UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

END-TO-END NEURO-SYMBOLIC APPROACHES FOR EVENT RECOGNITION

Ph.D. candidate:

Gianluca Apriceno

Advisor:

Prof. Luciano Serafini

Fondazione Bruno Kessler

Co-Advisor:

Prof. Andrea Passerini

University of Trento

XXXV°cycle

Abstract

Event detection is a critical challenge in many fields like video surveillance, social graph analysis, and multimedia processing. Furthermore, events are "structured" objects involving multiple components like the event type, the participants with their roles, and the atomic events in which it decomposes. Therefore, the recognition of an event is not only limited to recognize the type of the event and when it happened, but it involves solving a set of simple tasks. Exploiting background knowledge about events and their relations could then be beneficial for event detection. In the last years, neuro-symbolic integration has been proposed to merge the strengths and overcome the drawbacks of both symbolic and neural worlds. As a consequence, different neuro-symbolic frameworks, which combine low-level perception of neural networks with a symbolic layer, encoding prior domain knowledge (usually defined in terms of logical rules), have been applied to solve different atemporal tasks. In this thesis, we want to investigate the application of the neuro-symbolic paradigm for event detection. This would also provide a better insight into the strengths and limitations of neuro-symbolic towards tasks involving time.

Keywords

[event recognition, structured events, neuro-symbolic integration]

Contents

List of Tables

List of Figures

Acknowledgement

It seems to me like yesterday, when I decided to set sail for the PhD adventure. Like any adventure, I had to face many unknowns and difficult situations, but fortunately I was not alone. Indeed, these three years have given to me the possibility to meet extraordinary people that inspired and gave me the motivation to bring out the best in myself. Therefore, I would like to use the next few lines to explicitly express my gratitude to all those people that accompany me during my adventure. First, I want to say thanks to my advisor Luciano Serafini and my co-advisor Andrea Passerini that guide, help and support me from the beginning. Then, from the Fondazione Bruno Kessler, I want to thanks my friends: Tommaso Campari, Alessandro, Sagar, Tommaso Carraro, Leonardo, Loris, Mauro, Patrizio, Gabriele, Davide, Eleonora, Renan, Francesa and Andrei. From the University of Trento, I want to thanks: Antonio, Giovanni, Emanuele, Burcu, Stefano, Andrea, Steve, Francesco, Cesare, Marco, Debodeep, Gianni, Luca and Paolo. I would also thanks my universitymates, housemates, the top players guys and the guys from the ESN (list you all would require me an entire chapter ;)).

Last but not least, I want to thanks my parents who always support and have been closed to me in every moment.

If I am here now, it is thanks to all of you

Gianluca

List of Symbols

Calculus

$\int_a^b f(x)dx$	definite integral of from a to b of f with respect of x
$\frac{\partial f}{\partial x}$	partial derivative of f with respect to x
$\nabla_{\boldsymbol{w}} z$	the gradient of z with respect to $\boldsymbol{w}$

Functions and operators

$f(\cdot)$	a function
$\sum$	summation over a collection of elements
$\prod$	product over a collection of elements

Numerical objects

x	a scalar
$\boldsymbol{x}$	a vector
X	a matrix
$X[i,j]$	the entry of matrix X located at row i and column j
$\boldsymbol{X} = \{x_i\}_{i=1}^{k}$	a sequence of k elements

Probability theory

X	a random variable
P	a probability distribution
$P(X = q)$	the probability that random variable X takes value of q

Set theory

$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^+$	the set of positive numbers
$\mathbb{R}^{n\times m}$	the set of real matrices having n rows and m columns
$A \cup B$	union of sets A and B
$A \cap B$	intersection of sets A and B
$\lvert A\rvert$	cardinality of set A
$A \times B$	cartesian product of sets A and B

Chapter 1

Introduction

What is Artificial Intelligence?

> *"It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable"*
>
> —John McCarthy, *What is Artificial Intelligence?*

This is what the winner of the Turing Award (1971) John McCarthy said in the paper *"What is Artificial Intelligence"* [62] where he first coined and defined the term Artificial Intelligence (AI). Some of the readers may disagree by stating that the true precursor of the term is Alan Turing. Indeed, Turing, which also referred as the "father of computer science", published in 1950 an article called *"Computing Machinery and Intelligence"* [83] where he tried to answer the question '*can machines think?*'. To answer the question, he proposed the well-known Turing Test to test a machine's ability to exhibit intelligent behaviour equivalent to or indistinguishable from that of a human. According to Adrienne Mayor, research scholar, folklorist, and science historian at Stanford University, artificial intelligence dates back to ancient myths [61]. She sees Hephaestus, the Greek god of invention and blacksmithing, as the precursor of modern AI. Indeed, many of its creations can be seen as prototypes of modern AI products. No matter what the origin of the term AI is, nowadays, the results of AI are tangible and have radically changed our everyday life. AI products like personal assistants (e.g., Amazon Alexa, Siri and Google), and self-driving cars (e.g, Tesla and Waymo) have not only changed the way we interact with each other but with the surrounding environment too. All of these are examples of what is called weak AI (aka narrow AI), where an AI system is trained to solve specific tasks. This is still far from the more ambitious objective to develop an AI having an intelligence equal to that of a human (Artificial General Intelligence) or even able to exceed it (Artificial

Super Intelligence). Furthermore, looking deepen at the results of weak AI, it is worth observing that all the results have been achieved thanks to the availability of a huge amount of data over which AI models (mostly neural networks) have been trained on. This can be seen as a kind of gap for these models with respect to humans. Indeed, originally, neural networks have been proposed as a way to replicate the complex cognitive ability of the human brain. Therefore, keeping in mind this goal, one can see that it is still too far for current neural networks. Indeed, we as humans are able to solve tasks with only a few examples and have the capability to revise/create existing/new knowledge by reasoning over it. As a consequence, a new research paradigm has emerged in the field of AI: Neuro-Symbolic Artificial Intelligence (NeSy AI). NeSy AI (aka neuro-symbolic integration[1]) [39] is a subfield of AI that combines learning and reasoning by merging together neural networks and symbolic AI. The promise of NeSy AI is that by merging these two paradigms, we are able to obtain a system that combines the strengths of the two worlds. From the neural side, this involves having trainable systems over raw data and robust against noise, while for symbolic AI, the possibility to have an explainable system where expert knowledge can be exploited at different levels. Different ”general purpose” neuro-symbolic frameworks have been proposed in the literature [18, 78, 59, 56, 60, 92, 87] and have been shown to be successful to solve not only classical machine learning tasks like (multi-class) classification but also more complex tasks like Semantic Image Interpretation [28]. In all those frameworks, not so much attention has been devoted to tasks involving time, but time plays a key role in our everyday life. The actions we perform every day and the order in which we do them are based on decisions that keep into account different aspects of time, like actions' durations (e.g, a person is late for work, so he is going to take his car instead of the bus) or common sense temporal domain knowledge (e.g, to arrive on time at work, a person has to wake up early). When talking about temporal tasks, one of the main challenging tasks is event detection (ED). Indeed, ED from sequences of data is a critical challenge in various fields, including surveillance [20], multimedia processing [89, 54], and social network analysis [21]. Furthermore, events are structured objects that involve different components (e.g, the participants, their roles, the atomic events, etc.). Integrating commonsense and structural knowledge about events and their relationships can significantly enhance machine learning methods for ED. For example, in analyzing a park security video, the (temporal) knowledge of a person moving towards a bench, leaving a bag close to it, and then going away can aid event detection.

In this thesis, we investigate neuro-symbolic integration in the context of event recognition[2]. In particular, we tried to address the following research questions:

[1] in the manuscript, we are going to use these two terms interchangeably

[2] event detection and event recognition are used interchangeably

R1: How can we formalize and what kind of formalism can we use to define the event recognition problem in a neuro-symbolic context?

R2: What kind of supervision and background knowledge can we use/exploit in an end-to-end neuro-symbolic approach for event recognition?

R3: What are the issues emerging in those approaches (e.g., scalability)? How can we solve/mitigate them?

R4: Can we ”readapt” these approaches to solve lower tasks with respect to event recognition? For example, using these approaches to predict sequences of values, which have to be compliant with the background knowledge, that may be used to build an event recognition system (e.g, triggering an event warning when certain values are identified in a sequence).

1.1 Contributions

This thesis focuses on the application of the neuro-symbolic paradigm in the context of event recognition with the objective to answer to the aforementioned research questions. In line with this objective, my research has led to two (connected) directions:

- **Direction 1:** The integration of the temporal dimension (oriented to event recognition) into existing neuro-symbolic frameworks.
- **Direction 2:** Proposing a novel (temporal) neuro-symbolic approach for event recognition.

These two directions has led to the following publications:

- Apriceno, Gianluca, Andrea Passerini, and Luciano Serafini. *“A neuro-symbolic approach to structured event recognition.”* 28th International Symposium on Temporal Representation and Reasoning (TIME 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
- Apriceno, Gianluca, Andrea Passerini, and Luciano Serafini. *“A Neuro-Symbolic Approach for Real-World Event Recognition from Weak Supervision.”* 29th International Symposium on Temporal Representation and Reasoning (TIME 2022). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.
- Badreddine, Samy, Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. *“Interval Logic Tensor Networks.”* arXiv preprint arXiv:2303.17892 (2023).

- Apriceno Gianluca, Luca Erculiani, and Andrea Passerini. *"A Neuro-Symbolic approach for Non Intrusive Load Monitoring"*(currently under review at PAIS23)

1.2 Outline of the Thesis

The thesis is structured as follows:

Chapter 2:

The objective of this chapter is to provide the reader with the basic background concepts related to neural networks, symbolic AI, and NeSy AI. For the part related to neural networks, we briefly review their connection with the human brain, introduce their main components (i.e, neuron and activation functions) and focus on the three standard neural architectures which are used in almost every modern architecture, respectively, Feed Forward Neural Network, Convolution Neural Network, and Recurrent Neural Network. For symbolic AI, we focus on logic programming and define its main building blocks which contribute to defining what a logic programming is. For NeSy AI, we introduce the well-known neural extensions of logic programming, DeepProbLog, and the neuro-symbolic framework Logic Tensor Network. This part will be useful to better understand chapter 3 and chapter 5. We want to highlight that this chapter is not intended to be exhaustive, and therefore, for further details, the interested reader is referred to the references pointed out in the chapter.

Chapter 3:

In this chapter, we provide an explicit formalization for the event recogntion problem where events divides in structured and atomic events. Then, we present a neuro-symbolic prototype that we developed using the DeepProbLog framework and compare it with a (recurrent) neural baseline. The comparison has been evaluated on the recognition of synthetic events generated by our mnist-digit event video simulator. The results show the higher ability of our neuro-symbolic approach to recognize event as well as its ability to generalize to unseen outcomes with respect to the fully neural baseline.

Chapter 4:

In this chapter, we propose a neuro-symbolic approach for Temporal Event Detection in a real world scenario involving sport activities. We show how by incorporating simple knowledge involving the relative order of atomic events and constraints on their duration, the approach substantially outperforms a fully neural solution in terms of recognition accuracy, when little or even no supervision is available on the atomic events.

Chapter 5:

In this chapter, in line with the first research direction, we propose interval real logic a temporal (interval based) extension of real logic. The language of interval real logic allows to express temporal properties and relations about (trapezoidal) event and is interpreted over finite sequence. The implementation of interval real logic has been done through Logic Tensor Network and the overall framework has been tested over four synthetic tasks that require reasoning about events to predict their fuzzy durations. Our results show that the system is capable of making events compliant with background temporal knowledge.

Chapter 6:

In this chapter, we show how background knowledge can also be exploited to solve lower tasks on top of which event recognition approaches can be built. Briefly, the task we solve consists in disaggregating the total energy consumption of an house in the consumption of the individual appliances that compose it (e.g., kettle, microwave etc.). For this purpose, we readapt the approach presented in chapter 4, and devise a neuro-symbolic approach where background knowledge about the behaviour of the appliance in terms of its consumption over time is used to refine the prediction of the neural network representing the given appliance.

Chapter 7:

In this chapter, conclusions are drawn and directions for future works are briefly discussed.

Chapter 2

Background

In this chapter, we are going to cover the main concepts related to three topics: neural networks, symbolic AI, and neuro-symbolic integration. In addition, since in chapters 3 and 4, we formalize the event recognition problem in a neuro-symbolic context using an event calculus inspired formalization, we also briefly introduce the event calculus formalism. The overall goal of this chapter consists in providing to the reader the basic concepts/notions about the aforementioned topics in order to facilitate the understanding of the following chapters. As we have already mentioned, this chapter is not intended to be exhaustive, and then, for further details, the interested reader is referred to the references pointed out in the chapter.

2.1 Neural Networks

Artificial neural networks[52], commonly referred to as neural networks, have been proposed as an attempt to mimic the complex reasoning of the human brain. Nevertheless, it is highly recognized that the capability of the human brain is still too far by neural networks. This difference is not only in terms of neurons (billions of neurons compared to thousands or millions) but due to the lack of adaptability/flexibility and generalization which are characteristic of the human brain. Indeed, the human brain is able to adapt more quickly and generate new ideas in a more efficient way with respect to neural networks. Furthermore, while the learning process of the human brain lasts its entire life, the learning process of neural networks is a kind of "temporal" learning over a limited amount of data. Despite these differences, neural networks have been shown to achieve high accuracy in solving specific tasks.

2.1.1 Neuron

The human brain is one of the most complex organs of a human. Indeed, it is responsible for controlling many aspects of our everyday life like vision, memory, emotions, etc. From a computer scientist's point of view, we can see the brain as acting like a computer, it processes information coming from the senses and the body and sends messages back to the body.

The principal processing unit of the brain is a cell called neuron that propagates or transforms the incoming signal into an effector (e.g. feelings, movements). A more concise and schematic view of a neuron is depicted in Figure 2.1. As can be seen, a neuron is composed of four main parts:

- A set of weighted connections, also referred to as weights, each of which weighs the input signal $\boldsymbol{x}$ and connects it to the neuron k. In details, if $x_j \in \boldsymbol{x}$, its corresponding weight will be w_{kj}.
- An adder that sums up the input signal $\boldsymbol{x}$ with its corresponding weights.
- A bias b_k, an external (positive or negative) parameter, that applies an affine transformation to the output of the adder.
- An activation function (aka squashing function) that limits the amplitude of the output signal into a specific range.

From a mathematical point of view, we can describe the neuron of Figure 2.1 by the following equations:

$$z_k = \sum_{i=1}^{n} w_{ik} x_i + b_k \tag{2.1}$$

$$o_k = \phi(z_k + b_k) \tag{2.2}$$

where z_k is the pre-activation of the neuron (i.e., it is the result of the sum of the adder and the bias), ϕ is the activation function, and o_k is the output of the neuron, obtained by applying ϕ to z_k. Furthermore, we can avoid to explicitly mentioning b_k by including a weight $w_0 = 1$ and an input $x_0 = b_k$. Therefore, Equations 2.1 and 2.2, can be rewritten as:

$$z_k = \sum_{i=0}^{n} w_{ik} x_i + b_k \tag{2.3}$$

$$o_k = \phi(z_k) \tag{2.4}$$

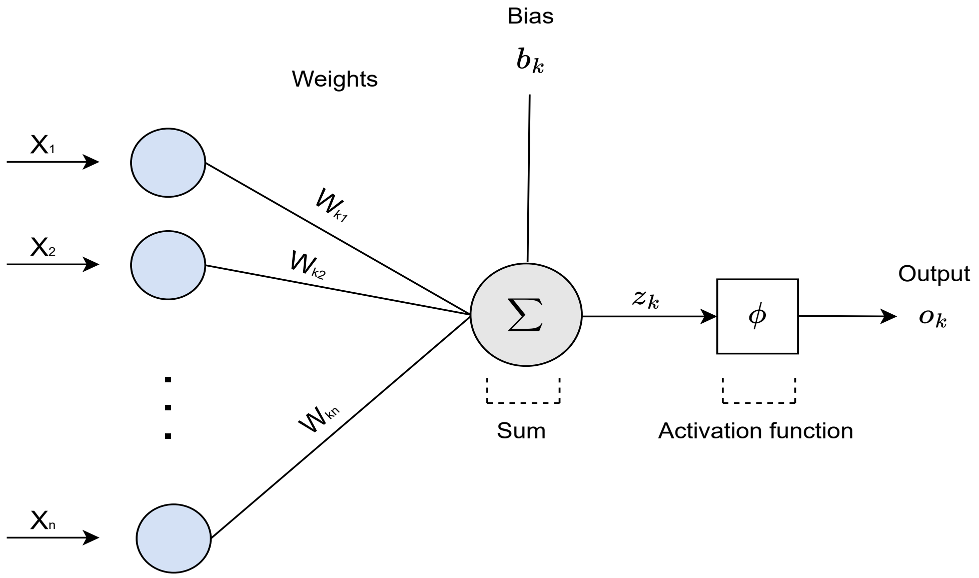


Figure 2.1: Neuron' structure

2.1.2 Activation Functions

Roughly speaking, the role of an activation function consists in determining if a neuron should be activated or not. The most common activation functions are:

- Threshold:

$$f(x) = \begin{cases} 1, & \text{if } x \geq t \\ 0, & \text{if } x < t \end{cases} \tag{2.5}$$

  The threshold function can be seen as a step function (Figure 2.2a). If the input x (i.e. the preactivation) is greater than a given threshold t, the neuron fires otherwise the incoming signal is not propagated any further. Even if it is simple, this activation function is not used in modern architecture since its gradient is zero and therefore no learning is going to occur.

- Sigmoid[1]:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

[1] used in 3.7.1, 4.3 and 5.5.1

As can be seen by looking at the Figure 2.2b, the sigmoid function is a soft version of the threshold function. This function squashes the input signal in the range [0,1] and, due to its differentiability, it allows to perform training. The drawback of this function is that, in cases of many layers, vanishing gradient problem may rise. Furthermore, training using sigmoid activation function is usually slow.

- ReLU[2]:

$$f(x) = \max(0, x) \tag{2.7}$$

ReLU, which stands for Rectified Linear Unit, is one of the most used activation functions (Figure 2.2c). It solved the vanishing gradient problem and ensure faster gradient convergence [68].

$$f(x) = \max(\alpha x, x) \tag{2.8}$$

As can be seen, by introducing the parameter α, the learning is going to occur, even for negative values.

- Softplus [3]:

$$f(x) = \log(1 + e^x) \tag{2.9}$$

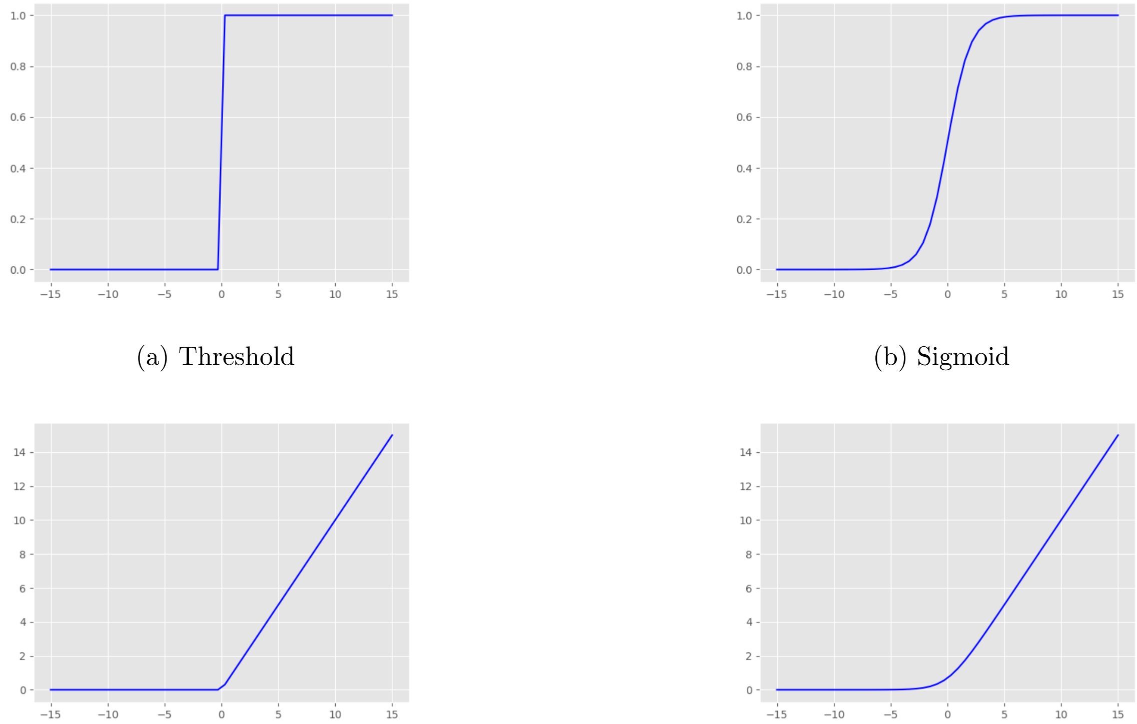
The Softplus function is usually used to replace the ReLU function (Figure 2.2d). Indeed, it can be seen as a smooth version of ReLU where the "knee" of ReLU is removed.

2.1.3 Neural architectures

In [33], McCulloch and Pitts proposed a neuron-based theoretical model of the nervous system. The model is simple: it is a network of neurons where each neuron is connected to the neighborhood neurons, through synapses. Even if it is simple, the model is able to perform extremely complex computations.

[2] used in 3.7.1

[3] used in 5.5.2



(a) Threshold

(b) Sigmoid

(c) ReLU

(d) Softplus

Figure 2.2: Activation functions

Fully-connected neural network

As can be seen by looking at Figure 2.3, a fully-connected neural network (FCN)[4] is a feed-forward network that is composed of three kinds of layers:

- One input layer
- One or more hidden layer
- One output layer

The input layer, which is composed of a set of source nodes, projects the input signal $\boldsymbol{x} \in \mathbb{R}^n$ onto the hidden layers. The hidden layers are the processing layers where the computation (i.e., the transformation of the input signal) is performed. Usually, more than one hidden layer is used in this kind of architecture, with each hidden layer having the same or a different number of neurons and adopting one of the activation functions presented in 2.1.2. This is done in order to learn (i.e. approximate) a more complex

[4] used in: 3.7.1

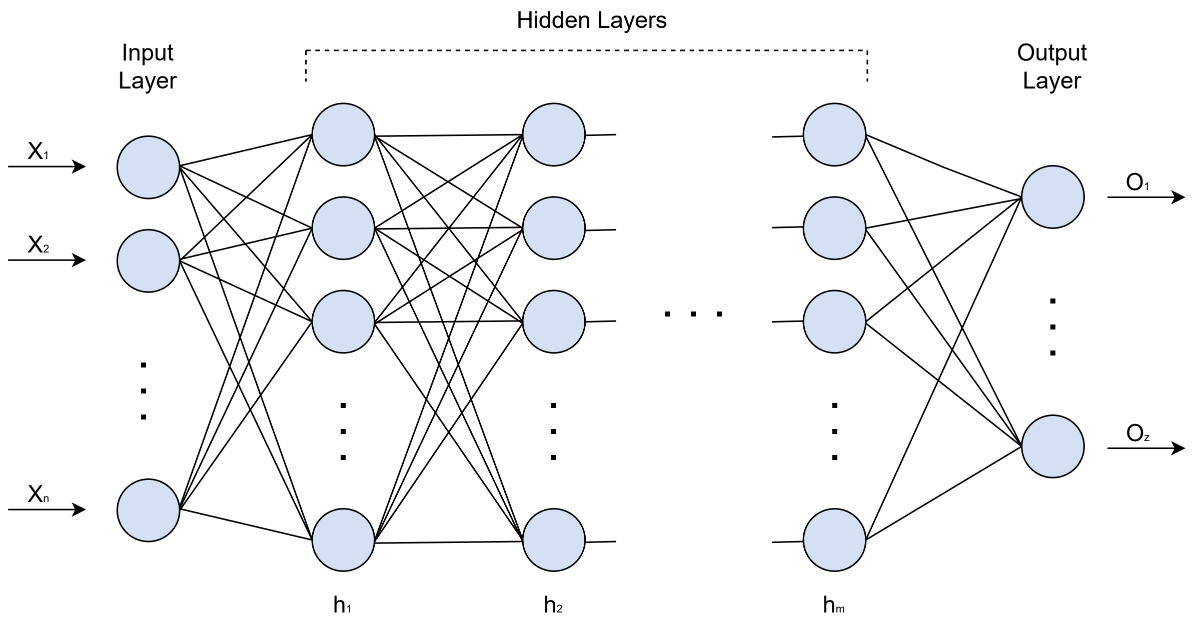


Figure 2.3: Fully-connected nueral network

function. Finally, the output layer returns the final results of the network (i.e., the result of the computation of the hidden layers). This kind of architecture is called "fully-connected" because every neuron in a layer is connected to every neuron in the next adjacent layer.

Convolutional Neural Network

When the input signal has a shape different than a vector, adopting a FCN may not be the right choice. For example, if the input is a grayscale image of size $W \times H \times 1$, where W and H denote the width and the height of the image, respectively (1 is the number of color channels), reshaping it to a vector would definitely determine the loosing of spatial information coming from the neighborhood pixels. Furthermore, due to the interconnectivity of a FCN, the number of parameters to learn will be quite high, with an impact on the computational cost. To overcome this issue, convolutional neural networks (CNNs) [5] have been proposed. As can be seen by looking at Figure 2.4, together with fully connected layers, a CNN has two additional layers:

- A convolutional layer
- A pooling layer

[5] used in: 3.7.1

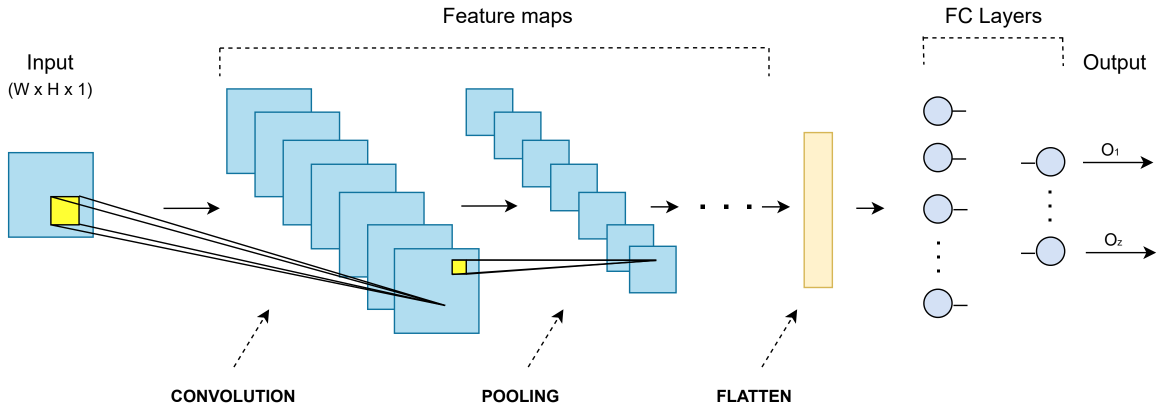


Figure 2.4: Convolutional neural network

A convolution layer is composed of a set of learnable filters (also referred to as kernels) that slide over an image. Roughly speaking, a filter is a matrix composed of learnable weights whose aim consists of learning features about the incoming (processed signal). These features are referred to as features/activation maps and encode useful information (like edge, color, etc.). When a CNN is composed of more than one convolutional layer, the idea is that the first layers usually learn more general features compared to the deep layers. In addition, adopting a convolutional layer with respect of FC layer would be cheaper in terms of learnable parameters. Generally, between two convolutional layers is common to insert a pooling layer. This layer is used to compress the incoming information in order to reduce the number of learnable parameters and save model computation. Examples of pooling layers include: max, average, and global max pooling (for more details on both convolution and pooling layers see Convolutional Neural Networks for Visual Recognition)

Recurrent Neural Networks

When dealing with sequential data, both FCNs and CNNs will not achieve great performance. This is mostly due to the fact that these approaches would make a prediction that only depends on the current input, without keeping into account the temporal dependency of the sequence. In order to model this kind of dependency, Recurrent Neural Networks have been proposed (RNNs). In simple words, RNNs, which have their roots in Elman networks [31], are a class of neural networks where the output from the previous step is fed as input to the current step (Figure 2.5). For each timestamp t, the temporal hidden representation h_t and the output o_t are expressed as follows:

$$
\begin{aligned}
h_t &= \phi_1(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \\
o_t &= \phi_2(W_{oh}h_t + b_o)
\end{aligned}
$$

where $W_{hx}, W_{hh}, W_{oh}, h_t, b_h, b_o$ are shared learnable weights, and ϕ_1, ϕ_2 are activation functions. Using a RNN has different advantages. First, as we have already stated, it captures temporal input correlations between timestamps. Second, it takes as input sequence of arbitrary lengths. Lastly, similar to CNNs, parameters are shared across time. One of the main issues of RNNs is their difficulty to capture long-term dependencies. This is due to the so-called *vanishing gradient* that appears during the training of an RNN. Intuitively, this is due to the fact that, during the backpropagation, the gradient will get smaller and smaller (almost zero) when approaching the initial layers. As a consequence, no update will occur for the weights of these layers. In order to overcome this issue, specific gates have been proposed, leading to an "evolution" of the standard RNNs that is called Long Term Short Memory [40][6]. These gates are the following:

- **Input gate** ($\mathcal{I}_g$): a gate that is used to quantify the information carried by the input at time t (i.e, x_t).
- **Forget gate** ($\mathcal{F}_g$): a gate that decides if we should retain or not the information at the previous timestamp.
- **Output gate** ($\mathcal{O}_g$): a gate that decides how much (final) information of the current cell we should return

and each of them realizes the following transformation:

$$G_g = \phi_s(W_{G_g} x_t + U_{G_g} h_{t-1} + b_{G_g}) \quad G \in \{F, U, O\} \tag{2.10}$$

where ϕ_s is the sigmoid function and $W_{G_g}, U_{G_g}, b_{G_g}$ are learnable weights specific for each gate.

2.2 Symbolic AI

At the beginning of AI, symbolic AI (aka symbolic reasoning) has been the dominant paradigm in AI. One of the reasons, it is due to the ability of symbolic approaches to encode and reason about knowledge in an explicit way, close to humans. In addition, symbolic approaches have been also shown to be:

- expressive
- easier to explain
- generalize from a few examples

[6] used in:3.7.2

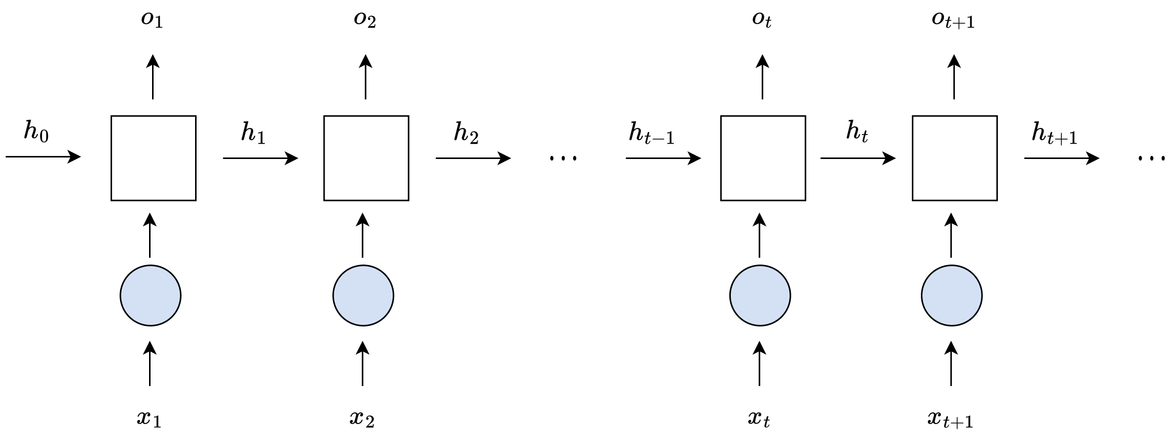


Figure 2.5: Recurrent Neural Network

In particular, logic programming, which encodes knowledge in terms of facts and rules, has been applied to solve effectively many different complex tasks. Furthermore, since its introduction, many families of logic programming languages have emerged, like Prolog, Answer Set Programming, Datalog, etc. In what follows, we are going to provide a general overview on logic programming, and we will see later how it has been combined with neural networks to create hybrid reasoning systems.

2.2.1 Logic Programming

Logic Programming, whose origin can be rooted in the debates between procedural and declarative knowledge representations, is a declarative programming paradigm where one only encodes the information about the domain and the goal to achieve without providing any details on "how” to reach it. This is in contrast with the imperative (and perhaps more known) programming paradigm where detailed instructions are provided to the system in order to achieve the final goal. To make an example, consider a robot in a flat that has to reach a given room. From a declarative perspective, the robot may have access to a total/partial map of the flat, knowing only its start position and the target destination. In this case, the robot has to choose the actions to perform in order to reach the destination. While, in the imperative paradigm, the actions that the robot has to execute are established a priori.

