

In Sec. 2.1.3, we will review existing work on TS analysis. Specifically, we will discuss how to analyze those time-dependent features that can easily reach high dimensions, how to reduce their representation space and how to learn the temporal dependencies. We will discuss the application of standard methods on the data and we will motivate the interest of Deep Neural Networks to learn both temporal and features relationships in the multivariate sequences.

2.1.2 Partial time series

A specificity of the problem is that we do not observe all the data points in the MTS at once. Rather, the problem involves temporal sequences that are generated dynamically, meaning

that the data points arrive sequentially. When the temporal sequence is not fully acquired, we say that we observe a *partial sequence*, or a *prefix* of the sequence (see Def. 2.1.3).

Definition 2.1.3 (Partial time series, or prefix)**.** At time step $t \leq T$, we observe the prefix $\mathbf{X}_{:t} \in \mathbb{R}^{P \times t}$ which is the first t data points in the sequence $\mathbf{X}$, such that $\mathbf{X}_{:t} = (\mathbf{x}_1, \cdots, \mathbf{x}_t)$.

Following the previous definition of $\mathbf{X}$ as a MTS, such that $\mathbf{X} \in \mathbb{R}^{P \times T}$, its prefix $\mathbf{X}_{:t}$ can be written in the matrix form:

$$\mathbf{X}_{:t} = \begin{pmatrix} x_1^1 & \cdots & x_t^1 \\ \vdots & \ddots & \vdots \\ x_1^P & \cdots & x_t^P \end{pmatrix}_{t \leq T} \tag{2.2}$$

Given previous notations, we have $\mathbf{X} = \mathbf{X}_{:T}$ and $\mathbf{X}$ refers to a complete sequence for which all its data points have been collected. In other words, $\mathbf{X}$ has been observed until time T. Fig. 2.1 summarizes the notations introduced in this section.

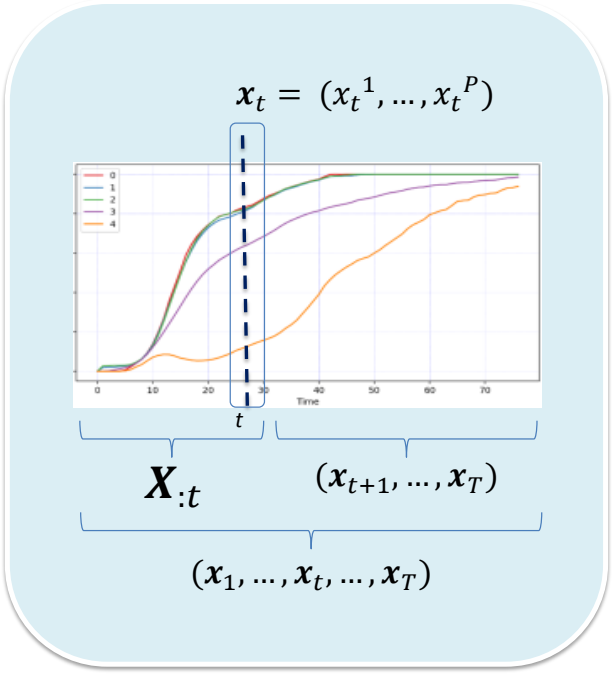

Figure 2.1: **Illustration of a MTS $\mathbf{X} \in \mathbb{R}^{P \times T}$.** In this example, the maximal length is $T = 77$ and the number of features is $P = 5$.

In conclusion, the method of this doctoral work has to analyze input data which have a fixed number of features, P, but which can have a variable temporal dimension depending on the prefixes length $t \in [1, T]$. However, the temporal dimension is bounded by T.

2.1.3 Related work on time series analysis

Generalities TS analysis has been widely studied over the past decades. Many domains collect measures over time leading to sequential data in finance, meteorology, seismology,

econometric, medicine, etc. Literature on TS analysis and more generally on temporal sequence analysis is large and depends the nature of the sequence (univariate/multivariate, stationary/non-stationary, symbolic/numeric, etc.) and on the problem to be solved (classification, forecasting, regression, clustering etc.) [2, 26, 30, 54, 56].

TS are high-dimensional data with time-dependent features (see Sec. 2.1). There is a need for TS representation in order to both condense information contained in the sequences in lower dimensional spaces, and extract temporal dependencies from the data. First, and outside the methods from the literature, a solution is to use expert knowledge and extract some specific hand-crafted features on the TS. These features are application-dependent and often discovered once the data have been widely manipulated. Nevertheless, this approach is based on data knowledge and is not always possible depending on the application. In the following, we conduct a literature review on methods from the literature for TS representation and features extraction.

Frequency approaches Some traditional approaches focus on reflecting global and local characteristics in the data. Discrete Fourier Transform is a popular method in the signal processing community. It seeks to capture the overall shape of a sequence [3] by decomposing it into a set of sine and cosine waves therefore allowing a representation in the frequency domain. The first Fourier coefficients (low frequencies) represent the main characteristics of the sequence while the last coefficients (high frequencies) model details and noise. A major limitation of this approach is that Fourier coefficients are meaningful when the sequences have periodic trends, which is often not the case in many real-life applications. **We remind that the thesis involves non-stationary MTS** and consequently they are not adapted to this transformation.

Another approach is the Discrete Wavelet Transform which seeks to capture both the overall shape of the sequence and the position of its patterns in time [16]. The transformation applies to non-stationary data and is highly flexible due to a large class of wavelets. However, the choice of a wavelet class is essential and requires some data expertise. **In this doctoral work, we seek to develop a method that can be applied to data on which the knowledge of the user is minimal**, which leads us to exclude the use of this transformation.

Nevertheless, we emphasize that applications that could easily benefit from these transformations can transform their input data into these new domains, and then apply the thesis methodology proposed in Chap. 3.

Window-based approaches The literature also offers a lot of research on window-based representation [2, 52, 53, 59, 62]:

- Piecewise Aggregate Approximation (PAA) consists in segmenting a TS into windows of equal size and averaging the subsequences [52], as illustrated in Fig. 2.2 extracted from [62].
- Adaptive Piecewise Constant Approximation is an alternative to PAA for which the windows can have variable length [53].
- Symbolic Aggregate Approximation consists in applying a PAA to the TS and then mapping the segments into symbols in order to be more memory efficient [59].
- As for Piecewise Linear Representation, it approximates a TS by pieces of linear functions [2].

These transformations allow to see the general trend in data and to reduce dimension space

in a very simple way. They are very useful when the sequences are large and when their data points belong to the same ranges of values between consecutive time steps. **In the particular case of the thesis, we focus on sequences with less than a thousand data points and for which significant changes can be observed between two consecutive time steps**, limiting the interest of window-based representation.

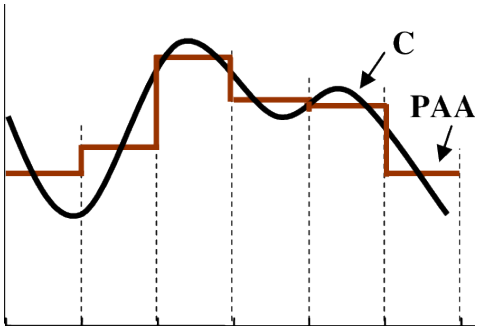

Figure 2.2: **PAA representation of a time series [62].** A time series C (shown in black curve) is represented by the mean values of equal segments (shown in red segments). The dimensionality of C is reduced to 6 segments (bounded by the dashed lines).

Model approaches Other traditional approaches aim to fit a model on TS data and then summarize the data by the model parameters. The choice of a model depends on the nature of the TS. A known example is the Hidden Markov Model (HMM), a type of generative model which supposes that a sequence is made of observed variables generated by hidden variables [83]. HMMs can deal with variable length sequences but they cannot take into account long-term dependencies because of the Markov assumption on the hidden variables. Another example is the Auto-regressive Integrated Moving Averages (ARIMA) model which aims to estimate the trend and seasonality in a TS [50], by making the assumption of data stationary. **The thesis applies to complex non-stationary data with high variability** which can not be summarized by those models.

More recent research papers seek to apply deep learning on TS data [26, 33, 57, 103]. In the past few years, due to the increasing amount of available public data and the development of processing power, Deep Neural Networks (DNNs) have largely been used to fit various types of data through complex non-linear models. For instance, DNNs have been used as generative models to learn data representation [13, 57]. This is the purpose of auto-encoders, a type of DNNs which seek to encode data into a small latent space and then to reconstruct it from its latent representation. Once trained, an auto-encoder can be used to summarize the information in a sample data, and reduce its dimension, by taking its latent representation.

One of the main advantage of DNNs is that they can be used on raw complex data directly, for classification or regression tasks, without the need to reduce the representation space nor pre-compute features. In the particular case of sequences, Recurrent Neural Networks are a popular type of Neural Network that can learn temporal dependencies in its input data, such as Long Short Term Memory networks [35]. Convolutional Neural Networks (CNN) have also been used to learn sequential patterns in the sequences such as [111], in which the authors learn temporal convolutions from raw human activity signals, as illustrated in Fig. 2.3 extracted

from their publication. **In this work, we will propose to learn EC models represented by DNN** (see Chap. 4). **Specifically, we will use CNN to learn both temporal relationships in data points, and relationships between the different features of the multivariate sequence.**

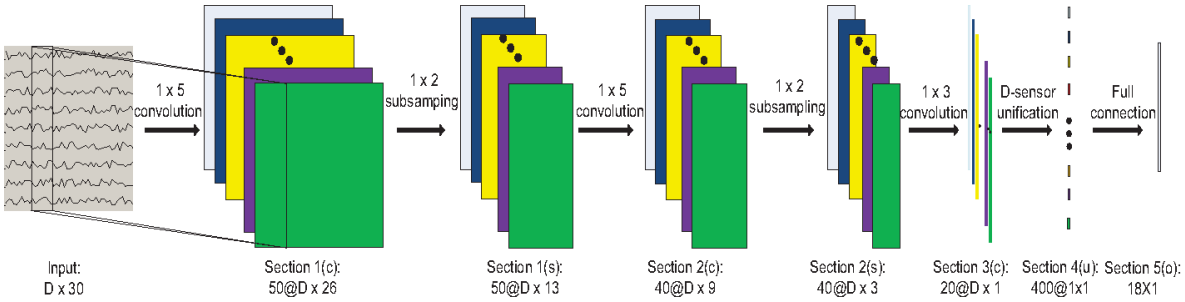

Figure 2.3: **CNN architecture for human activity recognition problems [111]**. Input data to the CNN are multichannel time series signals.

Public benchmarks Due to the popularity of TS data in many application fields, public archives of benchmark datasets are available. The University of California, Riverside (UCR) archive [21] is composed of 128 datasets with UTS. The University of East Anglia (UEA) archive [9] is composed of 30 datasets with MTS. **In this work, we will evaluate the proposed method on a public benchmark and we will demonstrate its applicability to datasets for which we have no expertise.**

2.2 Definition of the classification problem

2.2.1 Generalities about the classification model

In this doctoral work, we address a classification problem on the temporal sequences. We therefore suppose that each temporal sequence $\mathbf{X}$ is associated to a label $l \in \mathcal{L}$, where $\mathcal{L}$ is a finite set of $K \in \mathbb{N}^+$ distinct labels. Let $\mathcal{D}$ be an available training dataset composed of $N \in \mathbb{N}^+$ pairs of (complete) temporal sequences $\mathbf{X}$ and their label l:

$$\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N} \tag{2.3}$$

The sample pair $(\mathbf{X}, l)$ is referred to as an *individual*, *example*, or *sample* from the dataset. $(\mathbf{X}^n, l^n)$ is the n-th individual of $\mathcal{D}$. Furthermore, we distinguish between l supported by a superscript (such as l^n), referring to the label of an individual, and l supported by a subscript (such as l_k), referring to a label from $\mathcal{L}$.

The objective is to build a *classifier* (see Def. 2.2.1) which is able to associate a temporal sequence $\mathbf{X}$ to its label l. Specifically and following Def. 2.2.1, the classification problem is to *learn* a model which best estimates the function $f_{classif}$ from training data.

Definition 2.2.1 (Classifier)**.** A classifier is a mathematical function $f_{classif}$ mapping a temporal sequence $\mathbf{X}$ to its label l, such that $f_{classif} : \{\mathbf{X}\} \rightarrow \mathcal{L}$. We note $\widehat{l}$ the prediction made by the classifier on the temporal sequence $\mathbf{X}$, such that $\widehat{l} = f_{classif}(\mathbf{X})$.

In this document, we refer to $\widehat{l}$ as the *predicted* label on $\mathbf{X}$, while l is the *true* or *reference* label of $\mathbf{X}$. Also, $\widehat{l^n}$ refers to the classifier prediction on the n-th individual of the dataset.

In traditional machine learning, once the model is learned on a training dataset, it is then *evaluated* on a testing dataset to assess its performance on new unknown data. To that end, evaluation metrics can be calculated given ground truth (i.e. the reference labels) and the predicted labels. The selection of a metric for performance evaluation highly depends on the nature of the classification problem: binary vs. multiclass, balanced vs. imbalanced, cost-insensitive vs. cost sensitive. In all cases, it can be useful to look at the confusion matrix (see Def. 2.2.2) on the test dataset to quickly highlight the weaknesses of the classifier. The confusion matrix reports the classifier's performance on each label, as illustrated in Tab. 2.1. It can help to visually detect drop in performance on certain labels. The task is then to carefully derive a metric from the confusion matrix to *summarize* the general performance of the model.

Definition 2.2.2 (Confusion matrix)**.** A confusion matrix is a two-dimensional table where each row represents an instance of the predicted label and each column represents an instance of the true label. Incrementing the confusion matrix at row i and column j means that the classifier predicted label l_i on a sample with true label l_j.

		True label			
		l_1	$\cdots$	l_K	Total
Predicted label	l_1	n_{11}	$\cdots$	n_{K1}	$\sum_{k=1}^{K} n_{k1}$
	$\cdots$	$\cdots$	$\cdots$	$\cdots$	
	l_K	n_{1K}	$\cdots$	n_{KK}	$\sum_{k=1}^{K} n_{kK}$
	Total	$\sum_{k=1}^{K} n_{1k}$		$\sum_{k=1}^{K} n_{Kk}$	N

Table 2.1: **Confusion matrix of a multi-class classification problem with $\mathcal{L} = \{l_1, \cdots, l_K\}$ and N samples in the dataset.** The total number of correct predictions is $\sum_{k=1}^{K} n_{kk}$.

Both model learning and model evaluation have to take into account the nature of the classification problem. In the next section, we will define the specificities of that of the thesis (in relation to the industrial application), namely multi-class, ordinal, imbalanced, example-dependent cost sensitive and time-sensitive classification. In Sec. 2.2.3, we will review existing methods on how to learn and evaluate a classification model coping with those specificities.

2.2.2 Specificities of the classification problem

2.2.2.1 Multi-class classification

The problem involves a *multi-class* set of labels $\mathcal{L}$:

$$\mathcal{L} = \{l_1, ..., l_K\} \text{ such that } K \geq 2 \tag{2.4}$$

When $\mathcal{L}$ is a set of two values, the classifier is *binary*. In the particular scope of the thesis, $\mathcal{L}$ is a set two or more labels and the classifier is *multi-class*.

2.2.2.2 Ordinal classification

The set of labels $\mathcal{L}$ is supposed to be *ordinal*:

$$\mathcal{L} = \{l_1, ..., l_K\} \text{ such that } l_1 < l_2 < ... < l_K \tag{2.5}$$

Having *ordinal labels* means that data associated to a label l_k are closer to data with labels l_{k-1} or l_{k+1}, than data with more distant labels (l_{k-2}, l_{k+2}, etc.).

2.2.2.3 Class-imbalanced classification

One of the thesis specification is that the dataset $\mathcal{D}$ from the application is *imbalanced*, meaning that the labels are not equally represented in the training dataset, as illustrated in Fig. 5.4.

2.2.2.4 Example-dependent cost-sensitive classification

Misclassification costs A classifier $f_{classif}$ makes a classification error, or *misclassifies*, when it predicts a wrong label on a sample pair $(\mathbf{X}, l)$, such that $f_{classif}(\mathbf{X}) \neq l$. Let c_{l_j,l_i} be the cost of predicting label l_i on a sample with true label l_j. The cost matrix (see Def. 2.2.3 and Tab. 2.2) represents the set of classification costs for all possible predictions. The definition of these costs depend on the nature of the classification problem, namely cost-insensitive (see Def. 2.2.4) or cost-sensitive (see Def. 2.2.5).

Definition 2.2.3 (Cost matrix)**.** A cost matrix is a two-dimensional table where each row represents an instance of the predicted label and each column represents an instance of the true label. The value c_{l_j,l_i} at row i and column j represents the cost of predicting label l_i on a sample with true label l_j.

		True label		
		l_1	$\cdots$	l_K
Predicted label	l_1	0	$\cdots$	c_{l_K,l_1}
	$\vdots$	$\cdots$	0	$\cdots$
	l_K	c_{l_1,l_K}	$\cdots$	0

Table 2.2: **Cost matrix of a multi-class classifier with $\mathcal{L} = \{l_1, \cdots, l_K\}$.**

Definition 2.2.4 (Cost-insensitive classification)**.** In a cost-insensitive learning context, all types of classification errors are assumed to be equally important:

$$\forall l_i, l_j \in \mathcal{L}, c_{l_j,l_i} = \begin{cases} 1 & \text{if } l_i \neq l_j \\ 0 & \text{if } l_i = l_j \end{cases}$$

Definition 2.2.5 (Cost-sensitive classification)**.** In a cost-sensitive learning context, the type of errors made by the classifier is more or less penalized depending on the label:

$$\exists l_i, l_j, l_k \in \mathcal{L}, c_{l_j,l_i} \neq c_{l_j,l_k} \tag{2.6}$$

In relation to the industrial application, the thesis problem involves a cost-sensitive classification problem. An example of cost-sensitive application is given in [61] with cancer prediction in medical diagnosis. The classification error of declaring a patient with cancer as a healthy patient is far more serious than rising a false alarm. Because of a missed opportunity for treatment, the cancer patient risk dying and therefore the "cost" induced by this classification error is large. Other examples of cost-sensitive classification problems are those dealing with class-imbalanced datasets and ordinal labels. As with the cancer prediction example, cost-sensitive classification problems can intuitively imply a rank between the classification errors. However, deducing the true cost of an error is not always straightforward.

As part of this thesis work, the classification problem addresses an ordinal set of labels. We know that $l_1 < l_2 < ... < l_K$, meaning that label l_1 is closer to label l_2 than label l_K. As a result, the classification problem introduces an order of importance into the classification errors and thus in the misclassification costs. For example, for the label l_1, the costs are ordered such that:

$$c_{l_1,l_1} \leq c_{l_1,l_2} \leq c_{l_1,l_3} \leq \cdots \leq c_{l_1,l_K} \tag{2.7}$$

Example-dependent cost-sensitive classification In addition to class-dependent costs, the thesis application also addresses *varying* cost errors depending on the *data* itself. To illustrate an example of example-dependent cost-sensitive classification problem, we use the same example of cancer prediction in medical diagnosis as presented above. We can suppose that the classification error of declaring a patient with advanced cancer as a healthy patient is more serious than declaring a patient with cancer at its early stages as a healthy patient. In this example, the misclassification is the same: the predicted label is healthy while the true label is cancer. Nevertheless, the misclassification severity differs between the two patients, and the misclassification cost is therefore dependent on both the type of error and the data.

Let c_{n,l_j,l_i} be the cost of predicting label l_i when the true label is l_j regarding the n-th individual of $\mathcal{D}$. The costs due to misclassification vary between samples (each sample has its own cost matrix) such that:

$$\exists n, \tilde{n} \in [1, N], \exists l_i \in \mathcal{L} : \begin{cases} l^n = l^{\tilde{n}} \\ c_{n,l^n,l_i} \neq c_{\tilde{n},l^{\tilde{n}},l_i} \end{cases} \tag{2.8}$$

These example-dependent costs of misclassification are defined by the industrial application, from a business expertise and not by learning. They will be used in the application of the method from Chap. 3 (see Eq. 3.32). In Sec. 2.2.3, we will review existing methods on cost-sensitive learning to involve the example-dependent costs during both model learning and evaluation.

2.2.3 Related work on the classification problem

2.2.3.1 Time series classification

TS classification is a major topic of research in the literature on TS analysis [26, 103, 108]. A popular classification problem is the classification of Electroencephalograph (EEG) signals in order to identify mental states of the patients [34, 63]. In Sec. 2.1.3, we introduced

some models to learn temporal relationships in TS and we motivated the use of DNN. More information about TS classification models can be found in [6].

2.2.3.2 Performance metrics for multi-class classification

For binary classification problems, methods from the literature use different performance metrics in order to evaluate the classifiers, such as accuracy, sensitivity, specificity and precision for which definitions can be found in [95]. Some performance metrics for binary classification problems can be extended to the multi-class case, by measuring performance individually on each label and summarizing it into a confusion matrix, illustrated in Tab. 2.1. Combining performances on each label into a single metric can then be assessed through macro-averaging or micro-averaging [95].

Micro-averaging gives each prediction an equal contribution to the overall score and consequently reflects the classifier performance on large classes. For example, accuracy Acc_{micro} computed by micro-averaging on a dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ and with $\widehat{l^n}$ the classifier prediction on the n-th individual is defined by:

$$Acc_{micro} = \sum_{n=1}^{N} \mathbb{1}(\widehat{l^n} = l^n)/N \tag{2.9}$$

On the opposite, macro-averaging computes the performance scores on each label $l_k \in \{l_1, \cdots, l_K\}$ and then averages these scores. For example, we define the accuracy score Acc_k on a label $l_k \in \{l_1, \cdots, l_K\}$ by:

$$Acc_k = \frac{\sum_{n=1}^{N} \mathbb{1}(\widehat{l^n} = l^n) \times \mathbb{1}(l^n = l_k)}{\sum_{n=1}^{N} \mathbb{1}(l^n = l_k)} \tag{2.10}$$

and accuracy Acc_{macro} computed with macro-averaging is calculated over accuracy scores Acc_k on each label $l_k \in \{l_1, \cdots, l_K\}$:

$$Acc_{macro} = \sum_{k=1}^{K} Acc_k/K \tag{2.11}$$

As a consequence, macro-averaging then gives each class an equal contribution to the overall score. It reflects the classifier performance on each class, then giving more weight to small classes than micro-averaging. **During experimental evaluations, we will either measure performance through macro or micro accuracy depending on the dataset.** In the industrial application, we will complete the model evaluation with performance metrics related to the application.

2.2.3.3 Class-imbalanced classification

The literature offers several solutions for the class-imbalanced classification problem. First, the imbalance issue can be taken into account at the data-level [18, 61, 65]. The simplest strategies are to re-sample the dataset, either by under-sampling (i.e. removing samples from the majority labels) or by over-sampling (i.e. adding samples from the minority labels). Under-sampling may cause potential useful samples to be removed from the training set and therefore loss of information for the classifier to train on. In opposition, by replicating samples

from the minority class in the dataset, over-sampling can cause the classifier to fit too well on these replicates and then cause over-fitting.

Second, the imbalance issue can be taken into account at the algorithm-level. Some algorithms are known to perform well on classes with few samples, such as ensemble techniques which refer to the combination of several classifiers [32]. AdaBoost is a popular ensemble method which combines weak classifiers trained on a dataset with modified distribution [29]. The idea is to iteratively train a classifier on the dataset and then re-weight the samples on which the classifier performed badly. The next classifier is trained on a modified distribution of the dataset, with previously misclassified samples being more represented.

Another approach is to solve the class-imbalanced classification problem through cost-sensitive learning, introduced in Sec.2.2.2.4, by imposing a higher cost penalty on the minority class.

In this thesis work, we will be inspired by resampling methods from the literature which seek to achieve a class equilibrium in the data. In Chap. 3, we will propose a method based on the training of a classification model whose learning base evolves during its training, and we will ensure in Chap. 5 that the learning base remains balanced in its representation of the labels with a resampling strategy at the data level.

2.2.3.4 Ordinal classification

When the classification problem is ordinal, a naive approach is to assign an increasing numeric value to each class in the order of the classes, and then to solve a regression task with metrics such as Mean Square Error or Mean Absolute Error for which definitions can be found in [15].

[28] proposes to transform the multi-class ordinal classification problem for which there are K ordered classes into $K-1$ binary classification problems. The k-th binary problem seeks to predict on a sample data if its label is higher or lower than the label l_k. The final prediction is then performed by combining each of the $K-1$ probabilities.

In this doctoral work, we address a classification problem which is not only ordinal but also cost-sensitive and, therefore, we propose to take into account the order of labels in the definition of misclassification costs. The more two labels are distant from each other, the higher the cost of misclassification between these two labels (Eq. 2.7). As for the costs of misclassification, we will choose to integrate them during model learning, as mentioned below.

2.2.3.5 Cost-sensitive learning

Similarly to class-imbalanced problems, the literature propose balancing strategies [25, 61] for cost-sensitive applications. The methods aim at resampling the training dataset accordingly to the misclassification costs at the data level. Then, the newly distributed training dataset can be used to learn a classifier with conventional cost-insensitive classification algorithms.

Other literature studies [61, 92] propose a solution to cost-sensitive learning, specifically for classification models which can predict a probability on the label. They seek to determine an optimal probability threshold from which the class has to be predicted. To that end, empirical thresholding allows to determine on the training dataset the threshold values to select in order to achieve optimal classification costs.

Example-dependent cost-sensitive learning Literature on classification problems with varying misclassification costs across the data itself is recent and limited to a few applications, such as credit card fraud. In [10], the authors propose a method which incorporates the misclassification costs during training of a decision tree. They modify the impurity measure and the pruning strategy used to build the tree, making them consider a new splitting criterion which takes into account the misclassification costs.

Other approaches consist modifying the training dataset before training the algorithm, either through over-sampling or under-sampling [115]. A sample from the training dataset is either copied or rejected according to its normalized misclassification cost.

In this doctoral work, we will propose a method which consists in training a classification model that seeks to maximize a cost function related to misclassification costs (see Chap. 3, Eq. 3.32, Eq. 3.33). The method will be based on costs which are defined in advance and not learned.

2.3 Definition of the early classification problem

2.3.1 Time-sensitive classification

The thesis problem addresses time-sensitive classification, that is, *acquiring* or *waiting* for an additional data point $\mathbf{x}_{t+1}$ to complete a prefix $\mathbf{X}_{:t}$ is *costly*. For example, in microbiological diagnostics, each data acquisition is expensive because of the experiments it requires to conduct.

Let $f_{classif}$ be a classification model and $\hat{l}$ its prediction on the prefix $\mathbf{X}_{:t}$, such that $\hat{l} = f_{classif}(\mathbf{X}_{:t})$. In other words, we consider that the model classified the sequence at time t, using its first t data points $(\mathbf{x}_1, \cdots, \mathbf{x}_t)$. We define the total cost C_{total} of classifying the sequence $\mathbf{X}$ from the sample $(\mathbf{X}, l)$ into a label $\hat{l}$ and at time t by:

$$C_{total}((\mathbf{X}, l), \hat{l}, t) = C_{time}(t) + C_{classif}((\mathbf{X}, l), \hat{l}) \tag{2.12}$$

where:

- $C_{time}(t)$ is the cost of acquiring or waiting for the t first consecutive data points $(\mathbf{x}_1, \cdots, \mathbf{x}_t)$ in the sequence $\mathbf{X}$, before classifying $\mathbf{X}_{:t}$. It is determined by the application and will be referred to as the *cost of time*.
- $C_{classif}((\mathbf{X}, l), \hat{l})$ is the cost of predicting label $\hat{l}$ on data $\mathbf{X}$ with true label l, also referred to as the *cost of classification*. It depends on the nature of the classification problem, either cost-sensitive or cost-insensitive, and may be example-dependent. As explained in Sec. 2.2.2.4, the thesis addresses an example-dependent cost-sensitive classification problem and we defined the cost of classification on the n-th sample $(\mathbf{X}^n, l^n)$ from $\mathcal{D}$ as $C_{classif}((\mathbf{X}^n, l^n), \hat{l}) = c_{n,l^n,\hat{l}}$.

In conventional classification problems, the response time does not matter and therefore the cost of time is zero, $C_{time}(t) = 0$. By contrast, in the context of time-sensitive classification, there is a non-zero cost of delay or feature acquisition, $C_{time}(t) \neq 0$, and the time of classification therefore becomes a target to optimize. The objective is then to *early* classify input data with as *few* data points as possible while ensuring the *quality* of the classification. We will detail the EC objectives in the next section.

2.3.2 General objectives of early classification

Classification of incomplete sequences We first define an early classifier as a model capable of analyzing *incomplete* temporal sequences. It has to be able to classify a TS *at any time*, from the beginning of its acquisition to its completion. An early classifier is then a function $f_{classif}$ capable of performing classification on any prefixes of the sequences $\{\mathbf{X}_{:1}, \mathbf{X}_{:2}, ..., \mathbf{X}_{:t}, ..., \mathbf{X}_{:T}\}$:

$$\forall t \in [1, T],\ f_{classif} : \{\mathbf{X}_{:t}\} \rightarrow \mathcal{L} \tag{2.13}$$

Minimization of prediction time Second, we choose to define an early classifier as a model seeking to *optimize* the time of prediction. It has to perform the classification *as early as possible*, i.e. using the smallest prefixes of sequences. In Sec. 2.4, we will review how methods from the literature address the EC problem, and we will show that *time* does not always appear in the objectives definition nor in the methods optimization. By contrast, in this work, we want time to both appear in the optimization of the method, and be a criterion whose importance is adjustable by the user.

Adaptive prediction time Additionally, we consider that the time of prediction has to be *individually* decided on each sample, and not globally on the entire training set. Indeed, some sequential data might need more time than others to be accurately classified. On these complex data, the early classifier has to perform classification later (using more data points) than on easier data. Therefore, the time of classification has to be adapted and optimized individually on each sample.

Multiple optimization In this work, we consider that an early classifier simultaneously seeks to minimize the two competitive costs of classification and earliness. It aims at predicting as quickly as possible so that it lowers the cost induced by prediction time. It also aims at classifying as accurately as possible so that it achieves the lowest misclassification costs. These two competing objectives and the cost associated are presented in more details in Sec. 2.3.3.

2.3.3 The early classification trade-off

Formulation of EC as an optimization problem We define an early classifier as a model capable of individually analyzing prefixes of sequences, predicting the optimal time of prediction t^* and optimal label $\hat{l}^*$ defined by:

$$(t^*, \hat{l}^*) = \underset{t \in [1,T], \hat{l} \in \mathcal{L}}{\operatorname{argmin}}\ C_{total}((\mathbf{X}, l), \hat{l}, t) \tag{2.14}$$

$$= \underset{t \in [1,T], \hat{l} \in \mathcal{L}}{\operatorname{argmin}}\ (C_{time}(t) + C_{classif}((\mathbf{X}, l), \hat{l})) \tag{2.15}$$

where:

- $\hat{l}^*$ is the predicted label using the prefix $\mathbf{X}_{:t^*}$, such that $\hat{l}^* = f_{classif}(\mathbf{X}_{:t^*})$. Consequently, the predicted label depends on the time of prediction, and an increase or decrease in this time may change the predicted label.

- C_{time} is the *cost of time* introduced in Eq. 2.12. It is dependent of time and independent of the predicted label.
- $C_{classif}$ is the *classification cost* introduced in Eq. 2.12. It is dependent of the predicted label, and therefore indirectly of time.

Following this definition, EC is an optimization problem involving a joint optimization of two costs.

Competitive costs An early classifier has to simultaneously optimize the two competitive objectives of classifying accurately while using the smallest prefixes. The objectives of accurate and fast classification involve competitive costs. Indeed, a classification performed at an earlier time step costs less in data points acquisition, or in waiting time, but is likely to cost more (or equally) in classification quality due to fewer information in the shortest prefix.

C_{time} is the cost related to the time of the prediction. The origin of this cost comes either from the acquisition cost of new data points in the sequence, or more generally from the delay of classification, specific to each application. We consider that C_{time} increases during the sequence acquisition, such that for two time steps $t_1, t_2 \in [1, T]$ with $t_1 < t_2$, predicting a label at time step t_2 is more expensive than at time step t_1, regardless of the predicted label:

$$C_{time}(t_1) \leq C_{time}(t_2) \tag{2.16}$$

Indeed, at time step t_2, the prefix $\mathbf{X}_{:t_2}$ is composed of more data points than the prefix $\mathbf{X}_{:t_1}$ at time step t_1, which means that the user waited longer and/or required more data points for $\mathbf{X}_{:t_2}$ leading to an increase in C_{time}.

On the opposite, $\mathbf{X}_{:t_2}$ is supposed to be a richer sequence than $\mathbf{X}_{:t_1}$. We can suppose that performing classification on the prefix $\mathbf{X}_{:t_2}$ is made easier by the presence of more data points. We then suppose that the result of classification $f_{classif}(\mathbf{X}_{:t_2})$ is more accurate than $f_{classif}(\mathbf{X}_{:t_1})$. $C_{classif}$ is the cost related to misclassification. We consider that $C_{classif}$ decreases during the sequence acquisition:

$$C_{classif}((\mathbf{X}, l), f_{classif}(\mathbf{X}_{:t_1}) \geq C_{classif}((\mathbf{X}, l), f_{classif}(\mathbf{X}_{:t_2})) \tag{2.17}$$

We note that this assertion may not always be true for applications where noise is added to the data from a certain amount of data points. However, we hypothesize that up to a certain amount, the acquisition of new data points makes the classification problem simpler. We therefore consider that generally, when C_{time} increases, then $C_{classif}$ decreases, and conversely, which makes the costs competitive, as illustrated in Fig. 2.4.

2.4 Related work on early classification of temporal sequences

2.4.1 The first research paper on early classification

Before 2002, researchers on the topic of sequence classification focused on optimizing accuracy of classification models for temporal data. The topic of early classification first appears in [5] with a method capable of making predictions on variable length sequences and consequently on prefixes of TS. The authors use an ensemble of simple literals (the base classifiers) indicating if the TS increases, decreases or remains in a specific range of value given some time intervals.

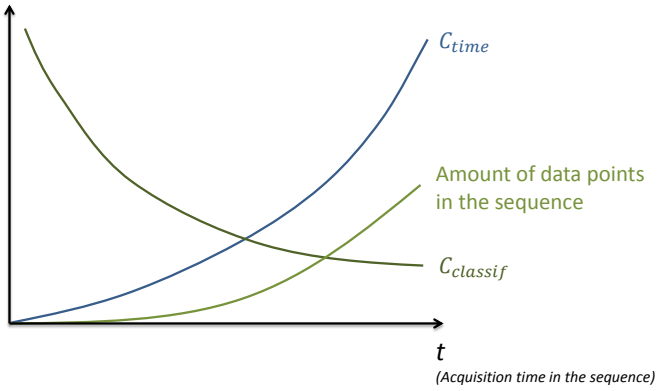

Figure 2.4: **Competitive costs of prediction time and classification quality.** The more data points are acquired in the sequence (shown in light green curve), the more expensive the acquisition or delay (shown in blue curve). Nevertheless, the classification is likely to be more accurate and its cost decreases (shown in dark green curve).

These base classifiers are combined together through an adaptation of boosting algorithm which results in a final classifier being a linear combination of the literals.

By omitting the literals with unknown results in the linear combination, this approach enables incomplete examples to be classified and the authors manage to classify partial TS. However they are not interested in finding the shortest prefix which will ensure a reliable prediction and **their method does not consider earliness as a factor to optimize, contrary to the thesis objective**.

2.4.2 Shapelet-based methods

The authors in [107] are the first to propose a method which takes into account the trade-off between earliness and accuracy of classification. Their method focuses on symbolic sequences, i.e. sequences for which the data points are symbols (e.g. a DNA sequence with nucleotide). It is based on the identification of optimal patterns in the sequences that are frequent, early and distinctive. These patterns are then used in an association rule classifier or a decision tree classifier. The authors showed that their method is efficient when applied to symbolic sequences, however it does not give good results on numerical sequences which need to be discretized, often responsible for information loss.

The problem of extracting sub-sequences in numerical sequences that would be relevant for classification is addressed in [113]. In this paper, the authors seek for useful "shapelets" which correspond to a sub-sequence associated to a distance threshold, with a high discriminative power between labels. The shapelets are then used as nodes of a decision tree for classification.

An adaption of this work was proposed by [110] who sought for local shapelets that were not only distinctive but also early. Their method is called Early Distinctive Shapelet Classification (EDSC) and it classifies a sequence as soon as it matches with one of the most useful pre-selected shapelets. [38] proposes to complete this approach with measures of confidence on the

shapelets' capacity to classify input examples. When a sequence matches several shapelets, their method estimates the classification uncertainty for each class represented by the matched shapelets and assign the label of the class with lowest uncertainty.

In [36], the authors generalize the use of local shapelets proposed in [110] for the problem of EC on MTS. They introduce a method called Multivariate Shapelets Detection. Just as univariate shapelets, multivariate shapelets are multiple subsequences of the MTS, where each subsequence is extracted from exactly one feature. Contrary to [36] where all segments of a multivariate shapelet have to be extracted in the same sliding time window at the same time, the authors in [43] are interested in finding useful shapelets for each feature independently. Their goal is to find, for each feature independently, distinctive shapelets that are early in the sequences. These shapelets are then used as core features in classifiers.

The advantage of shapelet-based solutions [36, 38, 43, 60, 107, 110, 112, 113] is that the classification is based on real patterns extracted from the sequences, which facilitates the interpretation of the classification results. However, these solutions are often time consuming and they are suitable for applications where classes can be easily discriminated by some typical patterns in the sequences.

In this doctoral work, we want to develop a method that can be used on complex data for which there might be no discriminating patterns in raw data without transformation. Moreover, in the industrial application, we are dealing with complex data with a large pattern diversity and for which classes overlap (see Fig. 5.5).

2.4.3 Distance-based methods

In [109], the authors aims at adapting the highly competitive nearest neighbors approach on sequence classification to EC with a method called Early Classification on Time Series (ECTS). During the training phase and for each sequence in the training set, the method finds the earliest time from which the sequence becomes stable in terms of neighbors. During the testing phase, a partial sequence is classified as soon as it has a neighbor which was stable at that time during training.

The method proposed in [109] involves univariate data and it does not propose adaptation for multivariate data. To adapt this solution to multivariate data, the user has to find a distance metric adapted to his problem and his data. In the industrial application of the thesis, we would have to adapt the solution to our complex dataset for which the sequences can be shifted in time and for which some features are more important. A related work can be found in [66] where the authors classify MTS with the Dynamic Time Warping measure to find the best alignment in time between two sequences and by combining it with the Mahalanobis Distance to assign non-constant weight to each feature of the sequence. In [66], the authors outlined the high computational cost of their framework.

In this doctoral work, we want to develop a method which allows the user to adjust the relative importance of prediction time compared to classification quality, which is not proposed in the method from [109].

2.4.4 Probabilistic methods

In [7, 79], the authors address EC as a problem of classification with confidence from incomplete information. Their solution is based on probability forecasting and on linear and quadratic discriminant functions as classifiers. They define the reliability of a prediction as

the probability that the prediction on a incomplete example would be the same than the one on the complete example. During the prediction phase, the method classifies a partial TS as soon as the estimated reliability of the prediction exceeds some user-defined threshold.

In [39], the authors propose a hybrid approach for the task of EC. They generate membership likelihoods on all segments of the sequence with Hidden Markov Models and then use those membership likelihoods as input to Support Vector Machines to predict the class probabilities. At test time, if the estimated class probabilities reach a user-defined threshold of confidence, the TS is classified.

Although in [7, 39, 79] and other probabilistic approaches [72] the user can have an impact on the earliness of the prediction by releasing the amount of confidence requested to the classifier, these methods do not explicitly take time into account in the classification decision. **In this doctoral work, we want the prediction time to directly appear as a criteria to minimize in the method.**

In [70, 71], the authors combine a set of probabilistic classifiers trained for each time step in the sequence with a stopping rule. Their stopping rule indicates if the classification can be trusted or delayed. It is based on time and on the probabilities estimated by the classifier. Specifically, it takes into account the amplitude of the greatest probability and at the difference between the two largest probabilities, reflecting if the label with largest probability is highly trusted and if this label stand out from other possibilities. The stopping rule's parameters are learned with a genetic algorithm by minimizing a cost function which takes into account the costs of accuracy and earliness.

In this paper, the authors can directly balance the importance of earliness over accuracy with the expression of a cost function taking into account the two objectives of an early classifier. However, since the classifiers are trained at each time step, this method requires that the training and testing sequences are aligned in time. **In this doctoral work, we do not want to train a set of classifiers where each classifier is dedicated to sequences of specific length. Instead, we want a method that can simultaneously analyze prefixes of any size.**

Additionally and in opposition to the authors' strategy of first optimizing the problem of classification, then that of the prediction time, **we aim at developing a method which can simultaneously solve these two sub-problems**.

2.4.5 Ensemble methods

In [41], the authors introduce a method for EC which uses an ensemble of classifiers. The idea is to label a TS as soon as a pair of classifiers agrees on the label. If the predicted label differ from a classifier to another, the classification results are rejected and the TS with a new data point will be treated by a new pair of classifier. By using a voting system, this method ensures a certain confidence in the classification but **it does not optimize the earliness of the prediction**.

2.4.6 Non-myopic methods

In opposition to the above mentioned methods which decide at each time step whether to make a prediction or to wait for more measures in sequential data, the authors in [20] propose a framework characterized as "non-myopic" which forecasts the (future) earliest time from

which classification can be made. The framework is built around a distance based method combined with clustering and a series of classifier trained at each time step.

The main disadvantage of the method lies in the clustering step which requires setting a number of parameters and which can greatly affect the performance of the method if it is not performed well. The same authors replace the clustering step with a segmentation method in [8] but they argue that their method is suitable for large training datasets only.

2.4.7 Methods with Neural Networks

In [4], the authors analyze videos and want to infer the action represented in the sequence of frames. Their goal is to infer the action label as soon as possible by using a portion of the frames only. To do so, they train a Long Short-Term Memory (LSTM) neural network with a modified version of the entropy loss. Their loss takes into account time and aims at penalizing false positives linearly over time.

At inference, they fix in advance the number of frames they will use for classification and they make use of an average pooling of predictions made by the network on these frames. There is no decision of acquiring more frames or not in order to predict the action label. **In this doctoral work, we seek to provide prediction times individually on each sample depending on their complexity and not globally on the entire training set.**

2.4.8 Early classification on other types of dynamic data

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, such as early or fast text classification in [23, 114] and classification with costly features in [49, 80].

Formulated as "learning when to stop thinking and do something" in [81], this problem was tackled by Reinforcement Learning. The authors are interested in "anytime algorithms" that can be interrupted at any time and for which we assume that the longer they "think", the better the quality of their response. In particular, the authors seek to build a policy that decides if an anytime algorithm should continue thinking or if it should return its current best answer. Their approach is policy-gradient-based and uses REINFORCE algorithm from [105].

In some applications featuring dynamic data, the acquisition of more data can be costly or unnecessary. In text classification for example, it is not always necessary to read an entire document to classify its content. In [23], 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 accomodate to small training datasets than standard non-sequential classifiers. In [114] the authors train an agent to classify texts as fast as possible and the agent is allowed to reread a sentence, read sequentially, and skip one or several token in the text.

The approach proposed in [23] consists in deciding between the collection of a single feature (the sentence) or the classification. This work was extended to multiple features selection by the same authors in [24]. 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 [80] is a policy-based method using REINFORCE algorithm 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 data points collected by the agent before and after making an action.

The problem of costly feature acquisition in the medical domain is also tackled in [48, 49] where the authors propose to optimize the trade-off between classification accuracy and the total feature cost using a Deep Reinforcement Learning based on Double Deep-Q-Network algorithm from [100]. The authors demonstrate the capability of their algorithm to solve classification problems efficiently. We will see that the solution proposed in Chap. 3 is close to the research work from [48] published at the same time as this doctoral work.

2.5 Conclusion

In this chapter, we introduced the data to be processed during the thesis, the classification problem to be solved, and the objectives of EC.

We showed that temporal sequences are specific data for which it is necessary to use models that can take into account the temporal relations in the data points. In the specific case of MTS, the models must also take into account the relationships between the different features of the sequence. For this purpose and as part of a method that can be applied to data on which we have no expertise, we identified the interest of DNNs, among which CNNs are a possible choice of model. We also showed that the data are all the more complex as they arrive sequentially and therefore have variable lengths.

We defined the classification problem of this doctoral work and we reviewed related work from the literature. We presented methods for calculating classification performance for multi-class and imbalanced problems, especially with macro and micro-averaging on usual binary classification metrics. For the problem of imbalanced class, we will be able to draw on resampling strategies from the literature that seek to create a balance in the data. For the problem of ordinal and cost-sensitive classification, we will propose in Chap. 3 a solution which differs from those of the literature.

We showed that EC is an optimization problem that implies a trade-off between classification quality and earliness. Among the methods from the literature, we showed that none can take into account all the specificities of the classification problem. In the rest of this study, we will therefore propose a method adapted to the precise problem of EC on MTS (and more generally on temporal sequences of various types) when labels are multi-class, imbalanced, ordinal and when classification is example-dependent cost-sensitive.

Chapter 3

EC formalization as a Partially Observable Markov Decision Process

In Chap. 2, we defined and analyzed the problem of early classification (EC) on multivariate time series (MTS) which are the core data of the industrial application. We showed that the thesis addresses time-sensitive applications (Sec. 2.3.1) for which EC is of primordial importance. Indeed, time-sensitive applications seek to get a classification result as early as possible on incomplete data either to take early actions or to minimize some acquisition cost. This is for example the case of medical diagnosis [37, 51, 80] which seeks to early identify diseases in order to adopt early treatments, and the case of predictive maintenance [45] which seeks to anticipate a machine's breakdown in order to minimize downtime.

In this doctoral work, we associate the problem of EC to applications which sequentially collect temporal sequences with new data points arriving at each time step. We consider that these applications are interested in making online decision, at each time step t, to perform classification on the partial sequence $\mathbf{X}_{:t}$ (Eq. 2.2) or to delay classification in order to get an additional data point $\mathbf{x}_{t+1}$.

In this chapter, we propose a mathematical framework for the general problem of EC on temporal sequences. We formalize EC as a sequential decision-making problem in Sec. 3.1. We evaluate the suitability of both Supervised Learning (SL) and Reinforcement Learning (RL) to solve the problem in Sec. 3.2 and Sec. 3.3. We show that the problem can be transposed into a RL framework by defining a Partially Observable Markov Decision Process (POMDP) for EC, noted EC-POMDP, in Sec. 3.4.

3.1 Formalization of a sequential decision-making problem

3.1.1 Definition of the end-to-end decision model

We address EC as a sequential decision-making problem for which some actions have to be taken: to classify or to wait. From an incomplete sequence $\mathbf{X}_{:t}$, the early classifier can decide to perform classification now (at time t) or to delay prediction for at least one more time step and get $X_{:t+1}$ as an updated sequence.

We propose to define $\mathcal{A}$ as the set of possible actions. An action $a \in \mathcal{A}$ is defined over the set $\mathcal{A}_c$ of classification actions plus an additional action a_d for delay:

$$\mathcal{A} = \mathcal{A}_c \cup a_d \tag{3.1}$$

In this work, the set $\mathcal{A}_c$ of classification actions is defined over the set $\mathcal{L}$ of labels $l_k, k = 1..K$:

$$\mathcal{A}_c = \mathcal{L} \tag{3.2}$$

If $a = l_k$, then the action is to predict label l_k.

We define an early classifier as an end-to-end decision-maker f which directly maps from prefixes of sequences to actions of classification or delay (Fig. 3.1):

$$f : \{\mathbf{X}_{:t}\}_{t\in[1,T]} \rightarrow \mathcal{A} \tag{3.3}$$

The time of prediction on a sequence $\mathbf{X}$ is t_{pred}. It is defined as the first time at which the decision-maker f chooses a classification action:

$$t_{pred} = \min_{t\in[1,T]} \{t \mid f(\mathbf{X}_{:t}) \in \mathcal{A}_c\} \tag{3.4}$$

The predicted label on a sequence $\mathbf{X}$ is $\widehat{l_{pred}}$. It is defined as the classification action taken by the decision-maker f at the time of prediction t_{pred}:

$$\widehat{l_{pred}} = f(\mathbf{X}_{:t_{pred}}) \tag{3.5}$$

We point out the differences between this problem definition and a classification problem. A classifier $f_{classif} : \{\mathbf{X}\} \rightarrow \mathcal{L}$ maps a *complete* sequence to a *label*, while an early classifier (a decision-maker) $f : \{\mathbf{X}_{:t}\}_{t\in[1,T]} \rightarrow \mathcal{A}$ maps the *prefix* of a sequence to an *action*.

3.1.2 Synthesis of the thesis objectives

Following the problem definition from Chap. 2 and its formalization as an end-to-end sequential decision-making problem, the aim of the thesis is to propose a methodology addressing the following objectives:

- The task is to train an end-to-end decision-maker model f for EC. For that purpose, the model has to directly map from prefixes of sequences to classification or delay, as defined in Eq. 3.3.
- With t_{pred} defined in Eq. 3.4 and $\widehat{l_{pred}}$ defined in Eq. 3.5, the model has to be a solution of the optimization problem from Sec. 2.3.3:

  $$(t^*_{pred}, \widehat{l^*_{pred}}) = \underset{t\in[1,T],\hat{l}\in\mathcal{L}}{\operatorname{argmin}} \left(C_{time}(t) + C_{classif}((\mathbf{X}, l), \hat{l})\right)$$

  with $C_{time}(t)$ is the cost of acquiring or waiting for the t consecutive data points $\mathbf{X}_{:t} = (\mathbf{x}_1, \cdots, \mathbf{x}_t)$ to perform classification and $C_{classif}((\mathbf{X}, l), \hat{l})$ the cost of predicting label $\hat{l}$ on data $\mathbf{X}$ with true label l.

- The user must be able to specify the relative importance of prediction time compared to classification quality.
- The model f has to be optimized simultaneously for the two sub-problems of classification and prediction time, i.e. we will not optimize each sub-problem separately.
- The method has to address the thesis specification detailed in Chap. 2. We remind that data $\mathbf{X}_{:t}$ are prefixes of non-stationary MTS with P features and maximal length T (Sec. 2.1). The set of labels $\mathcal{L} = \{l_1, ..., l_K\}$ is a finite and ordered set of $K \in \mathbb{N}^+$ distinct labels with $l_1 < l_2 < ... < l_K$. Labels are not equally represented in the training dataset and classification costs are both class-dependent and example-dependent (Sec. 2.2).
- In order to generalize the method to other applications, it has to be adapted to other types of temporal sequences (other than MTS) and for which we have no expertise.

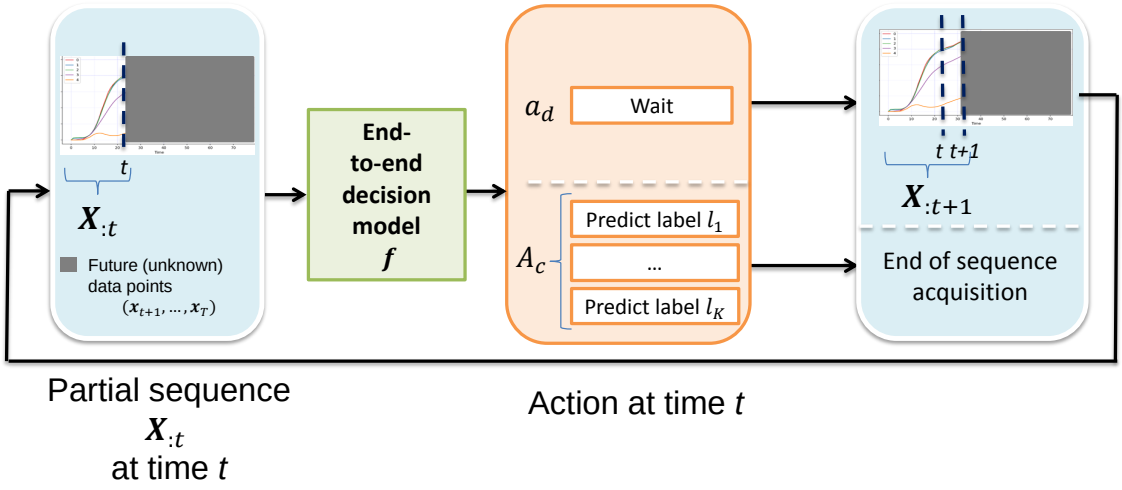

Figure 3.1: **Early classifier as an end-to-end decision-maker model.** At time t of the sequence acquisition process, the early classifier receives a prefix $\mathbf{X}_{:t}$ (see Fig. 2.1) and chooses an action $a \in \mathcal{A}$, either to wait for an additional data point $\mathbf{x}_{t+1}$ in the incomplete sequence, or to predict a label $l \in \mathcal{L}$. Once the early classifier chooses a classification action $a \in \mathcal{A}_c$, the sequence is no longer acquired and the process ends.

