Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Recognizing Human Actions and Goals in an Open Environment: A Brain-Inspired Approach

by
Franz Alexander Van-Horenbeke Echevarria

In Partial Fulfillment of the Requirements for the Degree of

Ph.D. in Advanced-Systems Engineering

35^{th} Cycle

Supervisor: Angelika Peer

Second Supervisor: Tamim Asfour

Exam Date:

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Angelika Peer, as well as my second supervisor Prof. Tamim Asfour, for their guidance and valuable support during the development of this thesis, as well as for their trust in me and the opportunities they offered me. Additionally, I would like to express my gratitude to EGTC Europaregion Tirol-Südtirol-Trentino for financing this research.

I would also like to thank my colleagues at the Human-centered Technologies and Machine Intelligence Lab for being always available and ready to help, and for their invaluable company and support, with a special mention to Elias and Isabel, who accompanied me during this whole journey. I am also grateful to the people at the High Performance Humanoid Technologies Lab, who helped me and guided me during my visiting period there, as well as to the participants of my study, and, in particular, to André, who kept providing me support even after my period at their lab.

Lastly, I would like to thank my family and friends, and, especially, my parents, sister and grandma, for their infinite patience and unconditional support.

ABSTRACT

The ability to understand human actions, plans, and goals in real environments will be an important requirement for future robotic systems to interact with humans in intuitive and safe ways. Indeed, we humans make use of this skill constantly in our daily lives when interacting with or observing others, which allows us to have more efficient communications and interactions. Consequently, if we want to achieve natural and intuitive human-robot interactions, it seems necessary to develop systems able to recognize actions and plans at a close-to-human performance. However, current computational systems are still far from the remarkably good human performance in this task, especially when it comes to real open environments, where the number of actions and plans that can be observed seems unbounded. A possible approach to overcome these limitations is to try to understand and mimic some of the main brain mechanisms that allow us humans to perform so well in this task. This thesis explores this direction by bringing ideas from the structure and function of the neocortex to the well-established fields of machine learning and neural networks. The resultant system is a hierarchical architecture that learns representations of the input temporal data at different levels of abstraction in a self-supervised way. Once the representations have been learned, classifiers can be trained on top of them that are simpler, perform better, and require less training samples than those trained directly over the input, making the system better adapted to open environments. The system has been shown to outperform other state-of-the-art self-supervised representation learning systems for temporal data in this task on different datasets, both in terms of the accuracies and F_1 scores achieved and in terms of the number of training samples required. Its internal behavior has also been compared against that of the primary visual cortex and shown to be analogous, validating it as a computational model of this (and possibly other) areas of the neocortex. In addition, the architecture was designed to be easily adaptable to other applications besides classification, including fully unsupervised applications. Following the same brain-inspired approach, it has already been extended to action and goal prediction and action selection tasks.

ZUSAMMENFASSUNG

Die Fähigkeit, menschliche Handlungen, Pläne und Ziele in realen Umgebungen zu verstehen, wird eine wichtige Voraussetzung dafür sein, dass zukünftige Robotersysteme auf intuitive und sichere Weise mit Menschen interagieren können. Tatsächlich nutzen wir Menschen diese Fähigkeit in unserem täglichen Leben ständig, wenn wir mit anderen interagieren oder sie beobachten, was uns eine effizientere Kommunikation und Interaktion ermöglicht. Wenn wir natürliche und intuitive Mensch-Roboter-Interaktionen erreichen wollen, scheint es daher notwendig, Systeme zu entwickeln, die in der Lage sind, Aktionen und Pläne mit einer menschenähnlichen Leistung zu erkennen. Allerdings sind aktuelle Computersysteme bei dieser Aufgabe noch weit von der bemerkenswert guten Leistung des Menschen entfernt, insbesondere wenn es um reale Umgebungen geht, in denen die Anzahl der Aktionen und Pläne, die beobachtet werden können, unbegrenzt zu sein scheint. Ein möglicher Ansatz zur Überwindung dieser Einschränkungen besteht darin, zu versuchen, einige der wichtigsten Gehirnmechanismen zu verstehen und nachzuahmen, die es uns Menschen ermöglichen, diese Aufgabe so gut zu bewältigen. Diese Arbeit untersucht diese Richtung, indem sie Ideen aus der Struktur und Funktion des Neokortex in die etablierten Bereiche des maschinellen Lernens und neuronaler Netze überführt. Das resultierende System ist eine hierarchische Architektur, die auf selbstüberwachte Weise Darstellungen der zeitlichen Eingabedaten auf verschiedenen Abstraktionsebenen lernt. Sobald die Darstellungen gelernt wurden, können darauf Klassifikatoren trainiert werden, die einfacher sind, eine bessere Leistung erbringen und weniger Trainingsbeispiele erfordern als diejenigen, die direkt über den Eingabedaten trainiert werden, wodurch das System besser an offene Umgebungen angepasst wird. Es hat sich gezeigt, dass das System andere hochmoderne selbstüberwachte Darstellungslernsysteme für zeitliche Daten bei dieser Aufgabe an verschiedenen Datensätzen übertrifft, sowohl hinsichtlich der erreichten Genauigkeiten und F_1-Scores als auch hinsichtlich der Anzahl der erforderlichen Trainingsbeispiele. Sein internes Verhalten wurde auch mit dem des primären visuellen Kortex verglichen und erwies sich als analog, was es als Rechenmodell dieses (und möglicherweise anderer) Bereiche des Neokortex bestätigte. Darüber hinaus wurde die Architektur so konzipiert, dass sie sich leicht an andere Anwendungen neben der Klassifizierung anpassen lässt, einschließlich vollständig unbeaufsichtigter Anwendungen. Dem gleichen gehirninspirierten Ansatz folgend, wurde sie bereits auf Aufgaben zur Aktions- und Zielvorhersage sowie zur Aktionsauswahl erweitert.

RIASSUNTO

La capacità di comprendere le azioni, i piani e gli obiettivi umani negli ambienti reali sarà un requisito importante affinché i futuri sistemi robotici possano interagire con gli esseri umani in modo intuitivo e sicuro. In effetti, noi esseri umani utilizziamo costantemente questa abilità nella nostra vita quotidiana quando interagiamo o osserviamo gli altri, il che ci consente di avere comunicazioni e interazioni più efficienti. Di conseguenza, se vogliamo ottenere interazioni uomo-robot naturali e intuitive, sembra necessario sviluppare sistemi in grado di riconoscere azioni e piani con prestazioni prossime a quelle umane. Tuttavia, gli attuali sistemi computazionali sono ancora lontani dalle prestazioni umane straordinariamente buone in questo compito, soprattutto quando si tratta di ambienti reali, dove il numero di azioni e piani che possono essere osservati sembra illimitato. Un possibile approccio per superare queste limitazioni è cercare di comprendere e imitare alcuni dei principali meccanismi cerebrali che consentono a noi esseri umani di svolgere così bene questo compito. Questa tesi esplora questa direzione portando idee dalla struttura e dalla funzione della neocorteccia ai campi consolidati dell'apprendimento automatico e delle reti neurali. Il sistema risultante è un'architettura gerarchica che apprende le rappresentazioni dei dati temporali di input a diversi livelli di astrazione in modo auto-supervisionato. Una volta apprese le rappresentazioni, è possibile addestrare su di esse classificatori che sono più semplici, funzionano meglio e richiedono meno campioni di addestramento rispetto a quelli addestrati direttamente sull'input, rendendo il sistema più adatto agli ambienti aperti. È stato dimostrato che il sistema supera gli altri sistemi all'avanguardia di apprendimento della rappresentazione autosupervisionata per i dati temporali in questo compito su diversi set di dati, sia in termini di precisione e F_1 scores ottenuti, sia in termini di numero di campioni di addestramento necessario. Il suo comportamento interno è stato anche confrontato con quello della corteccia visiva primaria e si è dimostrato analogo, validandolo come modello computazionale di questa (e forse di altre) aree della neocorteccia. Inoltre, l'architettura è stata progettata per essere facilmente adattabile ad altre applicazioni oltre alla classificazione, comprese le applicazioni completamente non supervisionate. Seguendo lo stesso approccio ispirato al cervello, è già stata estesa alle attività di previsione di azioni e obiettivi e di selezione delle azioni.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Acronyms .

ACT-R. Adaptive control of thought-rational.

ANN. Artificial neural network.

ART. Adaptive resonance theory.

BDI. Belief-desire-intention software model.

BERT. Bidirectional encoder representations from transformers.

BN. Bayesian network.

BPTT. Backpropagation through time.

BTSF. Bilinear temporal-spectral fusion.

CNN. Convolutional neural network.

CPC. Contrastive predictive coding.

CRF. Conditional random field.

DBN. Deep belief network.

DBN. Dynamic Bayesian network.

DNN. Deep neural network.

ELU. Exponential linear unit.

FSDD. Free Spoken Digit Dataset.

GAN. Generative adversarial network.

GCN. Graph convolutional network.

GRU. Gated recurrent unit.

HAMMER. Hierarchical, attentive, multiple models for execution and recognition.

HHMM. Hierarchical hidden Markov model.

HLRNN. Hierarchical locally recurrent neural network.

HMM. Hidden Markov model.

HTM. Hierarchical temporal memory.

HTN. Hierarchical task network.

KL. Kullback–Leibler.

KNN. K-nearest neighbors.

LDA. Linear discriminant analysis.

LHMM. Layered hidden Markov model.

LLM. Large language model.

LRB. Locally recurrent block.

LSTM. Long short-term memory.

MDN. Mixture density network.

MDP. Markov decision process.

MFCC. Mel-frequency cepstral coefficients.

MLN. Markov logic network.

MLP. Multilayer perceptron.

MNS2. Mirror neuron system II.

MOSAIC. Modular selection and identification for control.

MSE. Minimum squared error.

MTRNN. Multiple timescales recurrent neural network.

NILRNN. Neocortex-inspired locally recurrent neural network.

NLL. Negative log likelihood.

PCA. Principal component analysis.

POMDP. Partially observable Markov decision process.

RMM. Relational Markov model.

RNN. Recurrent neural network.

SFA. Slow feature analysis.

SIFT. Scale-invariant feature transform.

SimCLR. Simple framework for contrastive learning of visual representations.

SRL. Statistical relational learning.

SVM. Support vector machine.

t-SNE. t-distributed stochastic neighbor embedding.

TS-TCC. Time series representation learning framework via temporal and contextual contrasting.

TST. Time series transformer.

U-LRNN. U-shaped locally recurrent neural network.

VAE. Variational autoencoder.

WARD. Wearable Action Recognition Database.

Formalization symbols .

$\bar{\cdot}$**.** History.

$\emptyset$**.** Empty set.

$\hat{\cdot}$**.** Tuple element (or set of tuple elements).

$\pi(\cdot)$**.** Policy.

$\tilde{\cdot}$**.** Estimated.

a**.** Action.

A^{by}**.** Action space.

A^{over}**.** Affordance space.

$E(\cdot)$**.** Emission function.

$F(\cdot)$**.** Function.

G**.** Goal space.

g**.** Goal.

H**.** Agent.

i**.** Index.

j**.** Index.

K**.** Knowledge space.

k**.** Knowledge.

$M(\cdot)$**.** Sensor model.

N**.** Environment.

n**.** Number of elements.

o**.** Observation/observable state.

O^{by}**.** Observation space.

O^{from}**.** Observable state space.

P**.** Activity, plan, and/or goal recognition problem.

p**.** Probability.

S**.** State space.

s**.** State.

$T(\cdot)$**.** Transition function.

X**.** Actor.

Y**.** Observer.

Other symbols .

α**.** Optimization algorithm stepsize.

β**.** Sparsity loss term weight.

$\cdot^{\perp}$**.** Orthogonal.

δ**.** Slowness-oriented contrastive loss term maximum sample distance.

$\delta(x)$**.** Dirac delta function.

$\eta_{\downarrow}$**.** Downsampling scale factor (or temporal stride).

$\eta_{\hookrightarrow}$**.** Function shrinking factor.

$\eta_{\uparrow}$**.** Upsampling scale factor.

γ**.** Slowness-oriented contrastive loss term weight.

$\hat{\cdot}$**.** Estimated.

κ**.** Concentration parameter.

λ**.** L_2 regularization loss term weight.

μ**.** Mean.

ω**.** Window (or temporal kernel) size.

π**.** Mixture distribution coefficient (or weight).

ψ**.** Attention focus loss term weight.

ρ**.** Average neuron activity (i.e., sparsity) vector.

σ**.** Standard deviation.

τ**.** Delay.

a. Layer activity vector.

b. Neural network learnable bias units.

c. Context representation vector.

d. Neural network variable-size input element size.

$D_{KL}(\cdot||\cdot)$. Kullback-Leibler divergence.

f. Kernel (or filter) size.

$f_{\cdot}(\cdot)$. Probability density function.

g. Grid cell center coordinates.

h. Number of attention heads.

$h_{W,b}(\cdot)$. Neural network function (i.e., hypothesis).

i. Index.

J. Loss term.

j. Index.

$J(W, b)$. Loss function.

k. Truncated backpropagation through time truncation horizon.

l_h. 2D layer height.

l_w. 2D layer width.

m. Batch size (i.e., number of samples).

N. Neural network depth (i.e., number of layers or blocks).

n. Number of.

p. Dropout layer masking probability.

q. Query vector.

s. Layer size (i.e., number of neurons).

t. Stride.

$u(x)$. Heaviside step function.

W. Neural network learnable weights.

w. Weight vector.

X. Neural network variable-size input matrix.

x. Neural network input vector.

y. Neural network target output vector.

z. Local representation vector.

Chapter 1

INTRODUCTION

1.1 Problem Statement

Robots are becoming more and more common in our daily lifes (e.g., self-driving cars, collaborative robots at the factory, etc.), and their ubiquity will probably keep increasing in the near future [1]. Such expansion is motivated by different factors and applications, from optimizing processes in factories or reducing worker risks in dangerous environments to helping with domestic tasks, or even taking care of the elder, which may become a real need in the future due to the growth of the elderly population and the decrease of social and healthcare providers [2]. In order to be up to the needs and demands of such transition, robots will be required to be able to interact with humans in more natural, intuitive, and effective ways, both in terms of physical and social interaction. This involves endowing robots with skills that are typically associated to humans, such as communicating through natural language or understanding the internal state of others. Indeed, research has shown that humans tend to interact with human-like robots in similar social ways to how they do with other humans [3]. Hence, building robots that have social skills similar to those of humans seems the way to go to achieve natural and intuitive human-robot interactions.

One of the multiple skills involved in human-human collaboration that we may be interested in replicating in robots is the ability to recognize and understand what an observed human is doing and what the goal of that action is. Indeed, when we humans interact with or observe other people, we are constantly and effortlessly making sense of and understanding their actions and intentions. This allows us to understand the internal state of others, predict their next actions, share a space, collaborate or compete in more effective ways, and, in general, interact intuitively and naturally. However, while this is something that we humans perform with little effort and without even being aware of it, it is not a trivial task from the computational point of view, and the existing algorithms are still far from human performance, especially when it comes to real unconstrained environments, where the number of possible actions and goals seems unbounded.

Indeed, typical solutions to endow artificial systems with this ability make use of artificial neural networks trained in supervised ways and/or of knowledge-based plan recognition systems. Both approaches generally require that all possible actions are known beforehand, making them not appropriate for real open environments where new actions are often observed. There have also been attempts to develop systems able to recognize any type of action through zero-shot learning techniques. While, recently, some of these systems have been able to reach quite good

performances (e.g., [4]), they are not well suited for other related tasks that may be required in a robotics application, such as learning online to recognize new domain-specific actions or learning (or updating knowledge about) how a known action is specifically carried out in a certain context (e.g., putting things in boxes vs. arranging them in boxes according to some relevant criterion). These skills may be essential, for example, in a human-robot collaboration scenario where the robot is supposed to help the human in the task once it has recognized what the human is doing. In addition, these systems are in general not designed to perform segmentation in time, as each sample or recording they are fed with belongs to a single action, and they typically count on the whole action sequence, from the beginning to the end of the action. These characteristics are not appropriate for a robotics scenario where the human-robot interaction occurs and the ongoing action needs to be detected and updated (and often also predicted) in real time.

An alternative approach to this problem is to mimic the mechanisms from our brain that are believed to be involved in this task. This approach has several motivations: First, we are actually trying to mimic a human skill, so mimicking also how it is carried out seems a good starting point. Second, we are indeed very good at it, which means that, if we manage to properly replicate the underlying processes, we should be able to achieve good performances. And third, as we mentioned earlier, in human-robot applications in general, we are not only interested in the typical problem of action, plan, and goal recognition but also in associated problems such as understanding how an action is being performed. Since we humans also perform good in these tasks, a brain-inspired action and goal recognition system may be better adapted to address them if necessary. What is more, considering that such system would be designed to be part of a robotic system that also implements other human-like social skills, if they are implemented in a similar way, this could lead to a simpler and more homogenous brain-like integration, with components or architectures that may be involved in different functions (as occurs in our brain). While it is true that there is still much to be discovered in order to have a complete understanding of the functioning of the human brain, there is already plenty of knowledge and models available about its structure and the mechanisms and functions that rule its operation, which motivates the exploration of this direction, as it may lead to new successful systems able to address challenges that more conventional artificial intelligence systems usually struggle with.

In fact, multiple brain-inspired approaches have been already proposed that address the problem of action recognition as well as related problems (e.g., [5]). However, these approaches usually have limitations such as having architectures that are very specific to the problem, making them not so appropriate to be extended in functionality or to be integrated with other modules in relatively homogenous complete solutions. Instead, a brain-inspired approach that relies on more general models of regions of the brain to address the problem at hand may be better suited to be extended with additional functionality or to be integrated with models of other regions of the brain, probably leading to better integrated and more brain-like complete solutions at

different levels. What is more, such models and solutions are often used as computational models of the brain to study its functioning, contributing in this way to a better understanding of it and to potentially better future brain-inspired solutions. This is the direction that has been taken in this thesis.

1.2 State of the Art

The content of this section was published in [6].

1.2.1 Introduction

Developing artificial intelligence systems that address the problem of activity, plan, and goal recognition is something that has been ongoing for decades, with a large amount of systems with very diverse characteristics having been proposed tackling different challenges and designed to be used in different applications and domains, from surveillance to video games. Most of these systems focus only on part of the problem (e.g., activity recognition, plan recognition, etc.), with techniques to address the complete problem not being so thoroughly explored. Still, in this section we try to provide an overview of the problem as a whole. This holistic view may inspire the development of new solutions that address the problem in a unified way.

This section reviews and surveys the literature around the topic of activity, plan, and goal recognition. It is organized as follows: Section 1.2.2 presents an overview of the main mechanisms that are believed to be behind this ability in humans. Section 1.2.3 introduces some of the concepts and challenges involved in the problem. Section 1.2.4 describes and compares the main types of approaches that have been proposed to address it. Finally, Section 1.2.5 discusses on this literature study, the challenges that are still to be faced, and possible future directions.

1.2.2 Action, Plan, and Intention Recognition in Humans

The ability to attribute mental states to others is what philosophers and psychologists call **theory of mind** [7]. The term refers to our presumption that others have a mind, as we do not have direct access to it and just infer its existence from observations [7]. This ability allows us, first, to understand that other people's thoughts may be different from ours and, second, to think about what others (and also ourselves) are thinking [8], including emotions, desires, intentions, beliefs, and knowledge. While it is related to the concept of empathy, it is not the same: Empathy refers to emotional perspective-taking, while theory of mind concerns cognitive perspective-taking [9]. This ability contributes to social skills such as engaging in meaningful conversations, resolving conflicts, maintaining intimacy in friendships, and being more socially competent in general [10]. In particular, it allows us to understand why someone acts in a certain way and to predict how someone will act [11]. In fact, several research studies have shown that humans attribute plans and goals to observed agents performing sequences of actions and are able to

predict the next actions [12], [13]. All these skills also seem to contribute to executive function (which is responsible for the cognitive control of behavior), and this contribution seems to be bidirectional. Indeed, social competence has been shown to take part in the development of executive function [14] and vice versa [15], with children with lower levels of social competence showing deficits in executive function. An interesting fact about the theory of mind is that children start developing it at around age 3-4 [11], and, therefore, younger children are unable to understand that other people's beliefs or knowledge may be different from their own [16]. Then, these abilities continue being developed along adolescence and into adulthood.

There are mainly two theories that try to describe how this theory of mind works: the theory-theory and the simulation theory. The **theory-theory** states that humans hold a basic theory of psychology that allows them to infer the mental states of others [17]. According to this theory, children develop this ability by observing the world, gathering data, and revising their theories or beliefs accordingly [18], allowing them to better understand the intentions of others and predict their behavior. A detailed model that tries to describe the mechanisms behind this theory is the BDI (belief-desire-intention) model [19] (which has been used in the development of intelligent software agents).

Regarding **simulation theory**, it states that we infer the intentions and future actions of others by putting ourselves in their place and simulating their cognition, using our mind as a model of theirs [20]. Hence, this inference implies activating mental states that, if carried into action, they would produce a similar behavior to the one observed. This explanation has several advantages over other explanations of the theory of mind: It can easily explain some behaviors in children at much earlier ages than other theories, and it is a much more economical explanation. In addition, it has a high biological evidence, with mirror neurons being a good candidate supporting the validity of this theory.

Mirror neurons are a class of neurons, discovered in rhesus monkeys, that fire both when the monkey performs a motor action, such as grasping an object, and when it observes another individual (monkey or human) performing the same or a similar action. However, they do not fire when the monkey only observes the object or the hand mimicking the grasping without a target object [21]. Neurophysiological and brain imaging experiments have shown strong evidence that a circuit analogous to the mirror neuron system from monkeys exists in humans [22]. Therefore, since mirror neurons fire both when we observe and when we perform an action, they are believed to be involved in our understanding of the states and actions of others, by mirroring the observed actions in our brains as if they were being performed by us [23]. This same mechanism seems to be also involved in other functions such as imitation, as it provides motor copies of others' actions [24]. There is also evidence that mirror neurons take part in intention understanding: Research has shown that observing an action in a context that allows us

to understand the intention of the action activates mirror neuron areas that observing the action without the context does not [25]. For this reason, several authors have suggested that mirror neurons are the basis for the theory of mind, supporting the simulation theory [26].

1.2.3 The Problem of Activity, Plan, and Goal Recognition

Problem Definition

Having had this overview about the mechanisms that are believed to describe action, plan, and intention recognition in humans, we are ready to define the corresponding problem in machines, as well as to introduce its main challenges. We begin defining the concepts of activity, plan, and goal recognition in the context of artificial intelligence [27], [28]:

- **Activity recognition** refers to the problem of analyzing and adequately labeling low-level data from humans or other autonomous agents performing some action [29], [30]. This task usually involves processing noisy low-level sensory input streams, looking for patterns of interest in this data, discretizing them into meaningful subsequences, and labeling each of these temporal subsequences. Sometimes it is also referred to as *behavior recognition*. The decrease in sensor and computation costs, together with the advancements in machine learning and big data and the spread of wearable devices, have boosted this field in recent years and brought it to the forefront of research in computer vision and ubiquitous computing. Typically, these algorithms work with data coming from video cameras, accelerometers, motion capture sensors, RFID sensors, smart badges, Wi-Fi or Bluetooth signals, or GPS sensors, among others. This data is used to recognize very different types of activities, from daily-life activities such as walking or sitting to more application-specific activities such as those performed by a factory worker or a football player. A typical task within the problem of activity recognition is **action recognition** (or *action understanding*). Action recognition deals with recognizing short spatiotemporally localized actions or events [31], i.e., action recognition tries to segment the most elementary or primitive component of the activity (e.g., picking a dish) while activity recognition may work with longer sequences (e.g., washing dishes).

- **Goal recognition** refers to the problem of inferring the intention of humans or other autonomous agents by observing a set of actions performed by those agents or their effect on the environment [32]. This task usually gets as input an ordered sequence of action labels (which may come from an action recognition module), and, possibly, a set of conceivable goals, and outputs the goal label that best explains the observed actions (or a set of labels with their corresponding probabilities). This task is often also referred to as *intention recognition* or *intent recognition*. However, several authors discourage the use of these terms due to the ambiguity of the words *intention* and *intent*, which are defined in

very different ways across fields and even within the same field (e.g., Xu, Luo, Zhu, et al. [33] included the whole plan as part of what they called *intention*). On the other hand, the term *intent recognition* (or *intent classification*) is often used to refer to the particular task of understanding the intention of a person from a sentence in natural language, making it not the most appropriate to refer to the more general task of goal recognition. In addition to the apparent differences, goal recognition also differs from activity recognition in the predictive component: Goal recognition tries to infer the final objective of the agent, which will imply a set of future actions, while activity recognition focuses on the action occurring at that instant [34].

- **Plan recognition** is the problem of understanding the goal of humans or other autonomous agents, as well as the set of actions that have been or will be performed by those agents to reach that goal, given a set of observed actions performed by those agents [35]. This task has several things in common with the goal recognition task, but it is more general: It includes the goal recognition task and complements it with the task of defining a structure with the set of observed and predicted actions that will lead to that goal, as well as their relationships. While it provides a more complete solution to the problem than goal recognition, goal recognition may be more appropriate for applications where we need a fast detection of the goal and the detailed plan is not relevant.

It should be pointed out that, however, in many cases, recognition problems do not clearly belong to one of the classes described above, but they may have components of several of them. For example, in hierarchical architectures, the task of a layer may be seen as goal recognition with respect to the actions coming from the lower layer, but it may be considered action recognition from the point of view of a higher layer that takes its outputs as elementary actions. Fig. 1.1 illustrates this idea of hierarchy. In this figure, task "Go to bakery" can be seen as an action that can be taken to achieve goal "Get bread", but it can also be seen as a goal that can be achieved through a sequence of actions ("Exit house", etc.). Similarly, "Exit house" can also be seen as a goal for lower-level actions (e.g., "Open door"), and "Get bread" as an action to perform higher-level goals (e.g., "Have lunch"). Fig. 1.1 also shows how different plans formed of different sets of actions may lead to a same goal.

Taking all this into account, we can define the general problem of activity, plan, and goal recognition as the problem of, given a set of observations from the environment and/or the observed agent(s) (e.g., sensory streams, action labels, etc.), and, possibly, given a set of actions that can be performed (e.g., moving to get a better angle of view), finding the activity, plan, and/or goal(s) that best explain those observations. The problem is composed of three elements: the environment, the observed agent(s) (the actor(s)), and the observing agent (the observer) [28].

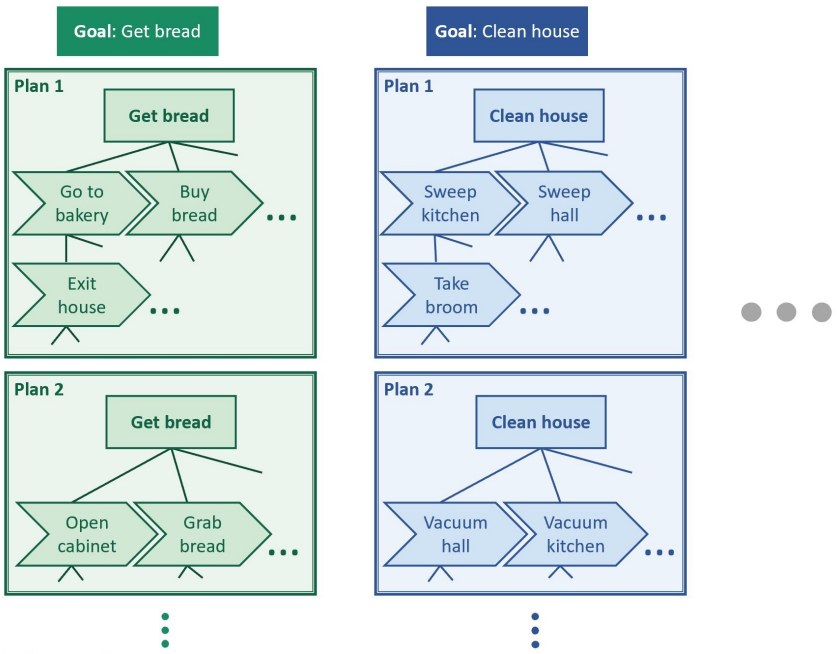


Figure 1.1: Illustration of the concepts of action, plan, and goal. A same goal may be achieved through different plans. These plans are often hierarchical structures whose elements can be seen as actions from the point of view of the higher-level elements and as goals from the point of view of the lower-level elements.

Regarding the input to the recognition systems, in addition to action labels or sensory streams, these systems can get other types of input that, while less straightforward, may be decisive in the recognition task if used appropriately. For instance, information about the environment where the action occurs (e.g., kitchen, airport, etc.) or about who is performing the action (e.g., a kid, a fireman, etc.) can help recognition systems better evaluate what action, plan, or goal explains best the observations.

System Classification

So far, we have seen a possible way of classifying these recognition systems attending to their objective (i.e., activity, plan, or goal recognition). In fact, many other criteria can be adopted to classify these systems. For example, they are often classified in terms of the type of approach they take to address the problem. According to this, we can divide them into **logic-based** approaches, **classical machine learning** approaches, **deep learning** approaches, and **brain-inspired** approaches. We discuss these approaches in greater detail later in Section 1.2.4. Another characteristic that defines the type of recognition problem is the behavior of the observed agent towards the observer. In these terms, we can define three types of systems: **agnostic**, **adversarial**, or **intended** [35]. In agnostic systems, the actor performs independently of the observer (the actor may even be unaware of being observed) [36]. In adversarial systems, the

actor tries to deceive the observer, either by occluding the actions or by performing actions with the purpose of generating confusion [37]. In intended systems, the actor tries to help the observer by, e.g., giving cues about the action being performed [38]. Similarly, recognition systems can be classified attending to whether the observing agent takes action and influences the actor or the environment with the purpose of making the recognition simpler. In this sense, recognition systems can be classified as **no intervention**, **offline intervention**, **online intervention**, or **direct communication** systems [28]. In no intervention systems, the observing agent does not act in any way over the actor or the environment to make the recognition task easier [37]. When these systems are also agnostic, they are known as **keyhole** systems. In offline intervention systems, the observer may introduce changes in the environment before the recognition process starts with the purpose of making the recognition easier [39]. In online intervention systems, the observer takes action over the actor or the environment during the recognition process with the purpose of revealing some new information or causing a reaction over the actor that helps in the recognition task [40]. A common way to do this is through what is known as active perception [41], which consists of taking action to increase or improve the input information (e.g., a robot moving to better see the ongoing action). In direct communication systems, the observer directly asks questions to the actor about the ongoing plans or goals and reasons according to the answers [42].

Another way to classify these recognition systems is attending to the characteristics of the environment. The environment can be **fully observable** (when the observer perfectly knows its state) or **partially observable** (when the observer only gets a possibly noisy fraction of the information about the state of the environment) [43]. It can be **deterministic** (if given a state and the action(s) performed over that state the next state is determined) or **stochastic** (if given a state and the action(s) performed over that state the environment may evolve to different states with different probabilities) [44]. And it can be **discrete** or **continuous** [45]. The systems can also be classified depending on whether the recognition process is done **offline** (the recognition is done at the end, with all the observations available) or **online** (observations are received incrementally, and the recognition is attempted to be done as soon as possible) [46]. A common characteristic of offline systems is that they often work with segmented sequences, with each complete sequence belonging to a single activity or plan. Online systems, on the other hand, generally require to be also able to segment, indicating what time sample belongs to what class (this is sometimes referred to as *continuous classification*). Another feature of these recognition systems is whether the set of all **possible activities/plans/goals** is **known** or **unknown** by the observer [47]. If it is unknown, the observer needs to be able to handle new classes of observations appropriately, e.g., by recognizing them as unknown activities and learning them, so that it can recognize them in the future. Finally, while most work in this area is directed towards recognizing the activity of a **single agent**, much research has also been done in **multiagent** systems [48]. This second

type of recognition is typically used in cooperative applications, where the agent belongs to a team and needs to understand the role of the other members to perform best [49]. It is also common when trying to understand the strategy used by the opponent in the military or sport domains [50]. Fig. 1.2 shows some example scenarios classified according to the criteria just presented.

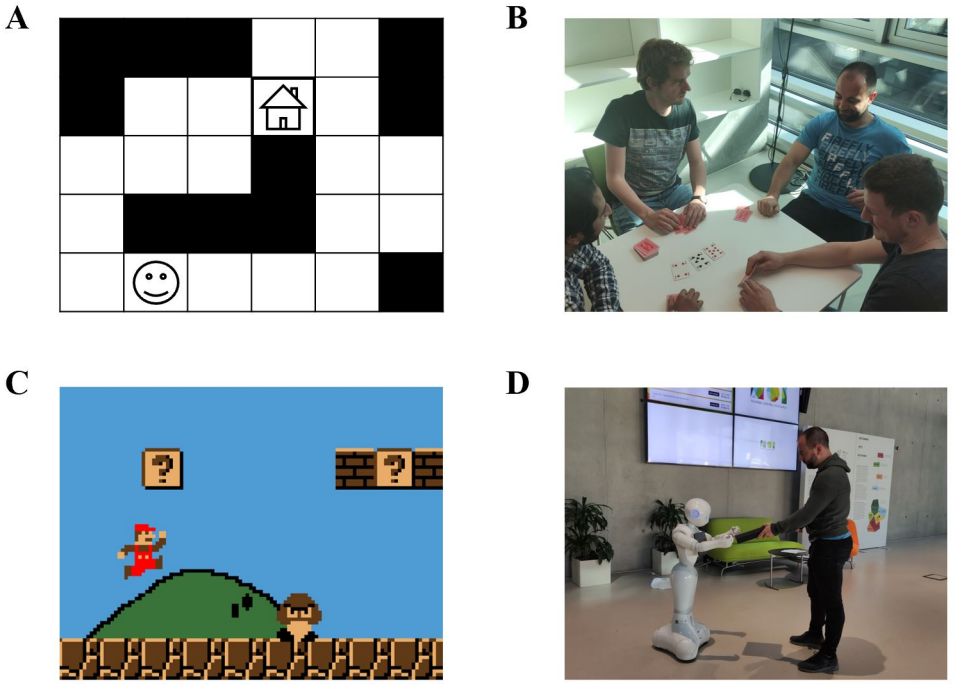


Figure 1.2: Examples of scenarios with different characteristics, (**A**) is a grid navigation scenario (agnostic, no intervention, fully observable, deterministic, discrete, single agent), (**B**) is a poker game (adversarial, direct communication, partially observable, stochastic, discrete, multiagent), (**C**) is a platform video game (agnostic, no intervention, partially observable, deterministic, continuous, single agent), (**D**) is a human-robot collaboration scenario (intended, online intervention, partially observable, stochastic, continuous, single agent).

A different way of classifying these systems is according to the **applications** for which they were designed. Indeed, while many of the algorithms developed for activity, plan, and goal recognition can be considered general-purpose, different applications often involve recognition problems with very different characteristics and types of input data (e.g., video, natural language, computer commands, etc.), making algorithms more or less appropriate depending on the application. In addition, those "general-purpose" algorithms also require being adapted to the particular problem, leading to more application-specific systems. There is a wide variety of applications where activity, plan, and goal recognition algorithms have been applied and proven to be useful. Some examples are smart homes [51], personal agent assistants [52], human-robot interaction [34], video surveillance [31], video games [53], natural language understanding [54],

assistive care for the elderly [55], software help systems [56], computer network security [57], decision support systems [58], and orthotics [59], among others. For example, assistive systems for the elderly need to understand the intention of the assisted person in order to anticipate and be able to help. Computer network security systems, on the other hand, need to analyze network activity and detect suspicious actions or actions performed with a malicious intention and act accordingly.

Challenges

Having introduced some of the main characteristics that describe activity, plan, and goal recognition systems, we can now present a number of challenges that these systems may need to overcome in order to complete their task effectively. We have classified these challenges into three categories: situations that the system may face and need to be able deal with, sources of information that the system may need to integrate to perform best, and system characteristics that may be required. Below we provide a non-exhaustive list of challenges for each of these categories. Starting with the list of situations that the system may need to be able to address:

- Multiple candidate hypotheses and uncertainty [60]
- Variability among different instances of the same action/plan [30]
- Incomplete knowledge, partial observability of the actor and the environment, and missed actions, as well as noisy input [35], [60]
- Unknown transitions between consecutive actions/plans (i.e., segmentation) [61]
- Interleaved plans or plans executed in parallel, either to achieve different goals or the same goal in alternative ways [60]
- Interrupted plans [61]
- Actions belonging to more than one plan or contributing to more than one goal [62]
- Actions performed by different agents to reach a common goal [60]
- Non-rational actions, reactive (not goal-directed) actions, exploration actions, irrelevant actions, actions executed by error, or actions executed with the purpose of misleading the observer [35], [60]

Regarding what information may be relevant to deduce the ongoing activity, plan, and/or goal, indeed, there are multiple sources and factors that may be decisive to successfully perform the recognition in addition to the human body pose and movements or actions. Some examples are:

- The context/environment [63]
- The objects/agents the actor interacts with [64]
- The history of previously observed actions/plans/events [35]
- The effects of possible previous actions [63]
- The characteristics and preferences of the specific agent being observed or of the type of agent [65]
- The temporal ordering of events [62]

Finally, depending on the application, some desirable (or required) characteristics of the system may be:

- Having predictive capabilities, or completing the recognition task before the action/plan is completed [34]
- Expressivity and interpretability of the system output [61]
- Scalability to a greater number of possible activity/plan/goal classes [35]
- Adaptability to different environments and applications and to multi-agent scenarios [35]

Existing Literature Reviews

Before deepening further in the different types of approaches taken to tackle these problems, it is worth mentioning some of the literature reviews available on this topic. The *Introduction* chapter of the book *Plan, Activity, and Intent Recognition. Theory and Practice* [27], for example, provides a short review of the history of the topic. Since the book was published in 2014, it mainly focuses on logic-based and probabilistic approaches, and the more recent deep learning approaches are not considered. Vrigkas, Nikou, and Kakadiaris [30] address the topic of activity recognition focusing on applications that get as input still images or video sequences. Their work also goes through the low-level task and algorithms for extracting features of interest from images (which are out of the scope of this review). A more thorough overview on these kinds of activity recognition systems can be found on the book *Human Activity Recognition and Prediction* [66], which besides going through the different techniques, also introduces some general relevant concepts on activity recognition. Another review on the problem of activity recognition is that of Wang, Chen, Hao, et al. [67], in this case focused on sensors different from cameras (e.g., body-worn sensors, etc.) and on deep learning approaches. Other reviews on activity recognition are Jobanputra, Bavishi, and Doshi [29], Lara and Labrador [68], and Poppe

[31]. On the other hand, Carberry [35] describes in her review the problem of plan recognition, together with its main challenges and approaches to address it. While this study was published in 2001, it is a quite complete work, and many of the concepts and challenges described are still relevant nowadays. Armentano and Amandi [61] also provide a review on the topic of plan recognition, which, while focused on interface agents, can also be of interest to other application fields. Finally, Sadri [60] reviews the main logic-based techniques used to approach the problem of goal recognition.

These reviews, in general, either focus on the problem of plan and goal recognition, going through the main existing logic-based and probabilistic solutions, or focus on the problem of activity recognition, covering machine learning approaches (and, in recent years, specially deep learning approaches). This goes in line with what can be found in the research studies themselves, with a clear separation between those subfields. However, to the best of our knowledge, there are no reviews that cover the whole problem of activity, plan, and goal recognition together with the most common approaches in a comprehensive way. Indeed, none of the consulted reviews includes logic-based, classical machine learning, and deep learning solutions altogether. Furthermore, we have found no review covering the most common brain-inspired methods to approach this recognition problem, which is a relevant topic when it comes to human-robot interaction. In this literature review, we try to fill this gap, by describing the whole problem of activity, plan, and goal recognition, presenting the main challenges and characteristics with which it can be described, and going through the different types of approaches used to address it. This view of the problem as a whole may also inspire researchers to think of new approaches that address the complete problem in a unified way, instead of focusing on just a part of it.

1.2.4 Approaches to the Problem of Activity, Plan, and Goal Recognition

This section describes in more detail the different types of approaches that have been taken to address the problem of activity, plan, and goal recognition. As expressed in Section 1.2.3, we can classify these algorithms according to the type of approach into four different classes: logic-based, classical machine learning, deep learning, and brain-inspired.

Logic-Based Approaches

Approaches based on logical reasoning [69] have mainly focused on the problem of plan and goal recognition, even though there have also been attempts to tackle the activity recognition problem. In fact, the first attempts that were made to address the problem of activity, plan, and goal recognition took this type of approach [12], [70]. Following the predominant tendency in artificial intelligence at these early times, researchers defined a set of domain-dependent rules that tried to capture the relevant knowledge that allowed the system to infer conclusions through deduction [27]. As occurred in other areas, this approach soon showed to be very limited in

several ways, particularly in maintainability and scalability. Some years later, a representation of plans as tree graphs was proposed, where the plan was represented as the top-level node of the tree, and the actions in which it was decomposed as the child nodes [62]. This work gave some structure and coherence to the field, and the conceptual framework proposed is still relevant today [27]. However, these early techniques still suffered from their purely deductive inference method: If there were more than one possible plan or goal compatible with all the observations, the system was unable to decide which one was the most likely explanation.

Since then, mainly two kinds of **reasoning** have been employed to cope with this issue of hypothesis selection: abductive reasoning and hybrid logic-probabilistic reasoning. **Abductive reasoning**, or abduction, is a form of logical inference that tries to find the simplest or most likely conclusion that can explain some given observation(s) [71]. Therefore, unlike deduction, the conclusions reached through abduction are not positively verified, but they are rather plausible conclusions understood as best explanations. This type of reasoning allows the inference system to choose among several hypotheses to explain an observation. In fact, hybrid logic-probabilistic approaches can be considered a special case of this kind of abduction. Sohrabi, Riabov, and Udrea [72], for example, associated a cost to each of the possible plans, as well as to the noisy or missing observations. By summing these values for a given observation and candidate plan, they obtained the weight for that hypothesis. The plan with the lowest weight was the most likely one. Jarvis, Lunt, and Myers [73], on the other hand, developed a terrorist activity detection system based on the definition of two concepts, called "frequencies" and "accuracies", that were associated to each of the actions. An action with a lower "frequency" or higher "accuracy" associated would be considered more relevant when it was observed.

As we just mentioned, **hybrid logic-probabilistic reasoning** can in fact be considered a special case of abductive reasoning which explicitly deals with probabilities. However, we consider it separately from pure abductive reasoning because, even though it is conceptually similar, it combines logic inference methods with probabilistic inference methods. These approaches have the advantage that they keep the expressivity of logic solutions while being able to handle uncertainty in a probabilistic way. There are different ways of combining those two types of reasoning. Some examples of commonly used hybrid models are relational Markov models and Markov logic networks [74]. Relational Markov models (RMM) generalize Markov models by allowing states to be of different types and to be hierarchically structured, while Markov logic networks (MLN) combine ideas of Markov networks with first-order logic, enabling probabilistic inference. Pereira and Han [75] proposed an intention recognition system for elder care based on causal Bayesian networks. These networks established probabilistic relationships among causes, intentions, actions, and effects and were used to extract the most probable intentions. The plausibility of these intentions in the given situation was then checked through a logic plan generator. Raghavan, Singla, and Mooney [76] proposed a very different approach, which lied

on the framework of statistical relational learning (SRL). The formalisms within this framework typically describe relational properties using first-order logic, while they handle the uncertainty through probabilistic models. In the paper, they proposed an extension of Markov logic networks and Bayesian logic programs to adapt them to abductive reasoning and perform plan recognition.

Some authors combine these abductive or hybrid reasoning techniques with some kind of logic-based **causal theory**, such as situation calculus or event calculus [69]. These formalisms allow reasoning in dynamic domains through the introduction of temporal constraints, which enable the definition of temporal properties such as preconditions or effects. Quaresma and Lopes [77] combined abductive reasoning with event calculus and some concepts from the theory of mind (commented in Section 1.2.2) to address the problem of recognizing plans and intentions behind speech acts. Their model described the mental state of the observed agent in terms of intentions and beliefs and reasoned according to that mental state and to the effects the agent was believed to expect from the actions.

So far, we have focused on the type of reasoning to classify the logic-based approaches to the problem of activity, plan, and goal recognition. These approaches can also be classified in terms of **how knowledge is represented**. Attending to this, we can divide them into two types: plan-library based and domain-theory based [69], [78]. Approaches based on **plan libraries** are sometimes referred to as *plan recognition as parsing*, because plans are usually represented as a hierarchy of lower-level actions, and the problem is reduced to finding the best fit of the observed actions into those plans. There are several ways of representing the knowledge in plan libraries. A common one is using hierarchical task networks (HTN) [69]. Hierarchical task networks represent tasks as a set of subtasks and constraints on them or among them. These tasks can be iteratively expanded until primitive tasks are reached, which would correspond to observable actions. Myers [79] used hierarchical task networks to recognize the high-level goal of a user of a collaborative planning system by observing a partial plan (a partial set of actions). Once a complete plan to which that partial plan belonged was identified, the system could complete the remaining actions. Another usual way of representing the knowledge in plan libraries is using grammars [69]. Grammars define a set of symbols and production rules that describe how they can be combined, enabling the generation of hierarchical tree-like structures. They were originally used in the parsing of natural language, but they have also been shown to be useful in plan recognition (note the similarity between both fields, dealing with hierarchical and sequential data). When the grammar rules are described through probabilities, they are known as stochastic grammars. Geib and Goldman [80] proposed a probabilistic plan recognition system based on plan tree grammars that was able to handle interleaved plans, partially ordered plans, and partial observations.

Regarding **domain-theory** based approaches, they are often known as *plan recognition as*

planning. In these approaches, off-the-shelf planning systems are used to generate candidate plans for the observed agent. These planning systems generally rely on planning languages such as STRIPS or PDDL [69], which allow them to describe the state of the environment and the effects of the possible actions, as well as to reason over that knowledge in order to develop candidate plans that lead to the achievement of given goals. These candidate plans are then weighted by the plan recognition system according to the observations obtained, and the most likely plan and/or goal is selected according to those weights. When the planning system used relies on Bayesian inference to reason, these recognition approaches are sometimes referred to as Bayesian inverse planning systems. Ramirez and Geffner [81] were the first ones suggesting the plan recognition as planning approach. They proposed an approximate planning method that generated an acceptable set of plans to the possible goals and that was able to scale well. Pereira, Oren, and Meneguzzi [82] went through some of the main state-of-the-art domain-theory based techniques to goal recognition focusing on the concept of landmarks, which are states or actions through which the observed agent needs to go in order to achieve a certain goal. They proposed a landmark-based goal recognition approach that was able to perform faster than other systems, while having a comparable accuracy.

Summary As a summary, we can say that logic-based and hybrid logic-probabilistic approaches are probably the most commonly used ones in the literature to deal with the problem of plan and goal recognition. Logic systems present several advantages that are very useful for this problem. One of the main ones is that they are very good at working with highly structured representations. Logic representations can define different kinds of relationships among entities, such as preconditions, mutual exclusions, or decompositions, which are very useful for the problem at hand. Besides, these logic relationships allow these systems to generate new plausible candidate plans online that can be compared against the actual observations. A second advantage is that logic representations are highly expressive, allowing their designers and users to easily understand the meaning of their output, as well as how they have reached that conclusion. In addition, when combined with probabilistic reasoning techniques, logic-based systems can reduce some of their main weaknesses, such as not being able to handle uncertainty.

However, even when combined with probabilistic reasoning methods, they still present some disadvantages. For example, logic systems are very rigid: They define entities in terms of a set of properties and relations, and observations need to fit them perfectly in order to be considered such entity. However, in general, real-world "entities" are more ambiguous and require more flexible approaches. Even when combined with probabilistic methods, the logic-based component of the system usually keeps part of this rigidity. This is particularly relevant in more uncertain environments, such as those partially observable, where the knowledge of the observed agent is unknown, or where interleaved or interrupted plans can occur. Another disadvantage of these

systems is that they usually require the designer to introduce manually the domain knowledge upon which the system will reason. This limits the use of these approaches to applications where the necessary knowledge can be expressed in such way. Indeed, even if the system is able to generate new candidate plans online, they will always be based on that limited domain knowledge, and plans that fall out of the scope of this domain will not be recognizable. In addition, in domains that are complex enough, introducing all the knowledge manually requires much domain-expert effort and is prone to errors. This makes these systems bad at scaling and generalizing. Finally, these systems usually assume that the observed agent is rational and try to find the optimal plan that best fits the observations. However, humans often act in non-rational ways [83], making these systems not so appropriate.

Classical Machine Learning Approaches

In the previous section, we saw some examples of hybrid logic-probabilistic approaches to the problem of activity, plan, and goal recognition. In fact, probably most of the proposed classical machine learning solutions to this problem are based on probabilistic systems such as Bayesian networks or Markov decision processes. Therefore, in this section we will focus mainly on these probabilistic approaches, depicting at the end also some non-probabilistic techniques that have been used for this problem. As we just said, many of these approaches are based on different types of **Bayesian networks** (BN) [69], such as dynamic Bayesian networks or hidden Markov models. Bayesian networks are generative probabilistic graphical models that represent random variables as nodes and conditional dependencies as arrows between them. They can provide the probability distribution of any set of random variables given another set of observed variables. In the case of activity, plan, and goal recognition systems, those random variables can represent, e.g., low-level observations, actions at different hierarchical levels, or final goals. Due to the temporal nature of the problem, the Bayesian networks used typically take the form of **dynamic Bayesian networks** (DBN) [69]. Dynamic Bayesian networks are a kind of Bayesian network that represents its variables at different time steps, as well as the conditional dependencies across time steps. For example, Liao, Fox, and Kautz [84] built a dynamic Bayesian network with a hierarchical structure that could infer the transportation modes or destination goals of a user from low-level GPS sensor measurements, as well as recognize abnormal or unknown activity. They used particle filtering to infer across the network.

A particular well-known type of dynamic Bayesian network that has been commonly used for the problem of activity, plan, and goal recognition is **hidden Markov models** (HMM) [69]. Hidden Markov models assume that an observable variable exists which depends on a hidden variable (i.e., the state), which depends on itself at the previous timestep. This allows to, given a sequence of observations, find the most likely hidden state(s). The use of this type of Bayesian network is quite spread in problems dealing with sequential data, such as natural language processing,

and well-known techniques exist to infer on it. Kelley, Tavakkoli, King, et al. [34] used hidden Markov models to model actions and goals within a human-robot interaction scenario. They built one hidden Markov model for each possible goal, with the hidden states representing the possible actions. However, hidden Markov models are quite simple models, with little structure, while, as we argued earlier, activities, plans, and goals often have a relatively complex and hierarchical structure. Some extensions of hidden Markov models exist that were designed with hierarchical structure in mind and that have been successfully applied to the problem of activity, plan, and goal recognition, such as layered hidden Markov models (LHMM) [85] or hierarchical hidden Markov models (HHMM) [86]. Bui, Phung, and Venkatesh [86], for example, proposed an extension of the hierarchical hidden Markov model that allowed two states to share the same child (e.g., the same action belonging to two different plans) and applied it to the problem of action and plan recognition in an airport scenario.

Another popular probabilistic model used in sequential classification problems, and that has been used mainly for activity recognition, is linear chain **conditional random fields** (linear chain CRF) [69]. Linear chain conditional random fields are discriminative probabilistic graphs that are applied and can deal with similar problems as hidden Markov models. However, they are more powerful, since they can model everything that hidden Markov models can and more. Zhao, Wang, Sukthankar, et al. [87] proposed an activity recognition system that extracted a set of features from patterns found in inertial data to feed a linear chain conditional random field. Similarly to hidden Markov models, linear chain conditional random fields are also quite simple in terms of structure. Liao, Fox, and Kautz [88] used a two-level so-called hierarchical conditional random field to predict a person's activity (first level) and the place at which the activity was taking place (second level) based on GPS data.

A different probabilistic approach to those already mentioned is modeling the observed agent as a **Markov decision process** (MDP) [69]. Markov decision processes model an agent decision process in a system where the transition between states, as well as the rewards obtained by the agent, depend probabilistically on the actions taken. This type of approach is sometimes referred to as inverse reinforcement learning. Oh, Meneguzzi, and Sycara [52] proposed a proactive assistant agent for domains such as emergency response and military peacekeeping operations that modeled the observed agent as a Markov decision process, allowing the system to infer the agent's goals, predict the following actions, and provide assistance accordingly. Markov decision processes assume that the state of the system is known by the agent. However, this is not the case in general. Partially observable Markov decision processes (POMDP) [69] deal with this by maintaining a probability distribution over the set of possible states. Baker and Tenenbaum [89] proposed a probabilistic theory of mind model where the mental state of the observed agent (beliefs and desires) where modeled as probabilistic distributions within a partially observable Markov decision process. Using Bayesian inference, the system could

estimate the belief state and reward function of the agent.

As we said earlier, other **non-probabilistic** classical machine learning techniques have also been used, mainly for the problem of activity recognition, such as support vector machines (SVM), decision trees, k-nearest neighbors (KNN), or shallow artificial neural networks (shallow ANN) [68], [69]. Support vector machines are binary classifiers that divide the feature space through a hyperplane, with each of the resulting subspaces corresponding to each category. While they are intrinsically linear, they can be extended to perform non-linear classification, as well as to work with more than two classes. Samanta and Chanda [90] used support vector machines to classify human activities represented through space-time features extracted from video data and tested their approach on several standard datasets. Decision trees are classification algorithms that use a tree-like model of decision. Each node is characterized by some criterion according to which the input samples are sent to one of its subnodes until the samples reach an end node or leaf, which has a class assigned. Fan, Wang, and Wang [91] used decision trees to classify daily-life activity data coming from the accelerometers of a smartphone. The system was able to classify the data successfully independently of the actual location of the smartphone.

Finally, few **unsupervised** learning systems have also been proposed to address the problem of activity recognition. These systems, however, while unsupervised, are quite limited in performance and in the types of applications for which they can be useful. For example, Vahdatpour, Amini, and Sarrafzadeh [92] proposed an unsupervised system for motif (recurring pattern) learning and detection for activity recognition using clustering techniques. Polyvyanyy, Su, Lipovetzky, et al. [93], on the other hand, proposed an unsupervised probabilistic goal recognition system based on process mining techniques.

Summary In summary, classical machine learning approaches (and, in particular, probabilistic approaches) to the problem of activity, plan, and goal recognition have shown as their main advantage being good at handling uncertainty. This makes them useful to deal with situations that are common in real environments, such as handling interrupted or interleaved plans, coping with partial observability or noisy data, or even dealing with non-rational agents or dynamic domains. These advantages are especially noticeable in the task of activity recognition, in which logic-based approaches are often not appropriate. In addition, classical machine learning approaches do not require a full manual introduction of the domain knowledge, as they can learn the parameters (e.g., probabilities) given enough training data. On the other hand, well-studied hierarchical networks exist that provide structure to the models, minimizing one of the limitations of these approaches.

However, classical machine learning approaches have also several disadvantages. First, these methods are less expressive than logic-based ones, and it is often hard to understand how

one of these models has reached a certain conclusion. In addition, while it is true that some classical machine learning systems scale well (see next section on deep learning approaches), probabilistic systems in general do not: As these systems become more complex, more variables and dependencies among them need to be modeled, estimated, stored, and properly used. Besides the fast growth in the number of dependencies, the structure of the network also needs to be designed. While this structure can also be learned from data, taking this approach requires a higher amount of data and leads to difficult-to-understand networks and relationships between variables. On the other hand, even though well-known hierarchical structures exist, they are still very limited on the types of relationships among variables they can model, and their structure is also very rigid. Regarding non-probabilistic systems, not dealing explicitly with probabilities also comes with some limitations, as they do not provide information on the certainty of the obtained output. This can be an issue in applications where having a "best guess" is not enough, but information on the confidence of that guess is also required. Online recognition systems may also need such information to understand if the ongoing action/plan/goal has already been recognized. Finally, classical machine learning approaches, similarly to logic-based ones, are generally not designed to learn online new activities, plans, or goals, something that may be necessary in real environments.