Logic concepts

Before going into the technical details and providing a more formal definition of what logic programming is, we briefly review the main concepts related to logic.

Suppose to have a domain $\mathcal{D}$. $\mathcal{D}$ can be described by a tuple $(\mathcal{C}, \mathcal{F}, \mathcal{P})$ where:

- $\mathcal{C}$ is a set of constant symbols
- $\mathcal{F}$ is a set of functors symbols
- $\mathcal{P}$ is a set of predicate symbols

Constants of $\mathcal{C}$ are used to represent individuals of $\mathcal{D}$. Functors symbols are used to refer to individuals for which exists a sequential connection with other individuals of $\mathcal{D}$. To make an example, suppose to have the individuals Abraham, Homer and Bart denoted respectively with constants a, h, b and the functor *fatherOf* accepting only one argument. Further, suppose that Abraham is the father of Homer and Homer is the father of Bart. *fatherOf(b)* refers to the constant h (i.e. Homer). If we apply the same functor a second time (*fatherOf(fatherOf(b))*), we refer to the constant a (i.e. Abraham). Predicates symbols are used to express relations among individuals (e.g., relation friends). Both predicates and functors are defined by indicating the number of arguments that they take as input (i.e. their arity). A generic individual of $\mathcal{D}$ is represented by a variable. Constants, functors, and predicates are represented using an initial lowercase letter, while variables are represented using an initial capital letter. A term t can be built from a constant, a variable, or the recursive application of functors. The input of predicates is terms and predicates applied to tuple of terms are referred to as atomic formulas or atoms. A ground term is a term that contains no variables, and a ground atom is an atom that contains no variables. A literal is an atom or its negation. A clause is a disjunction (logical or) of literals. A summary of the aforementioned concepts is given in Figure 2.6.

Logic program

A logic program L_p is composed of a set of clauses with at most one positive literal (aka Horn clauses[7]), having the following form:

$$h \leftarrow b_1,\ b_2,\ \dots\ b_n.$$

In the clause above[8], h and $b_1,\ b_2,\ \dots\ b_n$ are atoms that are referred as the head and the body of the clause, respectively. All clauses are assumed to be implicitly universally quantified.

In logic programming language Prolog [73], clauses can be classified in:

- Facts

[7] usually may be extended with negation in the body (negation as a failure).

[8] , stands for $\wedge$ and this a disjunction since $h \leftarrow b_1,\ b_2,\ \dots\ b_n \equiv h \vee \neg b_1 \vee \neg b_2 \vee \cdots \vee \neg b_n$

$$\begin{array}{lll}
& \mathcal{C} = \{j, k, z, \dots\} & \\
& \mathcal{V} = \{X, Y, Z, \dots\} & \\
& \mathcal{F} = \{f |\ f : \mathcal{C}^{\alpha} \rightarrow \mathcal{C}\}\ \alpha = 1, \dots, n & \\
& \mathcal{P} = \{p |\ p : \mathcal{C}^{\alpha} \rightarrow [0,1]\}\ \alpha = 1, \dots, n & \\
If : & & \\
& j \in \mathcal{C},\ \ X \in \mathcal{V} & \\
& f_1 \in \mathcal{F}\ \ and\ \ f_1 : \mathcal{C} \rightarrow \mathcal{C} & \\
& p_1 \in \mathcal{P}\ \ and\ \ p_1 : \mathcal{C} \rightarrow [0,1] & \\
Have : & & \\
& \mathcal{T} = \{j, X, f_1(j), f_1(X), \dots\} & G_T = \{j, f_1(j), \dots\} \\
& \mathcal{A} = \{p_1(j), p_1(X), \dots\} & G_A = \{p_1(j), p_1(k), \dots\} \\
& L = \{a_1, \dots, a_n\} & with\ \ a_i \in A\ \ and\ \ i = 1, \dots, n \\
& \mathcal{C}_l = \{c_1, \dots, c_m\} & with\ c_i = \bigvee_{j\,=\,1}^{n_j} a_{ij}\ \ and\ \ i = 1, \dots, m
\end{array}$$

Figure 2.6: Logic programming concepts: $\mathcal{V}$ is the set of variables, $\mathcal{T}$ is the set of terms, G_T is the set of ground terms, A is the set of atoms, G_A is the set of ground atoms, L is the set of literals and $\mathcal{C}_l$ is the set of clauses (α and n_j indicate respectively the arity of a functor/predicate and the number of literals of a clause c_i).

- Rules
- Queries

Facts (F), which are clauses with an empty body, are used to express unconditionally true statements on both objects and their relations. Rules (R), which are standard Horn clauses, are used to inject prior knowledge on both objects and their relations. Finally, queries, which are clauses with an empty head, are used to ask questions on both objects and their relations. Determine if a query q is logically entailed by L_p ($L_p \models q$) means using both F and R to find a set of inference steps leading to the derivation of q.

2.2.2 Event Calculus

Event Calculus (EC) is a logic programming formalism introduced by Kowalski and Sergot [51] in 1986 to represent and reasoning about events and their effects. The main components of EC are:

EC predicates	
happensAt(e, t)	Event e happens at time t
initially(f)	Fluent f holds from time 0
initiatedAt(e, f, t)	Fluent f starts to hold after event e at time t
holdsAt(f, t)	Fluent f holds at time t
terminatedAt(e, f, t)	Fluent f ceases to hold after event e at time t
clipped(t_1, f, t_2)	Fluent f is terminated between time t_1 and t_2
$t_1 < t_2$	Time point t_1 is before time point t_2

Table 2.1: Example of some EC predicates.

- **Events** also referred to as actions, include both concrete (e.g, Mary turns on the light) and abstract (e.g, sale fall off) events.
- **Fluents**: properties whose value may change over time. like numerical values (e.g. number of working hours) or propositions (e.g. is cold)
- **Time**: a discrete or continuous value used to define the time in which an event occurs or a fluent initiates/holds/terminates.

In the original formalization of EC, events happen at specific time and their occurrence causes the initiation (or termination) of a period of time in which fluent/s hold (or held). These facts are expressed using some well-defined predicates (some of those are reported in Table 2.1) in term of an EC narrative of facts. These predicates are then connected together to define both domain independent and dependent rules.
Different dialects to the original EC have been proposed in the literature [79]. In chapter 2, we will provide an event calculus inspired formalization in order to define the event recognition problem in a neuro-symbolic context. In chapter 4, we will provide a slightly modify version of the formalism introduced in chapter 3, by defining the distinction of events in structured and atomic events.

2.3 Neuro-symbolic integration

Neuro-symbolic integration (NeSy) is a subfield of AI that combines neural and symbolic approaches. The growing interest in NeSy AI is motivated by different reasons. One of the reasons, from cognitive science, is that neural and symbolic approaches represent two different abstractions, respectively. As we have already seen in Section 2.1, neural approaches based on neural networks represent an abstraction of the working of the human brain. On the other side, symbolic approaches, based on the representation of knowledge by formal languages, can be seen as an abstraction of the human reason mechanism. Therefore, it appears natural to ask itself how these two abstractions can be unified together. Another reason is that with the integration of neural and symbolic approaches, it is possible to merge the strengths and overcome the issues of the two worlds. For example, neural networks are robust to noise with respect to symbolic approaches but require a huge amount of data and are often referred to as black box models. On the other hand, symbolic approaches are self-explanatory and require less amount of training data. Looking at the literature, different neuro-symbolic frameworks that rely on explicit encoding of knowledge as first order formulas have been proposed over the last years like Logic Tensor Network [78], Lyrics [59], DeepProbLog[56], and NeurASP [92]. Among these, DeepProbLog and LTN which relies, respectively, on probabilistic reasoning and fuzzy logic have been proven to be successful in many different tasks. In what follows, we are going to provide an overview on both DeepProbLog and LTN, which will be useful later to understand chapters 3 and 5.

2.3.1 DeepProbLog: combining probabilistic logic programming with neural networks

The difference between logic programming Prolog and probabilistic logic programming ProbLog is that in ProbLog each fact $f \in F$ is labeled with a probability p that states that f is true with probability p (respectively false $1 - p$). By assuming that each f is mapped to a boolean random variable and that these variables are independent each other, it is possible to calculate the probability of a $L' \subseteq L$ (also referred to as world) as:

$$P(L') = \prod_{f_i \in L'} p_i \prod_{f_i \notin L'} 1 - p_i$$

Therefore, a probability distribution is defined over the possible worlds. In this context, the probability of a query q to L can be computed as:

Computing $P(q)$ by enumerating and summing all the worlds from which q can be derived is not feasible. Therefore, approximation techniques or techniques based on mapping boolean formulas to specific representations (e.g. Binary Decision Diagram [13] and Sentential Decision Diagram [25]) are usually adopted.

A DeepProbLog program [9] can be seen as a neural extension of ProbLog. The difference with respect to ProbLog program is that DeepProbLog introduces neural computed

[9] used in: 3.7.3

predicates that allow to instantiate neural computed facts. In this case, the output of the underlying networks is normalized in the range [0,1] in order to be interpreted as a probability value. In detail, the neural extension is realized by enhancing ProbLog with a primitive that allow to declare neural predicates:

$$nn(n_{id}, \boldsymbol{X_s}, Y, \boldsymbol{Y_s})$$

where nn is a reserved functor used to to declare a neural predicate, n_{id} is an identifier for the underlying neural network, $\boldsymbol{X_s}$ denotes a sequence of n input variables ($\boldsymbol{X_s} = \{X_i\}_{i=1}^{n}, n \in N$), Y is the output variable, and Y_s denotes a sequence of m possible values ($\boldsymbol{Y_s} = \{y_i\}_{i=1}^{m}, m \in N$) that Y can assume.

2.3.2 Logic Tensor Network

Logic Tensor Network is a neuro-symbolic framework where neural networks are combined with a rule-based symbolic reasoning through fuzzy logic. Before seeing the technical details of LTN, we introduce basic concepts related to fuzzy logic.

Fuzzy Logic

Fuzzy logic is a many-valued logic that extends classical boolean logic by allowing truth values in the range $[0, 1]$. It is based on the work of Zadeh on fuzzy sets [95] which generalize standard crisp sets by introducing the concept of "degree of membership" of an element to a set. Formally, if we denote with U the universe of discourse, (i.e. all the objects we can talk about) and with A a subset of U, we can describe A as:

$$A = \{x \mid m_A(x) = 1\} \quad \text{with } \ m_A(x) = \begin{cases} 1, \ x \in A \\ 0, \ x \notin A \end{cases}$$

where m_A is a function called membership function (also referred to as characteristic function) that decides if an element belongs or not to a set. If A instead is a fuzzy set, we have that $m_A(x) \in [0, 1]$. As we move from crisp to fuzzy sets, we should also define the fuzzy counterpart of the set operators. If A and B are two fuzzy sets, we have:

- $m_{A \cap B}(x) = \{\min(m_A(x), m_B(x))\}$
- $m_{A \cup B}(x) = \{\max(m_A(x), m_B(x))\}$
- $m_{\overline{A}}(x) = 1 - m_A(x)$

Fuzzy statements

Fuzzy logic is usually used to model uncertainty and vagueness related to linguistic variables. A linguistic variable is defined as a variable whose values are sentences in natural or artificial languages. For example, suppose that we have as a linguistic variable the temperature of a place. Related to the temperature, we have 3 expressions (i.e., values): hot, warm, and cold. For each of these expressions, there would be a fuzzy set with a corresponding membership function that we denote with m_{hot}, m_{warm} and m_{hot}, respectively. Continuing the example, a temperature of $20°C$ would have a higher degree of membership for warm compared to hot and cold (i.e, we would have that $m_{cold}(20) < m_{hot}(20) < m_{warm}(20)$). Furthermore, as in classical boolean logic, using logical operators, we would be able to combine these expressions to build more complex statements. In addition, since we are no more dealing with standard boolean logic, we should have a way to evaluate these fuzzy statements and return a value in the range $[0, 1]$. For this reason, different fuzzy interpretations have been proposed in the literature like Łukasiewicz, Product and Godel. These can be seen as a kind of soft interpretation of classical logical operators that still cover their semantics in case of crisp values (i.e. when statements can only be true or false).

Logic Tensor Networks

After talking about fuzzy logic, we are ready to introduce Logic Tensor Networks (LTN)[10]. LTN [78] is a neuro-symbolic framework where data driven models (e.g., neural networks) are enhanced with knowledge about world. The core of LTN is represented by real logic, a fully differentiable first-order logic language that maps logical concepts (e.g., propositions and predicates) into real vector space (i.e, to tensors). Formally, real logic is defined on a first-order language L which contains the following set of symbols:

- $\mathcal{C}$ a set of constants symbols
- $\mathcal{X}$ a set of variable symbols
- $\mathcal{F}$ a set of functor symbols
- $\mathcal{P}$ a set of predicate symbols

Examples of formulas that we can build from L are:

$$\forall x \exists y, z(FatherOf(y, x) \land MotherOf(z, x))$$
$$MotherOf(z, x) \land MotherOf(z, y) \rightarrow Brothers(x, y)$$

[10] used in: 5

that state, respectively, that every person has a father and a mother, and that if two persons have the same mother, then they are brothers. Since formulas involve objects of different types, a set $\mathcal{D}$ is introduced to represent domain symbols. The assignment of types to symbols of L is done by 3 functions, $\boldsymbol{D}$, $\boldsymbol{D_{in}}$, and $\boldsymbol{D_{out}}$. $\boldsymbol{D}$ returns the domain of a given constant or variable, $\boldsymbol{D_{in}}$ returns the domain of the arguments of a functor or predicate, and $\boldsymbol{D_{out}}$ returns the domains of the range of a functor. The connection between symbols of real logic and real vector spaces is achieved by another function that is called *grounding* and it is denoted by $\mathcal{G}$. This kind of interpretation for grounding is different from its standard interpretation, (renamed in [78] as instantiation), which replaces variables of a given term with constants or other terms that do not contains variables. More precisely, a grounding provides an interpretation of both domain symbols $\mathcal{D}$ and non-logical symbols of L. In details, a grounding $\mathcal{G}$ is a function that:

- $\forall d \in \mathcal{D}$, it assigns a set $\mathcal{G}(d) \subseteq \bigcup_{n_1 \dots n_z \in \mathbb{N}^\star} \mathbb{R}^{n_1 \times \dots \times n_z}$ and for every $d_1, \dots d_m \in \mathcal{D}^\star$, $\mathcal{G}(d_1, \dots, d_m) = \times_{i=1}^{m} \mathcal{G}(d_i)$ [11]
- $\forall c \in \mathcal{C}$, it assigns a tensor $\mathcal{G}(c)$ in the domain $\mathcal{G}(\boldsymbol{D}(c))$
- $\forall x \in \mathcal{X}$, it assigns a finite sequence of tensors $t_1, \dots t_k$ each one in the domain $\mathcal{G}(\boldsymbol{D}(x))$
- $\forall f \in \mathcal{F}$, it assigns a function that takes as input tensors of domain $\mathcal{G}(\boldsymbol{D_{in}}(f))$ and returns as output tensor belonging to domain $\mathcal{G}(\boldsymbol{D_{out}}(f))$
- $\forall p \in \mathcal{P}$, it assigns a function that takes as input tensors of domain $\mathcal{G}(\boldsymbol{D_{in}}(p))$ and returns as output a (truth) value in the range [0,1]

Regarding the (fuzzy) semantics of logical connectives, the following interpretations are used:

- $\neg$: $N_s(v_1) = 1 - v_1$ (standard negation)
- $\wedge$: $T_p(v_1, v_2) = v_1 v_2$ (product t-norm)
- $\vee$: $S_p(v_1, v_2) = v_1 + v_2 - v_1 v_2$ (dual product t-conorm)
- $\rightarrow$: $I_r(v_1, v_2) = 1 - a + ab$ (Reichenbach implication)
- $\exists$: $A_{pM}(v_1, \dots v_n) = \left(\frac{1}{n}\sum_{i=1}^{n} v_i^p\right)^{\frac{1}{p}} \quad p \geq 1$ (generalized mean)
- $\forall$: $A_{pME}(v_1, \dots v_n) = 1 - \left(\frac{1}{n}\sum_{i=1}^{n}(1 - v_i)^p\right)^{\frac{1}{p}} \quad p \geq 1$ (generalized mean w.r.t. the error)

where $v_1 \dots v_n \in [0, 1]$

[11] $\star$ denotes the Kleene star

Learning & Reasoning

Roughly speaking, learning in real logic consists in finding (according to a search criterion) the best parameters of a parametric grounding $\mathcal{G}_\theta(\cdot)$. More formally, if $T = \langle \mathcal{K}, \mathcal{G}_\theta(\cdot), \boldsymbol{\theta} \rangle$ is a first order theory in real logic, where $\mathcal{K}$ is a set of closed formulas built from the symbols of L and $\boldsymbol{\Theta}$ is the hypothesis space of the symbols' parameters, learning consists in finding the parameters $\boldsymbol{\theta}^*$ over $\boldsymbol{\Theta}$ that maximize the satisfiability of T:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}} \ \operatorname{SatAgg} \mathcal{G}_\theta(\phi) \tag{2.11}$$

where SatAgg is an aggregation function for the formulas of $\mathcal{K}$ (i.e., SatAgg : $[0,1]^* \rightarrow [0,1]$) that computes the satisfiabilty of T.

Reasoning in real logic is the task of verifying if a logical formula ϕ is a logical consequence of the theory T. Directly applying the logical consequence for fuzzy logic would not be possible since grounded theory $\langle \mathcal{K}, \mathcal{G}_\theta(\cdot) \rangle$ may have a satisfiability value less than one. To overcome this issue, one can define a "satisfiability interval" [q, 1] with $\frac{1}{2} < q < 1$ and state that a formula is true if its satisfiability value is into the interval. As a consequence, one can say that a formula ψ is a logical consequence of T, if, for every grounded theory $\langle \mathcal{K}, \mathcal{G}_\theta(\cdot) \rangle$, if $\operatorname{SatAgg}(\mathcal{K}, \mathcal{G}_\theta(\cdot)) \geq q$ then $\mathcal{G}_\theta(\cdot)(\phi) \geq q$. Nevertheless, the drawback in this case, may be that one has to query the truth value of ϕ for a potentially infinite set of groundings. One possible solution would be limit the number of grounding by focusing only on the grounded theories that maximize T. Therefore, the objective becomes to find all $\boldsymbol{\theta}^*$ that satisfy both T and ϕ:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \operatorname{SatAgg}(K, \mathcal{G}_{\boldsymbol{\theta}}) \quad \text{and} \quad \operatorname{SatAgg}(\mathcal{K}, \mathcal{G}_{\boldsymbol{\theta}^*}) \geq q \tag{2.12}$$

Alternatively to this reasoning, which is referred in [78] as querying after learning, one can reason by refutation (aka proof by refutation) and find a counterexample such that the logical consequence does not hold (for further details about learning and reasoning, see Subsections 3.2 and 3.4 of [78]).

Example 1. *Suppose we want to train a binary classifier S to recognize if a given email is a spam or not. In this scenario, we may have the following:*

- ***Domain:***

  emails

- ***Variables:***

  - *x_+ for spam emails*

  - x_- *for not spam emails*
  - x *all emails (i.e, all examples)*
  - $\boldsymbol{D(x)} = \boldsymbol{D(x_+)} =, \boldsymbol{D(x_-)}\ = emails$

- ***Predicates:***
  - $S(x)$ *(trainable) spam email classifier*
  - $\boldsymbol{D_{in}}(S) = emails$

- ***Rules:***
  - $\forall_{x_+} S(x_+)$
  - $\forall_{x_-} \neg S(x_-)$

- ***Grounding:***
  - $\mathcal{G}(emails) \in \mathbb{R}^{\ltimes}$ *(i.e, each email is represented by a feature vector of size n)*
  - $\mathcal{G}(x) \in R^{m \times n}$ *(a batch of* m *examples)*
  - $\mathcal{G}(x_+), \mathcal{G}(x_+) \in R^n$
  - $\mathcal{G}(S|\theta) : NN_\theta(x) \to 0, 1,$ *the binary classifier is mapped to a Neural Network (NN) parameterized by* θ

As stated before, learning consists in finding the set of parameters $\boldsymbol{\theta}^*$ that maximize the satisfiability of our theory. In this example, if we define with $\boldsymbol{D_e}$ the dataset containing all email examples our objective function becomes:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\operatorname{argmax}}\ \operatorname{SatAgg}_{\phi \in K} \mathcal{G}_{\theta, x \leftarrow \boldsymbol{D_e}} \phi(x)$$

where $K = \{\forall_{x_+} S(x_+), \forall_{x_-} \neg S(x_-)\}$ and $\mathcal{G}_{\theta, x \leftarrow \boldsymbol{D_e}} \phi(x)$ means tht the variables x is grounded with the data of $\boldsymbol{D_e}$ when grounding $\phi(x)$. Since, training a NN is usually done on batches of examples, the overall loss function becomes:

$$Loss = 1 - \operatorname{SatAgg}_{\phi \in K} \mathcal{G}_{\theta, x \leftarrow \boldsymbol{B}} \phi(x)$$

where $\boldsymbol{B}$ is batch randomly sampled from $\boldsymbol{D_e}$. After training, we simply test the perfomance of the model by checking if the predicted class (usually choose by comparing the output of the NN with a threshold) is equal or not to the ground truth class.

Chapter 3

A Neuro-Symbolic Approach to Structured Event Recognition

Events are structured entities with multiple components: the event type, the participants with their roles, the outcome, the sub-events etc. A fully end-to-end approach for event recognition from raw data sequence, therefore, should also solve a number of simpler tasks like recognizing the objects involved in the events and their roles, the outcome of the events as well as the sub-events. Ontological knowledge about event structure, specified in logic languages, could be very useful to solve the aforementioned challenges. However, the majority of successful approaches in event recognition from raw data are based on purely neural approaches (mainly recurrent neural networks), with limited, if any, support for background knowledge. These approaches typically require large training sets with detailed annotations at the different levels in which recognition can be decomposed (e.g., video annotated with object bounding boxes, object roles, events and sub-events). In this chapter, we propose a neuro-symbolic approach for structured event recognition from raw data that uses ”shallow” annotation on the high-level events and exploits background knowledge to propagate this supervision to simpler tasks such as object classification. We develop a prototype of the approach and compare it with a purely neural solution based on recurrent neural networks, showing the higher capability of solving both the event recognition task and the simpler task of object classification, as well as the ability to generalize to events with unseen outcomes.

3.1 Introduction

Events are structured entities with multiple components and relations with other entities [74]. The most important components of an event are the event type, the participants with their roles, the sub-events, and the event outcome. Therefore, the approaches for full fledged event recognition should be able to extract the information about all the components of the events that happen in a data sequence. To this aim, a system for event detection should solve a number of different simpler tasks like recognizing the objects involved in the events and their roles, the outcome of the events as well as the sub-events. In this context, having background knowledge about the event structure, specified in logic languages, could be very useful to solve the aforementioned challenges. However, looking at [88], one can see that the majority of neural approaches (aka sub-symbolic) applied in event recognition strictly rely on the features learnt by the underlying networks with limited, if any, support for background knowledge. Furthermore, the training of the underlying networks of these approaches requires a large amount of training data with a detailed supervision on all the events' components (e.g., a video annotated with events, sub-events, object roles and object bounding boxes). Alternatively, one could think to have data with an annotation limited to the occurrence of an event (i.e., a ”shallow” annotation) and exploit the background knowledge to infer information on the event components. For example, if a video clip is annotated with the event ”John is preparing a cappuccino to Mary”, one can infer from the background knowledge that the video is showing at least two people, one male and one female, that John is preparing the cappuccino by mixing milk and coffee into two cups, etc. As another example, if a video clip is annotated with the event ”A potential threat is happening” and the background knowledge defines the potential threat as a person moving, leaving his/her bag and then moving away, one can infer that the clip contains at least one person, which is wearing a bag, and that the (simple) sub-events move, leave a bag, and move away are happening one after the other. All of these inferred facts can be used as supervision to neural networks to solve the simpler tasks defined above and to recognize the structured event as well. In this case, neuro-symbolic frameworks e.g., DeepProbLog [59] that combine low-level neural perceptions with logic reasoning (also know as symbolic) seem to be suitable approaches to achieve these objectives. In this chapter, we propose a neuro-symbolic approach for structured event recognition from raw data that uses ”shallow” annotation on the high-level events and exploits background knowledge to propagate this supervision to simpler tasks such as object classification. We develop a prototype of the approach and compare it with a purely neural solution based on a recurrent neural network, showing the higher capability of solving both the event recognition task and the simpler task of object classification, as well as the ability to generalize to events with unseen outcomes. The detailed contributions of this chapter are the following:

1 a formal definition of the problem of structured event recognition from raw data sequences with ”shallow” annotations and of a neuro-symbolic solution combining

low-level neural-based predictions with high-level reasoning;

2 a framework for automatically generating simple videos containing events that are fully annotated;

3 a prototypical neuro-symbolic recognition approach based on DeepProbLog;

4 an experimental evaluation that compares our approach with a purely neural solution, showing the advantage of explicitly using background knowledge.

The rest of the chapter is organized as follows: Section 2 formally defines the problem of structured event recognition with ”shallow” annotation; Section 3 presents our proposed solution; Section 4 briefly reviews the state of the art approaches that have been proposed in the context of event recognition, and presents some of the most well-known neuro-symbolic frameworks; Section 5 describes the event generation framework; Section 6 presents the experimental setting; Section 7 describes the neural LSTM approach and our prototype approach based on DeepProbLog; Section 8 reports the experimental results; Finally, Section 9 draws some concluding remarks.

3.2 Related Work

Event recognition has always attracted researchers coming from different fields, like Computer Vision and NLP. The particular attention towards event recognition may be motivated by its multiple data stream nature and by its impact in people's daily life. Looking at the literature, approaches to event recognition can be classified into three categories: sub-symbolic, symbolic and neuro-symbolic. Sub-symbolic approaches (mostly based on neural networks) moved from manually crafted features to automatic features learning (see [2] and [88] for a survey). These approaches require a huge amount of training data with a rich annotation at different levels (e.g, the type and temporal location of the event). These data are hard to collect and it is difficult to ensure high-quality annotations for large amounts of example, so that high levels of annotation errors can affect the accuracy of the networks being trained. Furthermore, the trained model is a black box model that cannot explain its decisions and is not guaranteed to be consistent with existing background knowledge. An attempt to make sub-symbolic approaches more interpretable is represented by Concept Bottleneck Models [49] where the activation of (a subset of) human-specified concepts is used to explain the model's final decision. However, works like [49] focus on atemporal domains (e.g. image). In [41], authors propose a novel approach that addresses concepts explanation in videos, but they are not able to capture spatial and temporal relationships between concepts. On the other hand, symbolic approaches like [9], are explainable and knowledge-consistent, but are not robust in the presence of noise. Therefore, symbolic approaches dealing with uncertainty have been proposed [3]. In [80, 8], authors recognize higher events by combining evidence of simple events with domain knowledge using the probabilistic logic programming framework ProbLog [73] . In this case, knowledge on low level events is assumed to be given. Recently, neuro-symbolic approaches have started to be applied in the context of event recognition. In works like [46, 47, 91, 85, 34], pre-trained neural networks are used to extract lower events and then passed to a symbolic layer that encodes the knowledge of the domain in the form of logic rules. In [90], authors propose an end-to-end model where a neural network is also used to learn to simulate the symbolic layer. One of the drawbacks is that the neural network has to be re-trained in case of even minimal changes/updates of the existing knowledge. Among existing frameworks, DeepProbLog[56] is particularly appealing in terms of expressivity. DeepProbLog extends the probabilistic programming language ProbLog with neural predicates, achieving an elegant and powerful combination of neural networks, logic and probability. Furthermore, by using DeepProbLog, any change/update to knowledge can be easily integrated. We thus leverage DeepProbLog as the underlying integration framework for our neuro-symbolic event recognition prototype.

3.3 Problem definition

Let $\mathcal{L}$ be a first order language with three sorts, $\mathbb{O}$, $\mathbb{E}$, and $\mathbb{T}$. Terms of sort $\mathbb{O}$ denote objects, terms of sort $\mathbb{E}$ denote events, and terms of sort $\mathbb{T}$ denote time-points. The language contains the constants $0, 1, 2, \ldots$ of sort $\mathbb{T}$, used to name time points and the binary relation $<: \mathbb{T} \times \mathbb{T} \to \{\top, \bot\}$. The language $\mathcal{L}$ also contains a set of predicates $\mathcal{P}$ of sort $\mathbb{O}^k \to \{\bot, \top\}$, which are used to describe the time invariant properties and relations between objects. $\mathcal{L}$ contains also a set of function symbols $\mathcal{E}$ of sort $\mathbb{O}^k \to \mathbb{E}$ that are used to describe events that involve a (possible empty) tuple of objects. We also have a relationship $\mathit{outcome}(\mathbb{E}, \mathbb{O})$ that is used to describe the fact that the outcome of an event is an object. Finally, $\mathcal{L}$ contains the predicate $\mathit{happens}(\mathbb{E}, \mathbb{T}, \mathbb{T})$ that is used to describe the fact that a certain event happens within an interval of time. For example, the formula $\exists x.\mathit{happens}(\mathit{drop}(\mathit{John}, x), t_1, t_2)$ states that *John* drops an object x at some time between t_1 and t_2. Notice that events can "create" new objects, for example the result of mixing milk and coffee is a cappuccino. This is expressed by the formula $\mathit{milk}(x) \wedge \mathit{coffee}(y) \to \mathit{outcome}(\mathit{mix}(x, y), z) \wedge \mathit{cappuccino}(z)$. A *narrative* is an interpretation $\mathcal{I}$ of the language $\mathcal{L}$, where the terms of sort $\mathbb{T}$ are interpreted in the set of natural numbers and $<$ in the usual linear order. The terms of sort $\mathbb{O}$ are interpreted in a domain of objects $\Delta_{\mathbb{O}}$ and those of $\mathbb{E}$ are interpreted in a domain of events $\Delta_{\mathbb{E}}$. Since we are interested in finite narratives, i.e., narratives that involve a finite number of objects and a finite number of events and time points, we can specify a narrative by using the Herbrand Base. In particular, for every $k > 0$ we define a k-narrative as a pair $\mathcal{N} = (\mathcal{C}, \mathcal{F})$ where:

- $\mathcal{C}$ is a finite set of new constants for objects of type $\mathbb{O}$;
- $\mathcal{F}$ is a subset of ground atoms in the language of $\mathcal{L}$ extended with the constants in $\mathcal{C}$ and the constants $0, 1, \ldots, k$ of sort $\mathbb{T}$; such that: if $\mathit{happens}(e, t_1, t_2) \in \mathcal{F}$ then $t_1 \leq t_2$.

Our main aim is to reconstruct a narrative from a data stream using some neuro-symbolic method that is capable of combining low-level data processing capabilities with the ability to leverage background knowledge about the structure of events. More formally, let $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^k$ be a data sequence of length k, where each $\boldsymbol{x}_i$ is a low-level representation for sequence element i (like a real-valued vector, matrix or tensor). Our main objective is to generate a k-narrative that describes the events that happen in $\boldsymbol{X}$, when they happen, their participants, and their outcomes. In other words we want to extract from $\boldsymbol{X}$:

- a set of objects;
- the properties and the relations between the objects;
- the set of events that happen;
- the objects (arguments) that are involved in each event that happens;
- the outcomes of the events that happen.

Example 2. *Let $\boldsymbol{X}$ be a video showing two people, one moving, leaving a bag and then moving away, and the other standing. We would like to produce the following narrative:*

$$\mathcal{C} = \{p_1, p_2, b_1\} \qquad \mathcal{F} = \left\{ \begin{array}{c} \mathit{person}(p_1), \mathit{person}(p_2), \mathit{bag}(b_1), \\ \mathit{happens}(\mathit{move}(p_1), 0, 4), \mathit{happens}(\mathit{drop}(p_1, b_1), 4, 5) \\ \mathit{happens}(\mathit{move}(p_1), 5, 7), \end{array} \right\}$$

The type of supervision we suppose to have, in order to learn a model that extracts narratives from data, is partial and consists of a set of n data sequences labelled with *some* (not necessarily all the) ground facts about events happening in the sequence:

$$\left\{\boldsymbol{X}^{(i)}, \mathcal{F}_p^{(i)}\right\}_{i=1}^n$$

where $\boldsymbol{X}^{(i)} = \{\boldsymbol{x}_j^{(i)}\}_{j=1}^k$ is a data sequence and $\mathcal{F}_p^{(i)}$ is a set of positive and negative literals, denoting a subset of the events that happen or don't happen in $\boldsymbol{X}^{(i)}$. Notice that we do not need to have a complete labelling for all the events. Furthermore, notice that the supervision provided via $\mathcal{F}_p^{(i)}$ also provide a supervision for the subset of objects $\mathcal{C}^{(i)}$ that appear in the data stream $\boldsymbol{X}^{(i)}$, which is the set of constants of type $\mathbb{O}$ that appear in the positive literals of $\mathcal{F}_p^{(i)}$.