3.2 Limitations of Supervised Learning

In this section, we will investigate whether the decision-making problem of EC can be solved with SL and we will show that it suffers from incomplete supervision.

3.2.1 Background on Supervised Learning

SL is a branch of Machine Learning used to learn a model f that maps some data $\mathbf{X}$ to some output variable y [14]. When the output variable y belongs to a finite set of label, the task is known as *classification*. On the contrary, when the output variable y is a numerical value, the task is known as *regression*. In both cases, learning a model $f : \mathbf{X} \to y$ with SL requires a training dataset of labelled data $\{(\mathbf{X}, y)\}$, that is to say a set of samples with data $\mathbf{X}$ and their true output variable outcome y. In order to evaluate how well a model f fits the training samples, its error is measured through a loss function L, such that $L : \{y\} \times \{y\} \to \mathbb{R}$. With $\hat{y} = f(X)$ the model prediction on a sample pair $(\mathbf{X}, y)$, the loss on this sample is $L(y, \hat{y})$.

3.2.2 EC: a problem with incomplete supervision

The problem introduced in Sec. 3.1 is to learn an end-to-end decision-maker f for EC, such that $f : \{\mathbf{X}_{:t}\}_{t\in[1,T]} \to \mathcal{A}$, with $\mathcal{A}$ defined in Eq. 3.1. In this section, we investigate the solution which consists in learning a model for f with SL. As mentioned above, SL algorithms require a training set of sample pairs with input data and their output variable. The question is therefore the following: how to build a training dataset for supervised learning of an end-to-end early classification model?

Given the definition of f (Eq. 3.3) as a model mapping sequence prefixes to actions, a training dataset to learn a model with SL has to be composed of sample pairs $(\mathbf{X}_{:t}, y)$ with $\mathbf{X}_{:t}$ a partial sequence and $y \in \mathcal{A}$ the optimal action to take on this prefix. We note this dataset $\mathcal{D}_{sup}$:

$$\mathcal{D}_{sup} = \{(\mathbf{X}^1_{:1}, y^1_{:1}), \cdots, (\mathbf{X}^1_{:T}, y^1_{:T}), \cdots, (\mathbf{X}^N_{:1}, y^N_{:1}), \cdots, (\mathbf{X}^N_{:T}, y^N_{:T})\} \tag{3.6}$$

with $y^n_{:t} \in \mathcal{A}$ the optimal action to take at time t on the prefix $\mathbf{X}^n_{:t}$. We remind that we have a training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ with pairs of complete temporal sequences $\mathbf{X}$ and their reference label l. In the following, we will show that turning $\mathcal{D}$ into a dataset $\mathcal{D}_{sup}$ for SL is not straightforward. It requires to find optimal action labels $y \in \mathcal{A}$ for each prefix $\{\mathbf{X}_{:t}\}_{t\in[1,T]}$, as illustrated in Fig. 3.2.

A trivial labelling is to associate all complete sequences $\mathbf{X}_{:T}$ to the classification action associated to their true label l such that:

$$\forall n \in [1, N],\ y^n_{:T} = l^n \tag{3.7}$$

Indeed, once the sequence is fully completed, there are no more additional data points to collect and the only possible action is to classify the sequence. The optimal action is then to predict the label associated to the sequence. We deduce that $\{(\mathbf{X}^1_{:T}, l^1), \cdots, (\mathbf{X}^N_{:T}, l^N)\} \subset \mathcal{D}_{sup}$.

What about optimal action labels on prefixes of sequences? A first idea is to assign the action of predicting the true label l^n to each prefix of the complete sequence $\mathbf{X}^n$, regardless

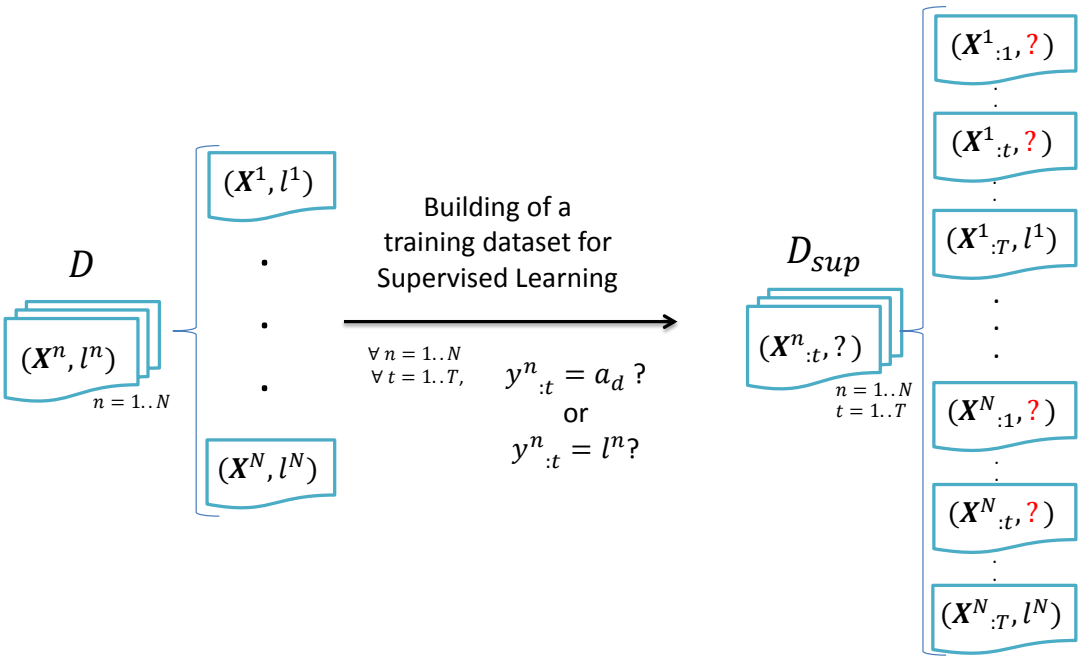

Figure 3.2: **Building of a training dataset $\mathcal{D}_{sup}$ for SL from the training dataset $\mathcal{D}$.** $\mathcal{D}$ is composed of N pairs of complete sequences $\mathbf{X}$ and their true label l. In order to build $\mathcal{D}_{sup}$ (of size $N \times T$), the problem is to label each prefix $\mathbf{X}^n_{:t}$ with an action label $y^n_{:t} \in \mathcal{A}_c \cup a_d$. Trivial labelling is to associate complete temporal sequences $\mathbf{X}_{:T}$ to their true label.

of its length. By doing so, we suppose that the model f is able to predict the true label at all time steps in the sequence, which goes against the definition of the EC problem. Moreover, the EC problem is not to classify a prefix, but rather to make the decision to classify this partial data (and consequently to stop its acquisition process) or continue to collect data points.

The challenge associated to partial sequences labelling for SL is the following. If we assign the action of predicting label l^n on prefixes from $\mathbf{X}^n$ that are too short, we "ask" for the model f to find a mapping between the partial data and the label, whereas the data has no discriminative pattern for classification yet. In other words, we "ask" for a mapping that does not exists yet. Conversely, if we assign the action of correct label prediction on prefixes that are too long (and if we assign the action of delay for the time steps that precede), the risk is to encourage the model f to provide slow predictions. Indeed, the model might learn to recognize discriminative patterns that appear at late times in the sequences only. For this reason, finding the optimal time step from which the action of correct label prediction has to be assigned is difficult. It depends on each data and can vary between samples. It requires to recognize from which time step the prefix is composed of a discriminative pattern.

Generally, we consider that adding a sample pair $(\mathbf{X}_{:t}, l)$ to $\mathcal{D}_{sup}$ means that $\mathbf{X}_{:t}$ contains a discriminant classification pattern allowing to identify the label l. If not, the prefix $\mathbf{X}_{:t}$ has to be labelled to the action of delay a_d. Unfortunately these information are not available without further efforts.

In conclusion, the task of finding when to assign the action of predicting the true label

is not an straightforward. Finding optimal action labels on partial sequences $\{\mathbf{X}_{:t}\}_{t<T}$ is a more difficult task because, on these incomplete sequences, the model can chose an action between delay and classification.

The general conclusion is that we can transform the available dataset $\mathcal{D}$ into a dataset $\mathcal{D}_{sup}$ for SL with N labelled samples (all related to complete sequences $\{\mathbf{X}_{:T}\}$) and $N \times (T-1)$ unlabelled instances (all related to partial sequences $\{\mathbf{X}_{:t}\}_{t<T}$):

$$\mathcal{D}_{sup} = \{(\mathbf{X}^1_{:T}, l^1), \cdots, (\mathbf{X}^N_{:T}, l^N), \mathbf{X}^1_{:1}, \cdots, \mathbf{X}^1_{:T-1}, \cdots, \mathbf{X}^N_{:1}, \cdots, \mathbf{X}^N_{:T-1}\} \tag{3.8}$$

The training dataset for SL $\mathcal{D}_{sup}$ is therefore composed of samples for which labels are missing. Given that we can not bring supervision on those samples, we have an incomplete supervision on the problem.

3.2.3 Related work on Supervised Learning with incomplete supervision

3.2.3.1 Learning from domain knowledge

When the user has a good knowledge of the application and data, he can establish hand-engineered rules and apply them to unlabelled samples for automated data tagging. As an example, assuming that no valuable information are available in the prefixes before time steps below a threshold ($t \leq t_{thresh}$), the user can annotate the prefixes $\{\mathbf{X}_{:t}\}_{\forall t \leq t_{thresh}}$ with the action label for delay a_d such that $\{(\mathbf{X}_{:t}, a_d)\}_{\forall t \leq t_{thresh}} \subset \mathcal{D}_{sup}$.

This solution is however limited and presents drawbacks. First, finding rules for automated annotation depends on the application and requires a good knowledge of data. It does not apply on complex data which are not easily understandable. Then, tagging samples with hand-engineered rules can insert some errors or sub-optimal labelling in the training dataset. This can lead to learning a model that is not optimal and consequently that does not solve the optimization problem posed in Sec. 2.3.3. In the following, we review existing work from the literature to train a model with SL when the dataset has unlabelled samples.

3.2.3.2 Learning from expert demonstrations

Some work in the literature propose solutions when an expert is available to demonstrate a task or annotate unlabelled samples.

Behavior cloning Behavior Cloning [58, 77, 86] consists in learning from demonstration and is used in applications for which humans can demonstrate the task of interest, such as Autonomous Land Vehicle Navigation [82] or flying an aircraft [69]. It supposes that an expert is available and it learns a policy (Sec. 3.3.1) with SL using the expert's demonstrations. More precisely, the idea is to observe some expert performing the task at stake, to collect its demonstrations and then to learn to replicate its behavior. The expert is assumed to behave optimally by picking the optimal action in every circumstances.

Behavior Cloning brings several limitations. First, the cost induced by the collection of demonstrations from experts can be prohibitive in some applications. Second, it is possible that we can not get demonstrations because we do not know how to behave optimally on the task at issue. In both cases, setting a training dataset is not possible. Furthermore, another limitation relates to performance. Behavior Cloning is used to replicate a behavior and not to improve it, whereas for some applications we might want to perform better than the expert.

Active learning Applications for which annotations and/or demonstrations from expert can be collected under a limited budget can address the incomplete supervision issue with Active Learning [76, 117]. The principle is that the learning algorithm can query an expert to annotate some new data points. The algorithm selects unlabelled data or generates new samples it wants to learn from [90]. In particular, it estimates which samples are the most useful to improve the model.

As with Behavior Cloning, a limitation of Active Learning is the query cost along with the assumption that an expert can optimally perform the task at hand. **In the case of the thesis, we have no annotations or demonstrations from experts.** Behavior Cloning and Active learning are not possible solutions to the problem.

3.2.3.3 Semi-Supervised Learning

Incomplete supervision can be solved with semi-Supervised Learning [17, 117, 118] which is a form of classification that can learn from datasets with unlabelled samples (without querying experts). In general, semi-supervised algorithms are highly used in applications for which unlabelled data are easy to collect while collecting labelled data is harder, expensive or time-consuming. These algorithms make the assumption that labelled and unlabelled data come from the same data distribution. They assume that clusters can be found in the data and that two data points belonging to the same cluster have the same output variable.

In the particular case of the EC problem, labelled data are complete sequences while unlabelled data are partial sequences. It is therefore expected that, because complete sequences contain more data points than partial sequences, short prefixes of sequences are unlikely to belong to the same cluster data points than labelled data. Also, labelled data are restricted to classification actions. Action for delay is not annotated in the training dataset. As a conclusion, we think that **semi-Supervised Learning is not a suitable solution for the particular problem of incomplete supervision from Sec. 3.2.2, because labelled samples exclude a whole class and are restricted to a sub-category of the data.**

3.2.4 Conclusion

In this section, we showed that solving the end-to-end decision-making problem of EC with SL necessitated to build a dataset $\mathcal{D}_{sup}$ with supervision on both actions of delay and classification at all time steps in the sequences. We then argued that an optimal supervision can be given on the complete sequences only, for which the decision-making problem is reduced to a classification problem. Without further efforts, we lack supervision regarding the optimal choice of action on partial sequences and, therefore, the problem of EC goes along with a training dataset $\mathcal{D}_{sup}$ with unlabelled samples. Finally, we showed that addressing the incomplete supervision issue with SL is not straightforward. Either the solutions require an expert for cloning or for annotations, or they are not applicable to the thesis problem, or they may result in sub-optimal supervision by imprecise or false annotations.

In previous chapter, we showed that some approaches from the literature addressed the EC problem in two steps, by isolating classification from the decision to predict a label, and used SL to solve the classification problem. However, as argued before, we seek to optimize the EC model in an end-to-end fashion, i.e. by optimizing the problems of classification and decision-making simultaneously.

In the remainder of the study, we will seek for another solution than SL and we will show that we can solve the decision-making problem of EC with RL in a more straightforward way.

3.3 Assets of Reinforcement Learning

In this section, we will show that the decision-making problem of EC can be solved within a RL framework going through the definition of a Markov Decision Process (MDP).

3.3.1 Background on Reinforcement Learning

RL is an active field of research for sequential decision-making [97]. It is traditionally used in games environment and has recently exhibited state-of-the-art results in these challenging tasks [44, 68, 100, 102]. It is also highly used in robotics for various tasks, from navigation in a room to grabbing objects [58].

RL refers to training an *agent*, also called a "decision-maker", to act optimally in an *environment* so that it maximizes its *rewards*. The agent's behavior is dictated by its *policy*, π, and the objective in RL is to find the optimal policy π^* the agent has to follow in order to maximize its rewards. To this end, the principle is to let the agent learn by successive *interactions* with the environment, as described in the following.

3.3.1.1 Interaction between the agent and the environment

At each time step $t \in \mathbb{N}^+$, the agent receives the state $s_t \in \mathcal{S}$ of the environment among the state space $\mathcal{S}$. Then, the agent chooses an action $a_t \in \mathcal{A}$ among the action space $\mathcal{A}$. The choice of action a_t is dictated by its policy π:

$$a_t = \pi(s_t) \tag{3.9}$$

with the policy π being a behavior function which returns the action $a \in \mathcal{A}$ to choose for each state $s \in \mathcal{S}$:

$$\pi : \mathcal{S} \rightarrow \mathcal{A} \tag{3.10}$$

As a response, the agent receives a reward r_t from the environment according to the reward function R:

$$r_t = R(s_t, a_t) \tag{3.11}$$

It also receives the new state of the environment $s_{t+1} \in \mathcal{S}$ according to the state transition model P:

$$s_{t+1} \sim P(s_t, a_t, .) \tag{3.12}$$

Fig. 3.3 illustrates an interaction $< s_t, a_t, r_t, s_{t+1} >$ at time t between the agent and the environment. Generally speaking, an interaction refers to the tuple <state, action, reward, new state>. The interactions between the agent and the environment go on until the agent reaches a terminal state, noted s_T.

The set of interactions between an agent and the environment, starting from an initial state s_1 to a terminal state s_T is an episode e:

$$e =< s_1, a_1, r_1, s_2, a_2, r_2, s_3, \cdots, s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}, \cdots, s_T > \tag{3.13}$$

In RL, the agent learns through trial and error during episodes of training.

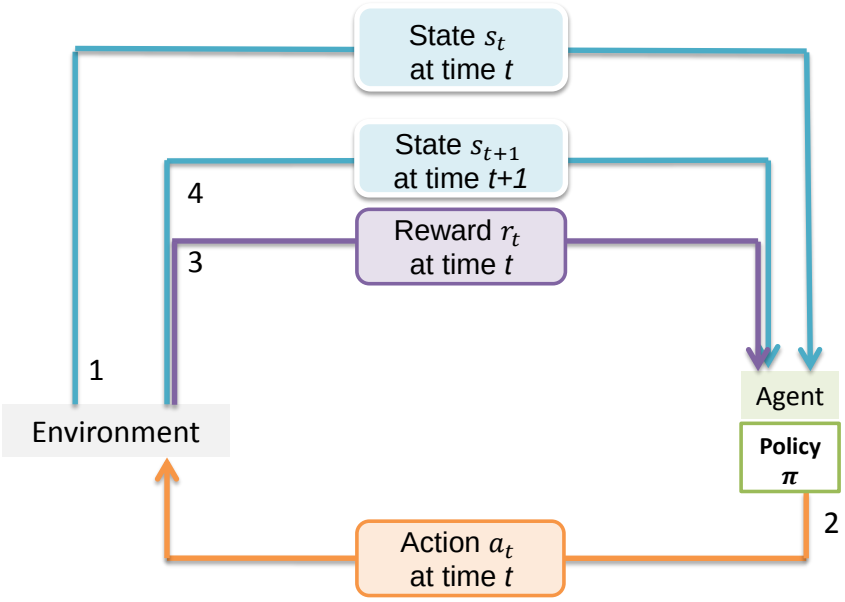

Figure 3.3: **Interaction at time t between the agent and the environment in a RL framework.** (1) The environment gives state s_t to the agent. (2) The policy π of the agent determines the action a_t to choose. In response, the environment simutaneously gives (3) a reward r_t and (4) the new state s_{t+1}. The sequence of steps 1-2-3-4 forms one interaction.

3.3.1.2 Return, value, action value

In the RL framework, we define the *return* g_t at each time step $t \in \mathbb{N}^+$ as the sum of immediate reward r_t plus future discounted rewards $\gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots$:

$$g_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{3.14}$$

$\gamma \in [0, 1]$ is the discount factor balancing immediate rewards versus future rewards.

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

$$V_\pi(s) = \mathbb{E}_\pi[g_t \mid s_t = s] \tag{3.15}$$

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 g_t the agent can hope to get by picking action a in state s and then following its policy π:

$$Q_\pi(s, a) = \mathbb{E}_\pi[g_t | s_t = s, a_t = a] \tag{3.16}$$

The Q-function indicates, for a given policy π, if selecting an action a in a particular state s is likely to have good repercussions in the following steps by getting large rewards or not. For two actions a_1 and a_2, the Q-function indicates which action is the optimal choice to make. If $Q_\pi(s, a_1) > Q_\pi(s, a_2)$, then action a_1 is a better choice to make when the agent is in state s than action a_2 because it is expected to lead to larger rewards. Another possible expression of the Q-value $Q_\pi(s, a)$ is given by the Bellman equation [97]:

$$Q_\pi(s, a) = \mathbb{E}_\pi[r_t + \gamma Q_\pi(s_{t+1}, \pi(s_{t+1}))|s_t = s, a_t = a] \tag{3.17}$$

This expression allows to express the action value of a state s_t in function of the action value of its following state s_{t+1} in a recursive form. If the action value of s_{t+1} is known, then the action value of s_t can easily be calculated by only knowing the reward r_t received in s_t for choosing a_t. This expression is at the basis of many RL algorithms [67, 68, 97, 104].

3.3.1.3 Environment description as a Markov Decision Process

An environment for RL is formally described by a Markov Decision Process (MDP). To learn an optimal policy π^* in an environment means to solve its MDP.

Definition 3.3.1 (Markov Process)**.** A Markov Process is a stochastic model which describes a sequence of random variables $s_1, s_2, s_3, \cdots$ that are related to each other.
A *first*-order Markov Process verifies the Markov property: it assumes that the future is independent of the past given the present, such that $\mathbb{P}(s_t|s_{t-1}, , \cdots, s_1) = \mathbb{P}(s_t|s_{t-1})$.
A m-order Markov Process assumes that the future depends on the m past states: $\mathbb{P}(s_t|s_{t-1}, \cdots, s_{t-m}, \cdots, s_1) = \mathbb{P}(s_t|s_{t-1}, \cdots, s_{t-m})$.

Definition 3.3.2 (Markov Decision Process)**.** A Markov Decision Process is defined by the tuple $(\mathcal{S}, \mathcal{A}, R, P, \gamma)$ with:

- $\mathcal{S}$ the state space. It contains all the possible states of the environment. The state of the environment at time t is $s_t \in \mathcal{S}$. In a MDP, all states are Markov (Def. 3.3.1, such that $\mathbb{P}(s_{t+1}|s_t) = \mathbb{P}(s_{t+1}|s_1, \cdots, s_t)$.

- $\mathcal{A}$ the action space. It contains all the possible actions that can be chosen in the environment.

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ the reward function. It gives the received reward according to the state of the environment and the action chosen in this state.

- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \longrightarrow [0, 1]$ the state transition model. It gives the probability of transitioning from a state to another in the environment depending on the choice of action. $P(s, a, s')$ gives the probability of transitioning from state s to state s' when the action is a.

- $\gamma \in [0, 1]$ the discount factor. It is used to discount the future rewards when calculating the value of a state (Eq. 3.15). It represents the importance that is given to the present compared to the future. If $\gamma = 0$, only the present matters.

3.3.1.4 Environment description as a Partially Observable Markov Decision Process

A particular case of MDPs are Partially Observable Markov Decision Processes (POMDPs).

Definition 3.3.3 (Partially Observable Markov Decision Process)**.** A POMDP is defined by the tuple $\{\mathcal{S}, \mathcal{A}, R, P, \gamma, \mathcal{O}, \Psi\}$. It has two more components than a MDP:

- $\mathcal{O}$ the set of observations,
- $\Psi : \mathcal{S} \times \mathcal{O} \to [0, 1]$ the emission observation probabilities. $\Psi(o|s)$ gives the probability of observing $o \in \mathcal{O}$ when the state is $s \in \mathcal{S}$.

In a POMDP, the agent can not access all the information about the state $s \in \mathcal{S}$ of the environment [46, 97]. Instead, it receives *partial* or *noisy* information about the state. This partial information is an *observation* $o \in \mathcal{O}$.

Common examples of partially observable applications are robot navigation [11] and video games. For example in Atari games [42, 47, 68, 100], a single frame of the game screen contains information about the position of elements of interest (the player, the enemies, the obstacles, etc.) but it does not provide information about their movement. Using a single frame, it is not possible to know the direction and speed. To reduce partial observation, methods from the literature generally concatenate the current frame with several previous frames so that the agent can infer additional information about movement and speed.

On the subject of partial observability in Mario video game, Guillaume exclaims *"It is always annoying to fall on the Hammer brothers and receive their projectiles. If I had known earlier, I would have sent a fireball".*

In POMDPs, the policy π of the agent is no longer defined over the state space $\mathcal{S}$ (because the agent can not access all information about the states) but rather over the observation space $\mathcal{O}$:

$$\pi : \mathcal{O} \to \mathcal{A} \tag{3.18}$$

3.3.1.5 RL methods to learn an optimal policy

The optimal policy π^* is the policy which maximizes the rewards collected by the agent. In other words, it is defined as the policy maximizing the value V (see Eq. 3.15) of all states:

$$\pi^* = \arg\max_{\pi} V_\pi(s) \quad \forall s \in \mathcal{S} \tag{3.19}$$

There are many RL algorithms to learn an optimal policy π^*. [97] gives an exhaustive review of basic RL approaches. The choice of algorithm depends on the nature of the MDP. Synthetically, there are two families of methods to learn an optimal policy. The first family relates to policy-based methods [67, 98, 105] which seek to learn the policy function π (Eq. 3.9) directly. The second family relates to value-based methods [44, 68, 102, 104] which seek to learn the optimal action-value function Q^*, defined as the maximum action-value function Q (Eq. 3.16) over all policies:

$$Q^*(s, a) = \max_{\pi} Q_\pi(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{3.20}$$

Value-based methods use the fact that, if the optimal action value function Q^* is known, then an optimal policy π^* can be inferred by acting greedily over Q^*, i.e. by choosing the actions maximizing the Q-values:

$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s,a) \quad \forall s \in \mathcal{S} \tag{3.21}$$

3.3.1.6 Deep Reinforcement Learning

A MDP is *finite* when both its set of state $\mathcal{S}$ and set of actions $\mathcal{A}$ have a finite number of elements. Additionally, a MDP is *small* when the number of elements is small. When the MDP is *finite* and *small*, the policy π of the agent is a lookup table with as many rows as there are states in $\mathcal{S}$ and as many columns as there are actions in $\mathcal{A}$ [97]. The table can contain all possible combination of state and action pairs. However in many real-world applications, MDPs are *large* or *infinite* [58]: there are a large or infinite number of elements either in the state space or in the action space. The policy can not be represented by a table and it is necessary to approximate the policy (resp. the action value) with a parametrized function π_Θ (resp. Q_Θ) defined over some parameters Θ [97, 98].

Deep Reinforcement Learning (DRL) relates to the specific case of RL using function approximator where the policy π or the action-value function Q are approximated by a Deep Neural Network [58, 67, 68]. In the literature, DRL is used to solve various types of problems. [58] gives a review on DRL. A few examples of DRL applications are video games [42, 44, 47, 68, 73, 100], robot navigation [11], medical diagnosis [80], and vehicle driving [91].

3.3.2 Temporal sequence acquisition: a Markov Process

Random variable Sequential data are generally derived from a stochastic variable. Given the value of a data point at a specific time step, we cannot be sure of the next value that will be collected. A same data point can lead to various future acquisitions and the distribution of the random variable is unknown. This is for example the case in predictive maintenance, for which the signals from the machines' sensors are not deterministic. In particular, the industrial application of the thesis is derived from a biological process which involves biological variability, and also non-deterministic evolution of the MTS that represent the living organisms behavior over time.

Markov property Temporal sequences $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ (Def. 2.1.1) are dynamic data for which data points $\mathbf{x}_t$ are acquired over time $t \in \mathbb{N}^+$. They are sequences of data points that are time-dependent and each data point $\mathbf{x}_t$ can be dependent on the previous data points $\{\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \cdots\}$. Because of the time dependence in data, it is natural to assume that recent past data points in a temporal sequence are more relevant than distant past data points to predict the future of the sequence. Consequently, temporal sequence acquisition is close to a Markov Process and we consider that temporal sequences can be expressed as high-order discrete-time Markov Processes (Def. 3.3.1) such that $\exists \eta \geq 1, \mathbb{P}(\mathbf{x}_t|\mathbf{x}_{t-1}, \cdots, \mathbf{x}_1) = \mathbb{P}(\mathbf{x}_t|\mathbf{x}_{t-1}, \cdots, \mathbf{x}_{t-\eta})$.

Rather that considering single data points $\mathbf{x}_t$ at each time step t, if we consider that the process is described by the prefixes $\mathbf{X}_{:t} = (\mathbf{x}_1, \cdots, \mathbf{x}_t)$ at each time step t, then the process

verifies the first order Markov assumption (Def. 3.3.1):

$$\mathbb{P}(\mathbf{X}_{:t}|\mathbf{X}_{:t-1}, \cdots, \mathbf{X}_{:1}) = \mathbb{P}(\mathbf{X}_{:t}|\mathbf{X}_{:t-1}) \tag{3.22}$$

where the prefix $\mathbf{X}_{:t}$ contains all the information about the past. The process of temporal sequence acquisition verifies the Markov Property and is therefore a Markov Process (Def. 3.3.1).

3.3.3 EC problem: a Markov Process with actions

In this work, we frame EC as a sequential decision-making problem. We consider an early classifier as an algorithm allowed to make decisions. While sequentially receiving data points in the sequence, the algorithm analyzes the partial sequence and has to take an action. It can *predict a label* based on the consideration that it collected enough information to identify discriminant patterns in the sequence, or it can *wait* for additional information in the future data points.

At time step $t \in \mathbb{N}^+$, the prefix $\mathbf{X}_{:t}$ continue to be observed if and only if:

1. the action a_t is to wait ($a_t = a_d$), and,
2. the sequence is not fully completed yet ($t < T$ with T the maximal length of sequences).

Otherwise, the early classifier predicts a label, $a_t \in \mathcal{A}_c$ and the process reaches a terminal state, as illustrated in Fig. 3.1. At each time step $t \in \mathbb{N}^+$, the Markov Process of temporal sequence acquisition is then conditioned on the choice of action a_t. The future of a state is dependent on its present and on the action. EC can be described as a Markov Process with actions.

3.3.4 EC trade-off: rewards in a Markov Decision Process

In Sec. 3.3.2 and Sec. 3.3.3, we showed that EC can be described as a Markov Process with actions. Since a MDP is a Markov Process with actions and rewards (Definition 3.3.2), we can define rewards associated to the EC problem and it will be described by a MDP. Consequently, we will be able to solve it with RL (Sec. 3.3.1). In the following of this work, we propose to complete the description of EC as MDP which is partially observable (Sec. 3.4), noted EC-POMDP, and train an agent for EC with DRL (Chap. 4).

The advantage of describing EC by a MDP is that we can set the rewards definition that best describes our objectives while taking into account the thesis specificities. We remind that the classification problem is time-sensitive (Sec. 2.3.1) and example-dependent cost-sensitive (Sec. 2.2.2.4). We will show in Sec. 3.4.2 how to encode the competitive objectives of early cost-sensitive classification into the reward function definition of the MDP:

- We will introduce a reward function $R = R_c + \lambda R_d$ (Eq. 3.28) balancing the competing costs of accuracy and earliness.
- We will define R_d as the reward function dedicated to delay actions and used to motivate early actions of classification.
- We will define R_c as the reward function dedicated to classification actions and used to assess the quality of classification. We will show how to handle the exemple-dependent costs of classification (Eq. 2.8) in the rewards for classification.

- We will define λ as the trade-off parameter allowing the user to directly set the degree of importance of time in the final prediction for his application.

3.3.4.1 Conclusion

In this section, we showed that the EC problem can be transposed into a RL framework through the definition of a MDP. We introduced the strategy of balancing the competitive objectives of EC in the reward definition of the MDP.

3.4 Proposition of EC-POMDP

In this section we define a Partially Observable Markov Decision Process for EC (EC-POMDP). We remind that a POMDP (Def. 3.3.3) is a mathematical description of an environment for RL, and is defined by the tuple $\{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ where:

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the action space,
- P is the transition model,
- R is the reward function,
- $\mathcal{O}$ is the observation space,
- Ψ gives the emission observation probabilities, and
- γ is the discount factor.

The environment here refers to the process of acquiring a sequence over time until classification, and each element of the EC-POMDP is defined below.

Fig. 3.4 adapts the EC problem described as a sequential decision-making problem in Fig. 3.1 to its formalization as an EC-POMDP.

3.4.1 States, observations, actions

States The state $s \in \mathcal{S}$ of the environment is characterized by the tuple $(\mathbf{X}, l, t)$:

$$s = (\mathbf{X}, l, t) \tag{3.23}$$

where:

- $(\mathbf{X}, l) \in \mathcal{D}$ is a pair of temporal sequence $\mathbf{X} = (\mathbf{x}_1, \cdots, \mathbf{x}_T)$ and its true label l from the training dataset $\mathcal{D}$. $\mathbf{X}$ is the complete sequence that is being acquired and which is sampled from a random variable with an unknown distribution.
- $t \in [1, T]$ is the number of time steps observed in the sequence, such that $(\mathbf{x}_1, \cdots, \mathbf{x}_t)$ has been observed, and $(\mathbf{x}_{t+1}, \cdots, \mathbf{x}_T)$ is still non-observed. In other words, it is the acquisition time of the process.

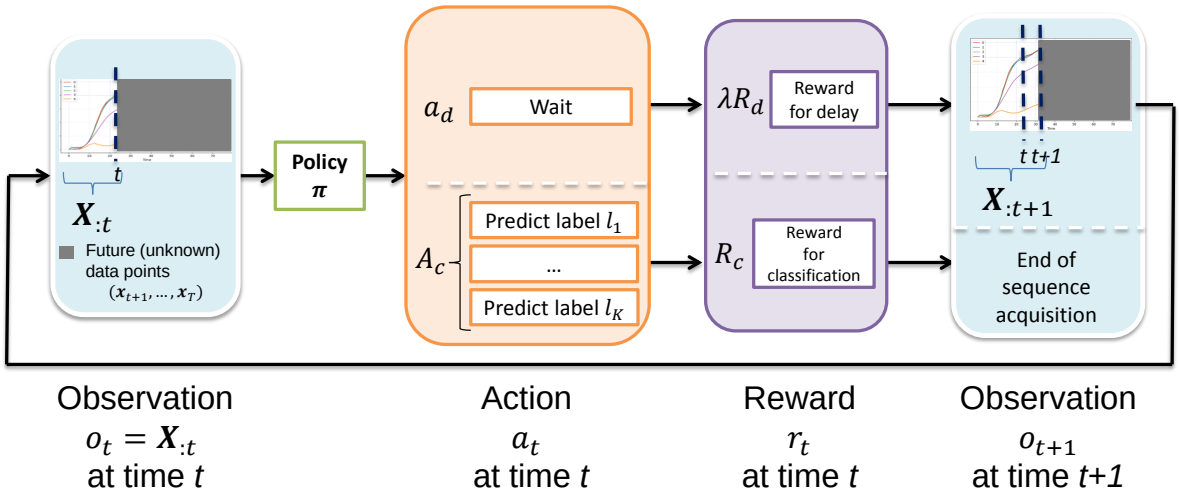

Figure 3.4: **EC-POMDP.** Illustration of the decision-making process for EC at time t. The agent observes o_t, i.e. the partial sequence composed of its first t data points. Following its policy π, it chooses an action a_t between delay or one of the K possible predictions of label. As a consequence, it receives a reward r_t. Either the process terminates or it receives next observation o_{t+1}, i.e. the partial sequence with an additionnal data point.

In the specific case of the industrial application, given the nature of the data which are continuous real-valued MTS, there is an infinite number of states.

Since the objective is to predict labels $l \in \mathcal{L}$ as early as possible, in real-life applications we do not have access to the full state information. Indeed, the label l and future data points $(\mathbf{x}_{t+1}, \cdots, \mathbf{x}_T)$ are unknown to the agent. We say that the states $s \in \mathcal{S}$ are *hidden* and we only observe *partial* information on the states. Consequently, the MDP is said to be partially observable (POMDP, see Sec. 3.3.1.4) and the agent will receive *observations* instead of full state information.

Observations The state $s = (\mathbf{X}, l, t)$ issues the observation o which is the partial sequence of data points $\mathbf{X}_{:t} = (\mathbf{x}_1, \cdots, \mathbf{x}_t)$ collected from $\mathbf{X}$ until time t:

$$o = \mathbf{X}_{:t} \tag{3.24}$$

The emission observation probabilities Ψ is therefore defined by:

$$\Psi(o|s = (\mathbf{X}, l, t)) = \begin{cases} 1 & \text{if } o = \mathbf{X}_{:t} \\ 0 & \text{elsewise} \end{cases} \tag{3.25}$$

As a consequence, the policy π of the agent is no longer defined over the state space $\mathcal{S}$ but rather over the observation space $\mathcal{O}$ which is composed of prefixes of temporal sequences $\mathbf{X}$ and is therefore continuous. We will see in the next chapter that this particularity will lead us towards the choice of DRL in order to solve the EC-POMDP.

Actions $\mathcal{A}$ is the action space. It is defined in Eq. 3.1, such that $\mathcal{A} = \mathcal{A}_c \cup a_d$ with a_d the action of delaying the prediction to collect an additional data point and $\mathcal{A}_c$ the set of classification actions. We precise that the action of delay a_d can only be chosen if there are some data points that are still not observed in the sequence $\mathbf{X}$, i.e. if the prefix did not reach the maximal length T:

$$\mathcal{A}(s) = \begin{cases} \mathcal{A}_c \cup a_d & \text{if } s = (\mathbf{X}, l, t), t < T \\ \mathcal{A}_c & \text{if } s = (\mathbf{X}, l, t), t = T \end{cases} \tag{3.26}$$

In Eq. 3.2, we defined the set of classification actions over the set of labels, such that $\mathcal{A}_c = \mathcal{L}$. The model can therefore predict all labels l_k, $k = 1..K$ from the label set $\mathcal{L}$. If $a = l_k$, then the action is to predict label l_k from $\mathcal{L}$. However, we point out that the user can define the set of classification actions differently. He may wish to merge some labels into a single class or not to predict certain classes at all: $\mathcal{A}_c \subset \mathcal{P}(\mathcal{L})$ with $\mathcal{P}(\mathcal{L})$ the powerset of $\mathcal{L}$.

In all cases, the action space is finite and small because there is a small number of possible actions. We will see in the next chapter that this particularity will lead us towards a value-based method for the EC-POMDP resolution.

Dynamics In real-life EC applications, the acquisition of data points is often costly and has to be shortened as much as possible. Once the system decides to perform classification ($a \in \mathcal{A}_c$), data points are no longer collected. We therefore define the state transition model $P : \mathcal{S} \times \mathcal{A} \longrightarrow \mathcal{S}$ by:

$$P((\mathbf{X}, l, t), a) = \begin{cases} \emptyset & \text{if } \{a \in \mathcal{A}_c\} \cup \{t = T\} \\ (\mathbf{X}, l, t+1) & \text{if } a = a_d \end{cases} \tag{3.27}$$

If the agent chooses to wait and if the sequence is not completed yet, the next state of the environment is characterized by the pair $(\mathbf{X}, l)$ and incremented time step $t + 1$. Otherwise, the agent reaches a terminal state and the process is ended.

By definition, state transitions are deterministic because the states contain all the information about the sequence that is being acquired. However, we wish to point out that, in real-life applications at test time, we do not have access to the full state information. The label and future data points are unknown and consequently the agent receives the partial incomplete sequence only. It cannot know for sure the next observations it will receive, because it does not know the distribution of the random variables that contribute to the data collection. Observation transitions are therefore stochastic and the environment is also stochastic.

Discount factor We remind that the discount factor γ is defined between 0 and 1. When $\gamma < 1$, rewards are discounted and more importance is given to immediate rewards. Also, for episodic environments with short horizons, the cumulative reward is finite and γ can be set

to 1. In the particular case of the thesis, environments for EC have horizon of size T which is the maximal length of sequences. When setting a value for the discount factor in experiments, we will therefore consider values both equal or inferior to 1.

3.4.2 Rewards

The reward function is part of the environment and is defined in the POMDP (Def. 3.3.3). Its definition is a major task in the formalization of the EC-POMDP and it has to represent the competitive objectives of classification earliness and quality. Indeed, when an agent is trained with RL, its behavior evolves according to the rewards it receives because the agent tries to maximize them. Since there is no direct supervision in RL – no supervisor tells the agent which action to choose – the only feedback is brought by the scalar rewards [97]. A wrong reward function definition can cause the agent to behave sub-optimally such as the bicycle problem [84] presented below. In this section, we review some major work from the literature related to the definition of rewards in RL. We then propose a definition of rewards for the EC problem which allows to encode the EC trade-off.

3.4.2.1 Related work on rewards in Reinforcement Learning

The definition of rewards in a MDP describing a sequential decision-making problem is not always straightforward. In the context of game environments, scalar scores are often provided by the emulator to the player during a game, and it is a common practice to use these scalar scores as rewards for the RL framework [42, 47, 67, 68, 100]. However, many real-world applications do not have innate rewards and there is a need to *design* a reward function for the task at hand, and therefore the problem description as a MDP. Furthermore, some real-world applications involve complex tasks which can hardly be described by simple reward functions. In the following, we present two major strategies from the literature commonly employed to define reward functions in a RL framework.

Learning from human demonstrations and/or feedback In some cases, the tasks can be performed by humans and a solution from the literature is Inverse Reinforcement Learning (IRL) [1, 27, 91, 106]. This approach refers to learning from experts. The idea is to observe an expert acting in the environment, collect a dataset of human demonstrations, and first learn the reward function that could explain the behavior of the expert and his objectives. The behavior of the expert is supposed to be near optimal and has to represent a target policy for the agent. Then, once the reward function is learned, the MDP can be solved with standard RL algorithms. IRL therefore differs from Behavior Cloning which uses human demonstrations for SL and does not seek to learn a reward function. In [27], the authors propose an IRL framework able to handle large MDPs with unknown dynamics.

Among alternative approaches to IRL, the authors in [19] introduce a method for the specific cases where there is no demonstrator of the task in question, but it is possible to compare two behaviors of the agent and indicate which is the best. **However, both IRL and the solution from [19] are limited by the need for human demonstrations or feedback which are not available in the case of the thesis.**

Potential-based reward shaping When the application involves complex tasks for which the reward function is not easy to design, a solution is to use potential-based reward shaping [75]. This method allows to incorporate domain knowledge into rewards definition and traditionally uses intermediate rewards to guide an agent towards its goal, and avoid sparse feedback from the environment. These intermediate rewards are related to a change of *potential* when moving from a state in the environment to another state. In [22], the authors investigate the use of potential functions that can vary over time. In [99], the authors give an online feedback to the agent during its training through reward shaping.

The use of potential-based reward shaping leads to hand-engineered definition of rewards which is generally tuned through experiments until the agent reaches a satisfying behavior for the application. As an example, in [84], the authors teach an agent to drive a bicycle towards a goal. They experimentally observed that by rewarding the agent to approach the goal, it learned to drive in cycle around the goal. They modified rewards by incorporating a punishment for driving away from the goal and the agent learned to reach the goal. This solution can thus take time and resources to be properly used.

In Sec. 3.4.2, **we will introduce a reward function inspired by reward shaping** where sparse rewards are replaced by smaller rewards given at each choice of action.

3.4.2.2 Reward function: trade-off between classification quality and earliness

In this work, the objective is to predict labels on incomplete sequences as early as possible while maintaining an acceptable quality of classification (Sec. 2.3). To this end, we propose to encode the trade-off between classification quality and earliness in the definition of the reward function R associated to the EC-POMDP. We seek to define rewards in a way that they guide the agent towards fast and accurate predictions.

Following previous definitions of states and actions, $R(s, a)$ is the scalar reward given to the agent for taking action $a \in \mathcal{A}_c \cup a_d$ in state $s = (\mathbf{X}, l, t)$. In order to balance between the two competitive objectives, we choose to reward the agent separately for its actions of classification $a \in \mathcal{A}_c$ and the delay action a_d:

$$R((\mathbf{X}, l, t), a) = R_c((\mathbf{X}, l, t), a) + \lambda R_d((\mathbf{X}, l, t), a) \tag{3.28}$$

R_d is the reward function for the delay action. R_c is the reward function for classification actions. In other words, the delay action a_d is ignored in R_c, and classification actions $a \in \mathcal{A}_c$ are ignored in R_d:

$$R_c((\mathbf{X}, l, t), a = a_d) = 0 \tag{3.29}$$

$$R_d((\mathbf{X}, l, t), a \in \mathcal{A}_c) = 0 \tag{3.30}$$

$\lambda \in \mathbb{R}^+$ is a parameter setting the trade-off between the two objectives of classification earliness and quality. It allows the user to control the compromise he is willing to make between speed and accuracy. The more earliness is important in comparison to accuracy, the larger λ has to be. Generally, the will to compromise is application-dependent and the user can set λ to his preference.

In the following, we will give some possible definitions of R_d and R_c. The general proposition is to give null or negative rewards in R_d when the agent delays the prediction (Sec. 3.4.2.4), and to give rewards that are related to the quality of its classification in R_c when the agent predicts a label (Sec. 3.4.2.3).

3.4.2.3 Reward function for classification actions

We first define rewards when $a \in \mathcal{A}_c$ is an action of classification. In Sec. 2.2.2.4, we made the distinction between classification problems that are cost-sensitive (i.e. misclassification induces costs depending on the true label and the predicted label), example-dependent cost-sensitive (i.e. misclassification induces costs depending on the true label, the predicted label and some additional information on the data) and those that are cost-insensitive (i.e. all types of classification errors are equally important). In this work, we define a reward function for each of these cases.

Cost-insensitive learning We define $R_{c,ins}$ as the reward function for classification actions in a cost-insensitive learning framework. We choose to reward the classification actions according to the accuracy of the predicted label. When the predicted label in a matches the true label l associated to a sequence $\mathbf{X}$, we give a positive or null reward r_+. On the contrary when the predicted label in a differs from the true label l, we give a negative reward r_-:

$$R_{c,ins}((\mathbf{X}, l, t), a) = \begin{cases} r_+ \geq 0 & \text{if } a = l \\ r_- < 0 & \text{if } a \in \mathcal{A}_c \setminus \{l\} \end{cases} \tag{3.31}$$

We point out that an objective can be encoded by several reward functions. Indeed, for a same objective of fast prediction using as few features as possible, the agent is rewarded positively with a score $r_+ = +1$ if the classification is correct in [80] while it receives null reward, $r_+ = 0$, for correction classification and negative rewards, $r_- = -1$, for incorrect classification in [49]. In future chapters, during experimental evaluations, we will use several values of r_+ and r_-, and these values will become hyper-parameters of the method that need to be tuned.

Cost-sensitive learning We define $R_{c,sen}$ as the reward function for classification actions in a cost-sensitive learning framework. We choose to reward misclassification actions depending on the cost carried by classification errors:

$$R_{c,sen}((\mathbf{X}, l, t), a) = \begin{cases} r_+ \geq 0 & \text{if } a = l \\ -c_{l,a} & \text{if } a \in \mathcal{A}_c \setminus \{l\} \end{cases} \tag{3.32}$$

with $c_{l,a}$ the cost of predicting label a on a sample with true label l, defined in Sec. 2.2.2.4. When the predicted label in a matches the true label l associated to the sequence $\mathbf{X}$, we employ the same strategy than with cost-insensitive learning and give a positive or null reward $r_+ \geq 0$.

Example-dependent cost-sensitive learning We define $R_{c,ed-sen}$ as the reward function for classification actions in an example-dependent cost-sensitive learning framework, and we consider that the costs of classification may vary between samples:

$$R_{c,ed-sen}((\mathbf{X}^n, l^n, t), a) = \begin{cases} r_+ \geq 0 & \text{if } a = l^n \\ -c_{n,l^n,a} & \text{if } a \in \mathcal{A}_c \setminus \{l^n\} \end{cases} \tag{3.33}$$

with $c_{n,l^n,a}$ the cost of predicting label a on the n-th sample with true label l, defined in Sec. 2.2.2.4. A correct classification yields a positive or null reward $r_+ \geq 0$.

Depending on the application, we will use one of the above definition: $R_c \in \{R_{c,ins}, R_{c,sen}, R_{c,ed-sen}\}$. Rewards for classification actions are summarized in Tab. 3.1.

R_c	Definition	$R_c((\mathbf{X}^n, l^n, t), a \in \mathcal{A}_c \setminus \{l^n\})$	$R_c((\mathbf{X}^n, l^n, t), a = l^n)$
$R_{c,ins}$ (Cost-insensitive)	All classifications errors are equally important.	$r_- < 0$	$r_+ \geq 0$
$R_{c,sen}$ (Cost-sensitive)	$c_{l^n,a}$ is the cost of predicting label a when the true label is l^n.	$-c_{l^n,a}$	$r_+ \geq 0$
$R_{c,ed-sen}$ (Example-dependent cost-sensitive)	$c_{n,l^n,a}$ is the cost of predicting label a when the true label is l^n regarding the n-th individual of $\mathcal{D}$.	$-c_{n,l^n,a}$	$r_+ \geq 0$

Table 3.1: **Reward function definition R_c for classification actions $a \in \mathcal{A}_c$.** $(\mathbf{X}^n, l^n)$ is the n-th sample pair from $\mathcal{D}$.

3.4.2.4 Reward function for delay action

We then define rewards when the actions is delay a_d. To encode the objective of earliness, we propose the two following strategies.

Reward shaping First, we propose to 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 [75, 97]. Therefore, 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.

We define $R_{d,shape}$ as the reward function for delay under the shaping strategy:

$$R_{d,shape}((\mathbf{X}, l, t), a_d) = -c(t) \tag{3.34}$$

with $c : [0, T] \to \mathbb{R}^+$ the cost function of delaying the prediction at time t.

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 data points and knowledge the agent has about the sequence, the worst it is to delay. As a consequence, we propose that the cost function c is a monotonic non-decreasing function of time. Moreover, we seek to define a simple parametrization of the cost function c, in order to derive a minimum of hyper-parameters to be optimized later. To that end, we propose a penalty increasing in time t, in the form of t^κ with $\kappa \geq 0$ the penalty coefficient. We normalize the delay reward function so that it is bounded independently of the sequence maximal length T:

$$c(t) = t^\kappa / T^\kappa \tag{3.35}$$

We point out that many other function definitions could achieve the same objective. Also, if available, including domain knowledge into the reward function can guide the agent towards

a better or faster learning. For example, if the user knows that no relevant appears before time t_{thresh}, he can set $\forall t < t_{thresh}, c(t) = 0$.

Reward discounting In the specific case where the agent is given strictly positive rewards for correct classification, such that $R_c((\mathbf{X}, l, t), a = l) > 0$ (or $r_+ > 0$), 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. We define $R_{d,discount}$ as the reward function for delay under the discounting strategy, that is the action of delay is not rewarded:

$$R_{d,discount}((\mathbf{X}, l, t), a = a_d) = 0 \tag{3.36}$$

In the following, we illustrate the effect of the discount factor γ on the policy π of the agent when it is given strictly positive rewards for correct classification, such that $r_+ > 0$. We suppose that the policy is $\pi(s) = \arg\max_a Q(s, a)$ (Eq. 3.21) and that the environment is associated to the sample pair $(\mathbf{X}, l)$. We consider three possible sequences of interactions between the agent and the environment, noted i_1, i_2, i_3, starting from time t and until the end of the episode:

- At time t, the agent chooses the delay action a_d which is not rewarded by definition of $R_{d,discount}$. It then continues to follow its policy π and we suppose that $\exists k \in \mathbb{N}^+$ for which, at time $t + k$, the agent chooses to predict the correct label l, yielding a positive reward r_+, and leading to the end of the episode.

  The first possible sequence of interactions is $i_1 =< s_t, a_d, 0, s_{t+1}, a_d, 0, \cdots, s_{t+k}, l, r_+ >$.

- At time t, the agent predicts the correct label l, yielding a positive reward r_+ and leading to the end of the episode.

  The second possible sequence of interactions is $i_2 =< s_t, l, r_+ >$.

- At time t, the agent predicts a wrong label in $\mathcal{A}_c \setminus \{l\}$, yielding a negative reward r_- and leading to the end of the episode.

  The third possible sequence of interactions is $i_3 =< s_t, a_t \in \mathcal{A}_c \setminus \{l\}, r_- >$.

In Eq. 3.16, we defined the action value function $Q_\pi(s, a)$ as the expected return $g_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, starting from state s, taking action a and then following policy π such that $Q_\pi(s, a) = \mathbb{E}_\pi[g_t | s_t = s, a_t = a]$. As a consequence, in the first possible sequence of interactions i_1, the return is $g_t = \gamma^k r_+$. In the second possible sequence of interactions i_2, the return is $g_t = r_+$. In the third possible sequence of interactions i_3, the return is $g_t = r_-$. The Q-values associated to each action a_t at time t are then:

$$Q(s_t, a_t) = \begin{cases} \gamma^k r_+ & \text{if } a_t = a_d \\ r_+ & \text{if } a_t = l \\ r_- & \text{if } a_t \in \mathcal{A}_c \setminus \{l\} \end{cases}$$

If $\gamma < 1$, then $Q(s_t, a_t \in \mathcal{A}_c \setminus \{l\}) < Q(s_t, a_d) < Q(s_t, l)$ and therefore $\pi(s_t) = l$. Setting the discount factor strictly lower than 1 ($\gamma < 1$) causes $\gamma^k r_+ < r_+$ and consequently motivates the agent to take early actions of classification.

R_d	Definition	Discount factor γ	$R_d((\mathbf{X}, l, t), a_d)$
$R_{d,shape}$ (Shaping)	Delay is penalized increasingly over time. $c(t)$ is the cost function of delaying the prediction at time t, a monotonic non-decreasing function of time.	$\gamma \leq 1$	$-c(t)$
$R_{d,discount}$ (Discounting)	Delay is not penalized. Rewards are discounted in the MDP definition. The agent must be given positive rewards for correct classification, $R_c((\mathbf{X}, l, t), a = l) > 0$.	$\gamma < 1$	0

Table 3.2: **Reward function definition R_d for delay action a_d.**

Rewards for delay action are summarized in Tab. 3.2, with $R_d \in \{R_{d,discount}, R_{d,shape}\}$. In Appendix A, we conduct an experimental evaluation to compare the proposed definitions of the delay reward function, i.e. reward shaping against reward discounting.