Deep Learning Approaches

The quick introduction and success in recent years of deep learning approaches into fields where more classical machine learning techniques were commonly used has also affected the field of activity, plan, and goal recognition. Indeed, deep learning [94] is becoming one of the main technologies to deal with the problem of activity recognition [67], and it has also been used for plan and goal recognition. Deep learning approaches generally make use of **deep neural networks** (DNN) [94]. Deep neural networks consist of a set of (more than three) layers composed of multiple neurons. The input data goes from the input layer through all the hidden layers until it reaches the output, being processed in each layer according to a function that has been learned. One of the simplest deep neural network architectures in terms of design, which has been used for activity recognition, is the (deep) **multilayer perceptron** (MLP) or fully-connected feedforward neural network, where there are no cyclic connections between neurons and all neurons of a layer are connected to all neurons of the previous layer. Hammerla, Halloran, and Plötz [95] designed a five-hidden-layer network to recognize activities from data coming from wearable sensors and compared this architecture with other popular deep architectures such as convolutional neural networks and long short-term memory networks. These last architectures, in general, outperformed the first one and were able to converge to a system with an acceptable performance much faster.

Convolutional neural networks (CNN) [94] are one of the most commonly used deep ap-

proaches to the problem of activity recognition. These networks are very popular when processing images or temporal data because they extract local patterns from elements close in the image or in time. When applied to temporal data, the input data is usually divided into time windows, and these windows are processed independently by the network. In the case of activity recognition, these networks have been shown to be very effective when working with prolonged and repetitive activities such as walking or running [95]. Bevilacqua, MacDonald, Rangarej, et al. [96] used and compared different configurations of inertial sensor data to train different convolutional neural network architectures to classify physical activities. Ronao and Cho [97] tried convolutional neural networks of different depths and kernel sizes to classify activities coming from smartphone sensors. A variant of convolutional neural networks that is often used when explicitly working with body pose data (e.g., in the form of a human skeleton model coming from a motion capture system) is that of graph convolutional networks (GCN) [98]. Graph convolutional networks generalize the convolution operation of images and time series, where elements have a fixed number of neighbors over which to apply the convolution, to graphs, where nodes can have arbitrary and different numbers of neighbors. This allows them to efficiently take advantage of the graph structure and aggregate neighbor node information. Yan, Xiong, and Lin [99], for example, addressed the skeleton-based action recognition problem by applying a graph convolutional network over spatiotemporal graphs defined for each sequence and that connected contiguous joints in the same frame and same joints in consecutive frames.

Another deep architecture that has been used for the problems of activity, as well as plan and goal recognition, is recurrent neural networks (RNN) [94]. In these architectures, cyclic connections between neurons exist, allowing the network to have memory and to keep track of the context of the data. This makes them very useful to process sequential data. Among the different recurrent architectures, the most widespread ones are **long short-term memory networks** (LSTM) and **gated recurrent units** (GRU) [94]. These networks are designed so that they can decide when to update their memory depending on the context, being in this way able to learn long-range relationships in the input data. This is an advantage over convolutional neural networks, which can only count on the information in the temporal window for the classification. In the case of activity recognition, recurrent neural networks have demonstrated to be very useful at classifying activities that are short in duration but have a natural ordering, thanks to their ability to take the context into account [95]. Amado, Aires, Pereira, et al. [100] proposed the use of long short-term memory networks for a goal recognition task dealing with sensory input data, requiring much less manual introduction of domain knowledge than other state-of-the-art goal recognition approaches. Ordóñez and Roggen [101] combined convolutional neural networks with long short-term memory networks for the task of activity recognition. They used the convolutional networks for low-level feature extraction at the first layers of the network, followed by long short-term memory networks that could capture the temporal dynamics and context of the

observations.

A different approach to the processing of sequential data that has become very popular in recent years is using **transformers** [102]. Transformers are neural networks that, instead of counting on recurrent connections to access information from the past, rely on a self-attention mechanism. Self-attention roughly consists of using the information of each sample to decide what other samples of the complete sequence are relevant and need to be paid attention to in order to understand and properly process that sample. This attention-based mechanism allows transformers to model longer-range dependencies, being able to directly pay attention to any sample within the defined context length. As a consequence, transformers have been able to outperform convolutional and recurrent neural networks in many different sequential tasks. In addition, while they require larger memories and costlier trainings to learn those dependencies, they admit much more efficient parallelizations than recurrent neural networks during both training and inference. Plizzari, Cannici, and Matteucci [103] proposed applying transformers to model dependencies both between body parts within a frame and between frames in an skeleton-based action recognition task. The system modeled those spatial and temporal dependencies in two different streams that combined spatial and temporal self-attention modules with temporal and graph convolutional networks, respectively. Xin, Liu, Liu, et al. [104] combined recurrent neural networks and transformers for a video action recognition task, using transformers to model spatial dependencies between regions in the current frame and recurrent hidden state, considerably reducing in this way the memory footprint.

In addition to fully supervised systems, there are deep architectures, such as **autoencoders** or **deep belief networks** (DBN) [94], that can learn useful feature extraction functions in an unsupervised way, obtaining in this way higher-level alternative representations of the input data and then requiring less labeled data to achieve acceptable performances. Zhang, Wu, and Luo [105] implemented a deep belief network for activity recognition that could run and be trained in a smartphone, showing that these networks can be computationally very efficient. Min, Ha, Rowe, et al. [106] used stacked denoising autoencoders to model the goals of players in an open-ended digital game, achieving considerably better accuracies than other state-of-the-art models. In recent years, an approach that has gained popularity when dealing with representation learning tasks is using self-supervised **contrastive learning** techniques, where the learned sample representations are encouraged to be more similar to representations of samples belonging to the same class (e.g., augmented/modified versions of the same sample) and less similar to representations belonging to other classes. Singh, Chakraborty, Varshney, et al. [107], for example, addressed a semi-supervised action recognition problem using contrastive techniques that relied on change-of-speed augmentations and on pseudo-labels obtained thanks to the limited labeled data. A closely related problem is that of **few-shot learning**, where systems need to learn to recognize new classes from very few labeled samples (while not

forgetting previously learned classes). One common way to approach this problem is by training a deep network to extract high-level sample representations (e.g., through contrastive learning) to then obtain prototype vectors in the representation space for each class that are used in a nearest neighbor-like classifier. Perrett, Masullo, Burghardt, et al. [108] applied this approach to action recognition in videos by comparing the input videos against prototype vectors constructed specifically for each input video. These input-specific prototype vectors where obtained using transformers to find best correspondences between frames in subsequences of the input videos and of the original support set prototype vectors.

Zero shot learning, on the other hand, allows systems to label, or associate semantic meaning, to classes that were not present in the training data. Note that these systems do not actually learn during this recognition process, they just describe those unknown classes in terms of what they already know. Zero-shot learning systems usually rely on two feature extraction systems, one for the data to be labeled (e.g., images) and one for the semantic meanings (e.g., image captions), that are trained to learn similar representations for corresponding samples and meanings and dissimilar representations for non-corresponding ones, being able in this way to associate new meanings to new samples (e.g., see [109]). Wang, Xing, Mei, et al. [4] applied this procedure to zero-shot action classification making use of pre-trained text and video encoders to avoid the associated large training computation costs. Finally, it is worth mentioning that, with the advent of **multimodal large language models** (multimodal LLM) [110], new ways to achieve zero-shot and few-shot action or goal recognition are arising. Multimodal large language models are systems that, besides being able to understand and generate general-purpose language, can also deal with other data types such as images or videos. Indeed, these systems are able to provide a general description or to answer specific questions about what is going on in a given image or video without further training, with some of them reaching more than acceptable performances. However, these good performances come at the cost of being extremely large models, requiring massive amounts of data and resources in terms of computation and memory, especially for training, and making them prohibitive for many applications. This also makes them not appropriate for applications where some form of incremental or continual learning is required (e.g., dealing with concept drift).

Summary To sum up, deep learning models have several strengths that have made them the de facto algorithms in different fields and that also apply to activity, plan, and goal recognition. One of the main advantages of these approaches is that they are able to learn very different patterns, and their hierarchical architecture in layers allows them to extract features from the input data at different levels of abstraction, which are then used by the next layers. This contributes to making the training faster and the data requirements smaller (although still high), as higher-level layers often require similar lower-level features. Thus, they are very flexible and hierarchical

by nature, something very useful for the problem of activity, plan, and goal recognition. In addition, the input data to the network is often the sensor data itself, and the first layers of the network learn automatically the best feature extraction functions from the sensor data for the task at hand during the training process. This is an important advantage for activity recognition, and also for plan and goal recognition when they deal with sensor data. Most logic-based and classical machine learning approaches use manually designed features from the sensor data as input to the algorithms. These features require human effort and domain knowledge, are not generalizable, and may not be optimal. On the other hand, similarly to classical machine learning models, deep learning approaches are good at dealing with uncertainty and with partial and noisy information, as long as those characteristics also exist in the training data. Finally, recent advances in deep learning are showing how, as larger models are trained with larger amounts of (possibly multimodal) data, these models not only achieve unmatched performances in the specific tasks for which they were trained, but often also learn to solve other problems for which they were not specifically trained, becoming multi-purpose systems that can address action recognition and other problems.

One of the main disadvantages of deep learning approaches is that they usually require large amounts of labeled training data to perform successfully. This is related to the fact that, to achieve their high flexibility, they have many parameters that need to be learned. This is a relevant limitation in the problem of activity, plan, and goal recognition, as different available datasets are built using different types of sensors, in different configurations, and labeled according to different activities or goals, and, therefore, it is hard to combine them and exploit them together. Another limitation of deep architectures is that they usually require significant computational resources, especially on the training phase. These high computational (and data) requirements are usually correlated with the performance of the system, with systems achieving best performances often requiring resources that are prohibitive for many applications. Besides, deep networks are difficult to interpret and are often seen as black boxes, with the reasons that lead them to a particular conclusion being unknown to their designers or users. This can be a limitation especially for plan recognition, where the description of plans usually requires high expressivity. In addition, most deep architectures are non-probabilistic, which brings a set of difficulties, as we discussed in the section on classical machine learning approaches (e.g., not providing information on the certainty of the outputs). Finally, the most common architectures are not designed to learn online new classes, and, even when they can, they generally require labeled data. This is a limitation in real open environments. Zero-shot learning systems can alleviate this issue by associating meaning to new actions, but they cannot in general adapt and learn new useful action features online, and neither can they learn to predict next actions, something often required within the problem of activity, plan, and goal recognition.

Brain-inspired Approaches

The approaches seen so far use algorithms and techniques that are, in general, well established within the artificial intelligence and machine learning communities. In fact, most of the work in the area of activity, plan, and goal recognition has been done using these types of approaches. However, as we have just seen, these standard approaches present some relevant limitations for the problem at hand, especially when dealing with real environments, where being able to handle ambiguity, partial information, and unknown actions or goals is very important. Therefore, several authors have tried alternative approaches to address this problem, inspired by the fact that the human brain is able to deal with these issues in a very effective way. These approaches have especially been used in human-robot interaction applications.

As we explained in Section 1.2.2, mirror neurons are believed to play a key role in our ability to understand the actions and intentions of others. As a result, several **models of mirror neurons** have been used for the tasks of action and goal recognition, as well as for imitation. For example, mirror neurons have been often modeled as auto-associative memories. These memories learn patterns of actions when executing movements, associating the actions to the corresponding observations during execution. Then, when the execution of a similar action is observed, the action is recognized [111]. A more elaborated model of mirror neurons is MNS2 (mirror neuron system II) [112], which models several regions of the brain as recurrent neural networks and is able to learn different actions through self-observation to later recognize them on others. A quite different model of mirror neurons is the Mental State Inference model [113]. This model is not based on the association of self-performed actions with their corresponding observations, but, instead, given some observations, it simulates the actions that would correspond to the possible intentions and compares them against the actual observations, choosing the intention with the best prediction as the most likely one.

There are also several models that, while they do not try to mimic directly the functioning of the mirror neurons, show several similarities and have been shown to be useful on the tasks of action and goal recognition, as well as imitation. One of these models is **MOSAIC** (modular selection and identification for control) [114]. MOSAIC is a learning control system that consists of several predictor-controller modules. The controller part of each module generates motor commands that are used by the predictor part to simulate the movements that those commands would imply, and those simulated movements are then compared against the observed ones, with modules with better predictions becoming more influential. Another popular system with a similar architecture of inverse-forward blocks is **HAMMER** (hierarchical, attentive, multiple models for execution and recognition) [115]. The modules in this system can be combined to form more complex modules, achieving a hierarchy of modules that can operate at a higher level and can represent more abstract or full behaviors. A limitation of these architectures is

the conflict between generalization and segmentation that arises when motor primitives overlap, where generalization leads to representing many similar primitives with the same module, while segmentation requires that different primitives are represented in different modules. **Multiple timescales recurrent neural networks** (MTRNN) were designed as an attempt to overcome this issue [5]. Instead of using separate modules, they work with self-organization mechanisms and neurons working at different timescales, which leads to the emergence of a functional hierarchy among actions. Some of these action recognition systems implement some form of explicit visuo-spatial perspective-taking (i.e., they project the point of view of the observed agent onto their own). Alkurdi, Busch, and Peer [116], for example, projected the frame of reference of the observed agent into that of the observer, to then feed an action recognition system that was built upon a cognitive framework known as dynamic field theory [117]. Their system first detected the object of interest over which the action would be performed, to then compare the observed trajectory against those corresponding to possible actions over that object.

Many other brain-inspired algorithms and **cognitive architectures** exist, such as ACT-R (adaptive control of thought-rational) [118], Soar [119], ART (adaptive resonance theory) [120], or HTM (hierarchical temporal memory) [121], that can be used to approach the problem of activity, plan, and goal recognition. These architectures try to mimic in different ways and integrate the existing known mechanisms in our brain (e.g., different types of memory, learning, attention, etc.) to achieve cognition. For example, HTM models neocortex layers as a set of nodes that learn following a Hebbian-like rule and that are activated according to a sparse coding paradigm. Hebbian learning is a simplified model of how neurons in the brain learn that is able to learn and extract patterns in an online and unsupervised way. Sparse coding is a representation paradigm that is believed to occur in the brain in which just a small set of all the elements are active for each represented concept. These elements may represent the presence of certain features in the represented concept. This form of representation has been shown to be very robust and useful at representing new concepts in terms of previously learned features. As an example of a recognition system, Oltramari and Lebiere [122] built an ACT-R-based system for the recognition of actions in a video surveillance application. However, most of these brain-inspired frameworks and architectures have been little explored for the problem at hand, and it is therefore difficult to predict how effective they and their mechanisms would be for this problem. An overview and comparison of some of the best-known existing cognitive architectures can be found in [123].

Summary We conclude that, due to the great variety and diversity of brain-inspired algorithms and architectures, it is difficult to outline general advantages or disadvantages of these methods over those described in the previous sections. However, we can comment on the mechanisms that are known to exist in the brain and that are often integrated in these frameworks. One of these mechanisms is Hebbian-like learning, which, as we said previously, is a form of

unsupervised learning method that is able to learn patterns online. This method, if applied successfully, can be very useful for real open environments and, in general, to relax the human effort requirements for training. Related to this, sparse coding can also help on the learning of new concepts online and in few shots. This adaptability also endows these systems with tools to handle uncertainty and dynamic environments. In addition, these algorithms often work in a predictive way, which is a usual requirement for plan and goal recognition systems. Some brain-inspired algorithms also implement self-organization mechanisms, which, besides working in an unsupervised manner, can improve the performance and interpretability of the system. Similarly, attention mechanisms are often included in these frameworks, which can also improve the interpretability and performance of the system, besides reducing the computational requirements. Finally, mimicking particular structures of the brain, such as the hierarchical organization of certain regions or the mirror systems for action learning and imitation, has also been shown useful in different tasks.

One limitation of these types of approaches is that we are still far from a deep understanding of the functioning of the brain, and this makes the task of actually replicating its functionality virtually impossible. Even though we already understand many mechanisms in the brain that have shown good results when implemented in artificial systems, there are still many tasks where those systems are far from the high performance reached by the human brain. Another disadvantage when trying to mimic the brain is the fact that the human brain has about 86 billion neurons and 100 trillion connections between them, and current computers are still far from such high computational power. Therefore, hardware limitations may be prohibitive when trying to mimic the human brain in performing certain tasks. Still, note that models may be found that emulate the behavior of certain brain regions with an acceptable performance while having a considerably lower internal complexity. Finally, similarly to classical machine learning and deep learning approaches, brain-inspired architectures are often less expressive than logic-based ones, being hard to interpret how they have reached a particular conclusion. However, this depends on the particular framework used, and there are mechanisms that can alleviate this issue.

1.2.5 Discussion

The main aim of this literature review was to provide a general view on the problem of activity, plan, and goal recognition. This is a relevant problem in applications with humans in the loop, where being able to understand the humans' actions and goals is necessary to predict their behavior (as well as to have natural human-like interactions with them if necessary). Several reviews exist that describe either the problem of activity recognition or the problem of plan and goal recognition, together with their corresponding most common approaches. However, to the best of our knowledge, there are no reviews covering the whole problem of activity, plan, and goal recognition in a comprehensive way. This review fills this gap from the literature. To do

so, we have provided a brief introduction to the mechanisms that are believed to take part in the process of action and intention recognition in humans, followed by a definition, possible taxonomy, and main challenges of the problem and a description of the main approaches that have been proposed to address it.

As we have seen in Section 1.2.3, there are many possible criteria to classify these systems, such as according to their objective (activity, plan, and/or goal recognition) or according to the type of approach followed to address the problem (logic-based, classical machine learning, deep learning, or brain-inspired). Other possible criteria attend to the characteristics of the environment and the agents. For example, real environments (e.g., in human-robot interaction applications) are generally characterized as being partially observable, stochastic, and continuous. In the case they are also open environments, the recognition system will need to deal with unknown actions, plans, and goals. Table 1.1 shows a classification of the original studies that have been presented along this literature review according to the criteria described in Section 1.2.3. These criteria may be used in future research studies to classify the presented approaches, allowing the community to better understand the scope of applicability and enabling a more straightforward comparison of approaches. In addition, they may inspire the development of more standardized performance evaluation and comparison benchmarks, which could consist, for instance, of a set of standard problems/datasets of different nature. These standard problems should allow us to better understand the scope of applicability of each approach, as well as their performance in the different types of applications.

Table 1.1, while not meant to be exhaustive, can also give an idea about what kinds of systems have been extensively studied and which ones are still not so well explored. For example, the table shows that most existing work considers agnostic actors, non-intervening observers, known possible activities/plans/goals, and single agent systems. However, applications in real environments, for instance, may require a further exploration of scenarios with both adversarial and intended actors, with online intervention and/or direct communication, with unknown possible actions/plans/goals, and with multiple agents.

On the other hand, very few of the surveyed systems do action and plan recognition at the same time: Most logic-based approaches focus on plan or goal recognition, while the rest of the approaches are mainly used for activity or goal recognition. This is coherent with the characteristics of those approaches, which are summarized in Table 1.2. This table compares the four proposed types of approaches attending to some of their properties and to their ability to deal with different challenges. Hybrid logic-probabilistic approaches are probably the most commonly used ones for plan and goal recognition due to their high expressivity and the possibility to define different kinds of relationships among entities, while at the same time being able to handle uncertainty. They have the limitation that they usually require a manual

Table 1.1: Classification of the original studies presented along the literature review according to their approach (logic-based, classical machine learning, deep learning, or brain-inspired), objective (activity/action, plan, or goal recognition), observed agent involvement (agnostic, adversarial, or intended), observer intervention (no intervention, offline intervention, online intervention, or direct communication), environment (fully or partially observable, deterministic or stochastic, and discrete or continuous), recognition time (offline or online), knowledge of possible classes, i.e., activities/plans/goals (known or unknown), number of agents (single or multiple), and application.

Reference	Objective	Agent	Intervention	Environment			Recognition	Classes	#Agents	Application
Logic-based approaches										
Shrager and Finin [36]	Goal	Agnostic	No	Fully obs.	Det.	Disc.	Online	Known	Single	Software help
Avrahami-Zilberbrand and Kaminka [37]	Plan	Adversarial	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Surveillance
Perrault and Allen [38]	Goal	Intended	No	Fully obs.	Det.	Disc.	Offline	Known	Single	Language und.
Keren, Gal, and Karpas [39]	Goal	Agnostic	Offline	Fully obs.	Det.	Disc.	Online	Known	Single	Destin. guess
Shvo and McIlraith [40]	Goal	Agnostic	Online	Part. obs.	Stoch.	Cont.	Online	Known	Single	General
Mirsky, Stern, Gal, et al. [42]	Plan	Intended	Direct	Fully obs.	Det.	Disc.	Online	Known	Single	Software help
Keren, Gal, and Karpas [43]	Goal	Agnostic	Offline	Part. obs.	Det.	Disc.	Online	Known	Single	Destin. guess
Wayllace, Hou, Yeoh, et al. [44]	Goal	Agnostic	Offline	Fully obs.	Stoch.	Disc.	Online	Known	Single	Destin. guess
Kaminka, Vered, and Agmon [45]	Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	General
Vered, Kaminka, and Biham [46]	Goal	Agnostic	No	Fully obs.	Det.	Cont.	Online	Known	Single	Destin. guess
Zhuo [47]	Plan	Agnostic	No	Part. obs.	Det.	Disc.	Offline	Known	Multiple	Theoretic
Genter, Agmon, and Stone [49]	Goal	Agnostic	No	Fully obs.	Det.	Disc.	Online	Known	Multiple	Video games
Skocir, Krivic, Tomeljak, et al. [51]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Smart homes
Ha, Rowe, Mott, et al. [53]	Goal	Agnostic	No	Fully obs.	Det.	Cont.	Online	Known	Single	Video games
Bouchard, Giroux, and Bouzouane [55]	Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Elderly care
[58]	Plan	Intended	Direct	Fully obs.	Det.	Disc.	Online	Unknown	Single	Decision support
Kautz and Allen [62]	Plan	Agnostic	No	Fully obs.	Det.	Disc.	Offline	Known	Single	Theoretic
Zhuo [65]	Plan	Agnostic	No	Fully obs.	Det.	Cont.	Offline	Known	Single	Experimental
Schank and Abelson [70]	Plan	Intended	No	Fully obs.	Det.	Disc.	Offline	Known	Single	Language und.
Sohrabi, Riabov, and Udrea [72]	Plan	Agnostic	No	Part. obs.	Det.	Disc.	Offline	Known	Single	Theoretic
Jarvis, Lunt, and Myers [73]	Plan	Adversarial	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Surveillance
Pereira and Han [75]	Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Elderly care
Raghavan, Singla, and Mooney [76]	Plan	Agnostic	No	Fully obs.	Det.	Disc.	Offline	Known	Single	Various
Quaresma and Lopes [77]	Plan	Agnostic	No	Fully obs.	Det.	Disc.	Offline	Known	Double	Language und.
Vered and Kaminka [78]	Goal	Agnostic	No	Part. obs.	Det.	Cont.	Online	Known	Single	Destin. guess
Myers [79]	Plan	Intended	No	Part. obs.	Det.	Disc.	Offline	Known	Single	Planning help
Geib and Goldman [80]	Plan	Agnostic	No	Part. obs.	Det.	Disc.	Online	Known	Single	Theoretic
Ramirez and Geffner [81]	Plan	Agnostic	No	Fully obs.	Det.	Disc.	Online	Known	Single	Destin. guess
Pereira, Oren, and Meneguzzi [82]	Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	General
Granada, Pereira, Monteiro, et al. [124]	Action/Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Cooking
Amado, Pereira, Aires, et al. [125]	Action/Goal	Agnostic	No	Part. obs.	Det.	Disc.	Online	Known	Single	Puzzle solving
Classical machine learning approaches										
Akkaladevi and Heindl [126]	Action	Intended	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Human-robot int.
Han and Pereira [32]	Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Elderly care
Kelley, Tavakkoli, King, et al. [34]	Action/Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Multiple	Human-robot int.
Saria and Mahadevan [48]	Action/Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Multiple	Experimental
Laviers, Sukthankar, Aha, et al. [50]	Goal	Adversarial	No	Fully obs.	Stoch.	Cont.	Online	Known	Multiple	Video games
Oh, Meneguzzi, and Sycara [52]	Plan	Intended	No	Fully obs.	Det.	Disc.	Online	Known	Single	Assist. agents
Horvitz, Breese, Heckerman, et al. [56]	Goal	Agnostic	No	Fully obs.	Det.	Disc.	Online	Known	Single	Software help
Rebelo, Amma, Gamboa, et al. [59]	Activity	Intended	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Orthotics
Liao, Fox, and Kautz [84]	Action/Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Destin. guess
Oliver, Horvitz, and Garg [85]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Office awar.
Bui, Phung, and Venkatesh [86]	Action/Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Surveillance
Zhao, Wang, Sukthankar, et al. [87]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Experimental
Liao, Fox, and Kautz [88]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Baker and Tenenbaum [89]	Plan	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Experimental
Samanta and Chanda [90]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Video tagging
Fan, Wang, and Wang [91]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Vahdatpour, Amini, and Sarrafzadeh [92]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Elderly care
Polyvyanyy, Su, Lipovetzky, et al. [93]	Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	General
Deep learning approaches										
Meng and Huang [54]	Goal	Intended	No	Fully obs.	Det.	Disc.	Offline	Known	Single	Language und.
Rahmat, Niyaz, Javaid, et al. [57]	Goal	Adversarial	No	Fully obs.	Det.	Disc.	Online	Unknown	Single	Net. security
Hammerla, Halloran, and Plötz [95]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Daily life
Bevilacqua, MacDonald, Rangarej, et al. [96]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Experimental
Ronao and Cho [97]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Daily life
Yan, Xiong, and Lin [99]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Amado, Aires, Pereira, et al. [100]	Goal	Agnostic	No	Part. obs.	Det.	Disc.	Offline	Known	Single	Experimental
Ordóñez and Roggen [101]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Daily life
Plizzari, Cannici, and Matteucci [103]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Xin, Liu, Liu, et al. [104]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Zhang, Wu, and Luo [105]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Daily life
Min, Ha, Rowe, et al. [106]	Goal	Agnostic	No	Fully obs.	Det.	Cont.	Online	Known	Single	Video games
Singh, Chakraborty, Varshney, et al. [107]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Perrett, Masullo, Burghardt, et al. [108]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Daily life
Wang, Xing, Mei, et al. [4]	Activity	Agnostic	No	Part. obs.	Stoch.	Cont.	Offline	Known	Single	Video tagging
Brain-inspired approaches										
Kuniyoshi, Yorozu, Inaba, et al. [111]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Human-robot int.
Bonaiuto, Rosta, and Arbib [112]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Experimental
Oztop, Wolpert, and Kawato [113]	Action/Goal	Adversarial	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Experimental
Haruno, Wolpert, and Kawato [114]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Unknown	Single	Human-robot int.
Demiris and Khadhouri [115]	Action/Goal	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Human-robot int.
Yamashita and Tani [5]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Human-robot int.
Alkurdi, Busch, and Peer [116]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Daily life
Oltramari and Lebiere [122]	Action	Agnostic	No	Part. obs.	Stoch.	Cont.	Online	Known	Single	Surveillance

Table 1.2: Summary comparison among the types of approaches in terms of their ability to represent structured data, their expressivity, their capacity to deal with uncertainty, their flexibility to adapt to different data, their robustness, their competency at handling sensor data, the human effort required to develop the system, their scalability, their aptness to deal with open environments, and the maturity of the technology.

Properties/challenges	**Logic-based**	**Classical machine learning**	**Deep learning**	**Brain-inspired**
Structure	+ +	–	+	+
Expressivity	+ +	o	–	o
Uncertainty	o	+ +	+ +	+ +
Flexibility	– –	+	+ +	+
Robustness	–	+	+ +	+
Sensory input	– –	o	+ +	+
Human effort	– –	–	–	o
Scalability	–	–	++	o
Open environment	–	–	o	+
Maturity	+ +	+ +	+	–

A score is presented (– –, –, o, +, + +) for each type of approach and criterion, with the scores ranging from – – (very bad at it) to + + (very good at it).

introduction of a relevant part of the necessary knowledge. This generally implies much domain-expert human effort and systems that are hard to generalize, bad at scaling, prone to errors, and unable to handle plans or goals out of the scope of their domain knowledge. Deep learning approaches, on the other hand, are the natural evolution of classical machine learning techniques and are becoming the main solution to the problem of activity recognition due to their ability to deal with raw sensor data, extract features at different abstraction levels, and learn very different patterns from the training data (e.g., intraclass variability, ambiguous activities, etc.). These approaches, however, usually require large amounts of labeled data, which, again, implies much human effort. In addition, they are not good at handling and learning new activities online. Finally, brain-inspired approaches are an alternative that is sometimes used when it comes to real environments where the recognition needs to be done online (e.g., in human-robot interaction). These algorithms are hard to evaluate and compare with the others due to their great diversity and to the fact that they have not been in general as extensively explored. However, they show characteristics that place them as promising candidates to solve, in the near future, some of the yet unsolved challenges present in real unconstrained environments (e.g., learning new actions, plans, and goals in an unsupervised and online way and in few shots, similarly to how we humans do).

While systems addressing the whole problem are not common in the literature, end-to-end systems that are able to recognize from primitive actions to high-level plans may be very useful and even necessary in different applications. One way to approach this problem is to combine existing action recognition and plan recognition systems. For example, Granada, Pereira, Monteiro, et al. [124] built a logic-based plan recognition system upon a deep learning action

recognition system. However, the similarities between the two sub-problems may justify the development of more homogenous systems. Indeed, as commented in Section 1.2.3, the three sub-problems present similar characteristics and need to deal with similar challenges, and the whole problem can be considered a hierarchical task in which the different levels of the hierarchy perform action or goal/plan recognition with respect to the higher or lower levels, respectively. Some examples of existing systems approaching the whole problem in a unified way are those relying on hierarchical probabilistic models (e.g., [48]). A further investigation of deep learning and brain-inspired approaches in the context of plan recognition may also lead to new insights in this direction. For example, appropriately integrating self-reflection mechanisms in multimodal large language models may allow them to describe videos step by step and perform zero-shot plan recognition while also expressing how they reached such conclusion, addressing in this way one of the main limitations of deep learning systems: expressivity. Another direction that could lead to successful unified solutions is to find suitable ways of combining logic-based and deep learning approaches in a way that the advantages of both worlds can be exploited, similarly to what was done with logic-based and probabilistic approaches (some work in this direction has already been done, e.g., see [125]). These hybrid systems could be very good at dealing with sensory data while at the same time being very expressive at describing complete and complex plans. However, both logic-based and deep learning approaches are generally limited when it comes to open environments where it may be necessary to interpret and learn actions or plans never seen before. In this regard, bringing ideas from the brain mechanisms involved in the human recognition of actions and plans to the well-established deep learning networks may address these challenges and contribute to new successful systems that are also able to perform in open unconstrained environments. Ideas from cognitive science and from our knowledge on disorders related to the recognition of intentions or actions may also inspire the development of new successful brain-inspired computational systems.

Concerning the applications, some commercially successful action, plan, and goal recognition systems already exist in areas as diverse as virtual assistants or fitness/activity tracking. However, even in these successful areas, the performance is often not up to the user expectations or demands. In addition, there are many areas where the transition of new technologies from academia to common-use real world applications, together with the expected increasing demand, may soon bring important breakthroughs, such as human-robot interaction or elderly care. On the other hand, considering the recent boost in the areas of computer vision and natural language understanding, we can also expect important advances in these areas in the following years. As a consequence, applications relying on these technologies to perform action or goal recognition will also experience important advancements, as well as applications relying on other technologies that can also take advantage from advances in deep learning.

In conclusion, hybrid logic-probabilistic approaches and deep learning approaches are probably

the ones that are showing better results as of today in the problems of plan/goal recognition and activity recognition, respectively. Nevertheless, they still require further research to improve their performance and to tackle the challenges that are not able to address properly yet (e.g., learning online new activities, plans, or goals), as well as to be able to deal with the complete problem. In this respect, brain-inspired approaches, while not so extensively explored, count on some promising characteristics, and a further development may lead to new methods able to successfully address those challenges. Furthermore, combining ideas from these different approaches may contribute to finding new ways to address the yet challenging issues, as well as to find end-to-end solutions to the whole problem.

1.3 Aims

The **main objective** of this thesis is to propose and **develop a new action and goal recognition system** designed for human-robot interaction in real unconstrained environments. The system should draw inspiration from models of regions of the brain and be designed considering the potential requirements that such a system may have in different human-robot interaction scenarios (e.g., real-time recognition, prediction, online learning, etc.). In addition, it should be easily extendable to other real-world online interactive tasks such as action selection or action sharing in a human-robot collaboration scenario, being designed as a possible multi-purpose component of a larger cognitive architecture. A number of partial objectives have been defined as part of this main objective, which are described below:

New Cognitive Framework

The first objective is to develop a new unsupervised cognitive framework that integrates and combines brain mechanisms that are believed to be relevant in the human task of action, plan, and intention recognition. By framework we understand a set of tools that can be used to build different systems, such as a number of brain-inspired neural network architectures that perform specific functions and criteria on how to train them and connect them to build larger systems. While these tools are designed with the problem of action and goal recognition in mind, they may also be applicable to other tasks that face similar challenges or involve similar brain mechanisms when performed by humans, or they may be extendable with additional brain-inspired tools when necessary. In addition, such set of tools may be used to build functional models of regions of the brain that contribute to a better understanding of them and of those brain mechanisms.

Action and Goal Recognition System

The second objective is to develop a system that is able to recognize known actions and goals in real time based on this framework. This objective does not seek to develop a fully functional application-generic action and goal recognition system for any human-robot interaction scenario

but rather to test the developed framework on a more specific classification task and check whether it is able to achieve a good performance (note that the framework is still designed with a more general problem in mind). In this context, by good performance we understand not only reaching good accuracies after enough training but also being able to reach acceptable accuracies using a relatively low number of labeled samples. Indeed, we humans are able to learn to recognize new actions by seeing them a very small number of times, skill that is often crucial in open environments. Similarly, the developed system can take advantage of unsupervised learning techniques to learn the structure of the input data and in this way train classifiers successfully in few shots. In particular, in the case of unsupervised representation learning systems, those two measures (accuracies reached and number of training samples required) can provide a good idea of how meaningful the learned representations are.

System Adaptation to Other Tasks

Finally, the third objective is to adapt the system to other fully unsupervised tasks such as action and goal prediction or action selection. While this is not really part of the main objective, it will serve, first, to show that the framework can be easily extended to other tasks that may be required in a human-robot interaction scenario and, second, to provide ideas on how such extensions can be implemented. In addition, further directions will be provided on how to keep extending the system to eventually achieve a complete cognitive architecture that can be used to build robots that interact with humans in a more natural and fully autonomous way. Such system could also have research applications within a number of different fields, from cognitive or developmental robotics to cognitive neuroscience.

1.4 Outline

In this thesis, the development of a new brain-inspired self-supervised representation learning neural network architecture for temporal data is presented. This system learns representations of the input at different levels of abstraction, mimicking the hierarchies in the neocortex. These representations can then be used for different tasks, being often better suited for the subsequent processing than the original input itself. The system has been applied over synthetic and real action and plan data (as well as other temporal data), and the learned representations have been evaluated within a classification scenario and compared with other similar state-of-the-art systems, with our system outperforming all of them in this task. The system has also been extended with top-down predictive capabilities at the different levels (mimicking the feedback connections in the neocortex), being in this way suited for other tasks such as action and goal prediction and action selection. Further ways on how to keep extending the system have also been proposed. Source code is available at `https://github.com/franz-vh/lrnn`.

This dissertation is organized as follows: **Chapter 2** proposes a formalization of the problem

of activity, plan, and goal recognition. This chapter approaches the issue that many different formalizations can be found in the literature, often incompatible and/or specific to their problem, making the review of the literature slower and sometimes confusing. In this context, the formalization proposed in this chapter is defined in a generic way so that it can be applied to most recognition problems. It defines the environment and agents involved as tuples of sets and functions that describe their possible states and interactions (actions and observations in both directions) and their relations and transitions. The formalization is generic enough to be also applicable to other problems with agents involved. The chapter also describes the main problem approached in this thesis in terms of this formalization, which can serve as an example of how to apply it.

Chapters 3, 4, and 5 describe the complete system developed in this thesis (see Fig. 1.3). **Chapter 3** introduces the Neocortex-Inspired Locally Recurrent Neural Network (NILRNN), which is a shallow neural network whose connectivity pattern and sought functionality are inspired by models of the primary visual cortex. Considering the common assumption that the neocortex is relatively uniform along its different areas in terms of structure and underlying functionality and the fact that the main model we have drawn inspiration from relies on a quite generic principle to learn structure from temporal data in an unsupervised way, the NILRNN is also expected to be able to model the feedforward connections of other areas of the neocortex and to be applicable to other types of temporal data besides visual stimuli. In particular, the NILRNN relies on a novel form of low-level semantic pooling that can be seen as a generalization of the spatial pooling of CNNs for temporal data. The NILRNN is thought as an elementary block to build hierarchical circuits analogous to those involving multiple areas of the neocortex. It has been tested on action and other temporal data and compared with other shallow self-supervised learning systems, being able to outperform them. It has also been fed with sequences of images to compare its internal emerging behavior against that known of the primary visual cortex, showing an analogous behavior and validating it as a model of this area of the neocortex.

Chapter 4 presents the Hierarchical Locally Recurrent Neural Network (HLRNN), which is basically a stack of (modified/enhanced) NILRNNs, mimicking in this way the hierarchies in the neocortex. This system is able to learn, in a self-supervised way, representations of the input at different levels of abstraction, with the higher-level representations being generally more appropriate for more complex or higher-level tasks. The HLRNN has been compared with several state-of-the-art self-supervised representation learning systems on action and other temporal data, being able to outperform them. A further analysis has been performed to better understand its behavior, consisting of a comparison of the quality of the learned representations at the different levels for different classification tasks and of an ablation study, which showed, among other things, that the modifications introduced to the NILRNN led to a better behavior.

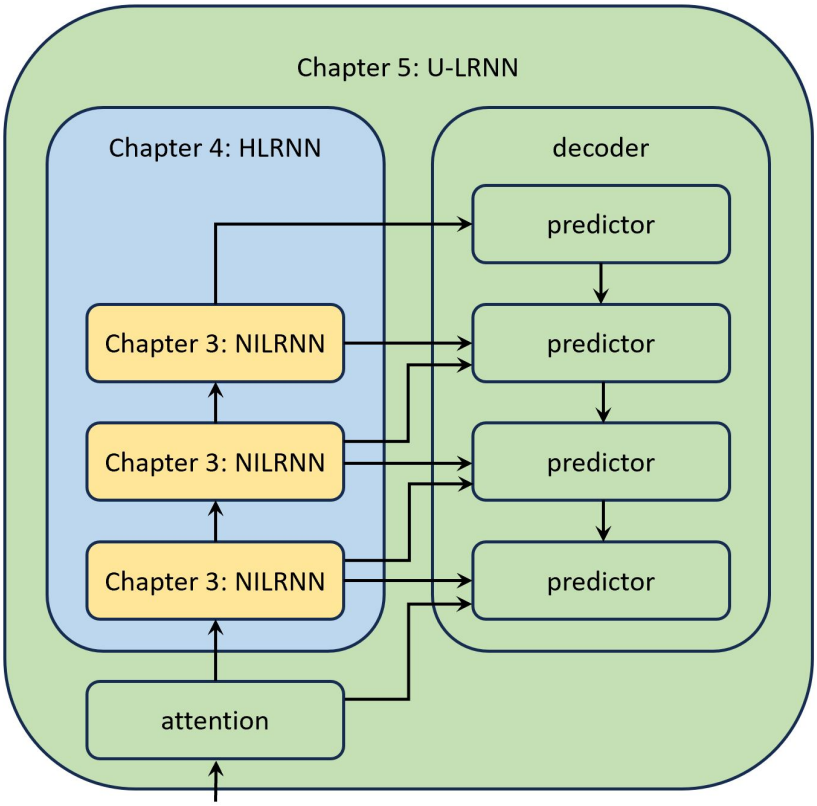


Figure 1.3: Outline of the main body of this dissertation. Chapter 3 introduces the NILRNN: a shallow neocortex-inspired neural network architecture for self-supervised representation learning and the main building block used in this thesis. Chapter 4 proposes the HLRNN: a deep neural network for self-supervised representation learning that mainly consists of a stack of NILRNNs. Chapter 5 presents the U-LRNN: a multi-purpose neural network with an encoder-decoder architecture that uses an HLRNN as encoder.

Chapter 5 introduces the U-shaped Locally Recurrent Neural Network (U-LRNN), which is a neural network that expands the HLRNN so that its learned representations can be used for unsupervised tasks such as action and goal prediction or action selection. The network consists of an encoder-decoder architecture, with the HLRNN taking the place of the encoder and the decoder being a sequence of neural networks that predict the next representation at the corresponding level of the encoder relying on the current representation and on the information coming from the level above. These predictions take the form of probability distributions that can be sampled and refed to the network to obtain multi-horizon predictions as well as information on their uncertainty. In the action selection configuration, the goal is introduced to the system at the top of the hierarchy, and the decoder decides what the next action representation is at each level based on the corresponding prediction. A number of possible directions are also provided on how to further extend the U-LRNN to transform it into a more complete cognitive architecture. On the other hand, this chapter proposes a set of augmentations and procedures to adapt the data coming from a motion capture system to the input of the U-LRNN, transforming

it into a sparse representation, which is the preferred form of input data for this system. In addition, it presents a self-supervised attention learning system for temporal data that allows the system to deal with variable-size inputs.

Finally, **Chapter 6** concludes the dissertation. This chapter provides a summary of the contributions of this thesis, going through what has been achieved and what is left for future work and proposing some possible directions on how to develop upon this work.

Chapter 2

FORMALIZATION OF THE PROBLEM OF ACTIVITY, PLAN, AND GOAL RECOGNITION

2.1 Introduction

This chapter proposes a formalization of the problem of activity, plan, and goal recognition. The motivation behind the development of this formalization relies on the multiple different and incompatible formalizations that can be found in the literature, often specific to their corresponding problem and/or using same terms to refer to different concepts, making the inspection of the literature sometimes confusing and slow and the comparison of approaches harder. Therefore, the purpose of this formalization is not to describe in detail each of the elements of this problem, but rather to provide a common general framework that can fit most recognition problems and which enables and eases their description and their comparison with others at a formal level. The framework proposed has been designed so that it is also easily adaptable to other problems involving a number of agents.

The problem of activity, plan, and goal recognition is composed of three (types of) elements: the environment, the observed agent(s) (the actor(s)), and the observing agent(s) (the observer(s)) [28]. These three elements can be considered sequential systems and, therefore, can be defined in terms of their possible inputs, internal states, and outputs and the functions that establish the relationships among those variables. In the following, we will define each of these elements.

2.2 The Environment

The environment can be defined as the tuple $N = \left(S_{env}, S_{env,0}, A_{env}^{over}, O_{env}^{from}, T_{env}, E_{env}\right)$, where:

- S_{env} is the set of possible states of the environment (i.e., the state space). It can be continuous or discrete.
- $S_{env,0} \subseteq S_{env}$ is the set of possible initial states of the environment (i.e., the initial state space). It is a subset of S_{env}. In a stochastic system, it can also be defined as the set of initial state probabilities, $\forall p \in S_{env,0} : p \in [0,1], \sum_{p \in S_{env,0}} p = 1$.
- A_{env}^{over} is the set of actions that can be performed over the environment (i.e., the affordance space). These actions are input variables to the environment system. At a certain instant, only one action can be performed over the environment. These actions can be defined as

tuples of action elements that may be performed over the environment, with each $a \in A_{env}^{over}$ being a tuple of n elements $a = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n)$, where $\hat{a}_i \in \hat{A}_{env}^{over}$ and $\hat{A}_{env}^{over}$ is the set of action elements that can be performed over the environment (note that constraints may exist among action elements). At a certain instant, zero, one, or multiple action elements may be performed over the environment. Each action element can only be performed by a single agent and over a single agent or environment, but the different action elements in an action may be performed by different agents. For example, the action of moving a heavy object may involve action elements performed by a number of agents.

- O_{env}^{from} is the set of observable states that can be obtained from the environment (i.e., the observable state space). These observable states are output variables from the environment system. These observable states can be defined as tuples of observable elements that may be obtained from the environment, with each $o \in O_{env}^{from}$ being a tuple of n elements $o = (\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_n)$, where $\hat{o}_i \in \hat{O}_{env}^{from}$ and $\hat{O}_{env}^{from}$ is the set of observable elements that can be obtained from the environment.

- $T_{env}\left(S_{env}, A_{env}^{over}\right)$ is the function that defines the next possible states of the environment given its current state and the action being performed over it (i.e., the transition function). The transition function can be deterministic or stochastic. A deterministic transition function assigns a new state to a given current state and performed action, $T_{env} : S_{env} \times A_{env}^{over} \rightarrow S_{env}$. A stochastic transition function assigns a probability to a given current state, performed action, and candidate new state, $T_{env} : S_{env} \times A_{env}^{over} \times S_{env} \rightarrow [0, 1]$. Note that we are modeling the process as a first order Markov model without loss of generality (every Markov model can be reduced to a first order Markov model).

- $E_{env}\left(S_{env}\right)$ is the function that defines the possible observable states given the current state (i.e., the emission function). The emission function can be deterministic or stochastic. A deterministic emission function assigns an observable state to a given current state, $E_{env} : S_{env} \rightarrow O_{env}^{from}$. A stochastic emission function assigns a probability to a given current state and candidate observable state, $E_{env} : S_{env} \times O_{env}^{from} \rightarrow [0, 1]$.

2.3 The Agents

An agent can be defined as the tuple $H = \left(S_{ag}, S_{ag,0}, A_{ag}^{by}, A_{ag}^{over}, O_{ag}^{by}, O_{ag}^{from}, T_{ag}, E_{ag}, M_{ag}, \pi_{ag}\right)$, where S_{ag}, $S_{ag,0}$, A_{ag}^{over}, O_{ag}^{from}, T_{ag}, and E_{ag} are analogous to those defined for the environment, with the transition function $T_{ag}\left(S_{ag}, A_{ag}^{over}, O_{ag}^{by}\right)$ depending also on the observation obtained by the agent, and:

- A_{ag}^{by} is the set of actions that can be performed by the agent (i.e., the action space). These actions are output variables from the agent system. At a certain instant, the agent can only

perform one action. These actions can be defined as tuples of action elements that the agent may perform, with each $a \in A_{ag}^{by}$ being a tuple of n elements $a = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_n)$, where $\hat{a}_i \in \hat{A}_{ag}^{by}$ and $\hat{A}_{ag}^{by}$ is the set of action elements that the agent can perform (note that constraints may exist among action elements). At a certain instant, the agent may be able to perform zero, one, or multiple action elements. Each action element can only be performed by a single agent and over a single agent or environment, but the different action elements in an action may be performed over different agents and/or the environment. For example, the action of moving an object ($\in A_{ag}^{by}$) may involve action elements modifying the state of the environment ($\in \hat{A}_{env}^{over}$) and of the agent ($\in \hat{A}_{ag}^{over}$).

- O_{ag}^{by} is the set of observations that can be obtained by the agent (i.e., the observation space). These observations are input variables to the agent system. At a certain instant, the agent obtains a single observation. These observations can be defined as tuples of observable elements that the agent may obtain, with each $o \in O_{ag}^{by}$ being a tuple of n elements $o = (\hat{o}_1, \hat{o}_2, \ldots, \hat{o}_n)$, where $\hat{o}_i \in \hat{O}_{ag}^{by}$ and $\hat{O}_{ag}^{by}$ is the set of observable elements that the agent can obtain. Each observable element can only be obtained from a single agent or environment, but the different observable elements in an observation may be obtained from different agents and/or the environment. These observable elements may or may not correspond to those defined for the observable states, depending on the sensor model of the agent (defined below). The actions performed over the agent A_{ag}^{over} may or may not be observed by the agent.

- $M_{ag}\left(S_{ag}, O_{all}^{from}\right)$ is the function that defines the observations that the agent can obtain given its state and the set of all the current possible observable states, $O_{all}^{from} = O_{env}^{from} \cup O_{ags}^{from}$, with $O_{ags}^{from} = \bigcup_i O_{ag,i}^{from}$ (i.e., the sensor model). The sensor model can be deterministic, i.e., $M_{ag} : S_{ag} \times O_{all}^{from} \rightarrow O_{ag}^{by}$, or stochastic, i.e., $M_{ag} : S_{ag} \times O_{all}^{from} \times O_{ag}^{by} \rightarrow [0, 1]$.

- $\pi_{ag}\left(S_{ag}\right)$ is the function that defines the action that the agent performs given its state (i.e., the policy). These action decisions may be taken in a deterministic way, i.e., $\pi_{ag} : S_{ag} \rightarrow A_{ag}^{by}$, or in a stochastic way, i.e., $\pi_{ag} : S_{ag} \times A_{ag}^{by} \rightarrow [0, 1]$. The policy will in general try to select the best action to reach the agent goal given the current agent knowledge, with the agent goal and knowledge being part of its state, as we describe below. In the case of solutions considering multiple policies depending on the agent knowledge and/or goal, all those policies would be part of the single policy in this formalization.

Each possible state of the agent $s \in S_{ag}$ can be defined as the tuple $s = (s', k, g)$, where $s' \in {S_{ag}}'$, $k \in K_{ag}$, and $g \in G_{ag}$, and:

- K_{ag} is the set of possible knowledge of the agent (i.e., the knowledge space). This knowledge can exist a priori or come from the observations O_{ag}^{by}. Each possible knowledge $k \in K_{ag}$ contains the knowledge about the environment and the knowledge about the agents and, therefore, can be defined as a tuple of n knowledge elements $k = \left(\hat{k}_1, \hat{k}_2, \ldots, \hat{k}_n\right)$, where $\hat{k}_i \in \hat{K}_{ag}^{env} \cup \hat{K}_{ag}^{ags} \cup \hat{K}'_{ag}$, being $\hat{K}_{ag}^{env}$ the possible knowledge elements of the agent about the environment, $\hat{K}_{ag}^{ags} = \bigcup_j K_{ag}^{ag_j}$ the possible knowledge elements of the agent about the agents, and $\hat{K}'_{ag}$ the remaining possible knowledge elements of the agent. Note that the knowledge about the environment may come both from observations of the environment and of the agents, and the same applies to the knowledge about the agents. We can define a knowledge transition function as a subset of the main transition function that defines the next possible knowledge of the agent given its current knowledge, its current goal, and the obtained observation $T_{ag}^{K}\left(K_{ag}, \hat{G}_{ag}, O_{ag}^{by}\right) \subset T_{ag}$. Note that it does not depend on the rest of the agent state or on the action performed over the agent, as the agent can only obtain knowledge about those variables through observations.

- G_{ag} is the set of possible goals of the agent (i.e., the goal space). At a certain instant, the agent can only have one goal. These goals can be defined as tuples of goal elements that the agent may have, with each $g \in G_{ag}$ being a tuple of n elements $g = (\hat{g}_1, \hat{g}_2, \ldots, \hat{g}_n)$, where $\hat{g}_i \in \hat{G}_{ag}$ and $\hat{G}_{ag}$ is the set of goal elements that the agent can have. At a certain instant, the agent may have zero, one, or multiple goal elements. These goal elements may be defined in a more general way as the motivations behind the actions of the agent, being able in this way to explain a larger number of actions (e.g., reactive actions). We can define a goal transition function as a subset of the main transition function that defines the next possible goals of the agent given its state, the action being performed over it, and the obtained observation $T_{ag}^{G}\left(S_{ag}, \hat{A}_{ag}^{over}, O_{ag}^{by}\right) \subset T_{ag}$.

- ${S_{ag}}'$ is the set of possible substates (i.e., the substate space), being the substate the remaining element of the state tuple as defined above. We can define a substate transition function as a subset of the main transition function that defines the next possible substates of the agent given its current substate and the action being performed over it ${T_{ag}}'\left({S_{ag}}', A_{ag}^{over}\right) \subset T_{ag}$. Note that it does not depend on the knowledge or goal of the agent or on the obtained observation.

The **actor** is an agent. Therefore, it can be defined as the tuple $X = \Big(S_{act}, S_{act,0}, A_{act}^{by}, A_{act}^{over}, O_{act}^{by}, O_{act}^{from}, T_{act}, E_{act}, M_{act}, \pi_{act}\Big)$. The policy of the actor π_{act} will in general select actions that the actor believes that lead to the accomplishment of its goal given its knowledge. As an example, in the common case of single actor keyhole systems (as in most of this study), in general, the actor will not obtain observations from the observer(s), $O_{act}^{obs} = \emptyset$, it will not perform ac-

tions over the observer(s), $A_{act}^{obs} = \emptyset$, and it will maintain no knowledge about the observer(s), $K_{act}^{obs} = \emptyset$. In addition, if we consider that the actor does not obtain observations from itself, $O_{act}^{act} = \emptyset$, and that it does not perform actions over itself, $A_{act}^{over} = \emptyset$, its description is reduced to $X_{keyhole} = \left(S_{act}, S_{act,0}, A_{act}^{env}, O_{act}^{env}, O_{act}^{from}, T_{act}, E_{act}, M_{act}, \pi_{act}\right)$. This may also hold for other no intervention scenarios, as far as the actor cannot observe and interact with the observer.

Similarly, the **observer** is an agent and can be defined as the tuple $Y = \left(S_{obs}, S_{obs,0}, A_{obs}^{by}, A_{obs}^{over}, O_{obs}^{by}, O_{obs}^{from}, T_{obs}, E_{obs}, M_{obs}, \pi_{obs}\right)$. The goal of the observer will always include goal elements $\hat{g}_{rec} \in \hat{G}_{obs}$ to recognize the activities, plans, and/or goals of the actor, possibly subject to a set of given constraints (e.g., without interacting with it). Consequently, the observer will update its knowledge (according to T_{obs}^{K}) and perform actions (according to π_{obs}) to reach this goal. Its knowledge about the actor(s) K_{obs}^{act} usually includes some model of it (e.g., of its policy, its possible observations, etc.) that is updated with the observations of the actions performed and the changes in the environment. This model helps the observer with the recognition task of inferring what activity the actor(s) is performing or what plan(s) or goal(s) would have led the actor(s) to perform the observed actions. Going back to the example of the keyhole system, in general, no actions will be performed over the observer, $A_{obs}^{over} = \emptyset$, no observable states will be obtained from the observer, $O_{obs}^{from} = \emptyset$, and the observer will not perform actions, $A_{obs}^{by} = \emptyset$, which implies that it will not have an emission function E_{obs} or a policy π_{obs} associated and that its substate ${S_{obs}}'$ is irrelevant, being its state (and transition function) just defined by its knowledge and goal (and corresponding transition functions). Therefore, the description is reduced to $Y_{keyhole} = \left(S_{obs}, S_{obs,0}, O_{obs}^{by}, T_{obs}, M_{obs}\right)$. Again, this may also hold for other no intervention scenarios.

As a comment, the environment can also be seen as an agent that does not perform actions, $A_{env}^{by} = \emptyset$, and does not obtain observations, $O_{env}^{by} = \emptyset$, and, therefore, does not have a sensor model M_{env} or policy π_{env} associated. Regarding its state, the environment does not have knowledge, $K_{env} = \emptyset$, or goal, $G_{env} = \emptyset$, being therefore the state only formed of its substate, $s = (s')$, with $s \in S_{env}$ and $s' \in {S_{env}}'$.

2.4 The Problem

Finally, we can define the complete activity, plan, and/or goal recognition problem as the tuple $P = \left(K_{obs,0}, S_{act,rec}, A_{obs}^{by}, O_{obs}^{by}, F_{sys}, g_{rec}\right)$, where $K_{obs,0}$ is the set of possible initial knowledge, $S_{act,rec}$ is the set of possible states of the actor to be recognized (activity, plan, and/or goal), and $F_{sys}\left(\bar{a}_{obs,[1,T]}^{by}, \bar{o}_{obs,[1,T]-t}^{by}, s_{act,rec}, t\right)$ is the unknown function that defines the observation at time $t \in \mathbb{N}$ given the history of performed actions $\bar{a}_{obs,[1,T]}^{by} = (a_1, a_2, \ldots, a_T)$ with $a_i \in A_{obs}^{by}$, the history of obtained observations $\bar{o}_{obs,[1,T]-t}^{by} = (o_1, o_2, \ldots, o_{t-1}, o_{t+1}, \ldots, o_T)$ with $o_i \in O_{obs}^{by}$ ($\bar{o}_{obs,[1,T]-t}^{by} = (o_1, o_2, \ldots, o_T)$ if $T < t$), and the unknown state of interest $s_{act,rec} \in S_{act,rec}$,

where T is the time of the latest data available (in the case of online recognition, $T \approx t-1$). We can phrase this problem in the following way: Given an initial knowledge $k_{obs,0} \in K_{obs,0}$, an unknown state of interest $s_{act,rec} \in S_{act,rec}$, a set of possible actions to perform at each instant A^{by}_{obs}, and an observation at each instant $o^{by}_{obs,t} \in O^{by}_{obs}$ defined by the unknown function F_{sys} (function of the actions performed, the rest of observations available, and the state of interest), solve the task g_{rec} of recognizing the state of interest of the actor subject to a set of given constraints (e.g., not taking actions that may modify that state). The general solution of the problem consists of finding an approximation $\tilde{s}_{act,rec} = \tilde{F}_{rec}\left(\bar{a}^{by}_{obs,[1,T]}, \bar{o}^{by}_{obs,[1,T]-t}, o^{by}_{obs,t}, t\right)$ to the solution for variable $s_{act,rec}$ of the unknown function F_{sys} together with a policy $\pi\left(\bar{a}^{by}_{obs,[1,T]-t}, \bar{o}^{by}_{obs,[1,T]}, t\right)$ that selects actions to make this task as simple as possible subject to the given constraints.