Events can be related to each other, and structured events can be defined in terms of simpler ones.

Example 3. *Let potential_threat represent a structured event corresponding to a potential threat represented by something happening in a video like the one in the previous example. The threat could be modelled by the following formula:*

$$\begin{aligned} &\mathit{happens}(\mathit{potential\_threat}, t_0, t_3) \leftrightarrow \\ &\qquad \exists x, y, t_1, t_2 . \mathit{person}(x) \wedge \mathit{bag}(y) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{move}(x), t_0, t_1) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{drop}(x, y), t_1, t_2) \wedge \\ &\qquad\qquad \mathit{happens}(\mathit{move}(x), t_2, t_3) \end{aligned} \tag{3.1}$$

An example of supervision in this context could be a set of videos $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(m)}$ of length k, each of which is annotated with either the single fact $\mathit{happens}(\mathit{potential\_threat}, 0, k)$ or with the single fact $\neg \mathit{happens}(\mathit{potential\_threat}, 0, k)$.

3.4 Proposed solution

Looking at the examples of the previous section, we observe that a structured event can be expressed in terms of simple events using logical languages. Simple events include the objects participating in the structured event, their relationships and their individual actions. Therefore, the correct recognition of the simple events combined with the definition of the structured event at the logical level will lead to the recognition of the structured event.

Our proposed approach has two aims, respectively learning to recognize the structured event happening in a data sequence and the simple events that compose it. To achieve these aims, we provide both background knowledge on the domain, expressed in terms of logical formulas, and ”shallow” annotations on the structured event, like the one for the *potential_threat* example of Section 3.3. In order to solve our problem, we have to complete three tasks:

object detection: in order to build the narrative, we have to find the set of objects $\mathcal{C}$[1] that appear in a data sequence $\boldsymbol{X}$.

[1] objects and constants are used interchangeably

object classification and relation detection: We also have to classify the objects in their types, e.g., a chair, a person, ..., and we have to detect relations between objects, e.g. if the person holds a bag or not.

event recognition: we have to recognize the events that happen in the video.

The traditional approach to solve the problem is to use a pipeline, where the above tasks are solved sequentially and the result of the solution of the previous task is provided as input to the next task. However, this requires supervision at all levels, the objects in the data, their class and the events. We instead have only partial supervision on some events.
In our solution, we propose to have a fully end-to-end approach in which both the supervision on data and the background knowledge are used to train some neural networks for more data driven tasks such as object detection and classification. We therefore suppose to have the following components:

- A neural network $\mathrm{Det}_{\mathrm{nn}}$ that takes as input a sequence $\boldsymbol{X}$ and returns a set of objects $\mathcal{C}$ each of which is associated with a set of numeric features $f(o)$. For example, if $\boldsymbol{X} = \{\boldsymbol{d}_i\}_{i=1}^{k}$ is a video, then $f(o)$ contains the bounding boxes of object o at each frame $\boldsymbol{x}_i$ and the crop of the image on the bounding box for each frame;

- for some (not necessarily all) object predicates we have a network P_{nn} that takes as input the features of an n-tuple of objects $f(o_1), f(o_2), \dots, f(o_n)$ and returns a sequence in $[0,1]^k$ which represents the level of truth or probability of truth of the predicate at each time point $0 \leq i \leq k$. For example, for the *person* predicate in the examples in the previous section we would have a network that given a sequence of cropped images outputs for each image the probability that it contains a person.

All the outputs of the neural networks defined above can be combined with the background knowledge which is described in terms of the axioms, such as Equation (3.1). The way in which this combination is achieved could be based both on probabilistic semantics or on fuzzy semantics. At this stage we do not want to commit on one or the other. A list of neuro-symbolic approaches that can be adopted to implement our architecture is provided in the related work section. In the following we provide a proof-of-concept implementation of the architecture described above using DeepProbLog [56]. Other neuro-symbolic approach based on logic programming, like NeurASP [92], may be used instead of DeepProbLog. However, we believe that the recent improvement in DeepProbLog [57][2] as well as its large application in the literature, makes it a more suitable candidate to develop our prototype. One could also think to solve the problem by moving from logic

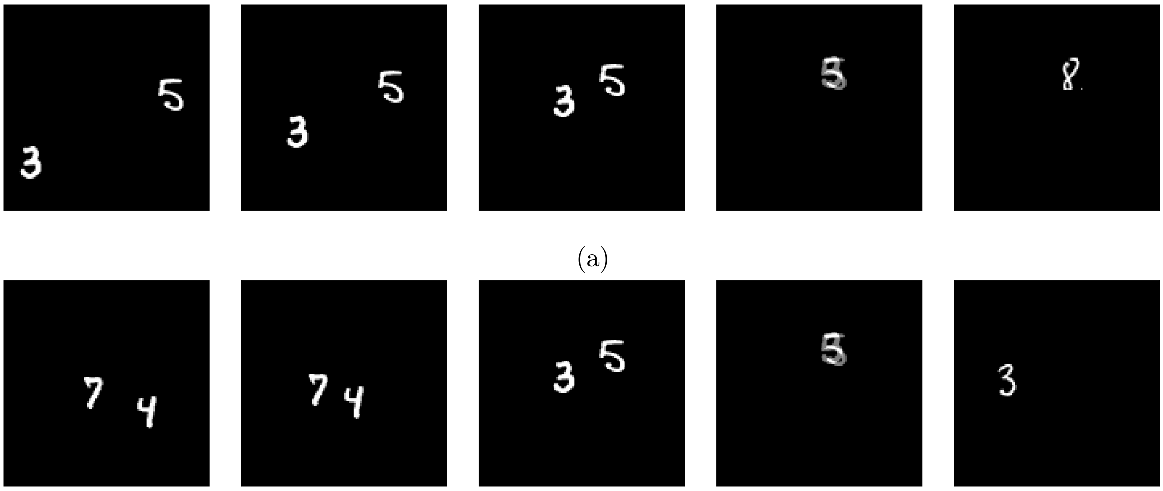
[2] this work has been published later with the respect to the article to which this chapter refers to

programming to fuzzy logics. For example, by adopting the LTN framework [78]. In this case, one has to reformulate the problem using real logic (i.e, many symbolic approaches for event recognition are based on logic programming). In addition, it is not clear how much scalable the LTN framework is. Indeed, most of the works in the literature have used [78] to solve atemporal task. As far as we know, the only work that have used LTN in a point based temporal reasoning, is the work by Umili et al. [84]. However, this work considers short length sequence and then it is not clear how the approach will scale when dealing with longer sequences.

3.5 Event generation framework

Most of the available datasets for event recognition provide only limited annotations on some but not all the elements of an event. For example, in the context of event recognition in video there are dataset like Olympic Sport [67] and UCF101 [81] that provide annotation only on the event happening in the video. Datasets like CAVIAR [15], MEVA [22], Cooking [53] and HiEve [55] provide a richer annotation, e.g., objects classes, object locations and distinction between simple and structured events, but they introduce a level of complexity in the visual part that requires pre-trained models for processing low level features. Here we are interested in developing a neuro-symbolic system that is trainable end-to-end, where the learning of the low-level processing is influenced by the high-level knowledge. Furthermore, the above mentioned datasets were manually curated, and cannot be extended to consider newly defined structured events without a tedious process of data collection and manual annotation. We, instead, would like to be able to quickly generate new data streams containing new events so as to support a fast prototyping and testing of recognition architectures. For these reasons, we have implemented a video generator of events involving MNIST digits. The generator allows to generate videos of different length and with a different number of digits that interact with each other, together to the narrative describing the objects and events in the video. The generator uses an object predicate $digit(x, v)$ to indicate that v is the value corresponding to object x. Concerning events, we distinguish between simple events that involve single digits and structured ones that involve combinations of digits. The simple events we defined are:

- $appear(x)$: a digit x appears in the video
- $disappear(x)$: a digit x disappears from the video
- $enter(x)$: a digit x enters in the video



(a)

(b)

Figure 3.1: Example of events generated by the framework: join_add (a) and join_sub (b)

- *exit*(x): a digit x exits from the video

The difference between *appear*/*disappear* and *enter*/*exit* is that in the former case the digit is always fully visible when in the video, while in the latter case the digit is only partially visible upon entering/exiting. Example of structured events definable in the framework are:

- *join_add*(x, y): two digits, respectively x and y, approach each other, overlap and then the digit that is the result of the sum of the two digits appears, i.e., $outcome(join\_add(x, y), z)$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_z = v_x + v_y$. Note that this event can only happen if the sum of the two digits is ≤ 9.
- *join_sub*(x, y): two digits, respectively x and y, approach each other, overlap and then the digit that is the result of the difference between the two digits appears, i.e., $outcome(join\_sub(x, y), z)$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_z = max(v_x, v_y) - min(v_x, v_y)$.
- *split*(x): a digit x splits into two digits whose sum or difference gives the value of x, i.e., $outcome(split(x), (y, z))$ with $digit(x, v_x), digit(y, v_y), digit(z, v_z)$ and $v_x = v_y + v_z$ or $v_x = max(v_y, v_z) - min(v_y, v_z)$.

Some examples of structured events produced by the generator are shown in Figure 3.1. For simplicity, each object is assumed to participate in at most one simple event and one structured event for each frame. For each video, a narrative file is also produced that contains the following information for each digit:

- the name: a label
- the class: the corresponding MNIST class
- the position: x and y coordinates inside the frame
- the simple event (if any) the digits is involved in
- the structured event (if any) the digits is involved in

3.6 Experimental setting

Our experimental evaluation is aimed at verifying whether a neuro-symbolic solution has an advantage in recognizing structured events with respect to a fully neural approach. In addition to the capability of correctly classifying each video into the corresponding structured event, we aim at evaluating the ability to learn to correctly classify the underlying objects (the digits) as well as the ability to generalize to unseen outcomes (e.g., the result of a `join_add` being a digit for which no explicit supervision was ever received). The scenario and learning setting we created to this aim are described in the following.

3.6.1 Scenario

The scenario consists of videos produced with the event generation framework described in Section 3.5. Each video consists of 10 frames, each frame showing one or two digits. Digits can appear anytime within the first half of the video, and only disappear if they join together. When present, the digits are always completely visible, apart from the frames in which they overlap with each other (e.g., right before a join). We generated three types of videos:

- *join_add*
- *join_sub*
- *no_join*

The first two refer to videos where the corresponding structured event as discussed in Section 3.5 takes place. The resulting digit can stay in the same position or move. To avoid ambiguities, we refrain from generating videos where one of the two operands is a zero. As consequence, the only way to get a zero is in a *join_sub* when both digits are equal. The third type refers to videos where neither of the two structured events takes place. In this case the video contains two arbitrary digits that wander around with no restrictions, possibly overlapping with each other.

3.6.2 Learning setting

Our goal is to test the ability of the different approaches to learn to recognize events with partial supervision. The idea is to provide supervision in terms of the structured event taking place (if any) and the outcome of the event (i.e., the result of the addition/subtraction). Supervision is thus provided in terms of sets like the following:

$$\{happens(join\_add(x,y),1,T), outcome(join\_add(x,y),z), digit(z,4)\}$$
$$\{happens(join\_sub(x,y),1,T), outcome(join\_sub(x,y),z), digit(z,2)\}$$
$$\{\neg happens(join\_add(x,y),1,T), \neg happens(join\_sub(x,y),1,T)\}$$

Note that this type of feedback provides information on the classification of the underlying objects, even if only when a join takes place, and only for the digit which is the result of the join. We thus build the task to additionally test the ability of the methods to *generalize* to unseen outcomes, i.e., digits that where never observed as the result of a join during training (or validation). For the sake of conciseness, in the following we will refer to the combination of structured event and outcome as the class of a video (with *no_join* being the class of a video where no join occurs). To generate the videos we first split the original MNIST dataset into training, validation and test set. Then, separately for each set, we randomly picked digits to generate a set of videos for each of the video classes, making sure that each class had the same number of videos. We generated training and validation videos containing *no_join*, *join_add* with outcome from 2 to 7 and *join_sub* with outcome from 0 to 7, for a total of 15 video classes. Test videos contain the same classes as the training and validation ones plus *join_add* with outcome 8 and 9 and *join_sub* with outcome 8, for a total of 18 classes. We generated 1500 videos for training, 150 for validation and 180 for testing, so that each class always contains 100 videos.

3.7 Event recognition approaches

In this section, we describe the learning approaches that we used to solve the aforementioned task. We start presenting the low-level neural networks that we use for object detection and classification and proceed describing the fully neural and the neuro-symbolic approaches that build on them. The object detector is pre-trained, while the object classifier is trained end-to-end both in the fully neural and neuro-symbolic approaches. Training is performed for 35 epochs, using the Adam optimizer with a learning rate of

0.001 and early stopping on the validation set. Training for more epochs does not lead to improvements in recognition quality.

3.7.1 Object detector and classifier

The object detector is a neural network that extracts (processed) patches from frames. Its architecture is shown in Figure 3.2 for the case of a single frame, as the same structure is repeated for all frames in a video. Its main module is a standard convolutional neural network that consists of two convolutional layers, each followed by a max-pooling layer, and two fully-connected layers. ReLU are used as activation in all layers apart from the output layer where a sigmoid is used. The module takes as input a frame of size 128×128 and gives as output a vector of length 6:

$$\boldsymbol{o}_{det} = \langle v_1, v_2, x_1, x_2, y_1, y_2 \rangle$$

where $v_i \in [0, 1]$ indicates whether the i-th digit is present in the frame and $x_i, y_i \in [0, 1] \times [0, 1]$ are the normalized digit coordinates (digits are ordered according to the distance between their predicted coordinates and the origin). The two patches corresponding to the coordinates are extracted from the frame, and their content is multiplied by the value of their visibility flag. In so doing, the detector outputs "soft" patches, that depending on the value of the visibility flag range from the patch itself ($v_i = 1$) to a completely black patch ($v_i = 0$).

The digit classifier has the same architecture as the main module of the detector, with the sigmoid replaced by a softmax in the output layer. The classifier takes as input an image of size of 28×28 which corresponds to a processed patch extracted by the detector and returns as output a vector of length 11, where the first 10 elements refers to the 0-9 digits and the last one indicates the absence of a digit. This module is repeated for all patches and all frames of the input video.

3.7.2 Fully neural approach

The fully neural approach combines the predictions of the digit detector and classifier on the different frames using an LSTM recurrent neural network [40]. The overall architecture is shown in Figure 3.3. For each frame in the input video, the detector extracts a pair of patches and sends them to the digit classifier. The predictions of the classifier are concatenated with the visibility and coordinate predictions from the detector and fed to an LSTM cell. After processing the entire input sequence, the LSTM outputs a prediction in three classes, $join\_add, join\_sub$ and no_join. The outcome of the join event is recovered from the output of the digit classifier on the first patch of the last frame. If the class with the highest prediction is no_join, the outcome prediction is ignored.

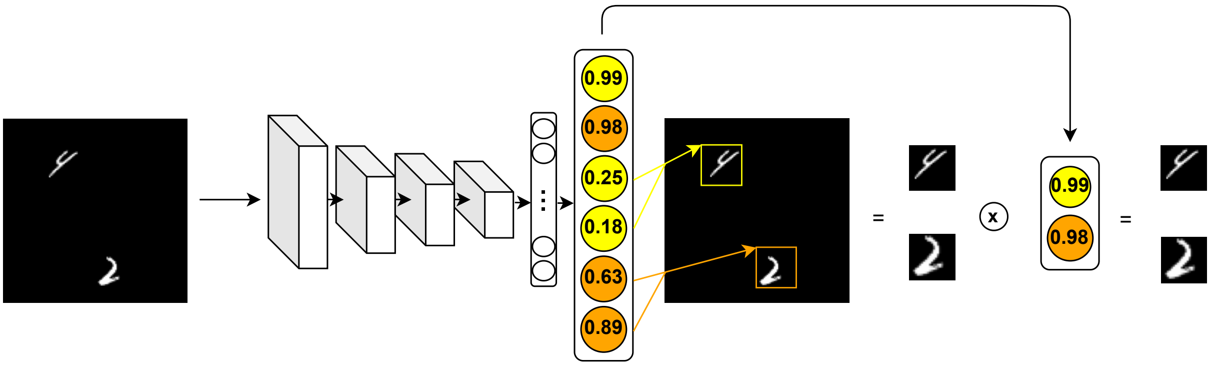


Figure 3.2: MNIST digit detector: A frame of shape 28×28 is given as input to the model that outputs a vector of length 6. The vector contains the visibility scores and the normalized digits coordinates of each digits (yellow and orange respectively). Both the score and the coordinates are used to output "soft" patches that are then passed to the MNIST digit classifier.

3.7.3 Neuro-symbolic approach

We developed a neuro-symbolic approach for structured event recognition using the DeepProbLog [56] framework. This framework can be seen as a neural extension of the probabilistic extension of Prolog, ProbLog [73]. Like ProbLog, the knowledge about the domain is encoded as a set of logical rules (i.e., Horn clauses). In addition, DeepProbLog introduces neural predicates that allow to instantiate facts as outputs of neural predicates processing raw data. The neural extension is realized by enhancing ProbLog with a primitive that allows to declare neural predicates:

$$nn(n_{id}, \boldsymbol{X}_s, Y, \boldsymbol{y}_s)$$

where nn is a reserved functor used to declare a neural predicate, n_{id} is an identifier for the underlying neural network, $\boldsymbol{X}_s$ denotes a sequence of n input variables, Y is the output variable, and $\boldsymbol{y}_s$ denotes a sequence of m possible values that Y can assume. Training of these neural predicates is done by providing supervision on the head of the logical rules expressed as standard logical queries. This means that in our prototype the ”shallow” annotations on the structured event will be mapped to queries, while simple events will be mapped to neural predicates.

The DeepProbLog program we defined to address the event recognition task is shown in Figure 3.4. It consists of the following predicates:

- $digit(X, V, T, V_x)$: a neural predicate that states that the X digit of video V at time T is V_x
- $join\_add\_res(V, V_z)$: a binary predicate that states that video V is a $join\_add$

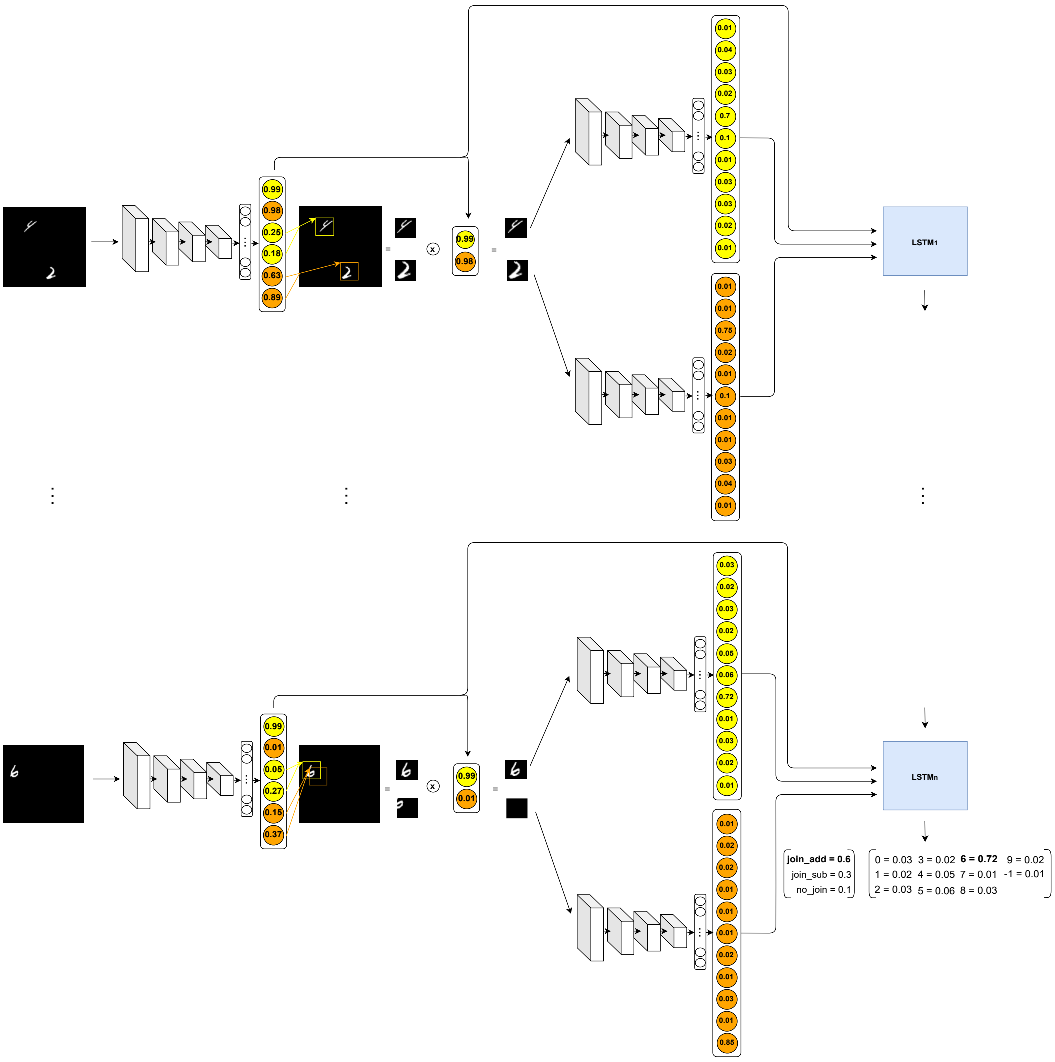


Figure 3.3: Fully neural approach: LSTM-based architecture.

and the resulting digit of the join is V_z

- $join\_sub\_res(V, V_z)$: a binary predicate that states that video V is a $join\_sub$ and the resulting digit of the join is V_z
- $no\_join(V)$: a unary predicate that states that video V is a no_join video

The neural predicate $digit(X, V, T, V_x)$ is mapped to the combination of digit detector and classifier shown in Figure 3.3, with the difference that only the output of the classifier (i.e., a probability distribution on the 0-9 digits plus the absence of the digit) is provided. The predicate $join\_add\_res(V, V_z)$ basically represents the combination of $join\_add(X, Y)$, $outcome(join\_add(X, Y), Z)$ and $digit(Z, V_z)$, with the addition of the V variable indicating the video (omitted for simplicity in the formalization throughout the paper). The

```
nn(mnist_net, [I, V, T], Y, [0,1,2,3,4,5,6,7,8,9,-1]) :: digit(I, V, T, Y).

join_add_res(V, Z) :-                       join_sub_res(V, Z) :-
    between(0, 4, T1),                          between(0, 4, T1),
    digit(0, V, T1, X),                         digit(0, V, T1, X),
    X > 0, X < 9,                               X > 0,
    digit(1, V, T1, Y),                         digit(1, V, T1, Y),
    Y > 0, Y < 10 - X,                          Y > 0,
    digit(0, V, 9, Z),                          digit(0, V, 9, Z),
    Z is X + Y, Z > 1,                          Z is abs(X-Y),
    digit(1, V, 9, -1).                         digit(1, V, 9, -1).

no_join(V) :- digit(1, V, 9, X), X =\= -1.
```

Figure 3.4: Neuro-symbolic approach: DeepProbLog program.

predicate checks whether there are two digits in the first half of the video and only one digit at the end that is the sum of the two. The *join_sub_res* predicate is similar to *join_add_res* with sum replaced by difference (in absolute value, so that digits do not need to be sorted). Finally, for a *no_join*, we know that there are two digits for the whole duration of the video. Therefore, we define a rule that only fires when both digits are visible in the last frame.

3.8 Results

In this section, we present and compare the results of the neural based LSTM approach with our proposed neuro-symbolic approach based on DeepProbLog on the tasks defined in Section 3.6.

Confusion matrices, where entries (i, j) denote the number of samples of true class i classified as class j, for the event recognition problem and related sub-problems for the two approaches are shown in Figure 3.5. The first row shows the confusion matrices for the event and outcome recognition (*join_add* with outcome from 1 to 9, *join_sub* with outcome from 0 to 8, *no_join*), while the second row reports the confusion matrices for the underlying task of digit classification (0-9, and -1 corresponding to no digit). The left column reports results for the fully neural approach, the right column those for the neuro-symbolic approach.

Looking at the top left confusion matrix, we can observe that the neural approach is able to recognize the events for which the supervision is provided, even if it sometimes

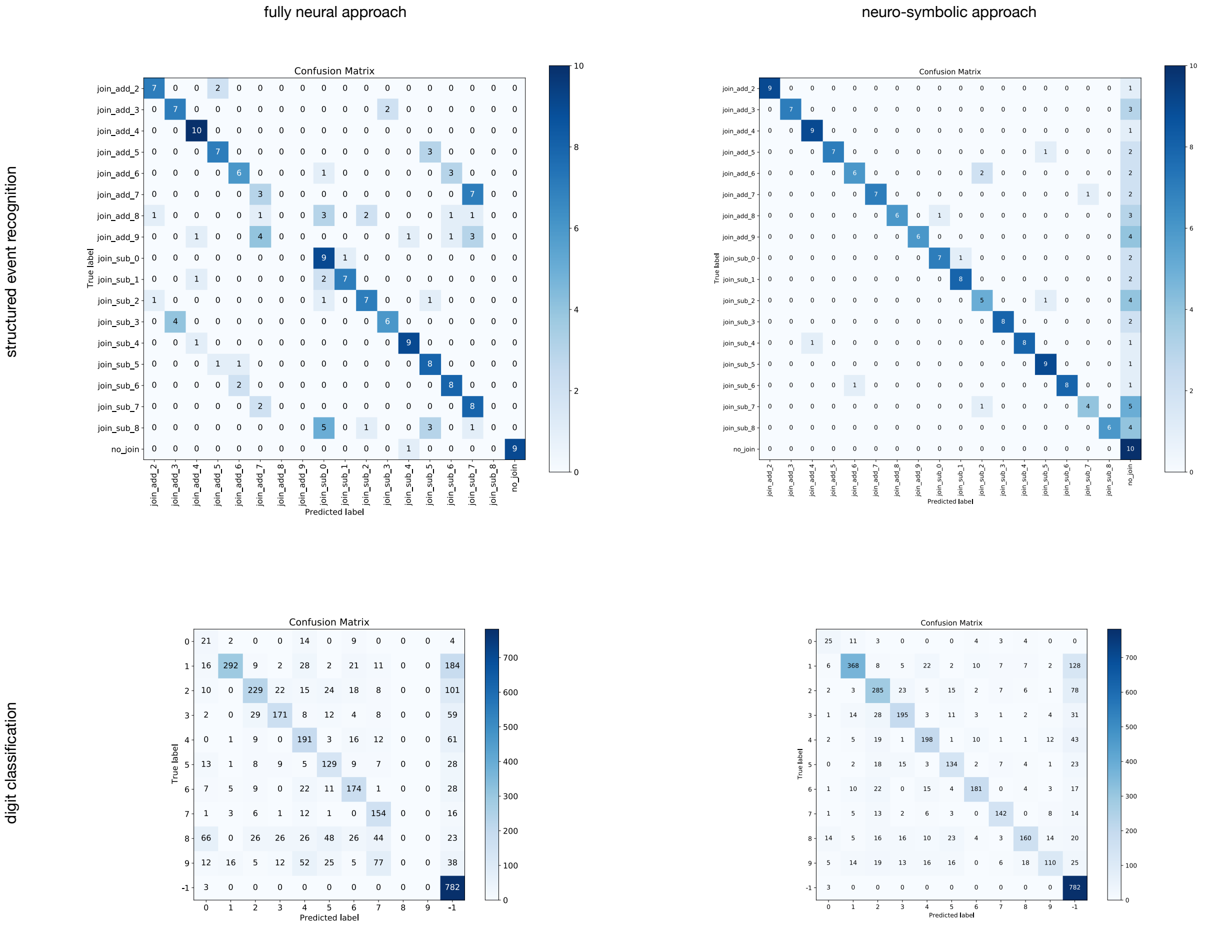


Figure 3.5: Experimental results: confusion matrices for event + outcome recognition (top row) and digit classification (bottom row). Left: fully neural approach; right: neuro-symbolic approach.

mistakes a *join_add* for a *join_sub* and vice-versa when the outcome is the same. On the other hand, it completely fails in generalizing to unseen events (*join_add* with outcome 8 or 9, *join_sub* with outcome 8). This fact highlights the difficulty of the neural approach in fully learning the semantic behind the join operations. The results of the confusion matrix on digits (bottom left) confirm these findings, as the network fails to classify digits for which no direct supervision is available (i.e., 8 and 9).

The situation with our neuro-symbolic approach is rather different (right column). Indeed, DeepProbLog is capable of predicting the unseen outcomes with reasonable accuracy, and the same holds for the underlying digit classification task. If we compare the confusion matrices on the digits of the two approaches (bottom row), we can observe that our approach has a higher accuracy even on digits for which direct supervision is available. These results clearly indicate the importance of the background knowledge in compensating partial supervision and allowing to generalize beyond what is observed during training.

3.9 Conclusion

In this work, we have proposed a neuro-symbolic approach for structured event recognition from data sequences, where background knowledge about event structure is combined with deep neural networks used to solve the sub-tasks of event recognition such as object detection and classification. The proposed architecture can be trained end-to-end with data streams containing only shallow annotations. We prototyped our architecture using DeepProblog as a neuro-symbolic integration framework and tested it on a structured event recognition problem defined on a synthetic dataset automatically generated. The experiments show that the background knowledge about structured events and their outcomes translates supervision on the structured event into supervision on lower-level predictive tasks like object classification, allowing to successfully train the neural components of the architecture. We compare our architecture with a purely neural solution that uses the same basic components for object detection and classification. The comparison shows how the use of background knowledge improves performance for both high-level and low-level prediction tasks. The advantages of these effects are multiple. The first advantage is the fact that we are able to train a classifier without a direct supervision on some of the classes (the classes 8 and 9 in our experiment); a second advantage concerns explanability: while in a fully neural approach it is not possible to explain the happening of an event in terms of its components (object participants, and their types), in our approach the reasoning process that infers the happening of a structured event on the basis of the recognition of some basic facts (detection of an object of a certain type) can be provided as an explanation.

Chapter 4

A neuro-symbolic approach for real-world event recognition from weak supervision

Temporal Event Detection (TED) is the task of detecting structured and atomic events within data streams, most often text or video sequences, and has numerous applications, from video surveillance to sports analytics. Existing deep learning approaches solve TED task by implicitly learning the temporal correlations among events from data. As consequence, these approaches often fail in ensuring a consistent prediction in terms of the relationship between structured and atomic events. On the other hand, neuro-symbolic approaches have shown their capability to constrain the output of the neural networks to be consistent with respect to the background knowledge of the domain. In this chapter, we propose a neuro-symbolic approach for TED in a real world scenario involving sports activities. We show how by incorporating simple knowledge involving the relative order of atomic events and constraints on their duration, the approach substantially outperforms a fully neural solution in terms of recognition accuracy, when little or even no supervision is available on the atomic events.