3.4.3 Specificities of the EC-POMDP

All but one of the actions terminate the episode. Actions are either to predict a label $l \in \mathcal{A}_c$ or to delay prediction: $\mathcal{A} = \mathcal{A}_c \cup a_d$. Since we terminate the acquisition of new data points 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:

$$\mathbb{P}(s_t \neq terminal) = \underbrace{\mathbb{P}(a_1 = a_d)}_{\leq 1}\underbrace{\mathbb{P}(a_2 = a_d)}_{\leq 1}...\underbrace{\mathbb{P}(a_{t-1} = a_d)}_{\leq 1} = \prod_{j=1}^{t-1}\underbrace{\mathbb{P}(a_j = a_d)}_{\leq 1} \tag{3.37}$$

Actions of classification are rarer than the delay action. 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.

3.5 Conclusion

In this chapter, we framed EC as a sequential decision-making problem where the model can decide at all time steps $t \in [1, T]$ to perform classification on the partial sequence $\mathbf{X}_{:t}$ or to delay classification in order to get additional data points.

We showed that solving the sequential decision-making problem with SL brought along a new challenge: to build a training labelled dataset with supervision on both classification and delay actions at all time steps in the sequences. We argued that the methods from the

literature were not directly applicable and therefore the resolution of the problem with SL was a major research topic.

We showed that EC can be described as a Markov Process with actions. We defined rewards associated to EC so that the decision-making problem can be described by a MDP and then solved within a RL framework. We proposed several strategies in the reward function definition to compromise between classification earliness and accuracy, through strategies of reward shaping and reward discounting. The solution allows the user to set the relative importance of time compared to classification quality for his application. Several definitions of rewards were proposed depending on the nature of the classification problem (cost-sensitive versus cost-insensitive) and the solution allows to involve the classification costs introduced in Sec. 2.2.2.4.

We showed that the MDP for EC was actually partially observable because during the online acquisition of sequences, we do not have access to future data points or the label to predict, while these information will be used by the environment during the training of the agent.

The mathematical framework has been proposed for the general problem of EC on temporal sequences and it can therefore be applied to different types of sequences. In the following, we will propose a solution to the EC-POMDP resolution and we will evaluate if the solution policy achieves the objectives of EC.

Chapter 4

EC-POMDP solving with Deep Reinforcement Learning

In Chap. 3, we defined early classification (EC) as a sequential decision-making problem and we described it by a Partially Observable Markov Decision Process (POMDP), noted EC-POMDP.

In this chapter, we aim at solving the EC-POMDP by finding its optimal policy π^* (Eq. 3.19). Specifically, we will train an agent with Reinforcement Learning (RL) in an environment described by the EC-POMDP — an *early classifier agent*. We seek to assess whether the early classifier agent can achieve objectives of EC as defined in Sec. 2.3.

In order to train the early classifier agent with RL, the first question to answer is whether to use a policy-based approach or a value-based approach (see Sec. 3.3.1.5). The action space $\mathcal{A}$ (Eq. 3.26) of the EC-POMDP is finite and small ($K+1$ actions), which makes it possible to learn the action value for each action $a \in \mathcal{A}$. As a consequence, in the following of this doctoral work, we choose to learn the policy π of the early classifier agent with a value-based approach, i.e. by learning its Q-function (see Sec. 3.3.1.5).

In order to find the optimal policy π^* of the agent, the objective is to learn the optimal Q-function Q^* (Eq. 3.20):

$$\pi^*(o) = \arg\max_{a \in \mathcal{A}} Q^*(o, a) \; \forall o \in \mathcal{O}$$

As explained is Sec. 3.4, the environment for EC in a RL framework is partially observable. The action value function Q is then defined over the set $\mathcal{O}$ of observations (Eq. 3.25) which is composed of prefixes of temporal sequences $\mathbf{X}$. Because $\mathcal{O}$ is continuous, the action value function Q cannot be represented by a finite table with action values on all pairs of observations and actions. There is a need to estimate the action value function Q with function approximation [97].

In this doctoral work, we use a solution for action value approximation which is Deep Reinforcement Learning (DRL, see Sec. 3.3.1.6). We motivate our choice for DRL in Sec. 4.1. DRL consists in approximating the agent's action value function $Q(o, a)$ with a Deep Neural Network (DNN) $Q_\Theta(o, a)$ with parameters Θ, as illustrated in Fig. 4.1. The policy π_Θ

associated to this DNN is then defined as:

$$\pi_\Theta(o) = \arg\max_{a \in \mathcal{A}} Q_\Theta(o, a) \quad \forall o \in \mathcal{O} \tag{4.1}$$

The objective in DRL is to find the optimal parameters Θ^* of the DNN that lead to optimal action value function Q^*:

$$Q^* = Q_{\Theta^*} \tag{4.2}$$

Several research papers from the literature aim at finding the optimal parameters Θ^* of the DNN Q_Θ and, in this doctoral work, we apply an existing DRL algorithm called the Double Deep-Q-Network (DDQN) algorithm [100], presented in Sec. 4.2.

In Sec. 4.3, we conduct some experimental evaluations to evaluate the method and to compare its performance to state-of-the-art EC algorithms. We present the experimental pipeline used during this work to solve the EC-POMDP with DRL and to train an end-to-end early classifier agent.

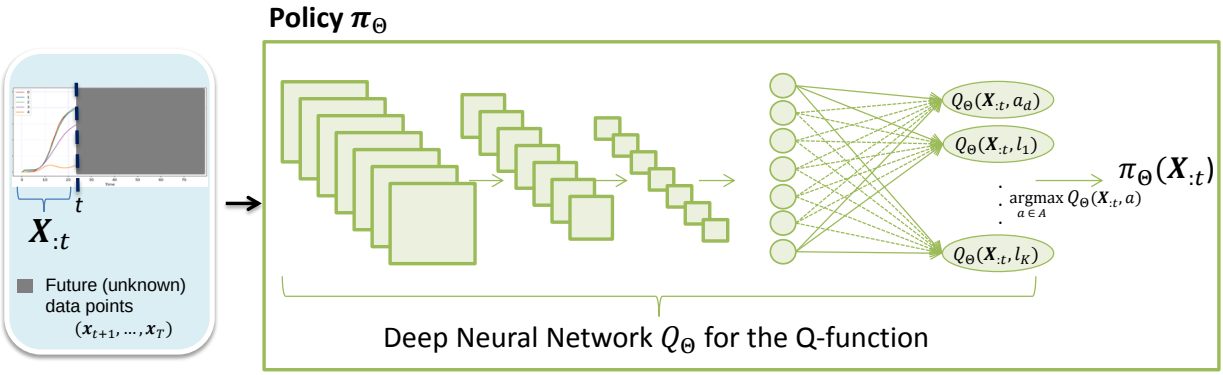

Figure 4.1: **DNN Q_Θ for the Q-function with parameters Θ.** It is defined over the set of observations $\mathcal{O}$ (Eq. 3.25). The output layer has one neuron per action $a \in \mathcal{A}$. Each output neuron predicts the Q-values of an observation $o = \mathbf{X}_{:t}$ for that action a. It is fully-connected to its previous layer and has a linear activation. The first layers are convolutional layers. The policy resulting from this DNN is $\pi_\Theta(o) = \arg\max_{a \in \mathcal{A}} Q_\Theta(o, a)$.

4.1 Motivation

We choose to approximate and learn the action value function Q with DRL for the following reasons. First, unlike policy tables which are only applicable to the states it learned from, DNNs (and more generally policies approximated by functions) are able to generalize from

seen states to unseen states. As a consequence, the use of a DNN allows to apply the policy on new data which were never seen during training.

Second, the industrial application of this thesis currently involves data of size 77×5 (with maximal length $T = 77$ and $P = 5$ features) but we wish to develop a method that could be transposed to complex sequences of images in a future work. When the observations of the environment are complex (multi-components, high-dimensional, etc.), DRL is particularly useful because the use of a DNN allows to learn a policy π on raw observations $o \in \mathcal{O}$ directly. As an example, in [68] the authors successfully learn a policy on images of size $84 \times 84 \times 4$ from video games.

Finally, DRL set the user free from a feature extraction step. Because of their ability to compose functions, DNNs can learn both low-level and high-level features on raw data and there is no need to use hand-crafted features or rules designed by experts. The autonomous learning of features for decision-making and classification makes the proposed method applicable to new data on which we have no features expertise. In Sec. 4.3, we will evaluate the method on public datasets for which we do not have any prior knowledge.

4.2 Double Deep-Q-Network algorithm

In this work, we define the agent's policy π through the Q-function and we approximate Q by a DNN $Q_\Theta(o, a)$ with parameters Θ. The objective is to find the optimal parameters Θ^* that lead to the optimal action value function Q^* (Eq. 3.20) and consequently the optimal policy π^*. To find optimal parameters Θ^*, we train the DNN Q_Θ with the Double Deep-Q-Network (DDQN) algorithm from [100], which is an adaptation of the Deep-Q-Network (DQN) algorithm from [68], itself being an evolution of the Q-learning algorithm [104].

In the following, we show how to apply DQN algorithm to the EC-POMDP, and we detail what differs in DDQN algorithm, namely the loss function used for the updates of the DNN's parameters.

4.2.1 Application of Deep-Q-Network algorithm to EC

DQN algorithm aims at approximating the optimal Q-function Q^* from episodes of training between an agent and its environment (see Sec. 3.3.1.1), when the Q-function is approximated by a DNN. It involves a RL framework such as the one presented in Fig. 3.3. The algorithm is introduced for online learning, with successive repetitions of interaction collection and policy optimization, as illustrated in Fig. 4.2.

The general principle of the algorithm is the following. The Q-function is learned from an integrated database of interactions which is built online, called the replay memory. After each interaction between an agent and its environment, the latter is stored in the replay memory and the Q-function is updated.

In particular, the application of the DQN algorithm to the EC-POMDP is the following. First, the agent's policy π_Θ is initialized with random parameters Θ. Then, the training of the agent starts (and lasts for a fixed number of training episodes) in order to update the parameters Θ. In each training episode, the environment is associated to a sample pair $(\mathbf{X}, l)$ with $\mathbf{X}$ a temporal sequence and l its reference label from the training dataset $\mathcal{D}$ (Eq. 2.3).

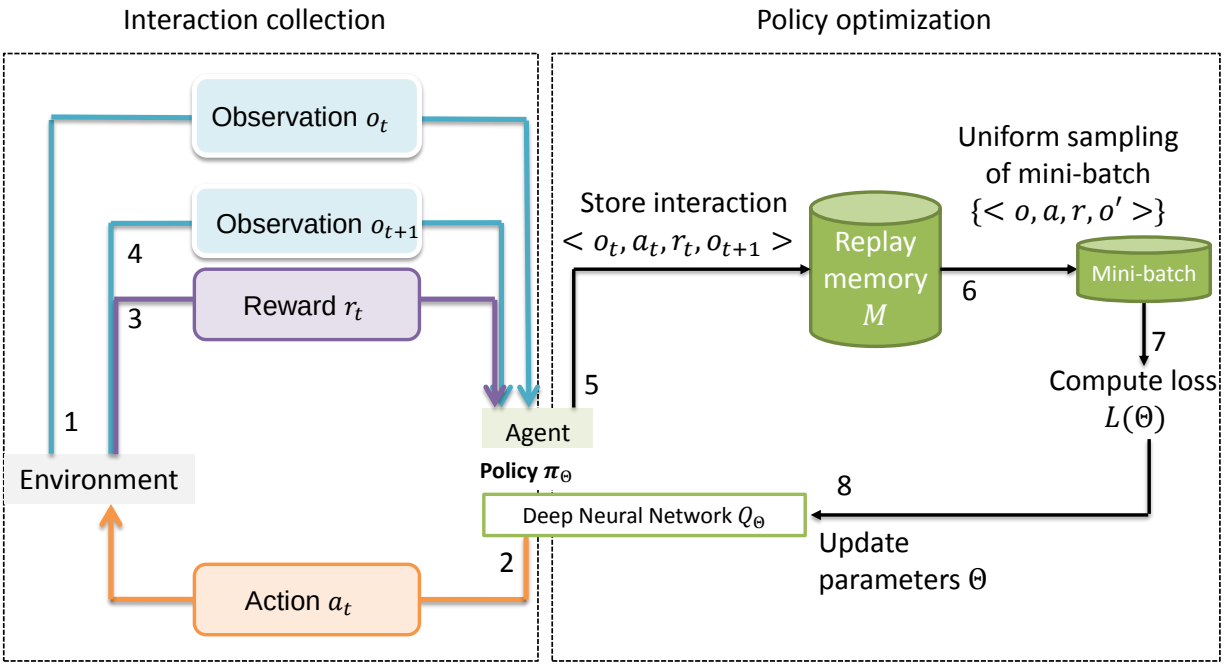

Figure 4.2: **DQN and DDQN algorithms.** Both algorithms follow the same scheme and only differ at step 7 (for the loss calculation). Steps 1 to 8 form one iteration of the algorithms. The sequence of steps 1-2-3-4 corresponds to one interaction collection between the agent and its environment in a RL framework, as illustrated in Fig. 3.3. The sequence of steps 5-6-7-8 corresponds to one update of the policy parameters Θ.
In this schematic illustration, the agent interacts at time t of the sequence acquisition process with the environment.
1: The agent receives observation $o_t = \mathbf{X}_{:t}$ (Eq. 3.25, Fig. 2.1).
2: Following its policy $\pi_\Theta(o_t) = \arg\max_{a \in \mathcal{A}} Q_\Theta(o_t, a)$ (Fig. 4.1), it chooses action $a_t = \pi_\Theta(o_t)$ with probability $1 - \epsilon$ or random action $a_t \in \mathcal{A}$ (Eq. 3.26) with probability ϵ.
3: The agent receives a reward r_t (Eq. 3.28, Fig. 3.4).
4: The agent receives next observation o_{t+1}.
5: The interaction $< o_t, a_t, r_t, o_{t+1} >$ is stored inside the replay memory $\mathcal{M}$ of the agent.
6: A mini-batch of past interactions $\{< o, a, r, o' >\}$ is uniformly sampled from the replay memory $\mathcal{M}$.
7: In DQN (resp. DDQN) algorithm, the loss from Eq. 4.3 (resp. Eq. 4.4) is computed on the mini-batch.
8: Parameters Θ of the DNN Q_Θ are updated.
The process is repeated from steps 1 to 8 starting with the new observation o_{t+1} at time $t+1$.

As illustrated in Fig. 4.2, at time step t of the training episode:

- The agent receives as observation the partial time series $o_t = \mathbf{X}_{:t}$ (Eq. 3.25).
- It has to decide between delaying the prediction ($a_t = a_d$) in order to gather more data points or making a label prediction ($a_t \in \mathcal{A}_c$), as defined in Eq. 3.26. Given its exploration rate $\epsilon \in [0, 1]$, the agent can *explore* and choose a random action from the set of actions $\mathcal{A}$ with probability ϵ. Or it can *exploit* and select the action a_t dictated

by its policy π_Θ, such that $a_t = \pi_\Theta(o_t)$, with probability $1 - \epsilon$.

- After each interaction, the exploration rate ϵ is annealed in order to encourage the agent to exploit more, the more it trains.

- Following its choice of action a_t, the agent receives a reward $r_t = R((\mathbf{X}, l, t), a_t)$ (Eq. 3.28).

- The episode terminates if there are no additional data points to collect in the sequence or if the agent chose to predict a label, $a_t \in \mathcal{A}_c$. Otherwise the episode continues and the agent receives a new observation $o_{t+1} = \mathbf{X}_{:_{t+1}}$ (Eq. 3.27).

The interaction at time step t of the training episode is then $< o_t, a_t, r_t, o_{t+1} >$. It is stored into a replay memory $\mathcal{M}$ allowing to reuse this experience later.

After each interaction, DQN algorithm applies a stochastic gradient descent to the DNN parameters Θ, with the following steps:

- It uniformly samples a mini-batch of past interactions $\{< o, a, r, o' >\}$ from the replay memory $\mathcal{M}$, with o' being the observation following the choice of action a given observation o.

- For each past interactions $< o, a, r, o' >$ from the mini-batch, the Q-value $Q_\Theta(o, a)$ for the action a on the observation o is calculated. Since this interaction has already been experienced by the agent in the past, it is known that the environment rewarded the agent by r and that the latter also received o' as a new observation. Consequently, it is possible to use this interaction to get a better estimate of the true Q-value following the Bellman equation (Eq. 3.17).

  Using the Bellman equation, DQN algorithm computes the loss function:

$$L(\Theta) = (r + \gamma \max_{a'} Q_{\Theta^-}(o', a') - Q_\Theta(o, a))^2 \tag{4.3}$$

  where γ is the discount factor (Sec. 3.3.1), Q_Θ the current Q-network and Q_{Θ^-} the target Q-network (see below).

  The loss $L(\Theta)$ seeks to minimize the difference between the Q-value predicted by the DNN, $Q_\Theta(o, a)$, and the true Q-value.

  However, the true Q-value is not known and is therefore estimated by the *Q-learning target*, $r + \gamma \max_{a'} Q_{\Theta^-}(o', a')$, which is the sum of true reward r received by the agent, plus the maximal action value $\max_{a'} Q_{\Theta^-}(o', a')$ estimated by the target DNN Q_{Θ^-} on the next observation o'.

- It then updates the DNN parameters Θ by back-propagation of the gradient calculated on the loss $L(\Theta)$.

After an update of parameters Θ, the training episode continues. Specifically, it moves to time step $t + 1$ with a new observation o_{t+1}, and the process is repeated, with the generation of a new interaction $< o_{t+1}, a_{t+1}, r_{t+1}, o_{t+2} >$ followed by the update of parameters Θ. If the agent reaches a terminal state, the episode ends and a new episode starts with another training sample $(\mathbf{X}, l)$ from $\mathcal{D}$.

Q-learning algorithm The DQN algorithm is a variant of the Q-learning algorithm [104]. In the original Q-learning algorithm [104], the Q-learning target is $r + \gamma \max_{a'} Q_\Theta(o', a')$. It is computed with the current parameters Θ that are being updated, and not with a target Q-network Q_{Θ^-}. Also, in the original Q-learning algorithm, there is no replay memory $\mathcal{M}$, and interactions $< o_t, a_t, r_t, o_{t+1} >$ are immediately used to update the parameters Θ.

In the following, we detail the strategies proposed in the DQN algorithm to adapt Q-learning to policies approximated by DNNs:

Experience replay First, experience replay allows to sample mini-batches of past interactions $\{< o, a, r, o' >\}$ from a replay memory $\mathcal{M}$ to perform stochastic gradient descent. Samples within a batch are likely to come from independent or distant interactions over time. As a consequence, it further reduces correlations in the DNN updates than the original Q-learning algorithm. Another advantage of experience replay is data efficiency. Experiences can be re-used several times in the DNN updates. It avoids to discard an experience immediately after an update.

Target Q-network Second, Q-learning targets in Eq. 4.3 are computed with a target Q-network $Q_{\Theta^-}(o, a)$ which parameters Θ^-. While parameters Θ of the DNN Q_Θ are updated after each interaction, the parameters Θ^- of the target Q-network Q_{Θ^-} are periodically updated after a fixed number of interactions. The authors in [68] argue that the use of a target network Q_{Θ^-} reduces correlations in the parameters update and improves the algorithm convergence.

Using these two strategies, the authors in [68] are the first to successfully approximate the agent's action value Q with a DNN combined with Q-learning [104]. They are able to train an agent which surpasses the performance of a professional human player across many Atari games.

4.2.2 DDQN loss function

In [100], the authors show that the action values are over-estimated when using DQN algorithm because of the maximization step involving the target Q-network from Eq. 4.3. In order to reduce DQN over-estimations of the action values, they modify the loss function used to update the DNN parameters:

$$L(\Theta) = (r + \gamma Q_{\Theta^-}(o', \arg\max_{a'} Q_\Theta(o', a')) - Q_\Theta(o, a))^2 \tag{4.4}$$

where γ is the discount factor (Sec. 3.3.1), Q_Θ the current Q-network and Q_{Θ^-} the target Q-network. Contrary to the loss from Eq. 4.3, the authors propose to decouple the steps of selecting the next action a' and its evaluation. They use the current network Q_Θ to select the next action a' and the target network Q_{Θ^-} to evaluate its value. With the exception of the loss calculation, the algorithm is the same as DQN. In Algo. 1 and Fig. 4.2, we summarize DDQN algorithm applied to EC.

Algorithm 1 DDQN algorithm applied to EC-POMDP

Require: Environment described by an EC-POMDP $\{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ as defined in Sec. 3.4 and corresponding training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$.
A parameterization of DDQN hyperparameters $h \in \mathcal{H}$ defined in [100].
Number of training episodes $M \in \mathbb{N}^+$ (if $M > N$, the agent will play several episodes of training on the same training sequences).
Ensure: Action value function $Q_{\Theta^*}(o, a)$ with optimal parameters Θ^*
Randomly initialize parameters Θ.
Set $\Theta^- = \Theta$.
Initialize replay memory $\mathcal{M}$: pre-fill a fraction of $\mathcal{M}$ with random policy.
for episode = 1 ... M **do**
Select a training sequence $(\mathbf{X}, l) \in \mathcal{D}$.
Initialize episode state $s_t = (\mathbf{X}, l, t)$ and observation $o_t = \mathbf{X}_{:t}$ with $t = 1$.
while episode is not *terminated* (i.e. $t < T$ and the agent has not classified the sequence yet) **do**
1: The agent receives observation o_t.
2: It chooses action $a_t = \arg\max_{a \in \mathcal{A}} Q_\Theta(o_t, a)$ with probability ϵ or random action with probability $1 - \epsilon$.
3: The environment computes reward $r_t = R((\mathbf{X}, l, t), a_t)$.
4: It moves to next state $s_{t+1} = (\mathbf{X}, l, t + 1)$ and gives next observation $o_{t+1} = \mathbf{X}_{:t+1}$.
5: Store interaction $< o_t, a_t, r_t, o_{t+1} >$ into replay memory $\mathcal{M}$.
6: Sample mini-batch of interactions $\{< o, a, r, o' >\} \sim \mathcal{M}$.
7-8: Update parameters Θ with gradient descent on loss function from Eq. 4.4 computed on the mini-batch $\{< o, a, r, o' >\}$.
Periodically update $\Theta^- = \Theta$
Increment time $t = t + 1$
end while
end for

4.2.3 Hyper-parameters

The neural network training with both DQN and DDQN algorithms depends on a set of hyper-parameters $h \in \mathcal{H}$, with $\mathcal{H}$ the combinatorial space of hyper-parameters. We mention the following:

- Some hyper-parameters of the DQN algorithm are related to the exploration rate ϵ of the agent, such as its initial and final values, as well as its annealing coefficient.
- Others are related to the loss L calculation, such as the mini-batch size and the discount factor γ.
- The implementation of the stochastic gradient descent can be performed with various algorithms [85], such as Adam and RMSprop algorithms. Each algorithm has its own set of hyper-parameters.
- The agent's replay memory $\mathcal{M}$ has a fixed size which value is an hyper-parameter.
- The number of interactions between each update of the target network's parameters Θ^- is also an hyper-parameter.

Optimal values for these hyper-parameters vary according to the dataset and must therefore be optimized for each application. We will detail in Sec. 4.3.3.1 how we optimize these hyper-parameters for a specific training dataset.

4.2.4 Related work on DQN variants

More work has been proposed to improve DQN algorithm, as reviewed in [44]. The authors in [102] introduce a dueling architecture of the DNN. A part of the DNN is dedicated to the state value V (Eq. 3.15) estimation while another part is dedicated to the advantage A estimation defined by $Q(s,a) = V(s) - A(s,a)$. They show that their method is useful when the environment has some "valuable" and "non-valuable" states. States are "non-valuable" when no choice of actions affect neither the transitions nor the rewards emitted by the environment.

In the EC problem, all classification actions terminate the episode regardless of the state of the environment. There are no states for which any action impact the environment. As a consequence, estimating the advantage A is not useful for the problem and we chose to evaluate the method with DDQN algorithm.

4.3 Experimental evaluation

In the following, we seek to evaluate the suitability of the method for the EC problem through experimental evaluation. More precisely, we evaluate if an agent trained with DDQN algorithm in an environment described by the EC-POMDP (Sec. 3.4) can achieve the objectives introduced in Chap. 2: to classify incomplete sequences as early as possible, with prediction time adapted individually to each data, and with a decision taken from end to end.

4.3.1 UCR dataset

To measure the agent's performance compared to state-of-the-art methods, we conduct experimental evaluation on The University of California, Riverside (UCR) archive [21]. The latter is a popular benchmark for classification and clustering of time series, and especially EC [7, 20, 36, 43, 70, 79, 87, 101, 109, 110].

We evaluate the solution on Gun-Point, Wafer and ECG datasets. These datasets are derived from various applications. They have various amount of training data and allow to evaluate the suitability of the solution when few training samples are available. To our knowledge these datasets are related to cost-insensitive classification problems. Tab. 4.1 summarizes information about these datasets.

Name	Type	Time series length T	Number of classes K	Size of training set	Size of testing set
ECG	ECG [1]	96	2	100	100
Gun-Point	Motion	150	2	50	150
Wafer	Sensor	152	2	1000	6174

Table 4.1: **Composition of ECG, Gun-Point and Wafer datasets from UCR archive [21].**

4.3.2 EC-POMDP model

In Chap. 3 we defined an EC-POMDP $\{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ for the EC problem such that rewards were defined in Eq. 3.28 by:

$$R((\mathbf{X}, l, t), a) = R_c((\mathbf{X}, l, t), a) + \lambda R_d((\mathbf{X}, l, t), a)$$

We encoded the objectives of fast and accurate predictions through several definitions of rewards according to the problem specificities (Sec. 3.4.2):

$$R_d \in \{R_{d,discount}, R_{d,shape}\}, \qquad R_c \in \{R_{c,ins}, R_{c,sen}, R_{c,ed-sen}\}$$

In this experimental evaluation, datasets from UCR are cost-insensitive and we define rewards for classification actions using Eq. 3.31, $R_c = R_{c,ins}$, such that the agent receives negative rewards of $r_- = -1$ for classification error and a positive reward of $r_+ = +1$ for accurate predictions:

$$R_{c,ins}((\mathbf{X}, l, t), a) = \begin{cases} +1 & \text{if } a = l \\ -1 & \text{if } a \in \mathcal{A}_c \setminus \{l\} \end{cases}$$

In addition, we define rewards for the delay action using Eq. 3.34, $R_d = R_{d,shape}$, such that the agent receives negative rewards at each decision of delay, with a penalty increasing in the number of data points collected in the sequence:

$$R_{d,shape}((\mathbf{X}, l, t), a_d) = -t^\kappa / T^\kappa$$

The penalty coefficient κ (Eq. 3.35) and the trade-off parameter λ are reward-related hyper-parameters of the agent's training (Sec. 4.3.3.1). We will fine-tune them experimentally.

4.3.3 Experimental pipeline

In this section, we introduce metrics and procedures used to (1) train the agent, (2) select optimal policies, (3) evaluate the agent and (4) monitor its training.

4.3.3.1 Training pipeline

Hyper-parameters setting: In Sec. 4.2, the agent's policy $\pi_\Theta(o) = \arg\max_{a \in \mathcal{A}} Q_\Theta(o, a)$ is defined by the DNN for the Q-function, $Q_\Theta(o, a)$, with parameters Θ. The DNN is trained with DDQN algorithm which requires to set a number of hyper-parameters $h \in \mathcal{H}$ (described in [68] and Sec. 4.2.3) before the training of the agent, with $\mathcal{H}$ the combinatorial space of hyper-parameters.

The combinatorial space $\mathcal{H}$ of the hyper-parameters being too large, we cannot perform an exhaustive search $\forall h \in \mathcal{H}$. Instead, we define a search zone $\tilde{\mathcal{H}}$ for optimal hyper-parameters by selecting a set of hyper-parameters $\tilde{\mathcal{H}} \subset \mathcal{H}$ in a restricted combinatorial space near optimal parameters presented in [68].

From this reduced space $\tilde{\mathcal{H}}$, we manually fine-tuned the hyper-parameter values on the training set, by observing the ones that worked best during training (according to performance metrics defined below).

Neural network setting: We approximate the Q-function with a DNN composed of convolution filters. Convolutions allow to learn both time dependencies and features dependencies in the sequences [103]. We add dropout to the DNN to prevent over-fitting [96].

The output layer of the DNN Q_Θ has $K+1$ neurons, one for each classification action $a \in \mathcal{A}_c$ and one for delay action a_d (see Eq. 3.26). It is fully-connected to its previous layer and has a linear activation function (because Q-values take values in $\mathbb{R}$).

The amount of layers, filters and dropout vary from a dataset to another depending on its complexity and amount of training data. Candidate architectures of DNNs are hyper-parameters of $\mathcal{H}$ and are manually tuned through the different experiments.

Input data pre-processing: In the practical implementation of the method, the DNN Q_Θ receives input data with fixed size, which are matrices of size $P \times T$. Indeed, for an input data $\mathbf{X}_{:t}$, we replace the future unknown data points $(\mathbf{x}_{t+1}, \cdots, \mathbf{x}_T)$ by zeros:

$$\begin{pmatrix} x_1^1 & \cdots & x_t^1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ x_1^P & \cdots & x_t^P & 0 & \cdots & 0 \end{pmatrix}_{t \leq T}$$

In experiments, we found useless to add a binary mask denoting if the data points were collected or not as proposed in [48].

Agent's training: Each dataset in the UCR archive is originally split into a training and a testing set. We use time series from the training set as episodes of training for the agent.

We run the DDQN algorithm for 100,000 iterations (updates of the DNN's parameters Θ with gradient descent on the loss from Eq 4.4). We experienced that this number of iterations was sufficient for the agent to learn to classify UCR datasets.

Because of the small size of UCR datasets ($M < N$, see Algo. 1), samples from the training set were passed several times as episodes to the agent during training.

4.3.3.2 Evaluation pipeline

Performance metrics: The time of prediction on the sequence $\mathbf{X}^n$ is t^n_{pred}. It is defined as the earliest time step for which the action value of a classification action $a \in \mathcal{A}_c$ outreaches the action value of delay a_d:

$$t^n_{pred} = \min_{t \in [1,T]} \{\pi_\Theta(\mathbf{X}^n_{:t}) \in \mathcal{A}_c\} \tag{4.5}$$

$$= \min_{t \in [1,T]} \{\arg\max_{a \in \mathcal{A}} Q_\Theta(\mathbf{X}^n_{:t}, a) \in \mathcal{A}_c\} \tag{4.6}$$

The label predicted by the agent on the sequence $\mathbf{X}^n$ is $\widehat{l^n}$:

$$\widehat{l^n} = \pi_\Theta(\mathbf{X}^n_{:t^n_{pred}}) \tag{4.7}$$

UCR datasets are cost-insensitive, i.e. classification errors are equally important, and we chose to evaluate the classification quality on the dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ by the accuracy Acc:

$$Acc = \sum_{n=1}^{N} \mathbb{1}(\widehat{l^n} = l^n)/N \tag{4.8}$$

Classification earliness is measured through the average prediction time t_{pred}:

$$t_{pred} = \sum_{n=1}^{N} t^n_{pred}/N \tag{4.9}$$

Trade-off between classification quality and earliness during training: To assess the agent's performance on both competitive objectives of accurate and fast predictions, we visualize Acc versus t_{pred} during training. At regular training intervals (every 5000 updates of the agent's DNN in Fig. 4.3), we evaluate the agent's policy on the entire training set. We compute Acc and t_{pred} on the training samples.

Optimal policy π^* selection: In [68], the authors evaluate the agent's policies during training and select the optimal policy π^* as the one with the highest score of reward. In the special case of EC for which two competitive objectives are optimized one against the other, the optimal policy π^* selection can be application-dependant.

We manually select the model that compromises best in terms of classification quality versus speed: we visualize the trade-off performed by the agent during its training and we keep the policy that fits best to the compromise allowed by the application.

The UCR archive is a benchmark for time series analysis. It was not built specifically for the EC problem and we do not have information about trade-off allowed by the applications. We use the following strategy for optimal policy π^* selection: among all policies with maximal Acc performance during training, we select the one with smallest prediction time t_{pred}, as illustrated in Fig. 4.3.

Remark: In these experiments, because of the small dataset sizes, we did not split the training sets into training and validation as it should be done. We are therefore aware of exposing ourselves to model selection biases by selecting a model from its results on its training data.

4.3.3.3 Monitoring pipeline

In order to monitor that the training of the agent goes well, we visualize the composition of its replay memory $\mathcal{M}$ during training. Specifically, we look at which actions a are the most represented in the stored interactions $\{< o, a, r, o' >\}$ in $\mathcal{M}$, as illustrated in Fig. 4.6, Fig. 4.10 and Fig. 4.12. We also monitor the length t of the prefixes represented in the replay memory $\mathcal{M}$, in other words the sequence acquisition time, as illustrated in Fig. 4.7, Fig. 4.9 and Fig. 4.13.

4.3.4 Results

4.3.4.1 Performance on UCR dataset

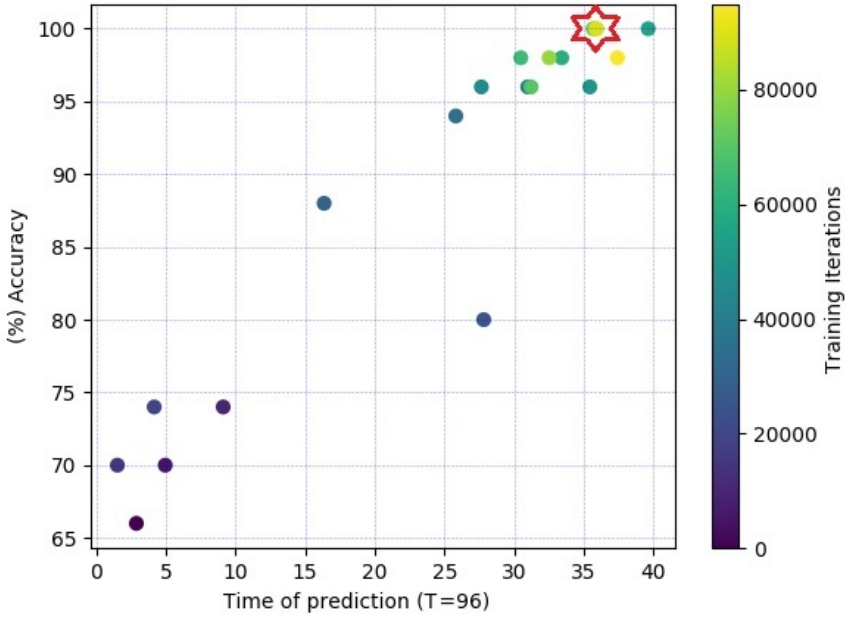

Figure 4.3: **Evolution of the agent's policy on Gun-Point training dataset during training.** The scatter plot shows the policy trade-off between accuracy Acc (in %, Eq. 4.8) in the y-axis and prediction time t_{pred} (Eq. 4.9) in the x-axis. The policy is evaluated on the complete training set every 5,000 iterations of DDQN algorithm. Blue dots correspond to the policy evaluation at early training (from 1 to 40,000 updates of the parameters Θ). Yellow dots correspond to the policy evaluation at a more advanced training (from 80,000 to 100,000 updates). In this experiment, the agent learned to slow its predictions down and improved its accuracy during training. The policy surrounded by the red star is selected as the optimal policy π^*. We evaluate π^* on the testing set and report its performance in Tab. 4.2.

Fig. 4.3 shows the evolution of the agent's policy during its training on Gun-Point training set. At the beginning of its training and until 20,000 updates of the parameters Θ (dark blue dots), the agent chose to classify the sequences with less than 10 data points in average, $t_{pred} < 10$, leading to a poor Acc (Eq. 4.8) around 70%. After 40,000 training iterations, it chose to acquire between 15 to 30 data points (out of the 96 available) before classifying leading to an increase in accuracy between 80% to 90%. After 80,000 training iterations (yellow dots), it learned to slow its prediction down to $t_{pred} = 35$ and reached an Acc superior to 95% on the training set.

This evolution of the policy is a consequence of the agent's training based on RL. It shows that the agent is able to continually adapt its behavior without human intervention, by RL. It can offer a variety of policies which are solutions to the EC problem and which propose variable compromises between accuracy and speed.

As explained in Sec. 4.3.3.2, one advantage of the proposed solution is that the user can

evaluate the agent's performance regularly during its training and then select the policy that performed best according to his compromise criterion. To save computational resources, the user can also stop the training of the agent at any time, once it reached a "good" compromise between earliness and accuracy according to his application's criteria.

From the training illustrated in Fig. 4.3, we chose to keep as the output of the learning process the policy that performed best on training set: among the most accurate policies, we selected the fastest one. We point out that the strategy of policy selection from the performance results on the training dataset is exceptional. It is due to the absence of a validation set and the small size of the training set. The selected optimal policy π^* is surrounded by a red star in Fig. 4.3. In Tab. 4.2, we report the performance of π^* on Gun-Point testing set.

Following the same procedure as for Gun-Point (detailed in Sec. 4.3.3), we also trained agents on Wafer and ECG training sets. We visualized the speed vs. accuracy classification trade-off during their training and we selected policies that performed best on the training sets. Then we reported their performance on Wafer and ECG testing sets in Tab. 4.2.

To compare the early classifier agent trained with RL with state-of-the-art methods for EC, we report in Tab. 4.2 the performance of Early Classification on Time Series (ECTS) [109] and Early Distinctive Shapelet Classification (EDSC) [110] methods. We did not reproduce their experiments but simply reported results mentioned in the original papers. We also indicate the performance of a 1 Nearest Neighbor (1NN) classifier on full time series with results provided in UCR archive.

On Gun-Point dataset, the agent stopped the acquisition of data points in the sequences and predicted labels once it received 22% of the full sequence in average. With less than a quarter of the sequences' data points, it managed to reach an accuracy superior to that of 1NN method on full sequences. The average prediction time is reduced by a factor two over the state-of-the-art methods ECTS and EDSC. Also, the agent's accuracy outperforms these methods.

On Wafer dataset, the reduction in predictive speed of the agent is significant compared to other methods, by at least a factor 7. It predicted labels once it received 4% of the full sequence in average. We found that the predictive speed was due to the identification of a very early discriminant pattern in sequences from the second label. The agent's accuracy is better than ECTS and EDSC methods and slightly inferior to that obtained by the full length 1NN method.

On ECG dataset, the agent gave fast predictions: it collected 17% of the sequences' data points. The predictive speed is reduced by a factor ≥ 1.9 compared to ECTS and EDSC. The agent achieved an accuracy comparable to these methods and the full length 1NN method.

From these experiments, we showed that the proposed solution can achieve EC while retaining an accuracy comparable to that of the full length 1NN classifier.

In Appendix B, we seek to experimentally evaluate if solving the EC-POMDP with a policy-based approach (Sec. 3.3.1.5) instead of this value-based approach leads to better policy performances on UCR datasets.

Dataset		RL agent trained on EC-POMDP	ECTS	EDSC	1NN Full
Gun-Point: 2 classes, $T = 150$ 50 train. samples 150 test. samples	*Acc*	**96%**	86.67%	94.67%	91.33%
	t_{pred}	**32.47**	70.39	69.3	150
	Coverage	100%	100%	100%	100%
Wafer: 2 classes, $T = 152$ 1000 train. samples 6174 test. samples	*Acc*	99.32%	99.08%	98.87%	**99.55%**
	t_{pred}	**5.73**	67.39	38.97	152
	Coverage	100%	100%	100%	100%
ECG: 2 classes, $T = 96$ 100 train. samples 100 test. samples	*Acc*	**89%**	**89%**	88%	88%
	t_{pred}	**16.09**	57.71	30.93	96
	Coverage	100%	100%	100%	100%

Table 4.2: **Evaluation of optimal policies on Gun-Point, Wafer and ECG testing sets.** Optimal policies were selected following procedure from Sec. 4.3.3.2 on the training set. *Acc* is defined in Eq. 4.8. t_{pred} is defined in Eq. 4.9. *Coverage* is the percentage of classified time series in the dataset. T is the maximal length of the time series. Best performances are written in bold. ECTS is the Early Classification on Time Series method from [109]. EDSC is the Early Distinctive Shapelet Classification method from [110]. 1NN Full is the 1 Nearest Neighbor method provided in UCR archive [21].

4.3.4.2 Sensitivity of the agent's training to hyper-parameters

During experiments and in general on all datasets, an effect of reward-related hyper-parameters has been observed on the behavior of the agent. A bad setting of the trade-off parameter λ (Eq. 3.28) and the penalty coefficient κ (Eq. 3.35) in the reward function definition can cause the agent to learn a sub-optimal policy. As an example, a low value of λ reduces the relative importance of rewards for delay compared to rewards for classification and it caused the agent to predict at the end of the sequences. A high value of λ gives more weight to rewards for delay (which are negative) and it caused the agent to predict immediately at the expense of accuracy.

To tune these reward-related hyper-parameters, we conducted a grid search and selected those achieving best performance on the training set. We observed that optimal values for λ and κ varied from a dataset to another depending on the complexity of the sequences.

Finally, on all three datasets and because of their small size, we experienced over-fitting when the DNN architecture was not appropriately sized.

4.3.5 Imbalanced replay memory

In the following experiments, we show some examples of the agent's training on ECG dataset from UCR Time Series Archive [21]. Fig. 4.4 shows labels distribution in the training and testing sets, illustrating the over-representation of label l_2 in comparison to label l_1. We apply the monitoring pipeline from Sec. 4.3.3.3 to observe the replay memory $\mathcal{M}$ of the agent during those trainings.

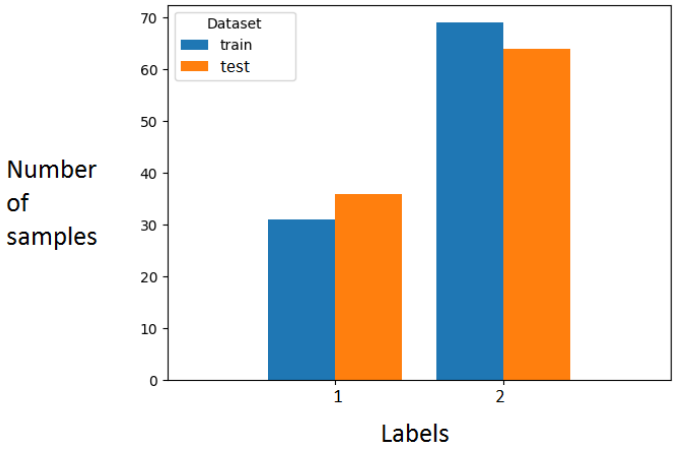

Figure 4.4: **Labels distribution in ECG training and testing sets.** ECG is a dataset from UCR Time Series Archive [21].

4.3.5.1 Example of rapid prediction biased toward the majority label

Fig. 4.5 illustrates one training of the agent during which its predictions are more and more accelerated at the expense of accuracy. When analyzing the results of this experiment, we saw that the agent fell into a sub-optimal policy of predicting more and more rapidly and exclusively the majority label l_2. Fig. 4.6 and Fig. 4.7 show the content of the agent's replay memory $\mathcal{M}$ during this experiment.

Fig. 4.6 shows that, at all times during training, the agent's replay memory is composed of more than 60% interactions associated to the delay action. At the beginning of its training, both labels are equally represented in the memory, by approximately 10% of the interactions. After 50,000 training iterations, the percentage of interactions associated to the prediction of the majority label l_2 increase up to 40% while those associated to the minority label l_1 drop to almost 0%.

Consequently, by only predicting the majority label at some point during its training, the agent stops experiencing the prediction of the minority label and its replay memory $\mathcal{M}$ becomes empty of these rare past experiences. The agent can no longer learn the Q-value of predicting label l_1, because past interactions associated to this action have been overwritten by more recent interactions. In other words, the learning base of the agent (its replay memory $\mathcal{M}$) gets reduced to two actions only, that of delay and that of predicting label l_2.

As shown in Fig. 4.7, after 5000 training iterations, 50% of the interactions in the replay memory are associated to partial sequences with 10 to 60 data points (shown by the interquartile range of the boxes). From 45000 updates of the policy, 75% of the interactions involve prefixes of sequences with less than 20 data points. Gradually as the agent learns to accelerate its prediction during training, its replay memory gets represented by short prefixes of sequences only. After 90000 training iterations, almost all interactions illustrate sequences with less than 5 data points.

Consequently, the more its training progresses and the less the agent can train on medium (and late) acquisition times, i.e. on medium size sequences. Its learning base gets exclusively represented by very short sequences.

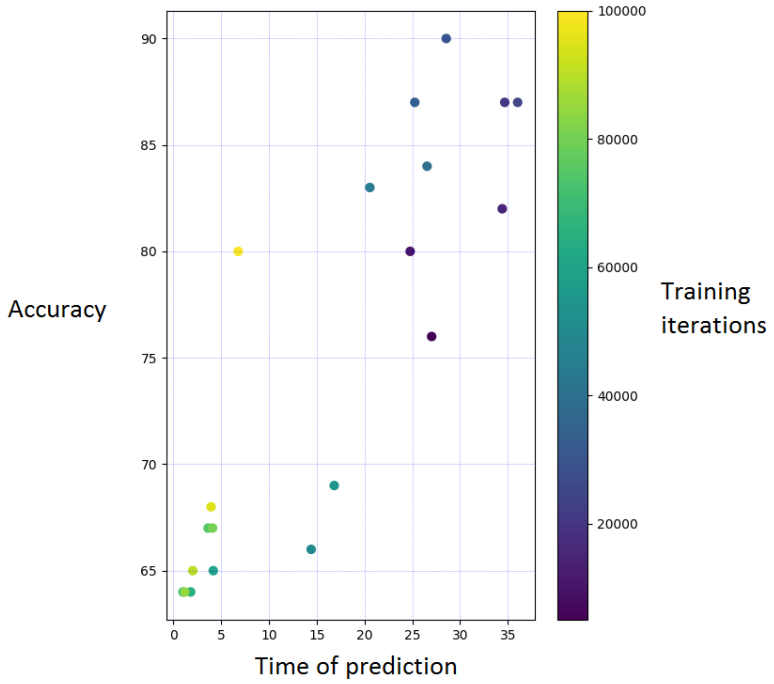

Figure 4.5: **Evolution of the agent's policy during training example n°1.** The scatter plot shows the policy trade-off between Acc (in %, Eq. 4.8) in the y-axis and t_{pred} (Eq. 4.9) in the x-axis. The policy is evaluated on the complete training set every 5,000 iterations of DDQN algorithm. Blue dots correspond to the policy evaluation at early training. Yellow dots correspond to the policy evaluation after 100,000 iterations of training.

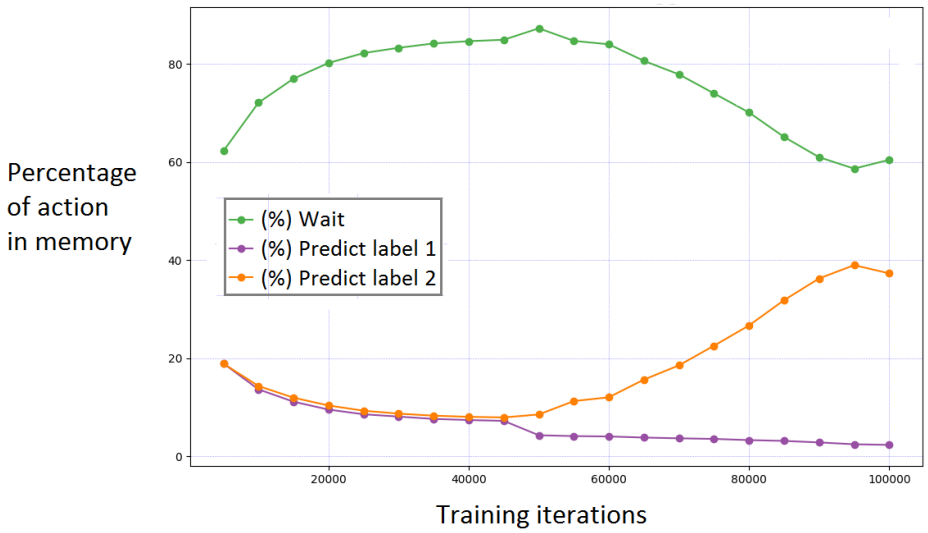

Figure 4.6: **Evolution of actions $a \in \mathcal{A}$ in the replay memory $\mathcal{M}$ during the agent's training example n°1.** The percentages of interactions representing each action are shown in the y-axis, every 5000 iterations of training.

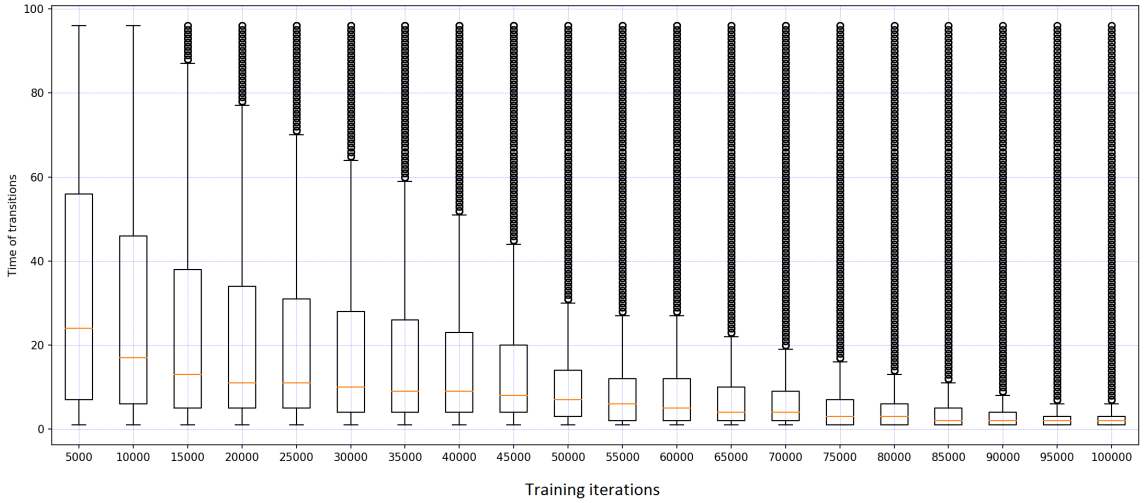

Figure 4.7: **Boxplot of sequence acquisition time $t \in [1, T]$ in the replay memory $\mathcal{M}$ during the agent's training example n°1.** Distributions of acquisition times are represented in the y-axis, every 5000 iterations of training. Outliers are represented by circles. The maximal length of sequences is $T = 96$.

In conclusion of this experiment, we observe some imbalance issues in the memory. The latter becomes essentially associated to very short sequences along training. Interactions with the action of predicting the minority label are overwritten and disappear from the memory. The agent can no longer learn the Q-value for this action which gets excluded from its learning base.

Finally, we can see that the more the replay memory $\mathcal{M}$ of the agent gets imbalanced during training, the more its performance degrades (and possibly inversely).

4.3.5.2 Example of medium speed prediction biased toward the majority label

In this experiment, we show a sub-optimal policy for which predictions are performed at various time steps (and no longer too rapidly as illustrated in the previous experiment) but still towards the majority label l_2. The evolution of the policy is illustrated in Fig. 4.8 and shows that the agent predicted at all times steps during training, from 1 to 98, nevertheless with a low accuracy that never exceeds 85%. Fig. 4.6 and Fig. 4.7 show the content over the agent's replay memory during this experiment.

As shown in Fig. 4.9, the distribution of acquisition times represented by the interactions in memory is stable during training. At all times during training, 75% of the interactions involve partial sequences with length inferior to 30, of which 25% are associated to prefixes of sequences with only a few data points. The remaining 25% involve longer sequences, ranging from 35 to 70 data points approximately. From 70 data points, there are only a few interactions in the memory associated with these acquisition times that appear as outliers in the figure. As a result, this shows that, even when the agent predicts at various times during training, much of its memory is associated with early acquisition times. Medium acquisition times are represented in a minority, and late acquisition times are exceptionally represented (if not at all).