In this study, we have mainly worked with labeled datasets, which means that it is a no intervention scenario where the observer can perform no actions, $A^{by}_{obs} = \emptyset$, and the possible states of the actor to be recognized $S_{act,rec}$ are known and defined by the dataset labels. We have defined the goal g_{rec} as the task of recognizing the label at each instant $s_{act,rec,t}$ in an online manner. Therefore, the unknown function $F_{sys}\left(\bar{o}^{by}_{obs,[1,t-1]}, s_{act,rec,t}, t\right)$ now does not depend on the actions performed by the observer and, at each instant, depends on the state of interest at that instant, with $T = t-1$. Since we make use of neural networks to approach the problem, we can define the initial knowledge $k_{obs,0} \in K_{obs,0}$ as the learned weights and biases during a previous training process, the observations $o^{by}_{obs,i} \in O^{by}_{obs}$ as the inputs to the neural network, and the guessed states $\tilde{s}_{act,rec,t} \in S_{act,rec}$ as the outputs of the neural network. Regarding the general solution, it does not require a policy, and the approximation to the solution of the unknown function for the state of interest is the trained neural network itself, $\tilde{s}_{act,rec,t} = \tilde{F}_{rec}\left(\bar{o}^{by}_{obs,[1,t]}, t\right)$.

Chapter 3

NILRNN: A NEOCORTEX-INSPIRED LOCALLY RECURRENT NEURAL NETWORK FOR SELF-SUPERVISED REPRESENTATION LEARNING IN TEMPORAL DATA

The content of this chapter was published in [127] and [128].

3.1 Introduction

We call feature extraction the process of extracting values of interest (features) from the input data with the intention of finding new representations of the data that are easier to work with or to interpret. Feature extraction is commonly used, for instance, in machine learning applications, to find alternative representations of the input data from which it is easier to extract the desired information (e.g., in speech recognition, working with the frequency components instead of with the audio signal itself simplifies considerably the task). It is also often used to reduce the dimensionality of the input data, by removing the redundant information while keeping the relevant data (e.g., to compare videos in a large database, working directly at the pixel level may be overwhelming, and extracting the information from keypoints of interest can help make the problem manageable).

The feature extraction process can be either handcrafted or learned [129]. Handcrafted feature extraction systems (e.g., scale-invariant feature transform, SIFT, for images [130], mel-frequency cepstral coefficients, MFCC, for audio [131], etc.) involve generally the work of domain experts that understand what features may be most relevant to best describe the input data. These systems are, therefore, application-specific, besides requiring expert knowledge for their design. Feature extraction learning systems (also known as feature learning or representation learning systems), on the other hand, automatically learn the feature extracting mechanism from the available data by finding patterns or correlations in the data. Therefore, they can be used in very different applications, and they generally require much less manual work of experts in the specific field. In addition, these learning systems often lead to features or representations that give better results than those coming from handcrafted systems, since they are able to automatically find more complex or barely perceptible correlations in the data. This has led them to become in the last years the main method to extract features in many fields and applications. A typical use case of these systems is within sequential transfer learning tasks [132], where a set of features are learned in a first step known as pre-training, followed by a second step where the learned features are used for a downstream task such as classification (typically, the learned feature functions are also fine-tuned for the specific task).

Representation learning systems are often classified into traditional systems (e.g., principal component analysis, PCA [133], linear discriminant analysis, LDA [134], etc.) and deep learning systems (e.g., stacked autoencoders [135], deep belief networks, DBN [136], etc.) [137]. Deep learning systems, while more demanding in terms of computational cost and of the amount of data required, have been shown to be able to find and extract more complex and useful features, which has resulted in an increase in their popularity during the last years. These systems appear to find representations of the input at different levels of abstraction, with the latter layers learning more abstract and complex features.

Regarding the representation learning process, it can be supervised (e.g., LDA, supervised neural networks [138], etc.) or unsupervised (e.g., PCA, autoencoders, etc.) [137]. Supervised representation learning systems count on the information about the desired output for each input sample and, therefore, can learn representations that are better adapted to the specific task. However, it is required that each training sample is labeled with its corresponding desired output. Unsupervised representation learning systems, on the other hand, learn representations using only the input data. One of the main advantages of unsupervised learning is that unlabeled data is in general more ubiquitous and accessible than labeled data, allowing unsupervised systems to be trained with large amounts of data and avoiding the human effort involved in the labeling of data (something that is even more relevant in the era of deep learning). In addition, even when all the available data is labeled, systems pre-trained in an unsupervised way are often able to outperform fully supervised systems on the downstream task (e.g., [139]), which indicates that unsupervised learning can sometimes find more robust features than supervised learning. Furthermore, the learned features can in general be considered general-purpose, being reusable in different applications without the need of learning new ones (note that features learned through supervised learning are also often general-purpose [138]).

Current unsupervised representation learning systems often make use of self-supervised techniques [140], i.e., they generate automatically (without human annotation) target labels, outputs, and/or modified inputs from the input data (or define desired output properties) that allow the system to be trained using standard supervised techniques such as backpropagation. Self-supervised representation learning has gained much attention in the last years, being successfully used in various applications and domains and applied to different types of data such as images [139], video [141], text [142], audio [143], or physiological signals [144]. Since they do not count on labeled information, these systems often require to be designed considering the specific characteristics of their input data so that they can learn useful features [145]. For example, systems dealing with temporal data (e.g., human action data) generally exploit this characteristic of the data to learn better representations (in this work, by temporal data we refer to time series with their samples taken at regular intervals in time, avoiding other forms of temporal data such as event data, generally processed in very different ways [146]). Still, the existing systems dealing

with temporal data (e.g., long short-term memories, LSTM [147], transformers [148], etc.), while have shown very good results in the last years for different time-related problems, do not yet seem able to learn representations of their input that are as good and robust as those learned by our brain, which is essentially also a temporal system. Indeed, these systems present certain limitations in real-life scenarios when compared to the human brain, which is very good at adapting to, interpreting, and reasoning over situations that it has never seen before, being able to apply its knowledge in more general ways [149]. In this sense, our brain can still be considered the best-known general-purpose system at dealing with temporal data. However, no structures similar to LSTMs or transformers have been found in our brain, and, therefore, the brain appears to deal with temporal information in a different way. Thus, it seems reasonable to search for new ways to adapt neural networks to temporal data by drawing inspiration from the functioning and structure of the brain. This may lead to the development of new architectures and systems that are able to deal better with temporal data and that are closer to the high performance of the human brain.

In this chapter, we propose a system that tries to take a step in this direction, by bringing structures observed in the neocortex to the well-established artificial neural networks. This new system is called Neocortex-Inspired Locally Recurrent Neural Network (NILRNN), and it is a recurrent neural network for self-supervised representation learning in temporal data that draws inspiration from connection patterns observed in the neocortex. This network relies on a novel 2D layer (the locally recurrent layer) with a pattern of connectivity able to develop semantic order along its neurons. This order consists of nearby neurons representing similar concepts and being pooled together, achieving in this way a form of low-level semantic pooling that can be seen as a generalization of the spatial pooling of convolutional neural networks (CNNs). The system proposed is actually a shallow network and does not pretend to compete in performance with more complex deep recurrent networks. It has been rather designed to explore a different direction on how to deal with temporal data, focusing on the problem of the unsupervised learning of representations while making it easily extendable to (or integrable into) deep architectures. In this regard, the system has been compared with several well-known shallow self-supervised learning systems in the task of learning representations for classification problems with data from different domains, being able to outperform them. In addition, to acquire a better understanding of its internal workings, its emerging behavior has been compared against that of the primary visual cortex, showing to be analogous upon all the visual cortex properties for which the NILRNN was evaluated. This indicates that the system is behaving as desired and, furthermore, positions it as a valid computational model of this (and possibly other) areas of the neocortex, i.e., as a potential tool to study them and gain a deeper understanding of their functioning. We believe that the promising results obtained in this study already place the NILRNN as a candidate for the task of representation learning in simple temporal applications. Its extension to deep

systems has also brought new successful results, as will be described in Chapter 4.

This chapter is organized as follows: Section 3.2 introduces concepts about the human brain and machine learning systems and mechanisms upon which the NILRNN has been designed. Section 3.3 describes the NILRNN architecture. Section 3.4 presents a comparison of the results obtained for the NILRNN and other similar systems and for different datasets, as well as a comparison of its behavior against known behavior of the neocortex. Section 3.5 discusses on the NILRNN and the results obtained and proposes some possible future directions. Finally, Section 3.6 summarizes the results obtained and their implications and concludes the chapter.

3.2 Background

This section introduces some of the main concepts and mechanisms upon which the NILRNN has been developed, including notions about the structure and function of our neocortex and about machine learning systems and techniques.

3.2.1 The Neocortex

The neocortex is a thin layered structure surrounding the brain that is involved in sensory perception, rational thought, voluntary motor control, language, and other high-level cognitive functions [150]. This region of the brain is divided in different areas, which perform different functions and are organized in hierarchical structures, with areas higher in the hierarchies encoding more complex or abstract concepts [151]. For example, the ventral stream of the visual cortex, which is believed to be involved in object recognition, is often described as a hierarchy of neocortical areas, with areas lower in the hierarchy getting less processed visual input and encoding simple features such as edges in the image and areas higher in the hierarchy using the features extracted by lower areas to encode more complex features, such as those describing whole objects [152]. These areas within the hierarchy are connected through feedforward, feedback and horizontal connections [153]. Still, the neocortex seems to be quite homogenous along most of its areas in terms of structure and operation, with most of its areas organized in a relatively uniform six-layered structure [154]. This suggests a common underlying algorithm governing the different areas and functions [155]. On the other hand, similarly to other regions of the brain, the strength of the connections between neocortical neurons evolves depending, among other factors, on the activation correlations between those neurons. These learning processes are often modeled through Hebbian rules, which state that connections between neurons with correlated activations get strengthened [156]. Homeostatic plasticity mechanisms also exist that maintain stability in the network while permitting changes to occur, e.g., by regulating neuron activity so that neurons don't spend too much time active or inactive [157].

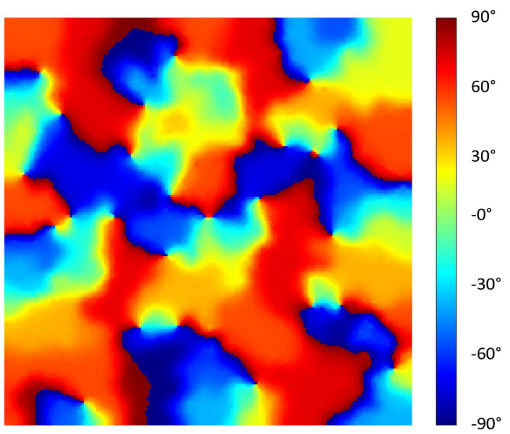
Probably, the most-studied and best-known areas of the neocortex are those belonging to the ventral stream of the visual cortex, and, in particular, the primary visual cortex, which is the

earliest area in the neocortex processing the incoming visual information from the eyes (i.e., the lowest area of the hierarchy). This area gets the visual input from the thalamus, processes it and forwards it to the next areas in the visual cortex [158]. Within these areas, the most-studied connections, and the ones that are usually considered when designing models of this hierarchy, are the feedforward connections, which are responsible for the flow of visual information from the eyes to the higher levels of the hierarchy (backward and horizontal connections seem to be more related to performing top-down predictions [159] or to work as top-down attention mechanisms [160], but their function is not so well understood). In general, feedforward connections mainly originate in layers 2 and 3 (L2/3) of the neocortex areas (or in the thalamus) and reach layer 4 (L4) of the next area in the hierarchy (which can be considered as the starting point of the feedforward information processing in the area). Then, the information propagates within the area through cortical columns (transversally to the layers) to L2/3, to then go through the feedforward connections to the next area of the hierarchy [154], [161]. As a consequence, models of the ventral stream that focus on the feedforward connections often model only L4 and L2/3, not considering the other layers of the neocortex.

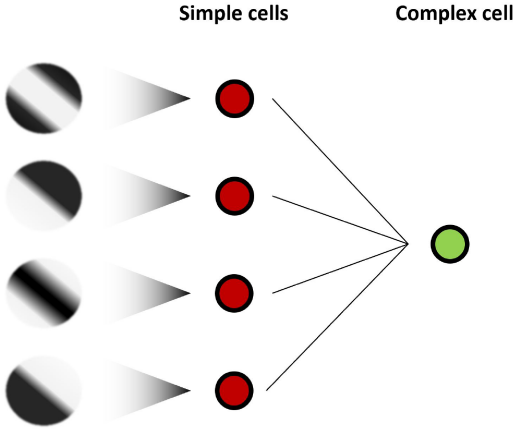
Neurons in L4 and L2/3 of the primary visual cortex are sensitive to small regions of the input stimuli known as receptive fields and are often classified as simple or complex cells: Simple cells tend to respond to edges in their receptive field with a specific orientation and position and are mainly found in L4, while complex cells tend to respond to edges with a specific orientation but are more position invariant (i.e., small shifts in the input image affect little their activity), being mainly found in L2/3 [162]. To study this behavior, sine gratings as those shown in Fig. 3.1 are typically used as visual stimuli, for which simple cells tend to fire after a particular orientation and phase, while complex cells fire after a particular orientation but are more phase invariant. The fact that these neurons respond so selectively to specific patterns in the input seems to indicate that the neocortex represents information through a sparse coding scheme, with most activity in a given time occurring only in a small proportion of the neurons, something that is consistent with physiological evidence [163]. Neurons in L4 and L2/3, besides being connected to neurons in other layers of the same or different areas, are also connected to neurons in the same layer through short-range excitatory and long-range inhibitory lateral connections [164], being these connections more numerous and relevant in L2/3 [165]. In addition, they are distributed in a way that neurons with similar receptive fields and orientation preferences (as well as other properties, e.g., ocular preference) are located close to each other, forming smooth ordered maps as the one shown in Fig. 3.2a [166], while no such organization seems to exist when it comes to phase preferences in L4 [167].

Many models have been designed with the purpose of describing the emergence of these properties in the primary visual cortex. The behavior of simple cells is usually achieved through Hebbian-like learning techniques, which, when applied over small regions of input images, lead

Figure 3.1: Examples of sine gratings of different orientations, spatial frequencies, and phases. These types of gratings are often used as visual stimuli to analyze the behavior of neurons in the primary visual cortex.



(a) Representation of the orientation maps in the primary visual cortex [166].



(b) Simplified model of the simple and complex cells in the primary visual cortex [168].

Figure 3.2: Orientation maps and simple and complex cells in the primary visual cortex. Orientation maps show the distribution of visual stimulus orientation preferences of neurons in the primary visual cortex, presenting properties such as homogeneous regions of similar orientation preference, periodicity, or pinwheels. The model of simple and complex cells shows how simple cells fire after patterns in their receptive field in the form of edges with specific orientations and phases, while complex cells achieve their orientation selectivity and phase invariance by pooling simple cells firing after similar orientations but different phases.

to the learning of edge patterns of particular orientations and phases. Sparse activities, on the other hand, can be obtained by implementing lateral inhibitory connections and homeostatic plasticity mechanisms. Regarding complex cells, their behavior is typically achieved by selectively pooling simple cells with similar receptive fields and orientations but with shifts in the position of the edge (i.e., with different phases), leading to a higher phase tolerance while maintaining the orientation selectivity, as shown in Fig. 3.2b [168]. If L4 contains simple cells with similar orientations but different phases located close to each other (as has been observed in the primary virtual cortex), the behavior of complex cells can emerge by just pooling indiscriminately a region of L4 (the order in the distribution of receptive fields is generally hardwired, mimicking the connectivity patterns observed). However, achieving this orientation order and phase disorder in a biologically plausible way has been more challenging. Many models achieve the orientation maps by defining strong short-range excitatory lateral interactions in L4. However, this generally leads to neurons next to each other firing simultaneously and, therefore,

having similar phases. This makes the emergence of complex cell behavior more problematic, as these cells would have to selectively pool simple cells with similar orientations but different phases that would be located in separated positions. Antolik and Bednar [169] proposed a model able to achieve this orientation order and phase disorder in a biologically plausible way by including strong lateral interactions in L2/3 and feedback connections back to L4. This brings a time delay in the coupling between nearby neurons in L4. As a consequence, nearby neurons are not anymore pushed to fire simultaneously and to learn the same input patterns. Instead, they are pushed to fire close in time, learning patterns that tend to appear in the input with certain delay. Input stimuli to our visual system generally shift smoothly in time, which implies that, for small regions and time windows, the orientation of input edge patterns stays constant, and only the phase changes. This pushes nearby neurons in L4 (which have similar receptive fields) to learn patterns with similar orientations but shifted in space, achieving an order analogous to that observed in the primary visual cortex. In fact, this interpretation of the emergence of orientation maps and complex cells in terms of time delays instead of in terms of image patterns may also describe what occurs in other areas of the neocortex processing other types of data (as the different areas of the neocortex appear to have a similar structure and operation, making some of the properties described here for the primary visual cortex also valid to other areas). This would mean that neurons in L2/3 of other areas of the neocortex are quite invariant to features in their input that change in short timescales, being mainly sensitive to slower-varying features (such as edges phases versus orientations for the case of the primary visual cortex).

3.2.2 Brain-Inspired Machine Learning Systems and Mechanisms

This section introduces some brain-inspired systems and mechanisms that have also served as inspiration to develop our system or that may be useful to better understand its design.

Probably, the best-known and most popular brain-inspired machine learning systems are **artificial neural networks**. These networks are formed of artificial neurons that are organized in layers and stacked forming hierarchies, with the lowest-level layer processing the input to the system and representing it in terms of simple features and the higher layers extracting more complex and abstract features [94]. This is similar to what has been observed in our brain, and in the neocortex in particular, as described in Section 3.2.1. **CNNs** are a good example of brain-inspired neural networks, as they mimic to some extent the feedforward connections in the ventral stream of the visual cortex. Especially in the earlier convolutional layers of CNNs, neurons tend to behave in an analogous way to simple cells in L4 of the primary visual cortex, learning specific patterns from small regions of their input (e.g., edges with a particular orientation and phase). Neurons in the early max pooling layers, for their part, tend to behave in a similar way to complex cells in L2/3, pooling neurons that detect shifted versions of the same input pattern [170]. CNNs, however, do not rely on learning a distribution of patterns that

show orientation order but phase disorder to then pool nearby neurons together, but they are explicitly designed (i.e., hardwired) to pool neurons detecting the same pattern at slightly shifted positions of the input image. A common interpretation on why **spatial pooling** in CNNs works comes from assuming that slightly shifted versions of an edge in a region of an image contribute essentially with the same information to its overall meaning, and, therefore, by grouping neurons detecting those shifted edges, the network loses little relevant information while simplifying the representation.

This interpretation is related to the concept of **slowness**: The slowness principle, inspired by behavior observed in the neocortex, states that the environment changes in a slower timescale than the sensory input we get from it (e.g., objects in a video vs. pixels) [171]. Hence, good representations of the environment should also change in such slower timescale. For example, slow feature analysis (SFA) [171] is an algorithm based on this principle that learns functions whose slow-changing output only depends on the input at each instant, avoiding in this way functions such as low-pass filters. This algorithm has been able to achieve similar behaviors to those observed in the brain, such as those of complex cells when having natural image sequences as input, detecting edges with specific orientations independently of their position [172]. This idea has also been applied hierarchically, leading to different timescales at different levels of complexity or abstraction. For example, the multiple timescales recurrent neural network (MTRNN) [5] relies on this idea to learn complex actions formed of concatenated simpler movements.

Sparse coding, already mentioned in Section 3.2.1, is another brain-inspired mechanism that consists of building representations with only a small percentage of active neurons for each given input. This form of coding is often very appropriate to represent the observations of the real world, as these observations can usually be described in terms of the presence, at each instant, of a small number of features out of a considerably larger number of possible features (e.g., edges of specific orientations, the presence of certain objects, etc.). In addition, sparse coding has shown several advantages when applied to artificial systems. For instance, its inherent redundancy seems to contribute to robustness and fault-tolerance [121], as well as to dealing with partial information [173], and its highly separated representations are also very appropriate for applications requiring incremental or few-shot learning, avoiding catastrophic forgetting [174].

Self-organization, on the other hand, is a behavior that not only appears in the brain (e.g., with neurons in the primary visual cortex responding to similar orientations appearing close to each other, see Fig. 3.2a) but also in nature in many different forms. It consists of the spontaneous emergence of some form of global order in a system due to the local interactions among its components, without the need of intervention of an external agent. Since self-

organization mechanisms in artificial systems can take many different forms, their advantages are also diverse. Still, a typical common advantage is that it contributes to both the adaptability and the robustness of the system [175]. It also shows advantages for incremental and few-shot learning applications, as well as in terms of the interpretability of the system.

Finally, **Hebbian learning** is a model of how neurons in the brain learn. It is an unsupervised and online learning model that consists of strengthening the connections between neurons with a correlated activity (and usually also weakening the connections with no correlated activity) [156]. Hebbian learning, while popular among models of brain regions, has some limitations that make it not the preferred option in machine learning solutions: It is unstable by nature, and it often requires supplementary processes or structures (e.g., homeostatic mechanisms, lateral inhibitory interactions, feedback circuits, etc.) to solve simple learning problems, making the system design process more complex [176].

3.2.3 Autoencoders

When it comes to unsupervised machine learning systems, autoencoders [94] are a common choice in most cases preferred to Hebbian learning, as their behavior is better explored and understood, relying on the better-established backpropagation learning technique. Therefore, these algorithms make it more straightforward to design systems with a certain desired behavior and performance.

The autoencoder is a typical example of self-supervised learning neural network, as it uses the backpropagation learning method (supervised) to learn new representations of the input data in an unsupervised way. This is done by setting as desired output a copy of the input data. In this way, the network learns alternative representations of the input data in its hidden layer(s), keeping the main information of the input data to then be able to reconstruct it. To make the autoencoder learn new representations and extract useful features from the data (and prevent it from just learning the trivial solution of keeping a copy of the input in its hidden layer(s)), autoencoders include constraints in their hidden layer(s), such as having a smaller number of neurons in the hidden layer(s) than in the input/output layer. An autoencoder with such constraint is called undercomplete (see Fig. 3.3). Other popular types of autoencoders are sparse autoencoders, stacked autoencoders, denoising autoencoders, and variational autoencoders [94].

Sparse autoencoders enforce a sparse activity in their hidden layer(s), i.e., they implement a mechanism so that, for each possible input, only a small percentage of the neurons in their hidden layer(s) are active (or most of them have activities close to 0). This is typically done by adding a term to the loss function that penalizes activities in the hidden layer(s) that are different from those just described [94]. For example, it is common to approximate the activity in each neuron of the hidden layer(s) by a Bernoulli random variable and define a desired Bernoulli

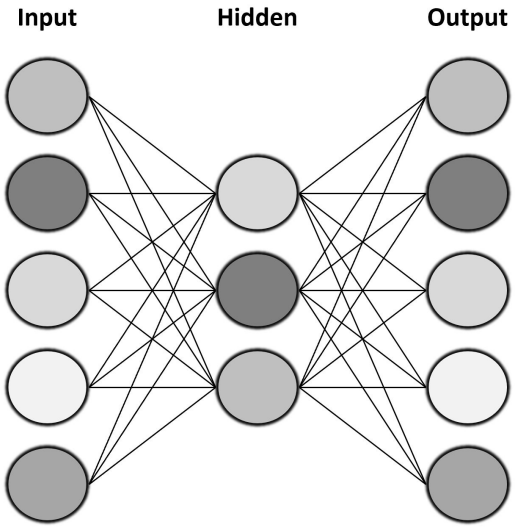


Figure 3.3: Example of a shallow undercomplete autoencoder. To best reconstruct the input at its output, the autoencoder needs to find a good strategy to compress the input information in its smaller hidden layer, learning in this way a new representation of the input data.

distribution for all those neurons. The loss term would then be given by the sum for all neurons of the Kullback-Leibler (KL) divergences between those two variables:

$$J_{sparse} = \sum_{i=1}^{s_{hidden}} D_{KL}(\rho||\hat{\rho}_i) = \sum_{i=1}^{s_{hidden}} \rho \cdot log\frac{\rho}{\hat{\rho}_i} + (1-\rho) \cdot log\frac{1-\rho}{1-\hat{\rho}_i}\,, \tag{3.1}$$

where J_{sparse} is the sparsity loss term, s_{hidden} is the size (number of neurons) of the hidden layer(s), ρ is the desired sparsity (or desired average activity for each neuron), $\hat{\rho}_i$ is the estimated average activity of neuron i, and $D_{KL}(\rho||\hat{\rho}_i)$ is the KL divergence of desired sparsity ρ from estimated average activity $\hat{\rho}_i$.

This method allows the system to learn useful sparse representations of the input data (e.g., edge patterns at different positions and orientations when taking images as input [177]) without the need of other more complex techniques (as would be required, e.g., when using Hebbian learning).

3.3 Methods

In this section, we describe the proposed NILRNN, which is a recurrent neural network for representation learning of temporal data that draws inspiration from models of layers L4 and L2/3 of the primary visual cortex. Further, the NILRNN relies on brain-related concepts such as sparsity, slowness, or self-organization, which, as we saw in Section 3.2.2, often bring computational benefits. On the other hand, to mitigate the issues related to the Hebbian learning methods often used in models of the visual cortex, the NILRNN relies on a self-supervised learning method similar to the one used with autoencoders. We will first present the architecture

of the feature extraction system to then describe the system used for the self-supervised learning of weights.

3.3.1 The Feature Extraction System

The NILRNN feature extraction system is to a large extent inspired by layers L4 and L2/3 of the neocortex, as well as by the convolutional and max pooling layers of CNNs. As we saw in Section 3.2.2, a possible interpretation of why CNNs and their spatial pooling mechanism are so effective is that, in an image, slightly shifted versions of a pattern in a region of the image contribute with very similar meanings and can therefore be grouped together. This idea, however, while inspired by models of the primary visual cortex, does not seem to describe anything occurring in other regions of the neocortex, nor seems to be applicable to most domains other than vision (e.g., shifting the elements of a generic sensory input or feature vector will in general change completely its meaning). As we mentioned in Section 3.2.1, Antolik and Bednar [169] proposed a model of the primary visual cortex relying on a more general principle that may actually also apply to other regions of the neocortex: In this model, what is grouped together is input patterns tending to occur close in time (e.g., shifted edges). Following a similar reasoning to that of CNNs, we could argue that such strategy works because input patterns that tend to occur close in time have associated very similar meanings, something that seems true for most domains dealing with temporal data (and that is strongly related to the concept of slowness, with the output of the L2/3 layer being invariant to fast-varying features and changing at slower timescales than the input). Hence, a neural network that relies on this principle could be seen as a generalization of CNNs for temporal data other than images, making use of a form of (sub-symbolic) semantic pooling that is more general than the spatial pooling of CNNs. The NILRNN has been designed based on this principle, pooling together neurons that tend to fire close in time. Drawing again inspiration from the neocortex, such behavior is achieved through a layer, which is to some extent analogous to L4 and to the convolutional layer, with its neurons arranged in two dimensions and in which neighbor neurons are connected through recurrent connections. We will refer to such layer as locally recurrent layer (note that this is different from the local recurrent connections defined in [178]). This connectivity pattern allows neurons to contribute to the firing of their neighbor neurons in the next timestep. This, added to the fact that, during the training phase, neurons in this layer are encouraged to have sparse activities (acting to some extent as fuzzy input pattern detectors), pushes nearby neurons to learn input patterns that tend to appear successively in the input. Hence, a form of self-organization mechanism emerges where local interactions among neurons lead to a global semantic order in terms of neurons with similar meanings being located close to each other. Then, these nearby neurons with similar meanings can be pooled together through a 2D max pooling layer (analogous to L2/3 and to the max pooling layer of CNNs), achieving the sought semantic pooling, which

allows the system to simplify and reduce the dimensionality of the representation while keeping the relevant information, as well as its ordered distribution in two dimensions.

Coming back to the locally recurrent layer, its neurons keep information on the current and previous inputs (e.g., with input images, a neuron may represent the input pattern as well as the direction and velocity in which it is moving). However, the max pooling layer pools neurons representing those different trajectories together, losing an important part of the recurrent information and behaving to some extent as a feedforward network (e.g., with input images, neurons in the recurrent layer detecting edges with similar orientations tend to be located together, independently of the velocity of the edge). This behavior, while may seem undesirable, can actually be seen as an emergence of the slowness principle and is analogous to what occurs in the SFA algorithm in particular: During the training phase, a set of functions are learned whose outputs change slowly in time while only depending on the input at each instant. On the other hand, the lack of a hidden layer between the input and the locally recurrent layer sets a limitation on the possible features that this layer can learn, being basically a linear feature detector (as neural networks require at least one hidden layer to behave as universal approximators [179]). Therefore, sparse representations at the input, which are generally highly redundant, highly separated, and with the main information relying on whether the neurons are active or inactive rather than on their precise level of activity, are preferred.

In summary, the NILRNN feature extraction system consists of a 2D locally recurrent layer followed by a 2D max pooling layer. The recurrent layer is characterized by its height l_h and width l_w (i.e., it has a size of $s = l_h \cdot l_w$), as well as by the number of neighbor neurons f_r to which each neuron is connected recurrently. This recurrent connectivity pattern (or kernel) takes a circular shape, with only neurons under a specific distance limit being connected to each other (as a consequence, parameter f_r can only take values that correspond to "circular" shapes). Regarding the layer input, different feedforward connectivity patterns are possible depending on the characteristics of the input data, with a fully connected input being the simplest and more general. In this study, we have used a fully connected input for the comparison with other systems and a partially connected input similar to that of the primary visual cortex to evaluate the NILRNN internal behavior (see Section 3.4.3). The max pooling layer, on the other hand, is defined by the number of neurons f_p that each neuron pools from a region of the recurrent layer (i.e., the kernel size) and by a stride t_p. Similarly to the recurrent connection pattern, the pooling kernel takes a circular shape. Since we are working with sparse representations, with the the desired neuron behavior being that of detectors of specific patterns, neurons in the recurrent layer make use of sigmoid activation functions. Figure 3.4 shows this feature extraction system applied over an input in a fully connected manner and within a classification task that applies logistic regression (i.e., a single neural layer) over the extracted features.

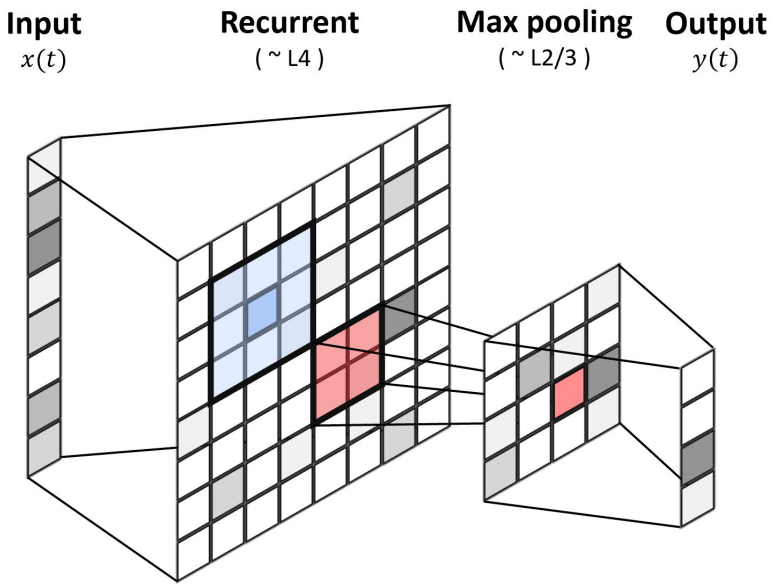


Figure 3.4: The NILRNN feature extraction system architecture (consisting of a 2D locally recurrent layer followed by a 2D max pooling layer) within a classification task and with fully connected input. Neurons in the locally recurrent layer located close to each other fire after patterns that tend to occur close in time. Neurons in the max pooling layer pool those nearby neurons together, gaining invariance against low-level fast-changing features. Blue cells represent locally recurrent connections. Red cells represent max pooling connections.

3.3.2 The Self-Supervised Learning System

While the system just presented can of course be trained in a supervised way through backpropagation, in this study we are interested in the task of learning representations in an unsupervised way (similarly to how the brain learns). Therefore, in this section, we propose a system designed specifically for the self-supervised learning of the NILRNN weights. As we mentioned earlier, the NILRNN relies on the backpropagation technique through a self-supervised approach similar to that used in autoencoders (see Section 3.2.3). In principle, this could be achieved by adding a layer at the end of the system and training the system to reconstruct its input. However, the role of the max pooling layer is precisely to get rid of the fast-changing low-level features (e.g., edge phases), which are necessary for the reconstruction. Since, in addition, this layer has no weights to learn, it is not included in the learning system (i.e., the input is reconstructed directly from the locally recurrent layer). On the other hand, the locally recurrent layer counts on information on the current and previous inputs. Therefore, it makes sense to design the self-supervised system so that, besides reconstructing the current input, it also reconstructs the previous inputs or predicts the next inputs. Preliminary tests, together with the idea that making the system predictive can push the system to learn features that represent the input within its context in a more meaningful way, led us to go for the second option. The influence of each of those predictions over the learning process can be weighted in the loss function (e.g., using

the weighted Euclidean distance when calculating the squared-error loss term). Hence, since counting on information from the past generally helps to achieve lower future sample prediction errors, and as neurons in the locally recurrent layer are only connected to their neighbors, the training process finds solutions for which nearby neurons in the recurrent layer learn patterns that tend to occur close in time, leading to the self-organization mechanism previously mentioned (and to the slowness phenomenon once the max pooling layer is added). Finally, to encourage a sparse activity in the locally recurrent layer, a sparsity term as that of sparse autoencoders is introduced in the loss function (see Section 3.2.3).

In summary, the NILRNN self-supervised learning system is a three-layer neural network as the one shown in Fig. 3.5 formed of an input layer, the same 2D locally recurrent layer as that of the feature extraction system, and an output layer formed of $n_{\hat{x}}$ channels with the size and shape of the input layer (i.e., of the size of the input times $n_{\hat{x}}$), being $n_{\hat{x}}$ the number of reconstructed/predicted input timesteps. Neurons in the output layer also use sigmoid activation functions, which means that, for the input reconstruction to be successful, the input needs to be normalized to the range $(0, 1)$ (which is in line with the preferred sparse representations at the input). The network is trained using truncated backpropagation through time (BPTT), characterized by a truncation horizon k, to minimize the following loss function:

$$J(W, b) = J_{error} + \lambda \cdot J_{regularization} + \beta \cdot J_{sparse}\,, \tag{3.2}$$

where W and b represent all the learnable weights and bias units of the network, J_{error} is the weighted minimum squared error (MSE) loss term, $J_{regularization}$ is the L_2 regularization term (or weight decay), J_{sparse} is the sparsity term as described in (3.1) applied to the locally recurrent layer, and λ and β are the L_2 regularization term and sparsity term weights in the loss function:

$$J_{error} = \frac{1}{2m} \sum_{i=1}^{m} \|\sqrt{w_{\hat{x}}} \circ (h_{W,b}(x_i) - y_i)\|_2^2\,, \tag{3.3}$$

$$J_{regularization} = \frac{1}{2}\|W\|_2^2\,, \tag{3.4}$$

with m being the batch size (number of time samples in the batch or sequence), x_i and y_i the input and target output corresponding to the i-th sample (with y_i being a concatenation of x_i and the next $n_{\hat{x}} - 1$ inputs), $h_{W,b}(x_i)$ the output of the network for input x_i, $w_{\hat{x}}$ the vector of weights controlling the influence of the different predictions (which appears within a square root because in the weighted Euclidean distance the weights multiply the terms being summed, i.e., the squares of the differences), $\sqrt{\cdot}$ the square root of the vector elements, $\circ$ the element-wise or Hadamart product, and $\|\cdot\|_2$ the L2 norm operator (or Euclidean norm, i.e., the square root of the sum of squares of all elements in the vector or matrix). Note that, in this work, by sample we refer to each time sample within a sequence, and not to a full sequence within a dataset, as we are interested in the more complete problem of sample classification rather than on classifying already segmented sequences.

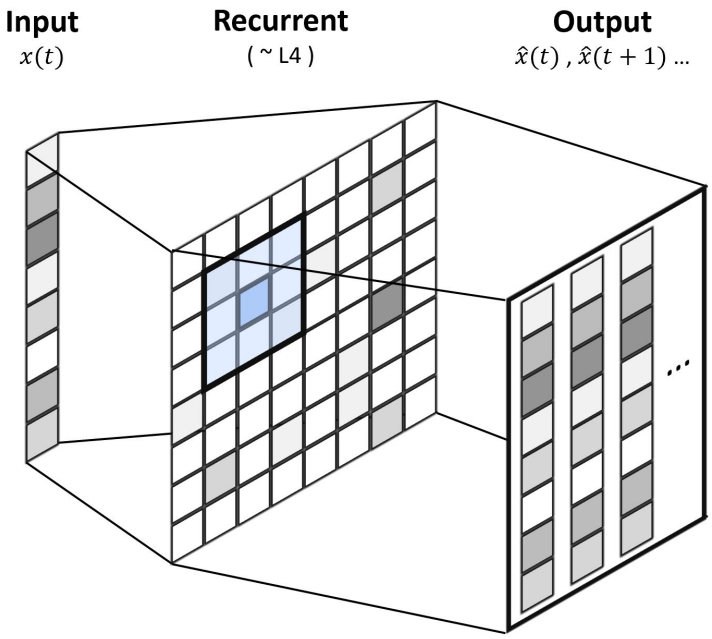


Figure 3.5: The NILRNN self-supervised training system architecture with fully connected input and fully connected output. The system is trained to reconstruct and predict at its output the current and next inputs. Since recurrent connections only exist between nearby neurons of the locally recurrent layer, to make the system able to use past information for prediction, the training leads to nearby neurons learning input patterns that tend to occur close in time. Blue cells represent locally recurrent connections.

3.4 Results

This section compares the NILRNN both in terms of performance with other shallow self-supervised representation learning systems and in terms of internal emerging behavior against known behavior of the primary visual cortex.

3.4.1 Data Inputs Used

To evaluate the performance of the NILRNN, we have opted to test it in two typical domains involving temporal data: action recognition and speech recognition. In particular, we have tested our system over WARD (Wearable Action Recognition Database) [180], containing human action data, and FSDD (Free Spoken Digit Dataset) [181], containing speech data. The WARD dataset consists of 1,267 recordings (summing up to 565,755 time samples) of 20 subjects with inertial sensors in their wrists, waist, and ankles performing 13 different common actions such as standing or walking. The FSDD dataset consists of 3,000 audio recordings (summing up to 126,750 time samples) of 6 subjects saying a number between 0 and 9. This information is summarized in the first columns of Table 3.1. These two datasets have the advantages that they are free popular balanced temporal datasets and that they are adequate to be processed by shallow systems such as the NILRNN (as well as by deep systems, depending on the downstream task).

Table 3.1: Main properties of the data inputs used to evaluate the performance of the NILRNN, as well as specific training parameters used.

	Properties			**Training parameters**
Data input	**sample size**	**# samples**	**# classes**	**# batches sup.**
WARD	25	565,755	13	2,000
FSDD	40	126,750	10	3,000
Synthetic actions	55	$\sim\infty$	4	500

Since speech data is typically processed in the frequency domain, we have used the spectrogram of the FSDD data as input (which is also a form of mimicking our auditory system [182]). This has the added advantage that the spectrogram can be considered to certain extent as a sparse representation, which, as mentioned in Section 3.3.1, is a preferable property of the input when working with the NILRNN (the WARD data comes from inertial sensors and, therefore, does not satisfy this property). The data from the two datasets has been then normalized through truncation to ± 3 standard deviations and through rescaling to the range $[0.1, 0.9]$.

Considering that the size of these two datasets is quite limited, and that, as we argued in Section 3.1, one advantage of learning in a self-supervised way is that the amount of unlabeled data is typically way larger than that of labeled data (allowing the system to learn from much more data and to generalize better), we have also tested our system over an input stream of human action data generated synthetically. This synthetic action input takes a sparse representation form and is generated from a virtual 2D environment that contains a hand and four objects (see Fig. 3.6). The hand can perform four different actions: pick-and-place, push, pull, and wait. At the beginning of the experiment, as well as periodically after a certain number of actions, the environment is restarted, and four new objects are selected randomly with replacement from a set of five different types of objects, which are placed in random positions of the environment. The target positions of the pick-and-place, push, and pull actions are also chosen randomly. The different types of objects have different affordances and identifiers, which are defined at the beginning of the training in a random way. Finally, white noise is added to the position of the hand with respect to the target object so that the "grab" is performed over different object points.

The actual synthetic input is generated from this 2D environment by expressing some of its variables in a sparse (or fuzzy) form (note that we don't want to deal with a computer vision problem). Some of these variables rely on a simple attention mechanism that defines the object of attention based on the trajectory of the hand and the speed of the objects in case they are moving (this also opens the possibility to work with a variable number of objects). These variables are the identifier of the object of attention, the velocity of the hand, the velocity of the object of attention, the position of the object of attention with respect to the hand, and a

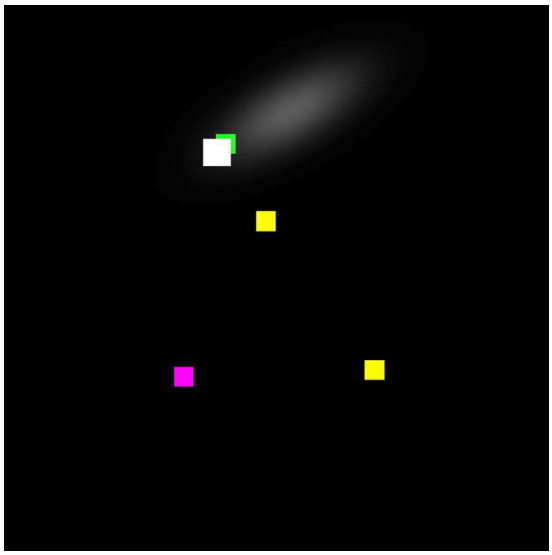


Figure 3.6: Virtual 2D environment used to generate the synthetic action input. The white rectangle represents the hand, and the squares with different colors represent different types of objects with different affordances and identifiers. The white shadow informs about the area of attention deduced from the hand trajectory.

measure of how open or closed the hand is. The identifier of each type of object is defined sparse and of size 10. The positions and velocities are expressed through the location of a Gaussian over a 2D grid (5×5 for position and 3×3 for velocity). The location of the Gaussian is determined by the magnitude and direction of the position or velocity vector after going through a non-linear mapping that assigns more resolution to smaller magnitudes, with the center of the grid corresponding to position $(0, 0)$. To express how open or closed the hand is, a vector of size 2 is used. This leads to a total input sample size of 55. Finally, to make the task more challenging, white noise is added to the input. The first columns of Table 3.1 summarize the main properties of this input data.

Finally, for the comparison of the NILRNN internal behavior against that of the primary visual cortex, the system has been trained using sequences of image patches of size 16×16 obtained by moving laterally a 16×16 window along an image at random velocities and directions, with the images used being whitened natural images [183] truncated to ± 3 standard deviations and rescaled to the interval $[0.1, 0.9]$.

3.4.2 Comparison with Other Systems

The performance of the NILRNN has been evaluated attending to its feature extraction ability for the task of classification. We have used NILRNN configurations similar to those presented in Figs. 3.4 and 3.5, i.e., fully connected input, shallow systems (without building hierarchies), and self-supervised learning of representations (with fully connected output). The datasets have been divided, using stratification, into a training set ($\sim$60% of the samples), a validation

set (for model selection, $\sim$20% of the samples), and a test set (for evaluation, $\sim$20% of the samples). Considering that we are working with temporal data, the classification task has been performed using both a logistic regression and a fully connected shallow vanilla recurrent neural network (RNN), i.e., a fully connected recurrent layer followed by a fully connected feedforward layer. Regarding the metrics for the evaluation of the classification performance, we have opted for accuracy, as it is a valid representative metric and one of the preferred ones when it comes to balanced datasets [184]. For the sake of comparison, the two classifiers have been applied over the features extracted by the NILRNN and by other shallow self-supervised representation learning systems, as well as over the input directly. In particular, we have compared the performance of our system with that of an undercomplete autoencoder and of a sparse autoencoder [94], as well as with that of an alteration of the NILRNN without the max pooling layer (to better understand how this layer contributes to the overall performance). All these neural networks have been initialized according to Xavier initialization and have been trained through backpropagation using the Adam optimization algorithm with stepsize α. A genetic algorithm has been used (together with the training and validation sets, randomly split with stratification in each iteration) to find the best hyperparameters for each system and dataset. Still, to simplify the model selection process, the values of few hyperparameters have been set before the application of the genetic algorithm: The batch size has been set to $m = 1000$ samples, the 2D layers have been set to a squared shape (i.e., equal height and width), and the prediction weights in the loss function have been set to 1 (i.e., $w_{\hat{x}} = 1$, all predictions have the same weight). The fitness function of the genetic algorithm has been obtained as the estimated accuracy of the network over 100000 samples after having trained the representation learning system (if it exists) over 1000 batches (10000 batches for FSDD) and the classification system over a number of batches that is different depending on the dataset and that is indicated in the last column of Table 3.1 (these values have been defined manually according to the accuracies achieved for the different datasets and number of batches). The obtained hyperparameters for the different systems and datasets are shown in Tables 3.2, 3.3, and 3.4.

With the hyperparameters defined, the systems have been evaluated in terms of accuracy (over the test set) after different numbers of classifier training batches (from the combined training and evaluation sets) to understand how fast they can learn as well as how good they can get. For each system, and with the representation learning component already trained over 5000 batches (10000 batches for FSDD), the classification accuracy has been measured after every 50 batches of classifier training until reaching 5000 batches. This process has been repeated 32 times per system, and the obtained results averaged, to reduce the influence of the variance that corresponds to different executions. These averaged results for the different systems and datasets are shown in Figs. 3.7, 3.8, and 3.9.

The obtained results show that, in general, the representations learned by both the NILRNN

Table 3.2: Hyperparameters for the different systems applied over the WARD dataset, where s is the size of the hidden layer, k the BPTT truncation horizon, f_r the size of the locally recurrent layer kernel, f_p the size of the max pooling layer kernel, t_p the stride of the max pooling layer, $n_{\hat{x}}$ the number of reconstructed/predicted inputs, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and ρ the desired sparsity in the hidden layer.

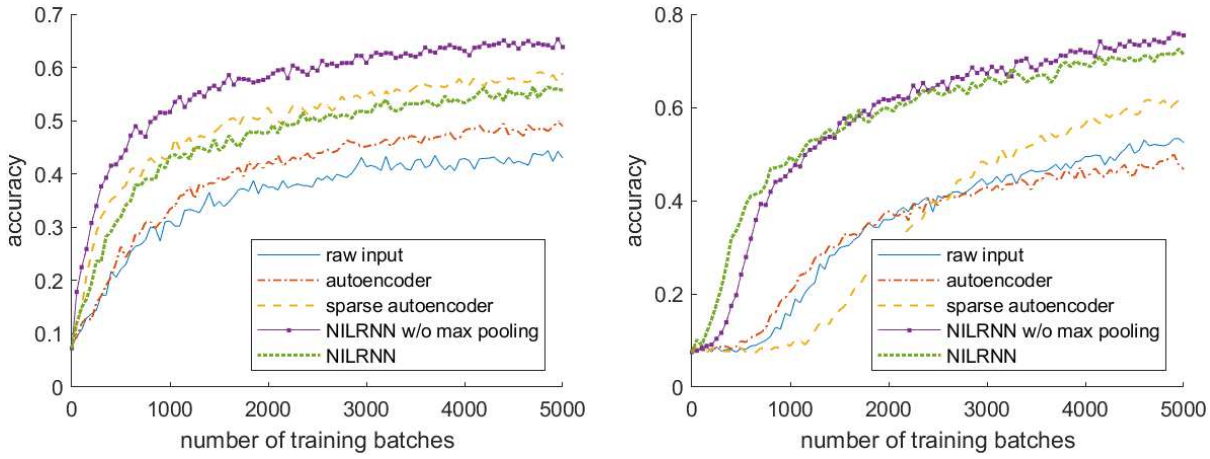
	Representation learning system										**Classification system**			
System	s	k	f_r	f_p	t_p	$n_{\hat{x}}$	α	λ	β	ρ	s	k	α	λ
Log. reg. over:														
Raw input	-	-	-	-	-	-	-	-	-	-	-	-	1.7e-2	7.29e-6
Autoenc.	208	-	-	-	-	-	3.66e-2	2.75e-69	-	-	-	-	1.28e-2	1.22e-5
Sparse autoenc.	919	-	-	-	-	-	8.35e-2	7.84e-31	4.5e-5	2.08e-2	-	-	3.66e-2	6.68e-7
NILRNN w/o pool.	529	32	69	-	-	8	1.31e-2	8.53e-66	3.44e-7	1.73e-9	-	-	1.15e-2	4.28e-6
NILRNN	529	17	49	61	3	9	1.24e-2	6.38e-34	4.44e-4	2.43e-5	-	-	1.42e-2	6.02e-7
RNN over:														
Raw input	-	-	-	-	-	-	-	-	-	-	94	7	9.02e-3	2.79e-6
Autoenc.	201	-	-	-	-	-	6.13e-2	8.82e-121	-	-	88	10	1.08e-2	4.39e-6
Sparse autoenc.	1099	-	-	-	-	-	7.63e-2	3.93e-106	3.05e-5	1.55e-2	137	14	1.05e-2	1.73e-8
NILRNN w/o pool.	256	23	45	-	-	11	1e-2	2.38e-66	1.57e-5	4.85e-38	161	13	5.14e-3	2.89e-7
NILRNN	841	15	69	45	2	8	1.07e-2	1.83e-38	1.18e-5	2.75e-121	109	9	2.91e-3	1.54e-6

Table 3.3: Hyperparameters for the different systems applied over the FSDD dataset, where s is the size of the hidden layer, k the BPTT truncation horizon, f_r the size of the locally recurrent layer kernel, f_p the size of the max pooling layer kernel, t_p the stride of the max pooling layer, $n_{\hat{x}}$ the number of reconstructed/predicted inputs, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and ρ the desired sparsity in the hidden layer.

	Representation learning system										**Classification system**			
System	s	k	f_r	f_p	t_p	$n_{\hat{x}}$	α	λ	β	ρ	s	k	α	λ
Log. reg. over:														
Raw input	-	-	-	-	-	-	-	-	-	-	-	-	2.33e-2	9.04e-6
Autoenc.	578	-	-	-	-	-	6.28e-2	1.06e-6	-	-	-	-	3.46e-2	3.48e-7
Sparse autoenc.	572	-	-	-	-	-	5.67e-2	1.09e-6	2.95e-33	7.93e-4	-	-	5.37e-2	1.14e-6
NILRNN w/o pool.	361	11	37	-	-	15	8.11e-3	2.57e-6	1.95e-23	7.69e-4	-	-	1.81e-2	1.75e-9
NILRNN	1024	10	45	37	3	20	6.93e-3	9.26e-7	5.44e-14	9.01e-4	-	-	1.56e-2	1.1e-5
RNN over:														
Raw input	-	-	-	-	-	-	-	-	-	-	116	6	7.69e-3	9.64e-8
Autoenc.	531	-	-	-	-	-	6.73e-2	9.71e-7	-	-	85	8	2e-2	2.93e-6
Sparse autoenc.	583	-	-	-	-	-	6.54e-2	9e-7	1.24e-125	1.14e-3	72	11	1.43e-2	4.55e-7
NILRNN w/o pool.	400	8	45	-	-	19	7.37e-3	1.97e-6	1.12e-5	8.02e-4	88	7	2.02e-2	1.86e-7
NILRNN	729	12	37	37	3	17	8.36e-3	1.24e-6	4.87e-55	8.5e-4	114	6	1.04e-2	5.62e-9

Table 3.4: Hyperparameters for the different systems applied over the synthetic action input, where s is the size of the hidden layer, k the BPTT truncation horizon, f_r the size of the locally recurrent layer kernel, f_p the size of the max pooling layer kernel, t_p the stride of the max pooling layer, $n_{\hat{x}}$ the number of reconstructed/predicted inputs, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and ρ the desired sparsity in the hidden layer.

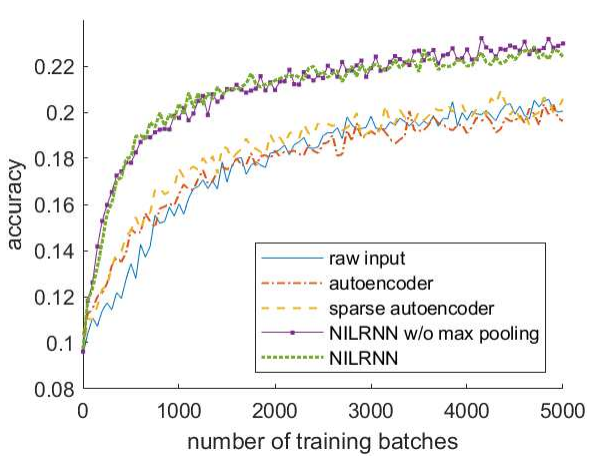
	Representation learning system										**Classification system**			
System	s	k	f_r	f_p	t_p	$n_{\hat{x}}$	α	λ	β	ρ	s	k	α	λ
Log. reg. over:														
Raw input	-	-	-	-	-	-	-	-	-	-	-	-	3.65e-2	2.35e-6
Autoenc.	1393	-	-	-	-	-	5.09e-2	2.08e-11	-	-	-	-	2.33e-3	1.09e-6
Sparse autoenc.	839	-	-	-	-	-	5.36e-2	1.12e-49	8.55e-6	4.08e-17	-	-	6.39e-2	1.45e-6
NILRNN w/o pool.	729	16	61	-	-	13	6.83e-3	1.92e-29	1.19e-3	1.3e-48	-	-	1.37e-2	5.38e-5
NILRNN	729	8	69	9	1	7	8.28e-3	2.28e-164	2.07e-3	1.74e-6	-	-	5.67e-3	1.17e-5
RNN over:														
Raw input	-	-	-	-	-	-	-	-	-	-	98	10	1.62e-2	1.71e-6
Autoenc.	1663	-	-	-	-	-	4.93e-2	2.23e-53	-	-	173	7	8.46e-4	5.5e-8
Sparse autoenc.	1403	-	-	-	-	-	3.83e-2	1.78e-122	1.09e-5	3.1e-63	93	8	1.72e-2	1.02e-7
NILRNN w/o pool.	1225	12	49	-	-	7	8.59e-3	5.09e-37	3.54e-4	1.47e-17	123	12	1.22e-2	3.73e-9
NILRNN	1089	18	57	9	3	8	7.9e-3	1.65e-311	3.26e-2	2.16e-105	68	7	1.92e-2	7.71e-7



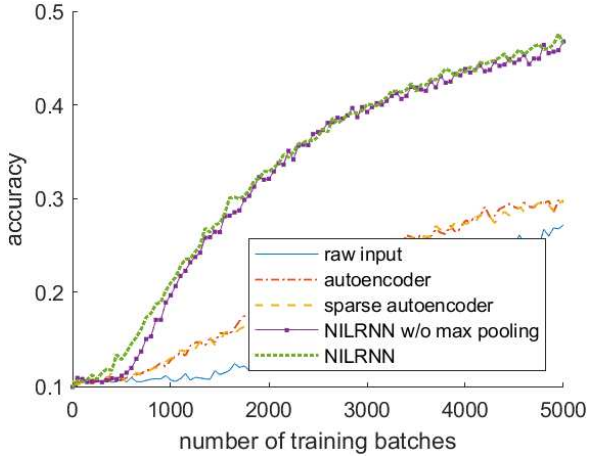
(a) Accuracies estimated using a logistic regression classifier.