4.1 Introduction

As we stated in the previous chapter, events are structured entities that involve multiple components, like the participants, their roles, the type, and the atomic events composing it. For example, in athletics, the event *long jump* involves one person (the athlete), performing the atomic events *run*, *jump* and *sit* in sequence. When talking about events, one of the main challenging tasks is temporal event detection (TED) which consists in detecting events within data streams, like text and video. Continuing the example, it consists in identifying the class of the atomic events and the interval of time where they occurred. Many sub-symbolic approaches, mostly based on neural networks, have been proposed for event recognition [2, 88]. One of the main drawbacks of this kind of approaches is the amount of training data. Indeed, having a large training set is fundamental for an appropriate and effective training of the model. Furthermore, "rich" annotations at different levels are required in order to solve the task (e.g., frame-by-frame annotation of atomic events). Large amounts of deeply annotated data are hard to collect. Additionally, errors in the annotations (e.g. in case of crowdsourced ones) may introduce noise in the model and compromise its accuracy. More importantly, purely neural approaches cannot guarantee consistency of the predictions with the domain knowledge, in terms of the relationship between the structured event and the atomic events that compose it. Neuro-symbolic approaches [39] have recently gained increasing popularity as a means to make the best of both worlds, by combining the effectiveness in low-level processing of deep learning technology with the ability of symbolic approaches to express complex domain knowledge. Popular frameworks including DeepProbLog [56], DeepStochLog [87], Logic Tensor Networks [78], LYRICS [59] and NeurASP [92] have been proposed and applied to solve different structured tasks, like Semantic Image Interpretation [28]. In the context of event recognition, the neuro-symbolic cognitive agent of [26] has shown promising results in rules learning from real world complex scenarios. The drawbacks is that one has to re-train the model from scratch even even for a slightly changed of the scenario. [12] proposed an inspired Cortex-system based model to recognize individual and intra human(s) actions. The model is trained on thousands of fully labeled videos. Recently, the DeepProbLog framework has proved effective in recognizing both structured and simple events as well as generalizing to unseen outcomes [6, 86]. However, these results have been obtained on artificial scenarios, and the framework has serious issues of scalability when the complexity of the setting increases [29]. In this chapter, we present a neuro-symbolic approach for structured event recognition in sport videos. The task is out of reach of popular neuro-symbolic frameworks like DeepProbLog, because of the computational complexity given by number of frames in a video combined with the temporal constraints of the background knowledge. We tackle the problem by combining neural predictions on individual frames with a mixed integer linear programming formulation enforcing satisfaction of (soft) temporal constraints from the background knowledge and similarity with the neural outputs. The approach is fully differentiable and end-to-end trainable. Our experimental evaluation shows how the neuro-symbolic approach provides substantially more accurate predictions with respect to a fully neural solution, with the additional feature of guaranteeing that predictions satisfy existing hard constraints.

Quite remarkably, the approach is capable of predicting the sequence of atomic events that constitute a structured event even without having any supervision on them, by simply leveraging background knowledge in terms of duration constraints to guide the learning of the underlying neural network. The rest of the chapter is structured as follows: Section 4.2 formally defines the problem; Section 4.3 describes our proposed approach; Section 4.4 presents the experimental setting; Section 4.5 reports the experimental results. Finally, Section 4.6 draws some concluding remarks.

4.2 Problem Definition

Our problem can be summarized as follows: Given a data sequence $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{l}$ of real-value tensors $\boldsymbol{x}_i$ and some background knowledge $\mathcal{K}$ about the relationship between structured and atomic events, we are interested in providing a description of the atomic and structured events that are happening during the sequence. Let us now specify all the details of the problem. To represent background knowledge about structured and atomic events we use a variation of the event calculus based on First Order Logic. Let $\mathcal{L}$ be the first order language that we introduce in Subsection 3.3 and $\mathcal{K}$ a knowledge base that expresses general knowledge about the event types in $\mathcal{E}$. We split functors symbols $\mathcal{E}$ in two disjoint sets $\mathcal{A}$ and $\mathcal{S}$ that represent atomic and structured events, respectively. Given a data sequence $\boldsymbol{X} = \{\boldsymbol{x}_i\}_{i=1}^{l}$, we have to find an (herbrand) interpretation $\mathcal{I}$, such that $\mathcal{I} \models \mathcal{K}$, and such that $\mathit{happens}(e, t_1, t_2) \in \mathcal{I}$ if and only if the data sub-sequence $\boldsymbol{X}_{t_1:t_2} = \{\boldsymbol{x}_i\}_{i=t_1}^{t_2-1}$ shows that an event of type e is happening. Intuitively, $\mathcal{I}$ describes the type and the class of the events happening in $\boldsymbol{X}$ and when they happened.

Example 4. *Let* $\boldsymbol{X}$ *be a video where a person is doing a long jump (structured event) in the interval* $[1, 25]$. *The background knowledge contains the fact that a long jump can be decomposed into a sequence of three atomic events: run, jump and sit, which can be expressed by the following formula:*

$$\begin{aligned}\forall b_{hj} e_{ij} (\mathit{happens}(\mathit{longjump}, b_{hj}, e_{hj}) \leftrightarrow \exists\, & b_r, e_r, b_j, e_j, b_s, e_s(\\ & \mathit{happens}(\mathit{run}, b_r, e_r) \wedge \mathit{happens}(\mathit{jump}, b_j, e_j) \wedge \mathit{happens}(\mathit{sit}, b_s, e_s) \wedge \\ & b_r = b_{hj} \wedge e_r = b_j \wedge e_j = b_f \wedge e_f = e_{hj}))\end{aligned}$$

Two examples of interpretations that satisfy the above constraints are:

$$\begin{aligned}
\mathcal{I}_1 = \{&\mathit{happens}(\mathit{longjump}, 1, 25), \mathit{happens}(\mathit{run}, 1, 15), \mathit{happens}(\mathit{jump}, 15, 21),\\
&\mathit{happens}(\mathit{sit}, 21, 25)\}\\
\mathcal{I}_2 = \{&\mathit{happens}(\mathit{longjump}, 1, 25), \mathit{happens}(\mathit{run}, 1, 17), \mathit{happens}(\mathit{jump}, 17, 22),\\
&\mathit{happens}(\mathit{sit}, 22, 25)\}\\
&\vdots
\end{aligned}$$

Since we may have more than one interpretation that satisfy $\mathcal{K}$, we define a cost function $c : I \rightarrow R$ and select the interpretation I_c^* with the minimum cost:

$$I_c^* = \operatorname*{argmin}_{I_c \models \mathcal{K}} c(I_c)$$

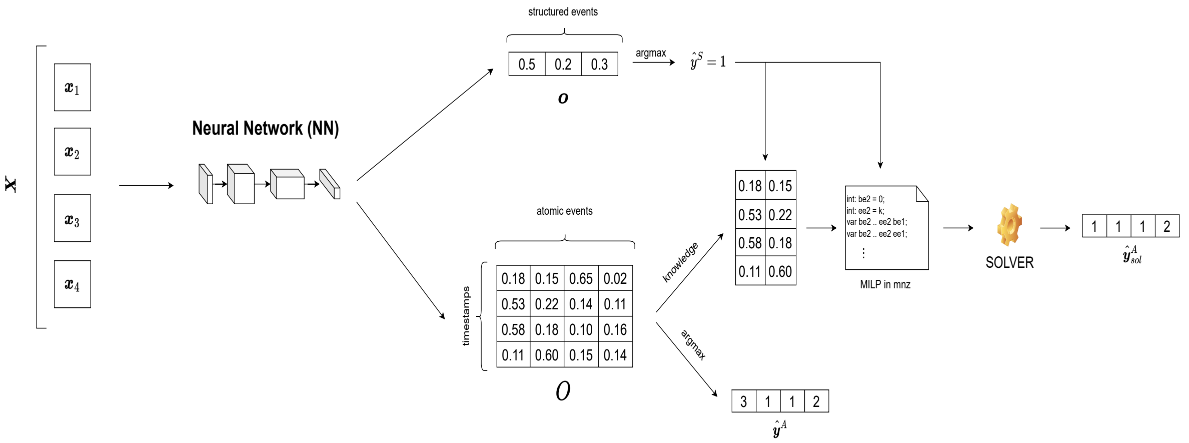
In order to find I_c^*, we define a neuro-symbolic approach that combines low-level neural processing with high level reasoning in terms of background knowledge on the events. The kind of supervision we provide to train a neuro-symbolic model in order to find I_c^* is.

$$\left\{\boldsymbol{X}^{(i)}, G_a^{(i)}\right\}_{i=1}^{n}$$

where $G_a^{(i)}$ is a set of ground atoms which are true in $\boldsymbol{X}^{(i)}$ (i.e., we have $G_a^{(i)} \subset \mathcal{F}_p^{(i)}$). Supervision is always assumed to be partial, including the case in which supervision is limited to structured events, and atomic events need to be learned in a fully unsupervised way. See experiments for the details.

4.3 Proposed Approach

Our objective consists in finding an interpretation I that has to explain what happened in $\boldsymbol{X}$ both in terms of structured and atomic events. In order to achieve it, we have to recognize the classes of structured and atomic events happening in $\boldsymbol{X}$ and the (interval of) time where they occurred. To achieve this objective, we use an end-to-end differentiable neuro-symbolic approach that combines low level processing of a neural network with a logic layer that leverages knowledge about structured and atomic events. An overall overview of our neuro-symbolic approach is depicted in Figure 4.1. As can be seen by looking at the figure, the first step consists in passing $\boldsymbol{X}$ to a neural network NN. NN may be any kind of network (e.g., Convolutional, RNN and LSTM) and has two different heads, one for structured and one for atomic events. The head for the structured events returns as output a vector $\boldsymbol{o}$ where $o_i \in [0, 1]$, with $i = 1, \ldots, k$ (assuming k is the number



```
int: be2 = 0;
int: ee2 = k;
var be2 .. ee2 be1;
var be2 .. ee2 ee1;
  ⋮
```

Figure 4.1: Inference steps of our neuro-symbolic approach on a data sequence of length 4, with 3 structured events and 4 atomic events. The structured event of class 1, which is the one predicted by the NN, is defined in terms of the sequence of atomic events 1 and 2, respectively.

of structured events), is the probability of the $i-th$ structured event. On the other hand, the head for the atomic events returns a matrix $O \in [0,1]^{l \times n}$ where entry $O[i,j]$, with $i = 1, \ldots, l$ and $j = 1, \ldots, n$ (assuming l and n are the length of $\boldsymbol{X}$ and the number of atomic events, respectively), represents the probability that event j happens at timestamp i. The predicted structured event for a video $\boldsymbol{X}$ is the one maximizing the probability of the corresponding output head of NN, i.e.:

$$\hat{y}^{\mathcal{S}} = \underset{i=1,\ldots,k}{\text{argmax}}\ o_i$$

In principle, the sequence of atomic events could be predicted in a similar fashion by maximizing for each frame the probability of the atomic event head of NN at that frame, i.e.:

$$\hat{y}_i^{\mathcal{A}} = \underset{1 \leq j \leq l}{\text{argmax}}\ O[i,j] \tag{4.1}$$

Indeed, this is how our fully-neural baseline works. However, the vector $\hat{\boldsymbol{y}}^{\mathcal{A}}$ of atomic event predictions for the frames of a video $\boldsymbol{X}$ may contain inconsistencies (e.g., predicting the atomic event *jump* as part of a *javelinthrow* structured event, predicting *fall* before *jump* within a *highjump*, or even predicting a *jump* that is much longer than the *run* that precedes it). Our neuro-symbolic architecture prevents these inconsistencies by combining neural network predictions with hard and soft constraints provided by the domain knowledge. The domain knowledge we exploit is quite simple, and provides hard constraints determining the sequence of atomic events that constitute a structured event, and soft constraints about minimal and maximal duration of each atomic event and relative duration between atomic events making up a structured event. Table 4.1 reports an example of the hard constraints that we consider for the *javelinthrow* structured event. Similar constraints are generated for the other structured events. Given the structured event $\hat{y}^{\mathcal{S}}$ predicted by NN, the corresponding sequence of atomic events is computed by solving a MILP problem encoding the (hard and soft) constraints combined with a scoring function measuring the compatibility of the sequence of atomic events with the NN outputs O. The MILP problem for a structured event (we have a separate problem for each possible structured event) is defined as follows:

$$\begin{aligned} \underset{V}{\text{minimize}} \quad & -f(V,O) + \sum_{j=1}^{m_s} \xi_t c_j(V) \\ \text{subject to} \quad & h_i(V) \quad \forall\, i = 1, \dots, m_h \end{aligned} \tag{4.2}$$

Here V is a sequence of triplets (a, b, e), where $a \in \mathcal{A}$ is an atomic event, $b, e \in \mathbb{N}$ are the starting and ending frames of the event respectively. The number of atomic events is determined by the structured event being modelled. The scoring function $f(V, O)$ computes the compatibility of V with O as follows:

$$f(V,O) = \sum_{(a,b,e)\in V} \left(\sum_{i=b}^{e} O[i,a] - \sum_{j=1}^{b-1} O[j,a] - \sum_{j=e+1}^{l} O[j,a] \right)$$

It basically computes the sum of the probabilities of each atomic event in the range in which it is predicted, and subtracts its probability outside of this range (l is the overall length of the video clip).

The soft constraints $c_j(V)$ encode duration ranges for atomic events or combinations of atomic events. For instance, the constraint that the sum of the durations of *run* and *jump* should be within the sum of the maximal and minimal durations respectively is formalized as follows:

$$\begin{aligned} & \min(|d_1 + d_2 - max_{run} - max_{jump}|, |d_1 + d_2 - min_{run} - min_{jump}|) \\ & \text{where:} \\ & \quad d_1 = e_1 - b_1 + 1, \; d_2 = e_2 - b_2 + 1 \\ & \quad a_1 = run, \; a_2 = jump \end{aligned}$$

The hard constraints $h_i(V)$ encode temporal relations between atomic events and with respect to the structured event. See Table 4.1 for examples. Intuitively, the solutions of the MILP problem for the predicted structured event $\hat{y}^{\mathcal{S}}$ where only hard contraints $h_i(V)$ are

Hard Constraints	
Generic Constraints (assuming k atomic events)	
$e_i > b_i \quad \forall\, i$	Events should end after they began
$b_1 = 1 \wedge e_k = l$	Sequence of atomic events should span the whole clip
$e_i = b_{i+1} - 1 \quad \forall\, i \in 0 \ldots l-1$	No gap among consecutive events
Specific Constraints (for the *javelinthrow* structured event)	
$a_1 = \mathit{run} \wedge a_2 = \mathit{throw}$	*javelinthrow* is a *run* followed by a *throw*
$d_1 > d_2$	*run* should take longer than *throw*

Table 4.1: Example of hard constraints for the *javelinthrow* structured event, divided into generic constraints that hold for any structured event, and those specific of the *javelinthrow* event.

considered, provide a set of candidate interpretations for $\boldsymbol{X}$. By including the objective $f(V, O)$ and the soft constraints $c_j(V)$ we obtain the interpretation with the minimum cost for $\boldsymbol{X}$ (i.e, Y_c^*). The label vector $\hat{y}^{\mathcal{A}}_{sol}$ in Figure 4.1, which is the neuro-symbolic counterpart of $\hat{y}^{\mathcal{A}}$ in Eq. 4.1, is obtained by "unrolling" the optimal V into an atomic label for each frame in its predicted range.

Example 5. *Let* $\boldsymbol{X}$ *a video of length 20 where a person is performing a structured event, but we do not know which kind of structured event. Now, suppose we have, in addition to the structured event highjump of example 4, the structured event javelin throw and that the background knowledge contains the fact that a javelin throw can be decomposed into a sequence of two atomic events: run and jump, which can be expressed by the following formula:*

$$\begin{aligned}\forall b_{jt} e_{jt} (\mathit{happens}(\mathit{javelinthrow}, b_{jt}, e_{jt}) \leftrightarrow \exists\, b_r, e_r, b_t, e_t, (&\\ \mathit{happens}(\mathit{run}, b_r, e_r) \wedge \mathit{happens}(\mathit{throw}, b_t, e_t) \wedge&\\ b_r = b_{jt} \wedge e_r = b_t \wedge e_t = e_{hj}))&\end{aligned}$$

Also, suppose that the head of NN for structure events predicted $\hat{y}^{\mathcal{S}} = \mathit{javelinthrow}$ *for* $\boldsymbol{X}$*. If we solve the MILP for the javelin throw where only hard constraints are considered (for example the ones in Table 4.1), we have that some of the candidate interpretations*

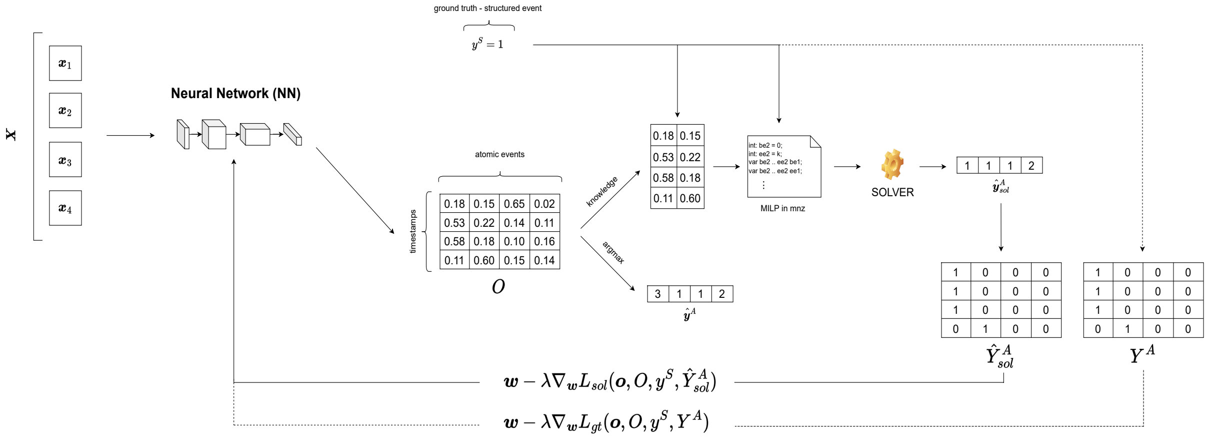


Figure 4.2: Training of our neuro-symbolic approach.

will be:

$$\mathcal{I}_1 = \{happens(javelinthrow, 1, 21), happens(run, 1, 13), happens(throw, 13, 21)\}$$
$$\mathcal{I}_2 = \{happens(javelinthrow, 1, 21), happens(run, 1, 17), happens(throw, 17, 21)\}$$
$$\vdots$$

By considering and solving the whole MILP, we obtain I_c^*. Continuing the example, if we have a soft constraint which penalizes interpretations having short duration for run, we have that $I_c^* = I_2$, that corresponds to the solution $V = [(run, 1, 17), (jump, 17, 21)]$ of Problem 4.2.

Figure 4.2 shows the training process of the architecture. As we assume that the ground-truth $y^{\mathcal{S}}$ of $\boldsymbol{X}$ is available at training time, the head for the structured events is not used anymore to predict $\hat{y}^{\mathcal{S}}$, but the ground-truth itself is used instead. Furthermore, if the ground-truth for the atomic events ($Y^{\mathcal{A}} \in R^{l \times n}$) is also available, we use it to train the head of the atomic events. If this information is not available, we use pseudo-labels generated from the architecture. The generation of such pseudo-labels consists of an inference step in the currently trained architecture as per Figure 4.1, with the only difference that the structured event is given by the ground-truth $y^{\mathcal{S}}$ rather than the NN output. The atomic event prediction vector $\hat{\boldsymbol{y}}^{\mathcal{A}}_{sol}$ is turned into a binary label matrix $\hat{Y}^{\mathcal{A}}_{sol} \in \{0, 1\}^{l \times n}$ by one-hot encoding atomic labels (i.e., $\hat{Y}^{\mathcal{A}}_{sol}[i, j]$ is set to 1 if $j = \hat{y}^{\mathcal{A}}_{sol}[i]$ and 0 otherwise). Then, we define two losses:

$$L_{gt}(\boldsymbol{o}, O, y^{\mathcal{S}}, Y^{\mathcal{A}}) = L(\boldsymbol{o}, y^{\mathcal{S}}) + L(O, Y^{\mathcal{A}}) \tag{4.3}$$
$$L_{sol}(\boldsymbol{o}, O, y^{\mathcal{S}}, \hat{Y}^{\mathcal{A}}_{sol}) = L(\boldsymbol{o}, y^{\mathcal{S}}) + L(O, \hat{Y}^{\mathcal{A}}_{sol}) \tag{4.4}$$

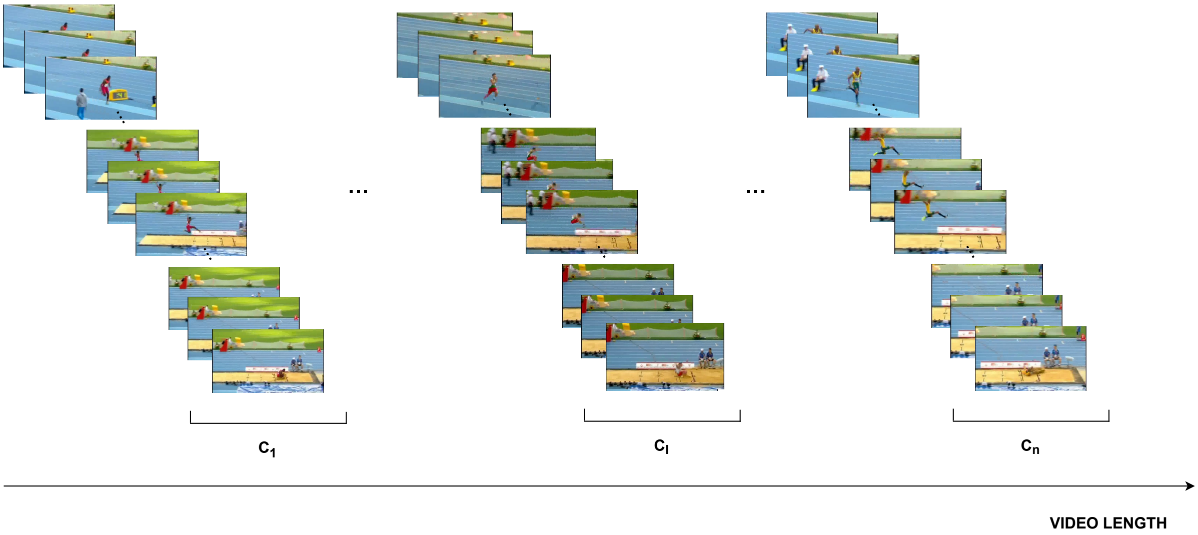


Figure 4.3: Extraction of clips of structured events from an untrimmed video.

Where Loss 4.3 refers to the case where both ground-truth (structured and atomic) are available, while Loss 4.4 refers to the case where the ground-truth for structured events is available and the ground-truth for the atomic events is not, and, then, we use the pseudo labels. In order to train NN to recognize both kinds of the events, we minimize, depending on the case, one of the aforementioned loss and use gradient descent to update its weights.

4.4 Experimental setting

Our experimental setting has the aim to show if our proposed neuro-symbolic approach leads to an advantage in the recognition of both structured and atomic events with respect to a fully neural approach, when both approaches are trained with weak and limited amount of supervision in terms of events. In details, we want to see how the knowledge is able to compensate in the case when no or few and potentially noisy labels for events are available. To achieve this objective, we first need a dataset of structured and atomic events. We build such dataset from the Multi-THUMOS untrimmed video dataset [93] that has been used widely for temporal action detection in untrimmed videos [24, 23, 82] . Since in [93], there is no explicit distinction between structured and atomic events, we define it and splits the events according. In particular, we consider as structured events those events that can be decomposed as a sequence of other (atomic) events, and cut each video into clips corresponding to structured events (Figure 4.3).

For each structured event, we do not consider all the clips, but we remove those clips where some of the atomic events defining the structured event are not present (e.g, replay).

The structured and atomic events we consider are shown in Appendix A.2. Then, after building the dataset, we define the scenario. The scenario consists of clips of different lengths where, in each clip, a person is performing one (and only one) structured event among the ones reported in Appendix A.2.

The learning setting we consider to evaluate the fully neural approach and our proposed neuro-symbolic approach consists in having full supervision at level of structured events and limited and potentially noisy (e.g., overlapping between atomic events) supervision in terms of atomic events. The kind of supervision we provide is as follows:

$$\begin{aligned}
&\{\mathit{happens}(\mathit{highjump}, 1, 50),\ \mathit{happens}(\mathit{run}, 1, 31),\ \mathit{happens}(\mathit{jump}, 31, 45),\\
&\quad \mathit{happens}(\mathit{fall}, 45, 50)\}\\
&\{\mathit{happens}(\mathit{hammerthrow}, 1, 30),\ \mathit{happens}(\mathit{windup}, 1, 15),\ \mathit{happens}(\mathit{spin}, 10, 25),\\
&\quad \mathit{happens}(\mathit{release}, 25, 30)\}\\
&\{\mathit{happens}(\mathit{javelinthrow}, 1, 30)\}
\end{aligned}$$

The first video is an example of noiseless labeling with full supervision on both structured and atomic events. The second video is fully supervised too, but atomic supervision is noisy, as there is an overlap between the *windup* and *spin* atomic events (this type of overlapping labeling is not rare in the dataset). The third video is a case where supervision is only provided at the structured event level, and there is no supervision on atomic events.

In this setting, we are interested in observing how the prediction in terms of atomic and structured events change when increasing the availability of data for atomic events. Furthermore, in the case of the neuro-symbolic approach, we are interested in seeing how complementing the supervision coming from the dataset with the supervision coming from the knowledge affects the predictions of the overall model. A noteworthy case is the one where no direct supervision at level of atomic events is provided at all, and then the model is completely trained with the supervision coming from the knowledge. The underlying model we use for both approaches is the one described in [82] where features extracted from a pre-trained two-stream I3D [14] are given as input in order to predict a matrix of events (more details about the model can be found in Appendix A.3). Differently from [82], we distinguish between structured and atomic events and consider two separate heads, as discussed in the previous section. About the training, we train the model for 20 epochs with learning rate of 1e−3 and weight decay of 1e−6, using Adam as optimizer. We also created a validation set of 10% of the training data in order to select the best model.

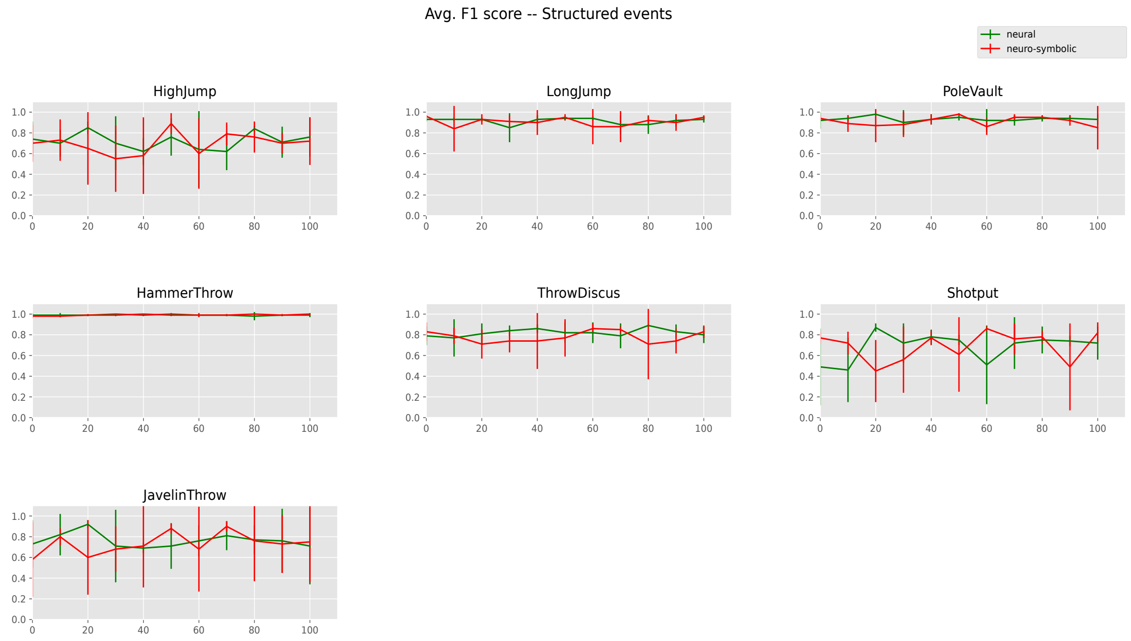


Figure 4.4: F_1 scores on structured events averaged over 5 runs for the fully neural (green) and the neuro-symbolic (red) approaches, for increasing amount of supervision on atomic events.

4.5 Results

In this section, we show and compare the results of the fully neural approach with respect to our proposed neuro-symbolic approach on the task described in Section 4.4. Figure 4.4 reports average F_1 scores of structured event prediction over 5 runs, for an increasing amount of supervision in terms of atomic events (from 0 to 100%). Note that in all cases supervision in terms of structured events is always provided. The green curve indicates the fully neural baseline, while the red curve indicates our neuro-symbolic approach. Results indicate that unsurprisingly, when there is full supervision on the structured event the addition of knowledge does not help in its identification.

Figure 4.5 reports average F_1 scores for the prediction of atomic events, again for a growing amount of supervision at the atomic level. The difference between the fully neural and the neuro-symbolic approach is striking. Substantial improvements of the neuro-symbolic approach can be observed for almost all atomic events. Only for the atomic events *run* and *jump*, we can see that the fully neural approach is really close to our approach. This is probably due to the temporal duration of these events, that is substantially higher than that of the others. This implies that a reasonable number of frames labelled as *run* and *jump* will be available for the neural network even with a small

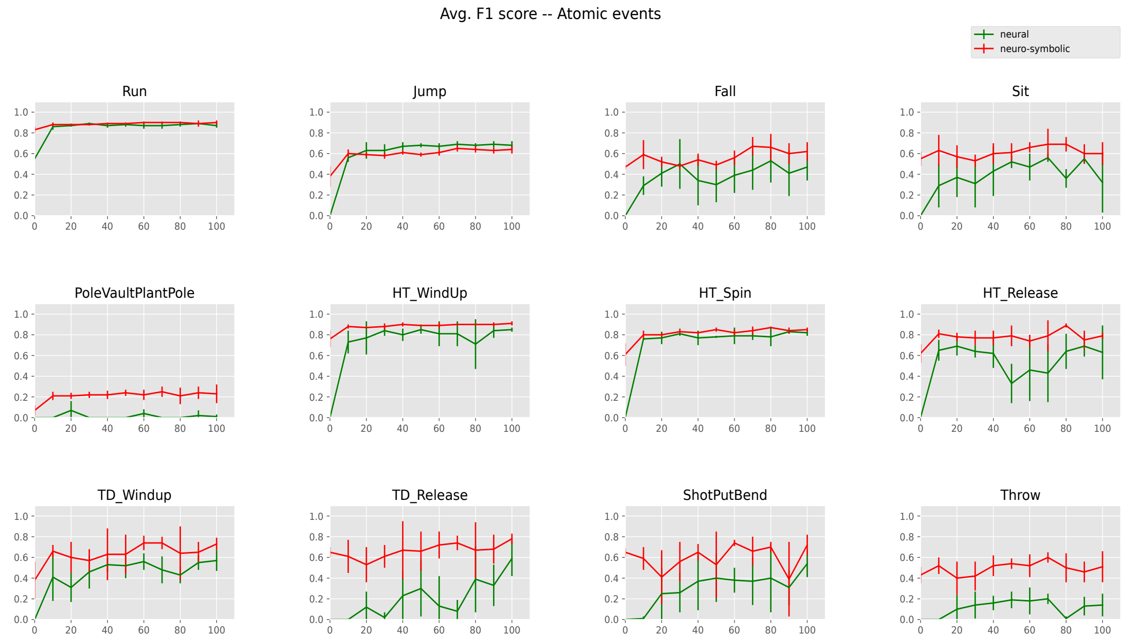


Figure 4.5: F_1 scores on atomic events averaged over 5 runs for the fully neural (green) and the neuro-symbolic (red) approaches, for increasing amount of supervision on atomic events.

fraction of labelled videos. On the other hand, atomic events like *release* and *throw* have a performance improvement that goes up as 60% and 50% respectively.

As stated in Section 4.4, a particularly significant case is the one where no direct supervision at all is provided at the level of atomic events. This corresponds to the leftmost point in the figures. The F_1 score of the fully neural approach is close to zero for almost all atomic events, corresponding to random guessing. On the other hand, the F_1 scores of the neuro-symbolic approach are often comparable to those of a fully supervised setting, showing how knowledge can be exploited to completely bypass the need for human supervision at the frame level, with major implications in terms of applicability and training costs.

Figure 4.6 shows some representative examples of the labeling provided by the two approaches highlighting the differences in prediction consistency between the two, both in terms of atomic events being detected and relative duration. Note that results are achieved with 100% supervision on atomic events, and highlight the importance of knowledge in guaranteeing the consistency of predictions. The figure shows prediction for all frames in a video for three videos, a *highjump*, a *hammerthrow* and a *longjump* respectively. For each video, we compare ground truth atomic labels (yellow) with fully neural predictions (green) and neuro-symbolic ones (green). In the *highjump* case (top), the fully neural approach completely misses the last event and mispredicts it as part of the *jump* event. On the other hand, the neuro-symbolic approach correctly detects the last event a *fall*, and has a better estimate of the duration of each event. In the *hammerthrow* case (middle), the fully neural approach detects a *run* event, that cannot be part of a *hammerthrow*, and misses the *release* event. Again, the neuro-symbolic approach provides quite accurate estimates of the duration of each event, despite the short duration of *release* with respect to *windup* and *spin*. Finally, in the *longjump* case (bottom), the neural approach correctly identifies the initial *run*, but breaks the rest of the video into a sequence of short *jump*, *fall*, *sit* and even *run* events which is completely inconsistent, while the neuro-symbolic approach again accurately recovers both sequence and duration of the atomic events.