A consequence of the agent's predictions at various times is shown in Fig. 4.10. At

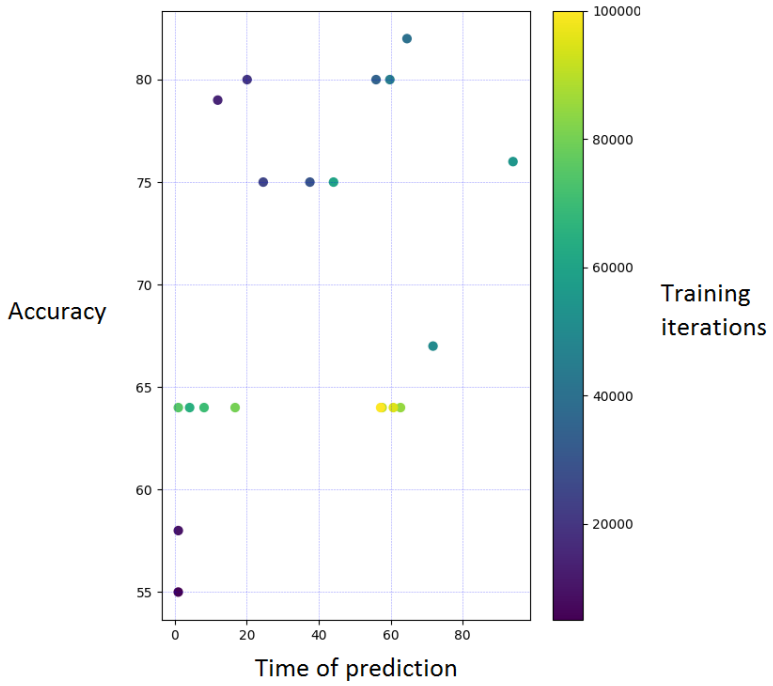

Figure 4.8: **Evolution of the agent's policy during training example n°2.** The scatter plot shows the policy trade-off between Acc (in %, Eq. 4.8) in the y-axis and t_{pred} (Eq. 4.9) in the x-axis. The policy is evaluated on the complete training set every 5,000 iterations of DDQN algorithm. Blue dots correspond to the policy evaluation at early training. Yellow dots correspond to the policy evaluation after 100,000 iterations of training.

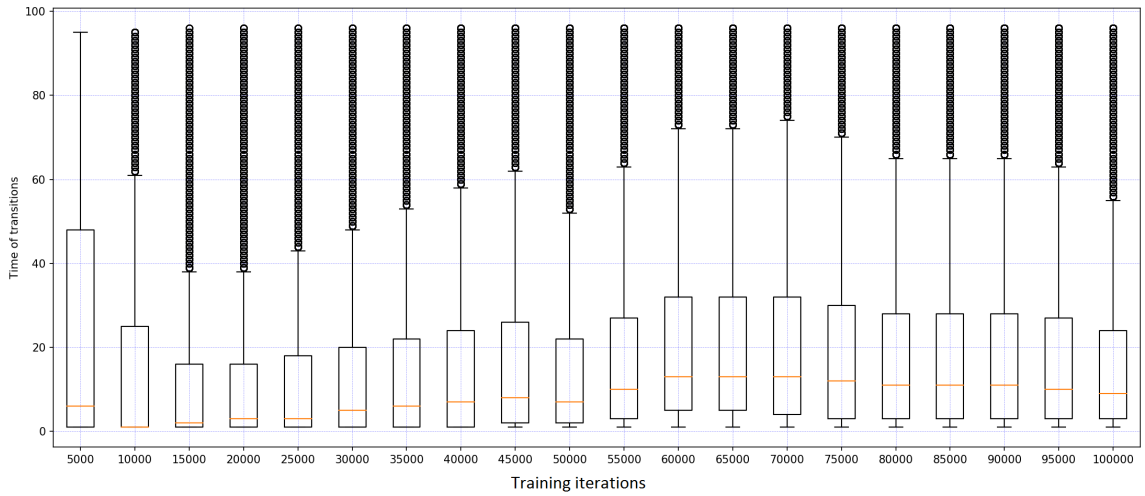

Figure 4.9: **Boxplot of sequence acquisition time $t \in [1, T]$ in the replay memory $\mathcal{M}$ during training example n°2.** Distributions of acquisition times are represented in the y-axis, every 5000 iterations of training. Outliers are represented by circles. The maximal length of sequences is $T = 96$.

early training, the memory equally represents all three actions. Then, from 10000 training iterations, the number of interactions associated to the delay action increase while that of classification actions decrease. From 60000 updates of the policy's parameters, more than 80% of the memory gets dedicated to delay interactions. In other words, the action of delay becomes over-represented in the replay memory $\mathcal{M}$. Also, from 80000 training iterations, the replay memory gets represented by two actions only, because the agent stopped predicting the minority label l_1.

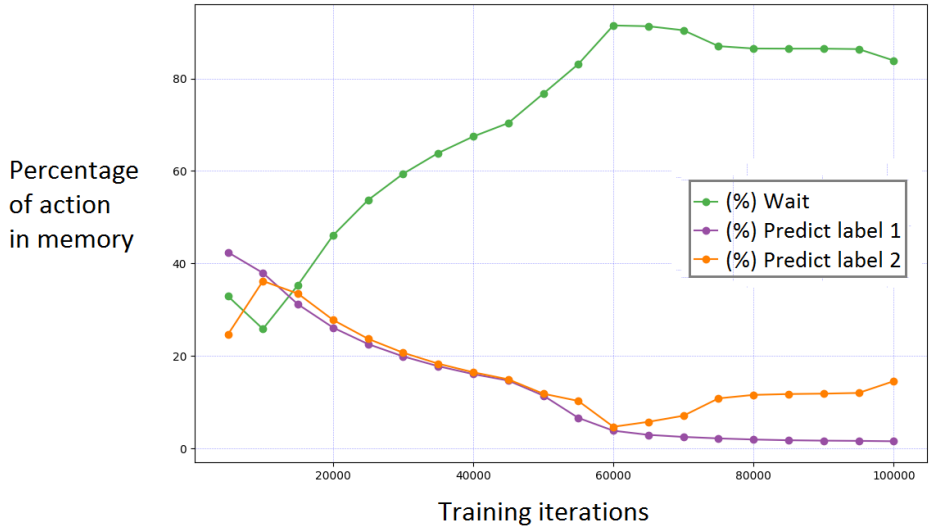

Figure 4.10: **Evolution of actions $a \in \mathcal{A}$ in the replay memory $\mathcal{M}$ during the agent's training example n°2.** Action percentages are represented in the y-axis, every 5000 iterations of training.

As a conclusion of this experiment, problems of imbalance in the memory are again observed. While the delay action is over-represented, that of the minority label prediction completely disappears. Also, the memory tends to essentially represent early to medium acquisition times.

4.3.5.3 Example of medium speed prediction

Fig. 4.12, Fig. 4.13 and Fig. 4.11 show another example of over-writing of the delay action in the replay memory $\mathcal{M}$.

As illustrated in Fig. 4.11, at some point during its training, the agent predicts in average around time step 20, on sequences of maximal length 96, in other words using approximately 20% of available data-points.

For a prediction at time step 20, the agent stores 19 interactions of delay and 1 interaction of classification. An episode of length 20 is therefore composed of 95% of delay actions for 5% of classification actions. Consequently and as shown in Fig. 4.12, its replay memory becomes empty of classification actions during training because these interactions are the rarest.

Fig. 4.11 shows that the agent's performance improves despite the imbalance in its replay memory, but it does not reach the optimal performance reported in Tab. 4.2 which has been obtained with the same amount of training time.

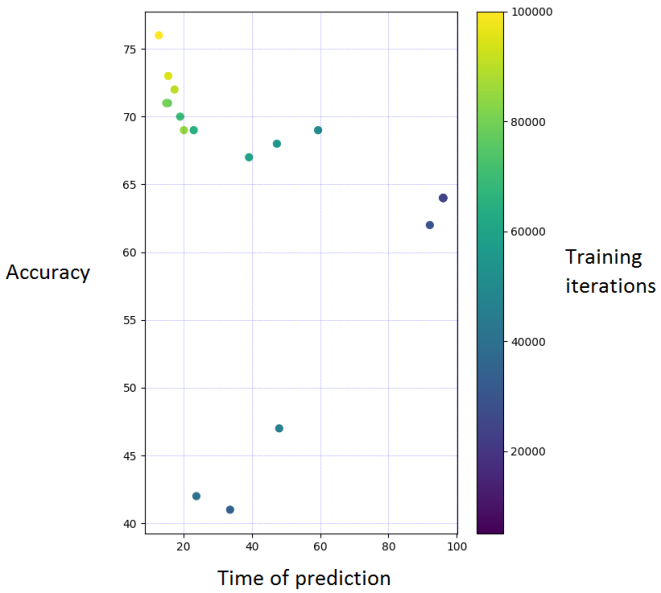

Figure 4.11: **Evolution of the agent's policy during training example n°3.** The scatter plot shows the policy trade-off between Acc (in %, Eq. 4.8) in the y-axis and t_{pred} (Eq. 4.9) in the x-axis. The policy is evaluated on the complete training set every 5,000 iterations of DDQN algorithm. Blue dots correspond to the policy evaluation at early training. Yellow dots correspond to the policy evaluation after 100,000 iterations of training.

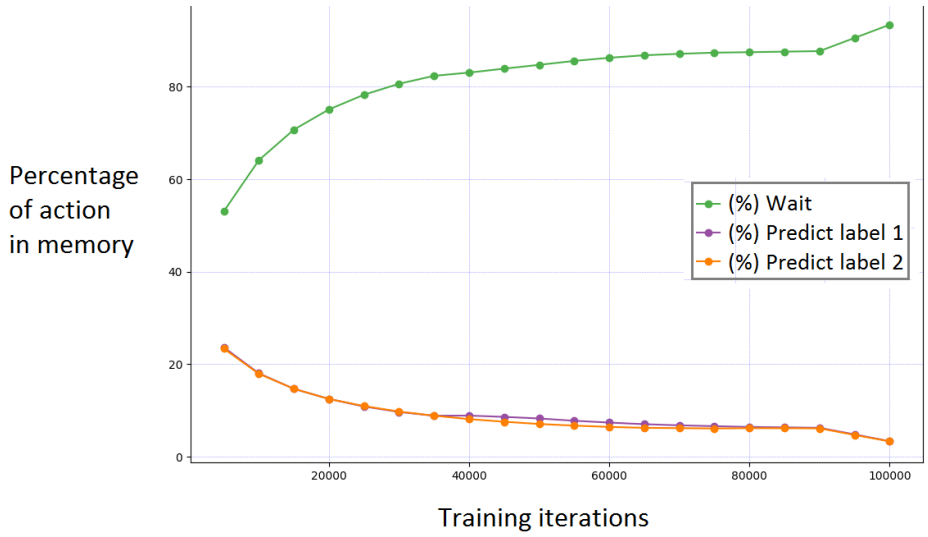

Figure 4.12: **Evolution of actions $a \in \mathcal{A}$ in the replay memory $\mathcal{M}$ during the agent's training example n°3.** Action percentages are represented in the y-axis, every 5000 iterations of training.

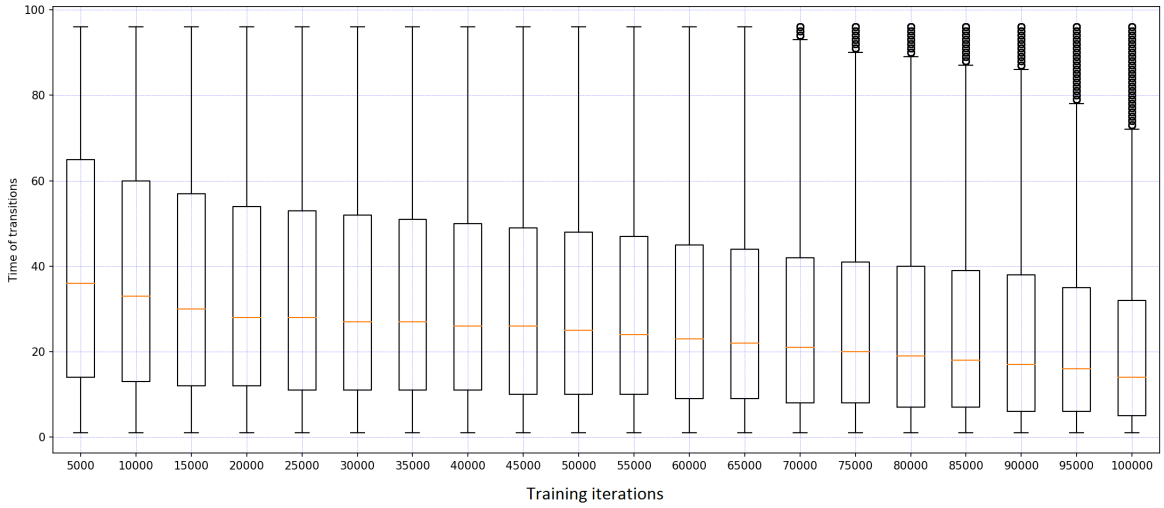

Figure 4.13: **Boxplot of sequence acquisition time $t \in [1, T]$ in the replay memory $\mathcal{M}$ during the agent's training example n°3.** Distributions of acquisition times are represented in the y-axis, every 5000 iterations of training. Outliers are represented by circles. The maximal length of sequences is $T = 96$.

4.4 Conclusion

In this chapter we proposed to solve the EC-POMDP from Chap. 3 with a value-based approach. We proposed a pipeline to train an agent with the DDQN algorithm and to select an optimal policy π^* for EC.

In experiments, we showed on UCR time series benchmark [21] that the EC problem can be solved with an end-to-end RL agent. The agent achieves EC objectives: to compromise between classification quality and its earliness. Moreover, it simultaneously learns classification features and decision-making rules on prediction times.

We showed that the agent is able to continually adapt its behavior without human intervention. By learning to find a better trade-off for the EC-POMDP, the agent goes through various policies for which the relative importance of accuracy vs. speed evolves. Therefore, the method offers a set of models with a wide range of possible compromises, and the user can select the one that best fits his application.

Also, we experimentally showed that the agent achieves similar or better results in accuracy and prediction time compared to state-of-the-art methods from the literature. In Appendix B, we will solve the EC-POMDP with a policy-based approach (Sec. 3.3.1.5) and report the performances on UCR time series benchmark.

We identified limits to apply the DDQN algorithm in its original form on the EC-POMDP. We showed that the replay memory of the agent can become imbalanced during its training because the agent falls into sub-optimal policies, weakening its overall learning. The question is now how to optimize DDQN for the EC problem. Specifically, we want to know if the resolution of the problem of poorly balanced replay memory can improve the training of the agent, which is the subject of the next chapter.

Chapter 5

Optimized EC-POMDP solving with robust memory management

In Chap. 4, we solved the Partially Observable Markov Decision Process (POMDP) for early classification (EC), noted EC-POMDP, by training an agent with the DDQN algorithm [100]. In experiments, we put forward limits to apply the DDQN algorithm in its original version. Some trainings showed that the replay memory $\mathcal{M}$ of the agent (which has a fixed size) can become imbalanced with respect to the stored actions a_t and the length of the observations o_t. This imbalance is naturally caused by the EC-POMDP specificities detailed in Sec. 3.4.3, namely (a) all but one of the actions terminate episodes and (b) actions of classification are rarer than the delay action. We argued that a negative consequence of this imbalance is the deterioration of the agent's training which is sensitive to the composition of its replay memory $\mathcal{M}$. The latter forms the learning base of the agent and poorly managed memory can lead to sub-optimal training.

In this chapter, we seek to optimize the EC-POMDP solving with robust replay memory management when applying DDQN algorithm. We raise three questions in relation to replay memory management and one question in relation to episode management:

(1) *Which interactions should be stored?*

(2) *Which interactions should be sampled?*

(3) *Which interactions should be discarded (when the replay memory is full)?*

(4) *How to initialize an episode of training?*

We study how to answer these questions while addressing the specificities of the EC-POMDP, and we propose revisions to the DDQN algorithm to fix the agent's imbalanced memory issue. In addition, we propose two adaptions of DDQN algorithm to the EC-POMDP depending on whether the EC application comes with a finite training dataset (batch learning) or, on the opposite, can generate new training data over time (online learning). In experiments, we evaluate if the contributions have a positive effect on the agent's overall training and performance.

Sec. 5.1 provides a literature review on the problem of memory management in Reinforcement Learning (RL).

In Sec. 5.2, we optimize DDQN algorithm for online learning (i.e. when there are successive repetitions of interaction collection and policy optimization) by proposing strategies of *prioritized sampling*, *prioritized storing* and *random episode initialization*.

In Sec. 5.3, we adapt DDQN algorithm for batch learning, i.e. for applications having a maximum number of possible interactions to collect between the agent and the environment, caused by a finite training set. We propose to decouple interaction collection from policy optimization and we apply the *prioritized sampling strategy*.

In Sec. 5.4, we introduce an evaluation pipeline to compare the different versions of the algorithm, with and without the proposed strategies. We then evaluate the effects of *prioritized sampling*, *prioritized storing* and *random episode initialization* on a dataset related to the thesis industrial application.

In Sec. 5.5, we train a set of static Deep Neural Networks (DNNs) to classify at each time step during the sequence acquisition. These static DNNs have an architecture equivalent to that of the agent. We evaluate how these static DNNs perform in terms of accuracy vs. speed compared to the EC agent trained with RL.

5.1 Related work on memory management in RL

In the original version of the DQN algorithm [68] (from which the DDQN algorithm [100] has been adapted), the authors uniformly sample past interactions from the replay memory $\mathcal{M}$ in order to update the parameters Θ of the DNN (see step 6 in Fig. 4.2). A consequence of this uniform sampling is that all type of interactions are sampled with the same probability, regardless of their importance. Later, some work in the literature adapt the algorithm to efficiently manage the agent's replay memory $\mathcal{M}$.

In [88], the authors propose an adaptation of the DQN algorithm [68] with prioritized experience replay (PER). The method seeks to sample "important" interactions more frequently than "non important" interactions. To that end, the method samples in *priority* the interactions $< o, a, r, o' >$ on which the agent's Q-value estimates $Q_\Theta(o, a)$ were far from the Q-learning targets $r + \gamma \max_{a'} Q_{\Theta^-}(o', a')$ used to compute the loss from Eq. 4.3. To avoid sampling exclusively the interactions with biggest *priority*, the authors introduce a stochastic prioritization. As a result, their method allows to learn on difficult or rare interactions on which the agent struggles predicting accurate Q-values, by re-sampling them more often.

In [73], the authors use the DQN algorithm and force that a fraction of the mini-batch used for loss update (Eq. 4.3) is associated to interactions with positive rewards. In others words, they give higher priority to interactions with positive rewards and they seek to learn more efficiently from these rewarding interactions.

In this work, we leave aside PER [88] which can be computationally expensive and we propose a less expensive solution inspired by [73]. We choose to exploit the fact that the interactions between the agent and the environment can be easily categorized into subgroups, according to the type of actions selected. Contrary to [73] where sampling is prioritized according to the scalar rewards received in the interactions, **we propose to use prioritize sampling by focusing on particular state–action pairs.**

5.2 Optimized EC-POMDP solving in online learning

In previous chapter, we detailed the application of DDQN algorithm [100] to the EC problem (see Algo. 1). In its original version, the algorithm is introduced for online learning, with successive repetitions of a) collecting training interactions between the agent and the environment, and b) updating the agent's policy. This is due to the fact that video games

usually come with an emulator which can generate an infinite number of training episodes and therefore the number of interactions is not bounded. In relation to EC applications, we point out that training an EC agent with online learning is suitable for applications allowing for streaming or multi-phases data collection. For example in predictive maintenance, the machine sensor signals are daily monitored and the training dataset for this application could regularly be increased.

Regarding the questions of memory and episode management, when applied to the EC problem, the DDQN algorithm uses the following strategies:

- The environment *initializes the training episode at the first stage of the process*, i.e. at $t = 1$. For the EC problem, the first observation given to the agent during a training episode is the first vector of data points $\mathbf{x}_1$ in the sequence $\mathbf{X}$ associated to the episode.
- The agent *stores every new interaction* into a replay memory of a fixed size.
- In order to update the parameters Θ of its policy π_Θ, the agent *uniformly samples* a mini-batch of past interactions $\{< o, a, r, o' >\} \sim \mathcal{M}$ from the replay memory (and then applies a stochastic gradient descent with respect to the loss function from Eq. 4.4).
- When the replay memory is full, the agent applies a seniority system and *replaces its oldest interactions by the most recent ones*, such as a "First In First Out" queue.

In this doctoral work, we propose a different memory and episode management to address the EC-POMDP specificities presented in Sec. 3.4.3. With regard to interaction sampling, we propose a *prioritized sampling strategy* (Sec. 5.2.1). Then, for the questions of interaction storing and memory update, we propose a *prioritized storing strategy* (Sec. 5.2.2). We also address the issue of time imbalance with a simple *random episode initialization strategy* (Sec. 5.2.3). Tab. 5.1 summarizes the strategies applied by the original algorithm and those proposed in this thesis.

	DDQN	Proposition of optimized DDQN in online learning
Memory management		
(1) Which interactions should be stored?	All new interaction is stored	*Prioritized storing*
(2) Which interactions should be sampled?	Uniform sampling	*Prioritized sampling*
(3) Which interactions should be discarded?	Oldest interactions are discarded (seniority system)	*Prioritized storing*
Episode management		
How to initialize an episode?	At time t=1	*Random episode initialization*

Table 5.1: **Memory and episode management strategies of original DDQN and optimized DDQN algorithms in online learning.** Comparison between DDQN original algorithm and the optimized algorithm proposed in this thesis.

5.2.1 Prioritized sampling

The EC-POMDP suffers from the over-representation of the delay action a_d compared to classification actions $a \in \mathcal{A}_c$ (Sec. 3.4.3). With DDQN uniform sampling in the replay memory $\mathcal{M}$, batches of interactions $\{< o, a, r, o' >\} \sim \mathcal{M}$ are highly imbalanced (mostly composed of delay interactions) and the agent may experience learning troubles or sub-optimal training. We therefore adapt DDQN to the EC problem with a simple *prioritized sampling strategy* presented in Algo. 2 to improve sample efficiency and to address the second question on memory management (2) *Which interactions should be sampled?*

The strategy is the following. To ensure that the agent can learn from classification interactions, which are rarer than delay interactions, we force that a fraction $\mu \in [0, 1]$ of the interactions within a mini-batch $\{< o, a, r, o' >\} \sim \mathcal{M}$ is associated with classification actions $a \in \mathcal{A}_c$, , as illustrated in Fig. 5.1.

Moreover, in order to be robust to class-imbalanced classification problems such as the one from the industrial application (Sec. 2.2), we force that the mini-batch is uniformly balanced among the different labels.

Algorithm 2 PrioritizedSampling

Require: Replay memory $\mathcal{M}$.
Prioritized sampling parameter $\mu \in [0, 1]$.
Ensure: Mini-batch of interactions $\{< o, a, r, o' >\}$ of size $b \in \mathbb{N}^+$
for $l \in \mathcal{L}$ **do**
Sample a random mini-batch of interactions $\{< o, a, r, o' >\} \sim \mathcal{M}$ of size b/K such that the observations o are associated to temporal sequences X of label l, with fraction μ of the mini-batch having $a \in \mathcal{A}_c$.
end for

5.2.2 Prioritized storing

The EC-POMDP results in interactions of the agent with the environment that are mainly composed of delay action (Sec. 3.4.3): a prediction at time t results in $t - 1$ delay actions for one classification action. Using a seniority system for in-memory updates is not appropriate when there are rare or under-represented actions because these interactions should be kept in priority. We therefore address the questions on memory management (1) *Which interactions should be stored?* and (3) *Which interactions should be discarded?* with a simple *prioritized storing strategy*.

The strategy is the following. To prevent the replay memory $\mathcal{M}$ from being over-represented by delay interactions, we allocate a fraction ρ of the replay memory $\mathcal{M}$ to classification interactions, and a fraction $1-\rho$ to delay interactions. $\rho \in [0, 1]$ is the prioritized storing parameter and is an hyper-parameter of the method that need to be fixed before the training of the agent. When the replay memory $\mathcal{M}$ is full, we discard the oldest interaction with the same action as the current interaction to store. This strategy is equivalent to considering two memories: one for delay and one for classification, as illustrated in Fig. 5.1.

If the problem involves a highly class-imbalanced training dataset, the *prioritized storing strategy* can also consider an additional division of the memory allocated for classification interactions. Indeed, a fraction of the classification memory can be reserved for each label, so

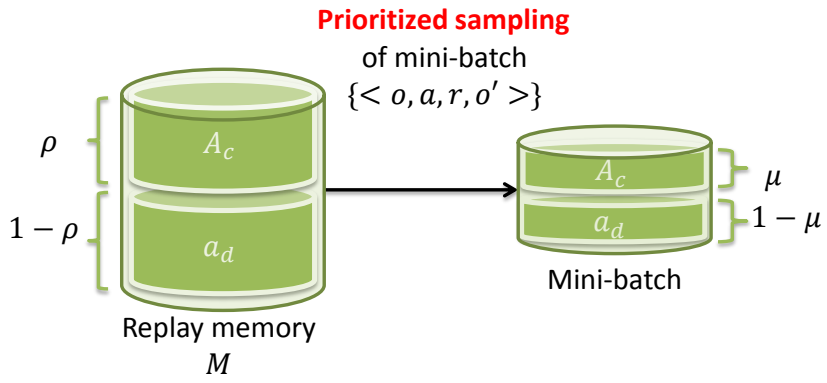

Figure 5.1: ***Prioritized storing* and *prioritized sampling* strategies.** Interactions are stored in the replay memory $\mathcal{M}$ with *prioritized storing strategy* with parameter ρ. The mini-batch $\{< o, a, r, o' >\}$ of past interactions is sampled from the replay memory $\mathcal{M}$ with *prioritized sampling strategy* with parameter μ.

that interactions associated with rare labels are not quickly over-written by those associated with dominant labels.

5.2.3 Random episode initialization

To answer the 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 $t = 1$ causes the replay memory $\mathcal{M}$ to be over-represented by interactions with classification performed at early time steps. We therefore adapt DDQN algorithm to the EC-POMDP with a *random episode initialization* strategy presented in Algo. 3.

The strategy is the following. During the training episodes, we initialize the beginning of the episode at a random time $t \sim \mathcal{U}[1, T]$ in the temporal sequence. Consequently, the agent receives the first t data points $(\mathbf{x}_1, \cdots, \mathbf{x}_t)$ all at once from the sequence $\mathbf{X}$, as a first observation of the training episode. If it chooses to delay prediction, its second observation will be the vector of data points $(\mathbf{x}_1, \cdots, \mathbf{x}_t, \mathbf{x}_{t+1})$, and so on. We emphasize that the agent could not act on the first t acquisition of data points. By doing so, we compel the agent to explore and train on all times of the sequence acquisition. The agent can receive partial sequences acquired until late times of the process.

In Algo. 3, the first time of the training episode is sampled from a uniform distribution between time 1 and time T. Nevertheless, we point out that other distributions could have been used, while ensuring that the agent can train on various times.

5.2.4 Algorithm

Algo. 4 summarizes the proposed adaptation of DDQN algorithm to EC. The algorithm is presented for online learning, i.e. with simultaneous interaction collection and policy opti-

Algorithm 3 EpisodeInitialization

Require: Training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ with pairs of temporal sequences $\mathbf{X}$ and their associated label $l \in \mathcal{L}$, described in Sec. 2.1.
Ensure: An initial observation o of a training episode
Sample training pair $(\mathbf{X}, l) \sim \mathcal{D}$
Sample time $t \sim \mathcal{U}[1, T]$
Return observation $o = \mathbf{X}_{:t}$

mization. It uses the three strategies of *prioritized sampling*, *prioritized storing* and *random episode initialization* presented in Sec. 5.2.3, Sec. 5.2.1 and Sec. 5.2.2. The algorithm is illustrated in Fig. 5.2.

In addition to initial DDQN hyper-parameters defined in [100], the optimized algorithm involves two additional hyper-parameters: the prioritized sampling parameter $\mu \in [0, 1]$ from Sec. 5.2.1, and the prioritized storing parameter $\rho \in [0, 1]$ from Sec. 5.2.2.

Algorithm 4 DDQN algorithm optimized for EC in online learning

Require: Environment described by an EC-POMDP $\{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ (defined in Sec. 3.4) and corresponding training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ (defined in Sec. 2.1).
Number of training episodes $M \in \mathbb{N}^+$ (if $M > N$, the agent will play several episodes of training on the same training sequences).
Prioritized sampling parameter $\mu \in [0, 1]$ from Sec. 5.2.1.
Prioritized storing parameter $\rho \in [0, 1]$ from Sec. 5.2.2.
A parametrization of DDQN hyper-parameters $h \in \mathcal{H}$ defined in [100].
Ensure: Action value function $Q_\Theta(o, a)$ with optimal parameters Θ^*
Randomly initialize parameters Θ. Set $\Theta^- = \Theta$.
Initialize replay memory $\mathcal{M}$: pre-fill a fraction of $\mathcal{M}$ with random policy.
for episode = 1 ... M **do**
Initialize episode with observation: $o_t =$**EpisodeInitialization**$(\mathcal{D})$ from Algo. 3.
while episode is not *terminated* (i.e. $t < T$ and the agent has not classified the sequence yet) **do**
1: The agent receives observation o_t.
2: It chooses action $a_t = \arg\max_{a \in \mathcal{A}} Q_\Theta(o_t, a)$ with probability ϵ or random action with probability $1 - \epsilon$.
3: The environment computes reward $r_t = R((\mathbf{X}, l, t), a_t)$.
4: It gives next observation $o_{t+1} = P((\mathbf{X}, l, t), a_t)$.
5: Store $interaction =< o_t, a_t, r_t, o_{t+1} >$ into replay memory $\mathcal{M}$:
$\mathcal{M} \leftarrow$ **PrioritizedStoring**$(\mathcal{M}, interaction, \mu)$ from Sec. 5.2.2.
6: Sample a mini-batch of interactions from memory:
$\{< o, a, r, o' >\}$ = **PrioritizedSampling**$(\mathcal{M}, \mu)$ from Algo. 2.
7-8: Update parameters Θ with gradient descent on loss function from Eq. 4.4 computed on the mini-batch $\{< o, a, r, o' >\}$.
Periodically update $\Theta^- = \Theta$
Increment time $t = t + 1$
end while
end for

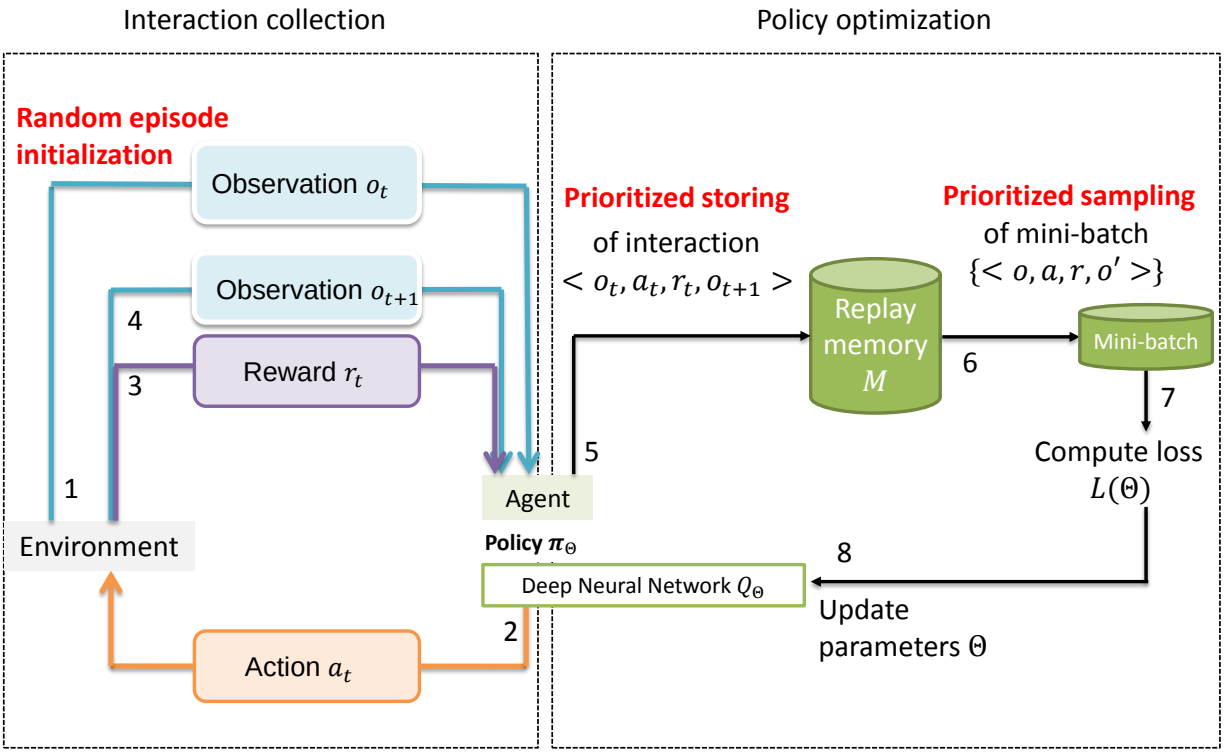

Figure 5.2: **Optimized DDQN algorithm for EC in online learning with *prioritized sampling*, *prioritized storing* and *random episode initialization*.** Steps 1 to 8 form one iteration of the algorithm.
1: Instead of initializing an episode of training at time $t = 1$, we apply *random episode initialization* from Algo. 3.
2-4: see Fig. 4.2.
5: Instead of applying a seniority system to the replay memory $\mathcal{M}$, we apply *prioritized storing* from Sec. 5.2.2.
6: Instead of applying uniform sampling from the replay memory $\mathcal{M}$, we apply *prioritized sampling* presented in Algo. 2.
7-8: see Fig. 4.2.

Sec. 5.4 conduct an experimental evaluation to compare the original DDQN algorithm in online learning (Algo. 1), and the proposed adaption of the algorithm (Algo. 4) with strategies of *prioritized sampling*, *prioritized storing* and *random episode initialization*. The results show that these strategies improve the performance achieved by an EC agent, compared to the original algorithm.

5.3 Optimized EC-POMDP solving in batch learning

5.3.1 Motivation

In recent years, the main applications of Deep Reinforcement Learning (DRL) were derived from video games and, consequently, DRL algorithms were designed for these applications

[44, 67, 68]. Video games come with an emulator which can generate an infinite number of episodes, leading to successive repetitions of interaction collection and policy optimization.

On the opposite, EC applications can not usually generate new interactions with the environment along the agent's training. Indeed, many real-life EC applications come with a finite training dataset, either because there was a single data acquisition phase or because data acquisition is expensive and can not exceed a certain amount of samples. For example, in microbiological diagnostics, data acquisition is expensive because of the experiments it requires to conduct, and it is common to be limited in the amount of data that can be collected.

Also, in some cases, the agent's policy can not be tested on the real environment while training and we must wait until we find the optimal policy before applying it to real cases. This is for example the case of medical diagnoses for which the agent can only train from a database of past interactions and for which we can not test its policy during training on real patients.

As a consequence, these applications are characterized by a POMDP for which there are a finite number of interactions to train on. For example, in the case of Gun-Point dataset from the UCR archive [21], there are 50 training sequences with length 150 and associated to two possible labels, and consequently the total number of possible interactions for training is $50 * 3 * 150 = 22500$.

We suggest to develop a batch version of DDQN algorithm where the interaction collection is decoupled from the training of the agent. We argue that all possible training interactions between the agent and the environment can be simulated and stored in an exhaustive replay memory before updating the agent's policy. We therefore propose a simplification of DDQN algorithm for batch learning, by first building the exhaustive replay memory of all possible training interactions and then optimizing the agent's policy.

5.3.2 Algorithm

Given a finite training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ (Eq. 2.3), the batch version of DDQN algorithm is the following. The first step is to build an exhaustive replay memory $\tilde{\mathcal{M}}$. To that end, we simulate all possible interactions between the agent and the environment, on all sequences from the finite training dataset $\mathcal{D}$. In other words, we simulate interactions on all prefixes of sequences from the training dataset and for all action choices.

The second step is to update the parameters Θ of the DNN Q_Θ. To that end, we repeat the sub-steps of:

1. sampling a mini-batch of interactions with the *prioritized sampling* strategy proposed in Sec. 5.2.1, and

2. updating parameters Θ with gradient descent on loss function from Eq. 4.4 computed on the mini-batch.

During the second step, the agent no longer generates training episodes. Instead, all the possible episodes were simulated during the construction of the exhaustive memory $\tilde{\mathcal{M}}$ in the first step. In other words, once $\tilde{\mathcal{M}}$ is built, the agent only updates the parameters Θ of its policy.

We point out that, by storing all the possible choices of action for all the acquisition times of the training sequences, the memory is balanced in action and time. The only imbalance that can arise concerns the labels represented in the memory and which is taken into account in the *prioritized sampling strategy*. Contrary to Algo. 4 in online learning, the method with

batch learning no longer necessitates the strategies of *prioritized storing* and *random episode initialization* introduced to balance the memory, as summarized in Tab. 5.2.

In addition to initial DDQN hyper-parameters defined in [100], the method involves the prioritized sampling parameter $\mu \in [0, 1]$ from Sec. 5.2.1. On the other hand, it no longer requires optimizing the hyper-parameters related to the exploration of the agent. We present in Algo. 5 and illustrate in Fig. 5.3 the adaptation of DDQN algorithm to EC in a batch learning.

	DDQN	Optimized DDQN for batch learning
Memory management		
(1) Which interactions should be stored?	All new interaction is stored	*Exhaustive replay memory*
(2) Which interactions should be sampled?	Uniform sampling	*Prioritized sampling*
(3) Which interactions should be discarded?	Oldest interactions are discarded (seniority system)	*Exhaustive replay memory*
Episode management		
How to initialize an episode?	At time t=1	No episodes

Table 5.2: **Memory and episode management strategies of original DDQN and optimized DDQN algorithms in batch learning.**

Batch learning or online learning? We point out that the two versions of the algorithm can be combined in the specific case where the user has a finite training dataset at first and will later collect additional training samples. It is possible to first build an exhaustive memory from the finite training dataset, learn a policy in batch learning, and then update the policy in online learning while processing newly collected data as they arrive.

5.4 Experimental comparison between memory management strategies

Experiments were conducted in Sec. 4.3 to evaluate the suitability of the method, i.e. if an agent could behave as an early classifier. We now seek to experimentally evaluate the benefits of the memory and episode management strategies proposed in this chapter. To assess the impact of prioritized sampling (Sec. 5.2.1), prioritized storing (Sec. 5.2.2) and random episode initialization (Sec. 5.2.3) on the agent's training, we will train early classifier agents with the original DDQN algorithm in online learning and with three adaptations of the algorithm:

- *DDQN-baseline* refers to original DDQN algorithm [100], as described in Algo. 1.
- *DDQN-ps* refers to DDQN with prioritized sampling and prioritized storing proposed in Sec. 5.2.1 and Sec. 5.2.2.
- *DDQN-ei* refers to DDQN with random episode initialization proposed in Sec. 5.2.3.

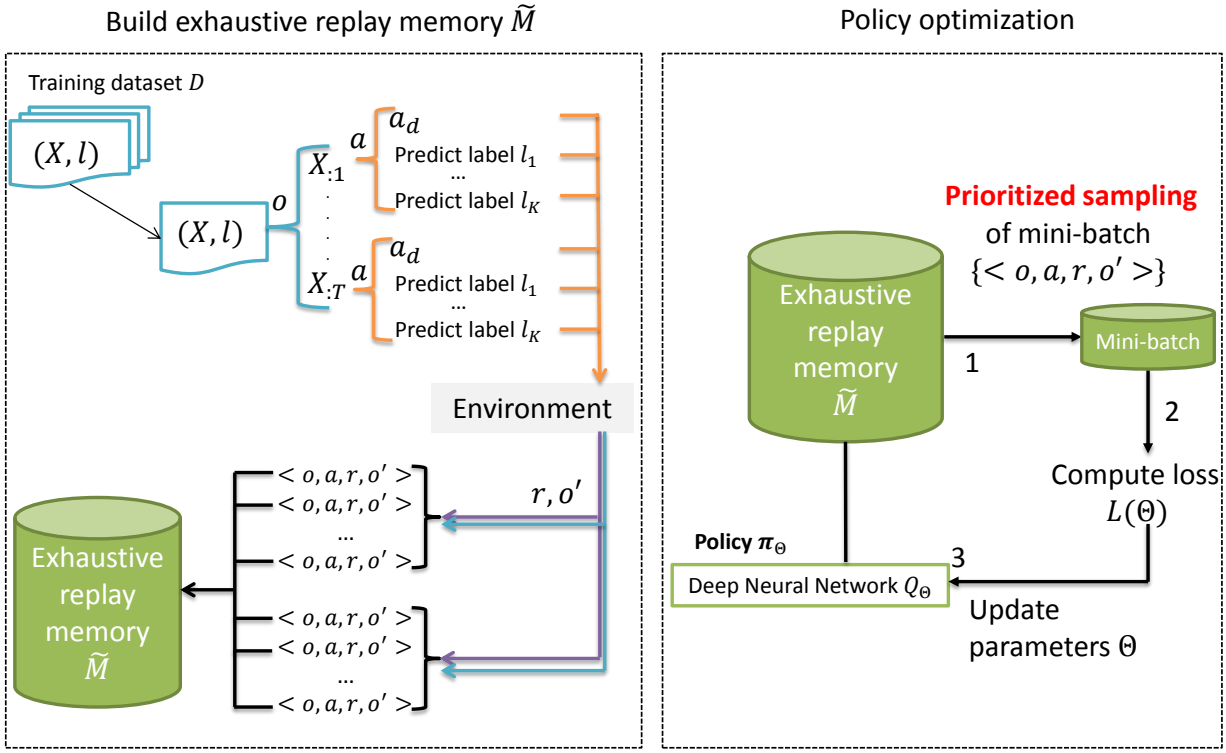

Figure 5.3: **Optimized DDQN algorithm for EC in batch learning with *prioritized sampling*.** The first step is to build exhaustive replay memory $\tilde{\mathcal{M}}$. Then the policy π_Θ is optimized.

- *DDQN-ps-ei* refers to DDQN with simultaneously prioritized sampling, prioritized storing and random episode initialization as synthesized in Algo. 4.

DDQN-baseline, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei* differ by their methods of memory and episode management. The objective is to compare the four methods and to determine if one of them results in a better training of the agent.

5.4.1 Industrial 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 $N = 3155$ temporal sequences $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_T)$ have length $T = 77$ and each data point $\mathbf{x}_{i\in[1,T]}$ is a 5-dimensional array ($P = 5$). With previous notations from Sec. 2.1, $\mathbf{X} \in \mathbb{R}^{5\times 77}$.

Labels Sequences are associated to labels in $\mathcal{L} = \{a, b, c, d\}$ depicting four classes of living organisms. Fig. 5.4 gives the distribution of the labels among the training, validation and testing sets. It shows that label c is under-represented in comparison to labels a and b.

t-SNE projection In Fig. 5.5, we represent the training set with a two-dimensional t-SNE (t-distributed stochastic neighbour embedding) of the (complete) temporal sequences using

Algorithm 5 DDQN algorithm applied to EC in batch learning

Require: Environment described by an EC-POMDP $\{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ as defined in Sec. 3.4 and corresponding training dataset $\mathcal{D} = \{(\mathbf{X}^n, l^n)\}_{n=1..N}$ as defined in 2.1.
Number of parameters updates $U \in \mathbb{N}^+$.
Prioritized sampling parameter $\mu \in [0, 1]$.
A parameterization of DDQN hyper-parameters $h \in \mathcal{H}$ defined in [100].
Ensure: Action value function $Q_\Theta(o, a)$ with optimal parameters Θ^*
Store all possible interactions in exhaustive replay memory $\tilde{\mathcal{M}}$:
for $n = 1$ ... N **do**
Select a training pair $(\mathbf{X}^n, l^n) \in \mathcal{D}$.
for $t = 1$ ... T **do**
Compute observation $o = \mathbf{X}^n_{:t}$
for $a \in \mathcal{A}$ **do**
Compute reward $r = R((\mathbf{X}^n, l^n, t), a)$
Compute next observation $o' = P((\mathbf{X}^n, l^n, t), a)$.
Store interaction $< o, a, r, o' >$ into exhaustive replay memory $\tilde{\mathcal{M}}$.
end for
end for
end for
Randomly initialize parameters Θ. Set $\Theta^- = \Theta$.
for update = 1 ... U **do**
1: Sample a mini-batch of interactions from memory:
$\{< o, a, r, o' >\}$ = **PrioritizedSampling**$(\tilde{\mathcal{M}}, \mu)$ from Algo. 2.
2-3: Update parameters Θ with gradient descent on loss function from Eq. 4.4 computed on the mini-batch $\{< o, a, r, o' >\}$.
Periodically update $\Theta^- = \Theta$
end for

algorithm from [64]. We observe overlapping clusters of points from different 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.

5.4.2 EC-POMDP model

We use the same EC-POMDP model as in Sec. 4.3, except for the classification rewards. We vary rewards for correct classification $R_{c,ins}((\mathbf{X}, l, t), a = l) \in \{0, +1\}$ in order to obtain policies with slow decision-making and to be able to compare the methods *DDQN-baseline*, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei* in late prediction times.

We point out that in the practical use of the method for the cost-sensitive industrial application (Sec. 2.2.2.4), we use the reward function definition $R_{c,sen}$ (Eq. 3.32) that involves the different costs of misclassification. However, in this experiment, we seek to evaluate the benefits of the memory management strategies for the general problem of EC.

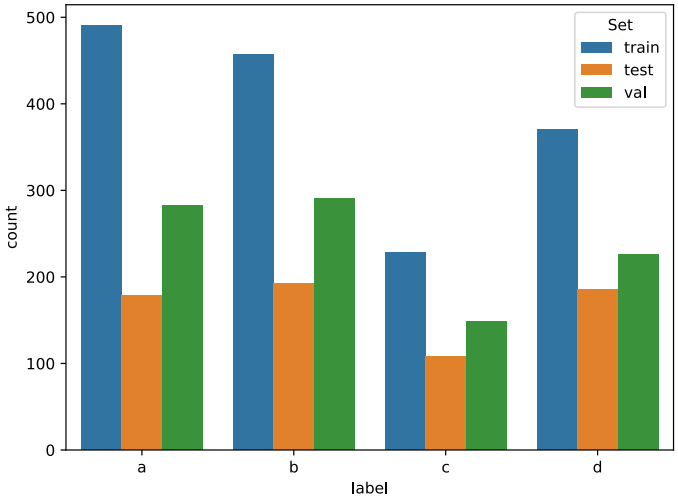

Figure 5.4: **Labels distribution in the industrial sets of training, validation and testing.** The set of labels is $\mathcal{L} = \{a, b, c, d\}$.

5.4.3 Experimental pipeline

The experimental evaluation consists in the following steps.

(a) We launch a set of trainings on the training set for each method *DDQN-baseline*, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei* (Sec. 5.4.3.1).

(b) At the same time that the agent is training, we regularly evaluate its policies on the validation set (Sec. 5.4.3.2).

(c) Once the trainings are completed and the agent has been evaluated on the validation set, we statistically compare its performances between each method on the validation set. This step allows us to compare the overall performance of each method (Sec. 5.4.3.3).

(d) We then select top-5 optimal policies on the validation set for each method. We evaluate and compare the performance of these optimal policies on the testing set. This step allows us to compare the best performances of each method (Sec. 5.4.3.4).

5.4.3.1 Training pipeline

The following training pipeline is applied for each method of memory and episode management.

Neural network setting and input data pre-processing: As detailed in Sec. 4.3.3.1, we zero-pad the sequences (i.e. the data points that are still not collected by the agent are replaced with zero values).

We approximate the Q-function with a DNN composed of convolution filters, with dropout applied at training time. The amount of layers, filters and dropout are hyper-parameters of the method (see below). An example of architecture used during experiments for the DNN is given in Appendix E.

Hyper-parameters setting: The DNN training depends on a set of hyper-parameters [100] which combinatorial space is too large for an exhaustive search of optimal parameters. We note $\mathcal{H}$ the combinatorial space of all hyper-parameters.

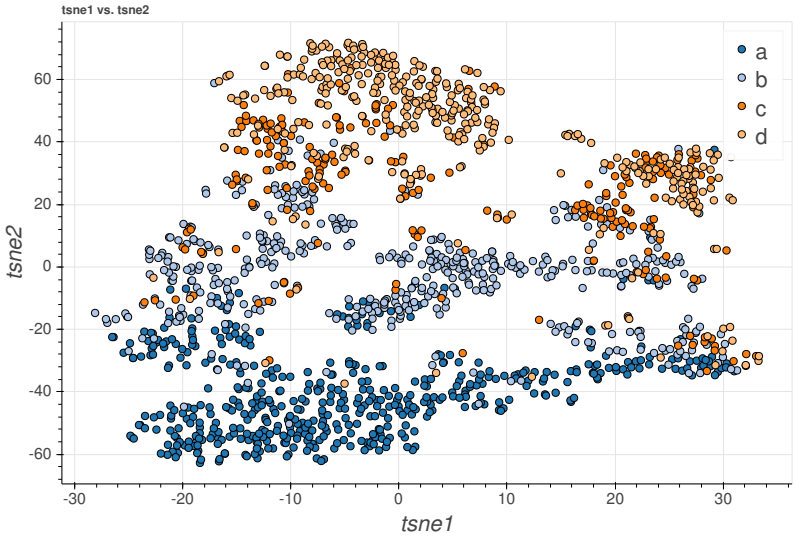

Figure 5.5: **Two-dimensional t-SNE embedding of the training set**. Each dot represents a complete temporal sequence $X^n_{:T}, n \in [1, N]$ from the training set. Dots are coloured according to their label a, b, c or d.

We first perform a rough optimization of the method by pre-tuning the hyper-parameters near optimal parameters presented in [68]. We then select a set of hyper-parameters in a restricted combinatorial space $\tilde{\mathcal{H}} \subset \mathcal{H}$ near those pre-tuned parameters. We also sample the EC-POMDP reward definition in $R_{c,ins}((\mathbf{X}, l, t), a = l) \sim \{0, +1\}$ to vary the learning dynamics of the agent.

In experiments from Sec. 5.4.4, we evaluate the methods for 100 combinations of hyper-parameters ($\tilde{\mathcal{H}} = \{h_1, \cdots, h_{100}\}$) and we train as many independent agents. We dedicate one agent per setting of hyper-parameters $h \in \tilde{\mathcal{H}}$ and each agent is trained separately between all settings h.

Agent's training: When a DNN is trained under supervision (for static classification or regression tasks), its parameters Θ are generally updated until the loss function stops decreasing on the validation set. As a consequence, the selection of the best DNN parameters Θ^* is straightforward: the selected parameters are the ones with lowest loss on the validation set.

At the opposite, when a DNN is trained with RL and specifically with the DDQN algorithm, the loss function from Eq. 4.4 on an interaction $< o, a, r, o' >$ is based on the Q-learning target, $r + \gamma Q_{\Theta^-}(o', \arg\max_{a'} Q_\Theta(o', a'))$, which is an approximation of future cumulated rewards. The loss is then estimated and it is generally not used either to stop the training procedure or to select optimal parameters Θ^*.

Instead, for each hyper-parameter setting $h \in \tilde{\mathcal{H}}$, we independently train an agent for a fixed number of episodes in the environment, until it reaches 100000 updates of its DNN parameters Θ with gradient descent on the loss from Eq 4.4.

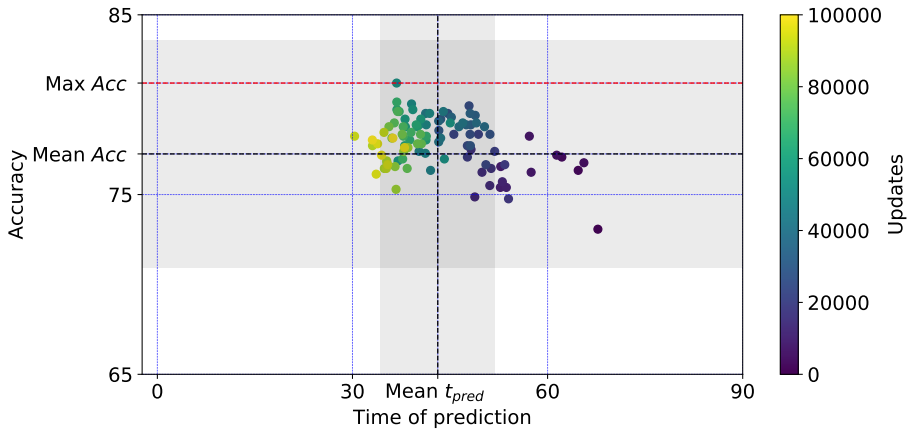

Figure 5.6: **Example of performance metrics from one training of the agent.** The scatter plot shows the policy trade-off between Acc (in %, Eq. 4.8) in the y-axis and t_{pred} (Eq. 4.9) in the x-axis during training. The policy is evaluated on the validation set every 1000 updates of the DNN parameters Θ. Blue dots correspond to the policy evaluation at early training. Yellow dots correspond to the policy evaluation after 100,000 iterations of training. The black vertical line (resp. band) gives the agent's mean (resp. stdev) t_{pred} during training. The black horizontal line (resp. band) gives the agent's mean (resp. stdev) Acc during training. The red horizontal line gives the agent's maximal Acc during training.

5.4.3.2 Evaluation pipeline

The following evaluation pipeline is applied simultaneously to each training, for each method of memory and episode management.

Performance metrics: We use the same performance metrics Acc and t_{pred} than Sec. 4.3.3.2 except that the accuracy Acc is computed with macro-averaging (Eq. 2.11) to handle the class-imbalanced nature of the dataset.