(b) Accuracies estimated using an RNN classifier.

Figure 3.7: Accuracies estimated for the WARD dataset and for different systems and number of training batches. With the logistic regression classifier, the NILRNN without the max pooling layer outperforms all other systems, probably thanks to its memory. With the RNN classifier, the NILRNNs with and without max pooling layer outperform all other systems. The NILRNN with max pooling layer has a better performance for a small number of training samples, but performs slightly worse for more samples, which may be related to the fact that the input data does not take a sparse representation form.

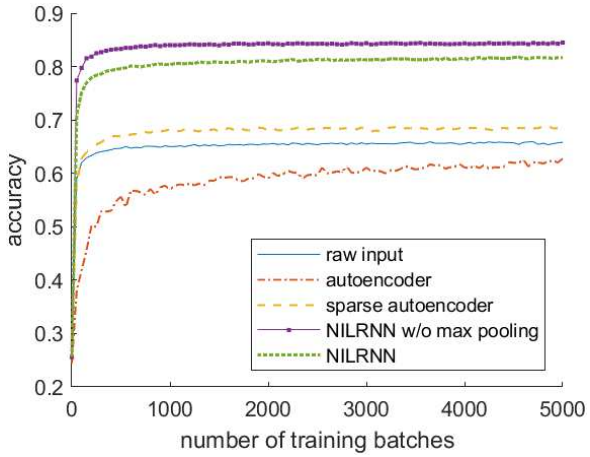


(a) Accuracies estimated using a logistic regression classifier.

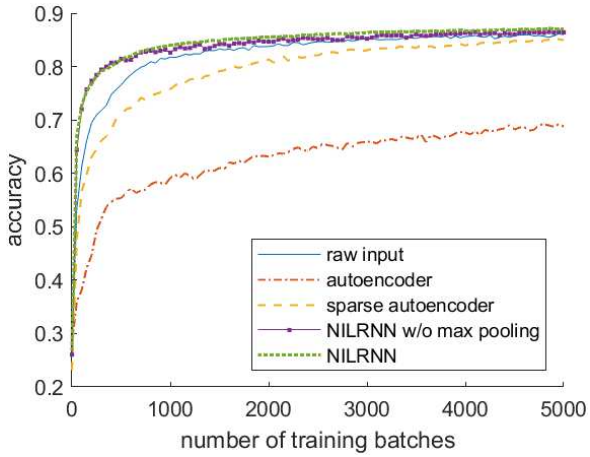


(b) Accuracies estimated using an RNN classifier.

Figure 3.8: Accuracies estimated for the FSDD dataset and for different systems and number of training batches. With both the logistic regression and the RNN classifiers, the NILRNNs with and without max pooling layer outperform all other systems. With the RNN classifier, the NILRNN with max pooling layer performs slightly better than the one without.



(a) Accuracies estimated using a logistic regression classifier.



(b) Accuracies estimated using an RNN classifier.

Figure 3.9: Accuracies estimated for the synthetic action input and for different systems and number of training batches. With the logistic regression classifier, the NILRNN without the max pooling layer outperforms all other systems, probably thanks to its memory. With the RNN classifier, the NILRNNs with and without max pooling layer outperform all other systems especially for a small number of training samples, with the NILRNN with max pooling layer performing slightly better.

with and without max pooling layer outperform the representations learned by other systems and the raw input in terms of accuracies achieved as well as in terms of the number of samples required to achieve specific accuracies. In principle, this could be associated to the fact that the NILRNN is a recurrent network and, therefore, contains also information about previous samples. However, as we mentioned in Section 3.3.1, the max pooling layer of the NILRNN makes it behave to some extent as a feedforward network. This is coherent with the fact that the NILRNN without the max pooling layer is in general outperforming the complete NILRNN when using a logistic regression classifier (Figs. 3.7a, 3.8a, and 3.9a), but this is not true anymore when using an RNN classifier (Figs. 3.7b, 3.8b, and 3.9b). Indeed, the logistic regression system has no recurrent connections, making the memory of the NILRNN without max pooling a clear advantage. However, since the RNN already has memory by itself, using representations that keep information about the previous samples may not be that relevant, and having the best possible representations describing the current input (which is what we are looking for) may be more important. This should also apply to the other representation learning systems, which, even when using an RNN classifier, achieve lower performances. In fact, Fig. 3.9b shows how most of the other systems, when using an RNN classifier, are able to achieve similar accuracies after a large enough number of training samples, indicating that they do have the capability to achieve such accuracies. This is not the case for Figs. 3.7b and 3.8b, but it may also occur for a larger number of training samples than those shown.

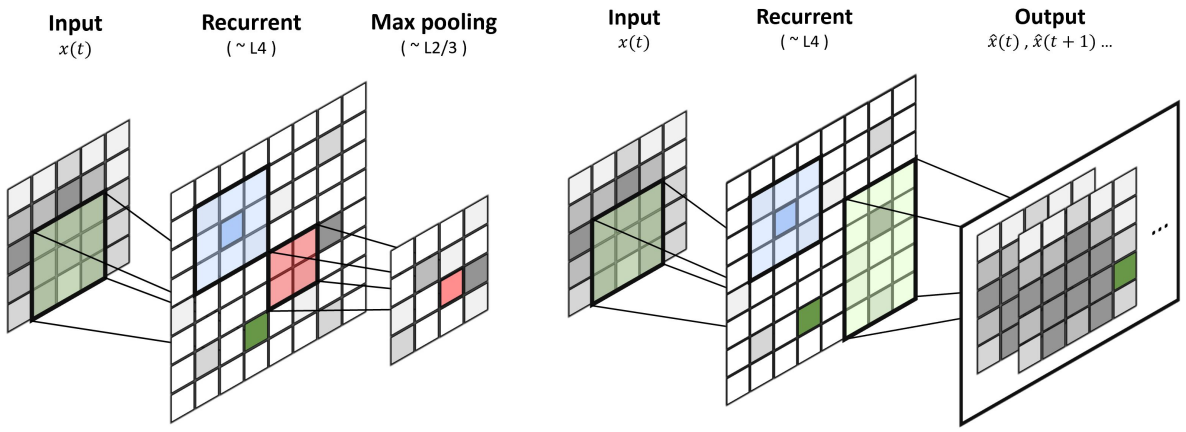
Focusing on the differences between the NILRNN with and without max pooling layer when using an RNN classifier, Figs. 3.7b and 3.8b show that, for the WARD and FSDD datasets, the full NILRNN is allowing a faster learning for a small enough number of samples, which is one of the main objectives of this system (for the synthetic action input, shown in Fig. 3.9b, both curves increase so fast that there is no perceptible difference). Regarding the accuracies reached after a large enough number of training samples, the complete NILRNN is slightly outperforming the NILRNN without max pooling on the FSDD dataset and on the synthetic action input (Figs. 3.8b and 3.9b). However, this is not the case for the WARD dataset (Fig. 3.7b), for which, after certain number of samples, the NILRNN without max pooling begins to outperform the complete version. This may be related to the fact that the input data does not take a sparse form, and, therefore, the NILRNN does not behave as desired. In any case, these small differences between the two systems, added to the fact that the number of batches in the fitness function has been chosen to some extent in an arbitrary way, indicate that further investigation is needed to understand better the behavior of the max pooling layer within the NILRNN. Still, the results obtained are already promising.

3.4.3 Comparison Against the Primary Visual Cortex

While the NILRNN has been designed with the main purpose of representation learning, its structure and desired functioning is highly inspired by models of the feedforward circuits in the neocortex. Thus, evaluating if it shows an analogous behavior to that of the primary visual cortex can help us understand whether it is behaving as desired. In addition, such analogous behavior would validate our system as a potential computational model of the primary visual cortex (and, considering their similar structure and functioning, maybe also of other areas of the neocortex processing other types of data). Computational models are used within the field of cognitive neuroscience to understand how different regions of the brain work and how different forms of cognition emerge, having the advantage over more descriptive or conceptual models that they allow us to simulate the brain region(s) and test different hypotheses, as they provide full control over and access to the parameters and state variables of the model and to their influence on its overall behavior. While often designed specifically to serve this purpose, there are also brain-inspired machine learning systems that, since they show analogies with regions of the brain in terms of behavior, have been useful in the advancement of our knowledge about their functioning. One example is CNNs, which, considering that they were inspired by the structure and connections of our visual cortex and that, due to their great success, they have been extensively and deeply studied, have been relevant sources of understanding our visual system [185].

In this context, and considering that the NILRNN is a more accurate model of the visual cortex than CNNs in terms of structure, this section studies the emerging behavior of the NILRNN when it takes shifting images as input and compares it against known behavior of the primary visual cortex. Since we work with images as input, we have made the input to the locally recurrent layer partially connected, following a connection pattern that mimics the one found in the primary visual cortex: Neurons in the recurrent layer are connected to a region of the input (i.e., their receptive field, see Section 3.2.1) in a way that, when moving along contiguous neurons in the layer, the corresponding receptive fields shift smoothly in the same direction, similarly to the connection patterns of convolutional layers. Therefore, neighbor neurons have same or very similar receptive fields (see Fig. 3.10a). This leads to an input connectivity pattern characterized by a circular receptive field (or kernel) size of f_x neurons and by a stride t_x (which can be lower than 1, indicating that multiple neighbor neurons share the same receptive field). Regarding the training output layer, each of its channels are connected to the recurrent layer following a pattern that is symmetric to that defining the connections between the input layer and the recurrent layer (see Fig. 3.10b).

To evaluate how the NILRNN has an analogous behavior to that of the neocortex, we have used a network that takes as input an image patch of size 16×16 and has a locally recurrent layer formed of neurons with a receptive field of size $f_x = 69$ pixels and connected in a recurrent



(a) Feature extraction configuration.

(b) Self-supervised training configuration.

Figure 3.10: The NILRNN architecture with partially-connected input and training output. The input and training output connectivity patterns are symmetric. Green cells represent input and output feedforward connections. Blue cells represent locally recurrent connections. Red cells represent max pooling connections.

way to $f_r = 29$ neurons of the same layer. Their specific receptive field is defined by a stride of $t_x = 0.33$ neurons (i.e., every three neurons, the receptive field shifts one pixel), leading to a layer size of 46×46 (i.e., $l_h = l_w = 46$). The max pooling layer is defined by a kernel size of $f_p = 21$ neurons and by a stride of $t_p = 1$ neuron, having therefore also a size of 46×46. Regarding the self-supervised learning system, its output is formed of $n_{\hat{x}} = 3$ channels (i.e., it has a size of $16 \times 16 \times 3$, reconstructing the current and next two inputs). The loss function is characterized by weights $\lambda = 1.5 \cdot 10^{-6}$ and $\beta = 0.15$ and by a desired sparsity parameter of $\rho = 0.04$. For the training, we have used truncated BPTT with a truncation horizon of $k = 4$ timesteps, Adam optimization with a stepsize of $\alpha = 2.5 \cdot 10^{-3}$, and a batch size of $m = 1000$ samples, and we have trained the system on 400.000 batches. Note that all these hyperparameters have been set manually, and, therefore, better results may be obtained using other values. The training has been carried out using sequences of shifting image patches as input, as described in Section 3.4.1.

Once the system has been trained, we have set images of drifting sinusoidal gratings with different phases, orientations, and spatial frequencies as those shown in Fig. 3.1 as input, and we have analyzed the responses of the neurons in both the locally recurrent and max pooling layers, similarly to how is done when studying the behavior of the primary visual cortex [186] or of models of it [169]. Fig. 3.11 shows the resultant weights from the training at the input feedforward connections, once normalized (i.e., the input patterns that the neurons at the recurrent layer have learned to detect). This figure shows that, as expected, the neurons learn to detect edges in the input, with neighbor neurons tending to detect edges with similar orientations but different phases. This can also be observed in the **orientation** and **phase maps**

of the recurrent and max pooling layers shown in Fig. 3.12. These maps are obtained by finding, for each neuron, the orientation and phase of the input pattern that draws the maximum response, for any spatial frequency. As can be seen in this figure, the orientation map for the max pooling layer looks similar and has similar characteristics to those typically obtained from the primary visual cortex (i.e., it has homogeneous regions appearing periodically, pinwheels where many different orientations meet, etc., see Fig. 3.2a) [186]. The orientation map of the recurrent layer has similar properties but with the regions being more scattered, which is something that also occurs in other models of the primary visual cortex and that, to the best of our knowledge, does not contradict any experimental evidence [169]. In addition, and as expected, regions at similar positions of the two layers match quite well in terms of orientation preferences. Regarding the phase maps, that of the recurrent layer does not appear to have any order, which is consistent with experimental evidence. On the other hand, some homogeneous regions appear at the phase map of the max pooling layer, but these regions are in general smaller than those at the orientation map and have the shape of the max pooling kernel (which is a 21 neuron kernel with the shape of a 5×5 square without its 4 corners). Indeed, they seem to appear simply because the maximum value for a neuron in the max pooling layer corresponds to the maximum of all the maximum values of the neurons it is pooling, so neurons in the recurrent layer having high maximum values will lead most neurons pooling them to have those same maximum values for the same input patterns (i.e., same phases). Other than that, it does not seem to exist any phase order.

In order to evaluate whether the neurons in both layers behave as simple or complex cells, we have calculated their **modulation ratios** [187]. The modulation ratio of a neuron is calculated as the ratio between the first harmonic and the average of the response to a drifting sine of the spatial frequency and orientation for which the maximum response is obtained. Neurons with a more simple-like behavior respond in a strong way to a particular phase with respect to the others and, therefore, have a higher modulation ratio, while those with a more complex-like behavior respond more homogeneously and, therefore, have a lower modulation ratio. Typically, neurons are classified as complex cells if their modulation ratio is below 1 and as simple cells otherwise. Fig. 3.13 shows the histograms with the modulation ratios of all the neurons of each layer of the NILRNN, as well as a typical distribution obtained when taking a sample of neurons from the primary visual cortex across layers. In this figure we can see how, as expected, most neurons in the locally recurrent layer behave as simple cells, while most neurons in the max pooling layer behave as complex cells, analogously to what has been observed in the primary visual cortex (see Section 3.2.1). In addition, if we consider the neurons of both layers of our system altogether, the shape of the resultant histogram would be very similar to that obtained for the neurons in the primary visual cortex (except for the relative height of the two peaks, which depends on the size of the sample of neurons at each layer and, therefore, should not be considered as a relevant difference).

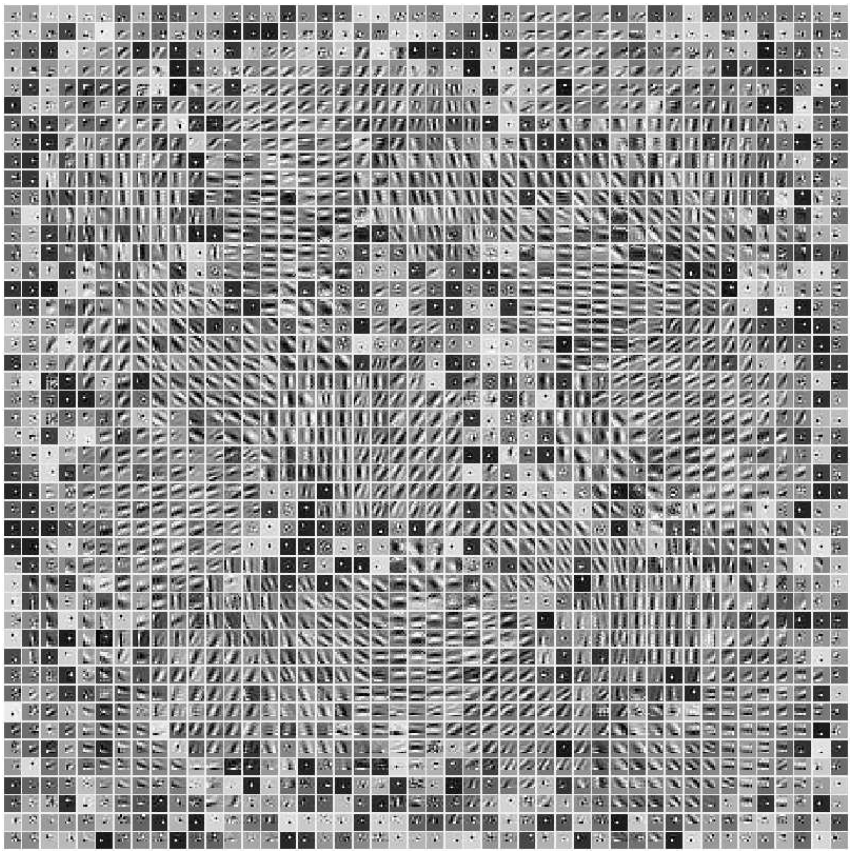


Figure 3.11: Normalized weights at the input feedforward connections of an NILRNN trained with shifting images as input. Neurons learn to detect edges at their receptive field and distribute forming regions of neurons that detect edges with similar orientations but different phases.

Finally, Fig. 3.14 shows the **orientation tuning curves** and **phase responses** of some representative neurons from the inside of the orientation-wise homogeneous regions of both layers of the NILRNN. The orientation tuning curves show the maximum response value of each neuron as a function of the orientation for any phase and spatial frequency. The phase responses show the response obtained as a function of the phase for the orientation and spatial frequency that give the strongest response. The orientation tuning curves show that all these neurons are indeed finely tuned to a narrow band of orientations. As expected, and analogously to what has been observed in the primary visual cortex [187], the orientation bands of the neurons in the max pooling layer are broader than those of the neurons in the locally recurrent layer. Regarding the phase responses, the figure shows how the neurons in the recurrent layer are also finely tuned to a narrow band of phases, while the neurons in the max pooling layer are much more phase invariant, which corresponds to the behavior of simple and complex cells, respectively, as commented in Section 3.2.1.

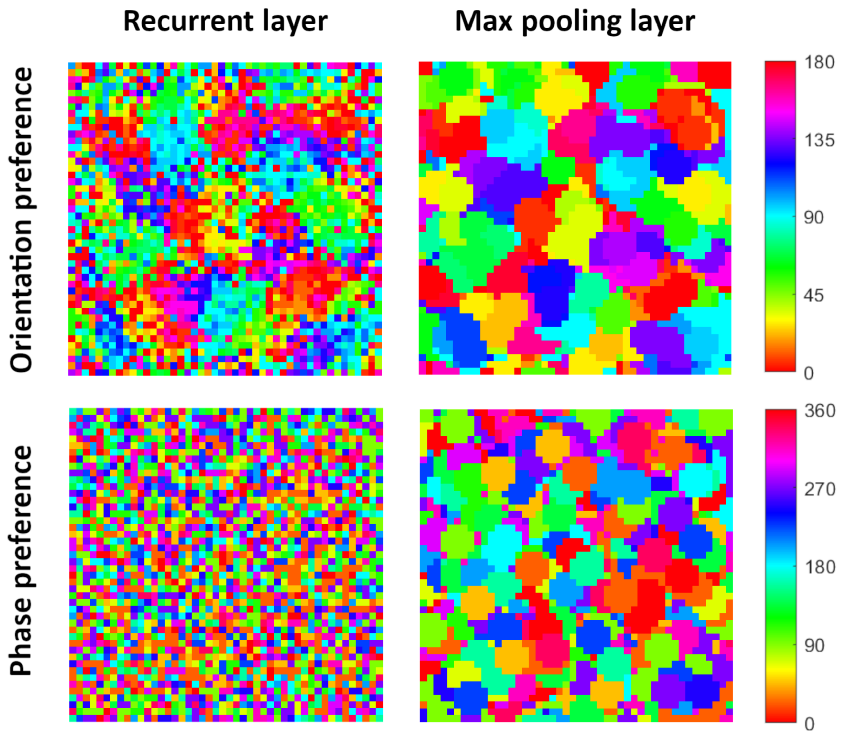
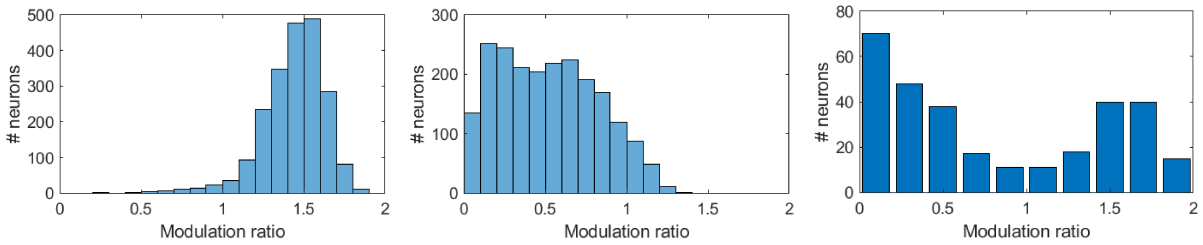


Figure 3.12: Orientation and phase maps at the locally recurrent and max pooling layers of an NILRNN trained with shifting images as input. Both orientation maps show properties similar to those obtained from the primary visual cortex (e.g., homogenous regions, periodicity, etc), with the position of those regions roughly matching in the two layers. Regarding the phase maps, they don't seem to show any order oder than small homogeneous regions in the max pooling layer with the shape of its kernel (consequence of how max pooling works).

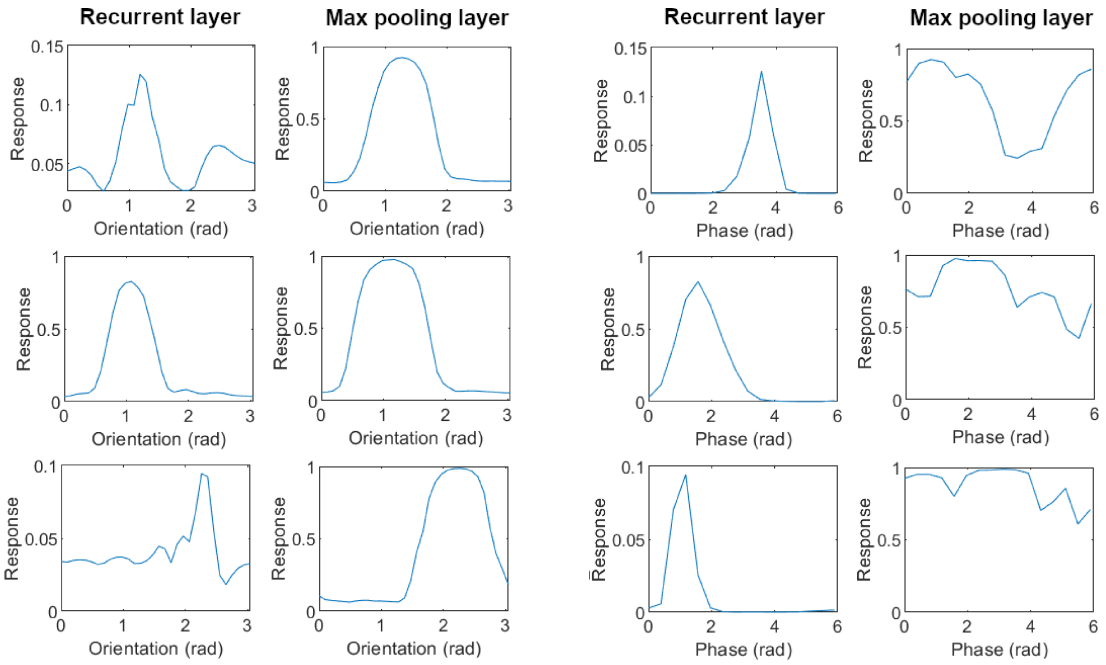


(a) Modulation ratios of the locally recurrent layer of the NILRNN.

(b) Modulation ratios of the max pooling layer of the NILRNN.

(c) Modulation ratios of the primary visual cortex of a macaque monkey [187].

Figure 3.13: Modulation ratios of the neurons of each layer of an NILRNN trained with shifting images as input and of a sample of neurons from several layers of the primary visual cortex of a macaque monkey. Most neurons in the locally recurrent layer behave as simple cells (modulation ratio above 1), while most neurons in the max pooling layer behave as complex cells (modulation ratio below 1). The combination of both histograms would lead to the typical bimodal shape of histograms obtained from the primary visual cortex.



(a) Orientation tuning curves.

(b) Phase responses.

Figure 3.14: Orientation tuning curves and phase responses of three representative neurons of each layer of an NILRNN trained with shifting images as input. Neurons in the locally recurrent layer are tuned to both specific orientations and phases, while neurons in the max pooling layer are tuned to (the same) specific orientations but are more phase invariant. The positions of the three neurons in both layers are (28,10) (top), (23,38) (middle), and (21,13) (bottom).

3.5 Discussion

In this chapter, we have presented the neocortex-inspired locally recurrent neural network (NILRNN): a self-supervised representation learning system for temporal data that brings ideas from the neocortex to the well-established fields of machine learning and artificial neural networks. In particular, this system tries to keep the learning and representation capabilities of models of the neocortex while simplifying their computational complexity and cost to make them more adequate for real-life machine learning applications. It draws inspiration from computational neuroscience concepts such as sparsity, self-organization, and slowness, as well as from other machine learning systems such as CNNs and autoencoders, and relies on a novel form of low-level semantic pooling that is a generalization of the usual spatial pooling. The results show that, in addition to showing an analogous emerging behavior to that of the neocortex, the NILRNN outperforms other shallow self-supervised learning systems such as undercomplete autoencoders or sparse autoencoders in the task of representation learning for classification. In this sense, the NILRNN is successful at fulfilling its purpose of learning representations that allow classifiers to learn their task more efficiently and in less iterations. In fact, the NILRNN is designed to learn from a relatively large amount of unlabeled data, so the results obtained may even be better

for situations where the amount of data is not as limited as in the speech and action datasets we have used. A larger number of batches during the self-supervised learning phase may also lead to better results (which has not been done due to computational and time constraints). On the other hand, instead of working with the average of the performances obtained from the different representations learned in different iterations, a real application can work with the best learned representations, leading to better results. In addition, once the classification system has been trained, the whole system (feature extraction + classification) can be trained end-to-end (unfreezing the representation learning system weights) to fine-tune the weights for the specific application and improve the accuracy. Finally, as we mentioned in Section 3.4.2, the fact that the data from the WARD dataset does not take a sparse representation form may be depriving the NILRNN from working at its best. When working with such type of data, transforming it into some sparse representation (or adding a new hidden layer that can automatically learn such sparse representation) would probably improve the behavior of the system. This needs to be further investigated.

3.5.1 Comparison with Other Systems

One way to better understand conceptually our system and to possibly unveil future directions is to compare it with other existing systems. In Section 3.3.1, we already went through some similarities between the NILRNN and CNNs, and we mentioned that our system can be seen as some sort of generalization of CNNs (and, in particular, of the block composed of a convolutional layer followed by a max pooling layer) to sequences of data other than images, with its form of semantic pooling being a generalization of the spatial pooling in CNNs. One important difference between CNNs and our system is that, in CNNs, the convolutional layer is composed of different channels, with each channel containing the output of one learned feature detector applied along the whole image. In the NILRNN, on the other hand, the recurrent layer has one single "channel", with all those channels of the CNN convolutional layer being represented in a single 2D layer while still being organized locally (being hence more analogous to L4). Regarding the NILRNN input connectivity, in this chapter, we have worked with both fully connected input (for generic data) and partially connected input (for images), with the latter one having a similar connectivity pattern to that of CNNs or of receptive fields in the primary visual cortex. This configuration can in fact also be applied to types of sequences of data (besides sequences of images) whose samples contain some form of topological information, with the elements in those samples being especially correlated to their neighbors (e.g., brain signals, spectrograms, etc.). On the other hand, CNNs assume that the statistical characteristics of images are homogeneous and independent of the position in the image, and, therefore, they learn the same features along the whole image (and apply what is learned in one region of the image to the other regions), which brings several computational and learning advantages. Our

system could also be adapted to take advantage of this homogeneity by, for example, making the input weights shared periodically along the layer in the two dimensions (e.g., having a hexagon of neurons whose weights are repeated periodically). In fact, as mentioned in Section 3.2.1, such periodicity has also been observed in the primary visual cortex [188]. This could be useful for topological input data that is homogeneous along its extension but in which input patterns don't necessarily tend to shift laterally (or shifted patterns don't imply similar information) as occurs in sequences of images (if they actually do, CNNs are probably a better option). In fact, while simple features of images (such as edges) usually behave in this way, this is not generally true for more complex features, such as the presence or absence of an object. Indeed, in images that are close in time, such object may not only shift laterally but also rotate or even change its shape (e.g., a closing hand). CNNs trained in a supervised way are probably able to create associations among those different appearances of the same object thanks to the large amounts of labeled data, but they have more trouble deducing those associations in a self-supervised way, which may be a reason why convolutional autoencoders are not that good at learning high-level features [185]. Taking this into account, the NILRNN may also bring advantages to the processing of sequences of images in the higher layers of the network.

When comparing the NILRNN with complete CNNs (or to other deep architectures), one advantage of these architectures is that they stack blocks of layers to build deep hierarchies and extract more abstract or complex representations. This is something that can also be done with our proposed system by stacking blocks containing the recurrent layer followed by the max pooling layer to build deeper supervised or self-supervised networks. The resultant system, while composed of many layers, can still be trained in a shallow self-supervised way, by training the blocks one by one, learning at each step more abstract representations (mimicking again somehow models of the neocortex [189]). Then, training successfully a classifier over those higher-level representations may just require a small number of labeled samples. In addition, this hierarchy could be useful to better investigate our system and understand if it is able to learn useful representations at different levels of abstraction or if, on the contrary, it is losing relevant information at each level. On the other hand, since, independently of the input data, the output of these blocks is always distributed in two dimensions and with neighbor neurons tending to fire close in time, making the input to the next block partially connected (as described before) may lead to good results while reducing the complexity of the network. One characteristic of this new hierarchical system would be that the activity in the layers changes slower in time as one goes higher in the hierarchy, achieving a hierarchy of timescales similar to that observed in the brain [190] and in other brain-inspired systems such as hierarchical SFA or MTRNNs (see Section 3.2.2), with the difference that, in our system, it would emerge spontaneously, without the need to impose explicit restrictions. Our system could take advantage of this behavior to reduce its computational cost by working at lower sample rates as it goes higher in the hierarchy.

Chapter 4 develops this idea of hierarchy of stacked NILRNNs working at different timescales.

Coming back to our shallow system, and as we already suggested in Section 3.3.1, the NILRNN has several similarities with SFA: Both systems learn representations of the input that change slower in time than the input. In addition, to some extent, and similar to SFA, while our system is recurrent, its output depends mainly on its input at that instant. Thus, we could say that both SFA and NILRNN learn in a "recurrent" way, considering the current sample as well as the previous ones (or next ones), while they then operate in a "feedforward" way, considering only the current sample. In fact, we could interpret that CNNs also behave in a similar way: While it is true that they are feedforward networks (both during learning and prediction), we could consider that CNNs have this temporal knowledge about the input "hardwired" by design (since the grouping of shifted versions of the same pattern can be learned from sequences of images). Taking all this into consideration, we could describe the NILRNN as a feature extractor for temporal data that learns slow-changing representations of the current input while also keeping internally information about previous samples.

It is also worth comparing the NILRNN with hierarchical temporal memory (HTM) [121]. HTM is a machine learning system that is also highly inspired by our knowledge about the neocortex. It relies on mechanisms such as sparse coding or Hebbian learning to learn patterns and sequences via unsupervised learning. The HTM layer is to some extent analogous to our locally recurrent layer: Both layers contain neurons connected both to the input and to other neurons in the same layer, learning in this way to encode input patterns within a "context" (i.e., they contain information about the current and previous inputs). Some differences are that HTM works with boolean-valued (instead of real-valued) neurons (which may be a limitation in terms of dealing with uncertainty and ambiguity) and that they use Hebbian learning (instead of backpropagation) for the training of the system. Hebbian learning has been in fact explored in the training of the NILRNN, but the unsatisfactory results obtained seemed to indicate that adapting the system for such learning may not be that straightforward. On the other hand, the self-organization mechanism that emerges in our recurrent layer, allowing a max pooling layer to then semantically group information, is something that does not exist in HTM. Still, HTM does group in columns neurons with shared input weights but different recurrent weights, i.e., neurons recognizing the same input pattern in different contexts. A similar arrangement has also been explored in our system, as it seemed it could push more together similar patterns occurring in different contexts (e.g., edges with similar orientations but shifting at different speeds), making our system closer to that desired "feedforward" behavior previously mentioned. However, the results obtained were not able to outperform those obtained without the columns.

3.5.2 Robustness

So far, we have compared the NILRNN with other existing systems, discussing possible future directions mainly about its architecture (e.g., deep hierarchy, partially connected input, etc.). Other ideas that can be investigated are those involving the increase of the robustness of the system. Regularization methods, while typically focused on reducing overfitting in situations with limited training data, often bring other advantages to the system related to its robustness. Our system already includes two regularization terms in its loss function: the L2 regularization term and the sparse regularization term. In addition, the predictive feature of our training system may also be acting as a regularization method: Preliminary results with systems reconstructing previous inputs instead of predicting next ones seemed to simply try to remember those previous inputs. With the predictive feature, instead, the NILRNN seems to keep a more meaningful context for the current input that allows it to better predict next inputs and handle uncertainty, penalizing overconfident predictions and leading to a better generalization and to a more robust system.

Other regularization methods could also be explored in our system, such as that of denoising autoencoders (i.e., introducing some kind of noise into the training input while leaving the output unchanged) or dropout (this direction is also explored in Chapter 4). Still, dropout may not be as effective in our system, as, besides being mainly designed to address the issue of limited data [191] (which is generally more critical in supervised systems), it doesn't seem to perform so well when using sigmoid activation functions [192], and it also presents some issues when being applied over recurrent networks [193].

3.5.3 Comparison Against Neocortical Behavior

Finally, a better understanding of the NILRNN could also come from a further comparison of its behavior and emerging characteristics against those observed in the neocortex. The results presented in this chapter already show that its behavior is in different ways analogous to that of the primary visual cortex. This means that it can function to some extent as a model of the primary visual cortex and contribute to obtaining new insights about its principles of functioning, similarly to CNNs [185]. In fact, the NILRNN is analogous to the primary visual cortex to a larger extent than CNNs both in terms of structure and of emerging behavior, allowing it for example to develop orientation maps. Furthermore, the NILRNN is based on ideas that do not only apply to the visual cortex but which may also apply to other areas of the neocortex (as it relies on grouping together input patterns that tend to occur close in time, and not on grouping together spatially shifted versions of the same pattern). This allows the NILRNN to be applicable to domains with very different properties from those of computer vision, as well as to possibly serve as a model of different areas of the neocortex and, thus, contribute to the advancement of our knowledge about those areas and the neocortex in general.

On the other hand, similar artificial systems could be built based on the same principles observed in the neocortex but using different techniques. Comparing such systems would also allow us to understand what the best performing implementation is. For instance, a system with similar properties could be built by substituting the recurrent layer by a feedforward layer with an added loss term that penalizes very fast changes in space and time in its activity (or that rewards smooth and slow changes in the activity along the layer). This would reduce the internal complexity of the system, besides making unnecessary the prediction terms in the output of the learning system. In addition, once trained, this system would also admit static data (e.g., images) as input. However, all the internal information in the recurrent layer about the previous inputs (which could be useful in some applications) would be lost. In this sense, this system could help to better understand the effects of the NILRNN recurrent connections (apart from the known effect of developing order in the locally recurrent layer). Comparing the NILRNN with such alternative versions could contribute to a better understanding of its behavior and to finding new directions to explore.

3.6 Conclusion

This chapter has introduced the neocortex-inspired locally recurrent neural network (NILRNN), which is a self-supervised representation learning shallow neural network for temporal data that draws inspiration from models of the primary visual cortex and other computational neuroscience concepts such as sparsity, self-organization, or slowness. The system has been compared with other shallow self-supervised representation learning systems in the task of learning representations for three different classification problems, outperforming all the other systems. It has also been compared with a variant of the system without its output max pooling layer. While the results seem to show that the complete system generally outperforms the ablated version (at least for input sparse representations), this is not yet completely clear and needs further investigation. In addition, its emerging behavior has been compared against that of the neocortex, showing analogous behavior and validating it as a model of the primary visual cortex.

Regarding future work, one possible direction involves building a hierarchy of stacked NILRNN blocks that can compete and be compared directly with other deep state-of-the-art systems in the task of representation learning of temporal data. With the appropriate design, one advantage of the deep version of our system could be that it can still be trained in a shallow manner. Another potential direction to explore is the introduction of modifications to the system in order to improve its performance, e.g., by bringing new ideas from computational models of the brain or from other machine learning systems. Indeed, the fact that it is a novel system leaves much room for analysis and exploration of different possible variants, which may lead to new improved alternative versions. These ideas are explored in Chapter 4.

Chapter 4

HLRNN: BUILDING A HIERARCHY OF LOCALLY RECURRENT NEURAL NETWORKS FOR SELF-SUPERVISED REPRESENTATION LEARNING IN TEMPORAL DATA

The content of this chapter has been submitted to an international journal.

4.1 Introduction

Representation learning (or feature learning) is the task of automatically learning functions that extract features or alternative representations of the input data with some specific purpose. Self-supervised representation learning, for its part, consists of learning those functions from unlabeled data while using supervised learning techniques such as backpropagation, typically by automatically generating alternative inputs and/or target outputs from the unlabeled data. Since the data lacks labels or target output information that can help the self-supervised system find the best representations for the downstream task, these systems often require to be trained to do tasks that are specific to the type of data or problem [145]. In particular, systems dealing with temporal data typically rely on mechanisms that exploit the temporal nature of the data to learn more useful representations. However, existing temporal systems are not yet able to reach the performance of the human brain in many daily-life tasks, suggesting that the representations learned by our brain are more robust and general-purpose. In this sense, drawing inspiration from the structure and functioning of our brain seems a good direction to develop better representation learning systems for temporal data. In Chapter 3, we explored this direction and proposed the neocortex-inspired locally recurrent neural network (NILRNN), a brain-inspired system for self-supervised representation learning in temporal data. However, this system is shallow and was not designed to compete with current state-of-the-art deep systems.

In this chapter, we propose the Hierarchical Locally Recurrent Neural Network (HLRNN), which is a deep extension of the NILRNN. The HLRNN is a deep neocortex-inspired neural network designed to work with temporal data that is able to learn pyramids of features at different levels of abstraction. These different levels are achieved by exploiting the idea of slowness, with layers higher in the hierarchy working at slower timescales. The system is trained one shallow block at a time in a greedy fashion, enabling also an iterative shallow hyperparameter tuning of the system. The HLRNN has been compared with two state-of-the-art deep time series representation learning systems on three different datasets in terms of the accuracies and F_1 scores obtained on a classification task using the learned representations as input, as well as in terms of how separated the projections of the representations of the different classes are in the

t-distributed stochastic neighbor embedding (t-SNE) space. An ablation study has also been performed to understand how the different elements contribute to the behavior of the whole system. The results indicate that the architecture and methods presented in this chapter, in addition to achieving state-of-the-art performance, open a new direction on how to approach the problem of self-supervised representation learning over temporal data.

This chapter is organized as follows: Section 4.2 introduces state-of-the-art work and methods in the field of self-supervised representation learning. Section 4.3 describes the architecture of the system. Section 4.4 presents a comparison of our system and other state-of-the-art systems, as well as an analysis of our system that includes an ablation study. Section 4.5 discusses the system and the results obtained and examines possible implications. Finally, Section 4.6 summarizes the outcomes of the study and concludes the chapter.

4.2 Background

Self-supervised representation learning is the task of automatically learning useful alternative representation or feature extraction functions in a self-supervised way, i.e., relying only on unlabeled data while using supervised learning techniques. This is achieved by automatically generating target outputs (and possibly modified inputs) from the unlabeled data and/or by defining and encouraging certain properties of the output using specific loss functions. Self-supervised representation learning systems can be classified in terms of whether they take a generative or discriminative approach [140]. Generative systems are trained to generate some target output with properties of the input data (e.g., a patch of an input image), while discriminative systems are trained to deduce properties (e.g., the angle of a rotated picture) or relationships (e.g., whether two augmentations come from the same sample) from the given input(s), and often rely on contrastive learning techniques designed to obtain similar representations for inputs with similar meanings (and dissimilar representations otherwise). A common advantage of generative approaches is that, since they keep most of the information in the input data (as it is necessary for the reconstruction), the representations learned are useful in many different applications, and they often do not require taking assumptions about the downstream task where the learned representations will be used. The other side of the coin is that, since discriminative approaches are typically designed considering the downstream task, the representations learned are often more appropriate for that specific downstream task, being more invariant to features that are irrelevant for the task [194]. Indeed, generative approaches require mechanisms to avoid learning only low-level features, as these are in principle necessary and sufficient to reconstruct the input. In addition, they also suffer from being very sensitive to outliers. Consequently, discriminative approaches are often preferred when dealing with images (e.g., the simple framework for contrastive learning of visual representations, SimCLR [139]), as low-level features (e.g., pixel-level features) are generally not necessary to capture the semantic information of the

image required by most downstream tasks. Generative approaches, on the other hand, are quite popular when dealing with natural language (often also combined with contrastive techniques, as in bidirectional encoder representations from transformers, BERT [142]), as low-level features in this case (e.g., words) are usually decisive for the semantic meaning of the text. Finally, there are also self-supervised representation learning systems that have elements of both generative and discriminative approaches, such as those based on generative adversarial networks (GANs) (e.g., bidirectional GANs [195]) [194].

Self-supervised representation learning systems can also be classified attending to the pretext task(s) used, i.e., the unsupervised learning problems addressed to learn the features (which mainly consist of defining what the inputs and target outputs are and how they are generated or what properties they should have). According to [145], pretext tasks can be classified into masked prediction tasks, transformation prediction tasks, instance discrimination tasks, and clustering tasks (or a combination of them). In masked prediction tasks, part of the input data is removed, and the system is trained to predict this missing data. Transformation prediction tasks rely on applying detectable transformations to the input data (e.g., rotating a portrait picture) and asking the system to predict the transformation. Instance discrimination tasks assume each sample (and usually augmentations of it) as its own class and ask the system to discriminate between classes. Finally, clustering tasks make use of clustering techniques to assign labels that are then used to train the system. According to [194], the first two types of tasks can be seen as context-instance tasks, while the last two can be seen as instance-instance tasks. In any case, this classification in terms of pretext tasks is quite orthogonal to the previous one mentioned in terms of approaches, and, in the literature, one can find generative and discriminative approaches using similar pretext tasks [140]. On the other hand, there are some popular pretext tasks that are exclusive of generative models and that are not considered in [145], such as those of autoencoders (which can be referred to as bottleneck tasks [140]), those of bidirectional GANs (which can be referred to as adversarial tasks [194]), or those of flow-based models (e.g., [196]).

When it comes to temporal data, the literature is in general divided into video data, with systems implementing mechanisms to deal specifically with images (see [197] for a comprehensive survey on the topic), and other forms of time series, with works that often propose solutions applicable to multiple domains. Focusing on this second, more general area, both generative and discriminative approaches have been extensively used for the self-supervised learning of representations. It is worth mentioning that these systems often draw inspiration from successful systems from the field of natural language processing, as this field has experienced significant progress in recent years and also deals with sequential data. Among the self-supervised representation learning systems for time series, generative approaches often make use of masked prediction pretext tasks, e.g., in the form of autoregressive models trained to predict future samples. For example, the time series transformer (TST) [198] learns representations of the input by

learning to predict masked segments of different variables of a multivariate time series, relying on transformers to do so. The NILRNN (see Chapter 3), on the other hand, is an autoregressive model that was designed inspired by the architecture of the brain and motivated by the slowness principle (commonly applied to extract structural characteristics of temporal data [140]), relying on a max pooling layer to get rid of the irrelevant fast-changing low-level information.

Discriminative approaches, on the other hand, often rely on mask prediction or instance discrimination pretext tasks (combined with augmentation techniques, see [199] for a review) and have become very popular in recent years. Among these systems, it is worth mentioning contrastive predictive coding (CPC) [200], which is trained to determine whether a given sample is a future sample of the current sequence (i.e., it can be seen as a contrastive autoregressive model) and has also been applied successfully to other types of data such as images or natural language, serving as inspiration to many other self-supervised representation systems. Another very popular example, though designed specifically for speech, is wav2vec-2.0 [143], which learns speech representations through a contrastive task that determines whether a quantized form of a sample representation corresponds to a masked sample. The time series representation learning framework via temporal and contextual contrasting (TS-TCC) [201], on the other hand, relies on two augmentations and two different contrastive tasks: one to determine whether two representations are augmented versions of the same sample and another to determine whether a given sample is a future sample of the other augmented version of the sequence.

Self-supervised representation learning systems dealing with temporal data can also be classified attending to whether they only learn representations for the whole sequence or they are also able to build representations for each input sample. An advantage of systems building representations for each input sample is that they don't require a previous segmentation process, being hence applicable to sequences formed of subsequences of different classes, such as those found in real-time applications (furthermore, the learned representations themselves can be used to segment appropriately the input sequence). In addition, a whole-sequence representation can be directly obtained from the individual sample representations, while the opposite is, in general, not true. All the systems presented so far learn representations for each sample. An example of a system that learns directly representations for the whole sequence is bilinear temporal-spectral fusion (BTSF) [202], which applies a contrastive task to decide whether two dropout-based augmentations come from the same input sequence, relying on a feature extractor that combines time and frequency information of the input. The representations learned for each input sample can also be classified in terms of whether they are generated mainly from the current sample and possibly the surrounding ones or from a broader set of samples (e.g., the whole sequence, all past samples, etc.). These last representations are typically referred to as context representations, and they are often obtained from the other "local" representations through long-term dependency units such as long short-term memories (LSTM) or transformers. CPC and TS-TCC generate

both local and context representations that rely solely on past samples, while NILRNN only generates local ones from past samples and TST generates directly context representations from the whole sequence.

4.3 Methods

This section describes the hierarchical locally recurrent neural network (HLRNN), which is a deep neocortex-inspired self-supervised representation learning system for temporal data that mainly consists of a stack of modified NILRNNs known as Locally Recurrent Blocks (LRB). The system is designed so that each LRB can be trained independently and iteratively, making the training shallow and fast and allowing hyperparameter tuning processes to be applied over parts of the network, dealing thus with smaller sets of hyperparameters while allowing them to be different at the different stages.

The NILRNN design was to a large extent inspired by the model of the primary visual cortex presented in [169], which has the advantage over other models that, due to its design, it may also serve as a model of other areas of the neocortex. Indeed, the neocortex is quite uniform in terms of structure and operation along most of its areas, suggesting that its different functions (e.g., perception, rational thought, language, etc.) may be performed by the same underlying algorithm [155]. This motivated the development of the NILRNN as a neocortex-inspired representation learning system for different types of temporal data. Thus, the NILRNN is not only able to learn patterns similar to those of the primary visual cortex or of convolutional neural networks (CNN) when fed with shifting images as input, but it may also show similar behaviors to other areas of the neocortex when fed with other types of input data. Considering the hierarchical structures in which areas in the neocortex are organized, one can also expect that, similarly to how occurs in the neocortex, by stacking a set of NILRNNs, the resultant system will be able to learn representations of the input at different levels of abstraction, with levels higher in the hierarchy learning representations that change at slower timescales. This is the motivation behind the HLRNN.

The HLRNN mainly consists of a set of N stacked LRBs (see Fig. 4.1). The LRB is basically a more robust version of the NILRNN that includes some modifications in the direction of making it a better performing machine learning system (note that the NILRNN was designed to a large extent as a model of the neocortex), as well as downsampling functionality (to have different timescales at different levels). The LRBs in the hierarchy take as input the output from the previous block (fully connected) and are trained in a greedy manner, i.e., one block at a time and once the previous blocks have been trained (roughly mimicking the idea of hierarchical maturation of the neocortex [189]). Similarly to the NILRNN, these blocks follow a self-supervised training process consisting of reconstructing and predicting the input to the block. The first LRB takes as input a temporal window layer applied over a downsampled input. This

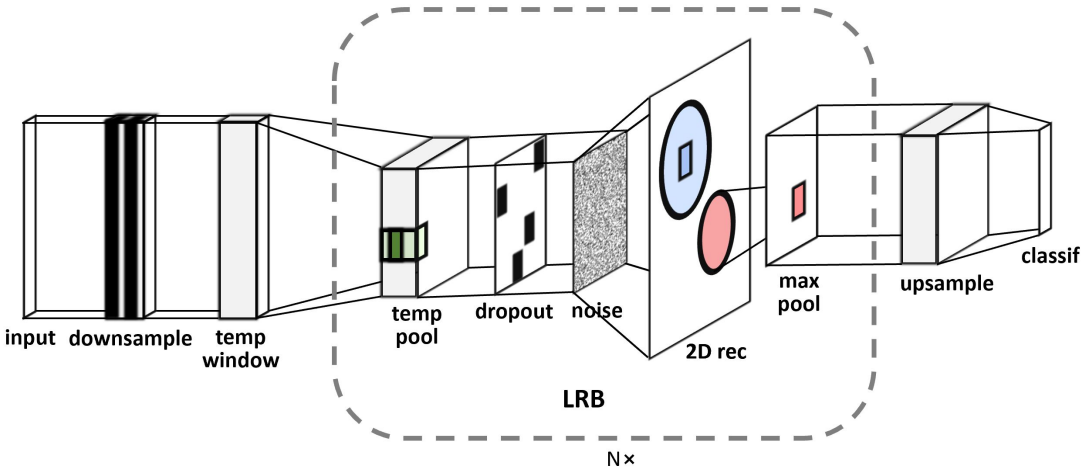


Figure 4.1: The HLRNN architecture, consisting of a downsampling and a temporal window layer followed by a series of LRBs that gradually extract slower and higher-level representations of the input. The LRB is a modified NILRNN that includes a temporal max pooling, a dropout, and a Gaussian noise layer at its input, learning in this way more robust representations.

layer will be commented on later in this section, but its main motivation is to make the system better at dealing with very small input sample vectors (e.g., univariate inputs) or with inputs with relevant information in the frequency domain, such as audio data. Taking into account that the data is downsampled at the input downsampling layer and at each LRB block (as is described later in this section), an upsampling layer of scale factor $\eta_{\uparrow}$ can be added at the top of the hierarchy if a representation per input sample is required (in this study, we have used a linear upsampling layer). Finally, Fig. 4.1 also shows a classifier at the top of the hierarchy using as input the upsampled higher-level features learned by the system.

As can be seen in Fig. 4.1, the LRB includes three new layers at its input that the NILRNN did not: a temporal max pooling layer, a dropout layer, and a Gaussian noise layer. The temporal max pooling layer performs a max pooling operation over the time axis and is characterized by a kernel (or window) size ω and a stride $\eta_{\downarrow}$ (note that the stride is also the downsampling factor). The main motivation behind this layer is the same as a downsampling layer would have: The representations higher in the hierarchy vary at slower timescales, and, therefore, it makes sense that blocks higher in the hierarchy work at slower timescales, both from a computational and from a functional point of view (in addition, it involves dealing with further-in-time predictions at the training output). On the other hand, since we are working with sparse representations, it also makes sense to group together input patterns that have been detected around the sampling time, which is why we use a temporal max pooling layer instead of a more standard downsampling layer (note also that, together with the 2D "spatial" max pooling layer at the output of the previous LRB, these layers form a 3D "spatiotemporal" max pooling layer). The next layer in the LRB is a standard dropout layer that maintains a constant mask along each batch during training

(i.e., the zeroed-out elements at each sample of the batch are the same) and is characterized by a masking probability p. Then, during training, the Gaussian noise layer adds zero-centered Gaussian noise to the input characterized by a standard deviation σ. The LRB is trained to reconstruct and predict the data at the output of its temporal max pooling layer. Therefore, both the dropout and Gaussian noise layers contribute to the robustness of the learned features, making the masked prediction pretext task more challenging and pushing the system to better exploit the relationships among elements within a sample (in part thanks to the constant dropout masks).

Another difference between the LRB and the NILRNN is that both the locally recurrent layer and the feedforward output layer used for training the LRB use ReLU (instead of sigmoid) activation functions, which lead in general to a faster convergence when using standard backpropagation for training. Therefore, since the activity in the locally recurrent layer is no longer guaranteed to be below 1, the LRB uses an L_1 regularization-based sparsity loss term as shown in (4.2) (instead of the Kullback-Leibler (KL) divergence-based one in (3.1) used for the NILRNN). In addition, a slowness-oriented contrastive term has been added to the loss function as an additional mechanism to push learned representations of close-in-time inputs to be more similar to each other. Negative samples are not required, since the trivial solution of learning the same representation for every input is already avoided through the generative pretext task. This slowness loss term consists of a sum of squared errors over pairs of representations at the output of the 2D max pooling layer that are at a maximum distance of δ timesteps within the same batch. Since it does not need negative samples and only relies on samples from the same batch, one advantage of this contrastive term is that its implementation for online learning scenarios is straightforward, without the need to keep in memory samples from earlier batches. Thus, the resultant loss function used to train an LRB is the following:

$$J(W, b) = J_{error} + \lambda \cdot J_{regularization} + \beta \cdot J_{sparse} + \gamma \cdot J_{slowness}\,, \tag{4.1}$$

where W and b represent the weights and bias units in both layers, J_{error} is the weighted Minimum Squared Error (MSE) loss term as described in (3.2), $J_{regularization}$ is the L_2 regularization term as described in (3.4), J_{sparse} is the L_1 regularization-based sparsity term, $J_{slowness}$ is the slowness-oriented contrastive term, and λ, β, and γ are loss term weights:

$$J_{sparse} = \frac{1}{m} \sum_{i=1}^{m} \|a_i^{(r)}\|_1\,, \tag{4.2}$$

$$J_{slowness} = \frac{1}{2 \cdot \delta \cdot (m - \delta)} \sum_{i=1}^{m-\delta} \sum_{j=1}^{\delta} \|a_i^{(p)} - a_{i+j}^{(p)}\|_2^2\,, \tag{4.3}$$

with m being the batch size (or sequence length), $a_i^{(r)}$ the i-th sample of the batch at the locally recurrent layer, $a_i^{(p)}$ the i-th sample of the batch at the 2D max pooling layer, $\|\cdot\|_1$ the L1 norm

operator, and $\|\cdot\|_2$ the L2 norm operator. Note that by sample we refer to a time sample in a sequence, not to a full sequence in a dataset.

As mentioned above, a downsampling layer and a temporal window layer have also been included at the input of the HLRNN. The downsampling layer, characterized by a scale factor $\eta_{\downarrow}$, addresses the issue of having data sampled at high rates as input (with respect to the speed at which it actually changes, i.e., data with high redundancy between consecutive samples). Thus, the motivation is similar to that of the temporal max pooling layer in the LRB, facilitating the efficient use of past information and reducing the computational cost. Regarding the temporal window layer, it is characterized by a window size ω. This layer allows the first LRB to have as input a number of successive samples from which it is easier to learn proper representations, especially when the input sample size is small and there is relevant information in the frequency domain (e.g., speech data). In addition, it increases the redundancy of the input, making the different input patterns more easily separable and more appropriate for the first LRB input, designed to work with sparse representations at its input (note that the other LRBs take as input the output from their previous LRB, which is already sparse). Still, when the input has a dense representation with relevant information on its precise values, this may not be enough. In this regard, a variant of the HLRNN has been designed that substitutes its first LRB by a deep LRB. The deep LRB is an LRB with a hidden layer just before its locally recurrent layer that allows neurons in this layer to learn arbitrarily complex patterns from the input. In its learning configuration, a hidden layer is also introduced between the locally recurrent layer and the output layer to allow the network to properly reconstruct/predict the dense input. A more standard sparse autoencoder at the input of the HLRNN was also considered to deal with this type of data, but unsatisfactory preliminary results prevented us from considering it further.

Finally, the HLRNN does not include transformers or other long-term dependency units such as LSTMs in its architecture. Therefore, the representations at its output can be considered local representations (as opposed to context representations as introduced in Section 4.2).

4.4 Results

This section analyzes the performance of the HLRNN by comparing it with other state-of-the-art self-supervised representation learning systems for temporal data and with a number of HLRNN variants. For this purpose, the systems have been tested on the task of learning sample representations for two popular temporal classification tasks: speech recognition and action and goal recognition. In particular, the systems have been tested on the same WARD [180] and FSDD [181] datasets introduced in Section 3.4.1 (the data has also been preprocessed and normalized in the same way), as well as on an extension of the synthetic action input introduced in the same section, which now includes complete plans (described in Appendix 4.A). Considering that only multivariate data has been used in this study (due in part to the fact that TST, one

Table 4.1: Main properties of the data inputs used to evaluate the performance of the HLRNN.