4.6 Conclusion

In this work, we have proposed a neuro-symbolic approach for (structured and atomic) event recognition where knowledge about the events and their temporal relations is exploited both at training and inference time. We have instantiated our approach on a real-world scenario consisting of clips of sports events. Our experimental evaluation showed how our neuro-symbolic solution achieves substantial improvements over a fully neural baseline in terms of recognition of the atomic events that constitute a structured event. The approach is capable of learning to detect atomic events even with no supervision at all on them during training, by simply combining supervision on structured events, low-level neural processing and knowledge.

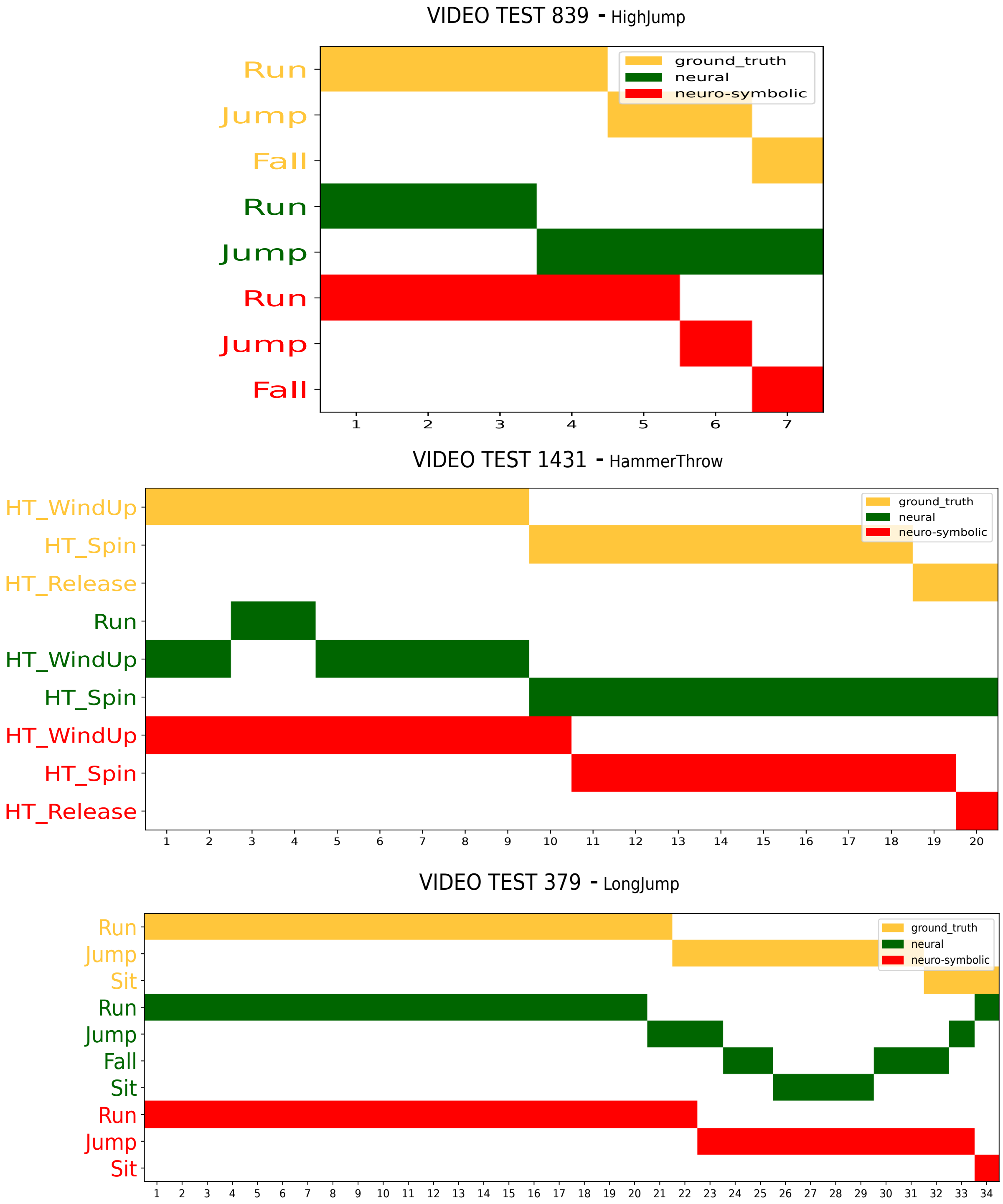


Figure 4.6: Prediction of the sequence of atomic events for three clips representing a *highjump* (top), a *hammerthrow* (middle) and a *longjump* (bottom) respectively. Ground truth is in yellow, while the neural and neuro-symbolic predictions are in green and red respectively. Both models were trained with 100 % supervision on the atomic events. Clips were selected to show examples of inconsistencies in neural predictions.

Chapter 5

Interval Logic Tensor Networks

In this chapter, we introduce Interval Real Logic (IRL)[1], a two-sorted logic that interprets knowledge such as sequential properties (traces) and event properties using sequences of real-featured data. We interpret connectives using fuzzy logic, event durations using trapezoidal fuzzy intervals, and fuzzy temporal relations using relationships between the intervals' areas. We propose Interval Logic Tensor Networks (ILTN), a neuro-symbolic system that learns by propagating gradients through IRL. In order to support effective learning, ILTN defines smoothened versions of the fuzzy intervals and temporal relations of IRL using the softplus activation. We show that ILTN can successfully leverage knowledge expressed in IRL in synthetic tasks that require reasoning about events to predict their fuzzy durations. Our results show that the system is capable of making events compliant with background temporal knowledge.

5.1 Introduction

Up to this point, we have shown that integrating commonsense and structural knowledge about events and their relationships can significantly enhance neural approaches for event detection. In addition, background knowledge has been shown to improve the detection of atomic (chapters 3 and 4), and complex events especially when training data is limited [94] Some approaches use temporal logic, such as LTLf, to embed temporal properties in deep-learning architectures processing image sequences [84].

To the best of our knowledge, all existing methods that incorporate background temporal knowledge in event detection adopt a point-wise approach, defining events based on properties that hold (or do not hold) at specific time points during the event's duration. However, the knowledge representation and formal ontology literature advocates

[1] my work has been focused mostly on the theoretical part of IRL

for event-centric representations, where events are treated as "first-class citizens" with properties that cannot be expressed solely in terms of time-point properties [50, 4, 64].

The traditional perspective of event representation characterizes events as crisp entities and represents the duration of an event, which is the time span during which it occurs, as a convex subset of integers or real numbers. However, this approach does not account for events that have smooth beginnings or endings, such as a snowfall. Furthermore, even crisp events can benefit from fuzzy semantics in representing relations between them. For example, the statement "Darwin (1809-1882) lived before Einstein (1879-1955)" is not as true as "March 13 comes before March 14," but it is also not entirely false. To address this limitation, knowledge representation formalisms have been proposed for fuzzy intervals and fuzzy relations between them [69, 76].

This chapter introduces a novel logical framework that enables the specification of dynamically changing propositions, as well as properties and relations between events. We refer to this framework as Interval Real Logic, which is an extension of Real Logic [78]. This logic is designed to capture knowledge properties and relations between objects that evolve over time, including properties and relations between events. Interval Real Logic is interpreted in the domain of real-data sequences, where objects are associated with trajectories, and events are associated with the objects that participate in the event, as well as the temporal interval during which the event occurs.

In addition, the chapter introduces the differentiable implementation of Interval Real Logic in a neuro-symbolic architecture, Interval Logic Tensor Networks (ILTN), to detect events from data sequences using background knowledge expressed in Interval Real Logic. To effectively propagate gradients through the logic, we propose modified trapezoidal fuzzy membership functions and temporal relations for fuzzy intervals that overcome vanishing gradient issues. We present a prototype implementation of ILTN and conduct basic experiments that yield promising and positive results.

The rest of the chapter is organised as follows: Section 5.2 presents related work on fuzzy temporal knowledge and neuro-symbolic approaches for event detection. Section 5.3 defines the language and the semantics of ILTN. Section 5.4 defines fuzzy trapezoidal intervals and their temporal relations. In Section 5.5, the neural architecture used to predict fuzzy events is described. In Section 5.6, the results on artificial experiments are discussed. Finally, in Section 5.7 conclusions are drawn.

5.2 Related Work

Modeling and reasoning about temporal knowledge is a well-studied problem [43, 4, 5, 42]. Temporal logics like Linear Temporal Logic (LTL) [72] and Computational Tree Logic

(CTL) [19] assume that the underlying (temporal) information is crisp, and do not consider that the knowledge may be characterized by vagueness and uncertainty. Following the seminal work of [95] on fuzzy sets, different works have been proposed to model both vagueness and uncertainty of temporal knowledge when this is expressed in terms of events and their relations via a fuzzy interval-based temporal model [30, 65, 69, 75]. These works however are not capable of processing low level information in an efficient way, and do not consider any learning. Indeed, fuzzy event recognition applications [44, 27, 63] simply rely on a (fuzzy) rule-based decision system. Recently, neuro-symbolic approaches [39], which integrate sub-symbolic and symbolic reasoning and allow to effectively integrate learning and reasoning, have been applied in the context of event recognition. A common solution consists in introducing a symbolic layer refining the output of a pre-trained neural network [46, 47, 91, 85, 34]. In [90], the symbolic layer is replaced by a neural network trained via knowledge distillation to emulate symbolic reasoning. The drawback is that this "neuro-symbolic" layer has to be re-trained from scratch even for a slight change of the knowledge. More recently, fully end-to-end differentiable neuro-symbolic architectures have been proposed, by encoding temporal reasoning primitives into existing frameworks like DeepProbLog [86, 6] or Learning Modulo Theories [7]. However, all these approaches reason in terms of time points, making them incapable of fully expressing the properties of temporal events. The solution we propose here aims to overcome these limitations by directly focusing on temporal intervals. LTN [78] is an end-to-end neuro-symbolic approach based on fuzzy logic where prior domain knowledge is expressed in terms of Real Logic formulas and interpreted using fuzzy logic semantics. LTN has been applied successfully to solve structured tasks like semantic image interpretation [28] and to improve state of the art object classifiers [58]. A first temporal extension of LTN has been proposed by [84], where Linear Temporal Logic over finite traces (LTLf) formulas are translated to fuzzy deterministic automaton and applied to solve a sequence classification task. However, as for the other previously mentioned neuro-symbolic approaches, LTLf reasons in terms of time points and thus shares their limitations. By extending LTN to deal with (fuzzy) interval logic primitives we aim to allow them to effectively and efficiently process temporal sequences towards complex event recognition.

5.3 Interval Real Logic

Let $\mathcal{L}_t$ be a first-order language that includes terms referring to the *trajectories* of objects over time. The syntax for terms and formulas in $\mathcal{L}_t$ follows the standard syntax of first-order logic.

Similarly, let $\mathcal{L}_e$ be a first-order language, referred to as the *language of events*, which includes a set of symbols $e_1, e_2, \ldots$ each associated with an arity $m \geq 0$. The terms of $\mathcal{L}_e$ are expressed in the form $e(t_1, \ldots, t_n)$ if e has arity m and t_i's are terms in $\mathcal{L}_t$. Intuitively, $e(t_1, \ldots, t_m)$ denotes an event that involves $t_1, \ldots, t_m$ as participants. Additionally, we assume that $\mathcal{L}_e$ contains the set of binary predicates that correspond to binary relations between events.

Example 6. *Suppose that we want to describe the events that happen when two particles move in a 2D space as shown in Figure 5.1.*

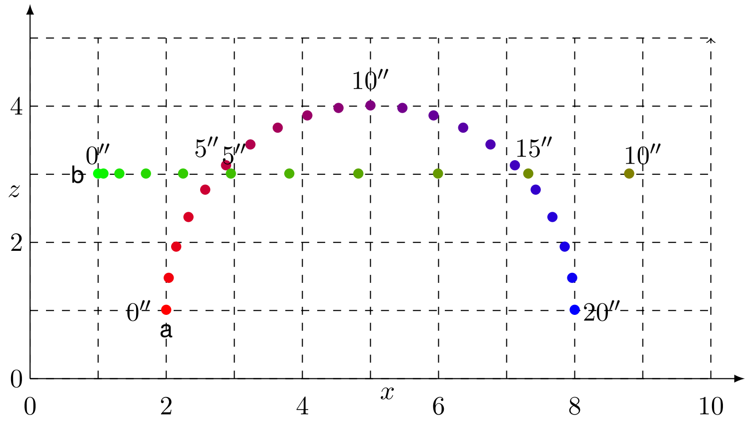


Figure 5.1: Trajectories of two particles in a 2D space. Several events happen over time. For example, around time 5″, the two particles intersect. From time 0″ to 10″, particle a rises whereas particle b accelerates from left to right. Additionally, over the whole trajectory, particle a is doing a jump while changing color.

$\mathcal{L}_t$ and $\mathcal{L}_e$ are used conjointly to describe Figure 5.1. In $\mathcal{L}_t$, the two particles are denoted by constants a *and* b*. Unary predicates such as* blue, red, *and* violet *are included to describe the particles' colors over time. The atomic formula* blue(a) *expresses that* a *is blue, with its truth value being time-dependent. To describe the proximity of the particles, $\mathcal{L}_t$ uses the binary predicate* close, *and* close(a,b) *is true around time* 5″ *and false otherwise.*

In $\mathcal{L}_e$, event symbols are used to describe the events in the figure. For example, e0(a) *can denote the jump of particle* a, e1(a) *can denote the color change of* a, *and* e2(a,b) *can denote the event of* a *and* b *intersecting. Predicates and functions on events are also included in $\mathcal{L}_e$. For example, unary predicates on events can be used to specify their types, as in the formula* Jump(e0(a)), *which states that* e0(a) *is of type jump, and* ChangeOfColor(e1(a)), *which states that* e1(a) *is of type color change.*

We require that $\mathcal{L}_e$ contains the unary functions on events and the binary relations of events shown in Table 5.1. In the table and in the rest of the paper we use ϵ (possibly with indices) to denote an event term $e(t_1, \ldots, t_m)$.

	Function symbols of $\mathcal{L}_e$
$\mathsf{before}(\epsilon)$	what happens before the starting of ϵ
$\mathsf{after}(\epsilon)$	what happens after the end of ϵ
$\mathsf{start}(\epsilon)$	the starting of ϵ
$\mathsf{end}(\epsilon)$	the end of ϵ
$[i, j]$	for $i \leq j \in \mathbb{N}$
	Allen's predicate symbols of $\mathcal{L}_e$
ϵ_1 bf ϵ_2	ϵ_1 happens before ϵ_2
ϵ_1 af ϵ_2	ϵ_1 happens after ϵ_2
ϵ_1 mt ϵ_2	ϵ_2 happens immediately after ϵ_1
ϵ_1 ol ϵ_2	the end of ϵ_1 overlaps the start of ϵ_2
ϵ_1 st ϵ_2	ϵ_1 is a starting part of ϵ_2
ϵ_1 dr ϵ_2	ϵ_1 happens during ϵ_2
ϵ_1 fin ϵ_2	ϵ_1 is an ending part of ϵ_2
ϵ_1 eq ϵ_2	ϵ_1 is equal to ϵ_2
	Other predicate symbols of $\mathcal{L}_e$
$\mathsf{H}(\epsilon)$	the event ϵ actually happened
ϵ_1 in ϵ_2	ϵ_1 is contained in ϵ_2

Table 5.1: Basic functions and relations on events

Finally, $\mathcal{L}_t$ contains a unary predicate Active that takes as input an event term. Intuitively, Active(ϵ) returns for every time step of the sequence if the event is running or not.

Example 7. *Following are some examples of formulas in $\mathcal{L}_t$ and $\mathcal{L}_e$. The atomic formula*

$$\textsf{\textit{Sunny(weather)}} \rightarrow \textsf{\textit{Happy(John)}}$$

is an example of a $\mathcal{L}_t$ formula that states that John is happy whenever it is sunny. This formula is evaluated along two traces, one for the weather and one for John, and can take different values at different time points.

The following $\mathcal{L}_e$ formula

$$\textsf{\textit{H}}(\textsf{\textit{e1(John,Mary)}}) \wedge \textsf{\textit{Meeting(e1(John,Mary))}}$$

states that a meeting between John and Mary happened.

The $\mathcal{L}_t$ formula

$$\text{Active}(\textsf{\textit{e1(John,Mary)}}) \rightarrow \textsf{\textit{Happy(John)}} \wedge \textsf{\textit{Happy(Mary)}}$$

expresses that during the meeting between John and Mary, they were both happy.

The $\mathcal{L}_e$ formula

$$\forall_t x, y.\textsf{\textit{Meeting}}(\textsf{\textit{e1}}(x,y)) \rightarrow \textsf{\textit{e1}}(x,y) = \textsf{\textit{e1}}(y,x)$$

states that in a meeting event, the roles of the participants are symmetric. Notice that the quantification is on trace variables (not on the events). This is highlighted by the index t of the universal quantifier.

Finally, the $\mathcal{L}_e$ formula

$$\forall_e x.\textsf{\textit{Meeting}}(x) \rightarrow \exists_e y.\textsf{\textit{PrepareAgenda}}(y) \land x \ \textsf{\textit{bf}} \ y$$

expresses that before every meeting there should be an event that is the preparation of the agenda. In this case, the quantification is on event variables, indicated by the index e of the quantifier.

The semantics of the trace logic $\mathcal{L}_t$ and the event-based logic $\mathcal{L}_e$ are defined in the context of a linear discrete structure, which models the progression of time. We use the natural numbers $\mathbb{N}$ with the standard order $<$ as the reference structure for time.

5.3.1 Trace Semantics

In $\mathcal{L}_t$, terms are interpreted as (possibly infinite) sequences of data, called *trajectories*. For each time point $i \in \mathbb{N}$, an $\mathcal{L}_t$ term corresponds to a feature vector in $\mathbb{R}^n$. Specifically, a trajectory is a function $\boldsymbol{t} : \mathbb{N} \rightarrow \mathbb{R}^n$ that assigns a feature vector in $\mathbb{R}^n$ to every time point. We denote the set of trajectories with features in $\mathbb{R}^n$ as $\mathbb{T}^n$. Trace variables in $\mathcal{L}_t$ refer to variables of individuals and are associated with batches of traces. Constants and closed terms (i.e., terms without variables) in $\mathcal{L}_t$ are interpreted as single traces.

Formulas in $\mathcal{L}_t$ are evaluated at all time instants. For every time $i \in \mathbb{N}$, an $\mathcal{L}_t$ formula is associated with a truth value in the range $[0, 1]$ that represents the level of truth of the formula at that time. As a result, an $\mathcal{L}_t$ formula is interpreted as a sequence of truth values, which we refer to as a function from $\mathbb{N}$ to $[0, 1]$. The set of such functions is denoted as $\mathbb{B}$.

The formal definition of the semantics for $\mathcal{L}_t$ is based on a *grounding* function $\mathcal{G}$ that must satisfy the following conditions:

- for every variable x in $\mathcal{L}_t$, $\mathcal{G}(x) \in (\mathbb{T}^n)^b$ is a batch of trajectories with the integer size $b \geq 1$,

- for every constant $c \in \mathcal{L}_t$, $\mathcal{G}(c) \in \mathbb{T}^n$ is a single trajectory,

- for every function $f \in \mathcal{L}_t$, with arity equal to m, $\mathcal{G}(f) : \mathbb{T}^{n_1} \times \cdots \times \mathbb{T}^{n_m} \to \mathbb{T}^{n}$, that is $\mathcal{G}(f)$ maps to a function that takes m input trajectories and returns a trajectory,

- for every predicate $p \in \mathcal{L}_t$, with arity equal to m, $\mathcal{G}(p) : \mathbb{T}^{n_1} \times \cdots \times \mathbb{T}^{n_m} \to \mathbb{B}$, that is $\mathcal{G}(p)$ maps to a function that takes m input trajectories and outputs a function from time points to truth values in $[0, 1]$.

Propositional connectives are interpreted according to fuzzy logic semantics which is applied point-wise. For example, if ϕ and ψ are $\mathcal{L}_t$-formulas, then $\mathcal{G}(\phi \wedge \psi) = T(\mathcal{G}(\phi), \mathcal{G}(\psi)) = \{T(\mathcal{G}_i(\phi), \mathcal{G}_i(\phi))\}_{i \in \mathbb{N}}$, where T is a t-norm such as the product t-norm. Universal and existential quantifiers are interpreted as aggregation operators. For example, $\mathcal{G}(\forall x \phi(x)) = \{\prod_{1 \leq j \leq b} \mathcal{G}_i(\phi(\mathcal{G}_j(x)))\}_{i \in \mathbb{N}}$.

Finally, we allow a special predicate that maps from events to $\mathcal{L}_t$:

- for every event ϵ, $\mathcal{G}(\text{Active}(\epsilon)) : \mathbb{E}^{\boldsymbol{n}} \to \mathbb{B}$; $i \mapsto T(\mathcal{I}(\epsilon)(i), \mathsf{H}(\epsilon))$ where T is a t-norm. The functions $\mathcal{I}$ and H, as well as the notation $\mathbb{E}^{\boldsymbol{n}}$, are defined in Section 5.3.2. Intuitively, Active(ϵ) maps an event to a Boolean trajectory that states *when* and *if* the event happens at each timepoint of the trajectory.

5.3.2 Event Semantics

An event is seen as a potentially infinite sequence of data, (i.e., a trajectory) and a mask that indicates the duration of the event. Formally, an event $\epsilon \in \mathbb{T}^{n_1} \times \cdots \times \mathbb{T}^{n_m} \times \mathbb{B}$ consists of m traces, which are the traces of the objects involved in the event ϵ, and a Boolean trace that indicates when the event is active. Specifically, let $\mathcal{I}(\epsilon)$ denote the Boolean trace $\mathbb{B}$ that is the activation sequence of ϵ. If $\boldsymbol{n} = (n_1, \ldots, n_m)$, we denote $\mathbb{E}^{\boldsymbol{n}}$ as $\mathbb{T}^{n_1} \times \cdots \times \mathbb{T}^{n_m} \times \mathbb{B}$, which represents the space of events involving m objects, each with features in $\mathbb{R}^{n_i}$. The formal semantics of $\mathcal{L}_e$ is defined in reference to the definition of an event provided in [36] and is given in terms of a function $\mathcal{G}$ that satisfies the following restrictions.

- For every event term $e(t_1, \ldots, t_m)$, $\mathcal{G}(e(t_1, \ldots, t_m)) \in \mathbb{E}^{\boldsymbol{n}}$ where $\boldsymbol{n} = (n_1, \ldots, n_m)$ and $\mathcal{G}(t_i) \in \mathbb{R}^{n_i}$ for $1 \leq i \leq m$,

- for every $[i, j] \in \mathbb{N}$, $\mathcal{G}(i) = \{\nVdash_{n \in [i,j]}\}_{n \in \mathbb{N}}$,

- for every function symbol $f \in \mathcal{L}_e$, with arity equal to m, $\mathcal{G}(f) : (\mathbb{E}^{\boldsymbol{n}_1} \times \cdots \times \mathbb{E}^{\boldsymbol{n}_m}) \to \mathbb{E}^{\boldsymbol{n}_1 \cdots \boldsymbol{n}_m}$,

- for every predicate symbol $p \in \mathcal{L}_e$, with arity equal to m, $\mathcal{G}(p) : (\mathbb{E}^{\boldsymbol{n}_1} \times \cdots \times \mathbb{E}^{\boldsymbol{n}_m}) \to [0, 1]$.

i			0	1	2	3	4	5	6
$\mathcal{G}(\mathsf{e2(a,b)})$	$\mathcal{G}(\mathsf{a})$	x	2.0	2.04	2.15	2.33	2.57	2.88	3.24
		y	1.0	1.47	1.93	2.36	2.76	3.12	3.43
		r	1.0	0.95	0.9	0.85	0.8	0.75	0.7
		g	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		b	0.0	0.05	0.1	0.15	0.2	0.25	0.3
	$\mathcal{G}(\mathsf{b})$	x	1.0	1.02	1.08	1.18	1.32	1.5	1.72
		y	4.0	4.0	4.0	4.0	4.0	4.0	4.0
		r	1.0	0.95	0.9	0.85	0.8	0.75	0.7
		g	0.0	0.05	0.1	0.15	0.2	0.25	0.3
		b	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	$\mathcal{I}$		_	▬	🟥	🟥	🟥	🟥	_

Figure 5.2: Grounding traces and events for the particle example. e2(a,b) is the event of the two particles intersecting.

Connectives in $\mathcal{L}_e$ are interpreted using fuzzy semantics. For example, $\mathcal{G}(\phi_1 \wedge \phi_2) = T(\mathcal{G}(\phi_1), \mathcal{G}(\phi_2))$ where T is a t-norm. Quantifiers of events are interpreted by aggregation functions.

Example 8. *The first segment of the grounding $\mathcal{G}$ of the particle a of Figure 5.1 is shown in Figure 5.2*

Examples of function symbols from $\mathcal{L}_e$ include $\mathsf{before}(\epsilon)$ or $\mathsf{start}(\epsilon)$, whereas examples of predicate symbols of $\mathcal{L}_e$ include $\mathsf{H}(\epsilon)$ (unary symbol) and ϵ_1 bf ϵ_2 (binary symbol in infix notation). These symbols are intuitively described in Table 5.1. Their actual grounding is discussed in Section 5.4.

5.4 Fuzzy Intervals and Relations

As previously mentioned, $\mathcal{I}(\epsilon)$ denotes the activation sequence of an event in $\mathbb{B}$. Notice that $\mathcal{I}(\epsilon)$ is a fuzzy subset of $\mathbb{N}$. A requirement imposed in [36] is that $\mathcal{I}(\epsilon)$ must be an interval, i.e., a convex subset of time points. However, in this paper, we consider the fact that such an interval is a fuzzy interval. Therefore, we propose imposing constraints on the shape of such a subset to be a *trapezoidal fuzzy number* [1].

Definition 1 (Fuzzy interval). *A fuzzy interval is a fuzzy set $I : \mathbb{R} \to [0, 1]$ such that* *there exists $a \leq b \leq c \leq d \in \mathbb{R}$.*

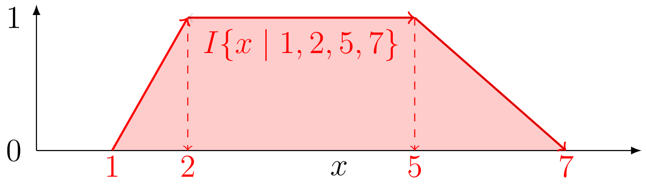
$$I(x) = \begin{cases} \frac{x-a}{b-a} & \text{if } x \in (a,b), \\ 1 & \text{if } x \in [b,c], \\ \frac{x-d}{c-d} & \text{if } x \in (c,d), \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

We also allow special cases of semi-infinite intervals, which we use to define the **before** *and* **after** *operators in Section 5.4.1.*

Left-infinity *A left-infinite fuzzy interval is characterized by the parameters $I = \{x \mid -\infty, -\infty, c, d\}$.*

Right-infinity *A right-infinite fuzzy interval is characterized by the parameters $I = \{x \mid a, b, +\infty, +\infty\}$.*

Example 9.



With this restriction, we impose that the activation function of every event $\mathcal{I}(\epsilon)$ is such that there is a trapezoidal fuzzy interval $I = \{x \mid a, b, c, d\}$ such that $\mathcal{I}_n(\epsilon) = I(n)$ for every $n \in \mathbb{N}$. For ease of notation, in the rest of the paper, we will commonly denote an interval simply by its four parameters $I = (a, b, c, d)$.

5.4.1 Basic Operations on Fuzzy Intervals

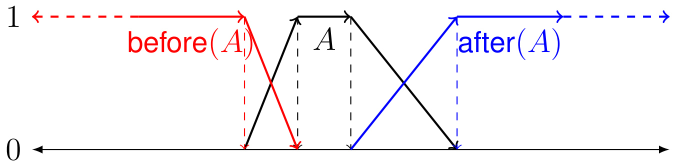
To provide the semantics for the functions and relations of $\mathcal{L}_e$, we first define a set of basic operations on fuzzy intervals. Our operations are inspired by [69] who defines such operations on any convex and non-convex interval. We simply specialize them on trapezoidal fuzzy intervals.

Duration

The duration of a trapezoidal fuzzy interval $A = (a, b, c, d)$, denoted by duration(A) or $|A|$ is equal to $\int_{-\infty}^{+\infty} A(x)dx$. If A is finite then $|A| = \frac{(c-b)+(d-a)}{2}$, otherwise $|A| = \infty$.

Before and After

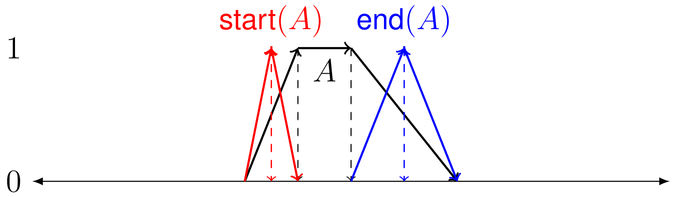
If $A = (a, b, c, d)$ then $\mathsf{before}(A) = (-\infty, -\infty, a, b)$ and $\mathsf{after}(A) = (c, d, +\infty, +\infty)$ and



Start and End

If $A = (a, b, c, d)$ is left-finite then $\mathsf{start}(A)$ is defined as $(\chi - \frac{\delta}{2}, \chi, \chi, \chi + \frac{\delta}{2})$, such that $\chi = \frac{a+b}{2}$ and $\delta = \max(\frac{b-a}{2}, \delta_{\min})$ where $\delta_{\min}$ is a small positive value to account for the crisp case $a = b$.

Similarly, $\mathsf{end}(A) = (\chi - \frac{\delta}{2}, \chi, \chi, \chi + \frac{\delta}{2})$ such that A is right-finite, $\chi = \frac{c+d}{2}$ and $\delta = \max(\frac{d-c}{2}, \delta_{\min})$.



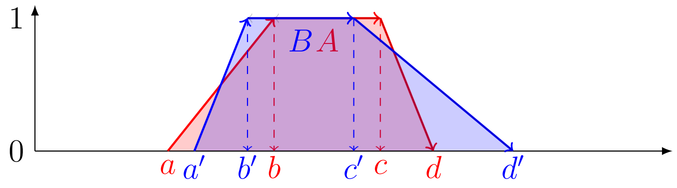
5.4.2 Relations between Fuzzy Intervals

Ohlbach [69] defines interval-interval relations by computing the integral of point-interval relations over the points in a set. To avoid the complexity associated with the integrals, and to be more compliant with Allen's definition in the crisp case, we define new relations based on simplified containment ratios.

In these definitions, the temporal relations take precedence over the fuzzy conjunction $\wedge$. For general fuzzy intervals, $|A \cap B|$ can be hard to compute. However, with trapezoidal intervals, the calculation of $|A \cap B|$ is derivable analitically by solving simple linear constraints system. We show how this is done in the following subsection.

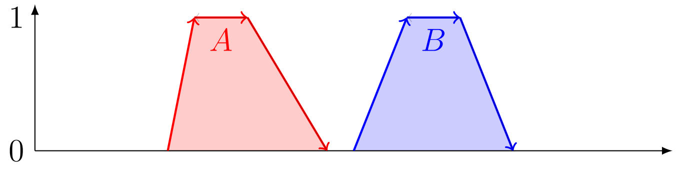
5.4.3 Area Intersection

Let us calculate Area$(A \cap B)$ for any two finite intervals $A = (a, b, c, d)$ and $B = (a', b', c', d')$. Without loss of generality, suppose that $a \leq a'$. Developing an explicit formula to compute Area$(A \cap B)$ is not immediate as the shape of $A \cap B$ can be a polygon with a varying number of edges (at most 6). For example:



We propose to first determine the vertices of the shape $A \cap B$, and then compute the area of the shape using the shoelace formula.

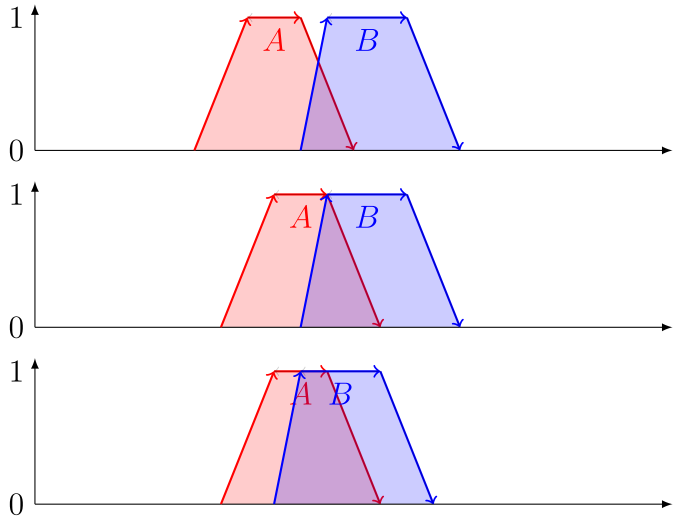
Empty intersection First, we dismiss the case $d \leq a'$, in which the two intervals do not intersect. Area$(A \cap B) = 0$.