We point out that the industrial application of the thesis is cost-sensitive (Sec. 2.2.2.4) and in the practical use of the method for industrial application, we use application-specific criteria that take into account the different costs of misclassification in addition to macro accuracy.

Agent's evaluation: During training, we simultaneously evaluate the agent on the validation set every 1000 updates of Θ. As a consequence, for each training dedicated to a set of hyper-parameter $h \in \tilde{\mathcal{H}}$, we obtain a set of 100 (=100000/100) policies, noted $\{\pi_\Theta\}_h$, which will be used for subsequent evaluations.

Fig. 5.6 reports the evaluations performed during one training of the agent, for one particular setting of hyper-parameter $h \in \tilde{\mathcal{H}}$. As illustrated in the figure, we evaluate the trade-off between classification quality and earliness at regular training intervals: we compute Acc versus t_{pred}.

5.4.3.3 Comparison pipeline

Once the trainings are completed and the agent has been evaluated on the validation set, we seek to measure if some of the proposed methods for memory and episode management have a positive effect on the agent overall training. The following comparison pipeline is applied at the end of all the method evaluations.

Comparison metrics: For each completed training, we compute the following criteria assessing its quality.

Best performance: To assess an agent's best classification performance during its training, we compute max *Acc* independently of the prediction time, as illustrated in Fig. 5.6.

Mean performance: To overcome that max *Acc* only reflects an agent's behavior at a single time step in its training and to globally assess an agent's performance during 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 Fig. 5.6. 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 (stdev) metric, as illustrated in Fig. 5.6. A high score of stdev *Acc* means that the policies evaluated along training were not equally accurate and very unstable.

Methods comparison: Once quality criteria have been measured, we are interested in assessing the robustness of each method regarding the hyper-parameter setting.

For each method, we compute the metrics max *Acc*, mean *Acc*, stdev *Acc*, mean t_{pred} and stdev t_{pred} on each training dedicated to a set of hyper-parameters $h \in \tilde{\mathcal{H}}$. We visualize the distribution of these metric scores for each method, as illustrated in Fig. 5.7.

We then statistically compare the distribution of the metrics between each method two-by-two and we report the p-values of Mann-Whitney rank statistical tests on the null hypothesis that the two compared methods are equivalent, as reported in Tab. 5.3.

5.4.3.4 Optimal policies comparison pipeline

Once the methods have been statistically compared in relation to their overall performances, we aim at comparing the best policies achieved by each method.

Optimal policy selection: Contrary to UCR benchmark used in experiments from Sec. 4.3, the dataset of this experimental evaluation offers a validation set which can be used to select optimal policies.

For all trainings (each one being dedicated to a set of hyper-parameters $h \in \tilde{\mathcal{H}}$), we obtain sets of policies $\{\{\pi_\Theta\}_h, \forall h \in \tilde{\mathcal{H}}\}$ that were evaluated on the validation set, with variable performances in *Acc* and t_{pred}. For regular time ranges $[t_{inf}, t_{sup}[\subset [1, T]$, we first select the policies from $\{\{\pi_\Theta\}, \forall h \in \tilde{\mathcal{H}}\}$ which have an average prediction time $t_{pred} \in [t_{inf}, t_{sup}[$ on the validation set. We then select among this restricted set of

policies the top-5 policies with best classification performance on the validation set. It is up to the user to set the time intervals according to the evaluation detail he wishes to obtain.

From this pre-selection, we then have as many optimal policy candidates as time ranges considered for t_{pred}. Among all candidates, we can then choose the optimal policy as the one satisfying the most our will to compromise between accuracy and speed.

Optimal policy evaluation: We report and compare the performances in *Acc* and t_{pred} of these top-5 policies on the testing set, as illustrated in Fig. 5.10 and Fig. 5.8. This allow to visualize the best classification results obtained by the agent for all earliness trade-off.

5.4.4 Results

On each method (*DDQN-baseline*, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei*), we perform 100 independent trainings of the agent (Sec. 5.4.3.1) on the training set. During training, we regularly evaluate the policies on the validation set (Sec. 5.4.3.2). Once trainings are terminated, we thus obtain a set of evaluations on the validation set for all four methods which we then compare with statistical tests (Sec. 5.4.3.3). In addition, we select top-5 policies on the validation set for all four methods (Sec. 5.4.3.4). We evaluate those top-5 policies on the test set and we report performances.

5.4.4.1 Statistical comparison between all four methods

The distributions of performance metrics from Sec. 5.4.3.3 are shown in Fig. 5.7 and statistically compared in Tab. 5.3.

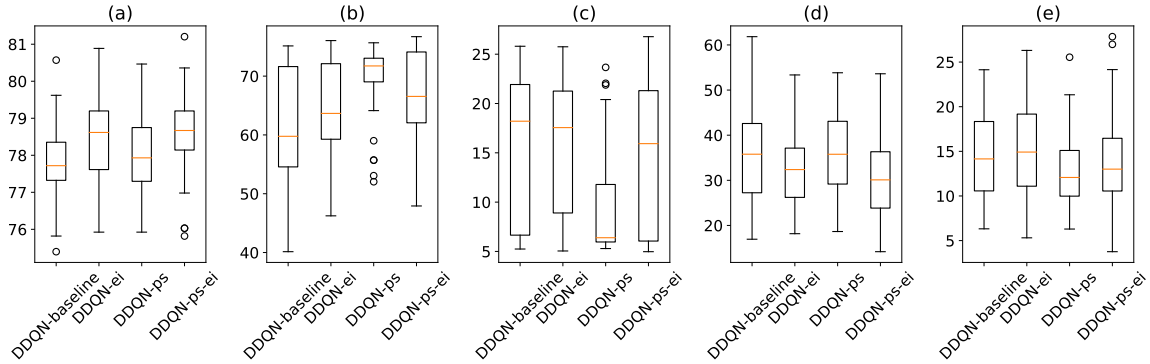

Figure 5.7: **Distribution of performance metrics on the validation set for each method of memory and episode management: *DDQN-baseline*, *DDQN-ps-ei*, *DDQN-ps* and *DDQN-ei*.**
(a) Max *Acc*. (b) Mean *Acc*. (c) Stdev *Acc*. (d) Mean t_{pred}. (e) Stdev t_{pred}. For each method, the distribution is calculated on all the evaluations that were performed on the validation set, for each set of hyper-parameters tested.

First, we compare the best classification performance achieved by the agent during its training sessions, on each method of memory and episode management. That is to say, on

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

Table 5.3: **Statistical comparison between *DDQN-baseline*, *DDQN-ps*, *DDQN-ei* and *DDQN-ps-ei* performance metrics.** The table reports p-values of Mann-Whitney rank tests on the null hypothesis that *DDQN-baseline* have a score comparable to *DDQN-ps* and *DDQN-ps-ei* for each performance metric (max *Acc*, mean *Acc*, stdev *Acc*, mean t_{pred} and stdev t_{pred}) from Fig. 5.7. The null hypothesis is rejected in favor of the alternative hypothesis on tests with a p-value below 0.05, shown in bold. The alternative hypothesis is that the metric performance is different between the methods. Fig. 5.7 shows which method has the greatest score.

each of training of the agent, we keep the policy that was the most accurate in classification. Tests from Tab. 5.3 show that both *DDQN-ei* and *DDQN-ps-ei* improve max *Acc* over *DDQN-baseline*. In other words, these methods result in policies with the best classification quality.

Then, we compare the average performance of the agent during its training sessions, by averaging the performance of each of its policies from the same training session. This allows to illustrate the overall performance of the agent throughout its training, and not at a specific moment of its training. Tests from Tab. 5.3 show that both *DDQN-ps* and *DDQN-ps-ei* improve mean *Acc* over *DDQN-baseline* which means that these methods improve the overall classification quality of the agent compared to the baseline. Also, both *DDQN-ei* and *DDQN-ps-ei* shorten mean t_{pred} over *DDQN-baseline* which means that these methods result in earliest classification times compared to the baseline. *DDQN-ps-ei* is then the method that leads to both best classification quality and earliest prediction times simultaneously. Both competitive EC costs are improved with this method.

In terms of stability, measured through the metrics of stdev t_{pred} and stdev *Acc*, the different methods are comparable except for *DDQN-ps* which is statistically less variable in terms of accuracy compared to *DDQN-baseline*.

As a conclusion, *DDQN-ps-ei*, which refers to DDQN combined with all of the proposed strategies (prioritized sampling, prioritized storing and random episode initialization), is the best memory and episode management method because it simultaneously improves the classification performance of the agent and fastens its prediction time.

5.4.4.2 Evaluation of top-5 policies for all four methods

Top-5 policies (Sec. 5.4.3.4) on all four versions of DDQN algorithm are shown in Fig. 5.8. For each method, accuracy rapidly increases when prediction time reaches $t_{pred} = 30$. Then, accuracy slightly gets better when prediction time increases up to $t_{pred} = 40$. We can observe that accuracy stops increasing (and even slightly decreases in some cases) when the prediction is performed approximately at $t_{pred} > 50$. This is due to the particularity of the application

for which more time passes and more the biological processes associated with different classes will have similar states.

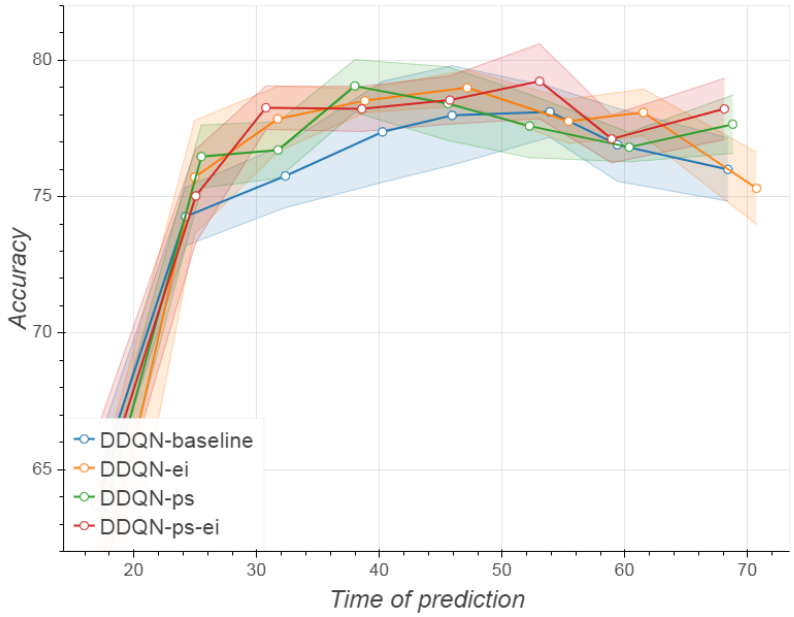

Figure 5.8: **Evaluation of top-5 policies from *DDQN-baseline*, *DDQN-ei*, *DDQN-ps* and *DDQN-ps-ei* on the test set.** The evaluation involves 8 distinct time intervals $[t_{inf}, t_{sup}[$ between $[1, T]$. For each time interval $[t_{inf}, t_{sup}[$, the top-5 policies (which have an average prediction time $t_{pred} \in [t_{inf}, t_{sup}[$ and highest Acc) were selected from the validation set. The full line represents mean accuracy and the band is the accuracy standard deviation on the 5 policies on the test set.

From Fig. 5.8, we observe that *DDQN-baseline* top-5 policies are globally the least accurate under all trade-offs of prediction time t_{pred}. Top-5 policies with highest Acc for different trade-off of t_{pred} are produced by *DDQN-ei* and *DDQN-ps-ei*. We can see that the different proposed methods of memory and episode management lead to optimal policies which are at least as good or better than those obtained with the original DDQN algorithm.

5.5 Experimental comparison between early classifier and naive static classifier

We seek to experimentally measure the added value of the proposed RL method for EC in comparison to static classification. Specifically, the objective of this evaluation is to measure if a DNN gives similar performances of EC (accuracy vs. speed) when trained with two different approaches:

- a common static classification training, carried out at fixed time steps, i.e. without decision-making on the prediction time,
- an early classification training based on RL, i.e. the network has the ability to adapt its prediction time.

We wish to compare the RL approach with the static classification approach on equal terms, that is when the DNNs have equivalent architectures.

The principle of the evaluation is the following. For an agent that would predict on average at t_{pred}, we seek to evaluate whether a static DNN classifier that would make the prediction with the same average speed (but always at the same time step t_{pred}) would achieve a better classification quality than the proposed agent. To perform the evaluation, we deactivate the decision-making capability of the model, i.e. the RL part, and train a set of equivalent static DNNs to classify at a list of predefined (static) time steps.

5.5.1 Experimental pipeline

Dataset We use the industrial dataset from Sec. 5.4.1, illustrated in Fig. 5.4 and Fig. 5.5.

Early classifier We perform 50 trainings of an EC agent, each being dedicated to set of hyper-parameters $h \in \tilde{\mathcal{H}}$, on the same EC-POMDP definition than Sec. 5.4. We apply DDQN algorithm in batch learning and with prioritized sampling, as introduced in Algo. 5. We use the same training pipeline as in Sec. 5.4.3.1 to obtain early classifier agents enhanced with decision-making capabilities. We simultaneously evaluate the agent's policies on the validation set with the same evaluation pipeline as in Sec. 5.4.3.2. As output of the 50 trainings of the agent, we then obtain a set of policies $\{\{\pi_\Theta\}_h, \forall h \in \tilde{\mathcal{H}}\}$ with variable performances in Acc and t_{pred} on the validation set. On this set of policies, we apply the same pipeline as in Sec. 5.4.3.4 to select and evaluate the agent's optimal policies. We report in Fig. 5.10 the performances in Acc and t_{pred} of these optimal policies on the testing set. Therefore, we visualize the best classification results obtained by the agent for all earliness trade-off.

Static classifier For all time steps $t \in [1, T]$, we train a DNN $f_{t,\Theta}$ with parameters Θ to map between the partial temporal sequences $\mathbf{X}_{:t}$ and the labels $l \in \mathcal{L}$, as illustrated in Fig. 5.9:

$$f_{t,\Theta} : \{X_{:t}\} \rightarrow \mathcal{L} \tag{5.1}$$

Each DNN $f_{t,\Theta}$ is trained on the training dataset $\mathcal{D}_t = \{(\mathbf{X}^n_{:t}, l^n)\}_{n=1..N}$ built from $\mathcal{D}$. The DNNs $\{f_{1,\Theta}, \cdots, f_{T,\Theta}\}$ are trained separately until the loss function stops decreasing on the validation set. As output of the trainings of the static classifiers, we obtain sets of classification models $\{\{f_{1,\Theta}\}, \cdots, \{f_{T,\Theta}\}\}$. For all time steps $t \in [1, T]$, we select among the set of classification models $\{f_{t,\Theta}\}$ the top-5 models with best classification performance on the validation set. We then report in Fig. 5.10 the performances in Acc of these models on the testing set.

As illustrated in Fig. 4.1 and Fig. 5.9, the neural network architectures used for both static classifiers $\{f_{1,\Theta}, \cdots, f_{T,\Theta}\}$ and the agent's policy $\{\pi_\Theta\}$ are similar, with identical convolutional layer architectures. The DNN architectures only differ in the output layer. Indeed, the output layer of the static classifiers has as many neurons as labels and a softmax activation while the output layer of the agent's policy is linear and has an additional neuron for the delay action.

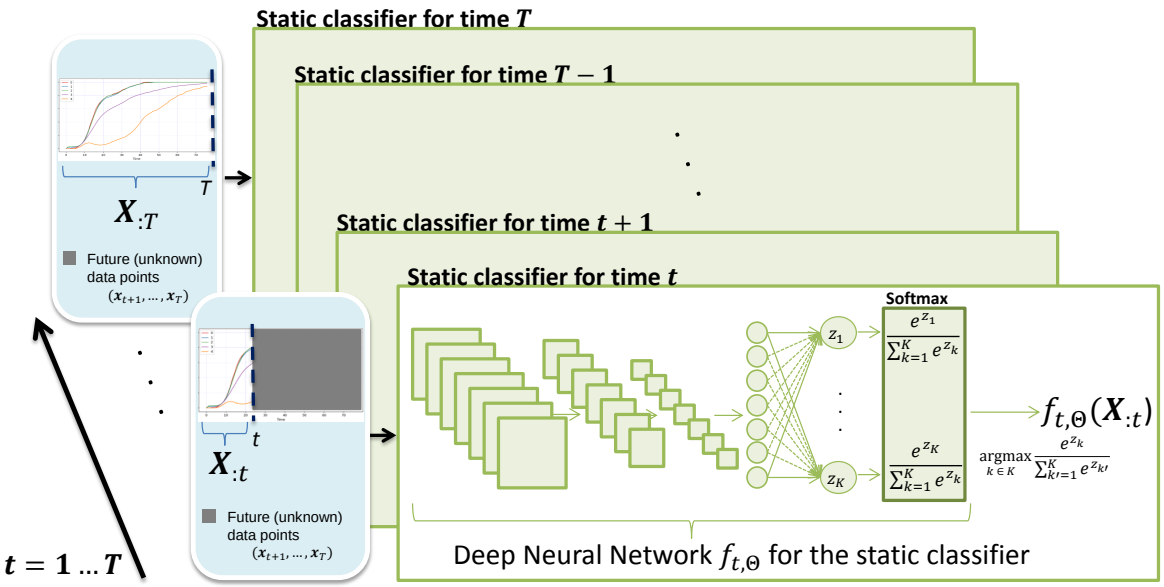

Figure 5.9: **Set of DNNs $\{f_{1,\Theta}, \cdots, f_{T,\Theta}\}$ with parameters Θ trained for static classification at all time steps $t \in [1, T]$.** Each DNN $f_{t,\Theta}$ receives as input data the partial sequences $\mathbf{X}_{:t}$. The DNNs have the same architecture than the policy DNN from Fig. 4.1, except for the output layer which has a softmax activation and K output neurons associated to the K labels (there is no longer a neuron associated with the delay action).

5.5.2 Results

In Fig. 5.10, we report top-5 policies performance for different ranges of t_{pred} (Sec. 5.4.3.2). Both static DNN 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 data points on sequences lacking discriminant patterns. As a consequence, the early classifier's capacity to individually compromise makes the classification more efficient than static networks using the same amount of data points in all sequences independently of their complexity. In the next chapter, we will illustrate some of the agent's predictions on test data and we will seek to interpret its decisions, individually on each data.

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.

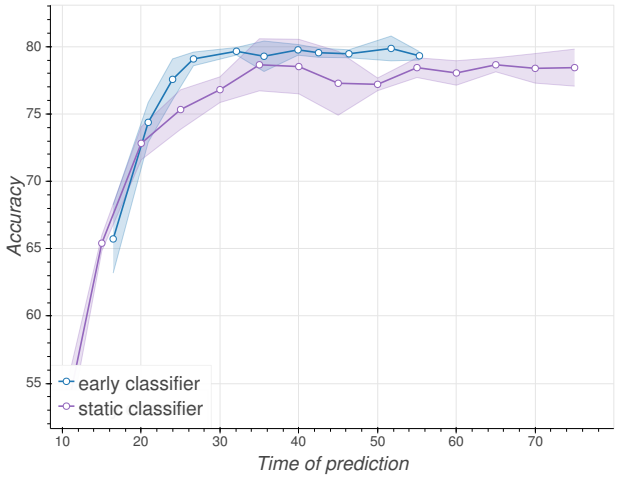

Figure 5.10: **Evaluation of top-5 policies of the early classifier agent and top-5 static DNN classifiers.** The early classifier agent is trained with Algo. 5. We select the top-5 policies and top-5 static classifiers in Acc on the validation set for several ranges of t_{pred}. We evaluate those policies and static classifiers on the test set. The full line represents mean Acc and the band is the stdev Acc.

5.5.3 Remark on external analysis

We emphasize that other types of neural networks are suitable for the EC problem on temporal sequences, other than Convolutional Neural Networks (CNNs) as used in this experimental evaluation. In an external analysis that is not presented in this document, we compared different DNN architectures for the EC problem on the industrial dataset. Specifically, we evaluated DNNs with Long Short Term Memory units [35], a type of Recurrent Neural Network (RNN). We experimentally observed that these RNNs did not bring significant improvement over the tested CNNs. Our intuition behind this result is that MTS are 2D matrices that offer the opportunity to treat them as images, a data type on which CNNs are known to be efficient prediction models. Indeed, one dimension of the 2D matrix relates to the time in the temporal sequence and the other dimension relates to the different features. From their convolution operations, CNNs can identify discriminant groups of contiguous data points which, in the case of sequences, are data points that are collected at consecutive time steps. Given these convolution operations applied both in the feature and time axes, CNNs are therefore able to learn temporal dependencies and features dependencies in MTS. We point out that other work from the literature [103] classify temporal sequences with CNNs and show satisfying classification performance on the UCR public benchmark, experimentally demonstrating that these networks can learn discriminant temporal features.

5.6 Conclusion

In this chapter, we optimized the DDQN algorithm which has imbalanced memory issues when applied to the problem of EC. We proposed different strategies for robust memory management: prioritized sampling, prioritized storing and random episode initialization. In experiments, we showed that the different strategies for memory management statistically

improved the performance of the agent in terms of accuracy and speed. Finally, we compared early classifiers trained with RL to static DNNs and we showed that the early classifiers capacities of adaptive prediction time improved the general trade-off of accuracy versus speed.

Chapter 6

Policies interpretation

In previous chapters, we defined early classification (EC) as a sequential decision-making problem and we described it by a Partially Observable Markov Decision Process (POMDP), noted EC-POMDP. We solved the EC-POMDP by training an agent with Reinforcement Learning (RL) and we showed that it can propose a range of policies illustrating the trade-off between classification accuracy and earliness. We remind that the agent's policy is a Deep Neural Network (DNN) $Q_\Theta(o, a)$ with parameters Θ which aims at approximating the action value function $Q(o, a)$ (Eq. 3.16).

In this chapter, we are interested in interpreting the policy of the agent and explaining its predictions and choice of actions. A first interpretation tool is given in Sec. 6.1 where we apply a method from the literature to identify the data points in partial sequences that influenced the most the agent's predictions. Illustrations are given on the industrial EC dataset. To enrich the policies interpretations, we visualize the Q-values estimated by the agent over the sequence acquisition in Sec. 6.2. Finally, in Sec. 6.3, we apply a method from the literature to estimate the uncertainty of the agent's predictions. A first experimental study is given on a simplified version of the industrial dataset for which there are less labels to predict.

6.1 Visualization of Class Activation Map

6.1.1 Method presentation

In [116], the authors introduce the Class Activation Map (CAM) method which seeks to interpret the predictions of a neural network (NN). A CAM is a matrix of value which allows to detect the data points in an input data that activate the most a specific output neuron of the NN. Specifically, it allows to decompose the final value predicted by an output neuron in relation to the data points identified in the input data. A CAM has as many values as data points in the input data, and each value gives the contribution produced by the associated input data point on the value predicted by the output neuron.

In Fig. 6.1 from [116], the authors apply the CAM method on the problem of images classification. As illustrated in the figure, in the case of classification problems, each output neuron of the NN is associated with a label and predicts a probability score for this label.

The CAM method therefore makes it possible to identify the data points in a sample that contributes to increase or decrease the probability associated to a specific label. We refer to a contribution that increases (resp. decreases) the final value of the output neuron of interest

as a *positive contribution* (resp. *negative contribution*). In other words, a positive (resp. negative) contribution implies that the odds of choosing this label increase (resp. decrease).

On a test data representing a child sitting in a car next to a dog, the authors manage to highlight the discriminating pixels that are used by the NN to predict a probability score for the label *Australian terrier*. The CAM (which is the image to the right of the equation) shows that the pixels contributing the most positively to the label prediction are those representing the dog's face, and specifically its muzzle. A part of the dog's body also contribute positively to the label prediction. On the contrary, the rest of the pixels including the child and the car contribute negatively to the label prediction.

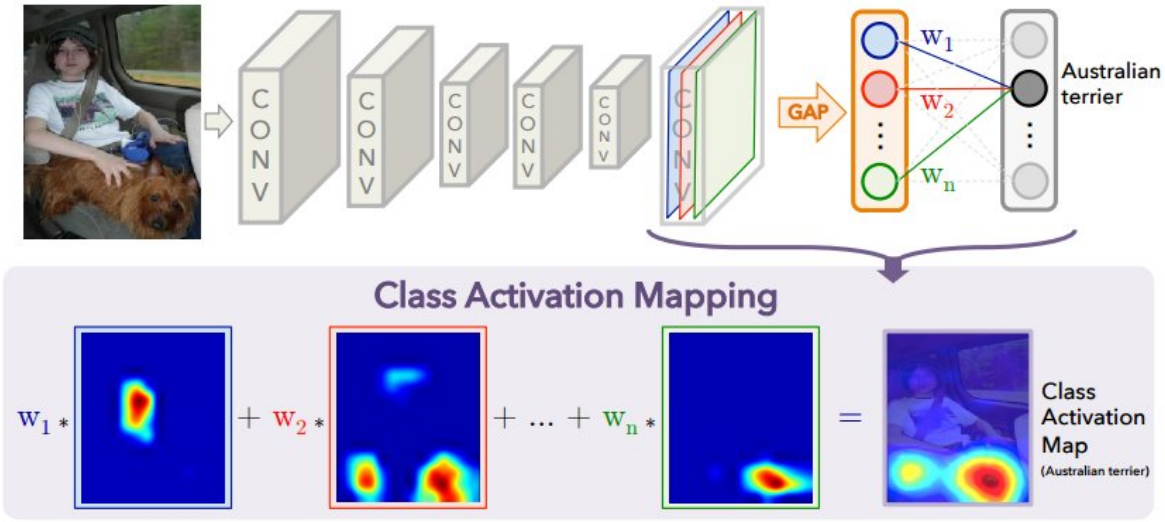

Figure 6.1: **Class Activation Map [116].** The predicted probability of the label *Australian terrier* is mapped back to the previous convolutional layer to generate the CAM which highlights the class-specific discriminating regions. Pixels highlighted in red are the most positively contributing ones for the prediction of label *Australian terrier*.

The calculation of the CAM on the example shown in the figure is the following:

- First, the authors train a NN for the problem at hand (classification in the example). Specifically, a restriction of the CAM method is to use a NN for which its last convolutional layer is connected to a Global Average Pooling (GAP) layer, itself fully-connected to the output layer.
- Then, they feed-forward the input test data (an image of the child sitting next to his dog in the example) to the trained NN.
- They collect the feature maps from the last convolution layer of the NN, when applied to the test data. There are as many feature maps as convolution filters in the layer, and they are the convolution results of these filters.
- For an output neuron of interest (the one associated to the label *Australian terrier* in the example), they collect the weights which bind each neuron in the GAP layer to the output neuron. For this output neuron, there are as many weights as feature maps in the last convolutional layer.

- For each feature map in the last convolutional layer, they multiply it by the weight associated with it for the output neuron of interest.
- Finally, they compute the weighted sum of the feature maps to obtain the CAM.
- In practice, if the features maps in the last convolution layer are smaller than the original input data (because of pooling operations), they up-sample the CAM to the size of the input data.

As a result, in the final map, there is one activation value per data point in the input data. Mathematical formulations can be found in [116] and will be given for the EC problem in the following.

6.1.2 Motivation

In this doctoral work, we use a DNN Q_Θ with parameters Θ for the approximation of the Q-value and for the definition of the agent's policy π_Θ (Eq. 4.1). The network differs from the ones used in [116] for classification, because Q_Θ is trained to predict Q-values of actions $a \in \mathcal{A}$ and not to predict probabilities on labels $l \in \mathcal{L}$ directly. As shown in Fig. 6.2, each neuron in the last layer of Q_Θ is associated with an action $a \in \mathcal{A}$ and predicts its Q-value. We remind that the action space $\mathcal{A}$ is defined over the set of labels $\mathcal{L}$ plus an additional action for delay a_d, $\mathcal{A} = \mathcal{L} \cup a_d$ (Eq. 3.1, Eq. 3.2). Q_Θ has therefore an output neuron for each label of the classification problem, plus a neuron for the delay action.

In this work, we argue that applying the CAM method on Q_Θ allows to explain the Q-value predictions for each action, and specifically for the ones associated to labels prediction. We remind that input data to the DNN are prefixes $\mathbf{X}_{:t} \in \mathbb{R}^{P \times t}$ of MTS $\mathbf{X}$ (Eq. 2.1, Fig. 2.1), with $P \in \mathbb{N}^+$ features and length $t \in [1, T]$ ($T \in \mathbb{N}^+$ being the maximal length of the sequences):

$$\mathbf{X}_{:t} = \begin{pmatrix} x_1^1 & \cdots & x_t^1 \\ \vdots & \ddots & \vdots \\ x_1^P & \cdots & x_t^P \end{pmatrix}_{t \leq T}$$

Each prefix $\mathbf{X}_{:t}$ is then a 2D matrix that can then be treated similarly to images in [116] for the application of the CAM method. Instead of identifying discriminating pixels, the method makes it possible to identify discriminating data points $x_t^p \in \mathbf{X}, p \in [1, P], t \in [1, T]$ in the sequences. Specifically, because the DNN Q_Θ uses convolutional filters, the method can identify a discriminant group of contiguous data points and, in the case of sequences, discriminant data points that are collected at consecutive time steps.

In the remainder of the study, we propose to apply the CAM method on test MTS, as part of the agent's policy π_Θ evaluation. The objective is to interpret the choice of action made by the agent on the partial sequences at test time. In particular, we seek to identify which data points $x_t^p \in \mathbf{X}, p \in [1, P], t \in [1, T]$ have the most effect on the Q-values predicted by the DNN Q_Θ.

6.1.3 CAM application to EC

Policy Let π_Θ be the policy of the agent that we wish to apply on test data, and Q_Θ the DNN associated to π_Θ following Eq. 4.1. As mentioned above, a restriction of the CAM

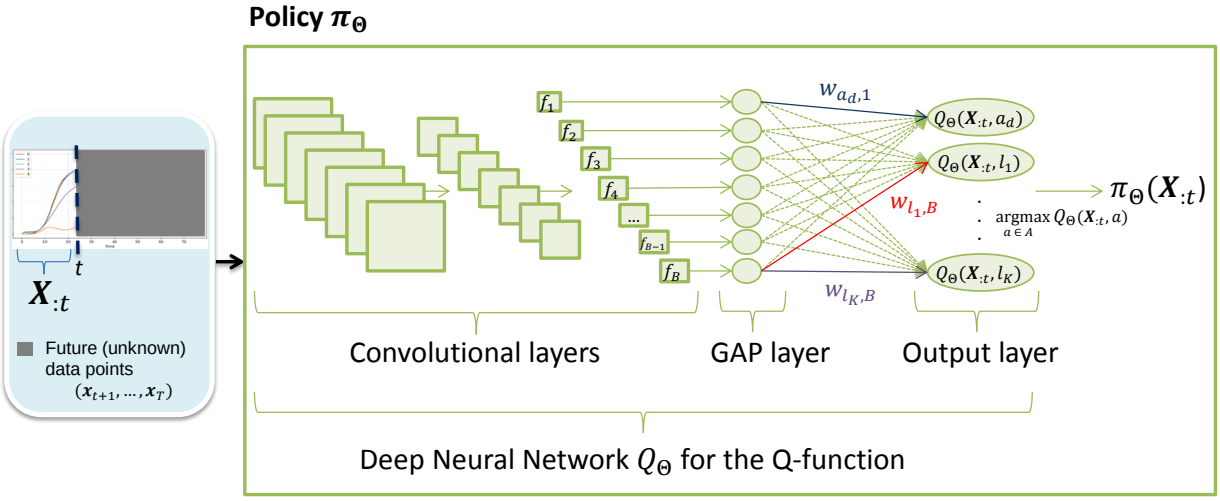

Figure 6.2: **DNN Q_Θ for the Q-function with parameters Θ.** It is defined over the set of observations $\mathcal{O}$ (Eq. 3.25). Given an input observation $o = \mathbf{X}_{:t}$, the output layer of the DNN predicts the Q-values for all actions $a \in \mathcal{A}$. The output layer is fully-connected to a GAP layer which reduces each feature map from the previous convolutional layer $F = \{f_1, \cdots, f_B\}$ to a single number representing the average of the feature map.

method is to use a DNN for which its last convolutional layer F is connected to a GAP layer, itself fully-connected to the output layer. As a consequence, the DNN architectures used in this doctoral work verify this condition, as illustrated in Fig. 6.2.

The output layer of Q_Θ differs from those used in [116]. Instead of fully-connecting the GAP layer to a softmax layer for classification, we fully-connect the GAP layer to an output layer with a linear activation in order to predict Q-values that can take values in $\mathbb{R}$.

The last convolutional layer in Q_Θ is noted F and is composed of B convolution filters:

$$F = \{f_1, \cdots, f_B\} \tag{6.1}$$

with f_b the b-th convolution filter, as illustrated in Fig. 6.2. The value of B is part of the hyper-parameters of the method, as well as the size of convolution filters and the operations to apply in relation to convolutions (padding, pooling, etc.).

Test data Let $\mathbf{X} \in \mathbb{R}^{P \times T}$ be a test MTS on which we seek to interpret the agent's predictions. At each time step t of the testing episode, the agent observes the prefix $\mathbf{X}_{:t}$, predicts Q-values for each action $a \in \mathcal{A}$ and chooses the action with maximal Q-value. We note t_{pred} the time of prediction and $\widehat{l_{pred}} \in \mathcal{L}$ the label predicted by the agent at that time, using the prefix $\mathbf{X}_{:t_{pred}}$. The objective is to find the data points in $\mathbf{X}_{:t_{pred}}$ that contributed the most to the prediction of label $\widehat{l_{pred}}$.

CAM calculation Following the CAM method from [116], we compute the map M_a for action $a \in \mathcal{A}$ on the prefix $\mathbf{X}_{:t_{pred}}$ as follows:

$$M_a(\mathbf{X}_{:t_{pred}}) = \sum_{b=1}^{B} w_{a,b} * f_b(\mathbf{X}_{:t_{pred}}) \tag{6.2}$$

where

- $f_b(\mathbf{X}_{:t_{pred}})$ is the feature map of the b-th convolution filter from F layer, given input data $\mathbf{X}_{:t_{pred}}$. Each feature map is a matrix whose size depends on the convolution operations performed in previous layers.

- $w_{a,b}$ is the scalar weight of the fully-connected layer which binds the GAP neuron of f_b to the Q-value of action a, as illustrated in Fig. 6.2.

At that stage, the map M_a is the size of convolution filters from F. The map M_a is then upsampled to the size of the input data $\mathbf{X}_{:t_{pred}}$, so that $M_a(\mathbf{X}_{:t_{pred}}) \in \mathbb{R}^{P \times t_{pred}}$:

$$M_a(\mathbf{X}_{:t_{pred}}) = \begin{pmatrix} m_1^1 & \cdots & m_{t_{pred}}^1 \\ \vdots & \ddots & \vdots \\ m_1^P & \cdots & m_{t_{pred}}^P \end{pmatrix} \tag{6.3}$$

with m_t^p the map value for the data point $x_t^p \in \mathbf{X}_{:t_{pred}}$. In order to obtain a map easily interpretable by the user, we average the map values on all features $p \in [1, P]$, at each time step $t \in [1, t_{pred}]$:

$$M_a(\mathbf{X}_{:t_{pred}}) = \big(\sum_{p=1}^{P} m_1^p/P, \cdots, \sum_{p=1}^{P} m_{t_{pred}}^p/P\big) \tag{6.4}$$

Consequently the map becomes a vector of length t_{pred} and allows to identify important time steps in the sequence. Finally, in order to compare the CAMs of each action, the maps are normalized:

$$M_a(\mathbf{X}_{:t_{pred}}) = M_a(\mathbf{X}_{:t_{pred}}) / \max_{\tilde{a} \in \mathcal{A}} \|M_{\tilde{a}}(\mathbf{X}_{:t_{pred}})\|_\infty \tag{6.5}$$

6.1.4 CAM illustrations on the industrial dataset

Dataset In this experimental evaluation, we apply the CAM method for EC presented above on the industrial dataset of the thesis. It involves a set of MTS $\mathbf{X} \in \mathbb{R}^{5 \times 77}$ associated to ordinal labels in $\mathcal{L} = \{l_1, \cdots, l_7\}$ such that $l_1 < l_2 < ... < l_7$. Its distribution between the different labels is shown in Fig. 6.3

In Fig. 6.4, the training set is represented with a two-dimensional t-SNE embedding of the (complete) temporal sequences using algorithm from [64]. We observe overlapping clusters of points from different labels which illustrates the dataset complexity. In the industrial EC application, samples within a same class can be more or less distant due to biological variability, leading to an overlap with other classes that can sometimes be important.

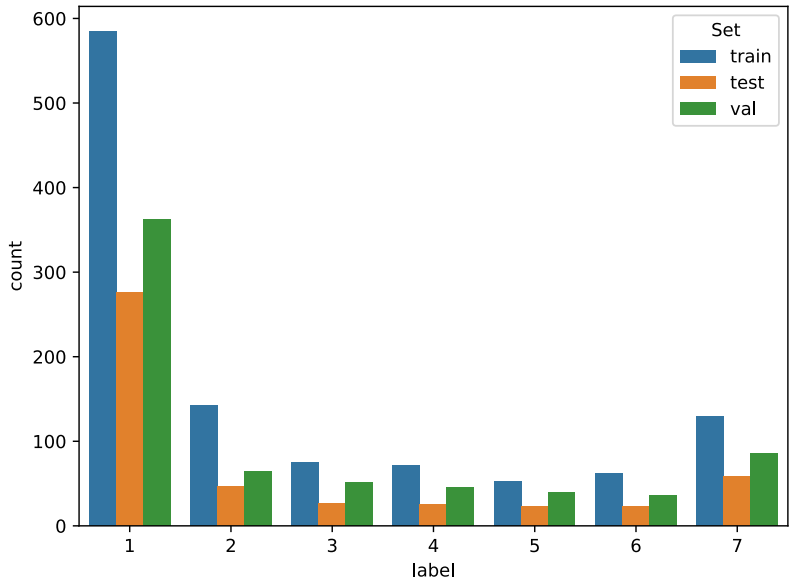

Figure 6.3: **Labels distribution in the industrial sets of training, validation and testing.** The set of labels is $\mathcal{L} = \{l_1, \cdots, l_7\}$

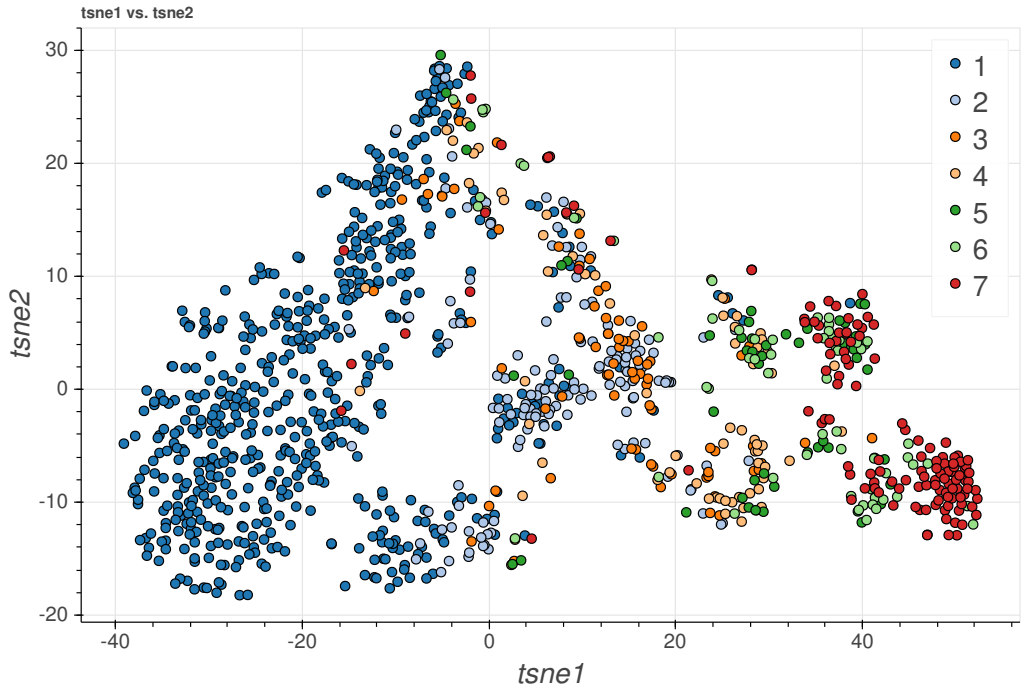

Figure 6.4: **Two-dimensional t-SNE embedding of the training dataset.** Each dot represents a complete temporal sequence $X_{:T}^{n}, n \in [1, N]$ from the industrial training set associated to an ordinal label in $\mathcal{L} = \{l_1, \cdots, l_7\}$. Each dot is colored according to the label.

Agent's training As in previous chapters, we train a set of agents on the training set, each one being dedicated to a set of hyper-parameters (see Sec. 5.4.3.1). We regularly evaluate the independent agents on the validation set (see Sec. 5.4.3.2). Among all the evaluated policies, we select one which satisfies our will to compromise between classification quality and earliness on the validation set. In the case of the industrial application, the classification quality is assessed with application-specific performance criteria that measure the severity of misclassification. The goal is now to apply this selected optimal policy on test data and interpret its predictions. Using the CAM calculation method presented previously, we calculate the CAM on test data in order to interpret the Q-values predicted by the agent. In Figs. 6.6, 6.7, 6.5, we visualize some input partial sequences analyzed by the agent and we superpose the CAMs for each action for these sequences.

Results In Fig. 6.5, the test data is associated with a true label l_5. It is acquired until time step $t_{pred} = 46$, time at which the agent correctly predicts label l_5. The explanation given by the CAM method for this prediction is as follows:

- The CAM shows that a pattern in the data points $(\mathbf{x}_{39}, \cdots, \mathbf{x}_{46})$ contribute the most negatively to the Q-value of labels l_1 and l_2. It also contributes negatively, but with less importance, to the Q-value of label l_3. It slightly contributes positively to the Q-value of label l_4, and largely positively to those of labels l_5, l_6 and l_7.
- We also identify that a pattern before the data points $\mathbf{x}_{39}$, around $(\mathbf{x}_{20}, \cdots, \mathbf{x}_{38})$, slightly decreases the Q-value of label l_7.
- Previous data points $(\mathbf{x}_1, \cdots, \mathbf{x}_{20})$ have a zero contribution to the predicted action values for all actions. This is also the case of unknown future data points $(\mathbf{x}_{50}, \cdots, \mathbf{x}_{77})$ that have been replaced by zeros: they do not contribute positively nor negatively to the Q-values of all actions.

In Fig. 6.6, the test data is associated with a reference label l_1. It is acquired until time step $t_{pred} = 34$, time at which the agent misclassifies the data by predicting label l_7. The CAM shows that the DNN Q_Θ identifies a pattern in the data points $(\mathbf{x}_{26}, \cdots, \mathbf{x}_{34})$ that largely reduces the Q-value of labels l_1, l_2, and moderately reduces the Q-value of label l_3. It also increases the Q-value of label l_5, and even more those of labels l_6 and l_7.

From this CAM and its identified patterns, the user is guided towards the explanation of the DNN misclassification. He can conclude on a bad training of the DNN if he recognizes a usual pattern for data labelled l_1. Otherwise, he can question the adequacy of the reference label on this test data, in the presence of the recognized pattern. On this industrial data and from our data expertise, we recognize that the pattern identified by the DNN is indeed atypical for a sample data labelled l_1, and is more generally characteristic of data associated with larger labels.

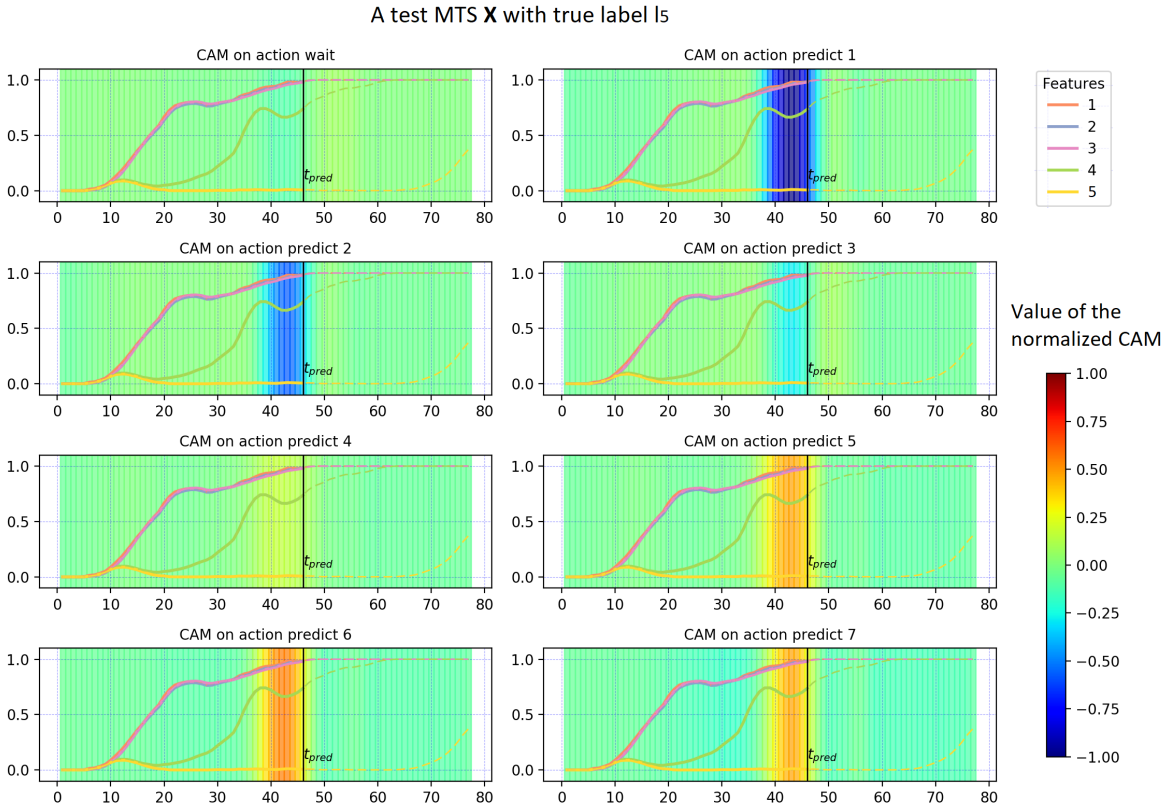

Figure 6.5: **CAMs for each action $a \in \mathcal{A}$, on a partial test MTS $\mathbf{X}_{:46}$ with reference label l_5.** The agent predicted label l_5 at $t_{pred} = 46$. CAMs in red (resp. blue) highlight the patterns in the MTS that contribute positively (resp. negatively) to the predicted Q-values. In full line are the data points $(\mathbf{x}_1, \cdots, \mathbf{x}_{t_{pred}})$ that have been observed until the prediction of a label by the agent. In dashed line are the data points $(\mathbf{x}_{t_{pred}+1}, \cdots, \mathbf{x}_T)$ that would have been observed if sequence acquisition had continued.

In Fig. 6.7, the test data is associated with a reference label l_1. It is acquired until the end of the sequence, at time step $t_{pred} = 77$, time at which the agent is forced to predict a label. It predicts label l_1. The CAM globally shows that several patterns are identified in the sequence and contribute negatively to the Q-value of classification actions, which therefore remain lower than the Q-value for delay. Indeed:

- The data points $(\mathbf{x}_{10}, \cdots, \mathbf{x}_{20})$ make a large negative contribution to the Q-values of labels l_2 and l_3 and l_4. The latter are not increased by the identification of positive patterns.
- The same data points $(\mathbf{x}_{10}, \cdots, \mathbf{x}_{20})$ contribute negatively to the Q-value of label l_1. The latter is however increased by a pattern in $(\mathbf{x}_{60}, \cdots, \mathbf{x}_{70})$ but not enough to exceed the Q-value of delay.
- The data points $(\mathbf{x}_{10}, \cdots, \mathbf{x}_{77})$ contribute negatively to the Q-value of labels l_5 and l_6. The data points $(\mathbf{x}_{18}, \cdots, \mathbf{x}_{77})$ contribute negatively to the Q-value of label l_7.

We globally observe that the shape of the whole sequence (that we know atypical) contributes

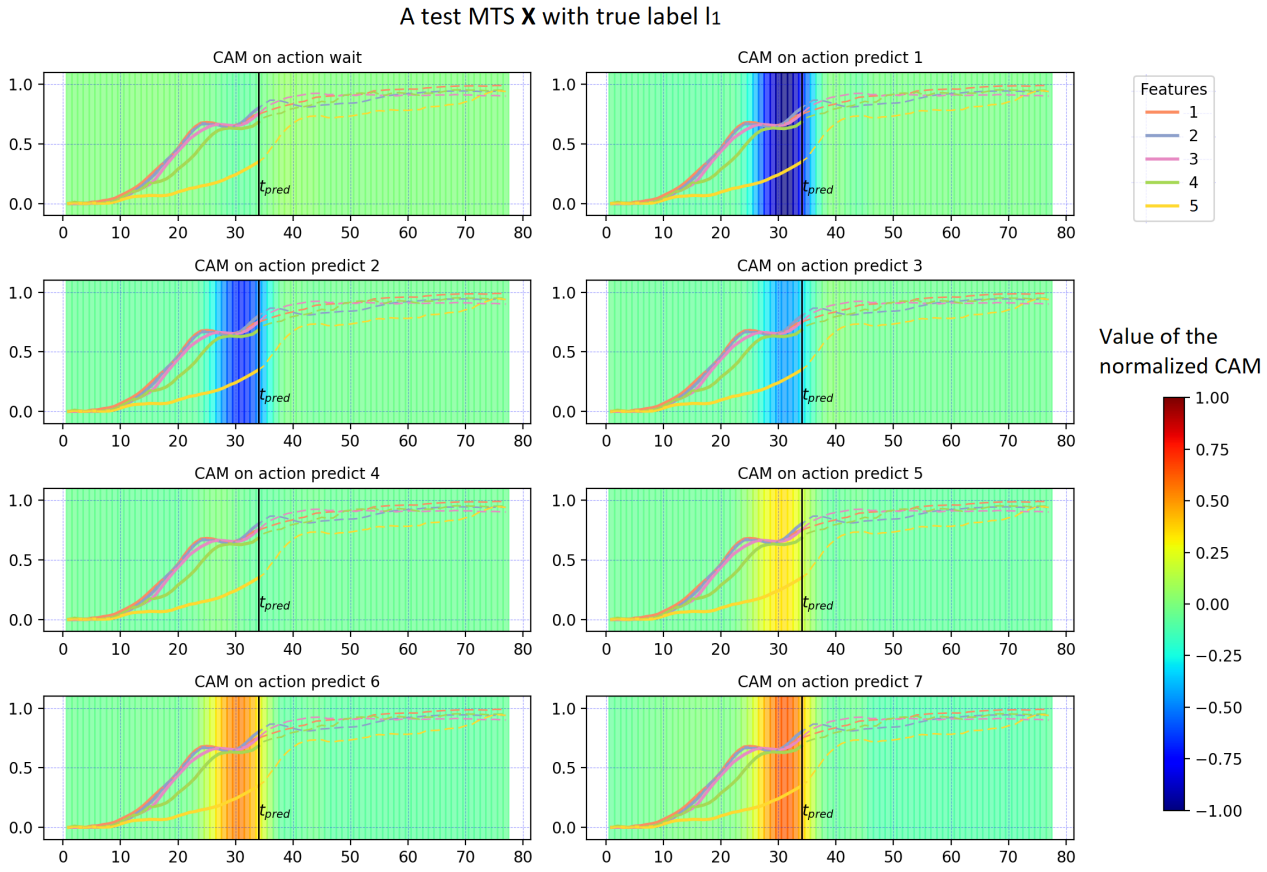

Figure 6.6: **CAMs for each action $a \in \mathcal{A}$, on a partial test MTS $\mathbf{X}_{:34}$ with reference label l_1.** The agent predicted label l_7 at $t_{pred} = 34$. CAMs in red (resp. blue) highlight the patterns in the MTS that contribute positively (resp. negatively) to the predicted Q-values. In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition had continued.

to the Q-value estimations. In previous examples, where the agent predicted before the end of the sequence, it was often the last collected data points that were the most meaningful in the decisions. Appendix C gives more illustrations of CAMs on test data from the industrial application.

6.1.5 Conclusion

As illustrated during the experimental evaluation, the application the CAM method on Q_Θ allows to identify how data points in sequences contribute to the Q-value predictions. We were able to draw some interpretations on the decisions made by the agent. This method can be used for two purposes. First, it can be used when applying a policy on test data, in order to provide a rationale for the agent's decisions. Or it can be used as a second stage of optimal policy selection, to compare several successful policies and select the one for which the decisions are most consistent with the application, based on our data expertise.