Data input	**sample size**	**# samples**	**# classes**
WARD	25	565,755	**13 actions**, 20 subjects
FSDD	40	126,750	**10 digits**, 6 subjects
Synthetic actions	155	$\sim\infty$	5 primitives, 4 actions, **5 goals**

of the systems with which our system is compared, is limited to work with this kind of data data by design), the performance of the HLRNN with univariate time series remains an open question. On the other hand, since WARD data does not take a sparse representation form, a deep LRB has been used at the input of the HLRNN when dealing with it. The sample sizes, number of samples, and number of classes per classification criterion used for these data inputs are shown in Table 4.1. When not specified, the classification criteria in bold have been used. The two datasets have been divided, using stratification with respect to all classification criteria, into a training set ($\sim$60% of the samples), a validation set ($\sim$20% of the samples), and a test set ($\sim$20% of the samples).

The LRBs and classifiers have been initialized via Xavier initialization and trained using back-propagation and the Adam optimization algorithm with stepsize α. For the task of classification, linear classifiers or shallow vanilla recurrent neural networks (RNN) have been applied over the learned representations. The linear classifiers have been used for the comparison with other state-of-the-art systems (on top of the upsampling layer to have one representation per input sample), as this is the more standard way to proceed in the literature when comparing representation learning systems. Still, since the HLRNN is designed to only learn local representations of the input and context information may also be relevant for the classification task, classifiers that can keep such context information are preferred for our system. In this regard, for the sake of simplicity, vanilla RNNs have been used for the rest of the HLRNN analysis (ablation study and hierarchy analysis), applying the upsampling layer after the classifier so that it can take advantage of the slower timescales and, in this way, it can better exploit longer-term relationships. To evaluate the systems, the accuracy and macro-averaged F_1 score evolution curves with respect to the number of classifier training samples have been used (since the datasets used are quite balanced, the results obtained for both metrics are similar). These curves, besides providing information about the reached performances, also show how fast the classifiers are able to achieve specific performances when taking the different learned representations as input.

The hyperparameters of the different systems have been tuned using Bayesian optimization. Since the HLRNN is trained sequentially, one block at a time, the hyperparameter search has also been carried out in a sequential way, doing one search per block and using vanilla RNNs as classifiers in the intermediate steps. This allows the different blocks to have different

Table 4.2: Number of samples per dataset used to train the self-supervised systems and classifiers during hyperparameter tuning.

Data input	LRB training	TST/TS-TCC training	classifier training
WARD	500,000	5,000,000	40,000
FSDD	1,000,000	2,000,000	60,000
Synthetic actions	500,000	2,000,000	10,000

hyperparameter values (similarly to how the different areas of the neocortex have different sizes, densities, etc. [155]). To avoid having to deal with very large search spaces, some constraints have been set on the LRB hyperparameters: The 2D locally recurrent layers take a squared shape (i.e., $l_h = l_w$), the number of inputs reconstructed/predicted is set to $n_{\hat{x}} = 5$, all those predictions are weighted equal in the error loss term (i.e., $w_{\hat{x}} = 1$), the dropout layer masking probability is set to $p = 0.2$, the Gaussian noise layer standard deviation is set to $\sigma = 0.2$, and the maximum sample distance of the slowness-oriented contrastive term is set to $\delta = 1$. Regarding the depth of the HLRNN, it has been set to $N = 4$ LRBs to work with a depth similar to that of the other two systems with which our system is being compared (although preliminary results have shown that further levels can continue improving the performance of the system). Since the datasets used are balanced, accuracy has been chosen as the metric to be optimized during search. During this hyperparameter tuning process, the self-supervised networks and classifiers have been trained for a number of samples that depends on the dataset, as indicated in Table 4.2 (in the case of the LRBs, that value refers to the number of samples at the layer being trained, i.e., corresponds to a number of samples at the input of the HLRNN that is greater than that value by the corresponding downsampling factor). The number of training samples for each self-supervised network has been chosen large enough so that increasing it didn't seem to improve the quality of the learned representations. Regarding the metric evolution curves, they correspond to the classifiers that performed best in the validation set among 32 equal classifiers trained in parallel. All the metric evaluations have been performed over 200,000 samples.

4.4.1 Comparison with Other Systems

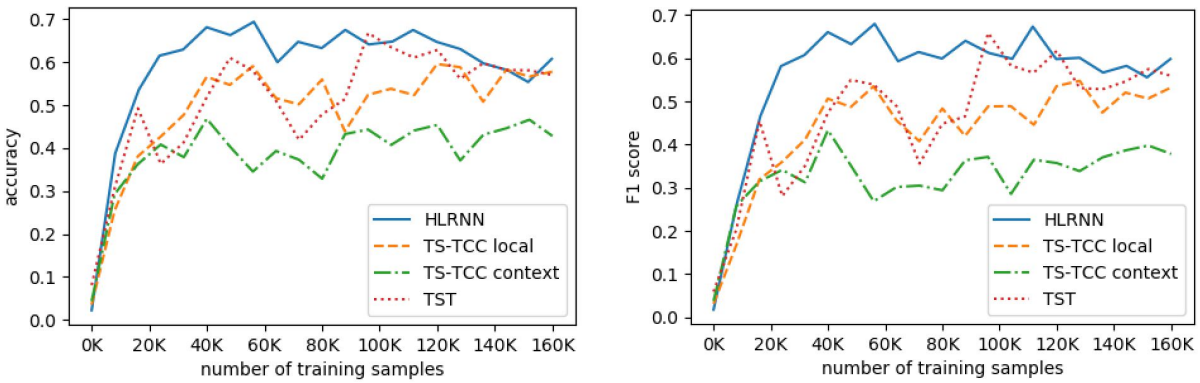
We have compared our system with TST [198] and TS-TCC [201], two of the state-of-the-art self-supervised representation learning systems for temporal data introduced in Section 4.2 that learn representations for each input sample. TST takes a generative approach through a masked prediction task. It applies transformers over a linear transformation of the input samples to extract context representations c, relying on all the samples of the sequence (also the future ones). TS-TCC, for its part, takes a contrastive approach through the combination of a masked prediction task (which takes an autoregressive form) and an instance discrimination task. It uses a convolutional neural network (CNN) to extract local representations z and, then, uses

transformers that rely only on the previous and current samples to build context representations c. As commented in Section 4.3, the HLRNN combines generative and contrastive pretext tasks and only extracts local features z.

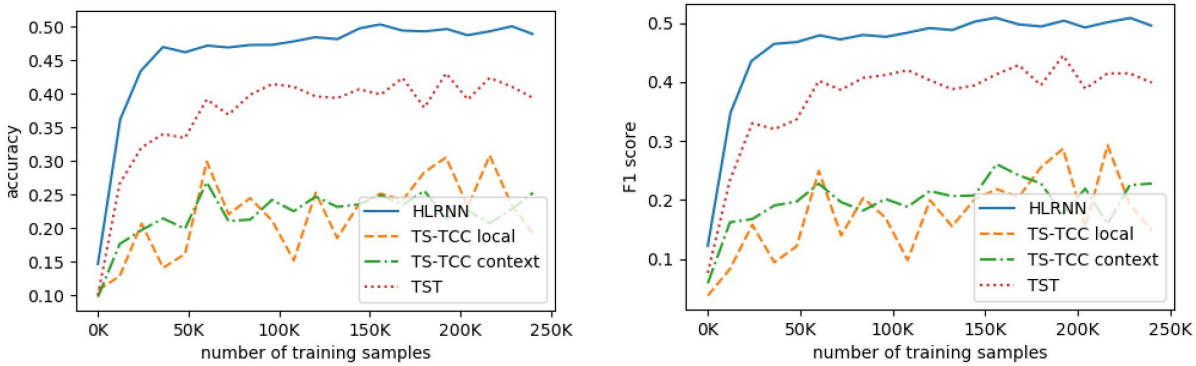
As mentioned earlier, linear classifiers have been used for this comparison. Since the intermediate hyperparameter searches for the first four LRBs of the HLRNNs use RNN classifiers, a fifth LRB has been added on top followed by a linear classifier for this comparison. Since TS-TCC also donwsamples the data, an upsampling layer has been included between the TS-TCC representations and the classifier. The TST and TS-TCC hyperparameters (those not hardcoded) have also been obtained using Bayesian optimization. The batch sizes of the classifiers on top of transformers have been set to the sequence length of those transformers. The number of training samples used during this hyperparameter tuning phase are indicated in Table 4.2. Tables 4.3, 4.4, 4.5, 4.6, and 4.7 from Appendix 4.B show the hyperparameters obtained for the HLRNN, TST, and TS-TCC for the different datasets. There are few hyperparameters that take values that are quite different from those in the original article, such as τ or λ_2 from TS-TCC, but fixing them to their original values led to worse results, so we decided to let them free for the search.

Figs. 4.2 shows the accuracy and F_1 score evolution curves obtained for the three datasets and for the different learned representations (i.e., for the HLRNN local, TST context, TS-TCC local, and TS-TCC context representations). The results show that the HLRNN outperforms the other systems in all datasets in terms of both the performance achieved and the number of training samples required to achieve such performance. This is especially noticeable for the FSDD dataset and for the synthetic action input, whose data is sparse and, therefore, very well suited for the network design, allowing the HLRNN to work at its best. Regarding the WARD dataset, the HLRNN reaches its peak performance for around 60,000 training samples and then degrades with successive training, matching in the end the performance of TST and of the TS-TCC local features. However, note that the hyperparameter search was performed exactly for that number of samples (see Table 4.2), for which the HLRNN achieves unmatched performance. In any case, this degradation may be suggesting that the HLRNN with a deep LRB at the input, while it has still a good performance, may not be working as optimally as the original HLRNN over sparse data (note that the deep LRB was basically built by adding a patch over a system that had been designed to get sparse representations as input). Finally, it is worth commenting that, while some accuracies may seem low (especially for the FSDD dataset), the classification is being done taking as input the learned representation of one time sample, and not of the whole sequence (e.g., for local features, the representation may only have information on the ongoing phoneme, and not on the whole word).

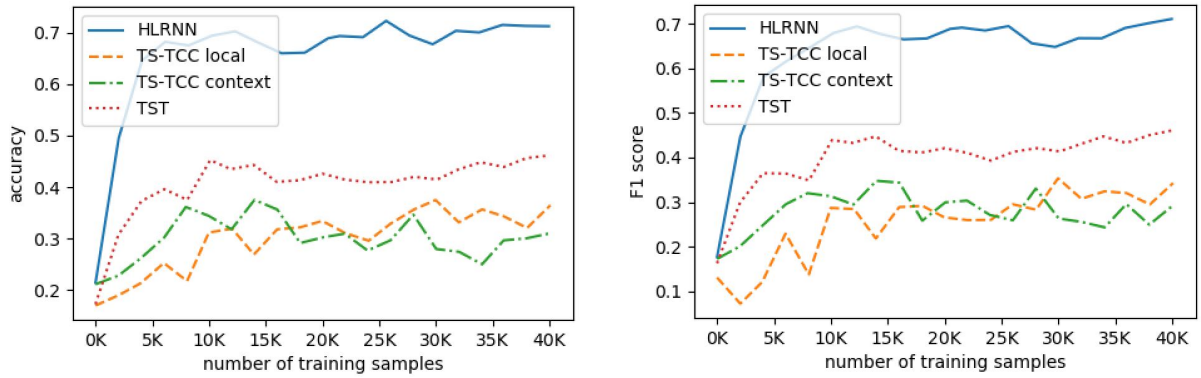
To complement the previous results and provide a visual understanding of how separated the learned representations of samples of different classes are, a t-SNE analysis has been employed.



(a) WARD dataset.



(b) FSDD dataset.



(c) Synthetic action input.

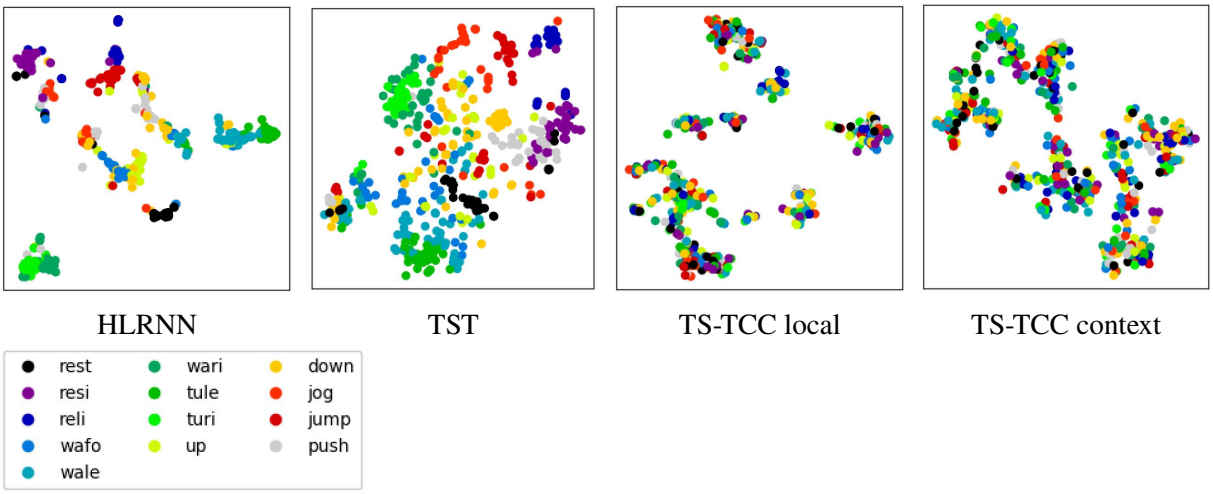
Figure 4.2: Accuracies and F_1 scores estimated for the different datasets, systems, and number of training batches. The HLRNN outperforms all the other systems for all the datasets. However, for the WARD dataset, the HLRNN performance degrades after a large enough number of samples, becoming similar to that of other systems, which may be related to the fact that the input data does not take a sparse representation form.

For this analysis, cosine similarity has been used as the similarity metric (since the representations are sparse), and a perplexity of 40 has been chosen, as this value has been shown to be appropriate to show the structure of the data. The sample representations used have been taken with enough separation in time between them to avoid the common issue of t-SNE of generating lines of points when applied over sequential data. Fig. 4.3 shows the t-SNE visualizations of the learned representations for the different systems and datasets. These figures are consistent with and point to conclusions similar to those obtained from the accuracy and F_1 score curves, with the HLRNN seeming to be able to learn more separated representations than the other systems. This is especially clear for the synthetic action input.

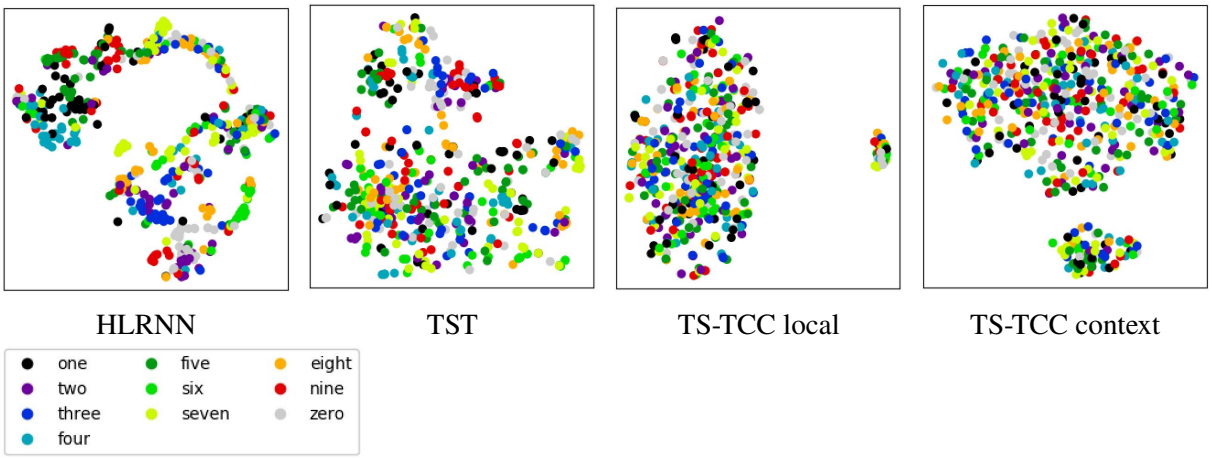
4.4.2 Ablation Study

To better understand how the different components of the HLRNN contribute to the overall performance of the system, we have compared the complete HLRNN against a set of variants of it. In particular, we have compared the complete HLRNN against HLRNNs with a deep LRB at the input (shallow LRB for the WARD dataset), without temporal max pooling layers, without 2D max pooling layers, without dropout and Gaussian noise layers, with computationally more efficient square instead of circular kernels at the locally recurrent and max pooling layers, without the slowness-oriented contrastive loss term, with sigmoid activation functions at the locally recurrent and training output layers, and with tanh activation functions at the locally recurrent and training output layers. In addition, we have also compared it with an HLRNN that shares hyperparameters across the different LRBs. For the sigmoid and tanh variants, we have used the KL divergence-based sparsity loss term in (3.1), with the tanh variant requiring the mapping of the estimated average activity per neuron in the locally recurrent layer $\hat{\rho}_i$ from the interval $(-1, 1)$ to the interval $(0, 1)$ before introducing it into the equation. In addition, for this variant, the input to the HLRNN has also been mapped to the interval $(-1, 1)$ so that it fits the range of the training output. Again, the hyperparameters of all these variants have been tuned using Bayesian optimization and the same number of samples and criteria described for the previous systems, having all of them a depth of four LRBs and using RNNs as classifiers at all levels of the search. These hyperparameters are shown in Tables 4.3, 4.4, 4.5, 4.8, 4.9, and 4.10.

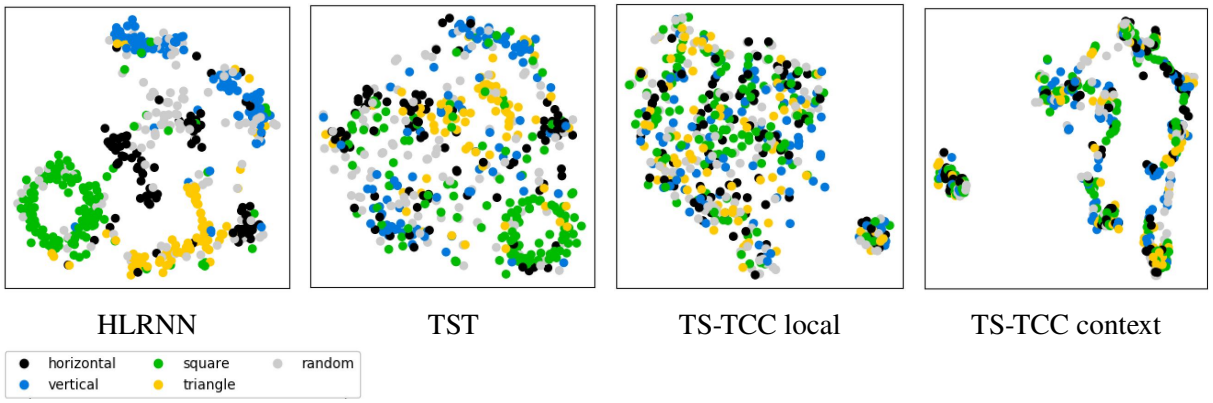
Fig. 4.4 shows the accuracy and F_1 score curves obtained for the three datasets and for the different HLRNN variants. Probably the most remarkable fact about these curves is that, in general, the standard HLRNN configuration outperforms all the other variants except for the tanh variant. Indeed, while the standard variant (with a ReLU activation function) has a better performance than the tanh variant on the synthetic action input, both variants perform similarly on the WARD dataset, and the tanh variant outperforms the standard one on the FSDD dataset. While the direct conclusion is that one or the other activation function performs better depending on the input



(a) WARD dataset.



(b) FSDD dataset.



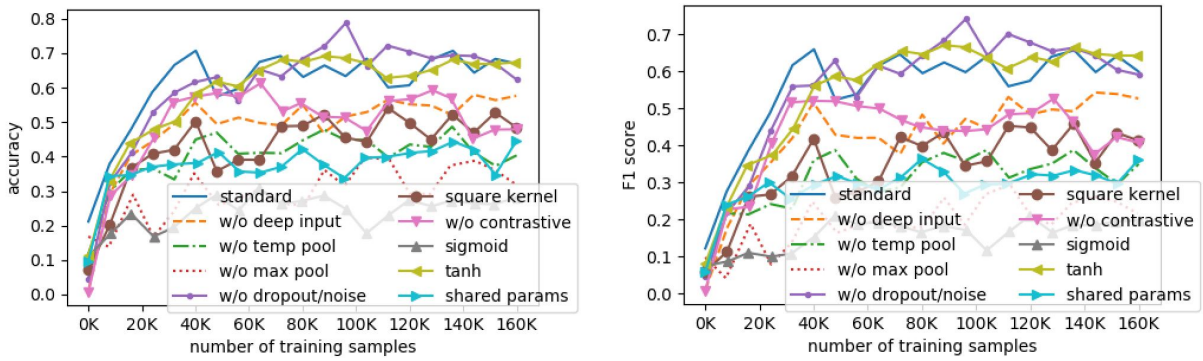
(c) Synthetic action input.

Figure 4.3: t-SNE visualization of the learned representations by the different systems and for the different datasets. The representations learned by the HLRNN seem to be more separated than those learned by the other systems.

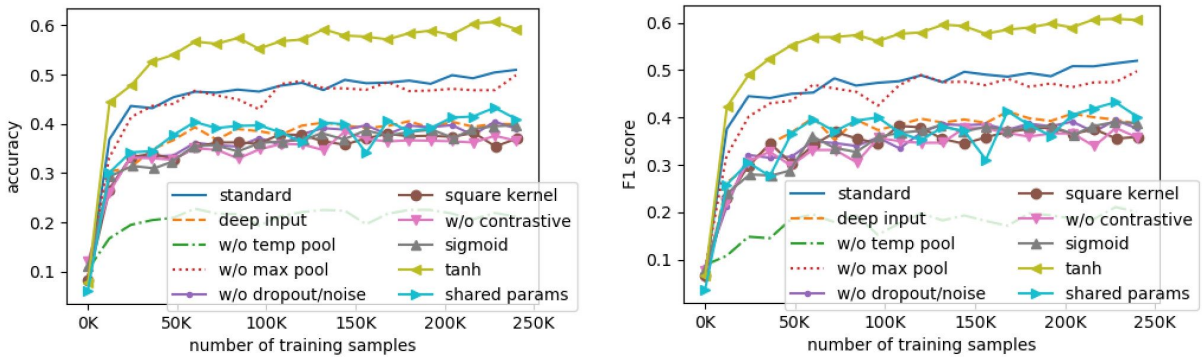
data, it is difficult to understand from these curves why this occurs or what characteristics of the input data make one or the other activation function a better choice. In fact, there are multiple differences between the two variants (activation function at the locally recurrent layer, activation function at the training output layer, sparsity function, effect of dropout, etc.), and maybe there is some intermediate configuration that leads to best results for both datasets (plus the configuration can also be different at the different HLRNN levels, with preliminary results suggesting that the change of activation function is especially relevant in the first LRB). For example, considering that ReLU is more appropriate to deal with sparse representations, as inactive neurons take a value of 0, while tanh is generally preferred in recurrent layers due to its symmetry and bounded range [203], using tanh at the locally recurrent layer and ReLU at the training output layer could perform better in general (together with the corresponding adaptation layers between LRBs, as the LRB output would be in the range $(-1, 1)$ and the training output layer would require LRB inputs above 0). Using a linear training output layer (without non-linearity) is also something that could be tested. In any case, this is open to further investigation.

The effect of the 2D max pooling layer and the contrastive loss term on the performance of the system is something that also deserves some analysis. These two mechanisms complement each other at bringing slowness to the output representations but seem to have a different relevance on the performance depending on the input data (as well as on the specific LRB or HLRNN level), with one of them even seeming enough in some cases. Indeed, the variant without 2D max pooling layers has a similar performance to the standard HLRNN on the FSDD dataset but a quite poor performance on the other two datasets. On the other hand, the variant without contrastive loss term reaches quite good performances on the WARD dataset and the synthetic action input, while this is not the case for the FSDD dataset. The interaction between these two mechanisms can also be appreciated on the hyperparameters, with the variant without contrastive loss for the FSDD dataset and the synthetic action input having larger kernels at the 2D max pooling layers and the variant without 2D max pooling layers for the WARD dataset having greater contrastive term weights. On the other hand, preliminary results on the HLRNN (as well as the results in Section 3.4.2) pointed to the fact that, when the input data is not sparse, the (shallow input LRB) variant without 2D max pooling layers has better behavior. Thus, the good performance of that variant on the FSDD dataset may also be suggesting that the data coming from a spectrogram is not sparse enough to get the best out of our system. On the other hand, it remains unknown whether this similar performance still occurs when removing the 2D max pooling layers from the system with tanh activation functions, which is the best performing variant for this dataset.

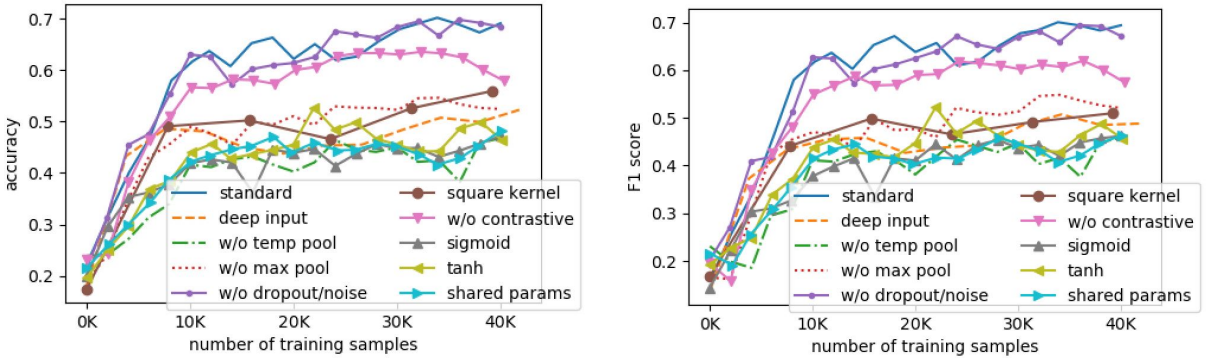
Regarding the other variants, the variant without the dropout and Gaussian noise layers also has a performance comparable to that of the standard HLRNN on the WARD dataset and synthetic action input but does not perform as well on the FSDD dataset. This seems to indicate that these two layers may or may not contribute to the performance of the system depending on the nature



(a) WARD dataset.



(b) FSDD dataset.



(c) Synthetic action input.

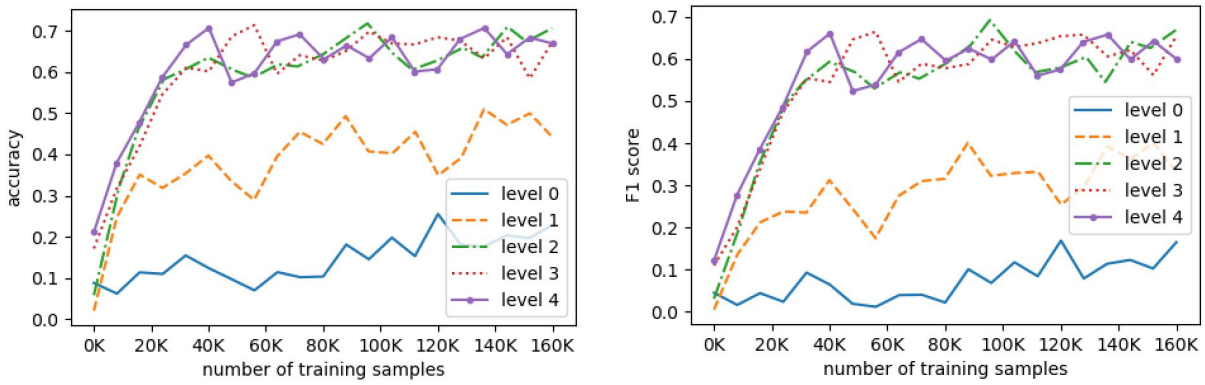
Figure 4.4: Accuracies and F_1 scores estimated for the different datasets, HLRNN variants, and number of training batches. Depending on the dataset, the standard, without dropout and noise, and/or tanh variants are the ones performing best. Among them, the only one not having a poor performance in any dataset is the standard variant, which is only clearly outperformed by the tanh variant on the FSDD dataset.

of the input data. However, note that the hyperparameters of these layers were set manually and not optimized, and different values may lead to better results for the standard HLRNN. The rest of the variants perform worse to a greater or lower degree than the standard HLRNN, indicating that the different components involved (temporal max pooling layers, circular kernels, slowness-oriented contrastive loss term, etc.) contribute to the performance of the system. Something worth mentioning is that, while it was expected that the HLRNN with a shallow LRB at its input would have a relatively poor performance on the WARD dataset, a good deep adaptation of the LRB should have reached performances similar to those of the standard LRB on sparse inputs. However, the performance of our deep LRB on the FSDD dataset and synthetic action input is quite poor, reinforcing the idea that, while the deep patch works, it is far from being ideal. Finally, it should also be noted how allowing the hyperparameters at the different levels to take different values brings a considerable boost in performance, which highlights the advantage of the step-by-step iterative search and training used on the HLRNN.

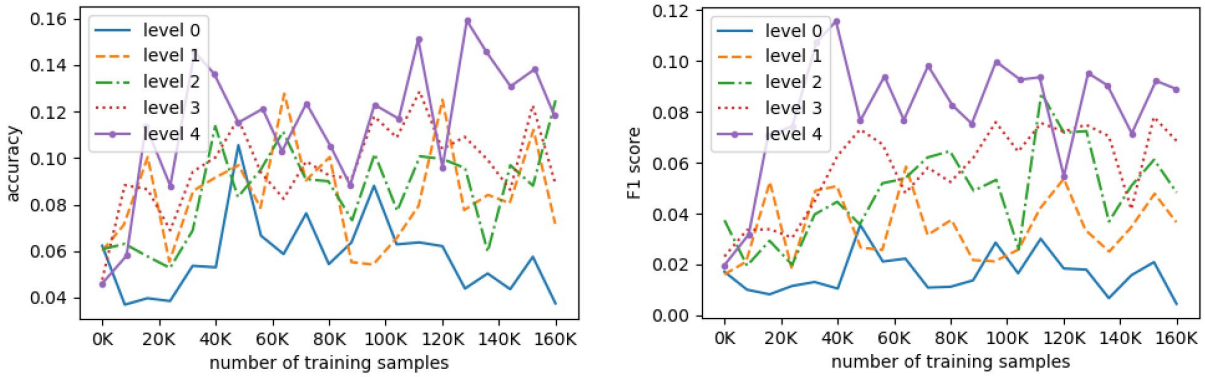
4.4.3 Hierarchy Analysis

Since the HLRNN learns representations of the input at different levels of abstraction, to better understand those representations and abstraction levels, as well as their utility in different tasks, we have set RNN classifiers for all classification criteria shown in Table 4.1 at the input (level 0) and at the output of the four LRBs of the hierarchy (levels 1 to 4). These classifiers have gone through the same hyperparameter tuning and training processes described for previous systems (note that the HLRNN itself has not been re-trained or tuned). The hyperparameters of these classifiers are shown in Tables 4.3, 4.4, and 4.5.

Figs. 4.5, 4.6, and 4.7 show the accuracy and F_1 score curves obtained for all levels, datasets, and classification criteria. Considering that the HLRNN hyperparameters were obtained attending only to the classification criteria in bold in Table 4.1, one can expect the learned representations at the different levels to be better suited for those classification criteria than for the others. Indeed, the curves show that, in general, the HLRNN performs quite poorly at extracting useful representations for classification criteria for which its hyperparameters have not been optimized (this is not the case of the FSDD dataset, for which it actually performs very good). In the case of the synthetic action input, the performance for the classification of primitives and actions actually decreases when going up the hierarchy. This was expected at least in the higher levels, as these classification criteria vary faster in time and require lower-level features, which are lost when moving up the hierarchy. To achieve more general-purpose representations at the different levels, something that could be tested is using multiple classifiers at each level during hyperparameter search (adjusting their weights on the fitness function according to the abstraction level). Regarding the classification criteria used for the HLRNN search, the classifiers keep increasing their performance at every level for the synthetic action input (in fact,



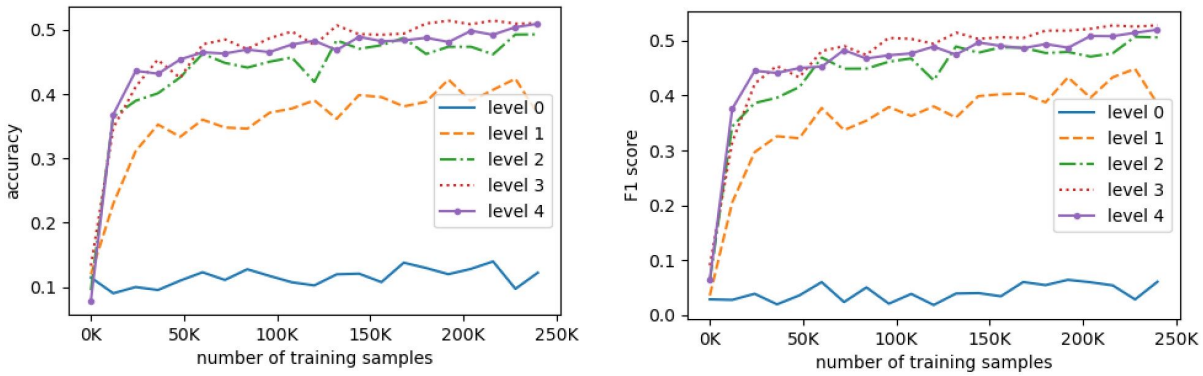
(a) Actions



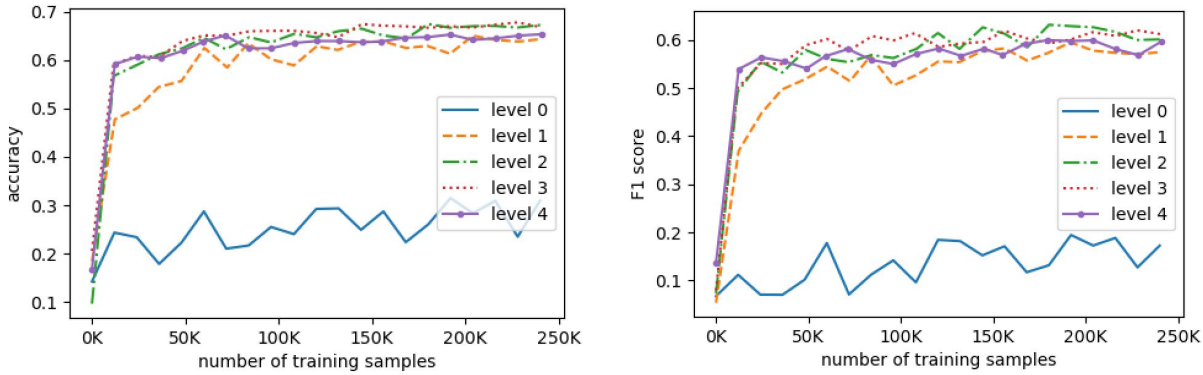
(b) Subjects

Figure 4.5: Accuracies and F_1 scores estimated for the WARD dataset and for different classification criteria, HLRNN levels, and number of training batches. For action clasification (criterion used for the HLRNN hyperparameter search), the performance increases with each HLRNN level until getting stuck at level 2. For subject classification, the performance increases with each HLRNN level (but the performance reached is quite poor).

preliminary results showed that this tendency continued for more levels), while the performance improvement gets stuck around levels 2 or 3 for the other two datasets. Considering also that, for those two datasets, the downsampling factor of the temporal max pooling layers and the kernel size of the 2D max pooling layers at most high-level LRBs take a value of 1 (see Tables 4.3, 4.4, and 4.5), this may be indicating that, due to the abstraction level of those classification criteria, they don't require HLRNNs more than two or three LRBs deep. If this is the case, higher-level classification criteria would be more appropriate to find good hyperparameters for the higher HLRNN levels, such as plans for WARD or sentences for FSDD. Alternatively, the fact that the hyperparameters of the lower LRBs are optimized to learn the best representations for the same high-level classification criteria may be leading to a too-fast shift to high-level representations that does not get the best of our system. In this sense, again, using multiple appropriately weighted classification criteria at each level during search may improve the resultant system architecture and the achieved performances.



(a) Digits

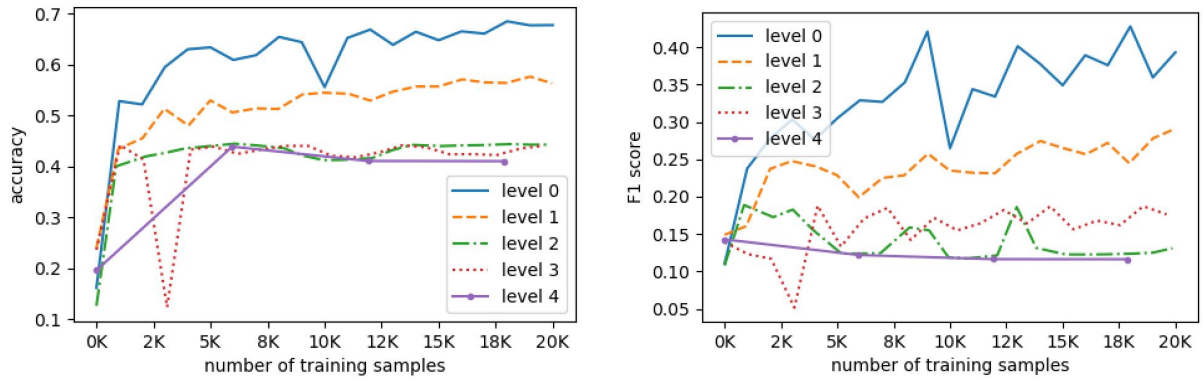


(b) Subjects

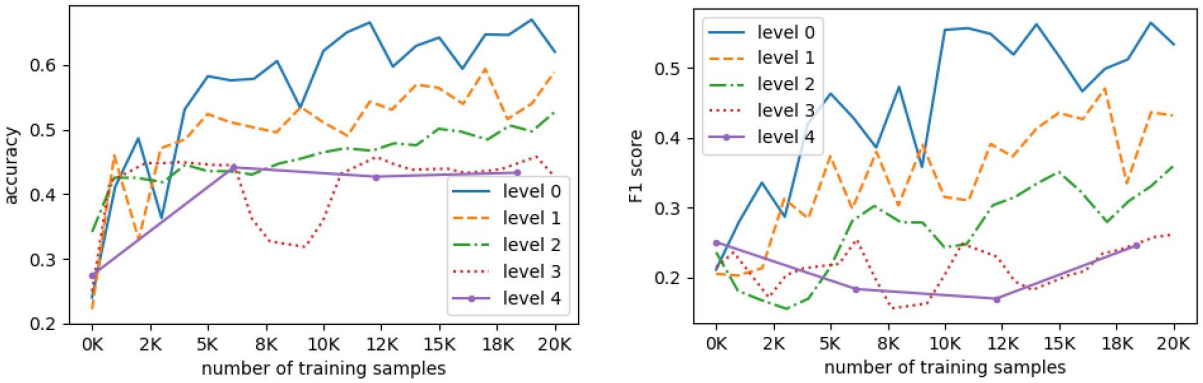
Figure 4.6: Accuracies and F_1 scores estimated for the FSDD dataset and for different classification criteria, HLRNN levels, and number of training batches. For digit clasification (criterion used for the HLRNN hyperparameter search), the performance increases with each HLRNN level until getting stuck at level 3. For subject clasification, the performance increases with each HLRNN level until getting stuck at level 2.

4.5 Discussion

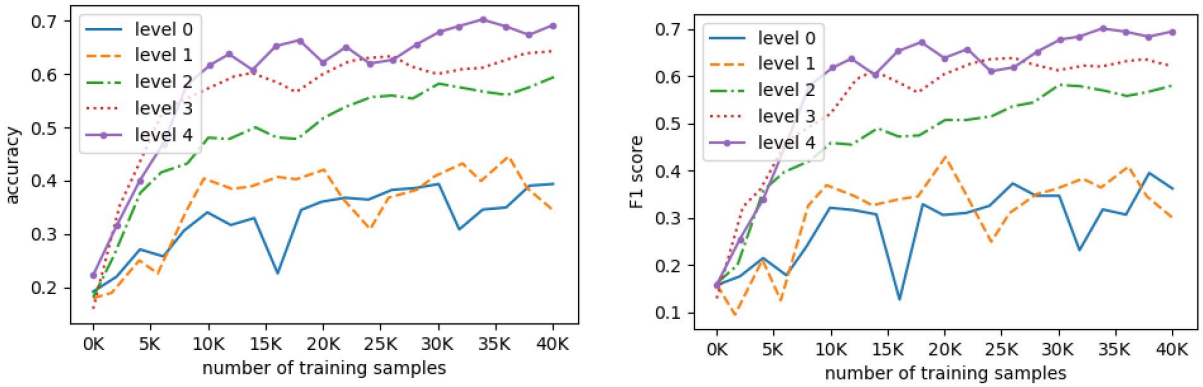
HLRNN is a self-supervised representation learning system for temporal data that combines generative and contrastive techniques to learn to extract representations of the input at different levels of abstraction. Taking into account the types of pretext tasks introduced in Section 4.2, the generative task, which reconstructs the masked/corrupted input and predicts the next inputs, can be seen as a combined masked prediction and bottleneck task. The contrastive task, on the other hand, which pushes representations of successive input samples to be similar, can be seen as an instance discrimination task. The HLRNN architecture draws inspiration from the feedforward circuits of the hierarchies in the neocortex, and, in particular, from the ventral stream of the visual cortex. In this regard, the LRB shows both structural and behavioral similarities with the primary visual cortex (see Section 3.4.3) and is based on ideas from computational neuroscience such as slowness, sparsity, hierarchy, or self-organization. In particular, inspired by the principle of slowness, the HLRNN is designed to pool together patterns that tend to occur



(a) Primitives



(b) Actions



(c) Goals

Figure 4.7: Accuracies and F_1 scores estimated for the synthetic action input and for different classification criteria, HLRNN levels, and number of training batches. For primitive and action clasification, the performance decreases with each HLRNN level. For goal clasification (criterion used for the HLRNN hyperparameter search), the performance increases with each HLRNN level.

close in time, as they probably belong to the same kind of higher-level event and, therefore, have very similar meanings, implementing in this way a form of semantic pooling. This, combined with the hierarchical structure of the network, leads to a temporal pyramid of representations where the lower-level features (i.e., the features that change faster in time) are lost at each level, achieving a slower and higher-level representation as one goes up the hierarchy while having a higher temporal resolution and richer low-level information as one goes down the hierarchy. In this respect, the results obtained show that, for different tasks, different levels of the hierarchy may be more appropriate.

4.5.1 System Mechanisms and Capabilities

The hierarchical slowness, combined with the downsampling temporal max pooling layers, allows the higher layers of the HLRNN to work at slower timescales, reducing the computational requirements of the system and easing longer-term dependencies. This idea of having higher-level layers working at slower timescales is not new and can be found in the literature, especially in works related to brain-inspired systems (e.g., [190]) or to systems related to action recognition (e.g., [204]). Still, how this idea is implemented varies much from system to system, with systems achieving the different timescales through different feedback connectivity patterns (e.g., p[190], [205]), through different types of temporal max pooling layers (e.g., [206], [207]), or simply through downsampling (e.g., [204]). In particular, the authors in [206] propose a learnable dynamic temporal pooling layer that can adapt to how fast the characteristics of the input change, using longer or shorter temporal kernels and downsampling factors accordingly. As a result, the layer downsamples the input while avoiding losing relevant sample information when fast changes occur (note however that most time/frequency information is lost, only keeping the sample order). In this regard, one drawback of the HLRNN is that, when it is sufficiently deep, the downsampling factor can be significant. This can pose challenges when dealing with both fast and slow events or when a certain temporal resolution is needed in terms of when an event finishes and a new one starts. Thus, exploring a similar variable temporal kernel and downsampling factor length within the HLRNN could be a way to approach this limitation. On the other hand, some of the mentioned studies build a temporal pyramid of features and use as input to the classifier a combination of the different level representations (e.g., [204]). While, in this study, we have only used the representation at a single level as input to the classifiers, combining representations from multiple levels to increase the input information (and temporal resolution) is something that could be tested. Alternatively, having classifiers for the same criteria at different levels could be useful to reduce computation by measuring the confidence of the classifiers and directly relying on those at low levels when they are confident enough (e.g., [208]).

Regarding the slowness-oriented contrastive term introduced in this chapter, although it is a quite

simple and straightforward application of the slowness principle, it has been shown to contribute positively to the overall performance of a system that had already slowness mechanisms, probably deserving further investigation. Considering that it is applied over sparse representations, distance metrics other than the squared error term used in this study could be more appropriate for the consecutive sample comparison, such as a cosine similarity-based distance. On the other hand, something that could be studied is whether this loss term (as well as the LRB connectivity pattern) seeks slowness through continuous smooth changes or through short fast changes followed by very stable states (this last case can be seen as somehow analogous to brain resonance). A squared error term seems more likely to promote continuous smooth changes, which may be more appropriate for latent spaces of a more continuous nature, such as those representing motion or location. On the contrary, a solution with stable states connected through fast changes may have better performance for applications in which the data has some form of latent categorical and hierarchical nature, as is the case of action and speech data, allowing in this way a very natural segmentation in time (the motivation is similar to that of quantized latent spaces, as in [209]). This would also fit very well with the dynamic temporal pooling layer commented on in the previous paragraph, and it would make the system better adapted in case it is to be used in combination with some higher-level large language model or knowledge-based (or other symbolic) system such as a plan recognition system. These two different ways of processing temporal information may show analogies with the *what* and *where* pathways of our visual system, where the *what* pathway (ventral stream) is more specialized in object recognition (categorical) and the *where* pathway (dorsal system) is more specialized in object location and motion [210].

One advantage of the HLRNN over other systems dealing with temporal data is that it is very well suited for online learning, as during both the training and prediction phases it only requires the current and surrounding samples. Indeed, even though it makes use of a contrastive loss term, it does not require keeping in memory previous negative samples, and the fact that the generative task is autoregressive also makes online learning more straightforward (as opposed to, e.g., cloze tests, which generally require data from the past or from further in the future). On the other hand, since it is trained stage by stage, low-level representations are learned relatively fast (requiring only small amounts of data when compared with deep systems), and higher-level representations are learned gradually. This can be an advantage in online learning applications where the performance is expected to improve with time and training but in which certain performance is already expected from quite early (e.g., robotics, reinforcement learning, etc.). In addition, note that this stage-by-stage training does not necessarily imply that larger amounts of data are required, as the same data used to train the lower blocks can be reused to train the higher blocks. In a similar direction, something that could be evaluated in our system is how good it is at representing samples of classes it has never seen before by, e.g., training the HLRNN

without specific classes and then including them when training and testing the classifiers.

Another advantage of the employed stage-by-stage training is that it can be combined with a stage-by-stage search, allowing search algorithms to find solutions where the different blocks are characterized by different hyperparameters without the need to deal with too large search spaces. Indeed, as the results have shown, our system leads to considerably better results when the different blocks are not constrained to have the same hyperparameter values, even if the solutions found without such constraints are suboptimal (as, except for the last block, all the block hyperparameters have not been optimized for the final problem, indicating that the results may be further improved). This is again analogous to what has been observed in the neocortex, with the different areas having a similar structure but different properties [155]. As mentioned in Section 4.4.3, a possible way to approach this suboptimal hyperparameter issue is to evaluate the performance at each level using multiple classifiers in a multi-task manner, giving more weight to the performances of those with classification criteria that fit better the abstraction level, as this would probably lead to a hierarchy of more robust and general-purpose representations. Of course, this requires input data labeled according to multiple classification criteria that are preferably ordered in some hierarchical form (as the synthetic input used in this study, labeled attending to primitives, actions, and goals). Although datasets with such characteristics are not very common, some can be found especially in the field of action recognition, such as [211] or [212]. In Chapter 5, such a dataset is used, enabling the use of multiple different classifiers on different levels. Finally, the number of training samples could also be adjusted appropriately according to the classification criterion and to the HLRNN level, with high-level tasks requiring fewer labeled samples at the top levels.

4.5.2 System Limitations

One design limitation of the HLRNN is that it only generates local representations that solely consider current and nearby samples, and not context representations as TST or TS-TCC. This did not seem to be a problem in the results obtained, but, in fact, in those results, context features did not outperform in general local features, suggesting that, for the used datasets, longer-term context information does not contribute much to classification. Therefore, to better understand this HLRNN limitation, the system should be tested on other datasets that require longer-term context information to perform a proper classification. This limitation could be addressed by adding transformers on top of the HLRNN and using, for example, a contrastive loss term similar to that of CPC [200]. One advantage of our system with transformers on top would be that, since the higher-level representations have been considerably downsampled with respect to the input, the transformers would be able to cover longer timespans with the same context length, or, alternatively, they would be able to cover similar timespans with shorter context lengths, making the system cheaper computationally and the attention criteria easier to learn.

The HLRNN also suffers from one of the common limitations of autoencoders and other generative self-supervised systems: The reconstruction and prediction of all variables in the input is given the same weight during training, independently of whether the variables are irrelevant or relevant to the task of interest, and, therefore, their influence on the learned representations is also often similar. This issue can be addressed by adding contrastive pretext tasks designed to focus on specific features of interest of the input data or by including weights in the generative loss term that balance the influence of the variables on the learned representations. Preliminary tests on our system showed that increasing the influence of the hand/attention component of the synthetic action input while decreasing the influence of the environment component (see Appendix 4.A) led to better results.

Finally, another possible limitation of the HLRNN, this time related to dropout, is that the activities of close-by neurons in the 2D layers are correlated (as well as the elements at the output of the temporal window layer corresponding to the same input vector element), which may lead to standard dropout not achieving the expected effect [213]. This could be addressed by applying the dropout to regions of neurons of the 2D input, where the neurons are more strongly correlated (as well as to the corresponding elements at the output of the temporal window layer), instead of just masking random neurons in a naive way (and something similar could be done with the Gaussian noise layer). The same vector elements of consecutive samples are also correlated, but this has already been addressed by keeping a constant dropout mask along each batch. In fact, these correlations within the 2D layers could also be exploited to reduce the computational complexity of the network by making LRB inputs only partially connected, following a pattern similar to that of convolutional or max pooling layers, as was described in Section 3.4.3.

4.5.3 Open Questions

One point that deserves further investigation regarding the HLRNN is when and why a ReLU or tanh activation function should be used. As commented in Section 4.4.2, tanh is generally preferred at recurrent layers, while ReLU is more appropriate to deal with sparse representations. This poses a problem for the locally recurrent layer, as it is a recurrent layer with a sparse activity. Hence, finding a good way to deal with tanh and sparsity may lead to better results for our system. In this study, when using tanh activation functions, we have defined -1 as the value of inactive neurons (by mapping it to 0 in the sparsity loss term). However, a neuron taking a value around -1 will contribute to the activity of its output neurons and will therefore not have the desired behavior that an "off" neuron taking a value of 0 has. How to deal with dropout also poses a problem here, as, if we define -1 as the inactive value, then we should also consider setting the dropped-out neurons to -1 instead of 0. However, this would probably also bring new issues due to the influence of these negative values on their output that would have to be addressed (in

this study, they were set to 0). In any case, it is not clear what the best way to deal with tanh and sparsity is, and further investigation seems necessary to shed light on this issue.

Another question that was left open in Section 4.4 is how to make the HLRNN better at dealing with non-sparse input representations, as the deep LRB at the input of the HLRNN does not seem to be getting the best of the system. One way to address this issue is to investigate other ways of adapting the first LRB to dense input representations. A different approach consists of introducing some module before that first LRB that makes the input sparse. Besides the sparse autoencoder, which did not lead to good results in our preliminary tests, other common ways to do this are using vector quantization, using product quantization, and building (possibly fuzzy) grids representing the ranges where the input variables may be (as those used in the synthetic action input, see Appendix 4.A). These representations have the added advantage over dense representations that they are better suited to deal with uncertainty, as they can represent multiple candidates simultaneously (and weight them appropriately), which is very useful when using generative methods. However, when working with dense representations, these methods tend to generate conservative blurry representations that are a weighted average of the candidates (see [214]). Looking at the brain may also provide some insight into this matter. In the brain, the input sensory information goes through different regions before reaching the neocortex, with these regions performing different types of processing depending on the sensors the information comes from. For example, the subcortical pathway processing auditory information from the ears seems to perform some kind of spectrogram-like processing [182]. Thus, handcrafting modules specifically designed to adapt each kind of input data seems a quite brain-like solution.

4.5.4 Possible Modifications and Extensions

Although in this study we have trained the HLRNN one block at a time in a greedy and self-supervised way, this system could also be trained end-to-end, either in a supervised or self-supervised way. This could be used to fully train or to fine-tune the system, probably requiring some modifications in the architecture such as introducing skip connections and batch or layer normalization layers. In addition, in the case of self-supervised learning, a hierarchical decoder would be required to reconstruct the original input (and possibly the intermediate representations) from the higher-level representations. Similarly to the U-Net architecture [215], such a system would require skip connections between blocks at the same level of the encoder and decoder to recover the low-level high-resolution information, as well as upsampling layers. One advantage of this architecture is that it could probably be used to do more accurate and further-in-time predictions, as it can take advantage of both the higher-level (and slower) contextual information and the lower-level more precise local information. Chapter 5 follows this direction, proposing a U-shaped architecture for action prediction and selection. Similarly, such an architecture could also be used to obtain high-level, high-resolution representations as those in [216].

An alternative approach to the end-to-end self-supervised problem that would still follow the slowness principle could be to rely only on the contrastive task, not asking the system to reconstruct and predict the input anymore and avoiding in this way the need for the decoder. Unlike the HLRNN, such a system would require negative samples during training to avoid trivial solutions. In fact, comparing the HLRNN against a variant of it without the generative part could be interesting to understand how the generative task contributes to its performance. In addition, the recurrent connections in the locally recurrent layer were mainly introduced so that the generative task pushes the learned representations of successive samples to be similar to each other, but the contrastive loss is also designed to generate this effect. Therefore, testing our system without the recurrent connections may also help to understand the effects of those connections. However, this would make the network fully feedforward, without any local context information. A temporal window (or convolutional) layer could be included at the input of each LRB to deal with this. The contrastive loss could also be enhanced with other pretext tasks typical of temporal contrastive systems, such as guessing the distance between samples (positive or negative) or reordering batches with shuffled samples or windows, possibly combined with time series augmentation techniques [199].

How to adapt the HLRNN to work with video data as input is also something that remains to be investigated. As we mentioned in Section 4.2, systems dealing with video data are in general specifically designed for that type of input. In fact, our system could in principle also work with video data, as the locally recurrent layer behaves similarly to a convolutional layer when having a sequence of shifting images as input, as shown in Section 3.4.3. Still, considering the common fast change of low-level features between consecutive frames in video data, we believe that substituting the locally recurrent layers in the low-level blocks of the HLRNN by convolutional layers would improve the behavior of the system. In addition, this modification would prevent these blocks from having to learn connection patterns that are already hardwired in the convolutional layers. Adding some pretext tasks typically used for video would probably also boost its performance [197].

Finally, in this work, we have mainly relied on machine learning techniques to adapt the NILRNN and make it more competent at learning representations of unlabeled temporal data. However, as a neocortex-inspired system, bringing other ideas from the structure and functioning of the brain may also lead to new insights. One possible direction could consist of exploring how to convert the HLRNN into a spiking neural network. These types of brain-inspired networks are gaining interest in recent years due to their information propagation speed, computational efficiency, representational capability, and compatibility with real-world and online learning applications [217]. A direction that is closely related but much more straightforward is that of working with complex-valued neurons instead of real-valued neurons [218]. Hebbian-like learning methods [219] (instead of backpropagation) could also be explored, but this would mean

reconsidering the pretext tasks used so far, as well as introducing architecture-wise modifications to achieve a stable and desirable learning process, as preliminary experiments on the NILRNN showed. Finally, the neurons in the 2D layers could be redistributed as hexagonal grids or lattices instead of as standard square matrices, and the same could be done with the shape of the layers and kernels. This hexagonal distribution has already been evaluated in convolutional neural networks, showing several advantages [220].