In the rest of the section, we assume that an intersection always exists.

Bottom vertices We call bottom vertices of the shape $A \cap B$, the ones on the line $y = 0$. There are always two. As $a \leq a'$, $(a', 0)$ is always a vertex of the shape. The second vertex is $(\min(d, d'), 0)$.

Top vertices We call top vertices the ones on the line $y = 1$. There can be zero, one, or two top vertices that delimit $A \cap B$, as shown in the below figures:

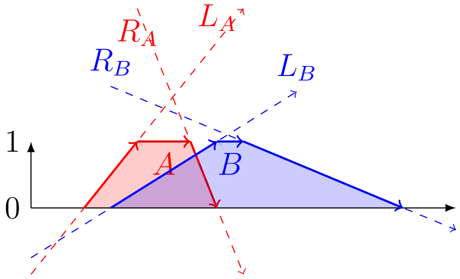


If $c < b'$ or $b > c'$, there are zero top vertices.

If $b' = c$, the only top vertex is $(c, 1)$. If $b = c'$, the only top vertex is $(b, 1)$.

In other cases, there are always two top vertices $(\max(b, b'), 1)$ and $(\min(c, c'), 1)$.

Side vertices To determine the side vertices that delimit $A \cap B$, we compute the intersection of the lines drawn by the edges of each trapezoid over the whole xy plane. Then, we keep the intersections where $y \in [0, 1]$. For example, in the below figure, there is only one intersection that defines a side vertex of $A \cap B$:



Let us denote $L_A \equiv y = \frac{x-a}{b-a}$ the line drawn by the left side of A, and $R_A \equiv y = \frac{x-d}{c-d}$ the line drawn by the right side of A. Similarly, we have $L_B \equiv y = \frac{x-a'}{b'-a'}$ and $R_B \equiv y = \frac{x-d'}{c'-d'}$ defined on B.

We are interested in finding the four intersections $L_A \cap L_B$, $L_A \cap R_B$, $R_A \cap L_B$, and $R_A \cap R_B$. Each is easy to determine by solving the system of two equations associated with the pair of lines. For example, $L_A \cap L_B$ is the point $(\frac{ab'-ba'}{a-b+b'-a'}, \frac{a-a'}{a-b+b'-a'})$.

Once we have determined the intersections, we keep the ones where $y \in [0, 1]$ to define the vertices of $A \cap B$.

Let us cover some of the edge cases about these intersections. Firstly, any of the edge lines can be vertical if the trapezoid is crisp on that edge. For example, if $a = b$, L_A is defined by the equation $x = a$. Regardless, the method is the same: we simply use this vertical equation in the system of two equations. Secondly, it is possible that there are no side vertices if some lines are parallel. For example, $L_A \cap L_B$ gives no solution if $a - b = a' - b'$ (or infinite solutions if the lines are the same). In such cases, we ignore the pair of parallel lines. Finally, it is also possible that a side vertex is a top or bottom vertex if the lines intersect on $y = 0$ or $y = 1$.

Area calculation Once we have determined all the vertices (x_i, y_i) of $A \cap B$, arranged in a counter-clockwise sequence of points, we can calculate the area using the shoelace formula:

$$\text{Area}(A \cap B) = \frac{1}{2} \sum_{i=1}^{n} (y_i + y_{i+1})(x_i - x_{i+1}) \tag{5.2}$$

Semi-infinite intervals We sometimes have to compute the area intersection in cases where A is left-infinite or B is right-infinite (for example, when using the operators bf or af). However, we can turn these semi-infinite intervals to finite intervals such that the area calculation is unchanged. If A is left-infinite, we can replace the infinite parameters with any $a \leq a'$ and $b \leq b'$. Similarly, if B is right-infinite, we can replace the infinite parameters with any $c' \geq c$ and $d' \geq d$. Doing so, we can reuse the method highlighted above.

5.5 Architecture

The main objective of introducing Interval Real Logic (IRL) is to use it to impose temporal constraints in a neural architecture for Event Detection. Given a temporal data sequence $\boldsymbol{u} = \{u_i\}_{i=0}^{T}$ we define a neural architecture, called *Interval Logic Tensor Networks (ILTN)* that is capable to recognize *if* and *when* a set of events $\epsilon_1, \ldots, \epsilon_k$ happens in the sequence, under the hypothesis, that certain constraints expressed in IRL are (softly) satisfied.

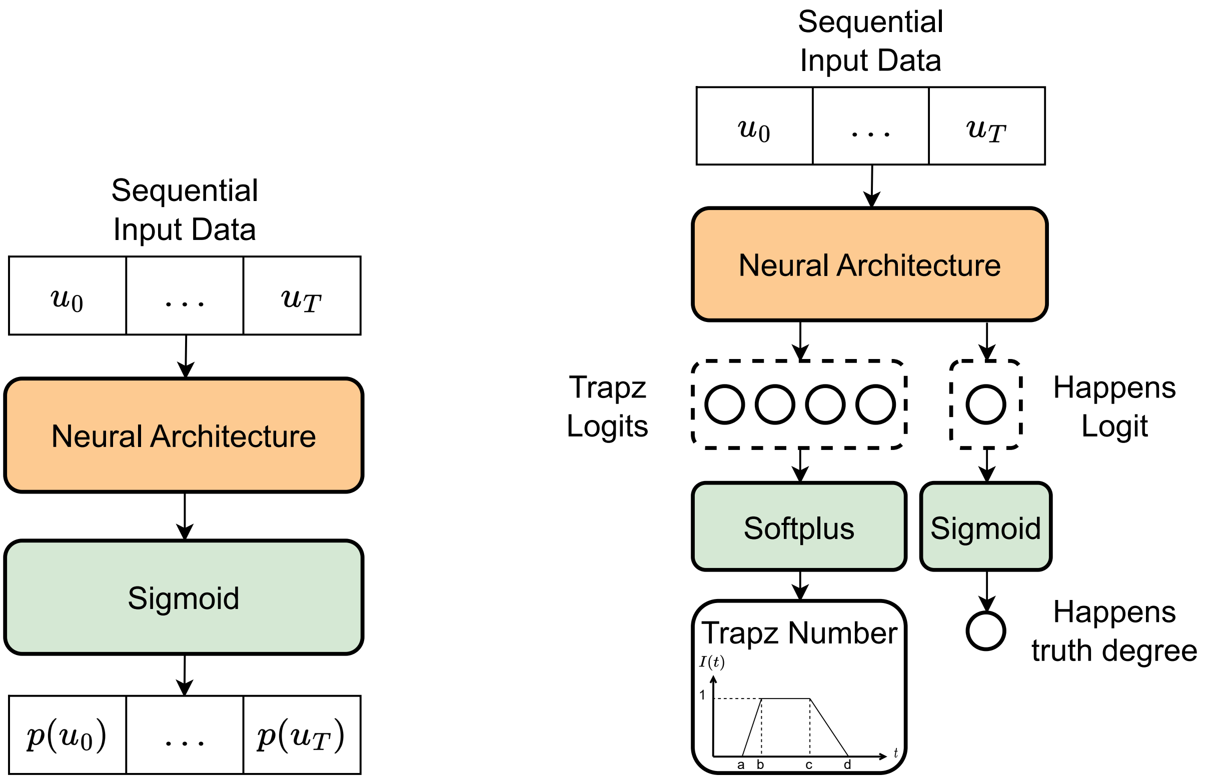
We implemented a first simple prototype of ILTN in TensorFlow as a wrapper of LTN [78]. The present section describes important design choices that enable the architecture. In the description, we concentrate only on the temporal prediction and not on the classification of the events.

5.5.1 Neural Architecture

Figure 5.3b illustrates how neural networks are used to ground any event ϵ. An event is characterized by two elements: a truth degree $\mathsf{H}(\epsilon)$ indicating if the event happens, and by a trapezoid interval defining the membership function and when it happens.

Let us call *logits* the vector of raw (non-normalized) predictions output by the neural model, as is common in the machine learning literature. The truth degree $\mathsf{H}(\epsilon)$ is easily implemented using a single logit node which is then passed to a sigmoid normalization function and constrained in the interval $[0, 1]$.

Directly defining the parameters (a, b, c, d) of the interval is difficult as the semantic constraint $a \leq b \leq c \leq d$ is hard to implement in a neural architecture. Instead, our neural architecture predicts the four values $(a, b - a, c - b, d - c)$. The only semantical constraint on these values is that each is positive. This is easily implemented using four logit nodes which are then passed through softplus activations.



(a) $p(\boldsymbol{u})$

(b) $\epsilon(\boldsymbol{u})$

Figure 5.3: Implementation of a temporal predicate symbol from $\mathcal{L}_t$ (left) and of an event symbol (fuzzy interval and happening predicate) from $\mathcal{L}_e$ (right). Examples of sequential neural architectures are Recurrent Neural Networks or Transformers.

5.5.2 Smooth Membership Functions

We notice an important vanishing gradient issue with trapezoidal interval functions. If x is in the flat regions $I(x) = 0$ or $I(x) = 1$, then $\frac{\partial I(x)}{\partial x} = 0$. To account for this, we define $I_\sim$, a smooth version of the membership function (5.1):

$$I_\sim(x) = \begin{cases} \mathrm{s}_+(x-a) & \text{if } x \leq a, \\ \mathrm{s}_+(\max(b-x, x-c)) & \text{if } b < x \leq c, \\ \mathrm{s}_+(d-x) & \text{if } d < x, \\ I(x) & \text{otherwise.} \end{cases} \tag{5.3}$$

Where s_+ is the softplus function defined by:

$$\mathrm{s}_+(x \mid \beta) = \frac{1}{\beta} \log\left(1 + e^{\beta x}\right) \tag{5.4}$$

$$\frac{\partial\, \mathrm{s}_+(x \mid \beta)}{\partial x} = \frac{1}{1 + \exp(-\beta x)} \tag{5.5}$$

Notice that, in (5.3), the inputs to the s_+ function are all negative values. Intuitively, looking at the graph of softplus in Figure 5.4, s_+ applied to negative values outputs a value that tends to zero with non-negative gradients.

We use I and $I_\sim$ to define an artificial operator with distinct properties in the forward pass and backward pass of the computational graph. [2] Let $\epsilon = e(t_1, \dots, t_m)$ be an event term associated with an interval I and a corresponding smooth version $I_\sim$. We use:

$$\epsilon(x) = I(x) \tag{5.6}$$

$$\frac{\partial \epsilon(x)}{\partial x} = \frac{\partial I_\sim(x)}{\partial x} \tag{5.7}$$

The motivation is demonstrated in Figure 5.5. The backward pass $\frac{\partial I_\sim(x)}{\partial x}$ has non-zero gradients everywhere that push x to fit in the center of the interval. The forward pass remains the accurate evaluation $I(x)$.

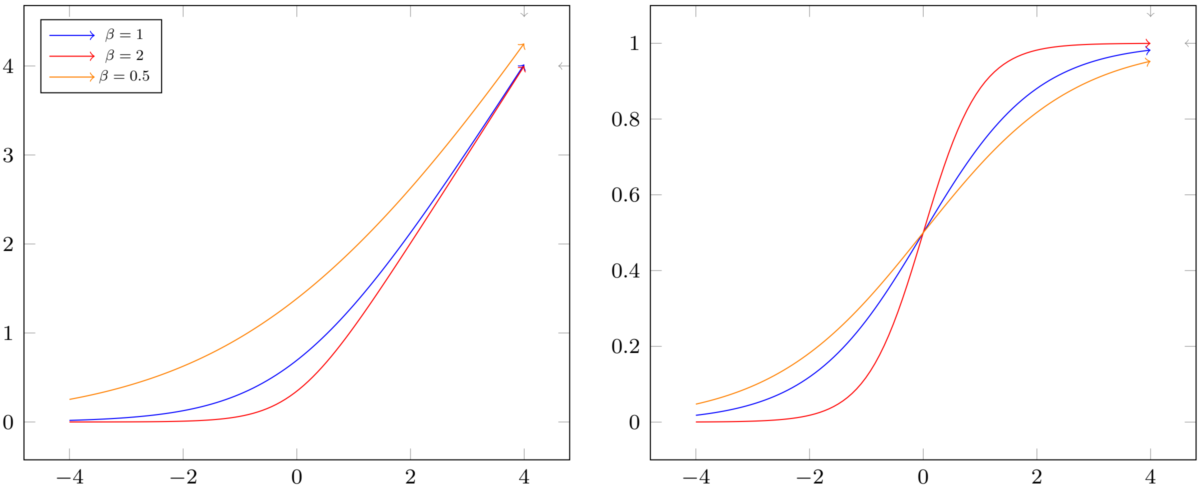
Finally, we use the parameter β to ensure the accuracy of the operator. For example, for large negative differences $x - a$, the output of $\mathrm{s}_+(x - a)$ gets very small and can become zero because of the way computers approximate real numbers. In float32 precision format, this happens with $x - a > 90$ and $\beta = 1$. In such cases, the gradients still vanish. We avoid this issue by setting $\beta = \frac{1}{T}$, where T is the largest time difference occuring in our data, or in other words T is the length of the trace in the experiment.

5.5.3 Smooth Relations

Let $A = (a, b, c, d)$ and $B = (a', b', c', d')$ be two trapezoids. Without loss of generality, suppose that $a \leq a'$. Similarly to how membership functions has zero gradients on some parts of the domain, the relations in 5.4.2 have vanishing gradients in two situations. The first is when $A \,\mathsf{in}\, B = 0$ and the trapezoids do not intersect. In other words, when $d < a'$. The second is when $A \,\mathsf{in}\, B = 1$ and A is fully contained in B. In other words, when $a > a'$, $b > b'$, $c < c'$, and $d < d'$.

Again, we solve this by defining a smooth operator for the backward pass relying on

[2] See also `https://www.tensorflow.org/api_docs/python/tf/custom_gradient`.



(a) $\mathrm{s}_+(x \mid \beta)$

(b) $\frac{\partial\, \mathrm{s}_+(x|\beta)}{\partial x}$

Figure 5.4: The softplus function.

the softplus operator:

$$(A \text{ in } B)_\sim = \begin{cases} \mathrm{s}_+(d-a) & \text{if } d < a', \\ \mathrm{s}_+(a'-a+d-d') & \text{if } A \text{ fully in } B, \\ A \text{ in } B & \text{otherwise.} \end{cases} \tag{5.8}$$

with the non-vanishing derivatives on the trapezoid edges of A and B. We use A in B in the forward pass of the computational graph and $\frac{\partial(A \text{ in } B)_\sim}{\partial x}$ in the backward pass, where x is any parameter defining A or B. Finally, we still set $\beta = \frac{1}{T}$ for the softplus function.

5.6 Experiments

We test the system on synthetic tasks that require a combination of learning and reasoning about temporal relations between fuzzy intervals. Let $\phi_1, \dots, \phi_n$ be n constraints written in Interval Real Logic defining a knowledge base $\mathcal{K}$. Like in LTN [78], the grounding of the knowledge base defines a satisfaction level to maximise. Optimising by gradient descent, we have the following loss function:

$$L(\mathcal{K}, \theta) - (\mathcal{G}(\phi_1, \theta) \wedge \cdots \wedge \mathcal{G}(\phi_n, \theta)) \tag{5.9}$$

where ϕ_i's are $\mathcal{L}_e$ formulas and θ is a set of trainable parameters used to define the grounding. We focus the experimental study on the training of events with constraints written using $\mathcal{L}_e$, which is the main innovation of this paper. Specifically, we focus on learning parameters that define the fuzzy trapezoid intervals of events.

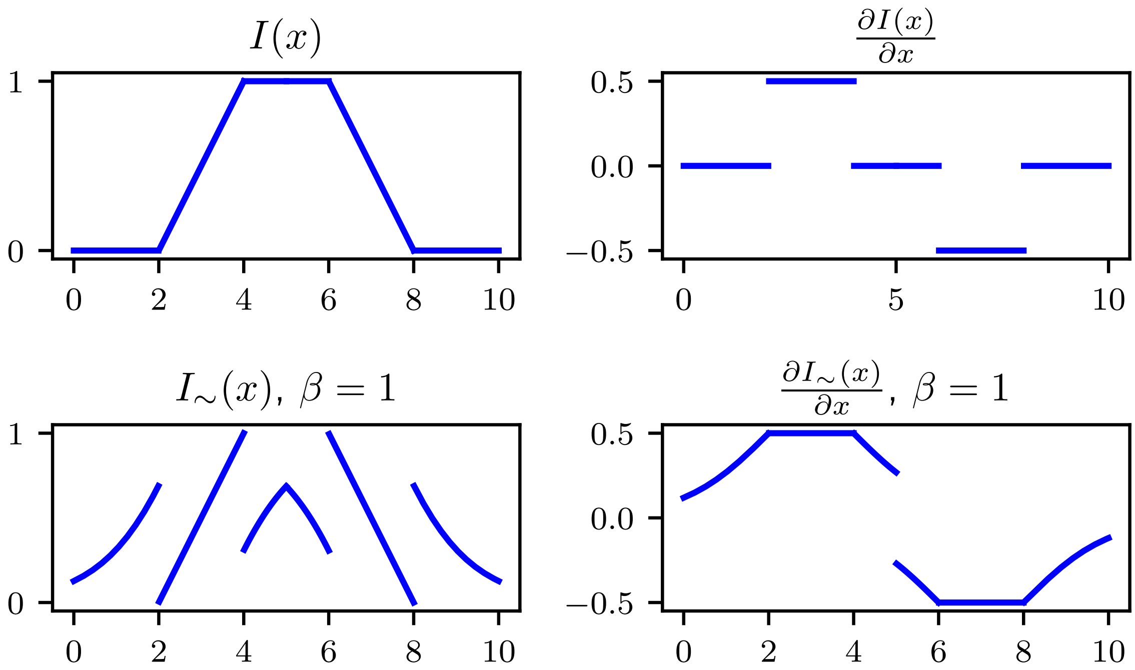


Figure 5.5: Smooth membership function. The forward pass uses $I(x)$ (top left). The backward pass uses $\frac{\partial I_\sim(x)}{\partial x}$ (bottom right).

Table 5.2 displays a list of training experiments where ILTN maximizes the satisfaction of temporal constraints. All tasks are trained using the Adam optimizer [48] with a learning rate of 0.1. In T1, T2, T3, and T4, the results are obtained after training for 50, 500, 5000, and 200 training steps, respectively. For the logical operators, we use the product t-norm $u \wedge v = uv$ and the standard negation $\neg u = 1 - u$.

We highlight the following features:

- In T1, T2 and T3, the system learns fuzzy intervals,
- In T4, the system learns a time point value x,
- T1, T2, and T3 display constraints using Allen's relational symbols af , bf , st , and ol ,
- T3 and T4 display constraints using membership functions,
- T4 displays a constraint using a functional symbol end,
- In T1 and T2, $u \approx v$ is a smooth equality predicate implemented as $\exp(-|u - v|) \in [0, 1]$.

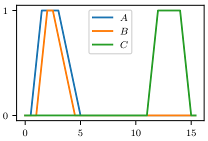
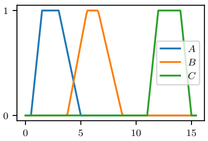
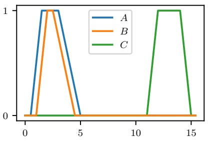
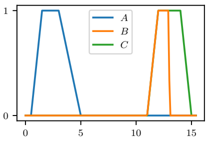
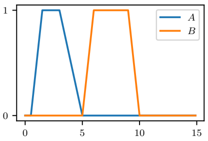
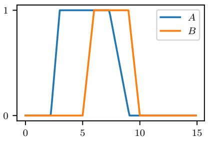
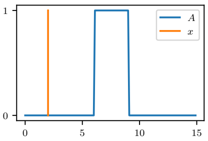
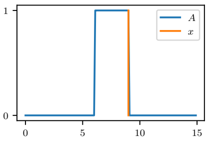
Task	Initial Conditions	Setting	Constraints	Result
T1		• B trainable, • A and C fixed.	1. $\lvert B\rvert \approx 2$ 2. B af A 3. B bf C	
T2		• B trainable, • A and C fixed.	1. $\lvert B\rvert \approx 1.5$ 2. B st C	
T3		• A trainable, • B fixed.	1. A ol B 2. $A(3)$ 3. $\neg A(2)$	
T4		• x trainable, • A fixed.	1. end$(A)(x)$	

Table 5.2: Experiments

5.6.1 Challenges for Future Work

In all tasks, the system learns to update the event groundings to satisfy the knowledge base. The experiments demonstrate that ILTN can successively backpropagate gradients through the Interval Real logic. Nevertheless, we highlight three limitations of our experiments that future work should explore.

Firstly, early stopping was an important factor in our experiments. Continuing training after reaching the maximal satisfaction level could sometimes lead to worsening the results. This is likely due to the smoothing of operators for obtaining non-vanishing gradients. This feature is important early in training. However, once a constraint is satisfied, having vanishing gradients is acceptable. Future work could explore reducing or stopping the smoothening of constraints when their satisfaction levels are high.

Secondly, the Adam optimizer is traditionally used with learning rates in the order of 0.001. In comparison, the learning rate of 0.1 used to train our synthetic tasks is unusually high. A lower learning rate led to experiments not converging fast enough. There is likely a scaling issue in the gradients of some operations. This could also explain why T3 required more training steps than the other tasks to reach convergence: this is the only task that mixes relational operators (ol) and membership functions. The two have gradients scaling differently which can challenge the training. Future work should analyse further the gradient properties of each temporal operator.

Thirdly, the present experiments do not showcase yet the power of learning events that depend on input features. For example, in Figure 5.3b, the present tasks only learn trapezoid logits that define a trapezoid number. There is no sequential data in input and neural architecture that builds on top of it. Future work should explore more elaborate tasks employing such architectures.

5.7 Conclusions

In this chapter, we introduce Interval Real Logic (IRL), a two-sorted logic that enables the prediction of properties that evolve within a set of data sequences (traces) and properties of events that occur within the sequences. IRL semantics are defined in terms of sequences of real feature vectors, and connectives and quantifiers are interpreted using fuzzy logic. We represent event duration through trapezoidal fuzzy intervals, and fuzzy temporal relations are defined based on the relationships between the intervals' areas and their intersections.

We also present Interval Logic Tensor Networks (ILTN), a neuro-symbolic system that leverages background knowledge expressed in IRL to predict the fuzzy duration of events. To prevent vanishing gradient during learning, we use softplus functions to smooth both events and their relations. We evaluate ILTN's performance on four tasks with different temporal constraints and show that it is capable of making events compliant with background knowledge in all four tasks.

Chapter 6

A Neuro-Symbolic Approach For Non-Intrusive Load Monitoring

A requirement of Smart Grids is the ability to predict the energy consumption patterns of their users. In the residential domain, this is usually not feasible due to the inability of the grid to converse with (legacy) domestic appliances. To overcome this issue Non Intrusive Load Monitoring (NILM) was introduced, a task in which a predictor is used to disaggregate household power consumption. Many of the newer approaches make use of Neural Networks to accomplish this task, due to their superior ability to detect patterns in temporal (thus sequential) data. These models unfortunately require a huge amount of data to achieve good performance, and have the tendency to overfit the training data, making them difficult to predict future consumptions. For these reasons, adapting them to optimally predict a (future) house's consumption requires expensive and often prohibitive data collection phases. We propose a solution in the form of a neuro-symbolic framework that refines neural network predictions via a constrained optimization problem modelling the characteristics of the appliances of a house. This combined approach achieves superior performance with respect to the neural network alone over two out of five appliances and comparable results for the remaining ones, without requiring further training data.

6.1 Introduction

The past few years have seen an increased awareness by people and institutions on climate change and its consequences. With the objective to avoid/limit its effects, people have started to change their habits (e.g. walk and cycle more or drive electric vehicles), while governments have committed themselves to reduce CO2 emissions [1] with the long term

[1] Paris agreement

goal to be climate-neutral by 2050[2]. Therefore, huge investments have been made in renewable (aka green) energies in order to meet the increasing demand and gradually replace the dependence on traditional source of energies [3]. Nevertheless, renewable energies have a limitation connected to the presence of their natural source. Therefore, investments in the development of a ”smart” electrical network, which is able to guarantee an effective distribution of energy, have been made in parallel. This network has been called smart grid and, differently to the traditional distribution, where there is an unidirectional flow from producer to consumer, a bidirectional exchange of information is achieved in order to guarantee an effective usage of the electricity.

One of the requirements of a smart grid consists in understanding and predicting the energy consumption pattern of the consumers. This is done by processing the data of the energy consumption of the individual appliances of all houses. To measure the energy consumption of the appliances, two approaches can be adopted: Intrusive Load Monitoring (ILM) and Non-Intrusive Load Monitoring (NILM). ILM is based on the installation of a sensor (e.g., smart plugs or smart sockets) for each appliance that monitors and sends information about the consumption of the appliance back. Even though, ILM ensures accurate measurements, it is usually expensive and often perceived as "too intrusive” by consumers (i.e., each sensor has to be installed inside the house of the consumer). On the contrary, NILM, where the consumption of each appliance is obtained from the disaggregation of the total energy of the house, represents a cheap and less invasive solution (even if less accurate) with respect to ILM.

Due to the success that deep learning models have achieved in solving tasks of different domains (e.g. computer vision and natural language processing), these models have been started to be applied in NILM. Nevertheless, these models require a huge amount of annotated training data and lack of the ability to generalize to unseen situations (e.g., situations not included in the training data). Therefore, in the last years, neuro-symbolic techniques, which combine neural networks with symbolic reasoning, have started to be applied to overcome these issues.

In this paper, we devise a neuro-symbolic algorithm that combines the prediction of the Neural Network with a Constrained Programming optimization problem, used to refine the raw prediction of the network. The optimization problem is used to encode the behavior of the appliance in terms of its consumption over time. For example, the characteristics that can be captured may be:

- Minimum/maximum duration of appliance activation.
- Minimum/maximum instantaneous used power.

[2] Net-zero

[3] Global electricity insights 2022

- Presence of multiple "states" with different duration and power adsorption.

All the characteristics, which are encoded as logic formulas, can then be used to correct the output of the network, and to find coherent intervals of times in which the the appliance was in use (the whole duration of a washing cycle of a washing machine, for instance), together with their expected consumption. Furthermore, the addition of this logic layer allows for the disambiguation of dubious predictions done by the Neural Network (due for instance to the presence of noise), increasing the overall disaggregation performance. Experiments have been performed on the UK-DALE dataset in the "seen" setting (i.e., test on the same houses' appliances used for training but on a different period of time), and show that our combined approach is able to outperform a fully neural model over the prediction of two out of five appliances, with comparable results for the remaining ones.

The rest of the chapter is organized as follows: Section 6.2 briefly reviews the state of the art on NILM, focusing mostly on deep learning approaches; Section 6.3 formally describes the problem; Sections 6.4 and 6.4.1 describes our proposed approach; Section 6.5 presents the experimental setting; Section 6.6 shows the experimental results; Finally, in Section 6.7 conclusions are drawn and directions for future works are briefly discussed.

6.2 Related work

Since its introduction by Hart [37], NILM has gathered the attention of researchers not only for the intrinsic challenges [38] that it involves but for the benefits that approaches like NILM could bring into our everyday life [32]. Over the last decade, more "classical" approaches to NILM (see [96] for a survey) have been replaced by deep learning methods. This is due to the success that deep learning has achieved in many different fields like computer vision and natural language processing (see [16] and [70] for a survey). In [97], authors instantiate a network (one for each appliance) and train it to learn a mapping between a sequence of mains to a sequence of appliance consumption. At inference time, multiple predictions for a generic time t are averaged in order to obtain a single prediction. [45] obtains more accurate results with respect to [97], by predicting from the current window of mains only the consumption of the appliance which corresponds to the middle point of the window. A more recent approach [71], adopts an attention mechanism to improve the generalization capability of the overall model. [66] proposes a novel architecture which integrates the Fourier transform and achieves comparable results with respect to the state-of-the-art approaches while being faster and smaller. In all the aforementioned approaches, prior (explicit) knowledge about the behaviour of the appliances is not exploited to perform the disaggregation (i.e. the disaggregation is completely learnt from data). As far as we know the only attempt to exploit prior knowledge to solve

NILM has been done by [11], but this is a fully symbolic approach and then no neural networks have been used. In the last years, neuro-symbolic integration, which integrates neural and symbolic AI, has emerged as a new paradigm to merge the strengths and reduce/limit the weaknesses of the neural and symbolic "worlds" [39]. Therefore, different neuro-symbolic frameworks have been proposed over the years like frameworks based on fuzzy-logic [78, 59], probabilistic logic programming [56, 92] or an Optimization modulo theories [77]/Mixed Integer Linear Programming (MILP) encoding (see [29] and [7]). Furthermore, these approaches have been applied to solve complex tasks like semantic image interpretation [28] and event recognition from different data sources (e.g. video [6, 7] and audio [86]). Inspired by [7], we encode the background knowledge about the behaviour of each appliance as a mixed integer linear programming problem (MILP) and use it to refine the prediction of the neural network. Differently from [7], we also learn the parameters related to the MILP problem (see Section 6.4 for details).

6.3 Problem definition

NILM consists in the disaggregation of the total energy consumption of a house into the consumption of the individual appliances belonging to it. Formally, denoting with $X_{tot} = (x_1, \dots x_t)$, $x_i \in \mathbb{R}^+$, the total energy consumption for the period of time starting at 1 and ending at t, and supposing the presence of n appliances, we can express the total energy consumption at a given time i as:

$$x_i = \sum_{j=1}^{n} y_{ji} + \gamma_i \;\; with \; 1 \leq i \leq t$$

where y_{ji} is the consumption of the $j-th$ appliance at time i and γ_i is a noise factor. We are interested in finding all the appliance consumption $Y_j = (y_1, \dots, y_t)$ $y_i \in \mathbb{R}^+$, from X_{tot}. To achieve this objective, we also assume to have a background knowledge $\mathcal{K}$ (defined over a first order language $\mathcal{L}$) about the consumption/behaviour of each appliance. Therefore, our problem consists in finding an interpretation $\mathcal{I}$ (i.e. predicting a sequence of values, one for each appliance), such that $\mathcal{I} \models \mathcal{K}$.

Example 10. *Suppose that for a given house h, we have that $X^h_{tot} = \{1000, 1000, 1200, 1400\}$. We want to predict the consumption for the appliance app_1 of h. The background knowledge $\mathcal{K}$ states that when app_1 is active, its consumption has to be less than* 500 *at time t and the sum of consumption of the next two timestamps (i.e, $t+1$ and $t+2$) has to be between* 700 *and* 1400*. We can write a first order logical formula that expresses the above* *conditions:*

$$\begin{aligned}
&\forall t_s, t_e \exists t_i, t_j, t_z \\
&\quad cons(x, t_i, y_{t_i}) \leq 500 \land \\
&\quad 700 \leq cons(x, t_j, y_{t_j}) + cons(x, t_z, y_{t_z}) \leq 1400 \land \\
&\quad active(x, t_i) \ \land \ active(x, t_j) \land \ active(x, t_z) \land \\
&\quad t_s \leq t_i < t_j < t_z \leq t_e \land \\
&\quad t_j = t_i + 1 \land t_z = t_i + 2
\end{aligned}$$

where t_s and t_e represent respectively the begin and the end of the period of consumption, $cons(x, t, y_t)$ is a function that returns the consumption of a generic appliance x at time t (i.e., it returns y_t) and active is a predicate that returns 1 if an appliance is active at a time t and 0 otherwise. Continuing the example, if we know that app_1 is active between time 2 and 4, some of the interpretations that satisfy $\mathcal{K}$ are:

$$\begin{aligned}
\mathcal{I}_1 = \{&cons(app_1, 2, 250), cons(app_1, 3, 500), cons(app_1, 4, 600), \\
&active(app_1, 2), active(app_1, 3), active(app_1, 4)\} \\
\mathcal{I}_2 = \{&cons(app_1, 2, 350), cons(app_1, 3, 600), cons(app_1, 4, 700), \\
&active(app_1, 2), active(app_1, 3), active(app_1, 4)\} \\
&\vdots
\end{aligned}$$

As can be seen, there may be more than one interpretation that satisfies $\mathcal{K}$ (we denote with $\mathcal{I}_c$ the set of such interpretations). Therefore, as done in [7], we introduce a cost function c that gives a score (i.e. a real value) for each $\mathcal{I}$ and select the interpretation with the minimum cost:

$$\mathcal{I}_{min} = \operatorname*{argmin}_{\mathcal{I} \in \mathcal{I}_c} c(\mathcal{I})$$

To find $\mathcal{I}_c$, we devise a neuro-symbolic approach where the initial prediction of the network Y_{NN_j} is refined in order to produce a new prediction $Y^*_{NN_j}$ that keeps into account the knowledge $\mathcal{K}$. To train the overall system, we have a training set of consumption of m houses:

$$D = \{(X^i_{tot}, \{Y^i_j\}^{n_i}_{j=1})\}^m_{i=1}$$

where Y^i_j denotes the consumption of the $j - th$ appliance in house i and n_i the number of appliances in the same house (i.e. different houses may have different appliances). It is not always that the entire D is used for the training phase (further details follow).