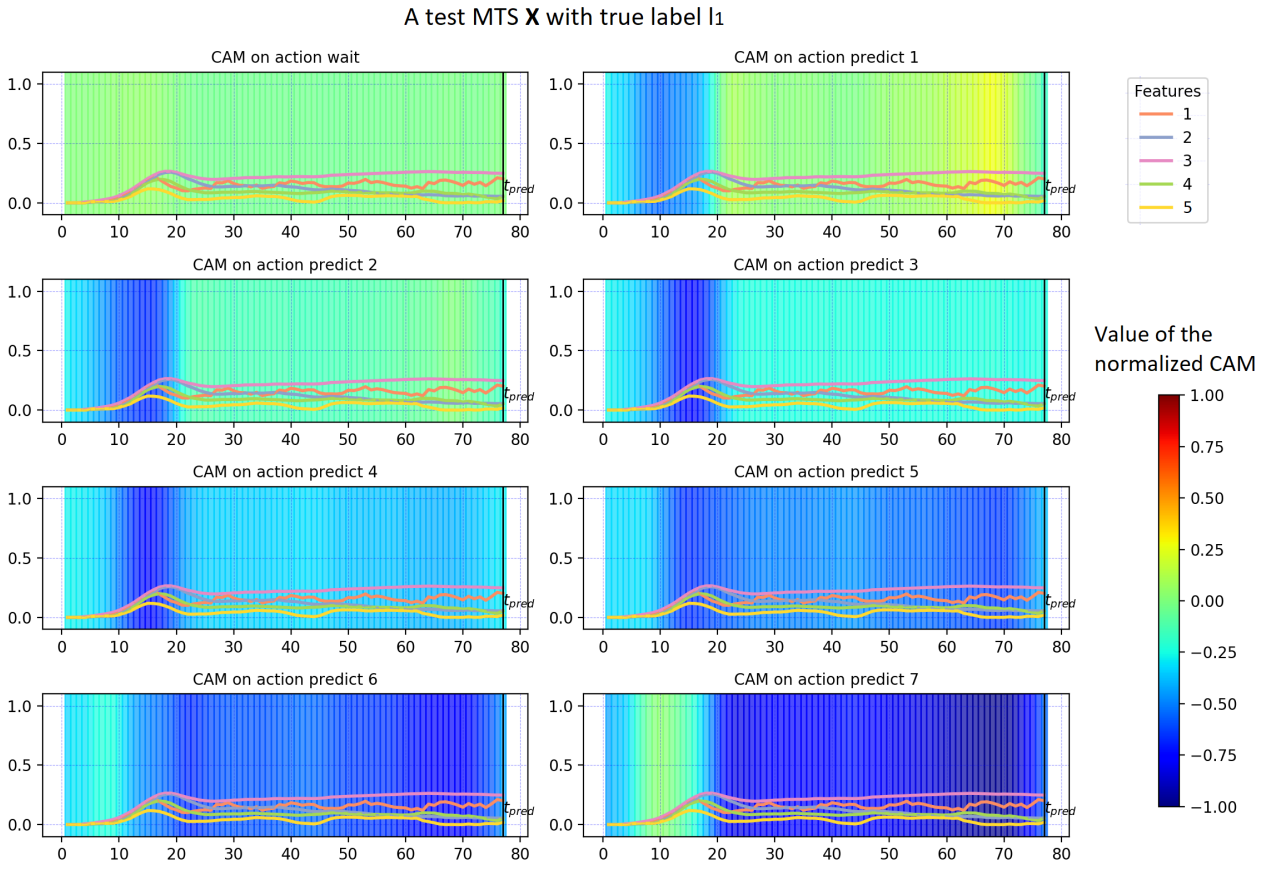

Figure 6.7: **CAMs for each action $a \in \mathcal{A}$ on a partial test MTS $\mathbf{X}_{:77}$ with reference label l_1.** The agent is forced to predict a class label at the end of the sequence acquisition. It predicts label l_1 at $t_{pred} = 77$. CAMs in red (resp. blue) highlight the patterns in the MTS that contribute positively (resp. negatively) to the predicted Q-values. In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition had continued.

6.2 Visualization of Q-values

In parallel to CAMs, we propose to visualize the Q-values predicted by the agent on test data, in order to go further in the evaluation of a policy and the interpretation of its decisions. In the following, we visualize the Q-values estimations made by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition, on test data from the same industrial dataset as in Sec. 6.1 and on the same policy.

We remind that the agent, whose policy is given by the DNN Q_Θ, has been trained for an EC problem involving a set of MTS $\mathbf{X} \in \mathbb{R}^{5\times 77}$ associated to ordinal labels in $\mathcal{L} = \{l_1, \cdots, l_7\}$ such that $l_1 < l_2 < ... < l_7$. We point out that a cost-sensitive reward function (Eq. 3.32) has been used in these experiments, due to the cost-sensitive nature of the classification problem (Sec. 2.2.2.4). Rewards used for this cost-sensitive learning are shown in the following figures (see the "*Rewards*" graph in Fig. 6.8, Fig. 6.9, Fig. 6.10). When the agent misclassifies, rewards vary according to the true and predicted labels.

Fig. 6.8 involves the same test data as in Fig. 6.5 where we applied the CAM method. This test data is associated with a reference label l_5. It is acquired until time step $t_{pred} = 46$, time at which the agent correctly predicts label l_5. We observe that:

- From time step 10, the Q-values of labels l_2, l_3 and l_4 increase and seem to stand out compared to other classification actions, but not sufficiently to make a prediction. In other words, the data points $(\mathbf{x}_{10}, \cdots, \mathbf{x}_{20})$ encourage the prediction of a label in $\{l_2, l_3, l_4\}$.
- From time step 25, the Q-values of labels l_2 and l_3 start decreasing. Those of labels l_4, l_5 and l_6 finally exceed them from time step 35. The Q-value of label l_5 remains higher to the others and is the first to exceed that of delay, at time step 46.

Globally, from time step 35, the agent manages to predict Q-values that are ordered, as in the reward function definition represented in the bottom figure.

In Fig. 6.9, the test data is associated with a reference label l_1. It is acquired until time step $t_{pred} = 48$, time at which the agent correctly predicts label l_1. We observe that:

- From time step 20, the Q-values of labels l_5, l_6 and l_7 get closer to the true rewards for these actions, represented in the bottom figure, which are the worst rewards that the agent can receive. Consequently the agent manages to eliminate these labels from possible predictions.
- Also, from time step 20, the Q-values associated to labels l_1, l_2 and l_3 increase, with that of label l_1 being always superior to the two others.

Globally, the agent manages to predict Q-values that are ordered, as defined in the reward function. Moreover, the difference between the predicted Q-values is close to the difference between the true classification rewards, which shows that the agent manages to learn an ordinal relationship between the labels.

The Q-values show that, from time 20, the agent predicts a Q-value for label l_1 which is very close to that of the delay action, but does not exceed it until time 48.

Given the predicted order in Q-values for each label, and given the shape of the Q-value curve for label l_1, we can see that the agent manages to identify the correct label early in the sequence, from time step 20. However, it does not make its predictions until later, after almost 30 additional acquisitions in the sequence, due to a late crossing of the Q-value for label l_1 on that ot the delay action.

In Fig. 6.10, the test data is associated with a reference label l_3. It is acquired until time step $t_{pred} = 55$, time at which the agent misclassifies by predicting label l_2. We observe that:

- From time step 25, the Q-values of labels l_1, l_4, l_5, l_6 and l_7 get closer to the true rewards for these actions, which are the worst that the agent can receive. Consequently the agent manages to eliminate these labels from possible predictions.
- Also, from time step 25, the Q-values of labels l_2 and l_3 increase and are almost similar, showing that the agent hesitates between these two labels.

- At time step 55, the Q-value of labels l_2 and l_3 exceed that of delay. The agent predicts label l_2 for which the Q-value is slightly superior.

Given the predicted order in Q-values for each label, and given the shape of the curves for labels l_2 and l_3, we can see that the agent manages to reduce its classification problem to two labels, early in the sequence, around time step 35. From that time, it hesitates between two labels and never seems to clearly distinguish between the two. This hesitation is not surprising because data associated to labels l_2 and l_3 can be very alike, as illustrated in Fig. 6.4.

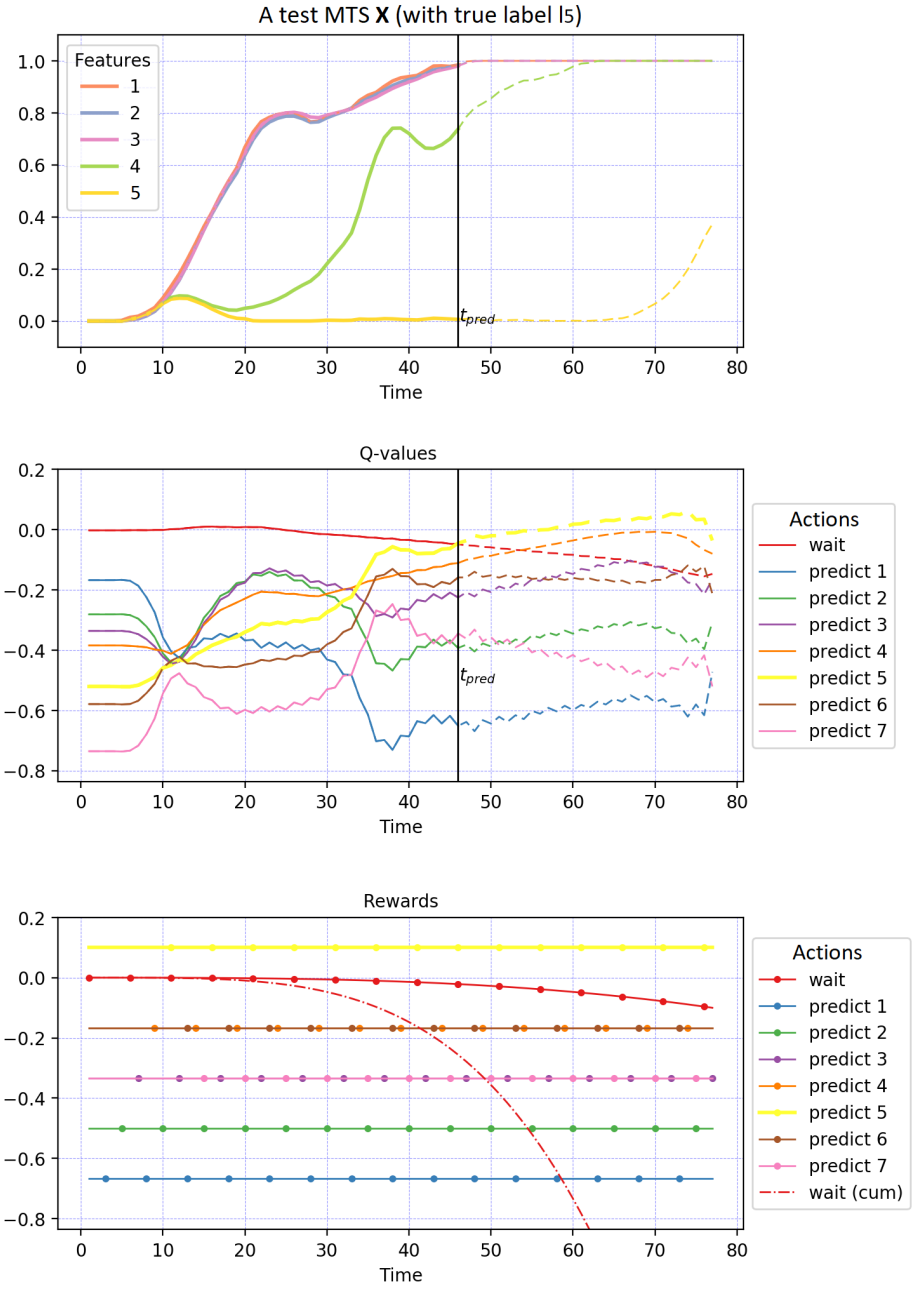

Figure 6.8: **Top**: **A test MTS $\mathbf{X}_{:46}$ with true label l_5.**
Middle: **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.** The sequence acquisition stops at time $t_{pred} = 46$ when the Q-value of the classification action for the label l_5 outreaches the Q-value of the delay action a_d.
Bottom: **Rewards definition for each action $a \in \mathcal{A}$.** Rewards for classification actions are independent of time. Rewards for the delay action decrease over the sequence acquisition. In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

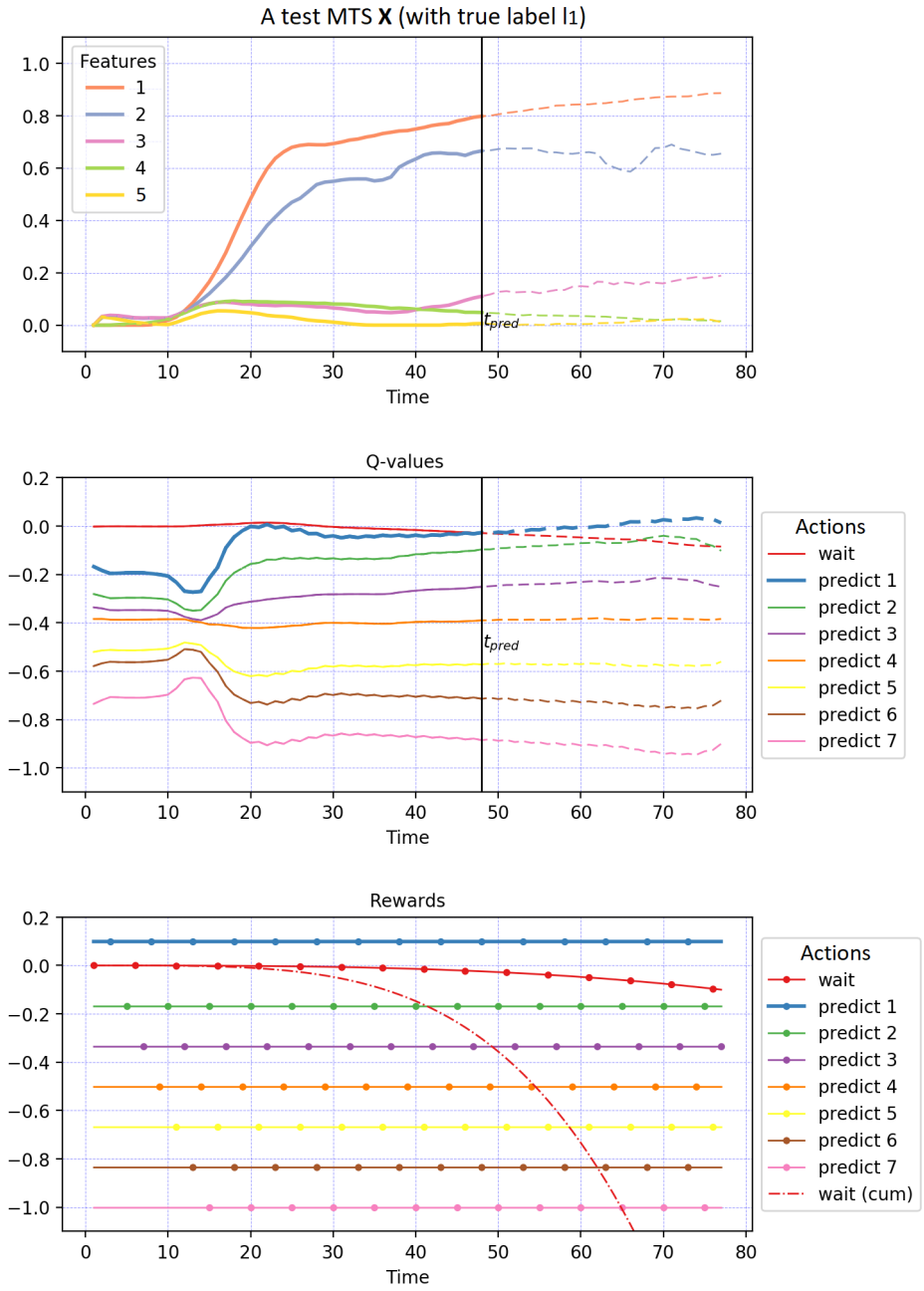

Figure 6.9: **Top**: **A test MTS $\mathbf{X}_{:48}$ with true label l_1.**
Middle: **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.** The sequence acquisition stops at time $t_{pred} = 48$ when the Q-value of the classification action for the label l_1 outreaches the Q-value of the delay action a_d.
Bottom: **Rewards definition for each action $a \in \mathcal{A}$.** Rewards for classification actions are independent of time. Rewards for the delay action decrease over the sequence acquisition. In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

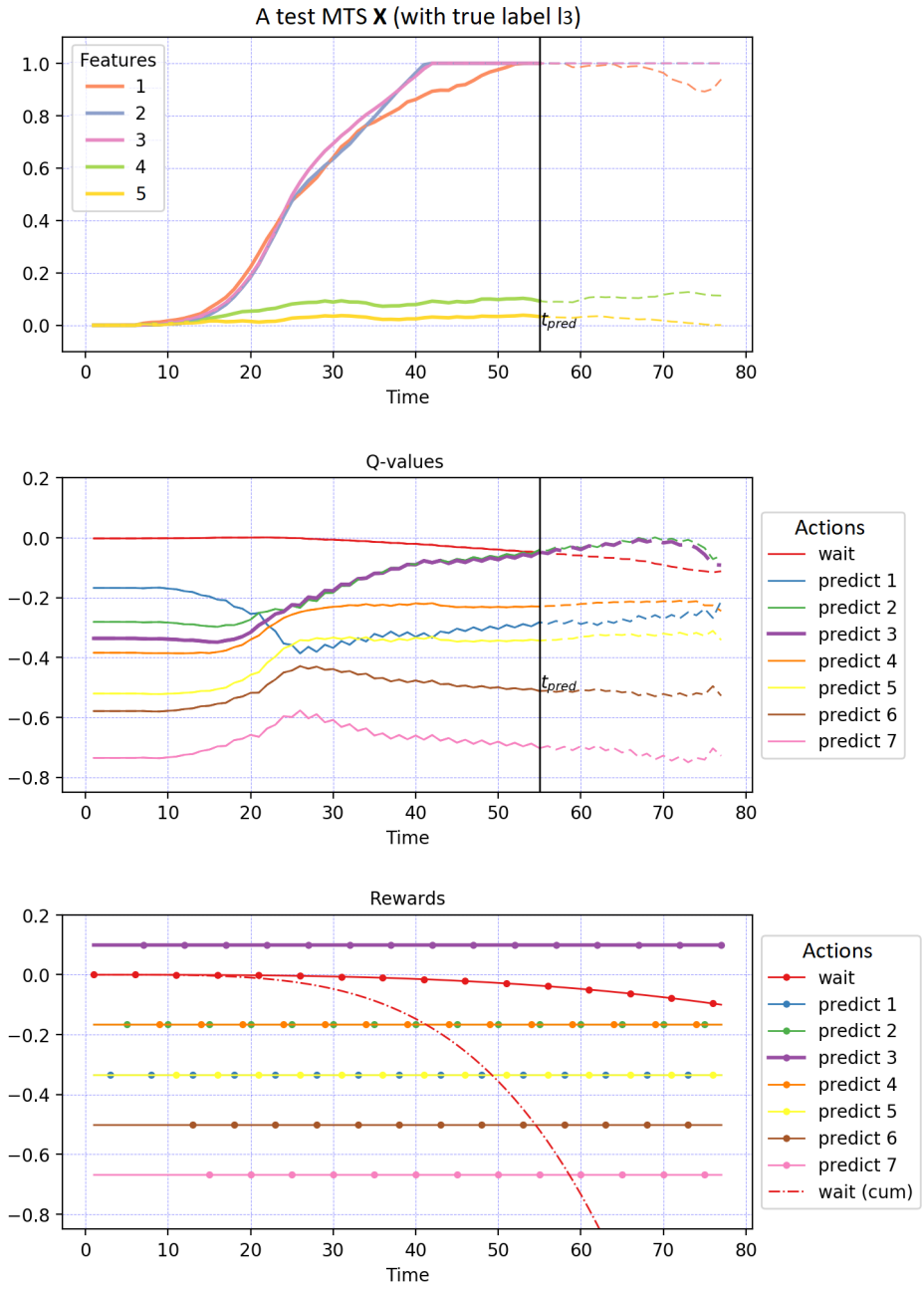

Figure 6.10: **Top**: **A test MTS $\mathbf{X}_{:55}$ with true label l_3.**
Middle: **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.** The sequence acquisition stops at time $t_{pred} = 55$ when the Q-value of the classification action for the label l_2 outreaches the Q-value of the delay action a_d.
Bottom: **Rewards definition for each action $a \in \mathcal{A}$.** Rewards for classification actions are independent of time. Rewards for the delay action decrease over the sequence acquisition. In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

6.3 Perspectives on calculating policy uncertainty

6.3.1 Motivation

In previous experimental evaluation, we visualized the predictions of Q-value made by the agent on test data from the industrial application, and we observed the following cases. First, it can be observed that the Q-value of the delay action is often close to the highest classification Q-value. In some cases, the greatest Q-value of classification remains slightly lower than the Q-value of delay, for many successive time steps, as illustrated in Fig. 6.9. As a consequence, during these time steps, the agent continues to choose the delay action while its Q-value predictions show that it has successfully managed to discriminate the labels. It is therefore a waste of time on the part of the agent, since a good classification could have taken place more quickly. Second, it can be observed that the agent manages to predict Q-values that are ordered, as defined in the reward function, but yet sometimes hesitates between several labels, as illustrated in 6.10. For these particular cases of Q-value predictions, we would like to justify if the agent keeps waiting because it is uncertain in the label for the test data or because of a sub-optimal training. We therefore seek to measure the DNN uncertainty in order to provide additional interpretation in the decisions of the agent.

Moreover and as a secondary objective for a future work, our motivation behind uncertainty estimation is the possibility of using it during complementary rules, to validate or reject some of the agent's predictions, and to eventually accelerate some predictions, in relation to the scenarios mentioned above. We think that it is possible to trigger prediction when uncertainty is low and when the highest Q-value of classification is close to that of delay for several consecutive time steps. In addition, we believe that some hasty decisions made by the agent can be rejected when uncertainty is too high. In conclusion, we aim at proposing a solution to further analyze the predictions made by the agent on test data, and possibly validate or reject them.

6.3.2 Related work

Bayesian Neural Networks [74] are a first family of method that can estimate DNNs uncertainty. They are NNs which learn probability distributions over the weights, as illustrated in Fig. 6.11 extracted from [93]. Their main drawback is to be computationally expensive notably because they increase the number of model parameters. In the case of the thesis, we can not afford to increase calculation time for the method training.

In [31], the authors propose an uncertainty estimation method for all DNNs with a dropout layer applied before every weight layer in their architecture. On a sample data at test time, they perform W stochastic forward passes through the DNN while enabling dropout. Each forward pass then differs from another because of dropout use. In order to get a prediction on the test sample, they compute the W predictions mean, and they estimate uncertainty by the W predictions variance.

In [78], the authors adapt DQN algorithm with a bootstrap strategy in order to improve exploration of the agent and they argue that their method, Bootstrapped DQN, can also be used to provide uncertainty estimates. Instead of a Q-network with a single output layer, they train a Q-network with K independent heads, as illustrated in Fig. 6.12 extracted from [78]. Each head is trained on a bootstrapped sub-sample of the training set, i.e. on a number

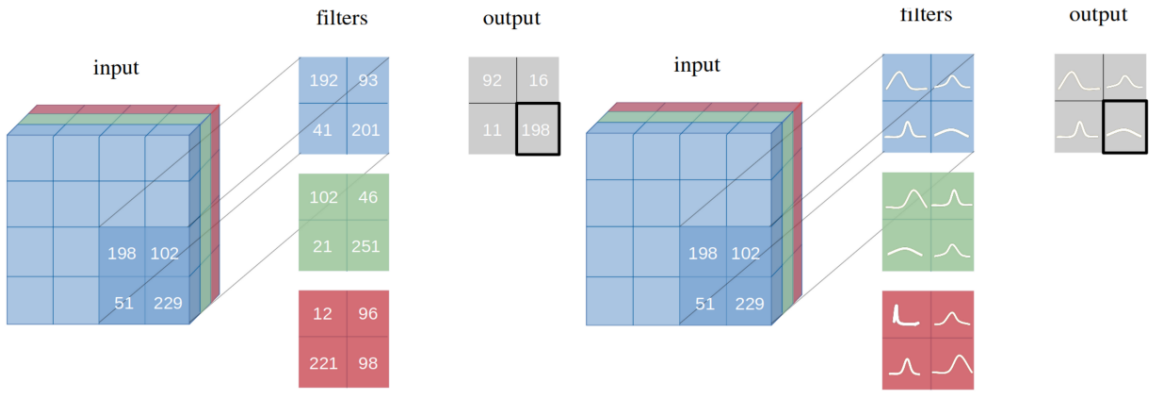

Figure 6.11: **Difference between traditional and Bayesian Neural Networks.** Input image with exemplary pixel values, filters, and corresponding output. Left: Traditional Neural Network with point estimates of the weights. Right: Bayesian Neural Network. The point estimates of the neural network's parameters are replaced by probability distributions [93].

of training samples that have been sampled with replacement. At test time, the distribution of predictions over each head can be used to measure uncertainty. In both [31] and [78] methods, uncertainty is measured through variance of predictions and we seek for a method with more interpretation elements.

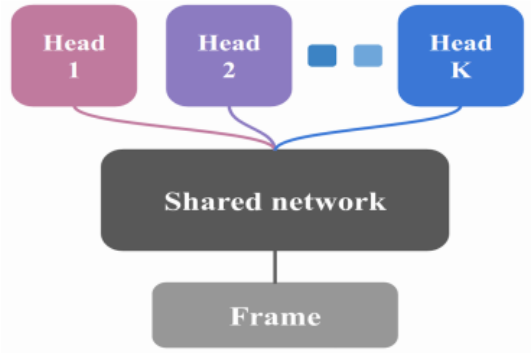

Figure 6.12: **Bootstrapped Neural Network with a shared network architecture. [78]**

In [12], the authors adapt DQN algorithm in order to learn the Q-values distributions for each action instead of their expectations. In Fig. 6.13 extracted from [12], the authors show on a frame of the Space Invaders game at test time that the agent manages to predict for each action a distribution of possible rewards. This approach allows to provide more interpretation on what the agent has learned during training, and on what it thinks it can earn for each action selection. Also, by looking at the rewards distribution for each action, it allows to detect if the distribution is wide and uniform (in that case, there are multiple possible outcomes for the action selection, and rewards are uncertain), or if the distribution forms a peak placed on a single value (in that case, the agent is certain of its reward prediction). In the case of

the thesis, it can be interesting to compare two actions with similar expected Q-values, and look at their predicted distributions in more detail, instead of their mean value. We leave this study for future work.

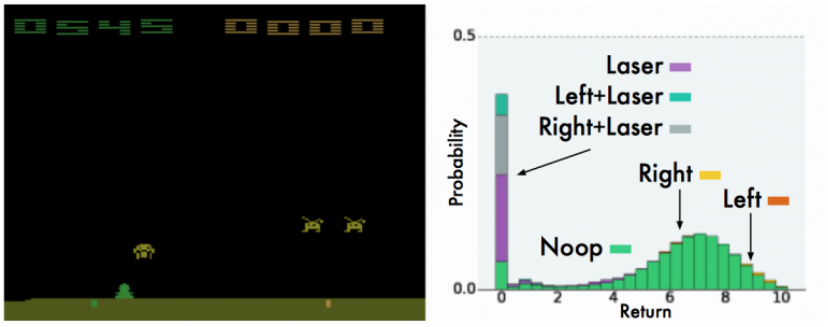

Figure 6.13: **Learned value distribution during an episode of Space Invaders.** Different actions are shaded different colours. [12]

In this work, we will rather choose to implement the method from [89] where the authors measure the uncertainty of a DNN prediction in the context of a classification problem. The method is detailed below.

6.3.3 Method presentation

Motivation In [89], the authors argue that DNNs trained for classification are usually built with a softmax activation in their output layer, as illustrated in Fig. 6.15. In Fig. 6.14 from [89], they illustrate the classification results of a digit 1 from the MNIST database when it is rotated, under two strategies. In the left figure, they illustrate how a softmax activation in the output layer of a DNN tends to return high probabilities estimates, even when the data is far from what was observed during training. They show that the softmax-based DNN misclassifies a rotated image of digit 1, with high probabilities estimates for digits 2 and 5. Moreover, other work on DNNs uncertainty [31] asserts that the probability scores calculated by a softmax activation are not a good metric to estimate the confidence of a DNN. From this observation, the authors seek an alternative to the usual softmax activation to derive probabilities estimates on labels for classification problems. In particular, they aim at solving classification problems with DNNs while evaluating their uncertainty.

General principle and main results Their strategy is the following. Instead of predicting a probability on the labels using a softmax activation, the authors predict a value of evidence on each label, this value being positive or null, and not bounded. From the values of evidence on each label, the authors propose to derive both probabilities scores on the labels, and an uncertainty score on the predictions. The mathematical formulas are given below.

In the right figure of Fig. 6.14, they show that their method allows to identify uncertainty in the DNN predictions when the digit is rotated, instead of high false probabilities estimates. Consequently, their method makes it possible to reject classification results, given uncertainty estimation.

In the experiments illustrated in this figure, the authors used the same DNN architectures to make a comparison between their approach and the traditional softmax-based approach.

They only modified the output layer of the DNN, as well as the loss function to be minimized, as illustrated in Fig. 6.15.

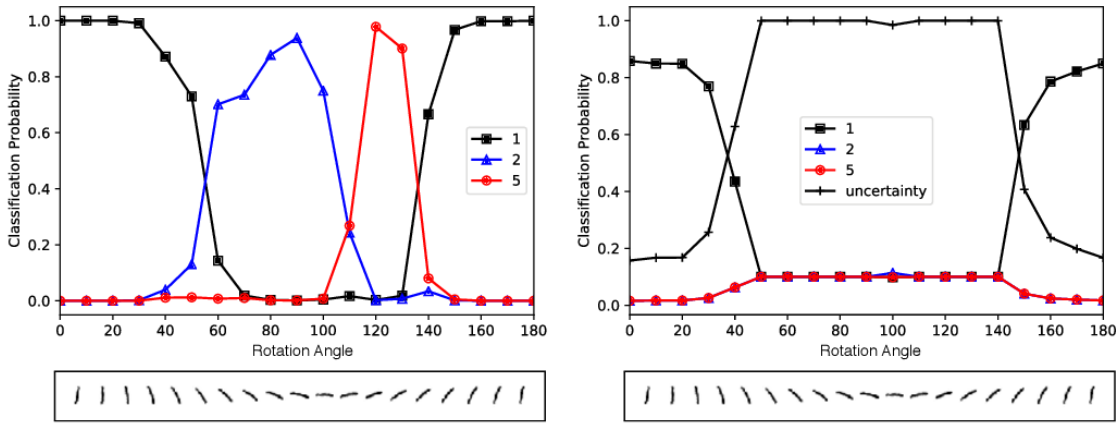

Figure 6.14: **Classification of a rotated digit 1 from MNIST dataset at different angles between 0 and 180 degrees.** The classification problem involves 10 labels in relation to handwritten digits from 0 to 9. Left: Classification probability is calculated using the softmax function. The digit 1 can be misclassified as digit 2 or digit 5 with high probabilities based on the degree of rotation. Right: Classification probability and uncertainty are calculated using the method from [89]. The digit 1 is either correctly classified or all predicted probabilities are low based on the degree of rotation. The method enables to predict a large uncertainty when the digit has a 90° rotation angle.

Mathematical background Standard DNNs predict labels probabilities using a softmax activation in the output layer. In [89], the authors propose to replace the softmax activation by an activation function ensuring a non-negative output. Specifically, they associate the predictions in the output layer of the DNN to a vector $\mathbf{e}$ of evidence on each label:

$$\mathbf{e} = (e_1, \cdots, e_K) \ , \ \forall k \in [1, K] \ , \ e_k \geq 0 \tag{6.6}$$

with K the number of labels in the classification problem, $\mathcal{L} = \{l_1, \cdots, l_K\}$, and e_k the evidence on label l_k. Evidence is positive or null.

With $\mathbf{e}^n = (e_1^n, \cdots, e_K^n)$ the vector of evidence predicted by the DNN on a sequence $\mathbf{X}^n$, the authors define the uncertainty u^n of the predictions on $\mathbf{X}^n$ such that:

$$u^n = \frac{K}{\sum_{k=1}^{K} e_k^n + 1} \tag{6.7}$$

Uncertainty u is bounded between 0 and 1. The higher the evidences $e_k, \forall k \in [1, K]$, the more uncertainty u tends to 0. Conversely, the lower the evidences $e_k, \forall k \in [1, K]$, the more uncertainty u tends to 1.

The authors define the expected probability $\hat{p}_k^n$ that a sample data $\mathbf{X}^n$ is associated to the k-th label such that:

$$\hat{p}_k^n = \frac{e_k^n + 1}{\sum_{j=1}^{K} e_j^n + 1} \tag{6.8}$$

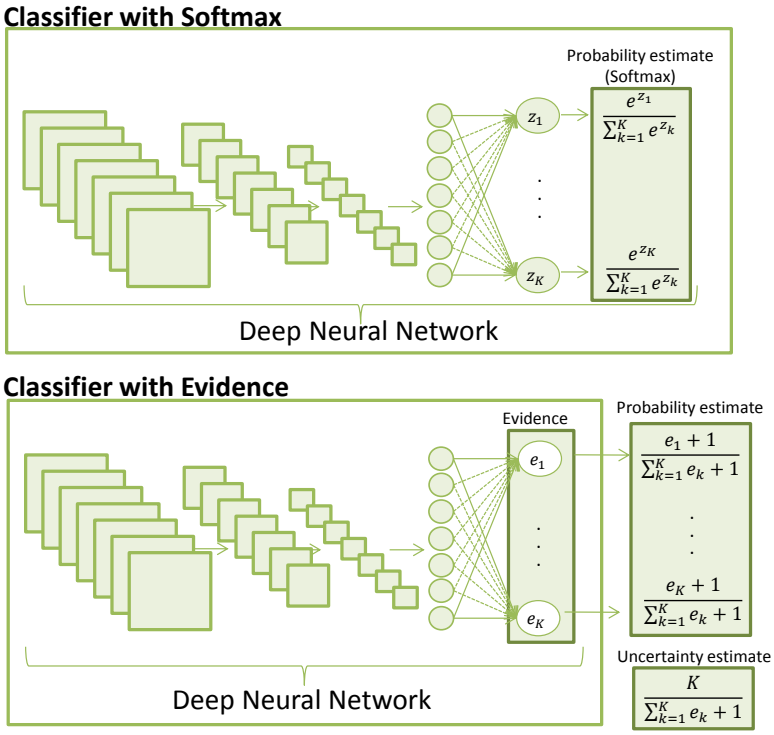

Figure 6.15: (Top) **Illustration of a DNN classifier with softmax activation in the output layer.** (Bottom) **Illustration of a DNN classifier with prediction of evidence in the output layer.**

With l^n the true label of $\mathbf{X}^n$, such that $(\mathbf{X}^n, l^n) \in \mathcal{D}$, they define the loss L^n_{ev} of the DNN with parameters Θ such that:

$$L^n_{ev}(\Theta) = \sum_{k=1}^{K} \mathbf{1}_{l^n = l_k} \big(\psi(\sum_{j=1}^{K} e^n_j) - \psi(e^n_k)\big) \tag{6.9}$$

with e^n_j the evidence predicted by the DNN on data $\mathbf{X}^n$ for label l_j and ψ the digamma function. Given the loss definition, the loss increases on a sample $\mathbf{X}^n$ with true label l^n if evidence is added on all labels except l^n, and/or evidence is removed from label l^n only. Conversely, the loss decreases if evidence is removed from all labels except l^n, and/or evidence is added on label l^n only.

6.3.4 Application to EC

In this work, our objective is to compute the vector of evidence $\mathbf{e}$ (Eq. 6.6) for each label $l \in \mathcal{L}$ (Eq. 2.5) of the classification problem, in addition to the Q-values for each action $a \in \mathcal{A}$ (Eq. 3.26) of the EC-POMDP. From the vector of evidence $\mathbf{e}$, we will be able to derive a probability score of each label $l \in \mathcal{L}$, and an uncertainty estimate of the DNN predictions of evidence. Moreover, we continue to obtain a policy for EC from the predicted Q-values, as done so far.

Multi-branch architecture To this end, we propose to use a multi-branch architecture of the DNN Q_Θ, as illustrated in Fig. 6.16, where:

- In the upper branch of the DNN, we compute the vector of evidence following the method from [89]. The upper output layer has then as K output neurons, one for each label $l \in \mathcal{L}$ of the classification problem. It is fully-connected to its previous layer and has a positive activation function (because evidence is positive or null).

- In the lower branch of the DNN, we compute the Q-values using the same loss function than DDQN algorithm ([100], Eq. 4.4). The lower branch has $K+1$ output neurons, one for each classification action $a \in \mathcal{A}_c$ and one for delay action a_d (see Eq. 3.26). It is fully-connected to its previous layer and has a linear activation function (because Q-values take values in $\mathbb{R}$).

The choice of where and how to set up the DNN branch for evidence calculation is up to the user, and becomes the source of additional hyper-parameters. In our case, we chose to set the DNN division into two sub-branchs at the end of the architecture, before the output layers of Q-values and evidence, as shown in Fig. 6.16. In this way, evidence and Q-value calculation is made upon a majority of shared lower-level features.

Moreover, we point out that evidence is positive and unbounded while Q-values can be small (in some cases < 1 in absolute value depending on rewards settings). Each sub-branch of the DNN therefore has to predict output values that differ in orders of magnitude. Consequently, in order to allow each sub-branch of the DNN to predict output values close to their definition space, we have injected a fully-connected layer before each output layer, following the separation of the main architecture into the two sub-branches. In Appendix E, we give an example of a multi-branch architecture used during experiments.

Notation. In the following, we continue to refer to the DNN of the agent as Q_Θ, even if it now includes predictions for both Q-values and evidence.

Training To train the DNN Q_Θ and find optimal parameters Θ^*, the RL method proposed in previous chapters is maintained and remains valid. We use the same RL algorithm as in Chap. 4 and we can keep applying the strategies of memory management and episode initialization introduced in Chap. 5. Also, we can train the DNN with both online learning (Algo. 4) and batch learning (Algo 5), depending on the what is most suitable for the application. The only difference to oper is in the loss calculation, when parameters Θ are updated (see step 7 in Fig. 5.2, and step 2 in Fig. 5.3). Indeed, in Algo. 4 and Algo 5, a single loss function was computed, that of the DDQN algorithm (see Eq. 4.4). We now seek to simultaneously minimize two loss functions: the one for the Q-values (Eq. 4.4) and the one for evidence (Eq. 6.9).

In practice, at each update of the DNN parameters Θ, we apply the following steps:

- We sample a mini-batch of past interactions $\{< o, a, r, o' >\}$ from the replay memory $\mathcal{M}$, with the prioritized sampling strategy introduced in Algo. 2.

- For each past interactions $< o, a, r, o' >$ from the mini-batch, with o being an observation generated by a training sequence $\mathbf{X}^n$ with true label l^n (such that $(\mathbf{X}^n, l^n) \in \mathcal{D}$), we compute two losses.

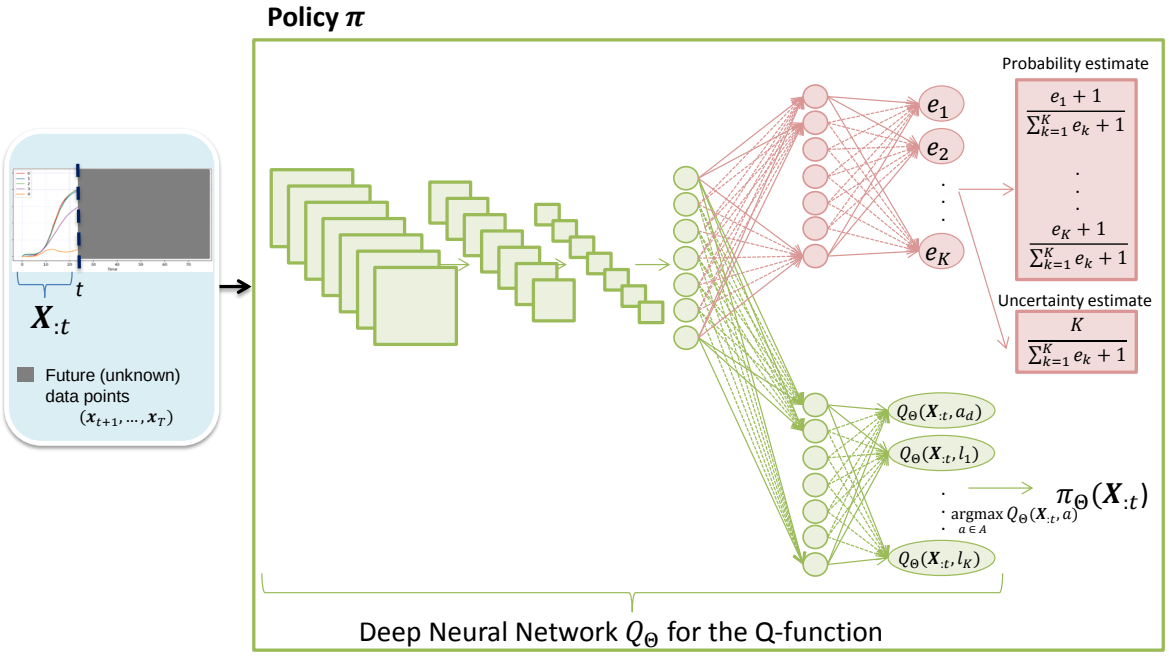

Figure 6.16: **DNN Q_Θ with a multi-branch architecture and parameters Θ for prediction of evidence and Q-values.** It is defined over the set of observations $\mathcal{O}$ (Eq. 3.25). The lower branch of the DNN predicts the Q-values on an observation $o = \mathbf{X}_{:t}$ and for all actions $a \in \mathcal{A}$. Its output layer has $K+1$ output neurons, is fully-connected to its previous layer and has a linear activation. The upper branch of the DNN predicts the evidence on an observation $o = \mathbf{X}_{:t}$ and for all labels $l \in \mathcal{L}$. Its output layer has K output neurons, is fully-connected to its previous layer and has a positive activation.

We first compute the DDQN loss (Eq. 4.4):

$$L^n_{DDQN}(\Theta) = (r + \gamma Q_{\Theta^-}(o', \arg\max_{a'} Q_\Theta(o', a')) - Q_\Theta(o, a))^2$$

where γ is the discount factor, Q_Θ the current Q-network and Q_{Θ^-} the target Q-network (see Sec. 4.2).

We then compute the evidence loss (Eq. 6.9):

$$L^n_{ev}(\Theta) = \sum_{k=1}^{K} \mathbf{1}_{l^n=l_k} \left(\psi(\sum_{j=1}^{K} e^n_j) - \psi(e^n_k)\right)$$

with e^n_j the evidence predicted by the DNN on data $\mathbf{X}^n$ for label l_j and ψ the digamma function.

Finally, we compute a weighted sum of the two losses:

$$L^n_{tot}(\Theta) = L^n_{DDQN}(\Theta) + \mu L^n_{ev}(\Theta) \tag{6.10}$$

with μ the weight associated to the evidence loss in comparison to the DDQN loss. μ is an additional hyper-parameter of the method that need to be fixed before the agent's training.

- We update the DNN parameters Θ by back-propagation of the gradient calculated on the total loss $L_{tot}(\Theta)$ on the mini-batch of past interactions.

6.3.5 Preliminary experimental evaluation

Dataset We conduct a preliminary evaluation of the method on a simplified version of the industrial dataset, with 3 labels instead of 7. The DNN Q_Θ of the agent has been trained on a dataset involving a set of MTS $\mathbf{X} \in \mathbb{R}^{5\times 77}$ associated to ordinal labels in $\tilde{\mathcal{L}} = \{l_2, l_4, l_7\}$, such that $\tilde{\mathcal{L}} \subset \mathcal{L}$ and $l_2 < l_4 < l_7$. Labels distribution is given in Fig. 6.17. In Fig. 6.18, the training set is represented with a two-dimensional t-SNE embedding of the (complete) temporal sequences using algorithm from [64]. We observe less overlapping clusters of points from different labels than in Fig. 6.4 which illustrates the dataset complexity has been reduced.

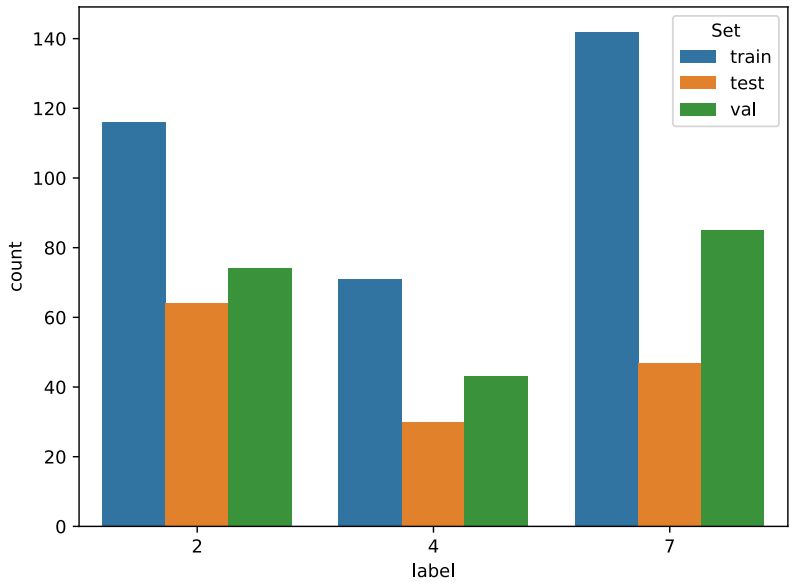

Figure 6.17: **Labels distribution in the simplified industrial sets of training, validation and testing.** The simplified set of labels is $\tilde{\mathcal{L}} = \{l_2, l_4, l_7\}$

EC-POMDP model As in previous analysis, we use a cost-sensitive reward function (Eq. 3.32) to address the natural costs of the industrial classification problem. We point out that in these experiments, we did not necessarily restricted rewards with values between -1 and 1, so that Q-values and evidence predicted in the DNN output layers can have similar order of magnitude. The following figures (entitled "*Rewards*" in Fig. 6.19, Fig. 6.20, Fig. 6.21) illustrate how misclassification is penalized depending on the true and predicted labels.

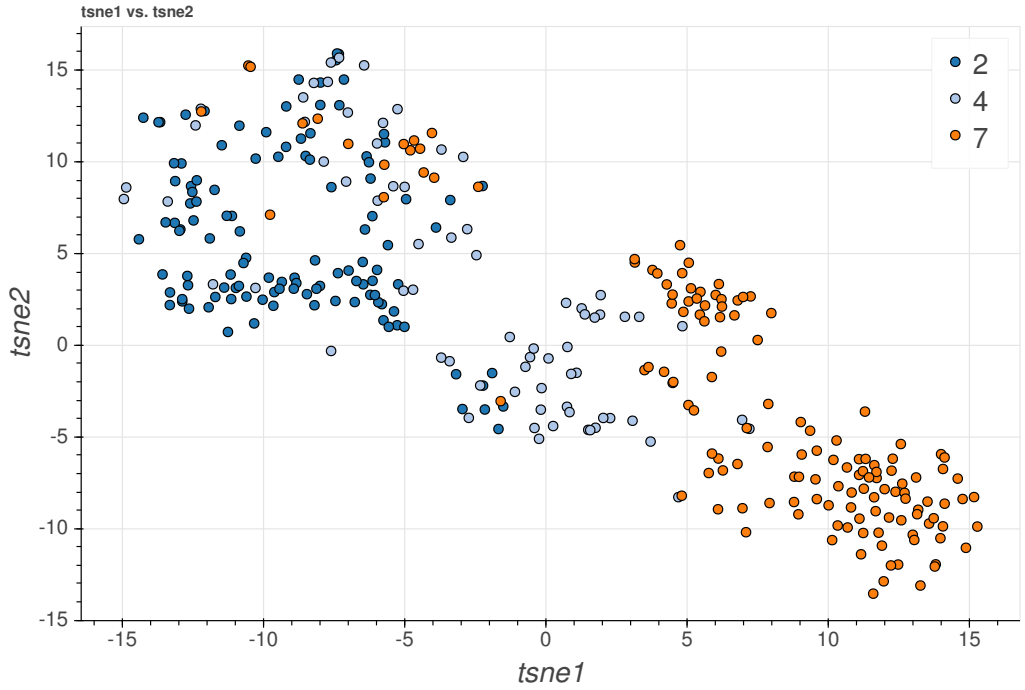

Figure 6.18: **Two-dimensional t-SNE embedding of the simplified industrial training set.** Each dot represent a complete temporal sequence $X_{:T}$ from the simplified industrial training set associated to an ordinal label in $\tilde{\mathcal{L}} = \{l_2, l_4, l_7\}$. Dots are colored according to labels.

Results In Fig. 6.19, the test data is associated with a reference label l_4. It is acquired until time step $t_{pred} = 28$, time at which the agent correctly classifies by predicting label l_4 (we can see that the Q-value of label l_4 outreaches that of delay at this time, as shown by the green and red curves in the "*Q-values*" graph). Also, we can see that the time of prediction caused by the Q-values estimations from the lower branch of the DNN is consistent with the prediction of evidence from its upper branch. Indeed, as shown by the green curve of the "*Labels evidence*" graph, from time step 20, the DNN starts predicting evidence on label l_4. Consequently, the expected probability for this label increases (see the green curve in the "*Labels probability*" graph) and uncertainty starts decreasing (see the orange curve in the "*Labels probability*" graph). At the time of prediction, $t_{pred} = 28$, the expected probability for the predicted label is almost up to 1, while uncertainty is almost down to 0. This is an example of prediction for which the evidence predicted by the DNN is consistent with the choice of action made by the agent.

In Fig. 6.20, the test data is associated with a reference label l_2. It is acquired until time step $t_{pred} = 12$, time at which the agent misclassifies by predicting label l_7 (as shown by the purple curve in the "*Q-values*" graph). From the computation of evidence for each label $l \in \tilde{\mathcal{L}}$, represented in the "*Labels evidence*" graph, we can see that low evidence was predicted for the label l_7 at the time of prediction t_{pred}. Consequently, the uncertainty at that time was still high, as illustrated in the "*Labels probability*" graph by the orange curve. Moreover, after time step t_{pred}, if the sequence acquisition had continued, the Q-value estimation for label l_7

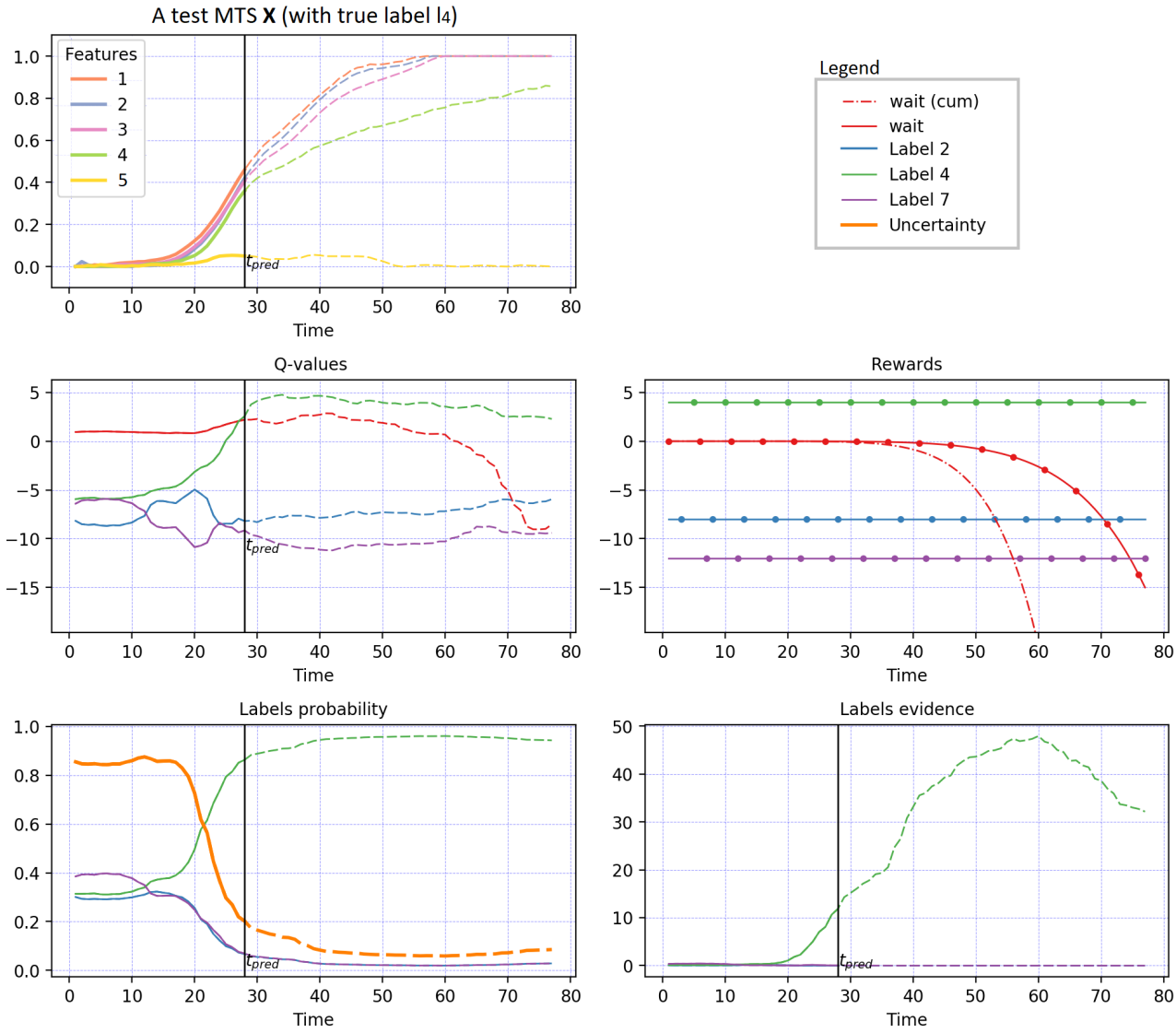

Figure 6.19: (Top left) **A test MTS $\mathbf{X}_{:28}$ with true label l_4.**
(Middle left) **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.**
(Middle right) **Reward definition for each action $a \in \mathcal{A}$.**
(Bottom left) **Probabilities estimates for each label $l \in \mathcal{L}$ and uncertainty estimation**, computed from the evidence predicted by the DNN upper branch.
(Bottom left) **Evidence for each label $l \in \mathcal{L}$.** It is predicted by the DNN upper branch.
In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

would have decreased, and that of label l_2 would have increased. Also, evidence on label l_2 would have increased, leading to an increase in expected probability for this label up to almost 1, and uncertainty would have decreased down to almost 0.