4.6 Conclusion

This chapter has presented the hierarchical locally recurrent neural network (HLRNN), which is a self-supervised representation learning system for temporal data that combines well-established techniques from the field of machine learning with ideas and mechanisms from computational neuroscience. The HLRNN consists mainly of a stack of shallow self-supervised learning blocks called LRBs, which are an enhanced version of the NILRNN introduced in Chapter 3. The network is able to learn step by step a hierarchy of representations of the input at different levels of abstraction. These representations can then be used within different types of problems (such as classification), with different level representations being more appropriate depending on the nature of the problem. The HLRNN has been shown to outperform other state-of-the-art self-supervised representation learning systems for temporal data in different classification tasks, placing it as a state-of-the-art system that, in addition, shows other potential advantages, such as its better fit to online learning applications or the possibility to be used as a model of the neocortex (with Section 3.4.3 showing how the NILRNN and the primary visual cortex exhibit an analogous behavior).

Although the results are indeed satisfactory, there are still many directions in which the system can be further investigated and possibly improved, some of them already commented on in Section 4.5. One of these directions is the adaptation of the system to visual input data (maybe by introducing a few convolutional layers at the input) and comparison with other self-supervised video representation learning systems. Another possible direction involves investigating a U-shaped variation of the HLRNN, either to train it end-to-end, to make better and further predictions by using high-level representations as context for low-level predictions, or to get high-level and high-resolution representations. Such U-shaped HLRNN could have transformers or RNNs on top to further improve its performance. This direction is followed in Chapter 5, where a multi-level action prediction and selection system is introduced taking advantage of the context information provided by the higher levels. In addition, the locally recurrent layer and its interaction with the max pooling layer are still not fully understood, and there is room to investigate how these local connections contribute to the self-organization of neurons and to the overall performance of the network. Finally, an analysis of how the HLRNN structure and behavior are analogous to those of the ventral stream of the neocortex (or of other

neocortical hierarchies) may confirm the HLRNN as a valid and useful model of the neocortex.

4.A Appendix. Synthetic Action and Plan Input

This section describes the synthetically generated human action and plan data used as input to obtain some of the results described in Section 4.4. This input is an extension of the synthetic action input described in Section 3.4.1 that also includes higher-level plans: Similarly to the original action input, the synthetic actions and plans occur in a virtual 2D environment containing one hand and four objects (see Fig. 4.8). In the figure, the white rectangle represents the hand, while the four colored squares represent four different objects (with the color indicating the type of object). In this environment, the hand performs a sequence of plans selected randomly out of a set of five possible plans, with the goals of these plans consisting of setting the four objects forming specific shapes: a horizontal line (at any vertical position), a vertical line (at any horizontal position), a square (at any orientation), a triangle (at any orientation), or a random shape (Fig. 4.8 shows an ongoing plan consisting of building a horizontal line). After a number of plans, the environment is reinitialized with four new random objects at random positions. Each of the plans is a sequence of five actions: one action to move each of the objects plus a waiting action at the end of the plan. The possible actions are those described in Section 3.4.1: pick-and-place, push, pull, and wait. The actions that can be performed on each object (i.e., its affordances) depend on the type of object. Each action on an object is selected randomly among the possible ones. The actions are defined as sequences of one or more primitives, where the possible primitives are move empty hand, pick, carry, place, and wait. The object-type identifiers, on the other hand, are 10-element vectors defined randomly following a sparse coding scheme. Finally, to make the task more challenging and realistic, Gaussian noise is added to the object identifiers and to the hand and object positions at each frame (this is different from the data in Section 3.4.1, where the noise was added to the final sparse representation).

The input data used in Section 4.4 consists of a set of handcrafted features extracted from this 2D environment. To extract these features, the input data is first low-pass filtered, and this filtered data is used to generate the two components of the input: The first component provides information about the distribution of the objects in the environment and consists of a 10×10 matrix representing the 2D environment and containing a Gaussian centered at the position of each object. The second component (which is the one described in Section 4.4) provides information about the type of the object of attention, the position of the object of attention with respect to the hand, the velocities of the hand and object of attention, and how open the hand is. To obtain it, the velocities of the hand and objects are first estimated. Taking into account the velocities of the objects and the velocity and trajectory of the hand, an attention weight is calculated for each object. The type of the object of attention is obtained as the weighted sum of the object-type identifiers. The position of the object of attention with respect to the hand is

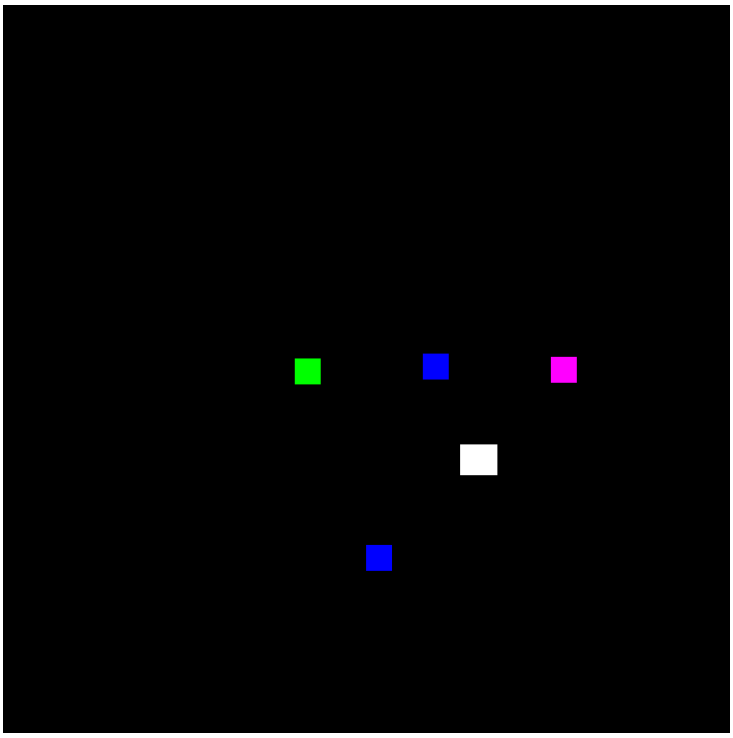


Figure 4.8: Virtual 2D environment in which the synthetic actions and plans take place. The white rectangle represents the hand. The four colored squares represent four different objects. Different colors represent different types of objects.

represented using a 5×5 matrix, with the center of the matrix representing the position of the hand. The matrix contains the weighted sum of four Gaussians indicating the relative position of each object. The mapping of positions is done in a non-linear way, with distances closer to the hand showing higher resolutions. The velocities of the hand and object of attention are represented in a similar way but using 3×3 matrices instead. Finally, a 2-element vector is used to represent how open or closed the hand is. This leads to an input of size $100 + 55 = 155$ that takes a sparse coding form.

4.B Appendix. Hyperparameters

This section reports the hyperparameters found during search for all systems and datasets in this study. In particular, Tables 4.3, 4.4, and 4.5 show the hyperparameters found for the standard HLRNN (one table per dataset), Table 4.6 shows the hyperparameters found for TST, Table 4.7 shows the hyperparameters found for TS-TCC, and Tables 4.8, 4.9, and 4.10 show the hyperparameters found for the different HLRNN variants (one table per dataset). In the HLRNN tables, each row corresponds to the hyperparameters of a block, characterized by its name ("LRB" or classifier classification name) and level of the hierarchy, whith "tail" referring to the input block formed of the downsampling and temporal window layers. The HLRNN level 5 classifiers and the TST and TS-TCC classifiers are linear classifiers, while the rest are shallow

Table 4.3: Hyperparameters for the deep input HLRNN applied over the WARD dataset, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	γ
tail	15	1	-	-	-	-	-	-	-	-	-	-
LRB 1	1	3	484	1	9	1	255	61	4.62e-4	6.64e-4	6.31e-3	3.28e-5
LRB 2	7	4	625	25	69	1	-	28	2.59e-3	2.87e-3	5.20e-4	770e-4
LRB 3	4	1	225	37	5	1	-	57	1.09e-3	4.04e-5	2.67e-2	3.55e-3
LRB 4	1	1	676	9	1	1	-	80	1.23e-3	3.28e-2	1.80e-2	2.40e-3
LRB 5	4	3	529	21	1	2	-	39	1.04e-4	1.72e-3	1.58e-1	1.59e-3
actions 0	-	-	-	-	-	-	255	134	5.86e-5	4.72e-2	-	-
actions 1	-	-	-	-	-	-	64	121	1.64e-3	2.48e-3	-	-
actions 2	-	-	-	-	-	-	55	185	2.19e-3	2.21e-2	-	-
actions 3	-	-	-	-	-	-	91	44	1.63e-3	1.55e-3	-	-
actions 4	-	-	-	-	-	-	50	44	1.63e-3	5.31e-3	-	-
actions 5	-	-	-	-	-	-	-	-	5.95e-2	3.25e-4	-	-
subjects 0	-	-	-	-	-	-	46	190	2.93e-4	1.15e-5	-	-
subjects 1	-	-	-	-	-	-	102	255	2.24e-3	8.46e-3	-	-
subjects 2	-	-	-	-	-	-	121	35	5.30e-4	5.49e-4	-	-
subjects 3	-	-	-	-	-	-	118	8	1.37e-4	1.00e-5	-	-
subjects 4	-	-	-	-	-	-	72	143	2.71e-3	8.78e-5	-	-

vanilla recurrent neural networks. Since TST and TS-TCC deal with multiple sequences per batch, their batch size parameter refers to the number of sequences per batch, and the number of samples per sequence is given by the sequence length. For the HLRNN and for the classifiers, the batch size parameter refers to the number of samples per batch or sequence. In particular, in the case of the LRBs, the batch size parameter refers to the number of samples at the layer that is being trained, i.e., corresponds to a number of samples at the input of the HLRNN that is larger than the parameter by the downsampling factor.

Table 4.4: Hyperparameters for the standard HLRNN applied over the FSDD dataset, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	γ
tail	14	2	-	-	-	-	-	-	-	-	-	-
LRB 1	15	2	169	69	5	1	-	45	3.28e-4	2.07e-5	1.95e-2	1.98e-2
LRB 2	5	2	441	1	9	1	-	14	5.69e-4	2.33e-3	3.83e-2	1.10e-3
LRB 3	2	1	576	13	1	1	-	17	2.20e-4	3.57e-4	5.59e-3	7.09e-5
LRB 4	3	1	361	25	1	1	-	78	1.69e-4	3.20e-2	8.86e-2	5.35e-3
LRB 5	2	1	529	1	1	1	-	51	1.61e-3	4.00e-3	1.89e-3	1.14e-3
digits 0	-	-	-	-	-	-	160	79	7.78e-4	3.56e-2	-	-
digits 1	-	-	-	-	-	-	155	21	6.26e-4	1.25e-3	-	-
digits 2	-	-	-	-	-	-	66	31	1.72e-3	1.55e-3	-	-
digits 3	-	-	-	-	-	-	96	56	1.00e-3	4.09e-4	-	-
digits 4	-	-	-	-	-	-	44	47	3.85e-3	6.44e-5	-	-
digits 5	-	-	-	-	-	-	-	-	1.23e-2	3.03e-3	-	-
subjects 0	-	-	-	-	-	-	42	255	4.26e-4	1.00e-5	-	-
subjects 1	-	-	-	-	-	-	22	255	1.72e-2	1.16e-5	-	-
subjects 2	-	-	-	-	-	-	33	112	4.00e-3	1.74e-5	-	-
subjects 3	-	-	-	-	-	-	43	160	6.96e-3	1.00e-5	-	-
subjects 4	-	-	-	-	-	-	91	255	7.27e-3	6.45e-5	-	-

Table 4.5: Hyperparameters for the standard HLRNN applied over the synthetic action input, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	γ
tail	3	2	-	-	-	-	-	-	-	-	-	-
LRB 1	4	6	400	45	13	1	-	57	4.05e-3	1.69e-3	5.63e-3	7.90e-3
LRB 2	6	2	529	9	1	1	-	123	3.22e-3	5.41e-3	3.73e-2	7.84e-4
LRB 3	3	1	529	5	1	1	-	62	8.55e-4	3.87e-4	6.20e-4	7.75e-2
LRB 4	3	1	361	21	1	1	-	148	2.96e-3	2.43e-4	1.54e-2	4.14e-2
LRB 5	5	1	576	49	1	1	-	53	6.50e-5	2.68e-4	1.20e-0	9.44e-3
primitives 0	-	-	-	-	-	-	255	8	8.40e-4	1.27e-4	-	-
primitives 1	-	-	-	-	-	-	45	8	4.08e-3	1.38e-2	-	-
primitives 2	-	-	-	-	-	-	16	18	7.51e-3	1.00e-0	-	-
primitives 3	-	-	-	-	-	-	255	43	6.47e-3	2.89e-2	-	-
primitives 4	-	-	-	-	-	-	19	248	8.49e-2	1.27e-1	-	-
actions 0	-	-	-	-	-	-	255	11	5.91e-4	2.16e-5	-	-
actions 1	-	-	-	-	-	-	23	8	3.43e-3	5.41e-3	-	-
actions 2	-	-	-	-	-	-	30	8	2.18e-3	4.82e-4	-	-
actions 3	-	-	-	-	-	-	16	32	9.49e-3	4.06e-2	-	-
actions 4	-	-	-	-	-	-	32	255	1.16e-2	1.37e-4	-	-
goals 0	-	-	-	-	-	-	255	255	7.86e-4	5.01e-5	-	-
goals 1	-	-	-	-	-	-	157	67	2.40e-3	1.99e-3	-	-
goals 2	-	-	-	-	-	-	46	19	5.51e-3	8.32e-5	-	-
goals 3	-	-	-	-	-	-	17	32	1.00e-2	1.93e-4	-	-
goals 4	-	-	-	-	-	-	56	17	4.54e-3	1.21e-4	-	-
goals 5	-	-	-	-	-	-	-	-	8.19e-2	2.01e-2	-	-

Table 4.6: Hyperparameters for TST applied over the different datasets, where d is the transformer model size, h the number of heads in the transformer, s the size of the hidden layers of the transformer feedforward networks, N the number of attention layers, p the masking probability of the dropout layer, r_m the proportion of masked elements in the input, l_m the average length of the masked subsequences in the input, L the sequence length, m the batch size, α the Adam stepsize, and λ the L2 regularization term weight.

	Representation learning system										**Classification system**	
Dataset	d	h	s	N	p	r_m	l_m	L	m	α	α	λ
WARD	440	11	203	2	2.39e-1	2.06e-3	2.03	159	88	6.56e-5	2.50e-2	7.24e-4
FSDD	418	11	175	2	9.71e-2	1.11e-5	1.46	37	107	3.59e-5	7.80e-3	1.04e-4
Synthetic actions	138	6	96	1	2.90e-1	5.00e-1	5.52	202	21	2.21e-3	2.22e-2	2.05e-3

Table 4.7: Hyperparameters for TS-TCC applied over the different datasets, where s is the size of the encoder output, d the transformer model size, f the size of the input convolution layer kernel, t_p the stride of the input convolution layer, δ the number of timesteps in the temporal contrasting task, p the masking probability of the dropout layer, j_w the jitter ratio of the weak augmentation, j_s the jitter ratio of the strong augmentation, $n_{max,s}$ the maximum number of segments in the strong augmentation, L the sequence length, m the batch size, α the Adam stepsize, τ the temperature of the contextual contrastive loss term, λ_2 the contextual contrastive loss term weight, and λ the L2 regularization term weight.

	Representation learning system													**Classification system**	
Dataset	s	d	f	t_p	δ	p	j_w	j_s	$n_{max,s}$	L	m	α	τ	λ_2	α	λ
Local repr. z:																
WARD	157	36	1	5	2	3.79e-1	3.61e-1	1.57e-1	2	35	115	1.35e-2	2.74e-4	1.19e-3	5.06e-2	9.24e-4
FSDD	236	84	1	1	6	2.63e-1	1.39e-3	1.21e-2	5	139	34	1.48e-5	4.24e-3	1.01e-4	1.55e-1	5.87e-5
Synthetic actions	255	176	1	1	10	2.30e-1	2.41e-4	8.46e-4	2	207	15	8.45e-4	1.76e-4	2.03e-4	5.00e-3	1.21e-5
Context repr. c:																
WARD	53	116	1	1	8	3.73e-1	1.78e-1	1.83e-2	2	55	112	1.72e-4	1.45e-2	1.85e-2	1.02e-3	3.50e-3
FSDD	255	220	4	7	1	4.54e-1	1.79e-1	3.10e-2	2	18	72	2.01e-4	1.68e-3	4.69e-3	3.77e-4	4.49e-4
Synthetic actions	138	64	2	5	4	4.54e-1	7.83e-2	9.88e-4	3	112	32	1.07e-3	6.57e-3	4.63e-3	6.46e-3	4.39e-2

Table 4.8: Hyperparameters for the HLRNN variants applied over the WARD dataset, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, ρ the desired sparsity in the locally recurrent layer, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	ρ	γ
(w/o) deep input:													
tail	5	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	3	576	9	21	1	-	110	2.36e-3	6.18e-3	1.67e-2	-	1.07e-2
LRB 2	12	2	225	69	9	1	-	37	4.47e-4	5.56e-2	3.40e-1	-	2.06e-3
LRB 3	4	2	256	5	5	1	-	40	3.10e-3	7.05e-3	6.85e-3	-	8.47e-3
LRB 4	4	1	361	21	21	1	-	71	6.32e-4	6.24e-3	6.72e-1	-	8.26e-2
actions 4	-	-	-	-	-	-	141	25	3.87e-4	4.00e-2	-	-	-
w/o temp pool:													
tail	3	5	-	-	-	-	-	-	-	-	-	-	-
LRB 1	-	-	169	9	1	1	143	29	2.22e-4	3.21e-5	5.84e-3	-	1.17e-4
LRB 2	-	-	729	1	21	1	-	16	7.23e-5	1.73e-4	9.57e-4	-	3.83e-4
LRB 3	-	-	576	5	21	1	-	92	6.15e-4	7.11e-5	4.96e-3	-	6.37e-5
LRB 4	-	-	841	1	9	1	-	27	2.51e-4	2.00e-4	4.45e-1	-	5.44e-3
actions 4	-	-	-	-	-	-	219	69	2.35e-4	6.25e-5	-	-	-
w/o max pool:													
tail	15	4	-	-	-	-	-	-	-	-	-	-	-
LRB 1	2	1	64	5	-	-	88	84	8.45e-4	1.26e-3	2.24e-2	-	6.69e-2
LRB 2	5	4	625	5	-	-	-	74	5.69e-5	3.61e-4	2.23e-2	-	1.09e-1
LRB 3	1	1	64	5	-	-	-	42	2.61e-4	1.49e-2	4.88e-2	-	1.40e-3
LRB 4	8	1	256	1	-	-	-	148	1.33e-4	8.77e-2	9.91e-2	-	3.90e-1
actions 4	-	-	-	-	-	-	32	95	1.47e-2	2.22e-4	-	-	-
w/o dropout/noise:													
tail	8	3	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	1	441	1	29	2	237	142	1.70e-4	1.80e-5	5.39e-2	-	1.90e-3
LRB 2	15	1	529	1	21	1	-	11	9.97e-4	3.38e-5	7.86e-4	-	1.54e-3
LRB 3	1	1	484	5	5	1	-	38	8.73e-5	1.10e-3	6.35e-2	-	3.43e-4
LRB 4	1	4	961	1	21	1	-	28	1.04e-3	3.83e-5	2.14e-2	-	9.41e-5
actions 4	-	-	-	-	-	-	54	48	1.67e-3	1.17e-3	-	-	-
square kernel:													
tail	15	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	14	2	100	25	25	2	207	63	9.85e-4	1.23e-2	3.56e-2	-	3.45e-1
LRB 2	13	1	81	1	9	1	-	11	3.02e-3	1.77e-3	6.33e-4	-	3.22e-4
LRB 3	6	1	256	9	9	1	-	227	9.62e-3	8.89e-3	5.72e-3	-	7.31e-4
LRB 4	2	1	841	9	9	1	-	31	1.31e-3	8.03e-4	9.29e-2	-	3.63e-3
actions 4	-	-	-	-	-	-	163	31	3.22e-4	1.53e-3	-	-	-
w/o contrastive:													
tail	7	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	4	3	676	45	1	1	82	43	3.10e-4	7.27e-4	2.89e-3	-	-
LRB 2	10	4	196	1	21	1	-	13	4.93e-4	3.98e-3	6.54e-3	-	-
LRB 3	1	2	289	21	5	1	-	36	3.24e-4	1.68e-3	3.63e-2	-	-
LRB 4	1	1	289	1	9	1	-	19	2.36e-3	1.25e-2	2.45e-3	-	-
actions 4	-	-	-	-	-	-	46	57	7.10e-3	6.39e-4	-	-	-
sigmoid:													
tail	15	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	11	5	196	1	1	1	20	190	8.63e-4	4.74e-5	1.97e-3	4.65e-3	9.13e-5
LRB 2	2	2	144	1	5	1	-	255	1.27e-1	6.14e-3	1.78e-3	4.31e-2	4.05e-1
LRB 3	2	1	144	109	5	1	-	136	1.07e-2	1.00e-5	2.19e-2	4.46e-2	3.98e-3
LRB 4	1	1	256	1	9	1	-	171	1.34e-2	1.94e-5	1.01e-2	1.19e-3	4.98e-3
actions 4	-	-	-	-	-	-	23	106	4.12e-3	1.01e-4	-	-	-
tanh:													
tail	8	3	-	-	-	-	-	-	-	-	-	-	-
LRB 1	4	2	729	45	1	1	33	144	3.09e-3	9.91e-5	2.81e-4	1.27e-4	4.86e-3
LRB 2	6	3	81	9	1	1	-	23	7.97e-4	3.32e-2	1.00e-5	1.75e-1	4.99e-5
LRB 3	3	1	529	9	1	1	-	66	2.60e-4	3.63e-4	1.02e-4	1.11e-1	9.34e-3
LRB 4	1	2	484	1	69	2	-	22	2.00e-3	3.99e-2	1.54e-1	1.22e-2	2.50e-4
actions 4	-	-	-	-	-	-	30	43	8.21e-3	2.97e-2	-	-	-
shared params:													
tail	3	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1 - 4	1	2	484	1	5	1	(155)	104	2.21e-4	9.37e-5	2.03e-2	-	3.92e-3
actions 4	-	-	-	-	-	-	48	49	9.07e-3	3.05e-4	-	-	-

Table 4.9: Hyperparameters for the HLRNN variants applied over the FSDD dataset, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, ρ the desired sparsity in the locally recurrent layer, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	ρ	γ
(w/o) deep input:													
tail	8	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	14	1	256	1	9	1	28	8	3.48e-4	5.94e-3	1.32e-3	-	3.21e-3
LRB 2	7	4	225	5	5	1	-	31	9.41e-4	1.93e-3	6.06e-2	-	7.89e-3
LRB 3	2	1	361	61	5	1	-	36	1.36e-3	9.85e-3	1.02e-1	-	1.26e-3
LRB 4	1	1	484	121	5	1	-	61	2.98e-3	2.24e-3	1.46e-2	-	1.17e-2
digits 4	-	-	-	-	-	-	174	46	4.10e-4	1.25e-4	-	-	-
w/o temp pool:													
tail	6	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	-	-	484	1	29	2	-	9	4.58e-4	6.03e-3	2.48e-2	-	1.72e-4
LRB 2	-	-	225	9	9	1	-	34	4.45e-4	3.37e-3	1.01e-1	-	6.56e-4
LRB 3	-	-	196	5	13	1	-	181	9.45e-4	4.12e-1	6.22e-1	-	1.56e-4
LRB 4	-	-	256	5	9	1	-	28	1.49e-4	1.44e-3	1.77e-0	-	4.69e-3
digits 4	-	-	-	-	-	-	112	101	3.72e-4	2.75e-5	-	-	-
w/o max pool:													
tail	13	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	15	2	144	69	-	-	-	66	9.50e-4	3.11e-3	1.59e-2	-	3.42e-2
LRB 2	6	1	324	21	-	-	-	17	1.65e-4	4.92e-4	2.44e-2	-	7.71e-3
LRB 3	5	1	400	5	-	-	-	29	3.59e-4	5.03e-5	9.98e-4	-	5.66e-5
LRB 4	3	1	841	21	-	-	-	79	3.53e-4	5.34e-4	4.36e-3	-	3.63e-3
digits 4	-	-	-	-	-	-	28	129	4.73e-3	4.06e-4	-	-	-
w/o dropout/noise:													
tail	12	5	-	-	-	-	-	-	-	-	-	-	-
LRB 1	15	4	196	25	1	1	-	58	2.54e-3	2.20e-3	7.01e-2	-	3.48e-4
LRB 2	2	1	961	21	9	1	-	58	9.77e-5	2.63e-4	5.82e-2	-	1.50e-4
LRB 3	1	1	196	21	5	1	-	50	2.36e-4	2.45e-3	4.65e-2	-	1.41e-4
LRB 4	1	1	784	5	9	1	-	12	9.89e-4	8.06e-4	1.85e-3	-	7.62e-4
digits 4	-	-	-	-	-	-	49	15	9.32e-4	5.74e-5	-	-	-
square kernel:													
tail	3	3	-	-	-	-	-	-	-	-	-	-	-
LRB 1	15	1	441	1	9	5	-	13	1.61e-4	2.04e-2	1.41e-3	-	9.29e-3
LRB 2	1	1	441	81	1	1	-	59	1.46e-4	1.78e-4	1.79e-3	-	1.07e-3
LRB 3	8	1	238	1	9	1	-	45	8.90e-4	2.89e-4	1.57e-3	-	9.41e-4
LRB 4	3	3	169	49	1	1	-	18	4.14e-4	5.16e-3	4.73e-3	-	2.87e-3
digits 4	-	-	-	-	-	-	71	122	1.88e-3	1.04e-2	-	-	-
w/o contrastive:													
tail	5	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	4	6	225	1	5	1	-	36	1.24e-3	3.66e-4	1.53e-3	-	-
LRB 2	8	1	361	5	37	1	-	92	2.89e-3	1.39e-2	1.99e-1	-	-
LRB 3	2	3	529	9	5	1	-	157	6.75e-4	4.13e-3	5.09e-2	-	-
LRB 4	1	1	484	9	9	1	-	38	7.71e-4	1.16e-3	2.00e-2	-	-
digits 4	-	-	-	-	-	-	70	39	1.20e-3	6.54e-3	-	-	-
sigmoid:													
tail	7	4	-	-	-	-	-	-	-	-	-	-	-
LRB 1	11	3	729	1	5	1	-	33	2.40e-3	2.49e-4	2.39e-2	2.40e-3	6.12e-2
LRB 2	1	1	100	5	1	1	-	89	1.19e.2	1.09e-5	1.89e-4	1.20e-2	1.44e-3
LRB 3	1	1	441	61	9	1	-	93	5.95e-3	4.56e-5	9.04e-3	6.95e-5	3.33-5
LRB 4	3	1	961	1	1	1	-	12	1.26e-3	7.61e-5	5.01e-3	7.39e-3	1.21e-5
digits 4	-	-	-	-	-	-	48	206	5.03e-3	1.53e-3	-	-	-
tanh:													
tail	8	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	14	1	256	9	1	1	-	90	8.35e-5	9.25e-3	2.95e-4	1.73e-3	1.95e-3
LRB 2	2	1	256	1	1	1	-	61	1.66e-4	9.44e-3	1.26e-4	6.21e-3	8.70e-3
LRB 3	1	2	676	1	1	1	-	105	3.37e-2	1.13e-4	1.97e-3	4.62e-2	7.36e-5
LRB 4	3	1	576	5	1	1	-	45	2.38e-3	7.48e-3	5.24e-3	1.21e-3	1.18e-3
digits 4	-	-	-	-	-	-	55	190	5.76e-3	1.51e-3	-	-	-
shared params:													
tail	6	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1 - 4	13	1	400	25	1	1	-	22	4.08e-5	4.92e-4	3.12e-4	-	1.43e-4
digits 4	-	-	-	-	-	-	146	104	5.05e-4	1.83e-4	-	-	-

Table 4.10: Hyperparameters for the HLRNN variants applied over the synthetic action input, where ω is the window size of the temporal layers, $\eta_{\downarrow}$ the downsampling factor of the downsampling layers, s_r the size of the locally recurrent layer, f_r the size of the locally recurrent layer kernel, f_p the size of the 2D max pooling layer kernel, t_p the stride of the 2D max pooling layer, s_h the size of the hidden layer, m the batch size, α the Adam stepsize, λ the L2 regularization term weight, β the sparsity term weight, ρ the desired sparsity in the locally recurrent layer, and γ the slowness-oriented contrastive term weight.

Block	ω	$\eta_{\downarrow}$	s_r	f_r	f_p	t_p	s_h	m	α	λ	β	ρ	γ
(w/o) deep input:													
tail	1	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	3	1	324	29	1	1	42	24	3.16e-4	5.81e-4	6.15e-3	-	1.42e-2
LRB 2	2	4	576	1	9	1	-	122	1.21e-2	1.42e-5	2.19e-3	-	4.01e-5
LRB 3	3	1	400	1	9	1	-	158	2.19e-4	5.72e-3	2.97e-0	-	2.41e-5
LRB 4	9	5	729	21	5	1	-	27	5.10e-3	9.56e-3	5.02e-2	-	4.68e-3
goals 4	-	-	-	-	-	-	95	190	8.13e-3	1.39e-3	-	-	-
w/o temp pool:													
tail	2	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	-	-	961	9	9	1	-	37	3.71e-4	1.11e-2	2.41e-1	-	4.51e-4
LRB 2	-	-	196	13	1	1	-	8	1.05e-3	2.70e-4	7.36e-4	-	7.23e-5
LRB 3	-	-	961	49	1	1	-	67	3.35e-4	1.13e-4	1.57e-2	-	2.01e-3
LRB 4	-	-	225	1	1	1	-	88	3.28e-4	1.22e-2	3.22e-2	-	1.12e-3
goals 4	-	-	-	-	-	-	78	135	5.58e-3	3.07e-5	-	-	-
w/o max pool:													
tail	1	3	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	1	961	5	-	-	-	87	6.89e-3	2.94e-3	1.23e-2	-	7.71e-3
LRB 2	6	1	625	81	-	-	-	76	1.55e-3	8.33e-3	1.97e-2	-	3.06e-4
LRB 3	7	1	238	1	-	-	-	255	1.78e-2	3.45e-5	1.33e-1	-	4.95e-2
LRB 4	5	5	144	5	-	-	-	77	1.51e-3	8.34e-4	1.33e-2	-	3.75e-3
goals 4	-	-	-	-	-	-	16	12	9.36e-3	9.10e-5	-	-	-
w/o dropout/noise:													
tail	11	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	6	400	29	1	1	-	120	1.87e-3	9.38e-5	7.18e-3	-	1.76e-4
LRB 2	6	4	400	45	1	1	-	24	7.61e-4	6.86e-3	3.57e-2	-	4.14e-2
LRB 3	3	1	324	9	1	1	-	55	1.64e-4	9.67e-3	4.08e-1	-	1.58e-2
LRB 4	1	1	400	113	5	1	-	184	1.92e-4	1.15e-4	5.12e-1	-	9.55e-2
goals 4	-	-	-	-	-	-	66	8	1.99e-3	4.12e-3	-	-	-
square kernel:													
tail	2	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	2	2	144	25	1	1	-	25	5.86e-4	1.58e-3	9.46e-3	-	2.94e-1
LRB 2	6	1	625	9	9	1	-	50	1.15e-2	1.45e-2	1.02e-2	-	2.41e-2
LRB 3	5	4	729	9	1	1	-	84	1.81e-4	1.29e-3	4.35e-2	-	1.41e-3
LRB 4	15	4	576	81	1	1	-	11	1.31e-4	3.07e-3	6.17e-2	-	2.12e-3
goals 4	-	-	-	-	-	-	56	245	5.39e-2	2.34e-3	-	-	-
w/o contrastive:													
tail	1	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	2	4	961	13	29	1	-	48	5.97e-3	1.37e-3	2.68e-2	-	-
LRB 2	1	2	729	121	5	1	-	72	3.18e-4	1.96e-4	9.09e-1	-	-
LRB 3	11	3	441	21	1	1	-	55	1.88e-4	1.46e-2	2.78e-1	-	-
LRB 4	1	2	441	21	9	1	-	130	4.45e-4	4.93e-4	2.66e-1	-	-
goals 4	-	-	-	-	-	-	74	14	5.72e-3	1.05e-1	-	-	-
sigmoid:													
tail	1	1	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	2	400	21	37	1	-	22	3.94e-2	1.32e-5	3.42e-2	1.86e-2	2.91e-4
LRB 2	3	2	529	5	1	1	-	80	1.56e-2	1.06e-3	7.48e-2	5.96e-2	1.96e-3
LRB 3	1	1	529	1	5	1	-	177	1.13e-1	4.46e-5	9.51e-3	1.11e-2	1.54e-2
LRB 4	3	3	121	1	1	1	-	69	4.12e-3	1.42e-3	1.32e-1	3.29e-3	1.87e-1
goals 4	-	-	-	-	-	-	31	8	3.31e-3	4.32e-2	-	-	-
tanh:													
tail	3	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1	1	1	100	9	1	1	-	22	1.63e-4	1.65e-2	4.91e-3	1.28e-5	1.16e-3
LRB 2	4	1	961	1	21	1	-	151	3.57e-4	7.86e-1	1.00e-1	4.48e-2	2.66e-4
LRB 3	5	2	484	5	1	1	-	31	8.42e-4	6.10e-2	1.15e-3	9.26e-3	2.23e-3
LRB 4	9	4	256	21	1	1	-	247	3.64e-3	1.30e-4	6.63e-3	1.15e-3	4.43e-1
goals 4	-	-	-	-	-	-	58	14	4.84e-3	4.79e-4	-	-	-
shared params:													
tail	1	2	-	-	-	-	-	-	-	-	-	-	-
LRB 1 - 4	11	1	324	5	1	1	-	14	7.84e-4	4.99e-3	2.28e-2	-	3.68e-2
goals 4	-	-	-	-	-	-	36	220	1.14e-2	5.85e-3	-	-	-

Chapter 5

U-LRNN: TOWARDS A U-SHAPED LOCALLY RECURRENT NEURAL NETWORK-BASED COGNITIVE ARCHITECTURE FOR ACTION AND GOAL RECOGNITION, PREDICTION, AND SELECTION

5.1 Introduction

The term *cognitive architecture* refers to a general-purpose model of cognition (of a natural or artificial agent), which is able to describe or perform multiple cognitive tasks at different levels and domains, such as perception, learning, reasoning, action selection, or attention [221], [222] (see [123] for a comprehensive list). These cognitive architectures are used both to model, simulate, and gain a deeper understanding of the human mind and to build artificial intelligent systems. Within the field of artificial intelligence, a cognitive architecture takes the form of a computational model that has the purpose of endowing an artificial system with some form of general-purpose intelligence to some extent analogous to that of humans (often including analogous internal mechanisms and developmental processes), making it able to understand, interact with, learn from, and adapt to its environment while seeking to achieve a number of goals. Such models are particularly useful when building multi-purpose autonomous agents such as elderly care robots or virtual assistants.

Existing cognitive architectures are in general based on different principles and theories, take different approaches, and seek different purposes, making their design and function very diverse. This makes them hard to compare and even to clearly delimit what a cognitive architecture is. Still, multiple criteria have been defined to classify and evaluate them according to what capabilities they have, what methods they use to achieve those capabilities, and how they perform on different tasks (e.g., [123]). One popular classification criterion is according to how the information is represented, distinguishing among symbolic, subsymbolic, and hybrid architectures. Symbolic architectures typically take a higher-level approach to cognition (similar to that of cognitive science and computationalism), trying to directly describe and model the known cognitive processes, and are often referred to as cognitivist architectures. Subsymbolic architectures, on the other hand, generally take a more bottom-up approach (similar to that of cognitive neuroscience and connectionism), working with a large number of connected nodes (e.g., neurons) and relying on the emergence of cognitive properties, and they are often also referred to as connectionist or emergent architectures. Finally, hybrid architectures combine both types of representations, being this combination implemented in very different ways depending on the architecture.

In this chapter, we propose some directions on how to expand the hierarchical locally recurrent neural network (HLRNN) introduced in Chapter 4 to transform it into a connectionist cognitive architecture. These directions are both inspired by well-known mechanisms of the human brain and by established machine learning techniques. The HLRNN is a brain-inspired neural network that learns, in a self-supervised and potentially online way, representations of its temporal input data at different levels of abstraction. It has mainly been used within sequential transfer learning pipelines, using its learned representations to make the subsequent classifier training process simpler and faster. While classification is a very specific task that is far from the more general-purpose system that we are looking for, the HLRNN was designed to some extent as a model of the neocortex and with the idea in mind of extending it to a more general-purpose system. Following this idea, we have expanded it with a decoder-like hierarchy that mimics to some extent the feedback connections in the hierarchies of the neocortex [153] and which can be used for prediction and decision making, as well as with a self-supervised attention learning system at its input, and we have tested it on the tasks of action and goal recognition, prediction, and selection (unfortunately, most results are not available yet). The resultant system, known as the U-shaped Locally Recurrent Neural Network (U-LRNN), is arguably not yet a full cognitive architecture, as it lacks common skills of these architectures such as long-term planning or metacognition and it has only been tested for a set of tasks within a same domain and with a single input modality. Still, it already shows the potential of this network architecture to be expanded to more general-purpose systems. Some ideas on how to continue expanding the system in this direction are also proposed.

This chapter is organized as follows: Section 5.2 describes the main input data used in this chapter, its adaptation to our system (which includes the new attention network), the full U-LRNN architecture, and its usage for the tasks of action and goal recognition, prediction, and selection. Section 5.3 presents the available results. Section 5.4 discusses on the system architecture and proposes some directions on how to further expand it to a more general-purpose cognitive architecture. Finally, Section 5.5 summarizes and concludes the chapter.

5.2 Methods

This section describes the different elements employed along this chapter: First, it introduces the action and plan dataset mainly used in this study and the modifications carried out to adapt the data to our system. Then, it presents a new self-supervised attention learning system that converts the variable-size data coming from the dataset to a fixed-size data. Finally, it presents the U-LRNN architecture, describing how to adapt it to the tasks of action and goal recognition, prediction, and selection. Note that, in this study, by plan we refer to a sequence of actions to achieve a specific goal, i.e., plans have a structure, while actions and goals can be defined by a single label.

5.2.1 The Input to the System

Along this chapter, the Extended KIT Bimanual Manipulation Dataset [223] has been mainly used as input to our system. This section introduces this human action and plan dataset and the set of procedures carried out to make the dataset better adapted to our system and problem. Procedures to adapt the data to the tasks of action and goal recognition, prediction, and selection, making these tasks more challenging, include defining and capturing a number of new recordings and implementing a set of data augmentations. Procedures to adapt the data to our system input include the definition of a sparse representation of the data and the development of a self-supervised attention system designed to learn to extract attention weights and focus on the more relevant data (described in Section 5.2.2).

The Extended KIT Bimanual Manipulation Dataset

The Extended KIT Bimanual Manipulation Dataset [223] consists of a number of recordings of subjects performing different kitchen activities. Some of these recordings take the form of short sequences in which the subject performs specific actions such as peeling or stirring [211], while others are longer sequences where the subject performs plans composed of a number of actions, such as preparing a salad [223]. The dataset is multi-modal, counting on data coming from a fixed RGB camera, an egocentric RGB camera, three fixed RGB-D cameras, a passive marker-based optical motion capture system (which tracks the subject and the objects), two motion capture gloves, and three IMUs. All the data is adequately processed and cleaned, and segmented and labeled according to two criteria per hand (i.e., four criteria in total) that we will refer to from now on as subactions and primitives, with the second one being a subsegmentation of the first one. This results in a hierarchy of classification criteria that, from top to bottom, consists of goals (describing complete long sequences), actions (describing complete short sequences), subactions (one per hand), and primitives (one per hand). This is shown in Fig. 5.1 (note that some subactions are not further segmented into shorter primitives because they are also considered primitives themselves). In addition, the subaction annotation includes information on the objects involved, leading to few extra classification criteria.

This dataset has a number of advantages to test our system that motivated its selection. For example, it counts on recordings of complete plans labeled at different levels of abstraction, and these recordings have explicit position and orientation information not only of the subject body parts but also of the objects involved. Still, the dataset was mainly designed for tasks such as imitation learning and human motion analysis. For this reason, the dataset presents some limitations on how challenging it is for an action and goal recognition task, making it, in principle, not so appropriate to test our system. To approach this issue, a number of new recordings have been described and captured, and several augmentations have been implemented, which are described in the following sections. On the other hand, for the sake of simplicity, and

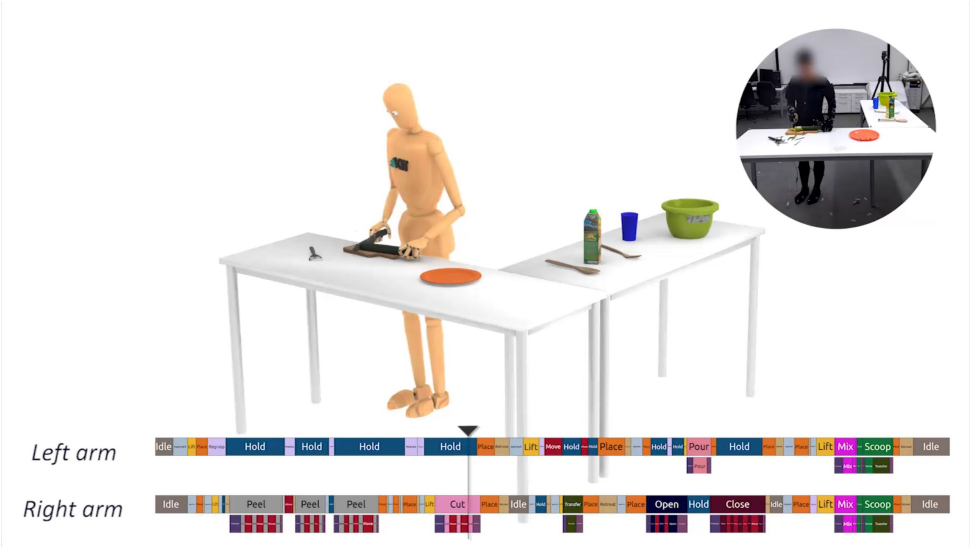


Figure 5.1: A frame of the Extended KIT Bimanual Manipulation Dataset. The frame is taken from a *prepare salad* plan. The main image shows the motion capture data of that frame (after some processing). The top right image shows the corresponding RGB frame. The bottom cells show the subaction and primitive segmentation and labeling for both hands. Image extracted from https://youtu.be/VRccEiYhc-4 [223] with permission of H^2T (KIT).

to avoid having to deal with an extra computer vision problem (as this is not the purpose of this study), we only use the data coming from the optical motion capture system. This leads to a variable-size input to the system (due to a different number of objects involved in the different recordings). We have approached this issue by designing an attention system that pays attention to a fixed number of objects, obtaining in this way a fixed-size input to the U-LRNN. Similarly to the U-LRNN, this attention system has been designed to work with sparse representations at its input. Therefore, a sparse representation for the input data has been defined, which is also described in the sections below.

New Recordings

The definition of new recordings has been mainly carried out attending to what challenges a goal recognition system should be able to tackle. Some of the challenges we have considered are the following (analogous challenges can be defined for the lower-level action recognition problem):

- Dealing with different **variations** of the same plan (e.g., different actions, different order, etc.).
- Recognizing the goal **as soon as possible** (since different plans may begin with same actions).

- Keeping **context** information (useful when the ongoing action is shared among multiple plans but the previous actions already determine the ongoing plan).
- Recognizing when a **new plan** has started (i.e., segmentation).
- Dealing properly with **uncertainty** when the goal has not yet been recognized (e.g., providing information about whether the goal has already been recognized, with what certainty it has, what candidate goals are being considered, etc.).

After careful analysis of the original dataset (and, in particular, of its long sequences), we noticed that, while it contains plenty of variations among recordings of the same class, the goals could be fully determined by knowing any of its action components and the object over which the action was performed, making the goal recognition task trivial once the ongoing action and corresponding object had been recognized, with no need for context information, segmentation, or uncertainty handling abilities. Guided by the challenges above, three new plans were defined that share actions and corresponding objects among them and with the previous plans. In particular, the previously existing plans were *prepare salad*, *prepare cake*, and *clean up*, and the new plans were *prepare grilled vegetables*, *prepare vegetables omelette*, and *prepare pizza*. While plan *clean up* is quite different from the rest, the other five plans are more similar to each other to different extents, sharing actions and corresponding objects at the beginning or later in the plan (with *prepare grilled vegetables* and *prepare vegetables omelette* being the plans sharing most actions and objects). Such overlaps among plans and diversity on the level of those overlaps make the resultant dataset more appropriate to evaluate a goal recognition system on the proposed challenges in different ways.

With the new plans properly defined, a set of recordings were carried out in collaboration with the High Performance Humanoid Technologies (H^2T) lab from the Karlsruhe Institute of Technology (KIT), owners of the dataset, in which five healthy subjects (three male and two female, four right-handed and one left-handed) were asked to perform those new plans (the old plans were also recorded for the left-handed male subject). These recordings were performed in the same setting and following the same procedure as those of the old recordings [223], introducing changes in the objects and their positions between recordings and asking subjects to introduce variations on how they performed the tasks. This study was approved by the ethics committees of the Free University of Bozen-Bolzano, Italy, and of the Karlsruhe Institute of Technology, Germany. All participants gave their written informed consent before the recordings. The data was properly anonymized after the recordings. By the time of the present study, not all new recordings had been fully post-processed and annotated as the old ones, making some of them not possible to use. Considering this, from now on we will refer to the data that has been used in this study, which corresponds to the state of the dataset by December 23rd, 2023. This

dataset includes the old recordings and the new recordings that had been properly processed and cleaned. Among these new recordings, few of them have been partially segmented and labeled, and the rest have no segmentation or labeling information. This leads to a partially labeled dataset consisting of 587 short recordings (classified into 12 actions) and 176 long recordings (classified into 6 goals) of kitchen activities performed by eight different subjects (four male and four female, seven right-handed and one left-handed). These recordings are (partially) segmented and labeled according to 21 subactions and 23 primitives per hand, with the subaction segments also including labels about the objects involved (which are 29 in total).

Data Augmentations

Another limitation of the dataset when used to test an action and goal recognition system is the fact that the different action and plan recordings include different sets of objects, having only objects that are relevant for the specific task. Therefore, by simply recognizing the objects present in the scene, the ongoing action or plan can be in most cases determined. To approach this issue, a random number of random static **objects** are **added** online to the top of the table of each sequence, with a maximum of 16 objects per scene. To determine their position and orientation in a realistic way, the positions and orientations of objects at the beginning and end of other sequences are borrowed, since, at those times, they are always on top of the table (except for the broom, which is always on the floor and leaning on the table). This augmentation, besides making the recognition task more challenging and the corresponding results more meaningful, serves the usual purpose of data augmentation of increasing the data variety to obtain more robust and less overfitted models. In addition, the augmented sequences count on objects that are never used, as well as on repeated object types, features that very rarely occur in the original data and that are quite common in the real world. Finally, this augmentation is very useful during the attention system hyperparameter search to avoid undesired solutions that only focus on objects that are very specific to certain actions (independently of whether they are being used or not).

The process to generate these random objects in realistic positions and orientations is as follows: First, the number and type of the objects to generate is decided using discrete uniform distributions. To determine the position and orientation of each object, the horizontal position coordinates and the orientation and vertical position coordinates are obtained separately. Regarding the horizontal position coordinates, we have assumed that they follow the same distribution for all types of objects (with the exception of the broom). However, they follow different distributions in the short and long recordings, since, in the short recordings, they lie on a single rectangular table, and, in the long recordings, they lie on the same and on an additional table forming an L shape with the first one (making the two coordinates dependent). At this point, we could have decided to manually define the limits of the tables and randomly generate

coordinates within those limits, but we opted for the solution of borrowing them from other sequences because it seems a more general solution requiring less explicitly introduced specific data. To ensure that both distributions are equally represented (since there are way more short sequences) and to avoid dealing with too many samples in the following step, 32 samples are randomly presampled from each set of coordinates. Then, the resultant 64 coordinates are weighted according to their distance to the closest object at the beginning or end of the current sequence. Finally, the resultant weights are used as probabilities to randomly select the horizontal coordinates of the new object. This allows the augmentation to avoid overlap between objects while not always selecting the coordinates with the maximum minimum distance. Regarding the orientation and height coordinates, these coordinates do depend on the type of object, with the height being determined by the type of object and its orientation (assuming it is on top of the table). Therefore, the new object coordinates are determined by randomly selecting from all the available coordinates corresponding to the same object type. In fact, sometimes, there are objects on top of others at the beginning or end of the recordings. To avoid using the coordinates of those objects (as it would imply having objects "floating" above the table), the coordinates with larger heights are discarded. Finally, white noise with similar properties to those found for the real static objects is added to the resultant position and orientation coordinates.

In addition to object generation, another augmentation that has been introduced is randomly **mirroring** the whole sequence. This augmentation, besides doubling the available data, approaches the issue of only having data from one left-handed subject, making the data handedness-agnostic and avoiding the issue of having very few samples of some subactions or primitives for the left or right hand. In addition, and similar to the previous augmentation, it is helpful during the attention system hyperparameter search. On the other hand, thanks to the sparse representation (described below), this mirror augmentation is obtained in a very straightforward and efficient way by just flipping arrays.

Finally, one limitation of this dataset is that both actions and plans are segmented according to their corresponding subactions, but plans are also composed of actions, and no segmentation or labeling exists in these terms. Still, there are subactions that are specific to a type of action (in fact, every action has some specific subaction), which allows us to know the action of the frames labeled with those subactions, i.e., to obtain a partial **labeling of** the **plans** at the action level. This partial labeling has the disadvantage that all the labeled frames belong to parts of the action at which the recognition task is probably simpler. In any case, that missing information is still present and can be learned from the short recordings (since the action of the whole sequence is known), while these new labels bring some new information in terms of transitions between actions. This label augmentation is particularly useful for the evaluation of our system on the action prediction task, where we make use of plan recordings with action labels.

Sparse Representation

Both the attention mechanism and the U-LRNN have been designed assuming that their inputs take the form of sparse representations. This makes the conversion of the dense coordinates of the subject body parts and involved objects into some sparse form necessary. We understand by sparse representations those representations for which most of their elements generally take values close to 0. Often, the elements of these representations can be interpreted as feature detectors, being the whole representation a set of indicators of what features are present or not (or to what extent) in the represented entity. Sparse representations can present a number of advantages over dense representations, such as being very appropriate to represent the real world, since real world entities can be generally described by their specific features out of a large number of possible features. In addition, their inherent redundancy allows them to be robust and fault-tolerant, as well to be highly separated in the representation space, which can simplify classification or clustering tasks and contributes to avoiding catastrophic forgetting during incremental or few shot learning (see Section 3.2.2). Furthermore, sparse representations can be very appropriate to deal with partial information and uncertainty, being sometimes analogous to probability distributions. This is particularly useful at the output of generative models, where such sparse representations can provide information about multiple potential candidates (and their likelihood) even when they come from, e.g., multimodal distributions or distributions with non-convex plausible sets. In contrast, in such situations, generated dense representations are often just non-realistic "blurry" weighted averages of the candidates [214].

Sparse representations can be especially beneficial when used to represent orientations. Indeed, finding good representations of orientations for machine learning applications is often problematic due to different reasons. Some characteristics that a good orientation representation should have according to [224] are being continuous, being unique (depending on the definition of continuity, this is actually a necessary condition for it, see [225]), not having degenerate configurations, and being invariant to object symmetry transformations (which is related to being unique). Even without considering symmetries, these constraints already discard some of the most popular orientation representations, such as Euler angles (they have discontinuities and degenerate configurations), quaternions (they double-cover the rotation space, i.e., they are not unique), or axis-angle representations (like quaternions but, additionally, with discontinuities and degenerate configurations). Other popular representations, such as using two or three (usually orthogonal) vectors (e.g., rotation matrices), do satisfy most of those characteristics but are not invariant to symmetry transformations. One possible way to approach this limitation in machine learning problems is augmenting the data by invariantly rotating or flipping symmetric objects. Still, the model would need to learn to be invariant to those transformations, which are different for different objects. If the representations themselves are already invariant, on the other hand, the system will also be by default. In this sense, sparse representations can

properly handle object symmetry invariance in a similar way to how they handle uncertainty. In this section, we describe a sparse data representation that has considered all these potential advantages in its design.

The data considered to build these representations includes information about the positions and orientations of the head, torso, and hands of the subject and types, positions, and orientations of the objects involved. The information about the body parts forms the fixed-size input data to our system, while the information about the objects forms the variable-size input data. All the positions and orientations are passed through a median filter of window size 5 to get rid of noise outliers (the data is sampled at 100 Hz) and converted to an egocentric reference frame with respect to the torso (its roll and pitch are not considered so that the vertical direction stays constant), being in this way independent from the arbitrary global reference frame. The information on the position and yaw of the torso is kept through their discrete derivative. Regarding the head, only its position, yaw, and pitch information is used, not its roll (since we are only interested in the gaze direction).

One natural way to obtain sparse representations of **positions** is discretizing the environment through the use of grids, with each element of the resultant representation corresponding to one cell in the grid. Such representations have the advantage that they can be used as probability distributions, keeping information about multiple candidates, but they have the limitations that they are coarse-grained and that they can only represent a limited previously-defined volume. The coarse-grained limitation can be approached by setting a Gaussian centered at the represented position and defining the value of each grid cell as the value that the Gaussian takes at the cell center. In this case, a probability distribution with multiple candidates would take the form of a discretized Gaussian mixture (see [226] for an example of such representation without the discretization). Regarding the issue with the limited volume, the coordinates in the range $(-\infty, \infty)$ can be mapped monotonically to the range $(-1, 1)$ through a squashing function such as tanh. Such mapping has the consequence that same differential position increments in the source domain lead to different increments in the target domain depending on how close the positions are to the origin, where those increments are maximum (and so the precision). We have defined such central "privileged" position 30 cm in front of and 40 cm below the torso, as, in the dataset, objects are commonly manipulated around that position. Finally, it is possible to tune how fast those increments change and how "privileged" the central position is by horizontally shrinking or stretching the mapping function. Considering all this, the expression used to obtain the sparse representations of the positions (which is very similar to that used for the synthetic actions in Appendix 4.A) is:

$$x_{pos}^{sparse} = e^{-\frac{\|g_{pos,i} - x_{pos}^{bounded}\|_2^2}{2 \cdot \sigma^2}}, \tag{5.1}$$

where $g_{pos,i}$ is each of the grid cell centers, σ is the standard deviation of the Gaussian, $\| \cdot \|_2$ is

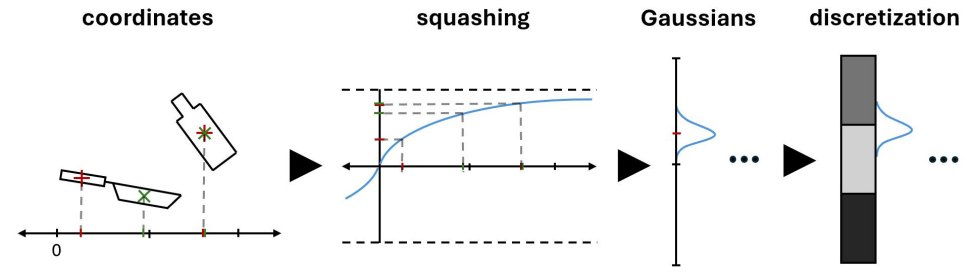
the L2 norm operator, and:

$$x_{pos}^{bounded} = \tanh\left(\eta_{\leftrightarrows} \cdot (x_{pos}^{orig} - x_{pos,ref})\right) , \tag{5.2}$$

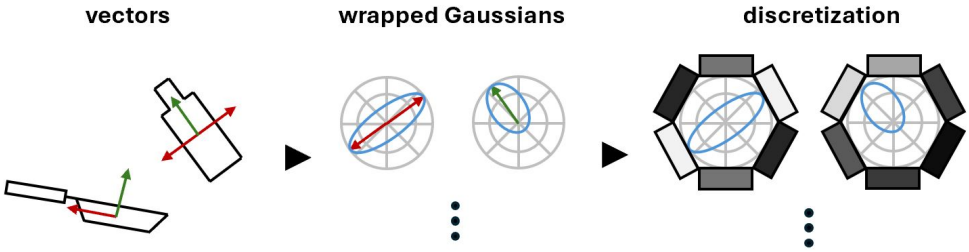
with $\eta_{\leftrightarrows}$ being the tanh shrinking factor, $x_{pos,ref}$ the mentioned reference position coordinates in front of the torso of $(30, 0, -40)$ cm, and x_{pos}^{orig} the original position coordinates (in the egocentric reference frame). The representation of the different coordinates can be obtained independently, using three unidimensional Gaussian functions (and grids), or combined, using multidimensional Gaussian functions. In our case, since the horizontal coordinates are strongly coupled (in part due to the conversion to the egocentric reference frame), and to avoid having too many cells (and a too sparse representation), a 5×5 2D grid is used to code the horizontal coordinates, and a separate 5-cell 1D grid is used for the vertical coordinate, leading to a position representation of size 30. The remaining constants have been set empirically to $\sigma = .25$ and $\eta_{\leftrightarrows} = 2^{-9}$ (the positions in the dataset are in mm). The resultant data contains, per frame, the positions of the head and hands and two positions per object: if the object has a handle, the center of the handle and the center of the object without the handle, and if it doesn't, the center of the object twice. One motivation behind using those two positions is the fact that the center of the handle can be more helpful to understand the interaction of the object with the hands, while the other center may be more useful in terms of the interaction of the object with other objects. This decision was also taken having in mind a more general system that needs to extract such information from unknown objects and using a different data modality such as video (having the added benefit that the potentially involved uncertainty can be properly handled in the representation). Furthermore, this representation has the additional advantage that it provides indirect information about whether the object has a handle and, if that is the case, about the size and orientation of the object. The process to obtain sparse position representations is illustrated in Fig. 5.2a.