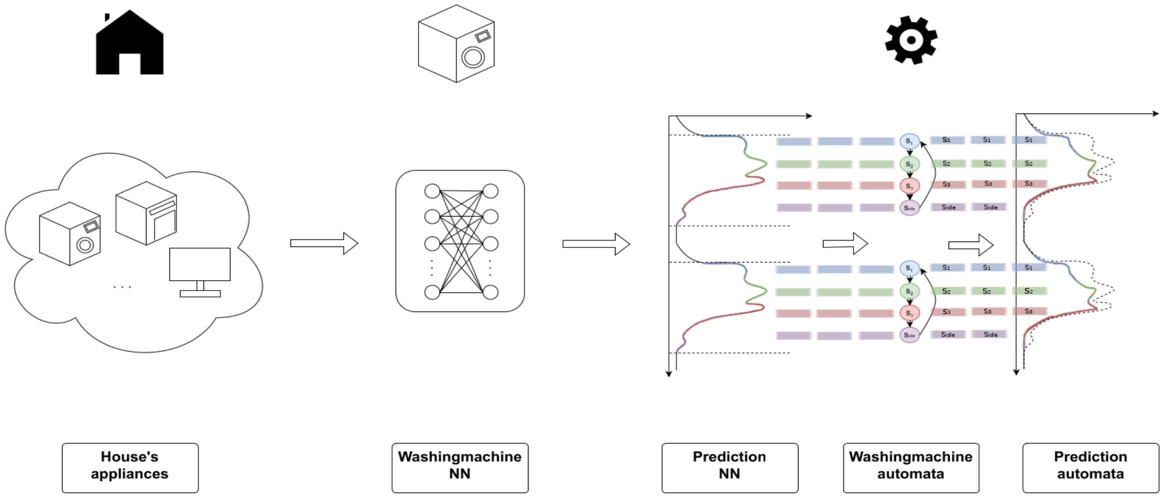


Figure 6.1: Overall approach on washingmachine appliance: the total consumption of the house (i.e., the total consumption of all its appliances) is passed to the washingmachine NN that provides an initial prediction for the consumption of the washingmachine. This prediction is then refined by its corresponding (learnt by CSP) automaton that changes the prediction of the washingmachine NN by leveraging the knowledge encoded in each of its state.

6.4 Proposed approach

The proposed approach consists in a neuro-symbolic framework that combines the generalization capabilities of a (pre-trained) Neural Network with a constrained satisfaction problem (CSP), which has built in its specifications the domain knowledge and enables to refine the predictions of the network (see Figure 6.1 for a high level overview of the proposed approach). Differently from the approaches found in literature [35, 10] the neural and symbolic models are not in competition but cooperate together. The framework gets an initial prediction from the neural network (i.e., the prediction of the energy consumption of the appliance the neural network has been trained to represent), which is then refined by the optimization problem. Differently from other approaches, our CSP problem is only partially defined, and before being used is meant to be trained onto a small set of labeled examples of activations of a specific appliance to fit it properly.

The goal of the CSP is to model the energy consumption patterns of a specific appliance. Before moving on, we must briefly formalize the expected pattern of the appliance. We can imagine the lifecycle as a contiguous infinite sequence of idle intervals (in which the appliance is not used/switch off/in standby) and activation intervals, in which most of the power gets consumed and some useful work is performed. Each activation j starts at a certain time t^j_{start} and ends at time t^j_{end} (we will use t_{start} and t_{end} later in the article when referring to a generic activation). An example of this behavior is depicted in Figure

6.2.

Following previous work on the field [17], we model an appliance as finite state automaton. Indeed, an appliance is a machine that is built to perform a predetermined sequence of actions cyclically. Therefore, a neuro-symbolic approach, which models an appliance's consumption as a sequence of states, has the potential to improve the accuracy of the prediction. In addition, a neural network trained to predict an appliance's consumption over time, generally reasons in terms of time-points (i.e., the consumption at time x is y), while our proposed neuro-symbolic approach reasons in term of intervals of time (i.e., for x seconds the consumption will follow a specific trend curve) making the prediction more coherent over time. In this conceptualization, the appliance is represented as a collection of n states $\{s_i\}_1^n$ each associated with a duration t_i and a function $f_i : \mathbb{R}^+ \rightarrow \mathbb{R}$ that maps each instant of the interval t_i with the power consumed by the appliance at that instant. Thus, the activation cycle of the appliance is described by a set of states $S = \{< s_i, t_i, f_i >\}_{i=1}^n$. There is then a special state s_{idle}, the *initial state*, with associated its function f_{idle}, that has no fixed duration. This is the state where the appliance is before is switched on and after is switched off (or put in stand-by).

A life cycle of an appliance starts in $s_0 = s_{idle}$, when it is activated it switches to s_1 and stays in that state for t_1 time. After that, the appliance moves to s_2 for t_2 time and then switches to the next state. After t_N time spent in state s_N, the appliance shuts off and goes back to s_{idle}.

Our CSP framework applies these ideas by first learning a set of states S, by fitting a small number of examples of consumption patterns of the target. These are then used to refine the predictions of the network for the time in which the appliance is considered active *i.e.* between t_{start} and t_{end}. We currently do not model the idle state s_0 (the CSP problem that the power consumption at idle is some constant value p_{idle}) , relying on the neural network to (roughly) identify the t_{start} and t_{end} of each activation. In the next sections, we will describe in detail both the training and inference procedure.

6.4.1 Training

As explained in previous sections, the training procedure of the CSP problem relies on labeled data about a few activations of the appliance. We can formalize it as a function $P : \mathbb{R}^+ \rightarrow \mathbb{R}$ that links each time instant with a power consumption. For training, we have a series of m intervals $\{< t_{j,start}, t_{j,end} >\}_{j=1}^m$ that encode the boundaries of the activations. The last bit of information is the constant idle power consumption p_{idle} of the appliance. This is necessary because, albeit the problem relies on the network for the prediction of power consumption when idle, it cannot assume that the activation intervals are perfectly timed, thus it must account for the appliance *potentially* being in

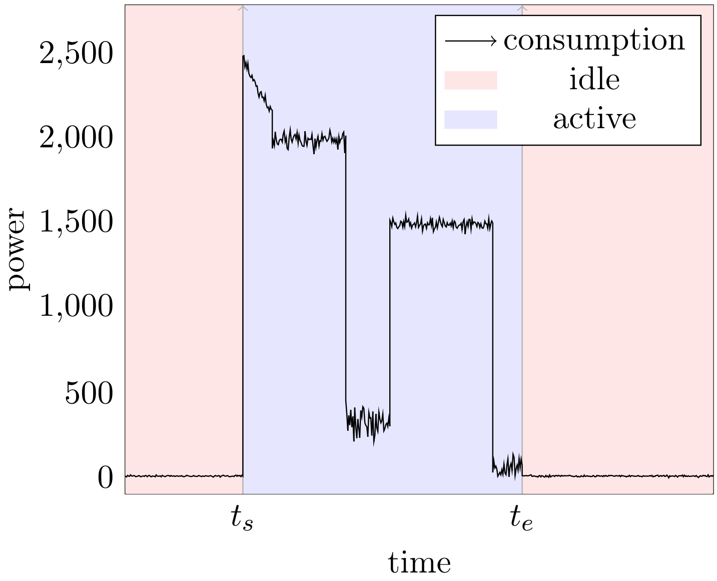


Figure 6.2: Example of an activation cycle of an appliance

idle near t_{start} and t_{end}. Figure 6.3 depicts the whole activation cycle of an appliance that follows these principles. The target is modeled using 3 active states s_1, s_2, s_3, with their corresponding state functions f_1, f_2, f_3. Aside from the "active" states, the framework uses two more states s_0 and s_{n+1} to model the idle state (before the first and after the last active state), both of them represented modeled by the function f_{idle}, that have a duration of t_0 and t_{n+1}. The objective of the fitting is to obtain a curve (by adding up state and idle functions) that closely matches the target consumption curve in the interval between t_{start} and t_{end}.

In order to formalize the fitting problem, we must introduce some constructs. We define $t^*_{j,i}$ as the summation of $t_{j,start}$ and of all the durations of all the $i-1$ states:

$$t^*_{j,i} = t_{j,start} + \sum_{k=1}^{i-1} t_{j,k}$$

Each function f_i is an exponential function in the form

$$f_{j,i}(t) = \alpha_{j,i} e^t + \beta_{j,i}$$

while f_{idle} has the form

$$f_{j,idle}(t) = C$$

The function that is learnt from a specific activation j is then:

$$f_j(t) = \begin{cases} f_{j,idle}(t) & t_{j,start} < t \leq t^*_{j,1} \\ f_{j,1}(t) & t^*_{j,1} < t \leq t^*_{j,2} \\ ... \\ f_{j,i}(t) & t^*_{j,i} < t \leq t^*_{j,i+1} \\ ... \\ f_{j,n}(t) & t^*_{j,n-1} < t \leq t^*_{j,n} \\ f_{j,idle}(t) & t^*_{j,n} < t \leq t_{j,end} \end{cases} \tag{6.1}$$

Defining the function encoding the interval of interest as

$$P_j(t) = \begin{cases} P(t - t_{j,start}) & t_{j,start} < t < t_{j,end} \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

we can define the difference between the real and predicted consumption as

$$\Delta c_j = \int_{t_{j,start}}^{t_{j,end}} |P_j(t) - f_j(t)| dt \tag{6.3}$$

Moreover, the training problem tries to minimize the deviation between the parameters. In particular the deviation between the duration of the states across the various training sequences:

$$\Delta s = \sum_{i=1}^{n} \max(\{t_{j,i}\}_{j=1}^{m}) - \min(\{t_{j,i}\}_{j=1}^{m}) \tag{6.4}$$

and the same deviation for the $f_{j,i}(t)$ parameters $\alpha_{j,i}$ and $\beta_{j,i}$:

$$\Delta \alpha = \sum_{i=1}^{n} \max(\{\alpha_{j,i}\}_{j=1}^{m}) - \min(\{\alpha_{j,i}\}_{j=1}^{m}) \tag{6.5}$$

$$\Delta \beta = \sum_{i=1}^{n} \max(\{\beta_{j,i}\}_{j=1}^{m}) - \min(\{\beta_{j,i}\}_{j=1}^{m}) \tag{6.6}$$

The overall training problem is then defined as:

$$\underset{C,\alpha_{j,i},\beta_{j,i},t_{j,i}}{\text{minimize}} \ \Delta s + \Delta \alpha + \Delta \beta + \sum_j \Delta c_j \tag{6.7}$$

6.4.2 Inference

Once the parameters $C, \alpha_{j,i}, \beta_{j,i}, t_{j,i}$ have been optimized in training, a new optimization problem is defined for inference. Differently from training, this time there are two distinct sources of input data: the actual raw aggregated power consumption (the same input given to the neural network) and the output of the neural network. The problem is expected to refine the neural output by using the "learnt behaviour" of the appliance (encoded in $C, \alpha_{j,i}, \beta_{j,i}, t_{j,i}$), while at the same time ensuring that the final prediction does not conflict with the actual instantaneous aggregated power consumption (e.g., predicting a peak appliance power that is higher than the aggregated one).

Due to the fact that the optimization problem models only the active state of the appliance, the algorithm is used to refine the output of the neural network only where an activation of the appliance is detected. The activation window is computed by looking at the neural network output. Each activation starts when a consumption greater than a value ACTSTART, and ends when the power consumption remains below ACTSTART for at least ACTTOLERANCE seconds. Both the values for ACTSTART and ACTTOLERANCE can be selected by looking at the ground truth power consumption over time of the appliance in the training set.

We can define the difference between the aggregated and predicted consumption as:

$$\Delta m_j = \int_{t_{j,start}}^{t_{j,end}} |P_j(t) - f_j(t) - \textsc{baseline}| dt \tag{6.8}$$

Where BASELINE is the average value of the mains power consumption before and after the current activation. The BASELINE offset is necessary due to the fact that in each instant there could be different sets of other appliances draining power, thus resulting in an unpredictable baseline.

The optimization problem takes into account both the predicted and aggregated data. As in training, it computes a cost that is used in the optimization objective. When it predicts an activation (and its corresponding $f_j(t)$), this cost is:

$$activecost = w_c \Delta c_j + w_m \Delta m_j \tag{6.9}$$

where w_c and w_m are two appliance dependent hyperparameters that weights the contributions of the two costs.

Given that the refinement is triggered by the prediction of the neural network, the model must also consider the hypothesis that the neural prediction is a false positive. In this case, the role of the problem is to discard the prediction. Therefore, the optimization cost is computed as the power consumption predicted by the neural network.

$$inactivecost = \int_{t_{j,start}}^{t_{j,end}} P_j(t) dt \tag{6.10}$$

Assuming that the parameters $\alpha^T_{j,i}, \beta^T_{j,i}, t^T_{j,i}, \Delta s^T, \Delta\alpha^T, \Delta\beta^T$ are the ones learnt during training, the optimization problem is defined as:

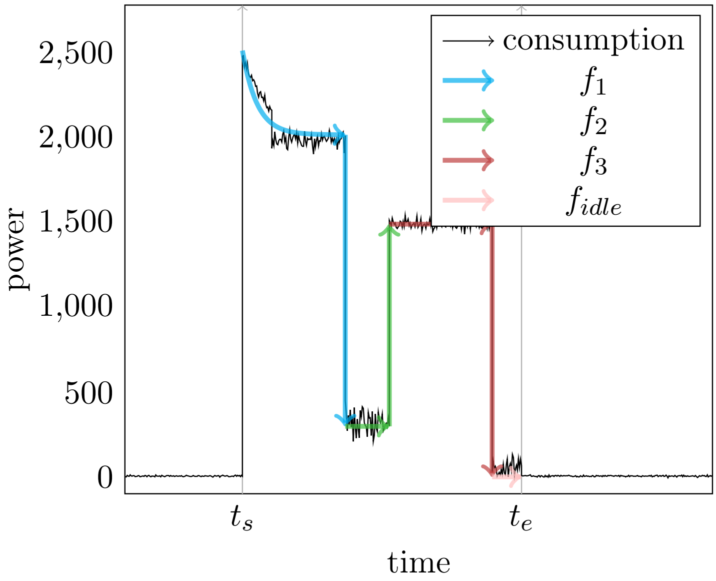


Figure 6.3: Representation of the result of the training procedure. The target activation was fitted using an automaton with 3 states s_1, s_2, s_3. In the figure are depicted the corresponding state functions f_1 (in cyan), f_2 (green), f_3 (red) and the idle function (a constant value, in this case 0) f_{idle} for the idle state.

$$
\begin{aligned}
&\underset{\alpha_{j,i},\beta_{j,i},t_{j,i}}{\text{minimize}}(\texttt{isactive})\ \mathit{activecost} + (\overline{\texttt{isactive}})\ \mathit{inactivecost} \\
&\text{subject to} \\
&\qquad t^T_{\text{min}} d_B - \Delta s^T d < t_{j,i} < t^T_{\text{max}} d_B + \Delta s^T d \\
&\qquad \alpha^T_{\text{min}} - \Delta\alpha^T < \alpha_{j,i} < \alpha^T_{\text{max}} + \Delta\alpha^T \\
&\qquad \beta^T_{\text{min}} h_B - \Delta\beta^T h < \beta_{j,i} < \beta^T_{\text{max}} h_B + \Delta\beta^T h \\
&\text{where} \\
&\qquad t^T_{\text{min}} = min(\{t^T_{j,i}\}^m_{j=1}) \quad t^T_{\text{max}} = max(\{t^T_{j,i}\}^m_{j=1}) \\
&\qquad \alpha^T_{\text{min}} = min(\{\alpha^T_{j,i}\}^m_{j=1}) \quad \alpha^T_{\text{max}} = max(\{\alpha^T_{j,i}\}^m_{j=1}) \\
&\qquad \beta^T_{\text{min}} = min(\{\beta^T_{j,i}\}^m_{j=1}) \quad \beta^T_{\text{max}} = max(\{\beta^T_{j,i}\}^m_{j=1})
\end{aligned}
\tag{6.11}
$$

Where `isactive` is a boolean variable that is true if the optimization problem predicts an activation, and false otherwise. The hyperparameters d_B, d, h_B, h are used to scale the values of the parameters $t_{j,i}, \beta_{j,i}$. The parameters $\alpha_{j,i}$ are not scaled to avoid losing the overall shape of the power consumption curve.

6.5 Experimental setting

This section describes the experimental setting that we defined to validate our proposed approach. In detail, we compare the predictions of a fully neural approach with respect with our neuro-symbolic approach. As a neural baseline, we use the model described in [45] which is also used as input to our neuro-symbolic approach. All the experiments have been run on the UK Domestic Appliance Level Electricity (UK-DALE) dataset which is one of the most used dataset in the literature to evaluate the performance of the disaggregation algorithms.

6.5.1 UK-DALE

UK-DALE contains the measurements of the energy consumption for the whole house and individual appliances of five UK houses. The readings have been collected by sampling every 6 seconds and refer to the period 11/09/2012-04/26/2017[4]. Each house hosts at least two occupants, with occupants be potentially different for type (familiy or not family) or habits (e.g. working all day). Therefore, the consumptions are not (always) the same for each house. More than 15 types of appliances are contained in the dataset but not all of them are in all houses. As done by other works like [45, 97], we focus on kettle, microwave, fridge, dishwasher and washing machine because these are the appliances having the highest impact on the total aggregated consumption.

6.5.2 Seen setting

We evaluate our neuro-symbolic approach and the neural baseline on the "seen" scenario. Roughly speaking, it consists in seeing how both approaches behave when they have to predict the appliances' consumption over a period of time that they have not seen during training. In detail, given a time window $w^h = [s^h_w, e^h_w]$, $s^h_w, e^h_w \in \mathbb{N}$ with $s^h_w < e^h_w$, for an house h, we define two windows, w^h_{train} and w^h_{test}, where:

$$\begin{aligned} w^h_{train} &= [s^h_{w_{train}}, e^h_{w_{train}}] \\ w^h_{test} &= [s^h_{w_{test}}, e^h_{w_{test}}] \end{aligned}$$

with:

$$s^h_w \leq s^h_{w_{train}} < e^h_{w_{train}} < s^h_{w_{test}} < e^h_{w_{test}} \leq e^h_w$$

and we train and test both approaches using the data over the time windows w^h_{train} and w^h_{test}, respectively.

[4] we used UK-DALE-2017l

6.5.3 Metric

As done in other works like [97, 45, 71], we evaluate the predictions using the mean absolute error (MAE):

$$\mathrm{MAE}(\hat{Y}_j, Y_j) \; = \; \frac{1}{l}\sum_{i=1}^{l} |\hat{y}_{ji} - y_{ji}|$$

where l denotes the length of the sequence (i.e. the length of the main) and $\hat{Y}_j$ and Y_j represent the predicted and the truth consumption sequence for appliance j (with $\hat{y}_{ji}$ and y_{ji} representing the consumption at time i).

6.6 Results

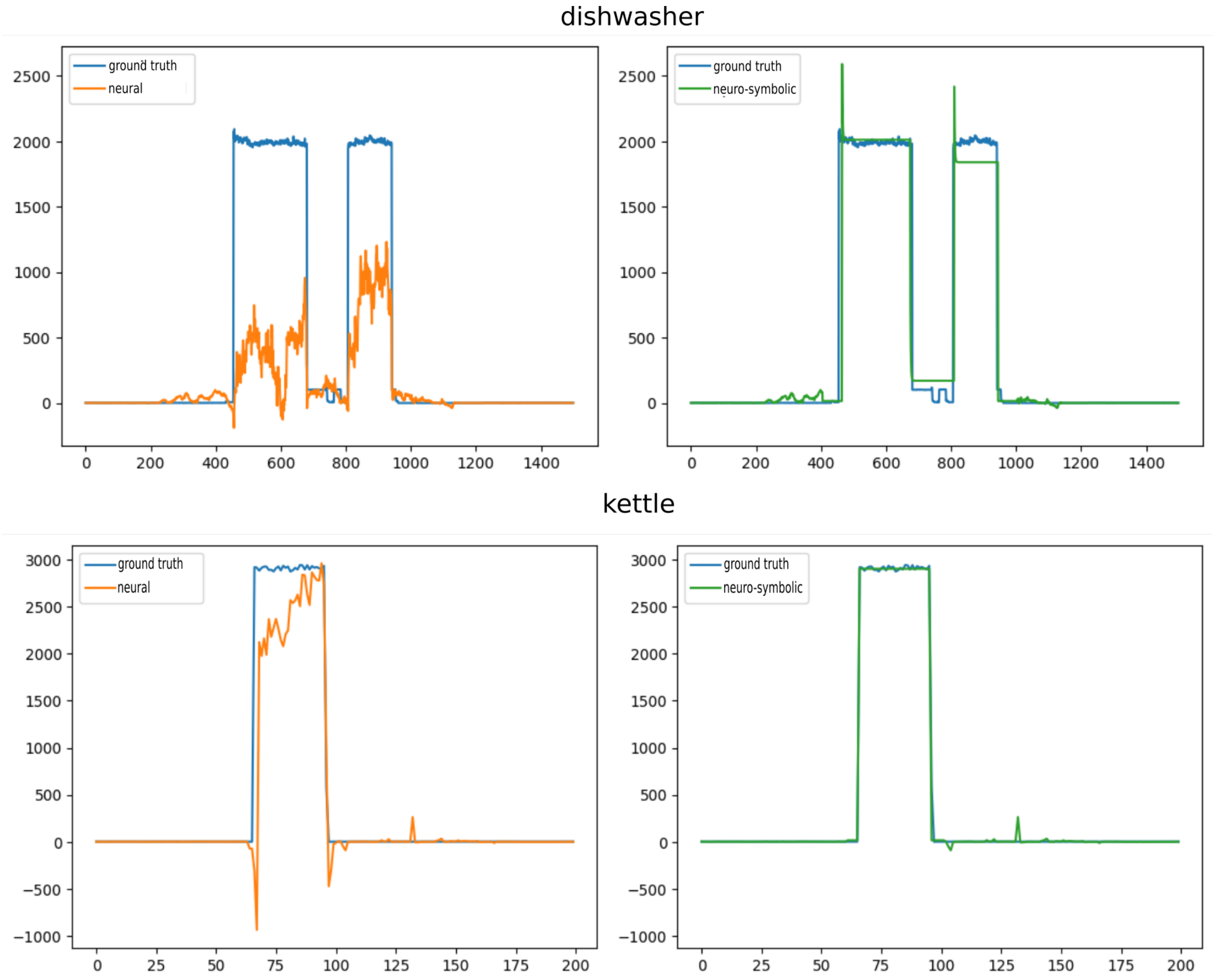
Appliance	MAE nn	MAE neuro-symbolic
dishwasher	17.7	**8.02**
kettle	7.25	**4.52**
fridge	15.8	15.8
microwave	7.92	7.98
washingmachine	8.55	8.93

Table 6.1: MAE over the test set.

In Table 6.1 are reported the MAEs for both the neural baseline and our neuro-symbolic approach on all the appliances. As can be seen by looking at the table, our neuro-symbolic approach outperforms the neural network on two out five appliances, while having comparable results on the remaining ones. In Figure 6.4 are shown the prediction of both the neural network and our neuro-symbolic approach. On the left, is shown the comparison between the ground truth (blue) and the neural network (orange), while on the right is shown the comparison between the ground truth and our neuro-symbolic approach (green). As can be seen by looking at the first two rows, the use of background knowledge provided by our neuro-symbolic approach is useful to the neural network when the network captures the underlying consumption pattern, but it is not able to completely fill the gap with respect to the ground-truth. More precisely, our neuro-symbolic approach works when the predictions of the neural network are already quite accurate but are not consistent over time. This is not surprising since a trained appliance's network reasons in terms of time points, while our neuro symbolic approach reasons in terms of intervals of time (see Section 6.4). For the remaining appliances, the lack of an improvement is due to different reasons: the objective of the constrained optimization

problem consists in refining the predictions of the network, if they are already optimal (fridge) or too wrong (microwave) then the impact of the background knowledge is almost null; the training set does not contain enough representatives samples to model correctly the behaviour of the appliance. This is the case of the washing machine appliance that has not so many samples/activations and whose data contains a lot of variability. As a consequence, its corresponding MAE is low even though the predictions are not good. As can be seen by looking again at Figure 6.4, the neural model may predicts a negative value of consumption (e.g., for dishwashwasher and kettle), this is due to the fact that no activation is used at last layer of the baseline model [45]. We decided to keep the same model and leave the task to remove this behaviour to the background knowledge[5].
In all the experiments, the automaton model is learnt from (a subset of) data of the target house and then is always available. If not, we can use the training data of the other houses, but this would led to a drop in terms of performance, since we are going to learn an automaton on a potentially different models of the same appliance which is not exactly what we want to predict. Furthermore, even though we consider only five appliances, the overall model can still be applied when there are more appliances. Indeed, despite having much more noise, which is due to the presence of more appliances that switch on and off, respectively, the model reasons on each appliance independently, and then it should work exactly the same.

[5] Another option would be to use an activation function to avoid negative values (e.g. ReLU).


dishwasher
ground truth
neural
ground truth
neuro-symbolic
2500
2000
1500
1000
500
0
0
200
400
600
800
1000
1200
1400
kettle
ground truth
neural
ground truth
neuro-symbolic
3000
2500
2000
1500
1000
500
0
−500
−1000
0
25
50
75
100
125
150
175
200

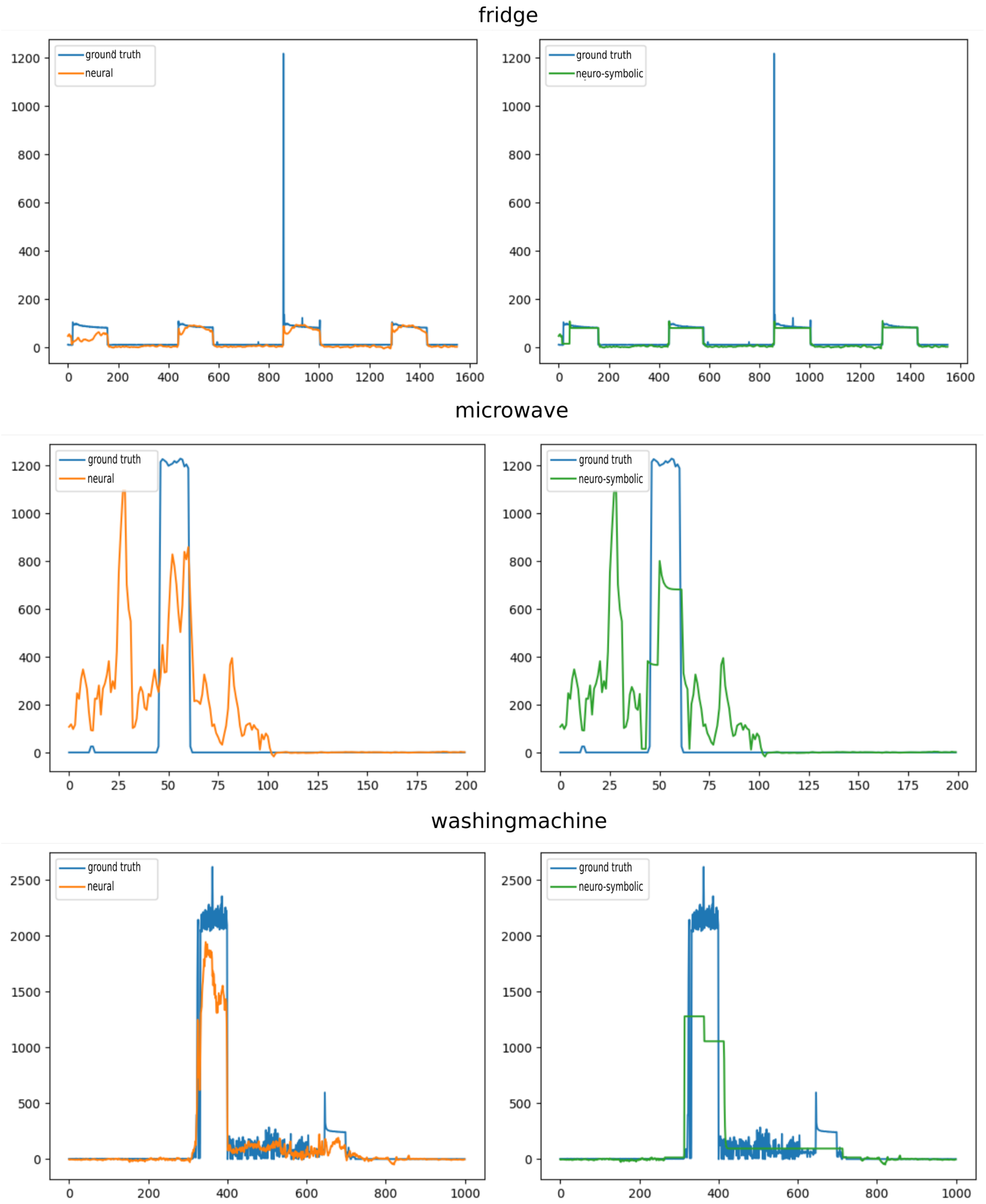


Figure 6.4: Prediction of the consumption for dishwasher, kettle, fridge, microwave, and washingmachine. Ground truth is in blue, while neural and neuro-symbolic are in orange and green, respectively.

6.7 Conclusion

In this chapter, we propose a neuro-symbolic approach for NILM where background knowledge about the behaviour of a house's appliances is used on top of a neural network to refine its predictions. The refinement step is done through a (learnt) automaton that change the prediction of the neural network according to the state's (exponential) function that has been learnt from the data (i.e., from a subset of activation windows). Experiments show that when the prediction of the neural networks are already quite accurate (i.e. the network is able to learn the pattern of consumption of an appliance) but are not consistent over time, the use of the background knowledge is able to ensure a greater consistency leading to a drastic increase of the performance.

Chapter 7

Conclusions and Future Works

In this thesis, we have investigated NeSy AI in the context of event recognition. In detail, our research has focused on giving an answer to the following research questions:

R1: How can we formalize and what kind of formalism can we use to define the event recognition problem in a neuro-symbolic context?

R2: What kind of supervision and background knowledge can we use/exploit in an end-to-end neuro-symbolic approach for event recognition?

R3: What are the issues emerging in those approaches (e.g., scalability)? How can we solve/mitigate them?

R4: Can we ”readapt” these approaches to solve lower tasks with respect to event recognition? For example, using these approaches to predict sequences of values, which have to be compliant with the background knowledge, that may be used to build an event recognition system (e.g, trigger an event warning when certain values are identified in a sequence).

In chapter 3, we addressed **R1** by providing an event calculus inspired formalization of the problem in which events have been defined as structured objects involving different components. In chapter 4, we slightly modified the previous formalization by defining the distinction of events in structured and atomic events. The kind of supervision we provided in chapters 3, 4 and 5, was weak (e.g., supervision on structured event) and limited to a (subset of) the events happening in the sequence. Furthermore, we incorporate simple knowledge involving the relative order of the events and constraints on their duration **R2**. We addressed **R3** in chapters 4 and 5. In chapter 4, we tackle the issue of scalability that emerged in chapter 3, by proposing a neuro-symbolic approach that combines neural prediction over individual frames with a MILP formulation encoding event background knowledge, and we applied it on a real-world event recognition scenario. In chapter 5, the same issue has been addressed by presenting a fuzzy event interval reasoning system based on what we called Interval Real Logic. **R4** has been addressed in chapter 6 where background knowledge has been used to refine the predictions of neural networks which predict the consumptions of an house's appliances. While these results are promising, there are several directions that are left open for future research. For chapters 4 and 5, a major direction consists in increasing the complexity of the scenarios being considered, by dealing with structured events involving multiple actors and complex relationships between the events, without making the underlying reasoning problem prohibitively expensive, something other neuro-symbolic frameworks currently struggle with. In addition, an interesting direction for chapter 4 would be to compare both the the sub-symbolic and our DeepProbLog prototype with a propabilistic graphical model like Conditional Random Fields (CRF) which has been used widely for event detection in text. Modelling the problem as PGM would lead to consider not only complexity and scalabality but other issues as well like features selection (i.e., finding features able to capture both spatial and temporal information) and the capability of the model to capture long term dependencies (i.e., events spans multiple frames and PGM may struggle to model these temporal dependencies). For chapter 5, together with the challenges that have already been discussed in the chapter, it would also be interesting to see how the framework behaves when real data are considered. For chapter 6, one direction could lead to a more in-depth search for the parameters of the CSP. Indeed, we notice that a correct setting of the values of those parameters is fundamental in order to achieve better results. Therefore, an extensive evaluation can be conducted in order to set their values. Currently, one of the drawbacks of the datasets used in NILM is the availability of only few activations for each appliance making their generalization more difficult. Collecting more activations from different houses may increase drastically both the variability and generalizability of our proposed approach. Another direction, which could be also be conducted in parallel with the previous point, would be to move the setting from “seen" to “unseen", and see how our neuro-symbolic approach behaves when the prediction has to be done on a house (i.e., on its appliances) that it has not seen during training Finally, in this thesis, we have focused mainly on event recognition in a single domain (i.e. event recognition in video), then it would be interesting to see how the presented approaches behave when moving to other domains (e.g., text and sound).