In this example, misclassification is due to a too early prediction. It could have been

avoided if the agent had waited for a few more time steps. Through this uncertainty evaluation method, predictions of evidence (and underlying uncertainty estimation) from the upper branch of the DNN could have been used during the application of the policy on the test data to raise an alarm on high uncertainty or to postpone the prediction.

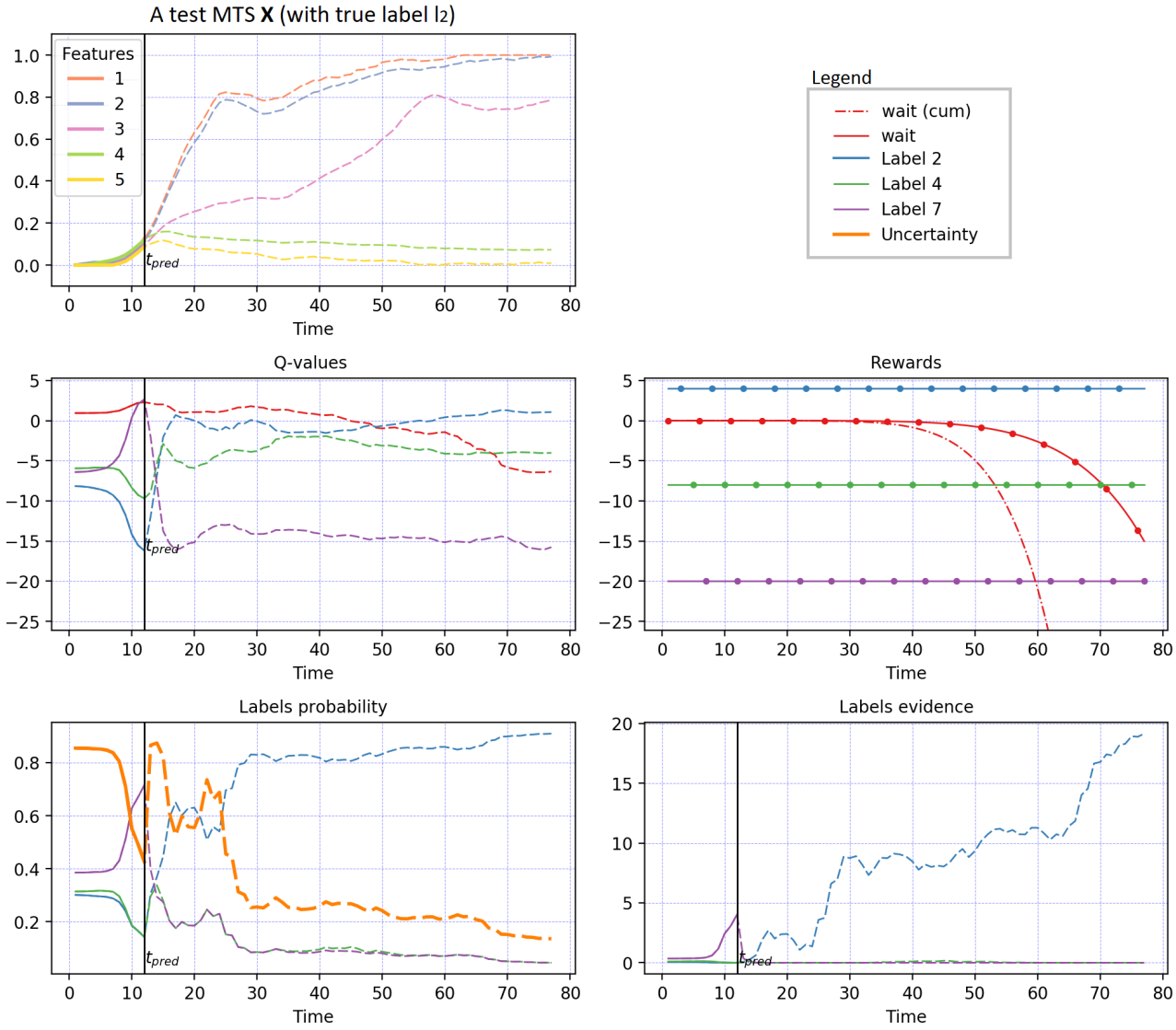

Figure 6.20: (Top left) **A test MTS $\mathbf{X}_{:12}$ with true label l_2.**
(Middle left) **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.**
(Middle right) **Reward definition for each action $a \in \mathcal{A}$.**
(Bottom left) **Probabilities estimates for each label $l \in \mathcal{L}$ and uncertainty estimation**, computed from the evidence predicted by the DNN upper branch.
(Bottom right) **Evidence for each label $l \in \mathcal{L}$.** It is predicted by the DNN upper branch.
In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

In Fig. 6.21, the test data is associated with a reference label l_2. It is acquired until time step $t_{pred} = 48$, time at which the agent correctly classifies by predicting label l_2 (as shown by the blue curve in the "*Q-values*" graph). As illustrated in the "*Q-values*" graph, the Q-value for the correct classification action becomes close to that of delay from time 30, and does not exceed it until time 48. This is an example close to the one from Fig. 6.9, where the greatest Q-value of classification remains slightly lower than that of delay, for many successive time steps.

As illustrated in the bottom figures, evidence slightly increases on label l_2 from time 30, and uncertainty decreases. Then, these two values remain on a plateau until time step 50, with the expected probability of label l_2 being larger than uncertainty. From time step 50, the expected probability increases up to 1, and uncertainty decreases down to 0.

In this particular case, for the same level of uncertainty in the predictions, the time of prediction could have been accelerated from at least 10 time steps.

6.3.6 Conclusion and perspectives

In this section, we showed how to apply the method from [89] for uncertainty estimation in order to interpret or validate the predictions made by the agent on test data. The existing work has been adapted to the EC problem when a RL agent is trained to predict Q-values with a DNN. Specifically, we showed how to compute evidence on the labels of the classification problem, and how to derive both probability estimates for each label and uncertainty estimates.

The drawbacks of the method are related to a larger number of hyper-parameters to tune. They are caused by the multi-branch architecture of the DNN, and the weight associated to the two losses for parameter updates.

Following the first experimental results, we believe that uncertainty estimation can allow to reach the initial objectives of this additional work, namely to improve the time of prediction of the agent and to make decision-making more robust. First, we illustrated that it can be used as a monitoring step to prevent label predictions when uncertainty remains *high*. To that end, the user must set an additional hyper-parameter for uncertainty threshold. Second, we illustrated how uncertainty can be used to accelerate the prediction, in some specific cases. Typically, when the greatest Q-value of classification action for a label l is *very close* to that of delay, for *consecutive steps*, and when the probability estimate on that same label l is *high* with *low* uncertainty, the user can put a trigger rule and predict the label in advance. Nevertheless, the setting of this trigger rule requires the user to set additional hyper-parameters (related to the number of consecutive steps, the high probability threshold, the low uncertainty threshold and the distance between the Q-value). These two possible applications of the method are left for future work.

The perspectives in relation to this preliminary study are the following. In experiments, we applied the method on a simplified version of the industrial dataset which has fewer labels to predict. In future work, the method should be validated on the complete dataset with more labels. Finally, we point out that the loss function related to evidence prediction does not take into account the fact that labels are ordinal (see Eq. 6.9). When evidence is added to a wrong label, the loss equally penalizes this addition, whether the label is near or far from the true label. In our case, the addition of evidence on a label close to the true label is less serious than adding evidence on a distant label. The loss can therefore be optimized for this problem in a future work.

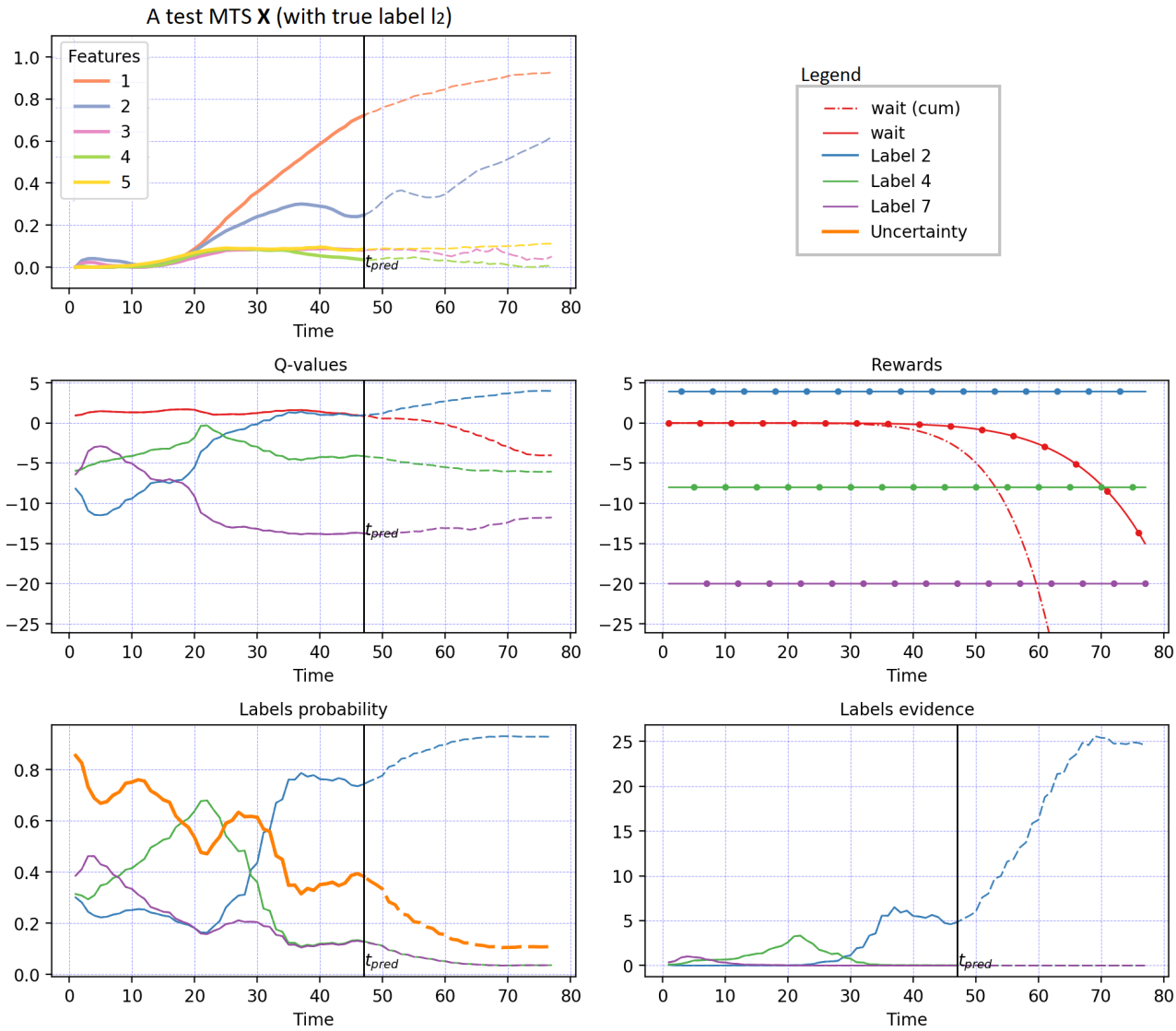

Figure 6.21: (Top left) **A test MTS $\mathbf{X}_{:48}$ with true label l_2.**
(Middle left) **Q-values predicted by the agent for each action $a \in \mathcal{A}$ over the sequence acquisition.**
(Middle right) **Reward definition for each action $a \in \mathcal{A}$.**
(Bottom left) **Probabilities estimates for each label $l \in \mathcal{L}$ and uncertainty estimation**, computed from the evidence predicted by the DNN upper branch.
(Bottom right) **Evidence for each label $l \in \mathcal{L}$.** It is predicted by the DNN upper branch.
In full line is what has been observed until the prediction of a label by the agent. In dashed line is what would have been observed if sequence acquisition continued.

Chapter 7

Conclusion and perspectives

The objective of this doctoral work was to propose a method for early classification (EC) of temporal sequences. Specifically, the method had to deal with an industrial application for which data are multivariate time series (MTS) and the classification problem is multi-class, ordinal, class-imbalanced and cost-sensitive.

7.1 Synthesis of the doctoral research

In Chap. 2, we defined the EC problem as the task of assigning a *label* to some data that is *sequentially* collected, with new data points arriving over time. The data can be incomplete depending on the state of the acquisition process and therefore it can have variable length. Specifically, we defined an early classifier as a model which has to decide during the sequence acquisition when to stop the acquisition in order to predict a label, with the objective to use as few data points as possible on each sequence. It has to *adapt* its prediction time on each sequence *individually*, depending on its complexity. It has to require more data points on sequences that can hardly be classified, before making its prediction. At the opposite, on easy sequences with early discriminant patterns, it has to quickly stop the acquisition process and render a prediction result.

Following the formulation of EC as an optimization problem, we outlined the EC *trade-off* which is to ensure optimal classification while minimizing the prediction time. We argued that EC is of major interest for applications concerned with the *costs* induced by the data points acquisition. It is also a major research topic for applications which seek for *rapid* label prediction in order to take early actions, such as predictive maintenance, medical diagnosis, etc.

Regarding the data to be processed, we argued that temporal sequences are specific data for which it is necessary to use models that can take into account the *temporal* relations in the data points. We pointed out that, in the specific case of MTS, the models must also take into account the relationships between the different *features* of the sequence. Due to the dynamic temporal dimension of the data, the models have to be able to analyze incomplete data. As a consequence and as part of a method that can be applied to data on which we have *no expertise*, we identified the interest of Deep Neural Networks (DNNs), among which Convolutional Neural Networks (CNNs) are a possible choice of model.

Throughout the chapter, we showed that there are no method in the literature that can take into account all the specificities related to the data and the classification problem.

In Chap. 3, we developed a method around the formalization of the EC problem as a sequential decision-making problem. We considered that the data are received online, with new data points arriving over time, and that the problem is to *decide* between *waiting* in order to gather more data points, or *predicting* a label on the incomplete sequence received so far. Following this formalization, we sought for a solution that can provide an *end-to-end* result, from the incomplete data to the decision of delay or label prediction.

First, we showed that solving the sequential decision-making problem with Supervised Learning (SL) brought along a new challenge: to build a training labelled dataset with supervision on both classification and delay actions at all time steps in the sequences. We argued that the methods from the literature were not directly applicable and therefore the resolution of the problem with SL was a major research topic.

Then, we showed that EC can be described as a Markov Process with actions. We defined rewards associated to EC so that the decision-making problem can be described by a Markov Decision Process (MDP) and then solved within a Reinforcement Learning (RL) framework. We proposed several strategies to compromise between classification earliness and accuracy in the reward function definition, through strategies of reward shaping and reward discounting. The solution allows the user to set the *relative importance of time* compared to classification quality for his application. Also, several definitions of rewards were proposed depending on the nature of the classification problem (cost-sensitive versus cost-insensitive) and the solution allows to involve the misclassification costs defined by the applications. We showed that the MDP for EC, noted EC-POMDP, was actually *partially observable* because, during the online acquisition of sequences, we do not have access to future data points nor the label to predict, while these information will be used by the environment during the training of the agent.

In this doctoral work, we specifically addressed the problem of EC on time series, both multivariate and univariate, depending on the experiments. However, the mathematical framework has been proposed for the general problem of EC on temporal sequences and it can therefore be applied to different types of sequences.

In Chap. 4, we proposed to solve the EC-POMDP with a value-based approach. Specifically, we aimed at approximating its action value function Q with a DNN leading to the selection of a Deep Reinforcement Learning (DRL) algorithm. We proposed a pipeline to train an agent with DDQN algorithm [100] and select an optimal policy for EC.

In experiments, we showed on UCR time series benchmark [21] that the EC problem can be solved with an end-to-end RL agent. The agent achieved EC objectives: to compromise between classification quality and its earliness. We showed that the agent is able to continually *adapt* its behavior without human intervention. It *simultaneously* learns classification features and decision-making rules on prediction times. Also, we experimentally showed that the agent achieved similar or better results in accuracy and prediction time compared to state-of-the-art methods from the literature.

We identified limits to apply DDQN algorithm in its original form on the EC-POMDP. We showed that the replay memory of the agent can become *imbalanced* during its training, weakening its overall learning. In the remainder of the study, we therefore aimed at optimizing DDQN algorithm for the EC problem and investigating if the resolution of poorly balanced memory improves the training and performance of the agent.

In Chap. 5, in order to optimize DDQN algorithm, we proposed different strategies for

robust episode and memory management: prioritized sampling, prioritized storing and random episode initialization. In experiments, we showed that the different proposed strategies statistically improved the performance of the agent in terms of accuracy and speed.

Finally, we compared early classifiers trained with RL to static DNNs, and we showed that the capacities of early classifiers to adapt their time of prediction individually on each sample improved the general trade-off of accuracy versus speed.

In Chap. 6, we then sought to provide *interpretations* about the decisions made by the agent. We applied and adapted the Class Activation Map method [116] for EC. It allowed to identify the data points in a sequence that contribute to the Q-value predictions, and to draw some interpretations on the choice of action made by the agent. The method can be used when applying an optimal policy on test data, in order to provide a rationale for the decisions of the agent. Or it can be used as a second stage of optimal policy selection, to compare several successful policies, and select the one for which the detected patterns in sequences are most consistent with the application, based on our data expertise. In this chapter, we also proposed a method to estimate the DNN *uncertainty*, based on prediction of evidence [89] on each label of the classification problem.

7.2 Main contributions and results

The main contribution of this doctoral work was to formalize the problem of EC of temporal sequences as a sequential decision-making problem where the EC model has to decide between classifying an incomplete sequence or delaying the prediction to collect additional data points. Specifically, we described the EC problem by a POMDP, the EC-POMDP, by defining a suitable set of states, actions and observations and by setting a reward function which encodes the EC trade-off. The second major contribution of this work was to adapt and optimize an existing DRL algorithm to the EC-POMDP specificities, with episode and memory management strategies. We initiated these strategies to solve the imbalanced memory issue caused by the original algorithm, and we showed that they improved the overall training of the agent. Finally, other secondary contributions were to propose solutions for policy interpretations.

To our knowledge and contrary to methods from the literature [7, 8, 20, 38, 39, 41, 70, 71, 79, 107, 110], the proposed method is the first that can simultaneously address both problems of classification and prediction time with an end-to-end model, be generic for various types of sequential data, directly minimize a criteria based on time, and address all specificities of the classification problem (multi-class, class imbalanced, ordinal and cost-sensitive). Indeed, following our work, we were able to show that DRL was a possible solution to the EC problem with many advantages:

- The method simultaneously solves the two sub-problems of EC: classifying incomplete sequences and predicting the optimal earliest time to perform classification. Indeed, the method offers a *single* model which is able to simultaneously learn optimal patterns in the sequences for classification and optimal strategic decisions for the time of prediction. There is no need to optimize each sub-problem separately. Instead, the method involves a single optimization phase. Therefore it allows for end-to-end learning of classification and decision rules, with *simultaneous optimization* of the two sub-problems.

- The method offers an *end-to-end* model for EC which does not require expertise on the data to be analyzed, and the method does not involve a preliminary step of *feature extraction*. Instead, the DNN is given input *raw* data, without dimension reduction nor pre-processing. The DNN is thus in charge of summarizing information in the data points, extracting relevant features and learning optimal decisions. As a consequence, the method does not necessitate to perform prior exploratory analysis, and it can be applied on *external* data on which we have *no expertise*.

- The method does not make assumptions about the input data. It is *generic* and can be applied to various types of sequential data, such as time series, symbolic sequences, images sequences, texts, etc. Moreover, in the specific case of time series, the sequences can be highly non-stationary. The user simply has to choose a DNN architecture that can be adjusted to its input data, with suitable layers, filters and activation.

- In the proposed approach, time directly appears as a *criteria* to minimize and the user can set its *relative importance* in comparison to classification quality. The method can therefore offer an ensemble of EC models, more or less rapid, and more or less accurate, according to the will of the application to make a compromise. In addition, the cost of time, if known, can be directly formalized in the definition of the EC-POMDP.

- The method allows to solve EC problems which are cost-sensitive, and also example-dependent cost-sensitive. Indeed, misclassification costs can be taken into account directly into the definition of the EC-POMDP, through the reward function. Consequently, the proposed approach can solve a large variety of EC problems, such as medical diagnosis which is known to be cost-sensitive due to the gravity of non-detecting a severe disease in comparison to raising a false alarm.

- The method can handle both binary and multi-class classification problems.

- The method can handle classification problems that are ordinal, by setting a reward function which penalizes misclassification even more when the predicted label is far from the true label. More generally, the solution is to define misclassification costs, and use these costs in the reward function definition, as suggested above for cost-sensitive classification problems.

- The method is robust to classification problems with an imbalanced training dataset, thanks to the memory management strategies proposed in the thesis. The user can indeed allocate a fraction of the replay memory of the agent to samples with under-represented labels, so that these samples are kept longer in memory and not over-written by those with over-represented labels.

- During training, the user benefits from a range of possible EC solutions with different trade-offs between classification quality and speed. Consequently, the user can choose the EC solution that best satisfies his application domain. At any time during training, the user can judge that he is satisfied with one of the EC solutions obtained so far and stop the training of the agent.

Future studies on the EC problem can continue this work and address one of the following perspectives.

7.3 Limits & Perspectives

7.3.1 In relation to the policy and optimization

7.3.1.1 Approximation of the policy by a Deep Neural Network

In this doctoral work, we aimed at proposing a generic method which can be applied to external data for which we do not have expertise. To this end, the method developed during the thesis makes use of DNNs in order to allow autonomous learning of features in the data.

We showed on UCR datasets that the method can achieve similar or better performances than state-of-the-art methods, even on small training sets (≤ 100 training samples). Nevertheless, because of the easily large number of parameters generated by DNN models, we faced some over-fitting issues on these datasets, and tuning the method was made harder. Additionally, most real-life applications involve finite (often small) training datasets due to limited budget constraints. We therefore argue that the use of Deep Learning in this method has some limits. It necessitates a training set with a sufficient number of training samples, this number being dependent on the quality of the samples, their diversity, their representativeness of the whole population, etc.

The perspectives related to this limit are multiple. First, the most obvious but not always possible is to employ data augmentation strategies. In the case of sequences, it can for example relate to scaling, shifting, adding noise to the data or applying generative models if the application allows it. Second, the DNN can also use more dropout or a lighter architecture (less layer, less convolution filters, etc.), helping to reduce the number of parameters in the model. Finally, for applications with few training samples but available expertise on feature selection, the user can replace the use of DNNs by more simple function approximators that are differentiable, such as linear combination of features. These more simple functions (with fewer parameters than DNNs) can play the role of the policy, and can analyze input data with pre-computed features, instead of raw data. They still aim at estimating the Q-values for all actions, and the policy remains to act greedily over these predicted action values. The user can then keep solving the EC-POMDP with the same method as in the thesis.

7.3.1.2 Hyper-parameters tuning

By the use of DNNs and the description of the problem as an EC-POMDP, the method proposed in this thesis requires a large number of hyper-parameters to be tune. These hyper-parameters are related to the architecture of the DNN (type of layer, number of layers, number of neurons/filters, size of filters, type of activation function, etc.), the gradient descent algorithm (learning rate, batch size, etc.), the replay memory of the agent (size), the exploration rate of the agent (initial value, final value, decay coefficient), the EC-POMDP (discount factor, rewards, trade-off parameter, etc.), DDQN algorithm (update frequency for the target network) and more. In this doctoral work, we employed a tuning strategy based on grid-search where several combination of hyper-parameters were tested independently and the best combination was kept as output of the method optimization.

This approach was computationally expensive and, we therefore proposed, at the end of the thesis, to use a more resource-efficient method. We initiated the use of Bayesian methods [55, 94] which consist in iteratively optimizing the method by testing new combinations of hyper-parameters. The principle is to select the next set of hyper-parameters to be tested from previous evaluations of the method. The results obtained with the previous sets of

hyper-parameters are used to estimate which combination of hyper-parameters will optimize the method, according to an objective function. By doing so, Bayesian methods seek to reduce the number of training and evaluation of the method. The difficulty lies in the definition of an objective function which allows to evaluate one training of the agent in a single metric score. The evaluation has to reflect the multi-objectives of the EC problem, by measuring both performance of accuracy and time of prediction. Moreover, another question in relation to the definition of an objective function is: should the objective function represents the *best* or *average* performance of EC achieved by the agent during its training?

7.3.1.3 Policies interpretation

In Chap. 6, we showed that the complexity of the industrial dataset was due to strong similarities between data with different labels (see Fig. 6.4). We therefore propose an additional interpretation tool for future work with regard to this specificity. In order to justify if the agent misclassifies (or hesitates) between two labels on a test data because of sub-optimal training, or because of the dataset complexity, we propose the following solution. We propose to provide a low dimension representation (e.g. t-SNE) of all training prefixes and of the test data, computed from their features representation in the Global Average Pooling layer of the DNN. In this way, by looking at the neighbors of a test data in the t-SNE representation, we will be able to visualize if the test data is close to training prefixes with discording labels, according to the feature representation learned by the DNN. This is a way to justify that the agent hesitates between several labels if, during training, it learned on prefixes with similar patterns (according to its representation) but different labels. In other words, it allows to see which training prefixes the test data is closest to, according to features learned by the DNN.

7.3.1.4 Uncertainty estimation

In Chap. 6, we proposed a method to evaluate the DNN uncertainty on test data, based on evidence prediction for all labels of the classification problem. As explained in Sec. 6.3.6, the loss function related to evidence prediction does not take into account the fact that labels are ordinal (see Eq. 6.9). It equally penalizes the addition of evidence on a wrong label, independently of its distance to the true label. A perspective is to modify the loss function for ordinal classification problems, and address the distance between true and predicted labels. When evidence is added to a wrong label, the loss has to further penalize this addition when the label is far from the true label. We think that this modification can facilitate the training and optimization of the method.

Another perspective is to predict evidence on subsets of labels rather than (or in addition to) singletons, in order to be able to quantify the uncertainty of the model between several consecutive labels. We think that this solution can allow the DNN to express its hesitation between several labels while predicting large evidence on a subset if it is sure that the true label belongs to this subset in particular.

7.3.2 In relation to the EC-POMDP definition

7.3.2.1 Rewards definition

Dynamic rewards One of the most important hyper-parameter to tune is the reward function of the EC-POMDP. We observed that too gentle penalties for delay encourages the agent

to classify at the end of sequences while too severe penalties lead to immediate predictions. In experiments, we tuned rewards through a grid search by launching independent trainings. Rather than training several independent agents on different reward definitions, a perspective is to modify rewards during training, in other words to define *dynamic* rewards. Starting with gentle penalties for delay, so that the agent is encouraged to discover the complete sequences and learn classification rules first, the user can then progressively increase the penalties for delay during training, so that the agent progressively fastens its predictions on shorter prefixes of sequences. We wonder if such a strategy can:

1. facilitate the training of the agent by progressively making it more difficult to reach a trade-off. Indeed, the solution is equivalent to gradually complicate the task of the agent, since during training it will receive less and less rewards (or more and more punishment depending on the strategy) for a prediction made at the same time step.

2. offer a wider range of EC solutions, with more diverse trade-offs of classification quality vs. speed. Indeed, early training should result in slow policies with accurate predictions, while more advanced training should result in faster policies with equal or less accurate predictions.

Nevertheless this perspective requires to choose how to dynamically adjust rewards for delay during training. A possible solution is to take inspiration from what is currently done with the exploration coefficient ϵ, i.e. to start with an initial value of penalty coefficient κ (see Eq. 3.35) or trade-off parameter λ (see Eq. 3.28) for example, and to decay/multiply this value at each iteration until reaching a final value fixed in advance. These initial, final and decay values would be additional hyper-parameters to the method.

Rewards based on data complexity Another more general perspective is to define rewards from the data. Indeed, in the industrial application of the thesis, we observed that some data were more complex than others, and we wanted the agent to wait longer on these data. Given expertise on data, we could define rewards from data, so that delay is not penalized by time only, but also according to data *complexity*. Nevertheless, this perspective requires to define the complexity of data, and can be at the origin of new hyper-parameters to tune in relation to the calculation of data complexity and definition of a new reward function.

7.3.2.2 Macro-actions

In this doctoral work, we defined EC as the sequential decision-making problem between postponing the prediction, or predicting a label on the incomplete sequence received so far, such that $\mathcal{A} = \mathcal{A}_c \cup a_d$. Consequently, there is an intrinsic hierarchy in the action definition and we wonder about the benefits that can be brought by the decomposition of the problem into two macro-actions: to delay or to classify, and then to choose a label when the macro-action is to classify.

A similar work is proposed in [51] where the objective is to diagnose a disease by querying as few symptoms as possible. The authors propose a RL approach for the problem. Specifically, they use Hierarchical Reinforcement Learning and they propose a hierarchy at the anatomic part. They segment the symptoms queries by anatomic part. The agent first selects an anatomic part and then selects a symptom on this part specifically.

7.3.2.3 Regression action

In the industrial application of the thesis, the classification problem is to predict labels that are ordinal. A perspective is to define a continuous action which is to make a regression on the label, such that $\mathcal{A}_c = \mathbb{R}$. The solution is then to infer a label among the finite set of labels $\mathcal{L}$ from the regression result. The resolution of the EC-POMDP would consequently be addressed with a policy based approach [98] or an actor-critic approach [67] which allow to learn continuous actions.

A benefit of performing regression instead of classification is to give more flexibility to the agent and to provide more interpretation. Instead of forcing the agent to select a label when it hesitates between two, the agent can predict an average value and consequently the user can deduct that the agent hesitates. Nevertheless this approach necessitates to set a mapping rule from the regression result to the label, and therefore involves additional hyper-parameters.

7.3.2.4 Observations definition

In this doctoral work, we defined observations given to the agent based on prefixes of sequences, such that:

$$o = \mathbf{X}_{:t}$$

A perspective is to augment these observations with more information. For example, observations could be completed with the DNN prediction of evidence $\mathbf{e}$ on the prefix $\mathbf{X}_{:t-1}$ at a time step before, such that:

$$o = (\mathbf{X}_{:t}, \mathbf{e})$$

In this way, the user can define a reward function that penalizes even more the choice of delay action when evidence was predicted on the true label, and inversely, decreases the penalization when evidence was predicted on a wrong label.

Otherwise, observations could be completed with the Q-values predictions of the DNN on the prefix $\mathbf{X}_{:t-1}$ at a time step before, $Q_\Theta(\mathbf{X}_{:t-1}, a)\ \forall a \in \mathcal{A}$, such that:

$$o = (\mathbf{X}_{:t},\ Q_\Theta(\mathbf{X}_{:t-1}, a_d),\ Q_\Theta(\mathbf{X}_{:t-1}, l_1),\ \cdots,\ Q_\Theta(\mathbf{X}_{:t-1}, l_K))$$

In this way, the user can define a reward function that allows to give a feedback on the Q-values. For example, he can penalize even more the choice of delay action when the Q-value of delay was close to the Q-value of the true label. Nevertheless this perspective makes the EC-POMDP definition more complex, especially for the definition of a reward function, and necessitates additional hyper-parameters.

Synthèse par chapitre

Cette thèse porte sur la résolution du problème de classification précoce des séquences temporelles par de l'apprentissage par renforcement profond. Elle a été réalisée dans le cadre d'une convention CIFRE entre les laboratoires GIPSA-lab, FEMTO-ST et le partenaire industriel bioMérieux.

Chapitre 1: Introduction

Dans le Chap. 1, nous introduisons le problème et les objectifs de la thèse et mettons en avant son intérêt pour de nombreuses applications du monde réel.

L'importance de la classification précoce des données séquentielles

Au cours des dernières années, de plus en plus de données ont été produites et stockées tous les jours dans tous les domaines d'activité. L'extraction des connaissances à partir des données est ainsi devenue l'un des sujets de recherche les plus en vogue du siècle. Les données peuvent avoir différents formats. Elles peuvent être disponible en une seule fois, sous la forme d'un ensemble de points indépendants ou non nécessairement structurés. Ou elles peuvent être un ensemble de points ordonnés, où l'ordre joue un rôle important dans la compréhension de la donnée. Ces données sont appelées des *séquences* et sont au cœur de cette thèse. Enregistrements vocaux, vidéos, textes, génomes, signaux de capteurs - toutes sont des données séquentielles. Les séquences étant largement produites dans tous les domaines d'activité (finance, médecine, statistiques, économétrie, sismologie, météorologie, géophysique, etc.), elles ont été largement étudiées au cours des dernières décennies [2, 30, 54, 56].

Dans de nombreuses applications, les séquences peuvent être associées à une *étiquette*. Par exemple, les électrocardiogrammes peuvent provenir d'un patient en bonne santé ou malade. Une séquence de fichiers log d'une machine peut correspondre à une utilisation normale du système ou à une cyberattaque. L'objectif est alors de modéliser la relation entre l'évolution des points dans la séquence et son étiquette, ce qu'on appelle un problème de *classification*. Un exemple populaire dans la littérature est la classification des signaux d'électroencéphalogrammes (EEG) afin d'identifier les états mentaux des patients [63].

Pendant des décennies, les chercheurs se sont intéressés au problème conventionnel de classification des séquences, c.-à-d. lorsque le *classifieur* reçoit la séquence entière avant de prédire l'étiquette. A partir des années 2000, de nouvelles motivations sont apparues. Les séquences sont des données de grande dimension avec beaucoup de points. Généralement, les points arrivent séquentiellement dans le temps et leur acquisition peut être coûteuse. L'utilisation de moins de points pendant l'analyse réduit alors le *coût d'acquisition* total. De plus, il est essentiel pour certaines applications de déduire les étiquettes le plus tôt possible afin que des

actions précoces puissent être envisagées. De telles *applications sensibles au facteur temps* sont par exemple le diagnostic médical pour adopter des traitements précoces [51, 80], la prévision des catastrophes pour anticiper les mesures de sécurité, la détection des intrusions pour se protéger contre les attaques informatiques, etc.

Généralement, la motivation pour la classification précoce (CP) réside dans le fait qu'il n'est pas toujours nécessaire d'observer une séquence entière pour prédire son étiquette. Les informations significatives peuvent être contenues au début de la séquence et les points supplémentaires sont alors inutiles pour la classification [40]. Le problème de CP est complexe. Premièrement, il diffère du problème conventionnel de classification des séquences par son double objectif de prédire l'étiquette et de choisir la longueur de séquence nécessaire pour effectuer la classification. Deuxièmement, il doit faire un *compromis*. Plus il y a de points utilisés pour la classification, plus celle-ci peut être précise. Néanmoins, l'acquisition est plus coûteuse et la prédiction est retardée. La CP est par conséquent un *problème d'optimisation* avec des *objectifs contradictoires*.

Objectifs généraux de la thèse

Dans ce travail de doctorat, nous visons à résoudre le problème de la CP et nous supposons qu'un jeu de données d'apprentissage est disponible, avec un ensemble de séquences complètes et leur étiquette. L'objectif est de proposer un classifieur précoce capable d'effectuer une prédiction en utilisant le moins de points possible. **Nous nous concentrons sur la proposition d'un classifieur précoce *bout-en-bout***, c.-à-d. un modèle unique qui peut à la fois identifier le temps de prédiction optimal sur une séquence incomplète entrante, mais aussi son étiquette de classification. Nous rejetons donc les stratégies qui optimisent les problèmes de classification et de temps de prédiction séparément, comme traditionnellement abordé dans les méthodes de la littérature [7, 8, 20, 38, 39, 41, 70, 71, 79, 107, 110].

Application industrielle La thèse est liée à une application industrielle qui implique certaines spécificités concernant le problème de classification:

- Les données sont des séries temporelles multivariées (MTS),
- Il y a plus de deux étiquettes de classe et elles sont ordonnées,
- Les étiquettes de classe sont inégalement représentées dans le jeu de données d'apprentissage,
- Les erreurs de classification entraînent certains coûts, variables selon l'étiquette réelle et prédite. Les coûts peuvent varier entre les échantillons, en fonction d'informations supplémentaires sur ces données.

Alors que l'objectif est de résoudre le problème général de la CP des séquences temporelles, nous développerons une méthode qui pourra répondre aux spécificités de l'application industrielle.

Hypothèse de recherche

Dans ce travail de doctorat, nous abordons le problème de la CP comme une décision de *classer* une séquence incomplète ou de *reporter* la prédiction. Le classifieur précoce devient

un modèle qui reçoit et analyse les séquences entrantes (mais incomplètes), et choisit séquentiellement certaines actions: soit classer, soit attendre des points supplémentaires. De plus, un classifieur précoce doit *faire un compromis* entre la *précision* et *précocité* de sa prédiction, et ce compromis doit être adapté à chaque échantillon individuellement.

Nous basons ce travail de doctorat sur l'hypothèse de recherche selon laquelle le problème peut être résolu avec de l'apprentissage par renforcement profond (DRL), une discipline populaire pour résoudre des problèmes de prise de décision complexes tels que les jeux vidéo [42, 44, 47, 68, 73, 100], la navigation des robots [11], le diagnostic médical [80], et la conduite de véhicule autonome [91].

Chapitre 2: Analyse du problème de classification précoce des séries temporelles multivariées

Dans le Chap. 2, nous introduisons les séquences temporelles et définissons la classification, en particulier la classification multi-classe, déséquilibrée, ordinale et sensible aux coûts. De plus, nous définissons les objectifs de la CP, introduisons son compromis et formalisons un problème d'optimisation. Tout au long du chapitre, nous introduisons les principales notations et proposons une revue de la littérature sur la manière dont les méthodes existantes abordent les données et le problème.

Classification précoce des séquences

Dans la Sec. 2.3, nous définissons la CP des séquences temporelles comme le problème d'assigner une *étiquette* de classe à des données collectées *séquentiellement*, c.-à-d. pour lesquelles des nouveaux points arrivent au cours du temps. Suivant l'état du processus d'acquisition, la séquence à classer peut être incomplète et par conséquent les séquences à analyser ont des longueurs variables. Plus précisément, nous définissons un classifieur précoce comme un modèle qui doit décider *quand arrêter l'acquisition* de la séquence afin de prédire une étiquette, avec l'objectif d'utiliser le moins de points possible sur chaque séquence. Le modèle doit *adapter* son temps de prédiction sur chaque séquence *individuellement*, suivant la complexité de la donnée. En particulier, il doit collecter plus de points sur les séquences difficilement discriminable et, à l'opposé, il doit rapidement arrêter l'acquisition des séquences plus "simples" dans lesquelles des motifs discriminants précoces peuvent être identifiés.

A la suite de la formulation d'un problème d'optimisation entre deux objectifs contradictoires (classer une séquence rapidement mais précisément), nous introduisons le *compromis* de la CP. Ce dernier consiste à assurer une prédiction d'étiquette optimale tout en minimisant le temps de prédiction (soit le nombre de points utilisés dans chaque séquence pour la classification).

Enjeux des séries temporelles multivariées

Concernant les données à traiter, nous mettons en avant dans la Sec. 2.1 que les séquences temporelles sont des données particulières pour lesquelles il est nécessaire d'utiliser des modèles pouvant prendre en compte les relations *temporelles* dans les points successifs. Dans le cas spécifique des séries temporelles multivariées (MTS, Def. 2.1.1), les modèles doivent également prendre en compte les relations entre les différentes *caractéristiques* mesurées dans chaque vecteur de points de la séquence. En raison de la dimension temporelle dynamique des données,

les modèles doivent pouvoir analyser des séquences incomplètes (Def. 2.1.3) et de longueur variables. En conséquence et dans le but de proposer une méthode applicable à des données sur lesquelles nous avons *aucune expertise*, nous identifions l'intérêt des réseaux de neurones profonds (DNN), parmi lesquels les réseaux de neurones convolutifs (CNN) sont un choix possible de modèle.

Particularités du problème de classification

En plus d'un type de données spécifique à traiter, nous décrivons le problème de classification répondant aux contraintes de l'application industrielle dans la Sec. 2.2, à savoir:

- multi-classe (c.-à-d. qu'il y a deux ou plusieurs étiquettes),
- déséquilibré en classe (c.-à-d. que certaines étiquettes sont sous ou sur-représentées dans le jeu de données d'apprentissage),
- ordinal (c.-à-d. qu'il y a une relation d'ordre entre les étiquettes),
- sensible aux coûts (c.-à-d. que les erreurs de classification impliquent certains coûts qui dépendent de l'étiquette réelle, de l'étiquette prédite et d'autres informations supplémentaires sur les données) et
- sensible au temps (c.-à-d. que la collecte de points dans la série chronologique est coûteuse et que le nombre d'acquisitions nécessaires pour faire une prédiction doit être minimisé).

Nous examinons les travaux connexes de la littérature pour ce problème de classification. Notamment, nous présentons des méthodes de calcul des performances de classification pour les problèmes multi-classes et déséquilibrés. Pour le problème de la classe déséquilibrée, nous serons en mesure de tirer parti des stratégies de ré-échantillonnage de la littérature qui cherchent à créer un équilibre dans les données. Pour le problème de la classification ordinale et sensible au coût, nous proposerons au Chap. 3 une solution qui diffère de celles de la littérature.

Revue de la littérature

Dans la Sec. 2.4, nous faisons une revue de la littérature sur le problème de la thèse et nous montrons qu'il n'y a pas de solution existante qui puisse prendre en compte toutes les spécificités liées aux données MTS et au problème de classification. De plus, les solutions proposées dans la littérature présentent plusieurs inconvénients, parmi lesquels: certaines solutions ne sont pas applicables à des données MTS complexes, d'autres optimisent séparément le problème de classification de celui du temps de prédiction, ou ne prennent pas explicitement en compte le temps dans la décision de classer, etc.

Dans la suite de cette étude, nous proposerons donc une méthode adaptée au problème précis de la CP sur MTS (et plus généralement sur des séquences temporelles de types divers) lorsque les étiquettes sont multi-classes, déséquilibrées, ordinales et lorsque la classification est sensible aux coûts.

Chapitre 3: Formalisation de la CP par un processus décisionnel de Markov partiellement observable

Au Chap. 3, nous développons une méthode autour de la formalisation d'un problème de prise de décision séquentielle. Nous considérons que les séquences $\mathbf{X}$ (Def. 2.1.1) sont reçues

en ligne, avec de nouveaux points arrivant au cours du temps $t \in [1, T]$. Le problème est de choisir une action $a \in \mathcal{A}$ (Eq. 3.1) à tout moment $t \in [1, T]$ à partir des séquences partielles $\mathbf{X}_{:t}$ (Def. 2.1.3). Les actions possibles sont:

- *attendre* afin de rassembler plus de points, notée a_d,
- ou *prédire* une étiquette $l \in \mathcal{L}$ sur la séquence incomplète reçue jusqu'à présent.

Suite à cette formalisation (Sec. 3.1), nous cherchons une solution pouvant fournir un résultat de *bout-en-bout*, c.-à-d. des séquences incomplètes à la décision d'attente ou de prédiction d'étiquette.

Dans la Sec. 3.2, nous montrons que la résolution du problème de prise de décision séquentielle par apprentissage supervisé (SL) entraîne un nouveau défi: construire un jeu de données d'apprentissage étiqueté, avec une supervision sur les actions optimales à tous les pas de temps de la séquence. Nous montrons que les méthodes de la littérature ne sont pas directement applicables et donc la résolution du problème avec SL est un sujet de recherche majeur.

Dans la Sec. 3.3, nous montrons que la CP peut être décrite comme un processus de Markov avec des actions. Dans la Sec. 3.4, nous définissons les récompenses associées à la CP afin que le problème de prise de décision puisse être décrit par un processus de décision de Markov (MDP) puis résolu dans un cadre d'apprentissage par renforcement (RL). Nous proposons plusieurs stratégies pour trouver un compromis entre la précocité et la précision de la classification dans la définition de la fonction de récompense. La méthode proposée permet à l'utilisateur de définir l'*importance relative du temps* par rapport à la qualité de classification pour son application. De plus, plusieurs définitions de récompenses sont proposées en fonction de la nature du problème de classification (sensible aux coûts versus insensible aux coûts) et la solution permet de prendre en compte les coûts de mauvaise classification définis par l'application s'ils existent. Nous montrons que le MDP décrivant le problème de CP, noté CP-POMDP, est en fait *partiellement observable* car, lors de l'acquisition en ligne des séquences, nous n'avons pas accès aux futurs points ni à l'étiquette à prévoir, alors que ces informations seront utilisées par l'environnement lors de la formation de l'agent.

Dans ce travail de doctorat, nous évaluons la méthode sur des séries temporelles à la fois multivariées et univariées, selon les expériences. Cependant, le cadre mathématique est proposé pour le problème général de la CP des séquences et il peut donc être appliqué à différents types de séquences (image, texte, etc.).

Chapitre 4: Résolution du CP-POMDP avec de l'apprentissage par renforcement profond

Dans le Chap. 4, nous cherchons à résoudre le CP-POMDP, c.-à-d. à trouver sa politique optimale π^* (Eq. 3.19). En particulier, nous cherchons à former un agent classifieur précoce avec de l'apprentissage par renforcement (RL) dans un environnement caractérisé par le CP-POMDP. L'objectif est alors d'évaluer si l'agent peut atteindre les objectifs de CP définis dans la Sec. 2.3.

Dans la Sec. 4.1, nous proposons de résoudre le CP-POMDP avec une approche d'apprentissage par renforcement basée sur la valeur. En effet, l'espace d'actions $\mathcal{A}$ (Eq. 3.26) du CP-POMDP est fini et petit ($K+1$ actions), ce qui signifie qu'il est possible d'apprendre la valeur Q (Eq. 3.20) de chaque action $a \in \mathcal{A}$. L'objectif devient alors d'apprendre la fonction

Q optimale, notée Q^*, de telle sorte que la politique optimale π^* puisse être déduite par:

$$\pi^*(o) = \arg\max_{a \in \mathcal{A}} Q^*(o, a) \; \forall o \in \mathcal{O}$$

Comme expliqué dans la Sec. 3.4, l'environnement caractérisé par le CP-POMDP est partiellement observable. La fonction Q est définie sur l'espace des observations $\mathcal{O}$ (Eq. 3.25) représentant l'ensemble des séquences partielles $\mathbf{X}_{:t}, \forall t \in [1, T]$ qui est continu. Du fait que $\mathcal{O}$ soit continu, la fonction de valeur d'action Q ne peut pas être représentée par une table pour chaque couple d'observations et d'actions. Au lieu de cela, la fonction Q doit être approximée par une fonction [97]. Dans ce travail de doctorat, nous utilisons de l'apprentissage par renforcement profond (DRL) pour former l'agent classifieur précoce, ce qui consiste à approximer Q par un réseau neuronal profond (DNN) Q_Θ défini sur des paramètres Θ. L'objectif est de trouver les paramètres optimaux Θ^* du DNN tels que:

$$Q^* = Q_{\Theta^*} \tag{7.1}$$

Dans la Sec. 4.2, nous proposons un pipeline pour résoudre le CP-POMDP avec du DRL et apprendre les paramètres optimaux Θ^*, en formant un agent avec l'algorithme Double Deep-Q-Network (DDQN) [100]. Le pipeline permet aussi de sélectionner la politique optimale de l'agent classifieur précoce.

Dans la Sec. 4.3, nous menons une étude expérimentale pour évaluer la méthode proposée et la comparer à d'autres solutions de la littérature. Nous montrons sur le jeu de données public UCR [21] que le problème de CP peut être résolu avec un agent par DRL et en offrant un modèle de bout-en-bout. Sur différents jeu de données, l'agent atteint les objectifs de CP: trouver un compromis entre la qualité de la classification et sa précocité. Nous montrons que l'agent peut continuellement *adapter* son comportement sans intervention humaine. Il apprend *simultanément* les descripteurs dans les séquences pour le problème de classification et les règles de prise de décision sur les temps de prédiction. En outre, nous montrons expérimentalement que l'agent obtient des résultats similaires ou meilleurs en termes de précision et de temps de prédiction par rapport aux méthodes de la littérature.

Nous identifions néanmoins des limites quant à l'application de l'algorithme DDQN pour la résolution du CP-POMDP. Nous montrons que la mémoire de relecture de l'agent peut devenir *déséquilibrée* au cours de sa formation, affaiblissant ainsi son apprentissage global. Dans la suite de l'étude, nous chercherons donc à optimiser l'algorithme DDQN pour la CP et à évaluer si la résolution d'une mémoire mal équilibrée améliore l'entraînement et les performances de l'agent.

Chapitre 5: Stratégies de gestion de la mémoire pour optimiser la résolution du CP-POMDP

Dans le Chap. 5, nous cherchons à optimiser la résolution du CP-POMDP avec une gestion robuste de la mémoire de l'agent lors de l'application de l'algorithme DDQN. Nous soulevons trois questions sur la gestion de la mémoire et une question sur la gestion des épisodes:

(1) *Quelles interactions doivent être stockées?*
(2) *Quelles interactions doivent être échantillonnées?*
(3) *Quelles interactions doivent être supprimées (lorsque la mémoire est pleine)?*
(4) *Comment initialiser un épisode d'entraînement?*

Nous étudions comment répondre à ces questions tout en abordant les spécificités du CP-POMDP, et nous proposons des révisions de l'algorithme DDQN pour corriger le problème de mémoire déséquilibrée de l'agent. De plus, nous proposons deux adaptations de l'algorithme DDQN, selon que l'application est livrée avec un jeu de données d'apprentissage fini (apprentissage par lots) ou, au contraire, peut générer de nouvelles données d'apprentissage au fil du temps (apprentissage en ligne). Dans les expériences, nous évaluons si les contributions ont un effet positif sur la formation et les performances globales de l'agent.

Dans la Sec. 5.1, nous faisons une revue de la littérature sur la gestion de la mémoire de l'agent dans le domaine du DRL.

Dans la Sec. 5.2, nous optimisons l'algorithme DDQN pour l'apprentissage en ligne, c.-à-d. lorsqu'il y a des répétitions successives entre (a) collecte d'interaction et (b) optimisation de la politique. Nous proposons des stratégies d'*échantillonnage prioritaire*, *stockage prioritaire* et *initialisation d'épisode aléatoire*.

Dans la Sec. 5.3, nous adaptons l'algorithme DDQN pour l'apprentissage par lots, c.-à-d. pour les applications ayant un jeu de données d'apprentissage fini, entraînant un nombre maximum d'interactions possibles à collecter entre l'agent et l'environnement. Nous proposons de dissocier la collecte des interactions de l'optimisation de la politique, et nous appliquons la *stratégie d'échantillonnage prioritaire*.

Dans la Sec. 5.4, nous introduisons un pipeline d'évaluation pour comparer les différentes versions de l'algorithme, avec ou sans les stratégies proposées. Nous évaluons ensuite les effets d'*échantillonnage prioritaire*, *stockage prioritaire* et *initialisation d'épisode aléatoire* sur un jeu de données lié à l'application industrielle. Nous montrons que les différentes stratégies améliorent statistiquement les performances de l'agent en termes de précision et de vitesse.

Dans la Sec. 5.5, nous entraînons un ensemble de réseaux neuronaux profonds (DNNs) statiques à classer les séquences partielles à chaque pas de temps. Ces DNNs statiques ont une architecture équivalente à celle de l'agent. Nous évaluons la performance des DNNs statiques en termes de précision et vitesse par rapport à l'agent formé par DRL. Nous montrons que l'agent classifieur précoce (pour lequel le temps de prédiction est adaptatif, contrairement aux DNNs statiques) améliore le compromis général de la CP entre précision et vitesse.