The sparse representations of the **discrete derivatives** of the torso position and yaw are obtained following the same process just described for positions. In the case of the torso position differences, a 3×3 2D grid is used for the horizontal coordinates and a 3-cell 1D grid for the vertical coordinate, leading to a representation size of 12, and the remaining constants have been set to $\sigma = .5$ and $\eta_{\leftrightarrows} = .25$ (the reference coordinates take of course a value of 0, as we are dealing with differences). Regarding the torso yaw, a 3-cell 1D grid is used, with the remaining constants being $\sigma = .5$ and $\eta_{\leftrightarrows} = 64$.

To obtain sparse representations of **orientations**, a process similar to that of positions is followed (see Fig. 5.2b), representing a number of vectors that determine the object orientation using (an approximation of) wrapped Gaussians around the sphere discretized on a spherical tiling grid (in this case, the non-linear mapping is not necessary). In particular, we use two orthogonal (sets of) unit vectors attached to the object to represent its orientation, defined again so that the



(a) Positions.



(b) Orientations.

Figure 5.2: Illustration of the process to obtain sparse position and orientation representations (simplified in 2D). To obtain position representations, the object and handle centers are determined (or twice the object center for objects without handle), those coordinates are squashed, and, for each coordinate, a Gaussian is defined and discretized into a grid. To obtain orientation representations, the object handle and top vectors are determined and replicated attending to the object symmetries, and, for each vector (or set of replicated vectors), an approximation of a wrapped Gaussian is defined and discretized into a grid.

representation is somehow consistent across objects and extendable to new objects. For objects without symmetries, we have defined one vector in the direction of the handle (or preferred point for grabbing it) and the other one in the direction of the top of the object, which is generally the part that interacts with other objects, and the open part for open objects (e.g., bottles, bowls, etc.). In the case of objects with symmetries, we define one first handle and top vectors according to the previous criteria and arbitrarily among the multiple symmetric candidates, and we replicate them using the invariant transformations. For example, a square bottle, which is invariant to rotations of 90° around its vertical axis, is characterized by a single top vector in the direction of its mouth and by four horizontal mutually orthogonal or opposite handle vectors in the direction of the sides of the bottle. Table 5.1 shows the symmetries considered for each object. Regarding the hands, their orientation is also described by two vectors, with one vector pointing in the direction of the wrist (analogous to the handle) and the other one in the direction of the palm of the hand (the "open" part). Finally, for the head orientation, we only use one vector in the direction of the face, as we don't consider its roll. These two (sets of) vectors are used to define the peaks of two (possibly multimodal) wrapped Gaussian-like functions around the sphere. In particular, for each vector, the function used is a von Mises–Fisher distribution rescaled so that

it takes a value of 1 at its peak. When there are multiple vectors per sphere due to symmetries, the corresponding functions are added. In the extreme case of circular symmetry (which would correspond to infinite vectors in a plane), an equivalent function is used that takes the form of a surface of revolution around the symmetry axis (orthogonal to that plane) whose generatrix has a peak with a value of 1 at its intersection with that plane (the actual function is obtained using a unit vector in the direction of the symmetry axis and the Pythagorean theorem). In the case of spheric objects, a constant function is used for both the handle and the top. This spheric orientation representation is used for the lids because their orientation information could not be captured properly during the data acquisition process (taking in this way advantage of the ability of this representation to show uncertainty). With the functions defined, the final step consists of building a grid around the spheres and discretizing the functions. A dodecahedron has been chosen as the spherical tiling polyhedron because, first, it is regular, which makes all the grid cells equal (and, if there were no preferred orientations, equally likely), second, its faces are regular pentagons, i.e., as close to circles as possible within regular polyhedrons (which makes the points of each grid cell better represented by its center for a given area), and, third, it has 12 faces, which seems an appropriate number of cells to represent in a sparse way the direction of a vector. Hence, the final orientation representation is generated by obtaining the value of the handle and top functions at the centers of the dodecahedron faces, leading to an orientation representation of size 24 (12 for the head). The representation values corresponding to a single vector of a non-symmetric object are given by:

$$x_{rot}^{sparse} = \frac{1}{e^{\kappa}} \cdot e^{\kappa \cdot x_{rot}^{orig} \cdot g_{rot,i}} , \tag{5.3}$$

where $g_{rot,i}$ is each of the unit vectors pointing to the dodecahedron face centers (or to the dual icosahedron vertices), κ is the concentration parameter of the corresponding von Mises–Fisher distribution (analogous to the inverse of the variance), and x_{rot}^{orig} is the original unit vector (in the egocentric reference frame). In the case of circular symmetry, the representation values are given by:

$$x_{rot}^{sparse} = \frac{1}{e^{\kappa}} \cdot e^{\kappa \cdot \sqrt{1-(x^{\perp}{}_{rot}^{orig} \cdot g_{rot,i})^2}} , \tag{5.4}$$

with $x^{\perp}{}_{rot}^{orig}$ being the unit vector in the direction of the symmetry axis. Parameter κ has been set to different values depending on the number of peaks, with the peaks being narrower as their number increases: It has been set to 4 for functions with one peak, to 8 for functions with two peaks, and to 16 for functions with four peaks or with circular symmetry (there are no other cases among the objects in the dataset). Regarding spheric objects, all their representation elements take a value of 0.1. The resultant sparse representations for orientations described here, besides being defined so that they are applicable to new objects, satisfy the previously mentioned desired characteristics to represent orientations, including that of being invariant to

Table 5.1: Symmetries and identifier features of each object. The symmetries are expressed according to the vectors affected (not the axis of rotation). Reflection symmetries with respect to the plane formed by the two vectors are not considered because they don't affect them. The identifiers are built attending to the following properties: object size without handle (big, medium, or small), geometric shape without handle (spherical, cylindrical, or prismatic), longness without handle (along the handle axis, along the top axis, along the third axis, medium, or flat), empty (yes or no), open (yes or no), continuous (yes or no), handle (long, short, or no), sharp (yes or no), and material (hard, medium, or soft).

	Symmetries		**Identifier features**								
Object	handle	top	size	geometry	longness	empty	open	continuous	handle	sharp	material
juice	90°	-	big	prismatic	top	yes	yes	yes	no	no	medium
juice lid	spheric		small	cylindrical	medium	yes	yes	yes	no	no	medium
broom	-	-	big	prismatic	third	no	no	no	long	no	hard
cucumber	180°	circular	big	cylindrical	handle	no	no	yes	no	no	medium
large cup	circular	-	big	cylindrical	top	yes	yes	yes	no	no	medium
small cup	circular	-	medium	cylindrical	medium	yes	yes	yes	no	no	medium
cutting board	-	180°	big	prismatic	flat	no	no	yes	short	no	hard
draining rack	-	-	big	prismatic	medium	yes	yes	no	no	no	medium
egg whisk	-	-	medium	spherical	handle	yes	no	no	short	no	medium
eggplant	-	circular	big	spherical	handle	no	no	yes	no	no	medium
frying pan	-	-	big	cylindrical	flat	yes	yes	yes	short	no	hard
knife	-	-	medium	prismatic	handle	no	no	yes	short	yes	hard
ladle	-	-	medium	spherical	medium	yes	yes	yes	short	no	medium
milk	90°	-	medium	prismatic	medium	yes	yes	yes	no	no	medium
milk lid	spheric		small	cylindrical	medium	yes	yes	yes	no	no	medium
big bowl	circular	-	big	spherical	medium	yes	yes	yes	no	no	medium
green bowl	circular	-	big	spherical	medium	yes	yes	yes	no	no	medium
small bowl	circular	-	medium	cylindrical	top	yes	yes	yes	no	no	medium
oil bottle	90°	-	big	prismatic	top	yes	yes	yes	no	no	hard
peeler	-	-	small	cylindrical	flat	no	no	no	short	yes	hard
plate	circular	-	big	cylindrical	flat	yes	yes	yes	no	no	medium
rolling pin	180°	circular	big	cylindrical	handle	no	no	yes	short	no	hard
salad fork	-	-	medium	prismatic	flat	yes	yes	no	short	yes	hard
salad spoon	-	-	medium	spherical	flat	yes	yes	yes	short	no	hard
spatula	-	-	medium	prismatic	flat	no	no	yes	short	no	medium
sponge	180°	180°	medium	prismatic	medium	no	no	no	no	no	soft
spoon	-	-	small	spherical	flat	yes	yes	yes	short	no	hard

symmetry transformations. In addition, these orientation representations provide information about the geometry of the objects, as they inform about their symmetries.

Finally, the object types are coded in terms of **identifiers** defined according to the object geometry and appearance (without considering specific functionality other than that of handles). Again, the possibility that this description could be successfully used on new objects and that an automatic (e.g., vision) system could determine those identifiers on unknown objects motivated this description. To build these identifiers, nine object characteristics are considered, each one taking up to five possible values, leading to a total (and to an identifier size) of 25 features. The characteristics and features defined, as well as the corresponding object descriptions, are shown in Table 5.1. The identifier elements indicate the presence or absence of each feature,

taking value 0.9 for features present in the object and value 0.1 otherwise. One advantage of this representation is that it also admits more fuzzy object representations according to "to what extent" each feature fits the object.

With the complete sparse representation described, we can now calculate its size. The fixed-size part has a total size of:

$$s_{fix} = \underbrace{12+3}_{\text{torso}} + \underbrace{30+12}_{\text{head}} + \underbrace{2\cdot(30+2\cdot 12)}_{\text{hands}} = 165\,, \tag{5.5}$$

and the variable-size part has, per object, a size of:

$$s_{var} = \underbrace{25}_{\text{identifier}} + \underbrace{2\cdot 30}_{\text{position}} + \underbrace{2\cdot 12}_{\text{orientation}} = 109\,. \tag{5.6}$$

5.2.2 The Self-supervised Attention Learning System

Since the U-LRNN requires a fixed-size input and the input data we have described has a variable size depending on the number of objects involved in the ongoing sequence, some input adaptation module is required. We have opted for a novel attention system that pays attention to a fixed number of objects at each timestep. This section describes this attention system, which learns to extract attention weights in a self-supervised way. The system has been designed without considering specific characteristics of human action and plan data, being therefore also applicable to temporal data from other domains with a similar structure, i.e., with an (optional) fixed-size part and a variable-size part consisting of equal-sized (and preferably sparse) representations of a variable number of entities to which to pay attention.

Attention mechanisms generally consist of systems that take a (possibly variable) number of candidate vectors as input, together with a query, and generate as output a weighted average of the so-called values, which are learned representations of the input vectors, with the weights representing the estimated relative importance of each input vector according to the given query (see [227] for a comprehensive survey). The weights are typically obtained by first calculating a score for each input vector from the query and the input vector key (which is also a learned representation of the vector). Then, an alignment function (e.g., the softmax function) is applied over all the scores to convert them into weights that sum up to 1. In the case of our system, the fact that the input representations are sparse brings some advantages that allow the system to be simpler. First, as mentioned in Section 5.2.1, weighted averages of a number of these representations are often able to keep most meaningful information (at least, of the representations with larger weights). Therefore, these sparse representations can be directly used as values in the attention system without causing blurry hard-to-interpret outputs when the attention is not fully focused on a single object, as would happen for dense representations. This makes the system and its training simpler, as it doesn't need to learn alternative representations

for the values. In addition, by using directly the input representations as values, we can rely on a self-supervised learning mechanism designed exclusively to learn the best attention weights, and we don't need to worry about learning value representations that keep all the relevant data for the next stages or downstream tasks. The fact that the input representations are sparse also allows the system to use them directly as keys (and to avoid again the need to learn these representations), obtaining the scores in a quite intuitive way: If we interpret the elements of the input sparse representations as indicators of the presence of specific features, we can define the query as a list of elements of the same length, with each element indicating how important the presence of each feature is. For example, an element of the query taking a positive value indicates that input vectors with the corresponding specific feature are preferred, with higher values indicating that the presence of that feature is more important according to the query. Similarly, a negative value indicates that input vectors without the feature are preferred, with more negative values giving more weight to this. Following this idea, the score of an input vector can be obtained directly through the dot product of the query and the vector. This way of obtaining the score is quite common among attention mechanisms, with the result being often divided by the square root of the key/query vector length when the posterior alignment function is a softmax. This prevents the softmax function from having too large inputs (which lead to very small gradients) when the lengths are large. The resultant attention mechanism is known as scaled multiplicative (or dot-product) attention, which is the one used in the popular transformers [102]. In our case, since the keys and values are the input representations themselves, the resultant attention is given by:

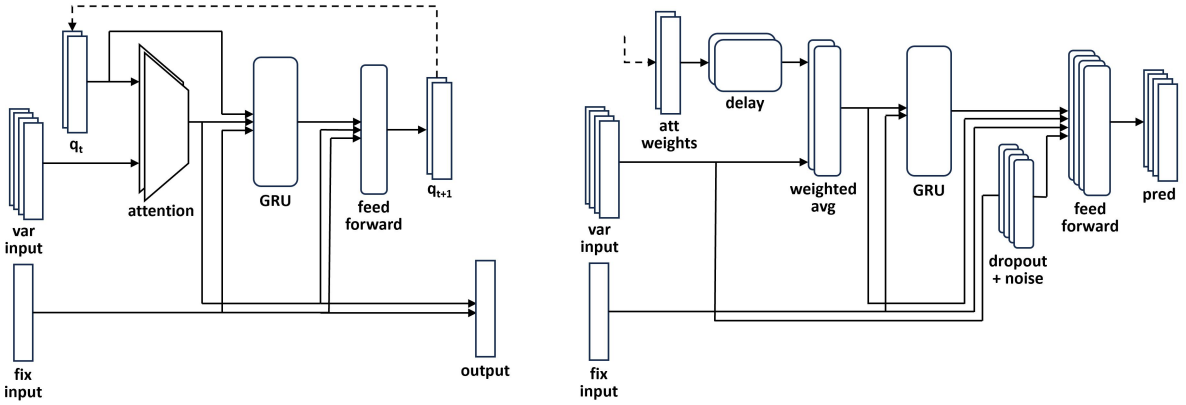
$$a = \text{softmax}\left(\frac{qX^T}{\sqrt{d}}\right)X\,, \tag{5.7}$$

where q is a row query vector, X is a matrix with the input vectors as rows, and d is the input vector (and query) size. Similarly to transformers, our attention system has multiple heads, i.e., it is composed of multiple attention modules that allow the system to pay attention to multiple objects simultaneously through multiple queries. The output of the system is the concatenation of the outputs of the different heads, which leads to an input size for the U-LRNN of:

$$s_{att} = s_{fix} + h \cdot s_{var}\,, \tag{5.8}$$

where h is the number of heads of the attention system.

The complete attention system is shown in Fig. 5.3. Assuming that the system is already trained (Fig. 5.3a), the **attention system** consists of the attention mechanism just described and of a circuit that obtains, at each timesample, the query for the next timesample (note that there is no attention in the time dimension, the attention mechanism only operates within the sample). This circuit is formed of a (single-layer) gated recurrent unit (GRU) that keeps information about the environment followed by a feedforward network that outputs the query. The GRU, of



(a) The attention system.

(b) The additional circuit to train the attention system in a self-supervised way.

Figure 5.3: The self-supervised attention learning system. The attention system mainly consists of a multi-head scaled dot-product attention mechanism that pays attention to specific objects at each timesample and of a GRU that generates the query for the next timesample. The training of the attention system includes an additional circuit to predict the representations of all objects given their weighted sum using recent past attention weights. This encourages the system to generate weights useful for the near future.

hidden size $s_{q,gru}$, takes as input the fixed-size part of the input data, the output of the attention mechanism, and the query itself. Taking the query as input helps the GRU understand what is being paid attention to, especially when the query does not match any input, and preliminary tests have shown that it contributes to make the attention system more stable and focused. In fact, it can also be seen as a form of multi-hop attention mechanism that refines the query at each timesample. Regarding the feedforward network, it takes as input the output of the GRU as well as the fixed-size input and the attention module output through skip connections, and it consists of a feedforward layer of size $s_{q,h}$ with a ReLU activation function followed by a linear layer without additive bias (i.e., a product with a weight matrix). The output layer has no non-linearity so that the query can take unbounded positive and negative values, as described earlier.

The **self-supervised learning mechanism** to train this system has been designed so that the weights learned, when used by a system in a close future, allow this system to keep best track of the environment state. This can be seen as a weight predicting system, where the system anticipates what object coordinates will be harder to predict and therefore focuses on them to enable a more accurate environment state information. Note that the system is not predicting the future state of the attended objects from the current state data, as this would be a more complex problem, and we are interested in keeping the problem as simple and centered solely on learning the weights as possible. Note also that the system in the near future is using directly the weights from the past, and not the queries. Otherwise, the queries would again need to adapt to the future

instead of current state of the input vectors, introducing complexity to the problem. Considering that the coordinates of static objects in our data can be deduced from their previous state and the torso motion, this self-supervised technique encourages the system to focus on objects that will probably be manipulated soon or that are currently being manipulated, which is the behavior that we are looking for. This mechanism may also have a good performance when used with data from other domains, as it does not rely on specific characteristics of human action data but rather on what it seems a quite generic principle. The circuit designed to achieve this behavior is shown in Fig. 5.3b. This circuit takes the weights from the attention mechanism, introduces a delay τ on them, and uses the delayed weights to obtain the weighted average of the inputs, obtaining in this way an alternative attention output that uses weights from the past. This attention output, together with the fixed-size input, is taken as input by a single-layer GRU of hidden size $s_{\hat{x},gru}$ that, again, is expected to keep track of the state of the environment (in this case, it does not take the query as input, as it is not used to generate new queries, plus the query would correspond to input vectors from the past). The output of this GRU, together with its input through skip connections and a corrupted version (via dropout and Gaussian noise) of each input vector, is fed to a feedforward network (consisting of two feedforward layers with ReLU activation functions) of hidden size $s_{\hat{x},h}$ that tries to reconstruct the corresponding original input vector. This mask prediction task was chosen over other more simple tasks (such as predicting the object position given its identifier or vice versa) because it is more general, enabling the presence of multiple objects with the same identifier and encouraging the GRU to keep more complete information of the environment (affecting the learned weights accordingly). In addition, it is directly applicable to data from other domains. On the other hand, the training assumes that the ground truth about all objects in the environment is known. Still, for applications in which this is not the case (e.g., coordinates extracted from the video of a moving camera), the attention system can basically work in the same way, relying only on the available data. The loss function used to train the whole attention system is:

$$J(W, b) = J_{error} + \lambda \cdot J_{regularization} + \psi \cdot J_{focus} \,, \tag{5.9}$$

where W and b represent the weights and bias units of the whole system, J_{error} is the weighted Minimum Squared Error (MSE) loss term as described in (3.2), $J_{regularization}$ is the L_2 regularization term as described in (3.4), J_{focus} is a loss term that pushes the attention weights to be more focused on a single input vector, and λ and ψ are loss term weights:

$$J_{focus} = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{h} \left(\|w_{i,j}\|_1 - \max(w_{i,j}) \right) \,, \tag{5.10}$$

where $w_{i,j}$ is the attention weight vector of sample i and head j. This loss term pushes to 0 all the weights of the weight vector but the maximum one (which is therefore pushed to 1) and can be seen as a sparsity term designed specifically for vectors whose elements add up to one and

where only one active element is desired (note that other more standard sparsity terms would not work as desired in this case).

5.2.3 The U-shaped Locally Recurrent Neural Network

The U-shaped locally recurrent neural network (U-LRNN) is an extension of the HLRNN that is able to perform a number of additional tasks such as prediction or action selection while still being trained in a fully self-supervised way. For the sake of versatility, this system was designed with two objectives in mind: First, it should be able to provide arbitrarily far predictions (together with the corresponding trajectories or sequences of samples leading to those predictions), and, second, it should be able to keep information on uncertainty, providing information about the distribution of probabilities of those predictions and trajectories. In addition, it should be able to perform these tasks at the different levels of the HLRNN, with the higher levels being better and faster at performing longer-term predictions and the lower levels being also able to predict low-level information. Performing such tasks on action and plan data at a high level, i.e., having a system that provides a set of candidate sequences of plausible coming actions and their probabilities, can be seen as quite close to a plan recognition task where actions are expressed as HLRNN internal representations. Such "subsymbolic plans" can already be useful for different tasks, but, if a more explicit representation is desired, the same classifiers used on top of the HLRNN blocks for action recognition can be used to obtain the corresponding sequences of action labels. Regarding the task of action selection, an imitation learning approach can be taken by using a similar prediction system, in this case with information about the desired goal and, therefore, performing predictions conditioned to that goal. Then, classifiers can be used to extract the selected action label, or, at a lower level, the desired movement of the hands or other body parts can be extracted without the need for labeled data.

One quite straightforward way to obtain arbitrarily far predictions and the corresponding trajectories is through multi-horizon iterative methods, i.e., by building an autoregressive one-step-ahead prediction system and using the prediction at each step as input to the next step [228]. However, this approach has the disadvantage that, if the system has been trained to minimize the square error between the prediction and the real sample, it will generate blurry unrealistic predictions. This issue will accumulate and get worse as the system is iteratively refed with these blurry predictions. In addition, the prediction system will be being fed with inputs that follow a different distribution from that with which it has been trained, possibly leading to unpredictable behavior (note that this is also true for sparse representations). Some ways in which this issue can be approached are by storing and appropriately reusing representative samples (e.g., through clustering) or through the use of generative models such as variational autoencoders (VAEs), generative adversarial networks (GANs), diffusion models, or flow-based models [229]. The approach of reusing already seen representations, while could show advantages in terms

of plan segmentation and one-shot learning as well as in terms of dealing with uncertainty (as in [214]), would in principle have the limitation, when compared with generative models, of not being able to generalize to, e.g., combinations of actions and environment states that have never been seen together (this issue could be alleviated by using product quantization as in [143], but then the advantage of having probability information would be lost unless the chosen subvectors are mutually statistically independent). Regarding generative models, GANs and diffusion models are very powerful in terms of generating realistic samples, but they are also computationally expensive to train. In addition, the distributions followed by the generated samples are generally not good approximations of the original distributions, with the models sometimes even suffering from mode collapse, only generating samples from specific regions of the set of plausible samples. Flow-based models and variational autoencoders, on the other hand, define a latent space where the data follows a well-known probability distribution and learn a mapping between this latent space and the space of original data, enabling in this way the extraction of realistic samples that approximately follow the real distribution by sampling in the latent space and transforming the sample according to the mapping. In fact, we believe that our sparse representations admit an even simpler approach by defining the well-known distribution directly in the original representation space. In particular, we have assumed that the distribution of HLRNN representations before the ReLU non-linearity can be approximated by a mixture of Gaussians, with each Gaussian representing a cluster of candidates to be the next sample. The actual representation can thus be approximated by a mixture of rectified Gaussians, which (in its univariate form) is equal to a mixture of Gaussians for values greater than 0 and takes at 0 a probability equal to that of the original mixture of being lower than 0. Hence, we will use mixture density networks (MDN) [230] to perform these predictions, which are neural networks that estimate the parameters of a mixture distribution given an input. Finally, to obtain the desired arbitrarily far predictions and an estimation of their probabilities, Monte Carlo methods can be used, bootstrapping from the original distribution and feeding the resultant samples iteratively to the network until the desired horizon is reached. The final set of samples can be used as an approximation of the sought distribution (if an explicit probability density function was required, it could probably be approximated using again a mixture of rectified Gaussians).

Fig. 5.4 shows a U-LRNN **architecture** with three levels (it is shown as an inverted "U" to be consistent with what is above and below in the hierarchy). It consists of an encoder-decoder architecture with the HLRNN taking the role of the encoder that goes up the hierarchy, a block able to keep longer-term context information on top such as a GRU or a transformer, and a decoder that basically consists of pairs of upsampling layers (of the same timescale factor as the corresponding temporal max pooling layers) and MDNs that predict the next state at their corresponding HLRNN levels and timescales. For convenience, in the HLRNN, the temporal max pooling layers have been represented out of the locally recurrent blocks (LRB). At each

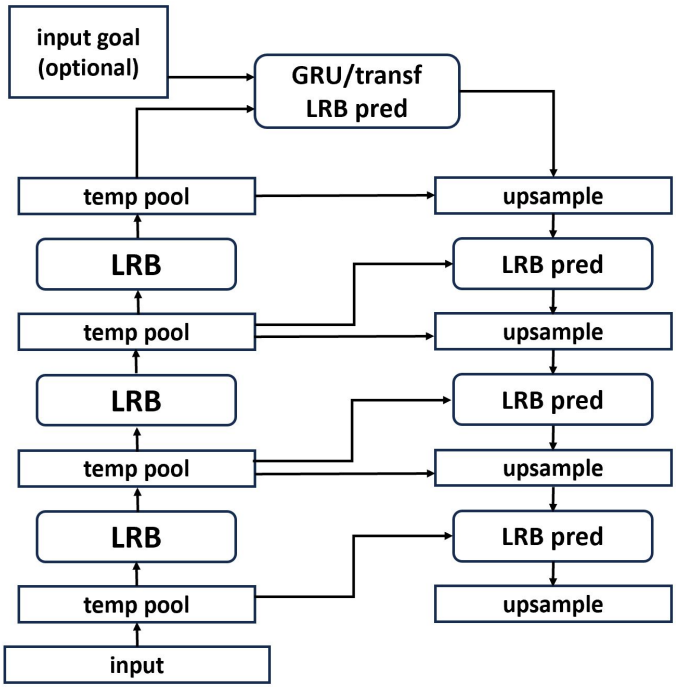


Figure 5.4: A three-level U-LRNN architecture. This system takes the form of an encoder-decoder, with the encoder being an HLRNN and the decoder mainly consisting of a stack of MDNs that predict the next representations at each HLRNN level. Between the encoder and the decoder, a context-aware unit predicts the next representation at the top of the HLRNN.

level, the MDN predicts the next representation on top of the corresponding temporal max pooling layer (e.g., the next action) based on the current representation (e.g., the current action), the upsampled current representation of the level above (e.g., the ongoing higher-level action that contains the current action and possibly the next action), and the upsampled predicted representation of the level above (e.g., the next higher-level action or goal towards which the agent is shifting). These MDNs are shallow feedforward networks that estimate the parameters of a mixture of rectified Gaussians distribution, where each rectified Gaussian has a diagonal covariance matrix (the dependencies among elements of the representation are modeled through the mixture itself). Thus, for a representation of size s and a mixture of n rectified Gaussians, the MDN estimates $n \cdot s$ means, $n \cdot s$ variances, and n mixture coefficients (for simplicity, the means and variances predicted refer to those of the original Gaussian from which the rectified Gaussian is derived). The MDN has ReLU activation functions at the hidden layer and different activation functions at the output: no activation function for the means, exponential linear unit (ELU) activation functions for the variances (as recommended in [230]), and a softmax layer for the mixing coefficients. The MDN at the top level has no level above and, therefore, uses the information from the context block on top instead. These two blocks form what we have called the context MDN, with the feedforward MDN sub-block getting as input the GRU output as well as the GRU input through skip connections.

One strength of this architecture is that, when only high-level predictions are required, only the encoder and decoder blocks of that level and above (which work at slower timescales) need to be used for the iterative prediction, reducing considerably the computation. On the other hand, it should be noted that, in this architecture, the representations that are mainly used at each level are those after the temporal max pooling layer, and not before, as was done with the HLRNN (in fact, the HLRNN does not have a temporal max pooling layer on top, this layer has been added for the U-LRNN). Indeed, during LRB hyperparameter search, it makes sense to work with the representations before downsampling, using the search to find the temporal max pooling layer hyperparameters that best contribute to the performance of the LRB on top of it (this is different for the 2D max pooling layer, which helps to find configurations with an emerging self-organization mechanism). However, once the HLRNN has been trained, it seems more convenient to work with the representations after the 2D and temporal max pooling layers (which work as a single 3D max pooling layer), having in this way a slower-changing and downsampled data, which, in addition, involves further-in-time one-step-ahead predictions. In fact, using these representations to go down the hierarchy makes the decoder connectivity somehow analogous to that of the feedback circuits of the neocortex, with their connections originating and terminating in the superficial and deep layers (such as L2/3, analogous to our max pooling layers), avoiding L4 (analogous to our locally recurrent layer) [161].

There are a few differences between the architectures used for prediction and for action selection, mainly related to the fact that, at each level, the action selection needs to be guided towards some specific goal, while this is not the case for prediction. Beginning from the top level, when doing prediction, the context MDN only takes as input the output of the top temporal max pooling layer, while, when doing action selection, it also takes as input the desired goal (e.g., in the form of a one-hot encoding vector representing the goal label). At the other levels, when doing prediction, the MDNs take as input the whole predicted distribution (i.e., the predicted parameters), which allows the MDNs to consider the uncertainty at the higher levels for their predictions. When doing action selection, on the other hand, an actual sample is selected from the distribution at each level, and the MDN below takes this sample as input (instead of the whole distribution), taking in this way its decision according to the higher-level decision. During training, this higher-level decision is the actual next state of the level above (independently of the prediction). During action selection, we have defined it as the mean parameter of the Gaussian with the highest mixing coefficient (setting to 0 its negative components), taking in this way a winner-take-all-like approach. This mechanism that is repeated at each level of the decoder can be seen as a form of conditional generative modeling, with the chosen representation generated at each level being conditioned on the selected action from the level above (or on the goal in the case of the highest level) [231]. It is also related to the idea of generative world modeling (as in [232]), with the system predicting the next input representation (or set of possible representations) based

on the previous input and selected action.

Similarly to the HLRNN, the U-LRNN can be **trained** one block at a time in a greedy manner, starting from the context MDN and then going down the decoder hierarchy, training the MDNs one by one. Since we are estimating the parameters of a probability distribution, the loss function used to train the MDNs is based on the negative log likelihood (NLL) function:

$$J(W, b) = J_{NLL} + \lambda \cdot J_{regularization} \,, \tag{5.11}$$

where J_{NLL} is the NLL term corresponding to a mixture of rectified Gaussians, $J_{regularization}$ is the L_2 regularization term as described in (3.4), and λ is a loss term weight:

$$J_{NLL} = -\frac{1}{m} \sum_{i=1}^{m} log \left(\sum_{j=1}^{n} \pi_{i,j} \cdot \prod_{k=1}^{s} f_{N^R}(y_{i,k}; \mu_{i,j,k}, \sigma^2_{i,j,k}) \right) \,, \tag{5.12}$$

with $\pi_{i,j}$ being the predicted mixing coefficient of the j-th rectified Gaussian of the i-th output sample, $y_{i,k}$ the k-th component of the i-th target output sample, $\mu_{i,j,k}$ and $\sigma^2_{i,j,k}$ the predicted mean and variance parameters, respectively, of the k-th component of the j-th rectified Gaussian of the i-th output sample, and $f_{N^R}(x; \mu, \sigma^2)$ the probability density function of an univariate rectified Gaussian distribution derived from a Gaussian distribution of mean μ and variance σ^2:

$$f_{N^R}(x; \mu, \sigma^2) = \int_{-\infty}^{0} f_N(x; \mu, \sigma^2) \mathrm{d}x \cdot \delta(x) + f_N(x; \mu, \sigma^2) \cdot u(x) \,, \tag{5.13}$$

where $f_N(x; \mu, \sigma^2)$ is the probability density function of the corresponding univariate Gaussian distribution, $\delta(x)$ is the Dirac delta function, and $u(x)$ is the Heaviside step function:

$$f_N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \,, \tag{5.14}$$

$$\delta(x) = \lim_{T \to 0} \begin{cases} \frac{1}{T} & |x| \leq \frac{T}{2} \\ 0 & |x| > \frac{T}{2} \end{cases} \,, \tag{5.15}$$

$$u(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \,. \tag{5.16}$$

To avoid numerical underflow, critical calculations of the NLL loss term are computed directly in the log domain appropriately using numericably stable implementations of the log-sum-exp and log-erf functions.

Due to how the U-LRNN is used, there are some minor differences between the HLRNN implementation in Chapter 4 and the one used here. First, the U-LRNN needs to be fully causal, relying only on current and past samples, both for action selection (as it doesn't know the next input yet) and for action prediction (as otherwise it would be a reconstruction rather than a

prediction problem). The temporal max pooling layers of the original HLRNN, however, use kernels centered at the current sample, meaning that their output depends on the nearby past and future samples. In addition, its upsampling layers rely on a linear interpolation algorithm, with each interpolated sample being generated from its previous and next known samples. This has been modified for this chapter, shifting the kernels of the temporal max pooling layers so that the current sample is the last sample they reach and making the upsampling layers work as zero-order holds, with the interpolation solely consisting of replicating the last sample. Another difference is that the U-LRNN needs to keep its hidden state across batches, and not only within batches, as was done in the original HLRNN (during training, the gradient is not backpropagated to the previous batch, though). Indeed, both during multi-horizon prediction and during action selection, the U-LRNN is fed with single-sample batches, and not keeping its state across batches would be equivalent to not having state at all (and, even when working with batches, the shorter batches of the lower levels of the decoder would not behave as desired when going through the downsampled higher levels). This state involves the hidden states of the recurrent layers but also the earlier samples at the input of the temporal window and temporal max pooling layers covered by their respective windows/kernels. It also involves the state of the downsampling and upsampling layers: For single-sample batches, the downsampling layers have sometimes no output (with the higher blocks in the hierarchy working at slower timescales only seeing some of the batches), and the upsampling layers need to generate batches also when they have no input. Working with single-sample batches also makes original transformers not a good choice for the top level, as they are only able to pay attention to samples in the same batch (or segment), and architectures that approach this issue such as Transformer-XL are more appropriate [233]. The integration of these two changes (making the network causal and keeping the hidden state) seems to involve a small decrease in performance at each level of the HLRNN that accumulates and becomes relevant when going up the hierarchy. Why exactly this happens and how to address it remains open for further research.

5.3 Results

This section presents the available results regarding the systems introduced in Section 5.2. Unfortunately, only the results about the attention system are available so far. Still, this section also describes how the U-LRNN is being tested.

The process followed to train the system and set the hyperparameters is similar to that used for the HLRNN: A hyperparameter search is performed iteratively block after block, using Bayesian optimization at each iteration. The first search of the hyperparameter tuning process defines the attention system, then the LRBs are defined one by one starting from the bottom of the hierarchy, then the context MDN is defined, and finally the feedforward MDNs are defined one by one starting from the top of the hierarchy. Each search configuration is trained in a self-supervised

way and then evaluated according to a different fitness function depending on the type of block. The fitness function for the attention system and for the LRBs is the sum of the macro-averaged F_1 scores obtained for one or more classifiers trained on top of the self-supervised blocks (these classifiers count on an upsampling layer at their output). The fitness function for the MDNs, on the other hand, is fully unsupervised and consists of the same NLL defined in (5.12) (i.e., the training loss function without the regularization term). As mentioned in Section 5.2.3, the top LRB hyperparameter search does not reach the top temporal max pooling layer, but it would also not be appropriate to define the hyperparameters of that layer during the context MDN search, as this block is trained to predict the representations on top of that layer. Indeed, doing so would lead to hyperparameters that make the prediction task easier, and not necessarily to hyperparameters that are appropriate for our task. Therefore, these hyperparameters are set manually. On the other hand, for the sake of simplicity, a single-layer GRU has been used as the context unit on top of the hierarchy.

The characteristics on which the complete architecture is evaluated are the following: the ability of the attention module to learn a good attention policy, the quality of the learned representations of the encoder to be used in an action or goal recognition task (as was done in Chapter 4), the ability of the system to predict actions at a given horizon and manage the involved uncertainty, and the ability of the system to select actions that lead to a given goal. The first three skills are evaluated using the KIT Bimanual Manipulation Dataset described in Section 5.2.1, while the action selection one is evaluated using the synthetic action and plan input described in Appendix 4.A for convenience during testing. The KIT dataset sequences have been divided, using stratification, into a training set ($\sim$60% of the sequences), a validation set ($\sim$20% of the sequences), and a test set ($\sim$20% of the sequences). The U-LRNN used for the KIT dataset includes the attention system described in Section 5.2.2 between its input donwsampling and temporal window layers, and its depth has been set to $N = 4$ LRBs. To make sure that the top levels learn to properly represent goals, when training the top LRB, the system is only fed with long sequences from the dataset, avoiding short sequences, which consist of individual actions. As suggested in Section 4.5, multiple classifiers have been used at each level of the encoder during search, with the corresponding classification criteria appropriately chosen to suit each level. In particular, the left primitives and left main objects classification criteria have been used for the attention block, left primitives and left subactions for the first LRB, left subactions and actions for the second LRB, actions for the third LRB, and actions and goals for the top LRB. Classification criteria corresponding to the right hand are not considered because the data is symmetric (thanks to the mirror augmentation), as well as the system architecture and the self-supervised training process. In this way, the number of classifiers to train is reduced. Still, the randomness involved could lead to systems arbitrarily biased towards one of the hands and to the search algorithm selecting systems biased towards (in this case) the left hand. The

results obtained seem to indicate that this is not happening (not to a large extent at least), supporting the decision of considering only one hand during search. Regarding the U-LRNN for synthetic data, the architecture and methodology described in Chapter 4 are used with some minor differences. These differences include those described in Section 5.2.3 (making the system causal and keeping the hidden state) and using F_1 scores instead of accuracies in the fitness function. Therefore, the HLRNN hyperparameter search and training processes have been repeated. In addition, to compensate for the cumulative decrease in performance coming from those minor differences, multiple classification criteria have been used for the hyperparameter tuning of each level: primitives, actions, and plans for the first LRB, actions and plans for the second and third LRBs, and plans for the top LRB. Preliminary results seem to indicate that, as expected, this approach leads to configurations that learn more general-purpose representations at the different levels, with the hierarchy presenting a more gradual (or slower) increase in the abstraction levels, probably requiring deeper architectures to get the best high-level representations. Regarding the choice of F_1 scores for the fitness function, the synthetic data is quite balanced, so it shouldn't be substantially affected, but the KIT dataset is imbalanced (especially in its lower-level classification criteria), making the F_1 score a better choice.

Some hyperparameters have been set manually to reduce the search space and, in this way, simplify the search. In the attention system, the number of heads has been set to $h = 3$ (one head per hand plus one more head for a possibly involved additional object), the hidden layers of the GRU and feedforward network of the circuit that generates the query have been constraint to have an equal size (i.e., $s_{q,gru} = s_{q,h}$), and likewise for the training circuit (i.e., $s_{\hat{x},gru} = s_{\hat{x},h}$). The constraints on the hyperparameters of the LRBs are those described in Chapter 4: squared locally recurrent layers ($l_h = l_w$), $n_{\hat{x}} = 5$ reconstructed/predicted timesteps, equal weights in the error loss term ($w_{\hat{x}} = 1$), dropout masking probability and Gaussian noise standard deviation of $p = \sigma = 0.2$, and slowness term sample distance of $\delta = 1$. All the classifiers used are shallow vanilla recurrent neural networks (RNN), i.e., a recurrent layer with tanh activation functions followed by a feedforward layer with a softmax activation function. The number of samples used to train each self-supervised block are 2,000,000 and 500,000 for the KIT data and synthetic data, respectively. These numbers refer to the samples at the block, i.e., correspond to a number of input samples larger by the downsampling factor. Regarding the classifiers, the number of training samples are 100,000 and 10,000, respectively, with this number referring to the input samples. The number of validation samples used to obtain the metrics for the fitness functions is 200,000 for both datasets.

5.3.1 Attention System

Attention systems can be evaluated through extrinsic or intrinsic measures, with extrinsic measures referring to common performance metrics of the complete system containing the attention

module, such as accuracy or F_1 score, and intrinsic measures referring to those that evaluate specific behavior of the attention system itself [227]. In this section, we are interested in an intrinsic evaluation of our attention system. This evaluation can consist of well-defined metrics and/or of direct subjective inspection of the weights predicted for each sample, with both approaches often assuming actual human attention as ground truth. Two common intrinsic metrics consist of comparing the input vectors that the system pays attention to against previously manually annotated ground truth vectors and comparing them against experimentally measured actual human behavior. In our case, we don't count on such annotated or measured data. Still, we have data on the main object involved per subaction and hand. This annotation does not perfectly fit what would be the annotated ground truth vectors (e.g., at the beginning of a grabbing action, we may not know what object to pay attention to, and, at the end of a release action, we may not be interested in the released object anymore), but it is strongly correlated and can provide an idea of how well the attention system is working. Hence, we have defined this metric (for each hand) as the percentage of time at which the maximum weight of at least one attention head corresponds to the main object, considering only the time labeled with subactions with known main objects. The metric has been calculated separately for each hand (since the trained system is not necessarily symmetric) and from a total of 200,000 samples, obtaining as results 68.5% for the left hand and 67.9% for the right hand, which seem quite good considering that the annotations used are not accurate ground truth. In any case, a visual evaluation (commented below) can help us better understand these results. On the other hand, both percentages are very similar, which supports the decision of considering only left hand classifiers during search.

The previous metric provides an idea of whether the attention mechanism has learned to pay attention to the appropriate objects. However, an attention system may perform well according to this metric and still provide blurry outputs because the attention is relatively distributed among a number of objects and not strongly focused on a single one. To measure the quality of our system in this sense, we have defined a new metric that evaluates the attention head that is paying attention to the main object in terms of how focused it is. This metric only considers the time at which the maximum weight of at least one attention head corresponds to the main object and consists of the average of those maximum weights (if there are multiple heads paying attention to the main object, it only considers the head with the maximum weight among them). Again, 200,000 samples have been used to obtain this metric for each hand, obtaining average weight values of 0.971 for the left hand and 0.972 for the right hand. These results show that (at least when the system is paying attention to the main object) the attention system is very focused on the object. In addition, the values obtained for both hands are very similar.

To have a better understanding of the metrics obtained and how the system actually works, a more qualitative analysis of the system behavior has been carried out by observing the evolution of the attention weights during different recordings. From this analysis, we can confirm that

the system is indeed paying attention in general to the objects we expect it to pay attention to: When the subject is not manipulating any object, the system is generally paying attention to objects in front of the subject, and, as one hand approaches some object, the attention quickly shifts towards that object, usually before the object starts being manipulated. However, while the most commonly observed behavior suits what we are looking for, sometimes the system also shows undesired behaviors. Maybe the most relevant issue observed is that the system seems to be biased too strongly towards objects just in front of the subject. This bias was expected because the position representation changes faster for objects in that region (due to the tanh mapping). In fact, this phenomenon can be positive in general, but it may be problematic when the manipulated object is not in that region and other objects are. For instance, when using the broom, the attention often shifts between the broom and some other object as the subject moves the broom back and forth. Sometimes, a similar behavior is observed even with objects close to the subject, with the attention shifting for short periods to some arbitrary object and then coming back to the manipulated object. This may also be occurring because the system needs to correct or refresh the actual coordinates of those static objects (in fact, we humans may act sometimes in similar ways). Another bias of the system (even though considerably weaker) is its preference for some objects over others, causing, sometimes, one of the attention heads to be focused on some irrelevant object. This may be due to the fact that some objects are more frequently manipulated than others, which may affect the training in this direction. Finally, a pretty common phenomenon is having two heads paying attention to the same object. This is not surprising, for instance, when the subject is using the rolling pin, as it uses both hands to manipulate it. However, when the subject is manipulating two objects, two heads are sometimes focused on one of them and none on the other one, shifting in some cases the attention between the two objects. This behavior is harder to explain considering that the training circuit of the attention system would have more information to perform better predictions if the different heads were focused on different objects. In any case, further analysis is required if we want to better understand these phenomena and improve the design of the system and the quality of the learned attention weights.

5.3.2 Other Experiments

While the results are not available yet, this section describes the ongoing experiments regarding the U-LRNN for action and goal recognition, prediction, and selection. To test the system on **action and goal recognition**, the procedure described in Chapter 4 is followed using the KIT dataset, i.e., an RNN classifier per classification criterion to test is set at the top of each HLRNN level (to work with a fixed-size input, level 0 corresponds to the top of the attention system), and the accuracy and F_1 score evolution curves with respect to the number of classifier training samples are obtained. These curves provide an idea of how good the learned representations

are at encoding information related to each tested classification criteria in terms of how well a simple classifier on top of them can perform and how many training samples it requires to achieve acceptable performances. The classification criteria considered are left primitives, left subactions, actions, and goals (again, we have assumed that the performance obtained for the left hand is representative for the right hand). While the HLRNN was already evaluated in these terms on data from the same domain (synthetic action and plan data and real inertial action data), this experiment can provide information on how our system performs on a real and more complete action and goal recognition scenario where relevant information about the objects involved is available and where we count on classification criteria at different levels for the evaluation. On the other hand, note that the architecture is slightly different from that in Chapter 4, with the HLRNN being now causal and keeping the hidden state across batches.

The task of **action prediction** is evaluated at level 3 of the hierarchy. This evaluation consists of feeding the system with long sequences (i.e., plans) and classifying, in terms of actions, the predicted representations at a horizon that is approximately the duration of one action (this is similar to predicting the next action). To do so, an action classifier is first set and trained at level 3 of the HLRNN (this time on top of the temporal max pooling layer), defining its hyperparameters through Bayesian optimization as described above. Then, the system is fed with long sequences, generating, per representation sample at level 3, a probability distribution of the next predicted representation sample. 32 samples are generated from this distribution. Each generated sample is refed to the system through the LRB at level 4, generating a new distribution from which one sample is extracted to feed again the system. This process is repeated 1000 times (equivalent to 10 seconds of data, which is approximately the average action duration), leading to 32 predicted samples at the desired horizon that we use as an approximation of the probability distribution at that horizon. Finally, by feeding the classifier with these samples, we obtain the predicted probability of each action. Considering the usual uncertainty among a small number of plausible next actions, we are not only interested in the most likely candidate according to the prediction but also in other plausible candidates. Therefore, to have a more complete idea of how well this prediction system is performing, we have defined three intuitive metrics: the accuracy (i.e., how often the most likely prediction is correct), how often the real label is among the 32 predicted labels (i.e., whether the system is considering the actual action as a plausible candidate), and how many plausible candidates the system proposes on average (a high value would correspond to a weakly informative distribution and would make the previous metric meaningless). To have a more profound idea of how good the system is at proposing next possible candidate actions, the actual relationships and constraints among actions and objects within each plan could be defined manually, evaluating the system in terms of this ground truth. This has been left for future work.

To test the U-LRNN on **action selection**, the synthetic action and plan data from Appendix 4.A

is used instead of the KIT dataset. For this purpose, a synthetic action and plan simulation environment similar to that of the input has been implemented. The system is fed, at each instant, with an input sample obtained from the state of this environment (as well as with a one-hot encoding vector representing the desired goal label), generating a level 0 representation prediction (using an MDN, since the input data takes a sparse form) from which the hand movement (velocity coordinates and how open it is) at the next step is extracted. The system is evaluated on whether it is able to execute the plan successfully and on how long it takes to do so. The simulation environment has a limitation related to the nature of the original data: It cannot be deduced with total certainty when a pick, place, push, or pull event occurs from solely the environment state and hand movement information. To detect those events, we rely on information about the relative position of hand and objects, the changes in hand velocity, and how open the hand is. However, these events are sometimes wrongly detected, leading, e.g., to the undesired movement of objects. Another limitation comes from the fact that, in the original data, Gaussian noise is added to the target object positions. Therefore, they can end arbitrarily far from the original target position, making the achievement of goals hard to evaluate. To have a measure of when we consider a goal achieved, we have defined as acceptable those positions within a range of 3σ from the ideal position ($3\sqrt{2}\sigma$ when working with relative positions), where σ is the standard deviation of the mentioned Gaussian noise. We consider a goal as not achieved when it has not been achieved after 600 samples (which is more or less four times the duration of the longer goals). The final metrics are calculated as averages over a total of 256 trials, with the goal of each trial being selected randomly among horizontal line, vertical line, square, and triangle (the random shape goal is not considered). In this experiment, a qualitative analysis of the system through visual evaluation of its behavior given different goals can also help better understand the obtained metrics.

5.4 Discussion

The U-LRNN is a multi-purpose self-supervised learning neural network for temporal data that has been tested on the tasks of action and goal recognition, prediction, and selection. It consists of an optional low-level attention block that allows the system to handle variable-size inputs followed by an encoder-decoder architecture, with the encoder being an HLRNN that learns representations of the input at different levels of abstraction and the decoder consisting of a set of MDNs that perform predictions of the encoder representations at the different levels and timescales based on their current state and on higher-level context information. Its architecture and design is in part inspired by models of the neocortex, with the LRBs mimicking areas of the neocortex and the encoder and decoder emulating the feedforward and feedback connections of the neocortical hierarchies, respectively. This, together with its multiple applications and a design that eases the integration of new capabilities, makes it a good candidate to be extended

to a more complete cognitive architecture. Unfortunately, the unavailability of results makes it hard to comment on the performance of the system on the different tasks, only having results on the novel attention mechanism, which has been shown to be able to learn to pay attention to the relevant input information using a non-domain-specific self-supervised learning pretext task. Still, as mentioned in Section 5.1, the main purpose of this chapter was to propose architectures and directions on how to extend the HLRNN to more general-purpose systems, rather than to evaluate the performance of a specific implementation. In this section we elaborate on this, proposing alternative or further ways on how our system can be expanded and used.

One characteristic of our system is that it uses the same representations and circuits to recognize and predict the actions of others (and the environment) and to take decisions on the own actions. This is something common among imitation learning systems and is also coherent with the theory of mind known as simulation theory, which states that, to recognize or predict the actions or intentions of others, we use our mind as a model of theirs [20]. In particular, the well-known mirror neurons in our brain back this theory, as they fire both when we perform an action and when we observe that same action, and seem to be involved in mechanisms such as imitation or intention understanding [234]. An example of a learning by demonstration system that relies on these ideas is the multiple timescales recurrent neural network (MTRNN) [235], which, similarly to our system, learns representations of the actions at different levels and timescales. In addition, it uses the same circuit and neurons to go up and down the hierarchy, being more brain-like in this sense. One step that could be taken in this direction regarding our system is merging the two decoders (prediction and selection) into a single one. This could be achieved by, first, adding, to the prediction hierarchy, a goal input as that at the top of the action selection hierarchy. In this case, this goal representation could be treated as a distribution with the probability of each goal, becoming a one-hot encoding vector only for action selection (alternatively, two context MDNs could be kept: one for prediction and one for action selection). On the other hand, each MDN at the action selection hierarchy should get as input from the higher level a representation of the complete distribution instead of a single selected sample. To keep a similar approach, a softer (more resonance-like) winner-take-all could be implemented, with the Gaussian with the highest peak being amplified and narrowed while the rest of Gaussians are inhibited and widened. This, besides making our system more brain-like, would also make its functionalities better integrated as a multi-purpose system. In a real application, both modes of working could be used at the same time in a human-robot collaboration scenario using two instances of the system, with one instance being used to understand and predict the actions and goals of the human and the other instance being used to take decisions on the robot actions (extra circuits and/or procedures communicating both networks would also be required in order to understand each one's role in the shared task and coordinate). Both instances could share the same weights, mimicking the mirror neurons in our brain.

In fact, the action selection process could also follow a similar procedure to that used for prediction: The generated sample representing the selected action could be refed to the system iteratively, allowing the system to somehow simulate the environment and understand to what possible states the selected sequence of actions leads. This could be repeated several times, for example, sampling the distributions instead of going for its peaks, obtaining in this way a number of candidate sequences of actions (i.e., plans) that can be evaluated in terms of how good they are at leading to the desired goal to then select the action according to that. If this led to very similar and unsuccessful candidate sequences, their diversity could be increased by reducing the probability of the regions where the samples came from (e.g., by reducing the mixture coefficient of the Gaussian with the closest peak). This action selection process could be repeated at each timestep to update the plans according to the real data, taking an approach similar to that of model predictive control. Similarly to prediction, this could be performed at different levels of the hierarchy depending on the use case. For example, if the generated candidate plans are preferred at a symbolic level, classifiers could be used to obtain the sequence of action labels (possibly combined with a temporal segmentation algorithm), and there would be no need to predict or select actions at the lower levels. The two modes of action selection could also be combined, using one or the other depending on the task, with the one we have used in this study being faster and more automatic and the one proposed here being slower and more deliberative, somehow analogous to the two brain modes of thought (system 1 and system 2, respectively) described in [236]. Alternatively, the slower mode could be used as a simulator of the environment to train and improve the policy of the faster mode through a reinforcement learning approach, similarly to how world models are often used (e.g., [232]). The slower form of action selection is also to some extent analogous to how our hippocampus seems to be involved in planning (and plan recognition), generating goal-directed trajectories and simulating outcomes of potential choices at the higher-order neocortical areas through replay-like mechanisms [237]. These replayed sequences are often temporally compressed or reversed. Implementing such a feature in our system could make it more optimal and versatile, e.g., by having context MDNs that directly perform longer-term predictions or that "predict" back in time to understand what may have happened before (e.g., generating past actions of an ongoing plan). The resultant generated sequences of high-level representations could be used as sequences of subgoals for the lower levels of the U-LRNN, which would generate the next lower-level action representations given the actual state and the given subgoal (something similar was done in [238], where, given the current state and a goal, the system generated subgoals that a low-level controller could achieve). Note however that this approach wouldn't be very helpful for our system to do long-term low-level predictions, as this would require to work again at the low-level timescales (unless we set context MDNs also at the low levels, but, then, the advantages brought by the hierarchy would be lost).

While much of the U-LRNN design has been inspired by the mammal brain, our system also relies on components and mechanisms that don't seem to behave in a brain-like way. For example, MDNs generate as output the parameters of a probability density function, something that seems unlikely to be occurring in the brain. Still, and despite its biases, our brain is quite good at dealing with uncertainty. To handle uncertain representations, our brain seems to rely on resonance, taking multiple processing stages to generate complete and stable representations that are useful for other tasks. Resonance may also be involved in understanding when the representations generated by this hierarchical resolution of uncertainty are good enough to be used for further tasks [239]. Taking this into account, an existing algorithm that could be used instead of MDNs to deal with uncertainty in a more brain-like way is adaptive computation time, which relies on a recurrent neural network through which each sample is passed multiple times, with the number of times being decided through an extra unit that outputs the probability at each pass that the output is good enough [240]. This approach would have the limitation that we don't get information about uncertainty or about different possible candidates. A popular way to address this and get an idea of uncertainty is by training and ensemble of models and relying on their (dis)agreement (e.g., query by committee [241]). Alternatively, to avoid the need of training multiple models, different dropout masks could be applied within the same model [242], or random noise could be added at the first step to generate multiple candidate representations (this is somehow similar to image-to-image diffusion models such as [243], which are also analogous to the brain in the sense that they consist of multiple denoising passes). Note however that the generated candidates would not necessarily be a good approximation of the actual probability distribution, as occurs with GANs and diffusion models. In this sense, an advantage of having good approximations of probability distributions and, thus, good measures of the (un)likelihood of the events observed after, is that they can be used to modulate the attention or learning accordingly (somehow mimicking the amygdala [244]), as well as to drive exploration within a reinforcement learning context.