Bibliography

[1] Saeid Abbasbandy and T Hajjari. A new approach for ranking of trapezoidal fuzzy numbers. *Computers & mathematics with applications*, 57(3):413–419, 2009.

[2] Kashif Ahmad Ahmad and Nicola Conci. How Deep Features Have Improved Event Recognition in Multimedia: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 15(2):39:1–39:27, 2019.

[3] Elias Alevizos, Anastasios Skarlatidis, Alexander Artikis, and Georgios Paliouras. Probabilistic Complex Event Recognition: A Survey. *ACM Computing Surveys*, 50(5):71:1–71:31, 2017.

[4] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[5] James F. Allen and Patrick J. Hayes. A common-sense theory of time. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence. Los Angeles, CA, USA, August 1985*, pages 528–531. Morgan Kaufmann, 1985.

[6] Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. A neuro-symbolic approach to structured event recognition. In Carlo Combi, Johann Eder, and Mark Reynolds, editors, *28th International Symposium on Temporal Representation and Reasoning, TIME 2021, September 27-29, 2021, Klagenfurt, Austria*, volume 206 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[7] Gianluca Apriceno, Andrea Passerini, and Luciano Serafini. A neuro-symbolic approach for real-world event recognition from weak supervision. In *TIME*, volume 247 of *LIPIcs*, pages 12:1–12:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[8] Alexander Artikis, Evangelos Makris, and Georgios Paliouras. A probabilistic interval-based event calculus for activity recognition. *Annals of Mathematics and Artificial Intelligence*, 89(1-2):29—-52, 2021.

[9] Alexander Artikis, Anastasios Skarlatidis, François Portet, and Georgios Paliouras. Logic-based event recognition. *The Knowledge Engineering Review*, 27(4):469—-506, 2012.

[10] Marco Balletti, Veronica Piccialli, and Antonio M Sudoso. Mixed-integer nonlinear programming for state-based non-intrusive load monitoring. *IEEE Transactions on Smart Grid*, 13(4):3301–3314, 2022.

[11] Marco Balletti, Veronica Piccialli, and Antonio Maria Sudoso. Mixed-integer nonlinear programming for state-based non-intrusive load monitoring. *IEEE Trans. Smart Grid*, 13(4):3301–3314, 2022.

[12] Henri Bouma, Gertjan Burghouts, Leo de Penning, Patrick Hanckmann, Johan-Martijn Hove, Sanne Korzec, Maarten Kruithof, Sander Landsmeer, Coen Leeuwen, Bas Van den Broek, Arvid Halma, Richard den Hollander, and Klamer Schutte. Recognition and localization of relevant human behavior in videos. volume 8711, 04 2013.

[13] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. 35(8), 1986.

[14] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4724–4733. IEEE Computer Society, 2017.

[15] Caviar: context aware vision using image-based active recognition, 2011.

[16] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[17] Dong Chen, David Irwin, and Prashant Shenoy. Smartsim: A device-accurate smart home simulator for energy analytics. In *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 686–692. IEEE, 2016.

[18] Liang-Chieh Chen, Alexander Schwing, Alan Yuille, and Raquel Urtasun. Learning Deep Structured Models. In Francis R. Bach and David M. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1785–1794. JMLR.org, 2015.

[19] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In Dexter Kozen, editor, *Logics of Programs*, pages 52–71, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[20] C. Clavel, T. Ehrette, and G. Richard. Events detection for an audio-based surveillance system. In *2005 IEEE International Conference on Multimedia and Expo*, pages 1306–1309, 2005.

[21] Mário Cordeiro and João Gama. *Online Social Networks Event Detection: A Survey*, pages 1–41. Springer International Publishing, Cham, 2016.

[22] Kellie Corona, Katie Osterdahl, Roderic Collins, and Anthony Hoogs. MEVA: A Large-Scale Multiview, Multimodal Video Dataset for Activity Detection. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2021, Waikoloa, HI, USA, January 3-8, 2021*, pages 1059–1067. IEEE, 2021.

[23] Rui Dai, Srijan Das, and François Brémond. Ctrn: Class-temporal relational network for action detection. In *British Machine Vision Conference*, 2021.

[24] Rui Dai, Srijan Das, Luca Minciullo, Lorenzo Garattoni, Gianpiero Francesca, and Francois Bremond. Pdan: Pyramid dilated attention network for action detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2970–2979, 2021.

[25] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. IJCAI'11, page 819–826. AAAI Press, 2011.

[26] Leo de Penning, Artur Garcez, Luís Lamb, and John-jules Meyer. A neural-symbolic cognitive agent for online learning and reasoning. pages 1653–1658, 01 2011.

[27] Sofia-Maria Dima, Dimitris Tsitsipis, Christos Antonopoulos, John Gialelis, and Stavros Koubias. Flogera — a fuzzy logic event recognition algorithm in a wsn environment. In *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 850–855, 2012.

[28] Ivan Donadello, Luciano Serafini, and Artur S. d'Avila Garcez. Logic tensor networks for semantic image interpretation. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1596–1602. ijcai.org, 2017.

[29] Paolo Dragone, Stefano Teso, and Andrea Passerini. Neuro-symbolic constraint programming for structured prediction. In Artur S. d'Avila Garcez and Ernesto Jiménez-Ruiz, editors, *Proceedings of the 15th International Workshop on Neural-Symbolic*

Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021, volume 2986 of *CEUR Workshop Proceedings*, pages 6–14. CEUR-WS.org, 2021.

[30] Didier Dubois and Henri Prade. Processing fuzzy temporal knowledge. *IEEE Trans. Syst. Man Cybern.*, 19(4):729–744, 1989.

[31] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[32] Corinna Fischer. Feedback on household electricity consumption: A tool for saving energy? *Energy Efficiency*, 1:79–104, 02 2008.

[33] Frederic B. Fitch. Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *The Journal of Symbolic Logic*, 9(2):49–50, 1944.

[34] José Roldán Gómez, Juan Boubeta-Puig, José Luis Martínez, and Guadalupe Ortiz. Integrating complex event processing and machine learning: An intelligent architecture for detecting iot security attacks. *Expert Syst. Appl.*, 149:113251, 2020.

[35] R Gopinath, Mukesh Kumar, C Prakash Chandra Joshua, and Kota Srinivas. Energy management using non-intrusive load monitoring techniques–state-of-the-art and future research directions. *Sustainable Cities and Society*, 62:102411, 2020.

[36] Nicola Guarino, Riccardo Baratella, and Giancarlo Guizzardi. Events, their names, and their synchronic structure. *Applied Ontology*, pages 1–35, 2022.

[37] G.W. Hart. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, 80(12):1870–1891, 1992.

[38] Yassine Himeur, Abdullah Alsalemi, Faycal Bensaali, Abbes Amira, and Ayman Al-Kababji. Recent trends of smart nonintrusive load monitoring in buildings: A review, open challenges, and future directions. *International Journal of Intelligent Systems*, 1, 03 2022.

[39] Pascal Hitzler and Md Kamruzzaman Sarker. *Neuro-Symbolic Artificial Intelligence: The State of the Art.* IOS Press, 2022.

[40] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735—-1780, 1997.

[41] Jeya Vikranth Jeyakumar, Luke Dickens, Luis Garcia, Yu-Hsi Cheng, Diego Ramirez-Echavarria, Joseph Noor, Alessandra Russo, Lance M. Kaplan, Erik Blasch, and

Mani B. Srivastava. Automatic concept extraction for concept bottleneck-based video classification. *CoRR*, abs/2206.10129, 2022.

[42] Woei Tzy Jong, Yuh Shin Shiau, Yih Jen Horng, Hsin Horng Chen, and Shyi Ming Chen. Temporal knowledge representation and reasoning techniques using time petri nets. *IEEE Trans. Syst. Man Cybern. Part B*, 29(4):541–545, 1999.

[43] Kenneth Kahn and G.Anthony Gorry. Mechanizing temporal knowledge. *Artificial Intelligence*, 9(1):87–108, 1977.

[44] Krasimira Kapitanova, Sang H. Son, and Kyoung-Don Kang. Using fuzzy logic for robust event detection in wireless sensor networks. *Ad Hoc Networks*, 10(4):709–722, 2012. Advances in Ad Hoc Networks (II).

[45] Jack Kelly and William Knottenbelt. Neural nilm: Deep neural networks applied to energy disaggregation. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, BuildSys '15, page 55–64, New York, NY, USA, 2015. Association for Computing Machinery.

[46] Abdullah Khan, Loris Bozzato, Luciano Serafini, and Beatrice Lazzerini. Visual Reasoning on Complex Events in Soccer Videos Using Answer Set Programming. In Diego Calvanese and Luca Iocchi, editors, *GCAI 2019. Proceedings of the 5th Global Conference on Artificial Intelligence, Bozen/Bolzano, Italy, 17-19 September 2019*, volume 65, pages 42–53. EasyChair, 2019.

[47] Abdullah Khan, Luciano Serafini, Loris Bozzato, and Beatrice Lazzerini. Event Detection from Video Using Answer Set Programing. In Alberto Casagrande and Eugenio G. Omodeo, editors, *Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019*, volume 2396 of *CEUR Workshop Proceedings*, pages 48–58. CEUR-WS.org, 2019.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[49] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5338–5348. PMLR, 2020.

[50] Robert Kowalski and Marek Sergot. A logic-based calculus of events. *New generation computing*, 4:67–95, 1986.

[51] Robert Kowalski and Marek Sergot. *A Logic-Based Calculus of Events*, pages 23–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[52] Miroslav Kubat. Neural networks: a comprehensive foundation by simon haykin, macmillan, 1994, isbn 0-02-352781-7. *The Knowledge Engineering Review*, 13(4):409–412, 1999.

[53] Paula Lago, Shingo Takeda, Sayeda S Alia, Kohei Adachi, Brahim Bennai, and Sozo Inoue Francois Charpillet. A dataset for complex activity recognition withmicro and macro activities in a cooking scenario. *CoRR*, abs/2006.10681, 2020.

[54] Viet Dac Lai. Event extraction: A survey. *arXiv preprint arXiv:2210.03419*, 2022.

[55] Weiyao Lin, Huabin Liu, Shizhan Liu, Yuxi Li, Rui Qian, Tao Wang, Ning Xu, Hongkai Xiong, Guo-Jun Qi, and Nicu Sebe. Human in Events: A Large-Scale Benchmark for Human-centric Video Analysis in Complex Events, 2020.

[56] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc D. Raedt. DeepProbLog: Neural Probabilistic Logic Programming. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, volume 31, pages 3753–3763, 2018.

[57] Robin Manhaeve, Giuseppe Marra, and Luc De Raedt. Approximate Inference for Neural Probabilistic Logic Programming. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, pages 475–486, 11 2021.

[58] Francesco Manigrasso, Filomeno Davide Miro, Lia Morra, and Fabrizio Lamberti. Faster-ltn: A neuro-symbolic, end-to-end object detection architecture. In *ICANN (2)*, volume 12892 of *Lecture Notes in Computer Science*, pages 40–52. Springer, 2021.

[59] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: A General Interface Layer to Integrate Logic Inference and deep Learning. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany,*

September 16-20, 2019, Proceedings, Part II, volume 11907 of *Lecture Notes in Computer Science*, pages 283—-298, 2019.

[60] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Integrating Learning and Reasoning with Deep Logic Models. In Ulf Brefeld, Élisa Fromont, Andreas Hotho, Arno J. Knobbe, Marloes H. Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*, volume 11907 of *Lecture Notes in Computer Science*, pages 517–532. Springer, 2020.

[61] Adrienne Mayor. *Gods and Robots: Myths, Machines, and Ancient Dreams of Technology.* Princeton University Press, USA, 2018.

[62] John McCarthy. What is artificial intelligence? 01 2004.

[63] Lalatendu Muduli, Prasanta K. Jana, and Devi Prasad Mishra. Wireless sensor network based fire monitoring in underground coal mines: A fuzzy logic approach. *Process Safety and Environmental Protection*, 113:435–447, 2018.

[64] Erik T Mueller. Event calculus. *Foundations of Artificial Intelligence*, 3:671–708, 2008.

[65] Gabor Nagypal and Boris Motik. A fuzzy model for representing uncertain, subjective, and vague temporal knowledge in ontologies. In Robert Meersman, Zahir Tari, and Douglas C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003*, volume 2888 of *Lecture Notes in Computer Science*, pages 906–923. Springer, 2003.

[66] Christoforos Nalmpantis, Nikolaos Virtsionis Gkalinikis, and Dimitris Vrakas. Neural fourier energy disaggregation. *Sensors*, 22(2):473, 2022.

[67] Juan C. Niebles, Chih-Wei Chen, and Li Fei-Fei. Modeling Temporal Structure of Decomposable Motion Segments for Activity Classification. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part II*, volume 6312 of *Lecture Notes in Computer Science*, pages 392–405. Springer, 2010.

[68] Chigozie Nwankpa, Winifred L. Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *ArXiv*, abs/1811.03378, 2018.

[69] Hans Jürgen Ohlbach. Relations between fuzzy time intervals. In *Proceedings. 11th International Symposium on Temporal Representation and Reasoning, 2004. TIME 2004.*, pages 44–51. IEEE, 2004.

[70] Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2021.

[71] Veronica Piccialli and Antonio M. Sudoso. Improving non-intrusive load disaggregation through an attention-based deep neural network. *Energies*, 14(4), 2021.

[72] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977.

[73] Luc D. Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A Probabilistic Prolog and Its Application in Link Discovery. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2462–2467, 2007.

[74] Fabrício Henrique Rodrigues and Mara Abel. What to consider about events: A survey on the ontology of occurrents. *Applied Ontology*, 14(4):343–378, 2019.

[75] Steven Schockaert and Martine De Cock. Temporal reasoning about fuzzy intervals. *Artif. Intell.*, 172(8-9):1158–1193, 2008.

[76] Steven Schockaert, Martine De Cock, and Etienne E Kerre. Fuzzifying allen's temporal interval relations. *IEEE Transactions on Fuzzy Systems*, 16(2):517–533, 2008.

[77] Roberto Sebastiani and Silvia Tomasi. Optimization Modulo Theories with Linear Rational Costs. *ACM Transactions on Computational Logics*, 16(2), 2015.

[78] Luciano Serafini and Artur d'Avila Garcez. Learning and Reasoning with Logic Tensor Networks. In Giovanni Adorni, Stefano Cagnoni, Marco Gori, and Marco Maratea, editors, *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, volume 10037 of *Lecture Notes in Computer Science*, pages 334—-348, 2016.

[79] Murray Shanahan. The event calculus explained. In *Artificial intelligence today*, pages 409–430. Springer, 1999.

[80] Anastasios Skarlatidis, Alexander Artikis, Jason Filippou, and Georgios Paliouras. A probabilistic logic programming event calculus. *Theory and Practice of Logic Programming*, 15(2):213—-245, 2015.

[81] Khurram Soomro, Amir R. Zamir, and Mubarak Shah. UCF101: A dataset of 101 Human Actions Classes From Videos in The Wild. *CoRR*, abs/1212.0402, 2012.

[82] Praveen Tirupattur, Kevin Duarte, Yogesh Singh Rawat, and Mubarak Shah. Modeling multi-label action dependencies for temporal action localization. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 1460–1470. Computer Vision Foundation / IEEE, 2021.

[83] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[84] Elena Umili, Roberto Capobianco, and Giuseppe De Giacomo. Grounding ltlf specifications in images. In *NeSy*, volume 3212 of *CEUR Workshop Proceedings*, pages 45–63. CEUR-WS.org, 2022.

[85] Marc Roig Vilamala, Liam Hiley, Yulia Hicks, Alun D. Preece, and Federico Cerutti. A pilot study on detecting violence in videos fusing proxy models. In *22th International Conference on Information Fusion, FUSION 2019, Ottawa, ON, Canada, July 2-5, 2019*, pages 1–8. IEEE, 2019.

[86] Marc Roig Vilamala, Tianwei Xing, Harrison Taylor, Luis Garcia, Mani B. Srivastava, Lance M. Kaplan, Alun D. Preece, Angelika Kimmig, and Federico Cerutti. Using deepproblog to perform complex event processing on an audio stream. *CoRR*, abs/2110.08090, 2021.

[87] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. Deepstochlog: Neural stochastic logic programming. *CoRR*, abs/2106.12574, 2021.

[88] Wei Xiang and Band Wan. A Survey of Event Extraction From Text. *IEEE Access*, 7:173111–173137, 2019.

[89] Wei Xiang and Bang Wang. A survey of event extraction from text. *IEEE Access*, 7:173111–173137, 2019.

[90] Tianwei Xing, Luis Garcia, Marc Roig Vilamala, Federico Cerutti, Lance M. Kaplan, Alun D. Preece, and Mani B. Srivastava. Neuroplex: learning to detect complex events in sensor networks through knowledge injection. In Jin Nakazawa and Polly

Huang, editors, *SenSys '20: The 18th ACM Conference on Embedded Networked Sensor Systems, Virtual Event, Japan, November 16-19, 2020*, pages 489–502. ACM, 2020.

[91] Tianwei Xing, Marc R. Vilamala, Luis Garcia, Federico Cerutti, Lance Kaplan, Alun Preece, and Mani Srivastava. DeepCEP: Deep Complex Event Processing Using Distributed Multimodal Information. In *IEEE International Conference on Smart Computing, SMARTCOMP 2019, Washington, DC, USA, June 12-15, 2019*, pages 87–92. IEEE, 2019.

[92] Zhun Yang, Adam Ishay, and Joohyung Lee. NeurASP: Embracing Neural Networks into Answer Set Programming. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1755–1762. ijcai.org, 2020.

[93] Serena Yeung, Olga Russakovsky, Ning Jin, Mykhaylo Andriluka, Greg Mori, and Li Fei-Fei. Every moment counts: Dense detailed labeling of actions in complex videos. *Int. J. Comput. Vis.*, 126(2-4):375–389, 2018.

[94] Shujuan Yin, Weizhong Zhao, Xingpeng Jiang, and Tingting He. Knowledge-aware few-shot learning framework for biomedical event trigger identification. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 375–380, 2020.

[95] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[96] Michael Zeifman and Kurt Roth. Nonintrusive appliance load monitoring: Review and outlook, 2011.

[97] Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton. Sequence-to-point learning with neural networks for non-intrusive load monitoring. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018.

Appendix A

Appendix

A.1 MILP for high jump

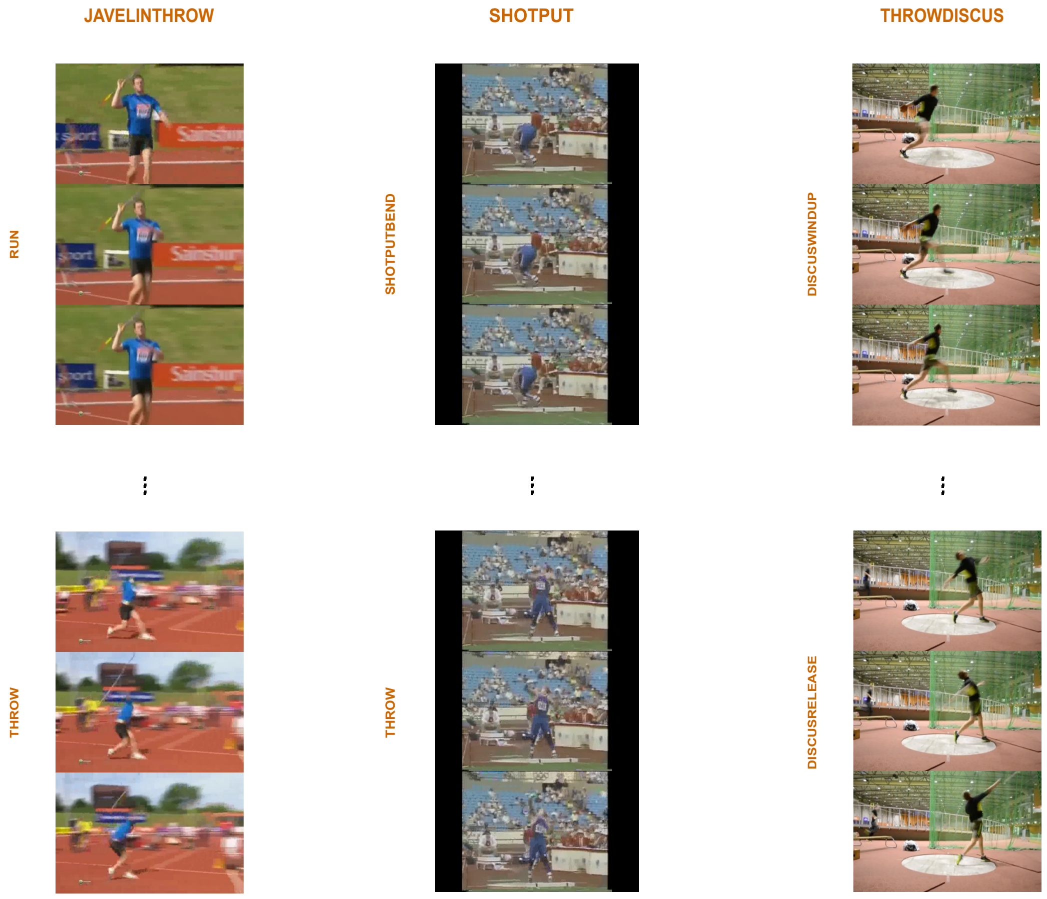
Listing A.1: Encoding high jump as a MILP using MiniZinc

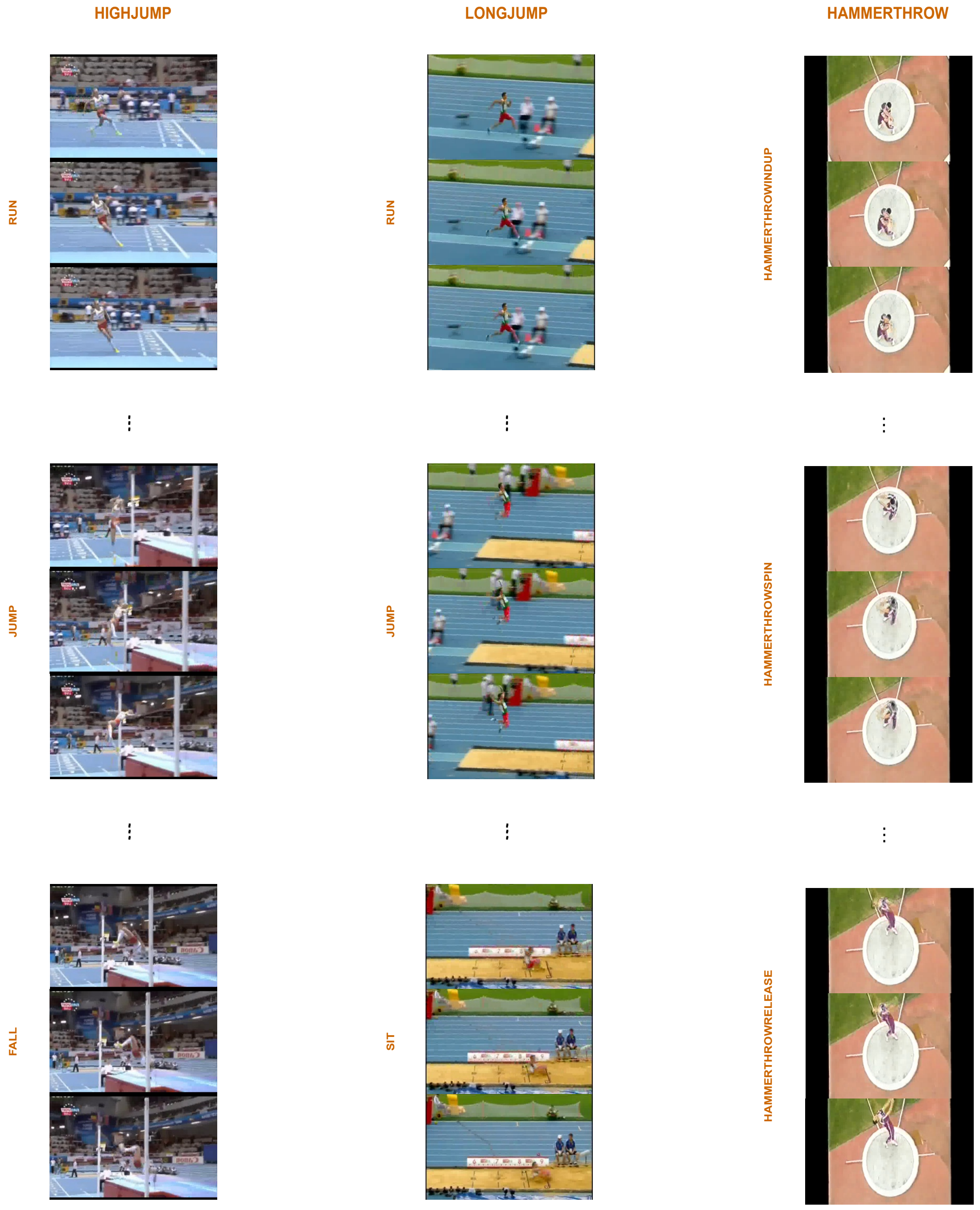
```
% HJ = High Jump
% R  = Run
% J  = Jump
% F  = Fall

int: bHJ;
int: eHJ;

int: minSum_R_J = 3;
int: maxSum_R_J = 48;
int: minSum_R_F = 4;
int: maxSum_R_F = 57;
int: minSum_J_F = 3;
int: maxSum_J_F = 33;

int: target_R_J = maxSum_R_J - minSum_R_J + 1;
int: target_R_F = maxSum_R_F - minSum_R_F + 1;
int: target_J_F = maxSum_J_F - minSum_J_F + 1;

var bHJ .. eHJ: bR;
var bHJ .. eHJ: eR;
var bHJ .. eHJ: bJ;
var bHJ .. eHJ: eJ;
var bHJ .. eHJ: bF;
var bHJ .. eHJ: eF;

var int: lenR = eR - bR + 1;
var int: lenJ = eJ - bJ + 1;
var int: lenF = eF - bF + 1;


constraint eR > bR /\ eJ > bJ /\ eF > bF;
constraint bR == bHJ /\ eR == (bJ-1) /\ eJ == (bF-1) /\ eF == eHJ;
constraint lenR >= (lenJ + lenF) /\ lenJ < (lenR + lenF) /\ lenF < (lenR + lenJ);


var int: cost_comp_run_pos = - sum (t in bR..eR) (ae_predictions[1,t]);
var int: cost_comp_run_neg = sum (t in (eR+1)..eHJ) (ae_predictions[1,t]);
var int: cost_comp_jump_pos = - sum (t in bJ..eJ) (ae_predictions[2,t]);
var int: cost_comp_jump_neg_1 = sum (t in bHJ..(bJ-1)) (ae_predictions[2,t]);
var int: cost_comp_jump_neg_2 = sum (t in (eJ+1)..eHJ) (ae_predictions[2,t]);
var int: cost_comp_fall_pos = - sum (t in bF..eF) (ae_predictions[3,t]);
var int: cost_comp_fall_neg = sum (t in bHJ..(bF-1)) (ae_predictions[3,t]);
```

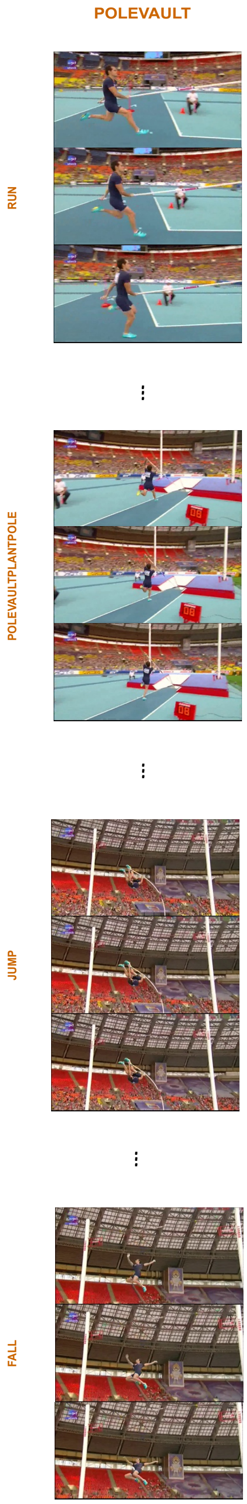
```
44
45  var int: cost = (
46    cost_comp_run_pos + cost_comp_run_neg
47    + cost_comp_jump_pos + cost_comp_jump_neg_1 + cost_comp_jump_neg_2
48    + cost_comp_fall_pos + cost_comp_fall_neg
49    + 1000 * abs(target_R_J - (lenR + lenJ))
50    + 1000 * abs(target_R_F - (lenR + lenF))
51    + 1000 * abs(target_J_F - (lenJ + lenF))
52  );
53
54  solve minimize cost;
```

In lines 6-7, the constants representing the begin and the end of the clip are declared. These are going to be filled at running time and will change depending on the length of the clip processed. The blocks of lines 9-14 and 16-18 contains, the minimum/maximum sum of the length of the intervals of two atomic events and the target length in which the sum of the two intervals has to lie in. The declaration of the optimizer decision variables is contained in line 20-25. These variables are going to be set by Gurobi at the end of the optimization. In lines 27-29, the variables representing the length of the interval of each atomic events are defined. Lines 32-34 represent hard constraints that Gurobi has to satisfy. In details, line 32 states that the end of each atomic event has to be greater than their corresponding begin, while lines 33 and 34 defines respectively temporal relations among events and algebraic constraints among the length of the intervals of atomic events. In addition to the satisfaction of those constraints, Gurobi has to minimize a cost function (lines 45-62). The cost function can be split in two parts. The former (lines 46-48) is composed by the sum of components defined in lines 37-43 where we want to maximize (i.e. minimize) the sum of probability where the solver states the atomic events are happening (pos), and minimize (i.e. maximize) the sum of probability where the atomic events are not happening (neg). The latter (lines 49-51) refers to soft constraints where the solver has to set the the sum of the length of two atomic events' intervals to be as closed as possible to their target length. In this case, the value of weights (i.e., 1000) have been chosen after trying the following values: 250, 500, 750 and 1000.

A.2 Structured and atomic events considered




HIGHJUMP
LONGJUMP
HAMMERTHROW
RUN
RUN
HAMMERTHROWINDUP
JUMP
JUMP
HAMMERTHROWSPIN
FALL
SIT
HAMMERTHROWRELEASE


POLEVAULT
RUN
POLEVAULTPLANTPOLE
JUMP
FALL

A.3 Temporal action localization model

The input of the model consists in a series of feature vectors $\boldsymbol{f}$ extracted by a pre-trained two-stream I3D [14], where each $f_i \in \boldsymbol{f}$ corresponds to 8 frames (or 0.33 seconds) and contains global information at both frame and video-clip level. A non linear transformation is applied on these features in order to obtain class level features $(C \times T \times H)$ with C representing the number of classes, T the number of timestamps and H the dimension of embedding space. Then, the class-level features are refined using L attention-based Multi-Label Action Dependency (MLAD) layers. These layers are composed by two disjoint branches which adopt a self-attention operation to model the relationships between actions that happened within the same timestamp (referred as Co-occurence Dependency branch) and actions happening at different timestamps (referred as Temporal Dependency branch). As a result, a refined set of features is returned by each branch, respectively. At the end, a linear combination of the two branches' features is applied (through a learnt value $\alpha \in [0, 1]$) and the result is passed to C individual classification layers which outputs class probabilities for each timestamp $(T \times C)$.