Chapitre 6: Interprétation des politiques

Au Chap. 6, nous proposons des outils pour interpréter les décisions prises par l'agent, c.-à-d. sa politique. Tout d'abord, nous appliquons la méthode Class Activation Map [116] pour mettre en évidence les points d'une séquence de test qui contribuent le plus aux prédictions de l'agent. Nous fournissons ensuite des visualisations sur les prédictions faites par l'agent, sur des séquences de test, et montrons certaines de leurs dynamiques. Enfin et dans une perspective de travaux futurs, nous appliquons la méthode introduite dans [89] pour mesurer l'incertitude des prédictions du réseau neuronal profond, et nous l'adaptons au problème de CP.

Class Activation Map

Un premier outil d'interprétation est donné dans la Sec. 6.1 où nous appliquons une méthode de la littérature afin d'identifier les points dans des séquences partielles qui influencent le plus les prédictions de l'agent. Nous appliquons la méthode Class Activation Map (CAM) [116] sur

le réseau neuronal Q_Θ et nous montrons, lors d'évaluations expérimentales, que la méthode permet d'identifier comment les points dans les séquences contribuent aux prédictions des Q-valeurs. Des illustrations sont fournies sur le jeu de données issu de l'application industrielle. Elles montrent comment nous avons pu tirer quelques interprétations des décisions prises par l'agent, notamment en mettant en évidence les motifs dans les séquences partielles identifiés comme les plus pertinents par l'agent lors de sa prédiction. A l'issue de l'évaluation, nous avons conclu que cette méthode peut être utilisée à deux fins. Premièrement, elle peut être utilisée lors de l'application d'une politique sur des données de test, afin de fournir une justification sur les décisions de l'agent. Ensuite, elle peut être utilisée comme une deuxième étape de sélection des politiques optimales, pour comparer plusieurs politiques entre elles et sélectionner celle pour laquelle les décisions sont les plus cohérentes avec l'application, sur la base de notre expertise des données.

Q-valeurs

Pour enrichir les interprétations des politiques, nous visualisons en Sec. 6.2 les Q-valeurs estimées par l'agent durant l'acquisition des séquences. Lors d'une évaluation expérimentale, nous avons visualisé les prédictions des Q-valeurs faites par l'agent sur les données de test issues de l'application industrielle, et nous avons observé les cas suivants.

Tout d'abord, nous avons observé que la Q-valeur de l'action d'attente est souvent proche de la Q-valeur de classification la plus élevée. Dans certains cas, la plus grande Q-valeur de classification reste légèrement inférieure à la Q-valeur d'attente, pour de nombreux pas de temps successifs, comme illustré sur la Fig. 6.9. Par conséquent, au cours de ces pas de temps, l'agent continue d'attendre tandis que ses prédictions de Q-valeurs montrent qu'il a réussi à identifier la bonne étiquette. C'est donc une perte de temps de la part de l'agent, puisqu'un bon classement aurait pu se faire plus rapidement.

Deuxièmement, on peut observer que l'agent parvient à prédire les Q-valeurs qui sont ordonnées, comme défini dans la fonction de récompense, mais hésite parfois entre plusieurs étiquettes, comme illustré dans la Fig. 6.10. Pour ces cas particuliers, nous voudrions justifier si l'agent continue d'attendre parce qu'il est incertain dans l'étiquette des données de test ou en raison d'une politique sous-optimale. Dans la suite du chapitre, nous cherchons donc à mesurer l'incertitude du DNN afin d'apporter une interprétation supplémentaire dans les décisions de l'agent.

De plus et comme objectif secondaire d'un travail futur, notre motivation derrière l'estimation de l'incertitude est de l'utiliser pour valider ou rejeter certaines prédictions de l'agent, et éventuellement accélérer certaines prédictions, par rapport aux scénarios évoqués ci-dessus. Nous pensons qu'il est possible de déclencher une prédiction lorsque l'incertitude est faible et lorsque la Q-valeur de classification la plus élevée est proche de celle de l'attente pendant plusieurs pas de temps consécutifs. De plus, nous pensons que certaines décisions hâtives prises par l'agent peuvent être rejetées lorsque l'incertitude est trop élevée.

Estimation de l'incertitude du DNN

Dans la Sec. 6.3, nous appliquons une méthode issue de la littérature pour estimer l'incertitude des prédictions de l'agent. Une première étude expérimentale est donnée sur une version simplifiée du jeu de données industriel pour lequel il y a moins d'étiquettes à prédire.

Dans cette section, nous avons montré comment appliquer la méthode de [89] pour

l'estimation de l'incertitude du DNN, afin d'interpréter les prédictions faites par l'agent sur les données de test. Nous avons montré comment calculer l'évidence sur les étiquettes du problème de classification et comment en retirer à la fois des estimations de probabilité pour chaque étiquette et une estimation d'incertitude. Les inconvénients de la méthode sont liés à un plus grand nombre d'hyper-paramètres à régler. Ils sont causés par l'architecture multi-branches du DNN.

Chapitre 7: Conclusion

Au Chap. 7, nous concluons sur le travail doctoral et donnons quelques perspectives pour de futures études. La principale contribution de ce travail de doctorat a été de formaliser le problème de la CP des séquences temporelles en tant que problème de prise de décision séquentielle. Le classifieur précoce doit décider entre classer une séquence incomplète ou retarder la prédiction pour collecter des points supplémentaires. Plus précisément, nous avons décrit la CP par un POMDP, noté CP-POMDP, en définissant un ensemble d'états, d'actions et d'observations associés au problème et en définissant une fonction de récompense qui représente le compromis de la CP. La deuxième contribution majeure de ce travail a été d'adapter et d'optimiser un algorithme de DRL existant aux spécificités du CP-POMDP. Pour cela, nous avons proposé des stratégies de gestion des épisodes et de la mémoire. Nous avons initié ces stratégies pour résoudre le problème de mémoire déséquilibrée de l'agent causé par l'algorithme d'origine, et nous avons montré qu'elles amélioraient la formation globale de l'agent. Enfin, d'autres contributions secondaires ont été de proposer des solutions pour l'interprétation des politiques de l'agent.

À notre connaissance et contrairement aux méthodes de la littérature [7, 8, 20, 38, 39, 41, 70, 71, 79, 107, 110], la méthode proposée est la première à vérifier tous les avantages suivants:

- La méthode traite simultanément les problèmes de classification et de temps de prédiction. Le modèle *unique* est capable d'apprendre simultanément des descripteurs optimaux dans les séquences pour la classification et des décisions stratégiques optimales pour le temps de prédiction. La méthode implique une seule phase d'optimisation et il n'est pas nécessaire d'optimiser chaque sous-problème séparément.
- La méthode offre un modèle *de bout-en-bout* qui ne nécessite pas d'expertise sur les données à analyser et qui n'implique pas une étape préliminaire d'*extraction des descripteurs*. Au lieu de cela, le DNN reçoit des données d'entrée *brutes*, ayant subi ni réduction de dimension ni pré-traitement. Le DNN est ainsi en charge de synthétiser les informations contenues dans les points de données, d'extraire les descripteurs les plus pertinents et d'apprendre les décisions optimales.
- La méthode ne fait pas d'hypothèses sur les données d'entrée. Elle est *générique* et peut être appliqué à différents types de données séquentielles, telles que des séries temporelles numériques, des séquences symboliques, des séquences d'images, des textes, etc.
- La méthode minimise directement un critère basé sur le temps et l'utilisateur peut ajuster son *importance* par rapport à la qualité de la classification.
- La méthode permet de résoudre des problèmes de classification qui sont sensibles au coût. En effet, les coûts de mauvaise classification peuvent être pris en compte directement

dans la définition du CP-POMDP, via la fonction de récompense. Par conséquent, l'approche proposée peut résoudre une grande variété de problèmes de CP. Par exemple, le diagnostic médical est connu pour être un problème de classification sensible aux coûts en raison de la gravité de la non-détection d'une maladie grave par rapport au déclenchement d'une fausse alarme.

- La méthode peut gérer à la fois les problèmes de classification binaires et multi-classes.
- La méthode peut gérer les problèmes de classification qui sont ordinaux, en définissant une fonction de récompense qui pénalise davantage les erreurs de classification lorsque l'étiquette prédite est éloignée de la véritable étiquette.
- La méthode est robuste aux problèmes de classification pour lesquels le jeu de données d'apprentissage est déséquilibré, grâce aux stratégies de gestion de la mémoire proposées dans la thèse.
- Pendant la formation de l'agent, l'utilisateur bénéficie d'une gamme de modèles pour la CP avec différents compromis entre qualité de classification et vitesse. Par conséquent, l'utilisateur peut choisir le modèle qui correspond le mieux à son domaine d'application, mais aussi arrêter la formation de l'agent dès qu'il est satisfait par un modèle obtenu.

Dans la Sec. 7.3, nous présentons des perspectives en relation avec ce travail de thèse.

Appendices

Appendix A

Experimental comparison between EC-POMDP models

In Chap. 3, we formalized the early classification (EC) problem as a Partially Observable Markov Decision Process (POMDP), noted EC-POMDP, and we proposed several definitions of rewards R_d for the delay action a_d (see Tab. 3.2):

$$R_d \in \{R_{d,shaping}, R_{d,discount}\}$$

In this study, we seek to assess the impact of delay reward shaping $R_{d,shaping}$ vs. reward discounting $R_{d,discount}$ in the reward definition of the EC-POMDP. We are interested in comparing if one of the above definition leads to better policies. To this end, we carry out an experiment using the same experimental pipeline as in Sec. 5.4.3.

A.1 EC-POMDP models

From reward definitions proposed in Tab. 3.2, we define two models of EC-POMDP:

- $M_{shaping} = \{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ is an EC-POMDP where the delay action a_d is rewarded negatively over time with $R_d = R_{d,shape}$ and rewards are not discounted, $\gamma = 1$.
- $M_{discount} = \{\mathcal{S}, \mathcal{A}, P, R, \mathcal{O}, \Psi, \gamma\}$ is an EC-POMDP where rewards are discounted, $\gamma < 1$. The action of delay a_d is not rewarded, $R_d = R_{d,discount}$. The agent collects rewards (positive or negative) from classification actions only.

Both $M_{shaping}$ and $M_{discount}$ use cost-insensitive classification rewards, such that $R_c = R_{c,ins}$ (Eq. 3.31) with $r_+ = +1$ and $r_- = -1$:

$$R_{c,ins}((\mathbf{X}, l, t), a) = \begin{cases} +1 & \text{if } a = l \\ -1 & \text{if } a \in \mathcal{A}_c \setminus \{l\} \end{cases}$$

A.2 Experimental evaluation

Dataset We use the industrial dataset from Sec. 5.4.1, illustrated in Fig. 5.5 and Fig. 5.4.

A.2.1 Experimental pipeline

We train early classifier agents on the two EC-POMDP definitions, $M_{shaping}$ and $M_{discount}$, with DDQN algorithm in batch learning and with prioritized sampling, as introduced in Algo. 5. We carry out an experiment using the same experimental pipeline as in Sec. 5.4.3. We remind that this experimental pipeline initially aimed at comparing several versions of DDQN algorithm, with and without the different strategies of episode and memory management proposed in Chap. 5. The objective of the experimental pipeline was to statistically evaluate if some versions of the algorithm resulted in policies with better EC performance compared to the other versions. More generally, this same experimental pipeline can be used to compare two sets of trainings, and to statistically evaluate if one of the two sets results in better policies than the other.

We perform 50 trainings of the agent on each model of EC-POMDP, following the same training pipeline as in Sec. 5.4.3.1. Simultaneously, to evaluate if both EC-POMDP models achieve comparable best classification accuracy under different trade-offs, we apply the evaluation pipeline from Sec. 5.4.3.2 for each training, on the validation set. Finally, we compare the two sets of training, each one being dedicated to a model of the EC-POMDP, with the comparison pipeline from Sec. 5.4.3.3 and Sec. 5.4.3.4.

A.2.2 Results

Comparison of best performance We report in Fig. A.1 the top-5 optimal policies obtained within several ranges of prediction times for each EC-POMDP model. For each model, accuracy rapidly increases when prediction time reaches $t_{pred} = 30$. Then, accuracy slightly gets better and stabilizes over the acquisition process. From Fig. A.1, we observe that $M_{shaping}$ results in top-5 policies that are the most accurate, i.e. with higher Acc than $M_{discount}$, under all trade-offs of prediction time t_{pred}. Also, tests from Tab. A.1 allow to reject the null hypothesis that both POMDP models achieve comparable max Acc along training. Indeed, Fig. A.2 show that $M_{shaping}$ improve max Acc over $M_{discount}$. In other words, $M_{shaping}$ results in policies with the best classification quality. As a conclusion, the best (and top-5) classification performance is achieved when the agent is trained under $M_{shaping}$.

Comparison of average performance Then, instead of comparing the best performance, we seek to compare the average performance of the agent when trained under the different models $M_{shaping}$ and $M_{discount}$. To do so, we average the performance of each of its policies from the same training session, allowing to illustrate the overall performance of the agent throughout its training and not at a specific moment of its training. Tests from Tab. A.1 show that $M_{shaping}$ and $M_{discount}$ models have statistically comparable mean Acc, which means that no model improves the overall classification quality of the agent compared to the other model. Nevertheless, $M_{discount}$ shorten mean t_{pred} over $M_{shaping}$ which means that $M_{discount}$ results in earliest classification times compared to $M_{shaping}$.

Comparison of stability In terms of stability, measured through the metrics of stdev t_{pred} and stdev Acc, the statistical tests from Tab. A.1 lead to the conclusion that $M_{shaping}$ is more uneven than $M_{discount}$ during its trainings. As shown in Fig. A.2, the accuracy performance

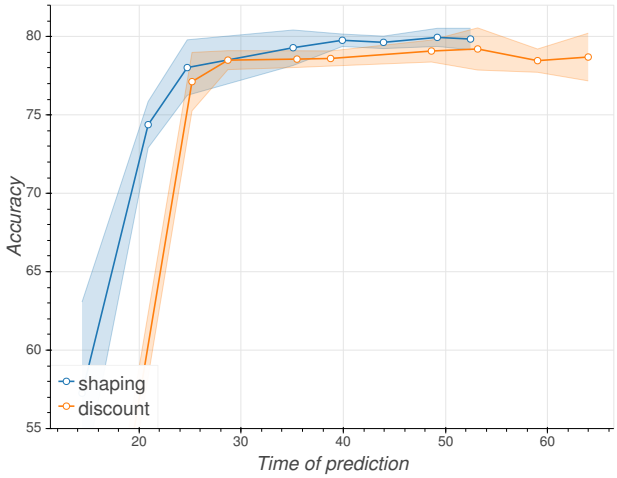

Figure A.1: **Evaluation of top-5 policies from $M_{shaping}$ and $M_{discount}$ on the test set.** The evaluation involves 9 distinct time intervals $[t_{inf}, t_{sup}[$ between $[1, T]$. For each time interval $[t_{inf}, t_{sup}[$, the top-5 policies (which have an average prediction time $t_{pred} \in [t_{inf}, t_{sup}[$ and highest Acc) were selected from the validation set. The full line represents mean accuracy and the band is the accuracy standard deviation on the 5 policies on the test set.

and prediction time of the agent's policies are more irregular during the trainings of $M_{shaping}$ compared to those of $M_{discount}$.

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 A.1: **Statistical comparison between $M_{shaping}$ and $M_{discount}$ performance metrics.** The table reports p-values of Mann-Whitney rank tests on the null hypothesis that $M_{shaping}$ and $M_{discount}$ have comparable score for each performance metric (max Acc, mean Acc, stdev Acc, mean t_{pred} and stdev t_{pred}) from Fig. A.2. The null hypothesis is rejected in favor of the alternative hypothesis on tests with a p-value below 0.05, shown in bold. The alternative hypothesis is that the metric performance is different between the two models. Fig. A.2 shows which model has the greatest score.

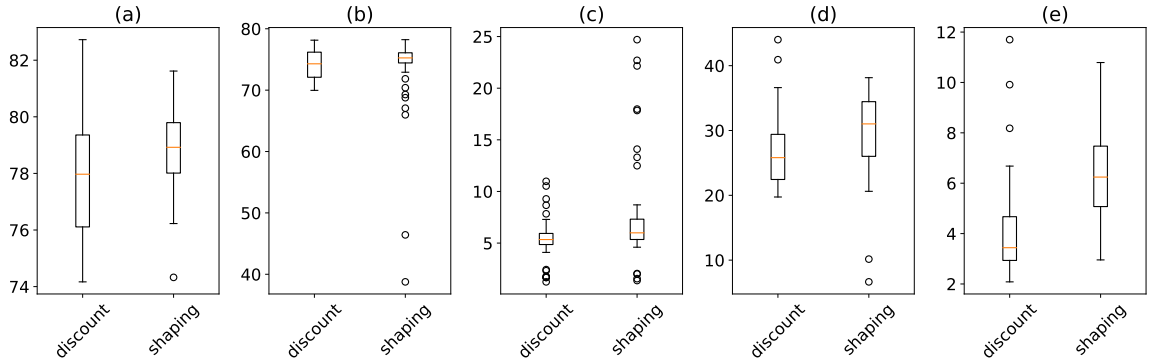

Figure A.2: **Distribution of performance metrics from $M_{shaping}$ and $M_{discount}$ on the validation set.** (a) Max *Acc.* (b) Mean *Acc.* (c) Stdev *Acc.* (d) Mean t_{pred}. (e) Stdev t_{pred}. For each model, the distribution is calculated on all the evaluations that were performed on the validation set, for each set of hyper-parameters tested.

Appendix B

Experimental evaluation on EC-POMDP solving with a policy-based approach

We present a study conducted as part of an internship and which was intended to directly approximate the policy π of the agent by a DNN π_Θ with parameters Θ, and not through the approximation of the Q-function as proposed in Chap. 4.

Motivation We proposed to solve the EC-POMDP with a policy-based approach, and specifically with the state-of-the-art algorithm at the moment: Asynchronous Advantage Actor Critic (A3C) algorithm [67]. The objectives were to evaluate if the policies learned using this algorithm achieved better performance of early classification than those learned with DDQN algorithm from [100].

Algorithm The A3C algorithm is a policy-based approach which seeks to directly approximate the policy π with a DNN π_Θ with parameters Θ. A3C algorithm updates the policy parameters Θ asynchronously using independent agents that each train on their copy of the environment. The algorithm is given in [67].

Experimental evaluation We use the same EC-POMDP model as in Sec. 4.3.2. Also, the experimental pipeline used for the training of the agent, its evaluation, and the selection of its optimal policy is the same as in Sec. 4.3.3.

Tab. B.1 reports the performance in accuracy *Acc* and prediction time t_{pred} of Early Classification on Time Series (ECTS) [109] and Early Distinctive Shapelet Classification (EDSC) [110] methods, on testing datasets from the UCR Time Series Archive [21]. It compares these performances to those achieved by the optimal policies trained on the EC-POMDP with both A3C and DDQN algorithms.

On ECG dataset, A3C algorithm provides a policy with faster prediction time t_{pred} than DDQN but with lower accuracy *Acc*. On Gun-Point and Wafer datasets, A3C algorithm provides policies with almost similar *Acc* than DDQN but with slower t_{pred}. Consequently, these experiments did not show any significant performance improvements with the policies trained using A3C algorithm compared to those trained using DDQN algorithm.

Dataset		ECTS [109]	EDSC [110]	Full 1NN [21]	A3C [67]	DQN [68]
ECG	*Acc*	**89**	88	88	85	**89**
	t_{pred}	57,71	30,93	96	**11,98**	16,09
CBF	*Acc*	85,2	87	85,2	**93,44**	
	t_{pred}	91,73	44,84	128	**38,76**	
Gun-Point	*Acc*	86,67	94,67	91,33	**96,67**	96
	t_{pred}	70,39	69,3	150	42,1	**32,47**
Synthetic Control	*Acc*	89	87,66	88	**98**	
	t_{pred}	53,98	33,36	60	**24,4**	
Wafer	*Acc*	99,08	98,87	99,55	**99,82**	99,32
	t_{pred}	67,39	38,97	152	26.11	**5,73**
Two Patterns	*Acc*	86,48	80,6	91	**99,98**	
	t_{pred}	111,1	**82,33**	128	86,55	
OliveOil	*Acc*	**90**	76,67	86,7	**90**	
	t_{pred}	497,83	**213,48**	570	249	

Table B.1: **Evaluation of optimal policies trained with A3C algorithm and DDQN algorithm, on testing sets from UCR Time Series Archive [21].** Optimal policies were selected following procedure from Sec. 5.4.3.4 on the training set. They are compared to the Early Classification on Time Series (ECTS) method from [109], the Early Distinctive Shapelet Classification (EDSC) method from [110], and the 1 Nearest Neighbor (1NN Full) method provided in UCR archive [21]. *Acc* is defined in Eq. 4.8. t_{pred} is defined in Eq. 4.9. Best performances are written in bold. DQN algorithm has not been tested on CBF, Synthetic Control, Two Patterns and Olive Oil datasets.

Appendix C

Illustrations of Class Activation Map

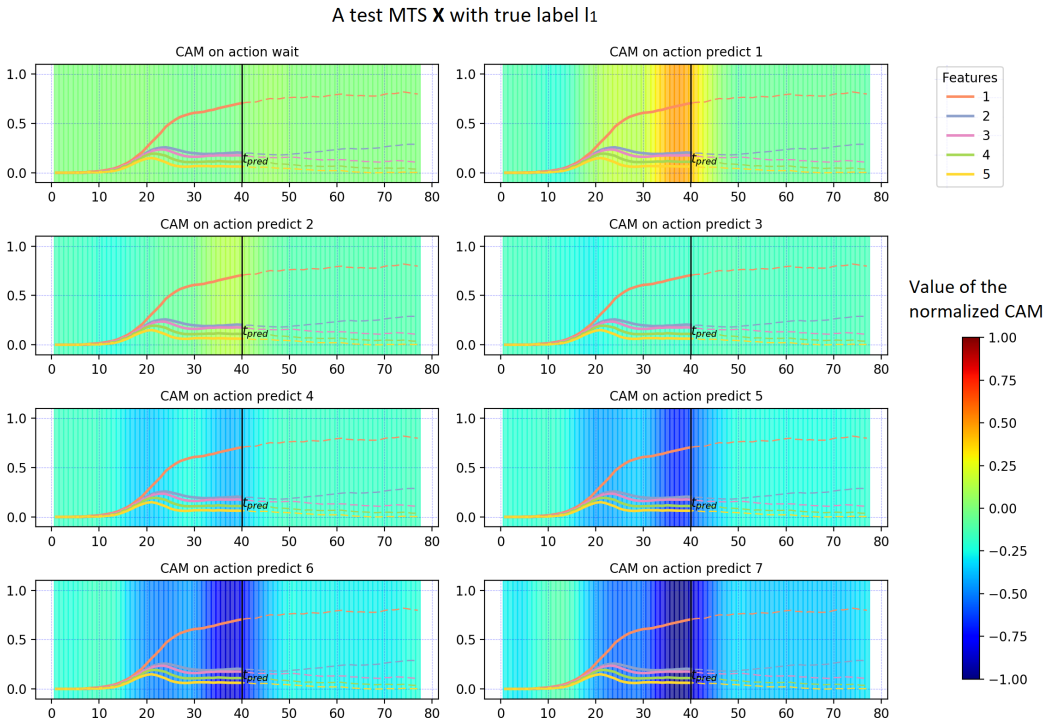

Figure C.1: **CAMs for each action $a \in \mathcal{A}$, on a partial test MTS $\mathbf{X}_{:40}$ with reference label l_1.** The agent predicted label l_1 at $t_{pred} = 40$.
CAMs in red highlight the patterns in the MTS that contribute positively to the action selection. CAMs in blue highlight the patterns in the MTS that contribute negatively to the action selection.
At time step 40 of the acquisition process, time at which the agent chooses to classify, the DNN Q_Θ identifies a pattern in the data points $(\mathbf{x}_{20}, \cdots, \mathbf{x}_{40})$ that contributes positively to the Q-value of label l_1, with the most significant data points being $(\mathbf{x}_{35}, \cdots, \mathbf{x}_{40})$. These last points also slightly contribute positively to the Q-value of label l_2. The Q-values for labels l_5, l_6 and l_7 are negatively affected by a pattern identified in the data points $(\mathbf{x}_{20}, \cdots, \mathbf{x}_{40})$.

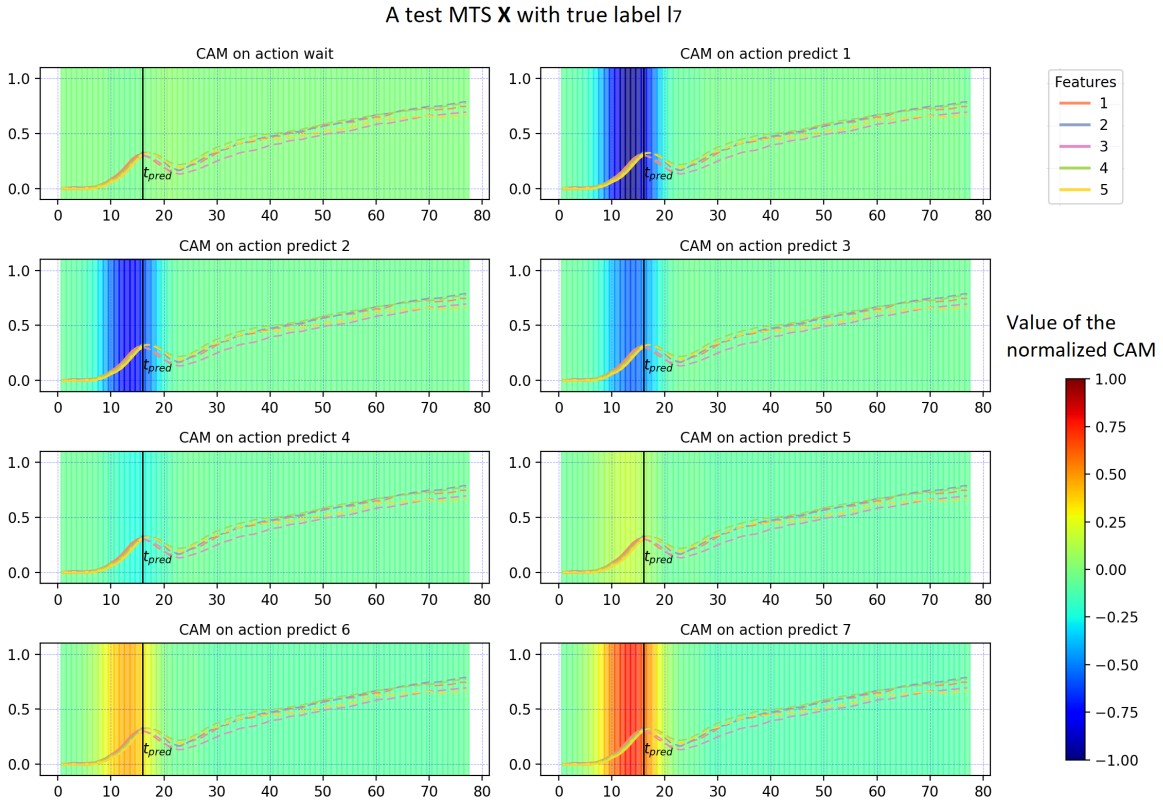

Figure C.2: **CAMs on a partial test MTS $\mathbf{X}_{:16}$ with reference label l_7.** The agent predicted label l_7 at $t_{pred} = 16$.
CAMs in red highlight the patterns in the MTS that contribute positively to the action selection. CAMs in blue highlight the patterns in the MTS that contribute negatively to the action selection.
At time step 16 of the acquisition process, time at which the agent chooses to classify, the DNN Q_Θ identifies a pattern in the data points $(\mathbf{x}_9, \cdots, \mathbf{x}_{16})$ that decreases the Q-values of labels l_1, l_2, l_3 (and slightly that of label l_4), and increases the Q-values of labels l_6 and l_7 (and slightly that of label l_5). Previous data points $(\mathbf{x}_1, \cdots, \mathbf{x}_8)$ have a zero contribution to the predicted action values for all actions. This is also the case of unknown future data points $(\mathbf{x}_{17}, \cdots, \mathbf{x}_{77})$ that have been replaced by zeros: they do not contribute positively nor negatively to the Q-values of all actions.

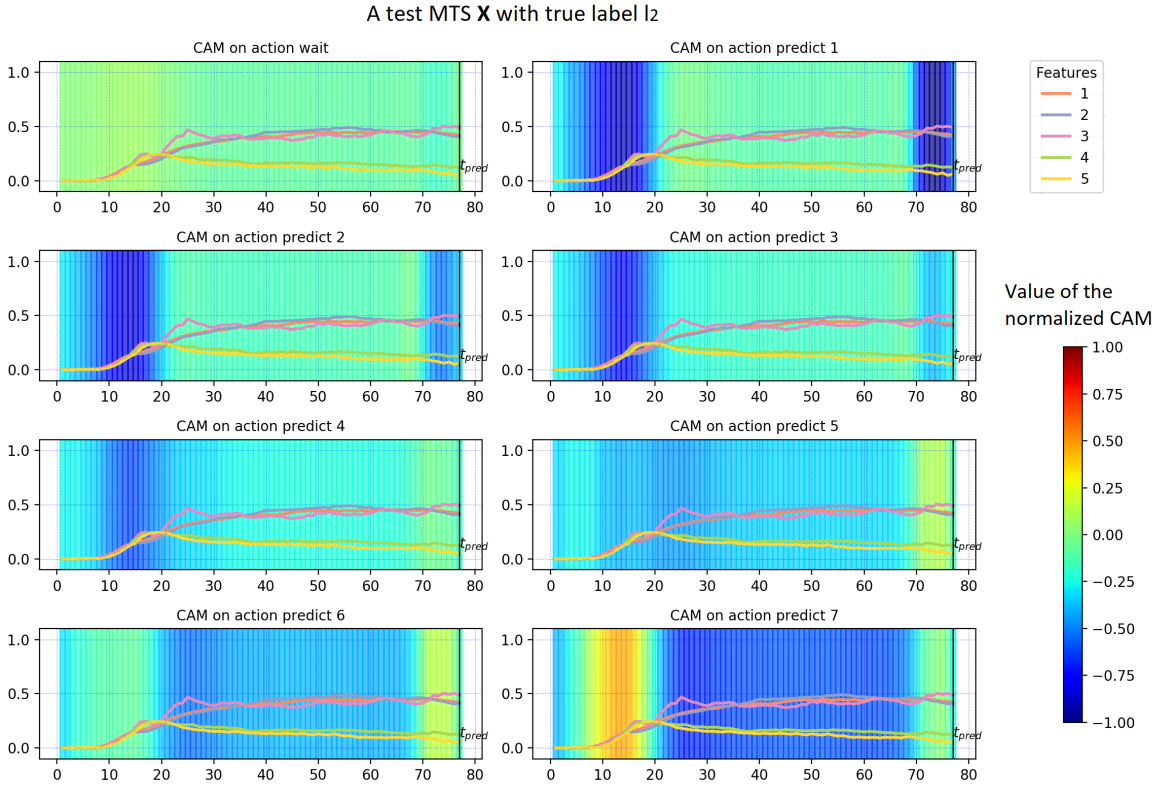

Figure C.3: **CAMs for each action $a \in \mathcal{A}$, on a partial test MTS $\mathbf{X}_{:77}$ with reference label l_2.** The agent is forced to predict a label at the end of the sequence acquisition. It predicted label l_3 at $t_{pred} = 77$.
CAMs in red highlight the patterns in the MTS that contribute positively to the action selection. CAMs in blue highlight the patterns in the MTS that contribute negatively to the action selection. At time step 77 of the acquisition process, the data points $(\mathbf{x}_1, \cdots, \mathbf{x}_{20})$ and $(\mathbf{x}_{70}, \cdots, \mathbf{x}_{77})$ make a large negative contribution to the Q-values of labels l_1, l_2, and l_3. Also, data points $(\mathbf{x}_1, \cdots, \mathbf{x}_{20})$ decrease the Q-value of label l_4. The Q-values of these labels are not increased by the identification of any positive patterns in the sequence. Also, the data points $(\mathbf{x}_{20}, \cdots, \mathbf{x}_{70})$ make a large negative contribution to the Q-values of labels l_5, l_6, l_7. Despite the identification of a pattern in the data points $(\mathbf{x}_{10}, \cdots, \mathbf{x}_{18})$ that contribute positively to the Q-value of label l_7, this Q-value is decreased by the pattern in data points $(\mathbf{x}_{20}, \cdots, \mathbf{x}_{70})$. The average contribution of each data point in the sequence to the Q-value of label l_3 is larger than that of label l_7.

Appendix D

List of hyper-parameters and their values

In this appendix, we present the list of hyper-parameters and the range of values tested during the experimental evaluations of Chap. 4 and Chap. 5. We remind that the method was to train an early classifier agent with Deep Reinforcement Learning, and specifically with the Double Deep-Q-Network algorithm from [100]. This algorithm involves a list of hyper-parameters that need to be set and optimized. Moreover, in Chap. 5, we introduced strategies of memory management that involved additional hyper-parameters to the method.

Hyper-parameter	Values	Description
mini-batch size	18-32-64-128	Number of transitions sampled from the replay memory to perform one update of the DNN's parameters.
learning rate λ	from 10^{-2} to 10^{-6}	The learning rate used for the stochastic gradient descent.
replay memory size	from 10^3 to 10^6	Size of the agent's replay memory.
replay memory start size	10% of the replay memory size	A random policy is run for this number of interactions to pre-fill the agent's replay memory.
target network update frequency	from 1000 to 10000	Steps at which the parameters of the target network are updated.
discount factor γ	from 0.5 to 1	Used in the EC-POMDP definition.
initial exploration ϵ	1	The exploration rate initial value at the beginning of an episode.
final exploration ϵ	0.1	The exploration rate final value at the end of an episode.
exploration decay	from 0.9 to 0.99999	Multiplying factor applied after each interaction to the exploration rate.
Prioritized sampling parameter μ	from 0.2 to 0.8	Hyper-parameter of the prioritized sampling strategy (see Sec. 5.2.1)
Prioritized storing parameter ρ	from 0.2 to 0.8	Hyper-parameter of the prioritized storing strategy (see Sec. 5.2.2)

Appendix E

Examples of Deep Neural Network architectures

In this appendix, we illustrate some architectures of Deep Neural Networks (DNNs) used during experimental evaluations of Chap. 4, Chap. 5 and Chap. 6. The DNNs aimed at approximating the policy of the agent from the action value function Q_Θ (Eq. 3.16) with parameters Θ.

Input data In all cases, input data to the DNNs are temporal sequences $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_T)$ with length $T = 77$ and for which each data point $\mathbf{x}_{i \in [1,T]}$ is a 5-dimensional array ($P = 5$). In other words they are multivariate time series (MTS) such that $\mathbf{X} \in \mathbb{R}^{5 \times 77}$. We remind that, for reasons of DNNs implementations, when the agent receives the partial sequence $\mathbf{X}_{:t}$, all unknown future data points $(\mathbf{x}_{t+1}, ..., \mathbf{x}_T)$ are replaced by zeros. In this way, the DNN receives fixed size input data.

Output layer In Fig. E.1, the output of the DNN has 8 neurons: 7 output neurons are dedicated to actions of classification for each label in $\mathcal{L}$ (the set of labels $\mathcal{L}$ being composed of 7 distinct labels) and one output neuron is dedicated to the action of delay.

In Fig. E.2, there are two output layers. The output layer in the right branch of the DNN has 4 neurons: 3 output neurons are dedicated to actions of classification for each label in $\tilde{\mathcal{L}}$ (we remind that the dataset has been reduced to three distinct labels instead of the initial seven: $\tilde{\mathcal{L}} \subset \mathcal{L}$) and one output neuron for the action of delay. The output layer in the left branch of the DNN has 3 neurons dedicated to evidence calculation for labels in $\tilde{\mathcal{L}}$.

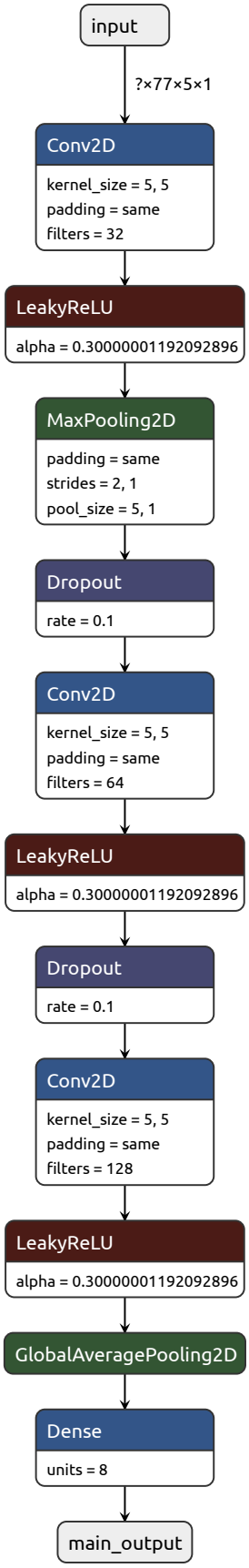

Figure E.1: **Example of a DNN architecture for the policy of the agent.** The DNN computes the Q-values from Eq. 3.17. It has been trained on a set of MTS $\mathbf{X} \in \mathbb{R}^{5\times 77}$ associated to ordinal labels in $\mathcal{L} = \{l_1, \cdots, l_7\}$.

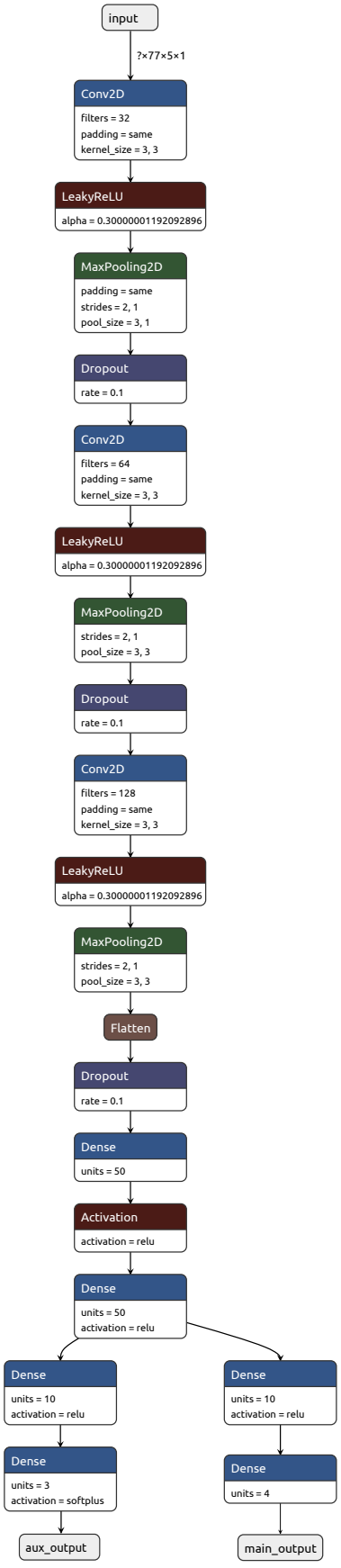

Figure E.2: **Example of a multi-branch DNN architecture for the policy of the agent**. The DNN computes the Q-values from Eq. 3.17 (right output layer) and vector of evidence $\mathbf{e}$ from Eq. 6.6 (left output layer). It has been trained on a dataset involving a set of MTS $\mathbf{X} \in \mathbb{R}^{5\times 77}$ associated to ordinal labels in $\tilde{\mathcal{L}} = \{l_2, l_4, l_7\}$.

Bibliography

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first International Conference on Machine Learning*, page 1.

[2] Aghabozorgi, S., Shirkhorshidi, A. S., and Wah, T. Y. (2015). Time-series clustering–a decade review. *Information Systems*, 53:16–38.

[3] Agrawal, R., Faloutsos, C., and Swami, A. (1993). Efficient similarity search in sequence databases. In *International Conference on Foundations of Data Organization and Algorithms*, pages 69–84. Springer.

[4] Aliakbarian, M. S., Saleh, F. S., Salzmann, M., Fernando, B., Petersson, L., and Andersson, L. (2017). Encouraging lstms to anticipate actions very early. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 280–289.

[5] Alonso González, C. J. and Diez, J. J. R. (2004). Boosting interval-based literals: Variable length and early classification. In *Data Mining in Time Series Databases*, pages 149–171. World Scientific.

[6] Aly, M. (2005). Survey on multiclass classification methods. *Neural Networks*, 19:1–9.

[7] Anderson, H. S., Parrish, N., Tsukida, K., and Gupta, M. R. (2012). Reliable early classification of time series. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 2073–2076.

[8] Asma, N., Cornuéjols, A., and Bondu, A. (2016). A novel algorithm for online classification of time series when delaying decision is costly. In *Conférence francophone sur l'Apprentissage Automatique*.

[9] Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). The uea multivariate time series classification archive, 2018. *arXiv preprint arXiv:1811.00075*.

[10] Bahnsen, A. C., Aouada, D., and Ottersten, B. (2015). Example-dependent cost-sensitive decision trees. *Expert Systems with Applications*, 42(19):6609–6619.

[11] Bakker, B., Zhumatiy, V., Gruener, G., and Schmidhuber, J. (2003). A robot that reinforcement-learns to identify and memorize important previous observations. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 430–435.

[12] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 449–458.

[13] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

[14] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[15] Cardoso, J. S. and Sousa, R. (2011). Measuring the performance of ordinal classification. *International Journal of Pattern Recognition and Artificial Intelligence*, 25(08):1173–1195.

[16] Chan, K.-P. and Fu, A. W.-C. (1999). Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*, pages 126–133. IEEE.

[17] Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542.

[18] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357.

[19] Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307.

[20] Dachraoui, A., Bondu, A., and Cornuéjols, A. (2015). 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*, pages 433–447. Springer.

[21] Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. (2018). The ucr time series archive. *arXiv preprint arXiv:1810.07758*.

[22] Devlin, S. M. and Kudenko, D. (2012). Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, pages 433–440.

[23] Dulac-Arnold, G., Denoyer, L., and Gallinari, P. (2011a). Text classification: A sequential reading approach. In *European Conference on Information Retrieval*, pages 411–423. Springer.

[24] Dulac-Arnold, G., Denoyer, L., Preux, P., and Gallinari, P. (2011b). Datum-wise classification: a sequential approach to sparsity. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 375–390. Springer.

[25] Elkan, C. (2001). The foundations of cost-sensitive learning. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 973–978.

[26] Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, pages 1–47.

[27] Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58.

[28] Frank, E. and Hall, M. (2001). A simple approach to ordinal classification. In *European Conference on Machine Learning*, pages 145–156. Springer.

[29] Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal of Japanese Society For Artificial Intelligence*, 14(771-780):1612.

[30] Fu, T.-c. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181.

[31] Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059.

[32] Galar, M., Fernandez, A., Barrenechea, E., Bustince, H., and Herrera, F. (2011). A review on ensembles for the class imbalance problem: bagging, boosting, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484.

[33] Gamboa, J. C. B. (2017). Deep learning for time-series analysis. *arXiv preprint arXiv:1701.01887*.

[34] Garrett, D., Peterson, D. A., Anderson, C. W., and Thaut, M. H. (2003). Comparison of linear, nonlinear, and feature selection methods for eeg signal classification. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 11(2):141–144.

[35] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm. In *9th International Conference on Artificial Neural Networks*.

[36] Ghalwash, M. F. and Obradovic, Z. (2012). Early classification of multivariate temporal observations by extraction of interpretable shapelets. *BMC bioinformatics*, 13(1):195.

[37] Ghalwash, M. F., Radosavljevic, V., and Obradovic, Z. (2013). Extraction of interpretable multivariate patterns for early diagnostics. In *IEEE 13th International Conference on Data Mining*, pages 201–210.

[38] Ghalwash, M. F., Radosavljevic, V., and Obradovic, Z. (2014). Utilizing temporal patterns for estimating uncertainty in interpretable early decision making. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 402–411.

[39] Ghalwash, M. F., Ramljak, D., and Obradović, Z. (2012). Early classification of multivariate time series using a hybrid hmm/svm model. In *IEEE International Conference on Bioinformatics and Biomedicine*, pages 1–6.

[40] Griffin, M. P. and Moorman, J. R. (2001). Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis. *Pediatrics*, 107(1):97–104.

[41] Hatami, N. and Chira, C. (2013). Classifiers with a reject option for early time-series classification. In *IEEE Symposium on Computational Intelligence and Ensemble Learning*, pages 9–16.

[42] Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[43] He, G., Duan, Y., Peng, R., Jing, X., Qian, T., and Wang, L. (2015). Early classification on multivariate time series. *Neurocomputing*, 149:777–787.

[44] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[45] Huda, A. N. and Taib, S. (2013). Application of infrared thermography for predictive/preventive maintenance of thermal defect in electrical equipment. *Applied Thermal Engineering*, 61(2):220–227.

[46] Jaakkola, T., Singh, S. P., and Jordan, M. I. (1995). Reinforcement learning algorithm for partially observable markov decision problems. In *Advances in Neural Information Processing Systems*, pages 345–352.

[47] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

[48] Janisch, J., Pevnỳ, T., and Lisỳ, V. (2017). Classification with costly features using deep reinforcement learning. *arXiv preprint arXiv:1711.07364*.

[49] Janisch, J., Pevnỳ, T., and Lisỳ, V. (2019). Classification with costly features using deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

[50] Kalpakis, K., Gada, D., and Puttagunta, V. (2001). Distance measures for effective clustering of arima time-series. In *Proceedings of the IEEE International Conference on Data Mining*, pages 273–280.

[51] Kao, H.-C., Tang, K.-F., and Chang, E. Y. (2018). Context-aware symptom checking for disease diagnosis using hierarchical reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[52] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2001a). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286.

[53] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2001b). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Sigmod Record*, 30(2):151–162.

[54] Keogh, E. and Kasetty, S. (2003). On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371.

[55] Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2016). Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*.

[56] Kleist, C. (2015). Time series data mining methods. Master's thesis, Humboldt-Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät.

[57] Längkvist, M., Karlsson, L., and Loutfi, A. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24.

[58] Li, Y. (2017). Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*.

[59] Lin, J., Keogh, E., Wei, L., and Lonardi, S. (2007). Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144.

[60] Lin, Y.-F., Chen, H.-H., Tseng, V. S., and Pei, J. (2015). Reliable early classification on multivariate time series with numerical and categorical attributes. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 199–211. Springer.

[61] Ling, C. X. and Sheng, V. S. (2010). Cost-sensitive learning. *Encyclopedia of Machine Learning*, pages 231–235.

[62] Lkhagva, B., Suzuki, Y., and Kawagoe, K. (2006). Extended sax: Extension of symbolic aggregate approximation for financial time series data representation. *22nd International Conference on Data Engineering Workshops*, 7.

[63] Lotte, F., Congedo, M., Lécuyer, A., Lamarche, F., and Arnaldi, B. (2007). A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 4:R1.

[64] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

[65] Maheshwari, S., Agrawal, J., and Sharma, S. (2011). New approach for classification of highly imbalanced datasets using evolutionary algorithms. *International Journal of Scientific & Engineering Research*, 2(7):1–5.

[66] Mei, J., Liu, M., Wang, Y.-F., and Gao, H. (2015). Learning a mahalanobis distance-based dynamic time warping measure for multivariate time series classification. *IEEE Transactions on Cybernetics*, 46(6):1363–1374.

[67] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937.

[68] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[69] Morales, E. F. and Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioural cloning. In *Proceedings of the twenty-first International Conference on Machine Learning*, page 76.

[70] Mori, U., Mendiburu, A., Dasgupta, S., and Lozano, J. A. (2015). Early classification of time series from a cost minimization point of view. In *Proceedings of the Neural Information Processing Systems Time Series Workshop*.

[71] Mori, U., Mendiburu, A., Dasgupta, S., and Lozano, J. A. (2017a). Early classification of time series by simultaneously optimizing the accuracy and earliness. *IEEE Transactions on Neural Networks and Learning Systems*, 29(10):4569–4578.

[72] Mori, U., Mendiburu, A., Keogh, E., and Lozano, J. A. (2017b). Reliable early classification of time series based on discriminating the classes over time. *Data Mining and Knowledge Discovery*, 31(1):233–263.

[73] Narasimhan, K., Kulkarni, T., and Barzilay, R. (2015). Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*.

[74] Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

[75] Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287.

[76] Nguyen, H. T. and Smeulders, A. (2004). Active learning using pre-clustering. In *Proceedings of the twenty-first International Conference on Machine Learning*, page 79.

[77] Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, 7(1-2):1–179.

[78] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034.

[79] Parrish, N., Anderson, H. S., Gupta, M. R., and Hsiao, D. Y. (2013). Classifying with confidence from incomplete information. *The Journal of Machine Learning Research*, 14(1):3561–3589.

[80] Peng, Y.-S., Tang, K.-F., Lin, H.-T., and Chang, E. (2018). Refuel: Exploring sparse features in deep reinforcement learning for fast disease diagnosis. In *Advances in Neural Information Processing Systems*, pages 7333–7342.

[81] Póczos, B., Abbasi-Yadkori, Y., Szepesvári, C., Greiner, R., and Sturtevant, N. (2009). Learning when to stop thinking and do something! In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 825–832.

[82] Pomerleau, D. A. (1991). Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:88–97.

[83] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[84] Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *International Conference on Machine Learning*, volume 98, pages 463–471.

[85] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[86] Sammut, C. (2010). Behavioral cloning. *Encyclopedia of Machine Learning*, pages 93–97.

[87] Santos, T. and Kern, R. (2016). A literature survey of early time series classification and deep learning. In *The International Conference on Knowledge Technologies and Data-driven Business*.

[88] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

[89] Sensoy, M., Kaplan, L., and Kandemir, M. (2018). Evidential deep learning to quantify classification uncertainty. In *Advances in Neural Information Processing Systems*, pages 3179–3189.

[90] Settles, B. (2009). Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

[91] Sharifzadeh, S., Chiotellis, I., Triebel, R., and Cremers, D. (2016). Learning to drive using inverse reinforcement learning and deep q-networks. *arXiv preprint arXiv:1612.03653*.

[92] Sheng, V. S. and Ling, C. X. (2006). Thresholding for making classifiers cost-sensitive. In *AAAI*, pages 476–481.

[93] Shridhar, K., Laumann, F., and Liwicki, M. (2018). Uncertainty estimations by softplus normalization in bayesian convolutional neural networks with variational inference. *arXiv preprint arXiv:1806.05978*.

[94] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959.

[95] Sokolova, M. and Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437.

[96] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[97] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT Press Cambridge.

[98] Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1057–1063.

[99] Tenorio-Gonzalez, A. C., Morales, E. F., and Villaseñor-Pineda, L. (2010). Dynamic reward shaping: training a robot by voice. In *Ibero-American Conference on Artificial Intelligence*, pages 483–492. Springer.

[100] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[101] Wang, W., Chen, C., Wang, W., Rai, P., and Carin, L. (2016). Earliness-aware deep convolutional networks for early time series classification. *arXiv preprint arXiv:1611.04578*.

[102] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

[103] Wang, Z., Yan, W., and Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *International Joint Conference on Neural Networks*, pages 1578–1585. IEEE.

[104] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3-4):279–292.

[105] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256.

[106] Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

[107] Xing, Z., Pei, J., Dong, G., and Yu, P. S. (2008). Mining sequence classifiers for early prediction. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 644–655.

[108] Xing, Z., Pei, J., and Keogh, E. (2010). A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48.

[109] Xing, Z., Pei, J., and Philip, S. Y. (2009). Early prediction on time series: a nearest neighbor approach. In *Twenty-First International Joint Conference on Artificial Intelligence*.

[110] Xing, Z., Pei, J., Yu, P. S., and Wang, K. (2011). Extracting interpretable features for early classification on time series. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 247–258.

[111] Yang, J., Nguyen, M. N., San, P. P., Li, X. L., and Krishnaswamy, S. (2015). Deep convolutional neural networks on multichannel time series for human activity recognition. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

[112] Yao, L., Li, Y., Li, Y., Zhang, H., Huai, M., Gao, J., and Zhang, A. (2019). Dtec: Distance transformation based early time series classification. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 486–494.

[113] Ye, L. and Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 947–956.

[114] Yu, K., Liu, Y., Schwing, A. G., and Peng, J. (2018). Fast and accurate text classification: Skimming, rereading and early stopping. In *International Conference on Learning Representations*.

[115] Zadrozny, B., Langford, J., and Abe, N. (2003). Cost-sensitive learning by cost-proportionate example weighting. In *IEEE International Conference on Data Mining*, volume 3, page 435.

[116] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. (2016). Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929.

[117] Zhou, Z.-H. (2017). A brief introduction to weakly supervised learning. *National Science Review*, 5(1):44–53.

[118] Zhu, X. J. (2005). Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.