So far, we have mainly commented on our system, alternative ways to use it and possible minor modifications. As we mentioned in Section 5.1, this system is arguably not yet a complete cognitive architecture, though, as it lacks some capabilities that a general-purpose system should have. To get a first idea of how close the U-LRNN is to being a full cognitive architecture, we can briefly evaluate it according to the criteria defined in [123]: information representation paradigm, perception, attention, action selection, memory, learning, reasoning, and metacognition. In terms of information representation, our system can be defined as a connectionist logic system. In terms of perception, the U-LRNN has been used with data of different nature as input (e.g., motion capture, audio, etc.), with the main requirements that the data has a time series nature and that each time sample takes a sparse representation form, often requiring preliminary modules for such adaptation. However, it has never been used in a multimodal manner, with no preferred

sensor data fusion strategy defined. In terms of attention, our system includes a perceptual attention module that can work with both variable-size and fixed-size inputs, transforming them to fixed-size outputs, which are in most cases more appropriate for subsequent processing. However, our system does not implement any form of cognitive attention that allows it to focus on specific information from a higher-level representation. In addition, the implemented attention is only bottom-up, lacking top-down attention mechanisms that allow the system to decide what information to focus on depending on the task or goal. In terms of action selection, our system is, in principle, able to determine both the next action and how to implement it in terms of motion thanks to its hierarchical nature. On the other hand, while it is not able to explicitly define plans, it can generate sequences of subgoals attending to a given goal, as described earlier. In addition, the system can probably deal with multiple goals simultaneously and with the corresponding soft priorities in a similar way to how it deals with uncertainty. In terms of memory, our system, similarly to other artificial recurrent neural networks, implements short-term (sensory and working) memory in the form of hidden neural states and temporal windows or kernels and long-term (semantic and procedural) memory in the form of weights and biases, and it does not include episodic memory mechanisms that allow it to remember specific events. In terms of learning, the self-supervised learning mechanism of our system can be seen as taking the role of different types of learning, such as perceptual learning, procedural learning, or non-associative learning. However our system does not implement any form of associative reinforcement-like learning. In terms of reasoning, our system mainly performs probabilistic and inductive learning, with no mechanisms included for deductive reasoning. Finally, in terms of metacognition, our system does not implement procedures of this nature, but the method described earlier of generating candidate sequences of actions, evaluating them, and carrying them out depending on whether they seem to take to the sought goal can be seen as an early form of it.

When it comes to building autonomous agents, one of the main limitations of the U-LRNN may be not having a high-level reasoning and planning module on top, function typically associated to the prefrontal cortex [245] (actually, note that a way to generate sequences of subgoals using our system was already proposed earlier in this section). A common approach to address this issue is to go for a hybrid solution, converting the learned representations to labels and then using symbolic reasoners or planners to take decisions. For instance, this symbolic system could generate more consistent sequences of subgoals that lead to a specific goal and feed them to the lower-level neural network. However, with this approach, the strength of our system of being fully unsupervised would be partly lost, as the new system would require supervised training for the labels, as well as the definition of the knowledge base for the reasoning and planning system. In fact, while reasoning and planning systems are typically symbolic, there have already been proposals of systems able to combine classical reasoning or planning systems with

neural networks while keeping an unsupervised learning approach, performing the reasoning in the latent space, such as [246]. The idea commented in Section 5.2.3 of relying on already seen representative samples (or parts of them) to be used as discrete symbols could also be brought back with this purpose. In a more brain-like direction, some studies have proposed neural networks designed specifically to directly perform high-level reasoning (e.g., [247]). In fact, something that is becoming popular recently when it comes to endowing subsymbolic systems with reasoning and planning capabilities is integrating them with large language models (e.g., [248]). Indeed, while not designed specifically for these tasks, large language models have been shown to perform quite well on them, especially when combined with the right prompting and reflection techniques, besides contributing with additional capabilities such as processing natural language, dealing with ambiguity, and certain level of common sense. Still, these approaches often require again classifiers for the actions and objects involved, with even the more multimodal solutions requiring action labels for action selection (e.g., [249]). On the other hand, in the context of cognitive architectures, such solutions don't seem appropriate when working with topics such as brain-like development or online/incremental learning, as those models are in general used as pre-trained systems (since they require massive amounts of resources to be trained).

Another limitation of our system is that it does not implement any reinforcement learning mechanism, relying only on imitation for action selection. Therefore, our system has no tools to fine-tune the learned motions and adapt them to its own physical structure and dynamics or to learn alternative motions or actions when imitation fails. In addition, reinforcement learning can be very useful to learn to pay attention to the most relevant information, either through active perception (e.g., by aiming the camera in a specific direction) or by learning what the most important components of the input (or internal) representations are and controlling an attention mechanism accordingly so that the following stages can mainly rely on them. This could be applied to our input attention system or to other attention systems acting over the internal representations of the different levels. Still, when applied over the internal representations, since different regions of the representation may have unrelated meanings and layouts, the typical attention implementation as a weighted sum of regions may not be the best option, and a gating mechanism such as that of LSTMs and GRUs, masking the irrelevant neurons (or regions of neurons, e.g., through Hanning windows) and possibly amplifying the relevant ones (as in dropout), may be more meaningful (and brain-like). An added advantage of this solution is that the system could still work without the attention mechanism, requiring in this case though a robust training strategy relying on the masking of regions if the attention mechanism is to be added later (if similar representations for different maskings are desired, the corresponding contrastive pretext task could be added). This robustness would also be useful to deal with incomplete information (e.g., when we don't know what the observed agent is seeing). Such

attention mechanism could be applied in a top-down manner, relying on the predictions from the decoder to deduce what information to pay attention to on the next timestep. This top-down mechanism would possibly reach our input attention system, allowing it to do better-informed decisions on what object to pay attention to next thanks to the higher-level information. Such a top-down attention mechanism would make our decoder responsible for three of the functions typically associated to the feedback connections in the neocortex: attention, expectation, and motor commands [250]. In addition, the system would include feedback connections going back to the encoder, being closer to a more brain-like system with the encoder and decoder integrated into a single hierarchy with feedforward and feedback connections. To avoid relying only on reinforcement learning to learn the attention weights, some additional self-supervised learning pretext tasks could be defined, such as predicting relative distances or velocities between objects. These pretext tasks could be more or less complex depending on the U-LRNN level. Finally, one limitation of our attention system (quite common among attention systems) is that it needs at least one object to pay attention to, not giving the option to the attention heads to pay attention to no object, which may be useful in some cases. This could be approached, for instance, by always having a "no object" option that the heads can pay attention to or by allowing the weights to sum up to less than one when the attention focus (i.e., the query) is far enough from every object (this is the case of the attention mechanism implemented for the synthetic actions and plans described in Appendix 4.A).

Regarding sensory multimodality, no preferred sensor fusion technique has been proposed for our system. This is something that can be addressed in a quite straightforward way by mimicking the posterior association area of the neocortex, which integrates incoming sensory information from unimodal sensory areas [251]. Hence, since our system is modeling the different areas of the neocortex using a same neural network structure, using stacks of LRBs for different sensory modalities (mimicking sensory cortical hierarchies such as the visual or auditory ones) whose concatenated top representations (synchronized and at a same timescale) are used as input to another (association) LRB or stack of LRBs seems a quite brain-like solution. Skip connections could also be included within each hierarchy to make the architecture more brain-like. Combining this with the robust training strategy proposed earlier consisting of masking regions, the system could be able to learn to understand and predict missing sensory modalities.

Finally, while the method proposed to train the U-LRNN is fully self-supervised, it doesn't seem appropriate for online learning if we want a system that can act (or perform in some way) already with relatively short training while improving its performance as it learns from its input and/or interacts with the environment. Indeed, the training method used in this study trains the whole HLRNN stage by stage before the modules of the decoder can be used at all, which are required, e.g., for action selection. Drawing inspiration from hierarchical neocortical maturation [189], one way to better approach this could be by first working with a single-level

version of the U-LRNN (i.e., a single LRB connected to a single MDN) and, then, gradually adding levels on top as the previous levels have gone through enough training. This process could be complemented with some learning curriculum that gradually increases the complexity of the input data to get the best out of it. If classifiers are needed, these classifiers could be trained first on top of the lower levels, and, then, as new levels are added, these could be added to the input of the classifiers (starting with very small weights) while the connections to the lower levels are possibly gradually removed, e.g., through pruning [252] (mechanism that is also present in our brain development [253]). Such a development process would also allow the previously proposed top-down attention mechanism to be integrated stage by stage, without the need of waiting to have the whole encoder hierarchy trained. Finally, if the hyperparameters of the system are to be determined through hyperparameter tuning and gradually, this approach would allow a fully unsupervised search, without the need for classifiers.

5.5 Conclusion

In this chapter, we have described the U-shaped locally recurrent neural network (U-LRNN), which is an extension of the HLRNN introduced in Chapter 4 that adds a decoder at the end of it that performs predictions of the internal representations of the HLRNN at every level in a top-down manner. It also includes an optional novel self-supervised attention system at its input that allows the system to deal with variable-size inputs and to take advantage of an intra-sample attention mechanism when it is appropriate for the type of input. The resultant system, fully self-supervised, can be used for a number of purposes besides those of the HLRNN, such as prediction or action selection, and is designed so that it can be extended to more multi-purpose architectures and, eventually, become a complete cognitive architecture that can be used, for example, to control fully autonomous agents. In addition, its multiple analogies with our brain at different levels makes it a good candidate model of it, possibly contributing to a better understanding of how cognition emerges. To test the U-LRNN and its hierarchical structure on the tasks of action and goal recognition and prediction, an existing human action and plan motion capture dataset labeled at different levels and with position and orientation information of the body parts and objects was selected. To make those tasks more challenging and the corresponding results more meaningful, a number of new plans were designed and added to the datasets, and several augmentations were also implemented. In addition, since the U-LRNN requires sparse representations at its input, a sparse representation of the motion capture data was proposed. This representation was designed so that it is coherent across objects and easily extendable to new ones, relying on geometric and appearance properties that can be determined automatically. While results on the full U-LRNN are still not available, these representations have led to good results on the new attention mechanism.

Several ways have been proposed on how to extend the U-LRNN to a more general-purpose

architecture, most of them drawing inspiration or being somehow analogous to our brain structure or function. To make the system better equipped for high-level reasoning and long-term planning, different approaches can be taken: from simply adding on top several recurrent units or transformers and applying multi-horizon predictive methods to generate multiple candidate plans to integrating the system with classical reasoners and planners or with large language models. The system also lacks reinforcement learning mechanisms that complement imitation learning and allow the system to better learn and fine-tune how to perform actions. Regarding attention, top-down attention mechanisms applied over the internal representations could contribute to a better performance of the system. These mechanisms could come together with a more robust training that allows the system to better learn associations among regions of the representations. The system could also be made multimodal in a quite straightforward way thanks to its unified way of processing different modalities, by simply concatenating outputs of unimodal encoder branches and using them as input to a multimodal branch. A system that integrates and is able to successfully perform the tasks mentioned so far could probably be considered general-purpose. If we also want a system that develops and improves its performance gradually while it interacts with and learns from the environment, e.g., for developmental robotics research, we can construct and train our system step by step starting from a single-level U-LRNN with a limited functionality that, as it learns stable representations of its input, is extended with more levels on top until it reaches the complete architecture.

Chapter 6

CONCLUSION

This thesis has presented a new neural network architecture that can be used to recognize actions and goals in human-robot interaction scenarios with different requirements, such as real-time recognition or online learning. The proposed network draws inspiration from the structure and function of the brain and was designed with versatility in mind, so that it can be adapted to different scenarios and even to different applications and domains, as well as extended to or integrated into more general-purpose architectures. In fact, it is precisely this sought versatility that motivated the brain inspiration, as the human brain is a general-purpose system with all the capabilities and flexibility to achieve natural and intuitive interactions with humans. The proposed system consists mainly of a neural network that learns representations of its input temporal data at different levels of abstraction in a self-supervised way. These representations can then be used in different ways and for different purposes. In this study, the quality of the learned representations has been tested within action and goal (as well as speech) classification tasks, showing that the system is able to outperform other similar state-of-the-art systems. In addition, the system has been extended to perform other tasks (action and goal prediction and action selection), showing how these representations can be used in different tasks in a more complete brain-inspired system. Further directions on how to keep expanding the system have also been proposed.

6.1 Summary of Contributions

The main contribution of this thesis is the proposed brain-inspired multi-purpose self-supervised representation and prediction learning neural network for temporal data known as U-LRNN (U-shaped locally recurrent neural network). This system has been shown to be able to learn better representations of action and plan data than other systems of its kind. Similarly, it has outperformed other systems in speech data, showing that it is also applicable to temporal data from other domains. The system can also be used for different purposes, being the configurations proposed for action and goal recognition, prediction, and selection examples of how it can be adapted. Indeed, the system has been designed to be easily extendable with further functionality and adaptable to other tasks and applications, and ideas on how to proceed in those directions have been provided. On the other hand, the system shows analogies to the brain in terms of structure at different levels, and it has also shown analogies in terms of behavior, making it a potential model of regions of the brain that can be used to simulate them and achieve a deeper understanding of them and their emerging properties. In addition to this main contribution, this

thesis has proposed a number of novel architectures, methods, and ideas that contribute to the knowledge in the field and that we comment on below:

Chapter 2 has introduced a generic formalization of the problem of action, plan, and goal recognition that is applicable to most variants of the problem. It addresses the issue of having many different scenario-specific formalizations in the literature that use same terms to refer to different concepts, making its review slow and confusing and the systems harder to compare. Thus, using a common formalization and nomenclature (as the ones proposed in this chapter) across research studies can considerably simplify their review and minimize misunderstandings, as well as enable direct comparisons. In addition, the formalizations of the environment and the agents proposed in this chapter are generic enough to be also applicable to other problems with interacting agents involved.

Chapter 3 has introduced the elementary block of the U-LRNN encoder, known as NILRNN (neocortex-inspired locally recurrent neural network). This block is a shallow neural network whose design mimics the feedforward circuits of an area of the neocortex, learning higher-level representations of its input in an unsupervised way. Its internal design and sought functioning is inspired by a computational model of the primary visual cortex that proposes an implementation of the slowness principle through a biologically plausible connectivity pattern, allowing the system to extract higher-level information from the structure of the input temporal data. In particular, the NILRNN implements a novel form of low-level semantic pooling that tries to generalize the spatial pooling of CNNs to types of data other than images by exploiting the temporal nature of the data. The NILRNN has been shown to be able to learn higher-level representations of its input data when tested on classification tasks with data from different domains. In addition, it has shown analogous behavior to that of the neocortex when fed with sequences of images, which indicates that, first, the emerging behavior is the one sought during its design and, second, it is a valid model of the primary visual cortex (and possibly of other areas of the neocortex) and can be used to study it.

Chapter 4 has introduced the encoder of the U-LRNN, known as HLRNN (hierarchical locally recurrent neural network). This network, mainly consisting of a set of stacked NILRNNs, mimics the feedforward connections of the hierarchies in the neocortex and is the part of the U-LRNN responsible for learning representations of the input at different levels of abstraction, with the higher levels working at slower timescales, which brings advantages in terms of computation and performance. Besides outperforming other state-of-the-art self-supervised representation learning systems, the learned representations of the input have indeed been shown to be at different levels of abstraction, being more or less appropriate for downstream tasks depending on the specific tasks. This chapter has also proposed a number of modifications to the NILRNN to improve its learned representations and make it more robust. An ablation study on the HLRNN

has shown that those modifications contribute to the overall performance of the system.

Finally, Chapter 5 has introduced the U-LRNN itself, adding a decoder to the HLRNN that predicts, in a top-down manner (roughly mimicking the feedback connections in the neocortex), the next representations of the HLRNN at the different levels. These predictions take the form of probability distributions. Therefore, they can be sampled and refed to the system multiple times, obtaining variable-horizon predictions and an idea of the uncertainty of those predictions. The chapter has also presented the specific U-LRNN configurations used for the tasks of action prediction and action selection and has proposed a number of directions on how the system can be further adapted or expanded with new functionalities and for other applications. On the other hand, this chapter has proposed a set of augmentations and procedures to adapt motion capture data of human and objects in an action and plan scenario to our system. This procedures include, among others, converting the positions, orientations, and object identifiers to a novel sparse representation that presents a number of advantages, such as being more coherent across objects or being more appropriate to deal with uncertainty. In particular, the proposed orientation representation does not suffer from the typical issues that orientation representations usually face, and it is able to represent symmetric objects in a consistent way, with the representation being invariant to symmetry transformations. Finally, an attention system has been proposed that learns, in a self-supervised way, to pay attention to part of the data in each input sample based on the information from previous samples. This attention system allows the subsequent system to focus on the relevant part of the input as well as to deal with variable-size data (such as data corresponding to a variable number of objects), and it has shown a good and predictable performance.

6.2 Future Directions

While the proposed system has already shown a good behavior in a number of classification tasks and has been shown to be adaptable to different problems, there is much open in terms of better understanding and improving its behavior, as well as in terms of extending it with more functionality.

6.2.1 Design Improvement

The NILRNN is one of the core elements of the proposed system, and, while its design has been already improved in Chapter 4 and its functioning analyzed both through ablation studies and through comparison against the neocortex, there are still things about its behavior that are not well understood. For instance, the interaction between the locally recurrent layer and the max pooling layer is still not fully clear, as well as when or where tanh or ReLU activation functions should be used. A better understanding of its behavior when used over input data of different types can help find its weaknesses and improve its performance. Some ideas on

how this can be performed, comparing the system with alternative implementations of the same principles, are proposed in Section 3.5. On the other hand, defining new self-supervised pretext tasks may also improve its performance. For example, masking regions of its input during training, where the neurons are often strongly correlated, may contribute to the robustness of the learned representations more effectively than the currently implemented naive dropout, forcing the network to learn associations across those regions.

Finding better NILRNN designs is probably the best way to improve the HLRNN performance, but there are also other approaches that can lead to this improvement. For instance, one relevant limitation of the HLRNN is that it requires sparse representations at its input, with the proposed solution in Chapter 4 of using a deep version of the NILRNN addressing the issue to some extent but still not performing as well as desired. While this is something that can be researched upon, the solution will most likely not be brain-like, since the brain seems to preprocess each sensory input modality in a different specific way. Other directions that can lead to an improvement of the HLRNN performance consist of modifying its general architecture, e.g., by introducing skip connections between blocks, something analogous to what is found in the hierarchies of the neocortex. Using variable length temporal kernels and downsampling factors that adapt according to how fast the input changes is also something that can be investigated. Finally, the training approach can be modified, going, for instance, for an end-to-end training with a new set of pretext tasks.

When it comes to the U-LRNN, both the attention system and (especially) the decoder have gone through much less testing than the rest of the architecture, which suggests that there is much more room for improvement. For instance, using MDNs as prediction blocks may have not been the best choice (besides not being brain-like), with alternatives such as VAEs, resonance-like RNNs, or difussion models possibly reaching better performances. Whether a GRU or a transformer or other type of neural network is more appropriate for the top level is also a question that remains open, as well as what pretext tasks should be used at this level for the training. These questions demand an analysis that may lead to a better understanding of the network and, eventually, to a much better performing U-LRNN. Merging the two decoder branches (prediction and action selection) into one is also something that should be considered. Regarding the attention system, while a quite good and predictable behavior has been achieved, it is still not ideal, and a study on why sometimes it does not perform as desired may lead to a better architecture.

6.2.2 Functionality Extension

A number of different ways on how to extend the proposed system and possibly transform it into a more general-purpose cognitive architecture have been proposed along this dissertation, and especially in Section 5.4. These ideas include adding a system on top with reasoning and planning capabilities, adding reinforcement learning functionality that complements the

implemented imitation learning mechanisms, adding internal attention mechanisms, making the system multimodal, and training it and developing it in more incremental ways so that it can be deployed untrained in online learning applications (e.g., in developmental robotics applications).

Three different ideas to enhance the system with reasoning and planning capabilities are using a classic reasoning and planning system, using large language models, and adding longer-horizon back and forth predictors on top that can generate candidate sequences of high-level representations and reason according to them. While the last solution is probably the more brain-like and also the one that fits best the design principles of our system, it is also (in principle) the most limited one in terms of reasoning capabilities. The other two solutions are more powerful, but they probably introduce other constraints to the system, such as being less adapted to online learning applications. Thus, the best choice will in general depend on the application requirements.

Regarding the internal top-down attention mechanisms, these can contribute to having high-level representations that are more meaningful to the specific task by focusing only on the relevant information from the level(s) below. The attention weights can be generated from the decoder predictions, somehow replicating one of the functions associated to the feedback connections of the neocortex. This can also be seen as a first step towards merging the encoder and the decoder into a single hierarchy with feedforward and feedback connections, making our system more brain-like.

Making the system multimodal opens a number of new questions on how to preprocess the different types of data to adapt them to our system by making the corresponding representations sparse. For audio inputs, a (brain-like) spectrogram seems to perform well enough. Regarding video, while the system has neocortex-like behavior when fed with a sequence of images, adding some pairs of convolutional and max pooling layers at the input may lead to a better performance, as proposed in Section 4.5. Other types of input probably require defining specific preprocessing pipelines. Once the different modalities have been preprocessed, they can be fed to their corresponding HLRNN branches, whose outputs can be concatenated and fed to a common branch, mimicking the association areas of the neocortex.

6.2.3 Towards a More Brain-like System

Another possible direction of research is modifying the system to make it more brain-like in different ways. This, on the one hand, can be useful to use it as a model of the brain or of regions of it, but it can also lead to better performing or more efficient implementations. These modifications can come in the form of modifying the architecture and the patterns of connectivity of the system. For instance, the encoder and the decoder can be fully merged into a single hierarchy with feedforward and feedback connections. This could lead to a new version

of the NILRNN that models a whole neocortical area, and not only those layers that are more relevant for the feedforward circuits. Regarding the patterns of connectivity, considering that the circular kernels have been shown to perform better than the square ones, the same could apply to the shape of the NILRNN layers, and approximately circular layers could lead to a better behavior. The same applies to having the neurons in these layers distributed following hexagonal instead of rectangular grids.

The modifications can also come in the form of emulating known mechanisms from the brain. For instance, the backpropagation training mechanism and the self-supervised pretext tasks can be substituted by a more brain-like Hebbian rule complemented with other techniques. This should come with the implementation of homeostatic plasticity mechanisms that regulate the level of neural activity. One possible advantage of these mechanisms is that they can regulate directly the activity of the network, while the "equivalent" mechanisms implemented in our system (in the form of regulation loss terms such as the sparsity term) only have a direct influence on the learned weights. Some modified version of the well-established batch normalization or layer normalization machine learning techniques, with the data normalization parameters being gradually adapted during inference, could serve this purpose. Converting the system into a spiking neural network, adapting the different layers accordingly, and analyzing the new behavior, is also something to explore. An alternative more straightforward modification in a similar direction is working with complex-valued instead of real-valued neurons. Finally, these different modifications can be carried out together with the adaptation and implementation of our system in neuromorphic hardware, possibly making the system faster, more efficient, and more brain-like.

6.3 Concluding Remarks

The system presented in this thesis has been shown to perform well on the tasks it has been tested and has also shown much potential to be adapted to different tasks and domains and to be extended with multiple capabilities. Indeed, while the results obtained so far are good, the main strength of this system may rely on its capacity to be improved and expanded, both in terms of modifying its current architecture and methods to obtain a better performance on the tasks it is already able to address and in terms of enhancing it with new functionalities so that it can address new tasks. Such improvement probably requires a deeper analysis and understanding of its behavior, which may come through a more direct study of its internal mechanisms or through the testing of the system in different applications and environments. On the other hand, the analogies that the system shows with the brain can contribute to an advancement and to a bidirectional transfer of knowledge between the artificial intelligence and cognitive neuroscience fields, with the updates in the system possibly making it a better tool to model and study the brain and the new brain knowledge motivating more powerful and general-purpose brain-like

artificial intelligence systems.

BIBLIOGRAPHY

[1] Juan Angel Gonzalez-Aguirre, Ricardo Osorio-Oliveros, Karen L Rodríguez-Hernández, et al. "Service robots: Trends and technology". In: *Applied Sciences* 11.22 (2021), p. 10702.

[2] Jordan Abdi, Ahmed Al-Hindawi, Tiffany Ng, et al. "Scoping review on the use of socially assistive robot technology in elderly care". In: *BMJ open* 8.2 (2018), e018815.

[3] Lee Sproull, Mani Subramani, Sara Kiesler, et al. "When the Interface Is a Face". In: *Human–Computer Interaction* 11.2 (1996), pp. 97–124. DOI: 10.1207/s15327051hci1102_1. eprint: https://www.tandfonline.com/doi/pdf/10.1207/s15327051hci1102_1.

[4] Mengmeng Wang, Jiazheng Xing, Jianbiao Mei, et al. "ActionCLIP: Adapting Language-Image Pretrained Models for Video Action Recognition". In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[5] Yuichi Yamashita and Jun Tani. "Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment". In: *PLoS computational biology* 4.11 (Dec. 2008), e1000220. DOI: 10.1371/journal.pcbi.1000220.

[6] Franz A Van-Horenbeke and Angelika Peer. "Activity, plan, and goal recognition: A review". In: *Frontiers in Robotics and AI* 8 (2021), p. 643010.

[7] David Premack and Guy Woodruff. "Does the chimpanzee have a theory of mind?" In: *Behavioral and Brain Sciences* 1.4 (1978), pp. 515–526. DOI: 10.1017/S0140525X00076512.

[8] S.M. Schaafsma, D.W. Pfaff, R.P. Spunt, et al. "Deconstructing and reconstructing theory of mind". In: *Trends in cognitive sciences* 19.2 (2015), pp. 65–72. DOI: 10.1016/j.tics.2014.11.007.

[9] Catherine A. Hynes, Abigail A. Baird, and Scott T. Grafton. "Differential role of the orbital frontal lobe in emotional versus cognitive perspective-taking". In: *Neuropsychologia* 44.3 (2006), pp. 374–383.

[10] Janet Wilde Astington. "Sometimes necessary, never sufficient: False-belief understanding and social competence". In: *Macquarie monographs in cognitive science. Individual differences in theory of mind: Implications for typical and atypical development*. Psychology Press, 2003, pp. 13–38.

[11] Daniela Kloo, Josef Perner, and Thomas Giritzer. "Object-Based Set-Shifting in Preschoolers: Relations to Theory of Mind". In: *Self- and Social-Regulation: Exploring the Relations Between Social Interaction, Social Understanding, and the Development of Executive Functions*. 2010. DOI: 10.1093/acprof:oso/9780195327694.003.0008.

[12] C.F. Schmidt, N.S. Sridharan, and J.L. Goodson. "The plan recognition problem: An intersection of psychology and artificial intelligence". In: *Artificial Intelligence* 11.1-2 (1978), pp. 45–83. DOI: 10.1016/0004-3702(78)90012-7.

[13] P.R. Cohen, C.R. Perrault, and J.F. Allen. "Beyond question answering". In: *Strategies for Natural Language Processing*. 1981, pp. 245–274.

[14] K. L. Bierman, M.M. Torres, C.E. Domitrovich, et al. “Behavioral and cognitive readiness for school: Cross-domain associations for children attending Head Start”. In: *Social Development* 18.2 (2009), pp. 305–323. DOI: 10.1111/j.1467-9507.2008.00490.x.

[15] N. Alduncin, L.C. Huffman, H.M. Feldman, et al. “Executive function is associated with social competence in preschool-aged children born preterm or full term”. In: *Early human development* 90.6 (2014), pp. 299–306. DOI: 10.1016/j.earlhumdev.2014.02.011.

[16] Henry Wellman and David Liu. “Scaling of Theory-of-Mind Tasks”. In: *Child development* 75 (Mar. 2004), pp. 523–41. DOI: 10.1111/j.1467-8624.2004.00691.x.

[17] Matthew Ratcliffe. “Folk psychology’ is not folk psychology”. In: *Phenomenology and the Cognitive Sciences* 5 (Mar. 2006), pp. 31–52. DOI: 10.1007/s11097-005-9010-y.

[18] B.J. Scholl and A.M. Leslie. “Modularity, development and ’theory of mind.’” In: *Mind & Language* 14.1 (1999), pp. 131–153. DOI: 10.1111/1468-0017.00106.

[19] Michael Bratman. *Intention, Plans, and Practical Reason*. Harvard University Press, 1987.

[20] Robert M. Gordon. “Folk Psychology as Simulation”. In: *Mind & Language* 1.2 (1986), pp. 158–171. DOI: 10.1111/j.1468-0017.1986.tb00324.x.

[21] V. Gallese, L. Fadiga, L. Fogassi, et al. “Action recognition in the premotor cortex”. In: *Brain : a journal of neurology* 119.2 (1996), pp. 593–609. DOI: 10.1093/brain/119.2.593.

[22] Giacomo Rizzolatti and Laila Craighero. “The Mirror-Neuron System”. In: *Annual review of neuroscience* 27 (Feb. 2004), pp. 169–92. DOI: 10.1146/annurev.neuro.27.070203.144230.

[23] Vittorio Gallese, Christian Keysers, and Giacomo Rizzolatti. “A Unifying View of the Basis of Social Cognition”. In: *Trends in cognitive sciences* 8 (Oct. 2004), pp. 396–403. DOI: 10.1016/j.tics.2004.07.002.

[24] Marco Iacoboni, Roger Woods, Marcel Brass, et al. “Cortical mechanisms of human imitation”. In: *Science, v.286, 2526-2528 (1999)* (Jan. 1999).

[25] Marco Iacoboni, Istvan Molnar-Szakacs, Vittorio Gallese, et al. “Grasping the Intentions of Others with One’s Own Mirror Neuron System”. In: *PLoS biology* 3 (Apr. 2005), e79. DOI: 10.1371/journal.pbio.0030079.

[26] Vittorio Gallese and Alvin Goldman. “Mirror neurons and the simulation theory of mind-reading”. In: *Trends in Cognitive Sciences* 2 (1998), pp. 493–501.

[27] Gita Sukthankar, Christopher Geib, Hung Hai Bui, et al. “Introduction”. In: *Plan, Activity, and Intent Recognition. Theory and Practice*. 2014. DOI: 10.1016/B978-0-12-398532-3.00022-1.

[28] Sarah Keren, Reuth Mirsky, and Christopher Geib. *Plan Activity and Intent Recognition Tutorial*. Online. 2019.

[29] Charmi Jobanputra, Jatna Bavishi, and Nishant Doshi. “Human Activity Recognition: A Survey”. In: *Procedia Computer Science* 155 (Jan. 2019), pp. 698–703. DOI: 10.1016/j.procs.2019.08.100.

[30] Michalis Vrigkas, Christophoros Nikou, and Ioannis Kakadiaris. “A Review of Human Activity Recognition Methods”. In: *Frontiers in Robotics and Artificial Intelligence* 2 (Nov. 2015). DOI: 10.3389/frobt.2015.00028.

[31] Ronald Poppe. “A Survey on Vision-based Human Action Recognition. Image and Vision Computing”. In: *Image Vision Comput.* 28 (June 2010), pp. 976–990. DOI: 10.1016/j.imavis.2009.11.014.

[32] The Anh Han and Luís Pereira. “State-of-the-Art of Intention Recognition and its Use in Decision Making – A Research Summary”. In: *Ai Communications* 26 (Jan. 2013). DOI: 10.3233/AIC-130559.

[33] Yanxiang Xu, Tiejian Luo, Tingshao Zhu, et al. “The principles of intention computing”. In: *Proceedings - 2009 1st IEEE Symposium on Web Society, SWS 2009* (Aug. 2009). DOI: 10.1109/SWS.2009.5271806.

[34] Richard Kelley, Alireza Tavakkoli, Christopher King, et al. “Understanding Activities and Intentions for Human-Robot Interaction”. In: Feb. 2010. ISBN: 978-953-307-051-3. DOI: 10.5772/8127.

[35] Sandra Carberry. “Techniques for Plan Recognition.” In: *User Model. User-Adapt. Interact.* 11 (Mar. 2001), pp. 31–48. DOI: 10.1023/A:1011118925938.

[36] J. Shrager and T. Finin. “An expert system that volunteers advice”. In: *Proceedings of the Second National Conference on Artificial Intelligence*. 1982, pp. 339–340.

[37] Dorit Avrahami-Zilberbrand and Gal Kaminka. “Keyhole Adversarial Plan Recognition for Recognition of Suspicious and Anomalous Behavior”. In: *Plan, Activity, and Intent Recognition: Theory and Practice* (Mar. 2014), pp. 87–119. DOI: 10.1016/B978-0-12-398532-3.00004-X.

[38] R. Perrault and J. Allen. “A plan-based analysis of indirect speech acts”. In: *American Journal of Computational Linguistics* 6.3-4 (1980), pp. 167–182.

[39] Sarah Keren, Avigdor Gal, and Erez Karpas. “Goal Recognition Design”. In: *ICAPS*. 2014.

[40] Maayan Shvo and Sheila A. McIlraith. “Active goal recognition”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 06. 2020, pp. 9957–9966.

[41] Ruzena Bajcsy, Yiannis Aloimonos, and John K. Tsotsos. “Revisiting active perception”. In: *Autonomous Robots* 42.2 (2018), pp. 177–196.

[42] R. Mirsky, R. Stern, Y. Gal, et al. “Sequential plan recognition: An iterative approach to disambiguating between hypotheses”. In: *Artificial Intelligence* 260 (2018), pp. 51–73. DOI: 10.1016/j.artint.2018.03.006.

[43] Sarah Keren, Avigdor Gal, and Erez Karpas. “Goal Recognition Design with Non-Observable Actions”. In: *AAAI*. 2016.

[44] Christabel Wayllace, Ping Hou, William Yeoh, et al. “Goal Recognition Design with Stochastic Agent Action Outcomes”. In: *IJCAI*. 2016.

[45] Gal Kaminka, Mor Vered, and Noa Agmon. “Plan recognition in continuous domains”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[46] Mor Vered, Gal A. Kaminka, and Sivan Biham. “Online goal recognition through mirroring: humans and agents”. In: 2016.

[47] Hankz Hankui Zhuo. “Multiagent Plan Recognition from Partially Observed Team Traces”. In: *Plan, Activity, and Intent Recognition: Theory and Practice* (Mar. 2014), pp. 227–249. DOI: 10.1016/B978-0-12-398532-3.00009-9.

[48] Suchi Saria and Sridhar Mahadevan. “Probabilistic Plan Recognition in Multiagent Systems”. In: (Mar. 2004).

[49] Katie Genter, Noa Agmon, and Peter Stone. “Role-Based Ad Hoc Teamwork”. In: *Plan, Activity, and Intent Recognition: Theory and Practice*. Vol. 2. Jan. 2011. DOI: 10.1016/B978-0-12-398532-3.00010-5.

[50] Kennard Laviers, Gita Sukthankar, David Aha, et al. “Improving Offensive Performance Through Opponent Modeling”. In: Jan. 2009.

[51] Pavle Skocir, Petar Krivic, Matea Tomeljak, et al. “Activity Detection in Smart Home Environment”. In: *Procedia Computer Science* 96 (Dec. 2016), pp. 672–681. DOI: 10.1016/j.procs.2016.08.249.

[52] Jean Oh, Felipe Meneguzzi, and Katia Sycara. “Probabilistic Plan Recognition for Proactive Assistant Agents”. In: *Plan, Activity, and Intent Recognition: Theory and Practice* (Mar. 2014), pp. 275–288. DOI: 10.1016/B978-0-12-398532-3.00011-7.

[53] Eun Yong Ha, Jonathan P. Rowe, Bradford W. Mott, et al. “Recognizing Player Goals in Open-Ended Digital Games with Markov Logic Networks”. In: *Plan, Activity, and Intent Recognition: Theory and Practice*. 2014, pp. 289–311. DOI: 10.1016/B978-0-12-398532-3.00012-9.

[54] Lian Meng and Minlie Huang. “Dialogue Intent Classification with Long Short-Term Memory Networks”. In: Jan. 2018, pp. 42–50. ISBN: 978-3-319-73617-4. DOI: 10.1007/978-3-319-73618-1_4.

[55] Bruno Bouchard, Sylvain Giroux, and Abdenour Bouzouane. “A Keyhole Plan Recognition Model for Alzheimer’s Patients: First Results”. In: *Applied Artificial Intelligence* 21 (Aug. 2007), pp. 623–658. DOI: 10.1080/08839510701492579.

[56] Eric J. Horvitz, John S. Breese, David Heckerman, et al. “The Lumiere Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users”. In: *Fourteenth Conference on Uncertainty in Artificial Intelligence*. 1998, pp. 256–265.

[57] Safia Rahmat, Q. Niyaz, Ahmad Javaid, et al. “Application of deep learning as a pattern recognition technique in information security”. In: Nov. 2018, pp. 141–172.

[58] Sailik Sengupta, Tathagata Chakraborti, Sarath Sreedharan, et al. “RADAR-A Proactive Decision Support System for Human-in-the-Loop Planning”. In: *AAAI Fall Symposia*. 2017, pp. 269–276.

[59] Diliana Rebelo, Christoph Amma, Hugo Gamboa, et al. “Human Activity Recognition for an Intelligent Knee Orthosis”. In: *BIOSIGNALS*. 2013, pp. 368–371.

[60] Fariba Sadri. “Logic-Based Approaches to Intention Recognition”. In: *Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives* (Jan. 2011). DOI: 10.4018/978-1-61692-857-5.ch018.

[61] Marcelo G. Armentano and Analía Amandi. “Plan recognition for interface agents : State of the art”. In: *Artif. Intell. Rev.* 28 (Aug. 2007), pp. 131–162. DOI: 10.1007/s10462-009-9095-8.

[62] Henry A. Kautz and James F. Allen. “Generalized Plan Recognition”. In: *AAAI*. Vol. 86. 1986.

[63] Clint Heinze. “Modelling Intention Recognition for Intelligent Agent Systems”. In: 2004.

[64] Nazli Ikizler-Cinbis and Stan Sclaroff. “Object, scene and actions: Combining multiple features for human action recognition”. In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part I 11*. Springer. 2010, pp. 494–507.

[65] Hankz Hankui Zhuo. “Human-Aware Plan Recognition”. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. Ed. by Satinder P. Singh and Shaul Markovitch. AAAI Press, 2017, pp. 3686–3693.

[66] Yun Fu. *Human activity recognition and prediction*. Springer, 2016.

[67] Jindong Wang, Yiqiang Chen, Shuji Hao, et al. “Deep Learning for Sensor-based Activity Recognition: A Survey”. In: *Pattern Recognit. Lett.* 119 (2019), pp. 3–11.

[68] Oscar Lara and Miguel Labrador. “A Survey on Human Activity Recognition Using Wearable Sensors”. In: *Communications Surveys & Tutorials, IEEE* 15 (Jan. 2013), pp. 1192–1209. DOI: 10.1109/SURV.2012.110112.00192.

[69] Russell Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach 3rd ed.* Prentice Hall, 2016.

[70] Roger C. Schank and Robert P. Abelson. “Scripts, plans, goals and understanding, an inquiry into human knowledge structures”. In: *Journal of Pragmatics* 3.2 (1977), pp. 211–217.

[71] John R. Josephson and Susan G. Josephson. “Abductive Inference: Computation, Philosophy, Technology”. In: Cambridge University Press, 1994.

[72] Shirin Sohrabi, Anton Riabov, and Octavian Udrea. “Plan Recognition as Planning Revisited”. In: *IJCAI*. 2016.

[73] Peter Jarvis, Teresa Lunt, and Karen Myers. “Identifying Terrorist Activity with AI Plan Recognition Technology”. In: Jan. 2004, pp. 858–863.

[74] Henry A. Kautz. “Survey of Probabilistic Activity and Plan Recognition”. In: *Plan Recognition (Dagstuhl Seminar 11141)*. Vol. 1. 4. 2011.

[75] Luís Pereira and The Anh Han. “Elder Care via Intention Recognition and Evolution Prospection”. In: vol. 6547. Nov. 2009, pp. 170–187. DOI: 10.1007/978-3-642-20589-7_11.

[76] Sindhu Raghavan, Parag Singla, and Raymond Mooney. “Plan Recognition Using Statistical-Relational Models”. In: *Plan, Activity, and Intent Recognition: Theory and Practice* (Mar. 2014), pp. 57–85. DOI: 10.1016/B978-0-12-398532-3.00003-8.

[77] Paulo Quaresma and José Gabriel Lopes. “Unified logic programming approach to the abduction of plans and intentions in information-seeking dialogues”. In: *The Journal of Logic Programming* 24.1-2 (1995), pp. 103–119. DOI: 10.1016/0743-1066(95)00032-F.

[78] Mor Vered and Gal Kaminka. “Heuristic Online Goal Recognition in Continuous Domains”. In: Aug. 2017, pp. 4447–4454. DOI: 10.24963/ijcai.2017/621.

[79] Karen Myers. “Abductive Completion of Plan Sketches”. In: (Dec. 1997).

[80] Christopher Geib and Robert Goldman. “A probabilistic plan recognition algorithm based on plan tree grammars”. In: *Artificial Intelligence* 173 (July 2009), pp. 1101–1132. DOI: 10.1016/j.artint.2009.01.003.

[81] Miquel Ramirez and Hector Geffner. “Plan Recognition as Planning”. In: *Proc. 21st Intl. Joint Conf. on Artificial Intelligence, IJCAI*. Jan. 2009, pp. 1778–1783.

[82] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. “Landmark-based approaches for goal recognition as planning”. In: *Artificial Intelligence* 279 (2020), p. 103217.

[83] Hubert Dreyfus. “Detachment, Involvement, and Rationality: are we Essentially Rational Animals?” In: *Human Affairs* 17 (Dec. 2007), pp. 101–109. DOI: 10.2478/v10023-007-0010-0.

[84] Lin Liao, Dieter Fox, and Henry Kautz. “Learning and Inferring Transportation Routines”. In: *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*. Jan. 2004, pp. 348–353.

[85] Nuria Oliver, Eric Horvitz, and Ashutosh Garg. “Layered Representations for Human Activity Recognition”. In: *ICMI ’02: Proceedings of the 4th IEEE International Conference on Multimodal Interfaces* (Sept. 2002). DOI: 10.1109/ICMI.2002.1166960.

[86] Hung H. Bui, Dinh Q. Phung, and Svetha Venkatesh. “Hierarchical hidden Markov models with general state hierarchy”. In: *Proceedings of the National Conference on Artificial Intelligence*. July 2004, pp. 324–329.

[87] Liyue Zhao, Xi Wang, Gita Sukthankar, et al. “Motif Discovery and Feature Selection for CRF-based Activity Recognition”. In: Aug. 2010, pp. 3826–3829. DOI: 10.1109/ICPR.2010.932.

[88] Lin Liao, Dieter Fox, and Henry Kautz. “Extracting Places and Activities from GPS Traces Using Hierarchical Conditional Random Fields”. In: *I. J. Robotic Res.* 26 (Jan. 2007), pp. 119–134. DOI: 10.1177/0278364907073775.

[89] Chris Baker and Joshua Tenenbaum. “Modeling Human Plan Recognition Using Bayesian Theory of Mind”. In: *Plan, Activity, and Intent Recognition: Theory and Practice* (Mar. 2014), pp. 177–204. DOI: 10.1016/B978-0-12-398532-3.00007-5.

[90] Soumitra Samanta and Bhabatosh Chanda. “Space-Time Facet Model for Human Activity Classification”. In: *Multimedia, IEEE Transactions on* 16 (Oct. 2014), pp. 1525–1535. DOI: 10.1109/TMM.2014.2326734.

[91] Lin Fan, Zhongmin Wang, and Hai Wang. “Human Activity Recognition Model Based on Decision Tree”. In: *2013 International Conference on Advanced Cloud and Big Data* (2013), pp. 64–68.

[92] Alireza Vahdatpour, Navid Amini, and Majid Sarrafzadeh. “Toward unsupervised activity discovery using multi-dimensional motif detection in time series”. In: *IJCAI’09: Proceedings of the 21st international jont conference on Artifical intelligence*. July 2009, pp. 1261–1266.

[93] Artem Polyvyanyy, Zihang Su, Nir Lipovetzky, et al. “Goal Recognition Using Off-The-Shelf Process Mining Techniques”. In: (2020).

[94] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[95] Nils Y. Hammerla, Shane Halloran, and Thomas Plötz. “Deep, Convolutional, and Recurrent Models for Human Activity Recognition Using Wearables”. In: *IJCAI*. 2016.

[96] Antonio Bevilacqua, Kyle MacDonald, Aamina Rangarej, et al. “Human Activity Recognition with Convolutional Neural Networks”. In: Sept. 2018.

[97] Charissa Ronao and Sung-Bae Cho. “Human activity recognition with smartphone sensors using deep learning neural networks”. In: *Expert Systems with Applications* 59 (Apr. 2016). DOI: 10.1016/j.eswa.2016.04.032.

[98] Si Zhang, Hanghang Tong, Jiejun Xu, et al. “Graph convolutional networks: a comprehensive review”. In: *Computational Social Networks* 6.1 (2019), pp. 1–23.

[99] Sijie Yan, Yuanjun Xiong, and Dahua Lin. “Spatial temporal graph convolutional networks for skeleton-based action recognition”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[100] Leonardo Amado, João Aires, Ramon Pereira, et al. *LSTM-Based Goal Recognition in Latent Space*. Aug. 2018.

[101] Francisco Ordóñez and Daniel Roggen. “Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition”. In: *Sensors* 16 (Jan. 2016), p. 115. DOI: 10.3390/s16010115.

[102] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[103] Chiara Plizzari, Marco Cannici, and Matteo Matteucci. “Spatial temporal transformer network for skeleton-based action recognition”. In: *Pattern recognition. ICPR international workshops and challenges: virtual event, January 10–15, 2021, Proceedings, Part III*. Springer. 2021, pp. 694–701.

[104] Wentian Xin, Ruyi Liu, Yi Liu, et al. “Transformer for skeleton-based action recognition: A review of recent advances”. In: *Neurocomputing* (2023).

[105] Licheng Zhang, Xihong Wu, and Dingsheng Luo. "Real-Time Activity Recognition on Smartphones Using Deep Neural Networks". In: Aug. 2015, pp. 1236–1242. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.224.

[106] W. Min, E.Y. Ha, J. Rowe, et al. "Deep learning-based goal recognition in open-ended digital games". In: *Proceedings of the 10th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014* (Jan. 2014), pp. 37–43.

[107] Ankit Singh, Omprakash Chakraborty, Ashutosh Varshney, et al. "Semi-supervised action recognition with temporal contrastive learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 10389–10399.

[108] Toby Perrett, Alessandro Masullo, Tilo Burghardt, et al. "Temporal-relational crosstransformers for few-shot action recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 475–484.

[109] Alec Radford, Jong Wook Kim, Chris Hallacy, et al. "Learning transferable visual models from natural language supervision". In: *International conference on machine learning*. PMLR. 2021, pp. 8748–8763.

[110] Jiayang Wu, Wensheng Gan, Zefeng Chen, et al. "Multimodal large language models: A survey". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE. 2023, pp. 2247–2256.

[111] Yasuo Kuniyoshi, Y. Yorozu, Masayuki Inaba, et al. "From visuo-motor self learning to early imitation - A neural architecture for humanoid learning". In: *Proceedings - IEEE International Conference on Robotics and Automation*. Vol. 3. Oct. 2003, 3132–3139 vol.3. ISBN: 0-7803-7736-2. DOI: 10.1109/ROBOT.2003.1242072.

[112] J. Bonaiuto, E. Rosta, and M.A. Arbib. "Recognizing Invisible Actions". In: *Workshop on Modeling Natural Action Selection* (2005).

[113] Erhan Oztop, Daniel Wolpert, and Mitsuo Kawato. "Mental state inference using visual control parameters". In: *Brain research. Cognitive brain research* 22 (Mar. 2005), pp. 129–51. DOI: 10.1016/j.cogbrainres.2004.08.004.

[114] M. Haruno, D. M. Wolpert, and M. Kawato. "Mosaic model for sensorimotor learning and control". In: *Neural computation* 13.10 (2001), pp. 2201–2220. DOI: 10.1162/089976601750541778.

[115] Yiannis Demiris and Bassam Khadhouri. "Hierarchical, Attentive, Multiple Models for Execution and Recognition (HAMMER)". In: *Robot. Auton. Syst. J.* 54 (Jan. 2005), pp. 361–369.

[116] Laith Alkurdi, Christian Busch, and Angelika Peer. "Dynamic contextualization and comparison as the basis of biologically inspired action understanding". In: *Paladyn, Journal of Behavioral Robotics* 9 (Mar. 2018), pp. 19–59. DOI: 10.1515/pjbr-2018-0003.

[117] Gregor Schöner and John Spencer. *Dynamic Thinking: A Primer on Dynamic Field Theory*. Oxford University Press, 2016. DOI: 10.1093/acprof:oso/9780199300563.001.0001.

[118] Frank E. Ritter, Farnaz Tehranchi, and Jacob D. Oury. "ACT-R: A cognitive architecture for modeling cognition". In: *Wiley Interdisciplinary Reviews: Cognitive Science* 10.3 (2019), e1488.

[119] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, 2012. ISBN: 9780262301145.

[120] Leonardo Enzo Brito da Silva, Islam Elnabarawy, and Donald C Wunsch II. "A survey of adaptive resonance theory neural network models for engineering applications". In: *Neural Networks* 120 (2019), pp. 167–203.

[121] J. Hawkins, S. Ahmad, S. Purdy, et al. "Biological and Machine Intelligence (BAMI)". Initial online release 0.4. 2016.

[122] A. Oltramari and Christian Lebiere. "Using ontologies in a cognitive-grounded system: Automatic action recognition in video surveillance". In: *CEUR Workshop Proceedings* 966 (Jan. 2014), pp. 20–27.

[123] Iuliia Kotseruba and John K. Tsotsos. "40 years of cognitive architectures: core cognitive abilities and practical applications". In: *Artificial Intelligence Review* 53.1 (2020), pp. 17–94.

[124] Roger L. Granada, Ramon Fraga Pereira, Juarez Monteiro, et al. "Hybrid activity and plan recognition for video streams". In: *Proceedings of the 31st. AAAI Conference: Plan, Activity and Intent Recognition Workshop, 2017, Estados Unidos.* 2017.

[125] Leonardo Amado, Ramon Fraga Pereira, Joao Aires, et al. "Goal recognition in latent space". In: *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2018, pp. 1–8.

[126] Sharath Akkaladevi and Christoph Heindl. "Action Recognition for Human Robot Interaction in Industrial Applications". In: *IEEE International Conference on Computer Graphics, Vision and Information Security (CGVIS)*. Nov. 2015, pp. 94–99. DOI: 10.1109/CGVIS.2015.7449900.

[127] Franz A Van-Horenbeke and Angelika Peer. "NILRNN: A Neocortex-Inspired Locally Recurrent Neural Network for Unsupervised Feature Learning in Sequential Data". In: *Cognitive Computation* (2023), pp. 1–17.

[128] Franz A Van-Horenbeke and Angelika Peer. "The Neocortex-Inspired Locally Recurrent Neural Network (NILRNN) as a Model of the Primary Visual Cortex". In: *IFIP International Conference on Artificial Intelligence Applications and Innovations*. Springer. 2022, pp. 292–303.

[129] Martin Längkvist, Lars Karlsson, and Amy Loutfi. "A review of unsupervised feature learning and deep learning for time-series modeling". In: *Pattern Recognition Letters* 42 (2014), pp. 11–24.

[130] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. IEEE. 1999, pp. 1150–1157.

[131] Vibha Tiwari. "MFCC and its applications in speaker recognition". In: *Int J Emerg Technol* 1.1 (2010), pp. 19–22.

[132] Sebastian Ruder. “Neural transfer learning for natural language processing”. PhD thesis. NUI Galway, 2019.

[133] Ian T Jolliffe and Jorge Cadima. “Principal component analysis: a review and recent developments”. In: *Philos Trans R Soc A Math Phys Eng Sci* 374.2065 (2016), p. 20150202.

[134] Ronald A Fisher. “The use of multiple measurements in taxonomic problems”. In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[135] Ganggang Dong, Guisheng Liao, Hongwei Liu, et al. “A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images”. In: *IEEE Geoscience and Remote Sensing Magazine* 6.3 (2018), pp. 44–68.

[136] Nicolas Le Roux and Yoshua Bengio. “Representational power of restricted Boltzmann machines and deep belief networks”. In: *Neural computation* 20.6 (2008), pp. 1631–1649.

[137] Guoqiang Zhong, Li-Na Wang, Xiao Ling, et al. “An overview on data representation learning: From traditional feature learning to recent deep learning”. In: *The Journal of Finance and Data Science* 2.4 (2016), pp. 265–278.

[138] Jeff Donahue, Yangqing Jia, Oriol Vinyals, et al. “Decaf: A deep convolutional activation feature for generic visual recognition”. In: *International Conference On Machine Learning*. PMLR. 2014, pp. 647–655.

[139] Ting Chen, Simon Kornblith, Mohammad Norouzi, et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

[140] Huanru Henry Mao. “A survey on self-supervised pre-training for sequential transfer learning in neural networks”. In: *arXiv preprint arXiv:2007.00800* (2020).

[141] Christoph Feichtenhofer, Haoqi Fan, Bo Xiong, et al. “A large-scale study on unsupervised spatiotemporal representation learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 3299–3309.

[142] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[143] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, et al. “wav2vec 2.0: A framework for self-supervised learning of speech representations”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 12449–12460.

[144] Hubert Banville, Isabela Albuquerque, Aapo Hyvärinen, et al. “Self-supervised representation learning from electroencephalography signals”. In: *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE. 2019, pp. 1–6.

[145] Linus Ericsson, Henry Gouk, Chen Change Loy, et al. “Self-Supervised Representation Learning: Introduction, advances, and challenges”. In: *IEEE Signal Processing Magazine* 39.3 (2022), pp. 42–62.

[146] Senzhang Wang, Jiannong Cao, and Philip Yu. "Deep learning for spatio-temporal data mining: A survey". In: *IEEE Trans Knowl Data Eng* (2020).

[147] Yong Yu, Xiaosheng Si, Changhua Hu, et al. "A review of recurrent neural networks: LSTM cells and network architectures". In: *Neural computation* 31.7 (2019), pp. 1235–1270.

[148] Changhan Wang, Yun Tang, Xutai Ma, et al. "Fairseq S2T: Fast Speech-to-Text Modeling with Fairseq". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: System Demonstrations*. Suzhou, China: Association for Computational Linguistics, Dec. 2020, pp. 33–39. URL: https://aclanthology.org/2020.aacl-demo.6.

[149] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, et al. "Building machines that learn and think like people". In: *Behav Brain Sci* 40 (2017).

[150] Katarina Lukatela and Harvey A Swadlow. "Neocortex". In: *The corsini encyclopedia of psychology* (2010), pp. 1–2.

[151] M-Marsel Mesulam. "From sensation to cognition." In: *Brain: A Journal of Neurology* 121.6 (1998), pp. 1013–1052.

[152] Dwight J Kravitz, Kadharbatcha S Saleem, Chris I Baker, et al. "The ventral visual pathway: an expanded neural framework for the processing of object quality". In: *Trends in cognitive sciences* 17.1 (2013), pp. 26–49.

[153] Victor AF Lamme, Hans Super, and Henk Spekreijse. "Feedforward, horizontal, and feedback processing in the visual cortex". In: *Current opinion in neurobiology* 8.4 (1998), pp. 529–535.

[154] Rajeevan T. Narayanan, Daniel Udvary, and Marcel Oberlaender. "Cell Type-Specific Structural Organization of the Six Layers in Rat Barrel Cortex". In: *Frontiers in Neuroanatomy* 11 (2017), p. 91. ISSN: 1662-5129. DOI: 10.3389/fnana.2017.00091. URL: https://www.frontiersin.org/article/10.3389/fnana.2017.00091.

[155] Vernon B Mountcastle. "The columnar organization of the neocortex." In: *Brain: A Journal of Neurology* 120.4 (1997), pp. 701–722.

[156] Yoonsuck Choe. "Hebbian Learning". In: *Encyclopedia of Computational Neuroscience*. Ed. by Dieter Jaeger and Ranu Jung. New York, NY: Springer New York, 2015, pp. 1305–1309. ISBN: 978-1-4614-6675-8. DOI: 10.1007/978-1-4614-6675-8_672.

[157] Gina Turrigiano. "Homeostatic synaptic plasticity: local and global mechanisms for stabilizing neuronal function". In: *Cold Spring Harbor perspectives in biology* 4.1 (2012), a005736.

[158] Frank Tong. "Primary visual cortex and visual awareness". In: *Nature Reviews Neuroscience* 4.3 (2003), pp. 219–229.

[159] Tai Sing Lee and David Mumford. "Hierarchical Bayesian inference in the visual cortex". In: *JOSA A* 20.7 (2003), pp. 1434–1448.

[160] Shaul Hochstein and Merav Ahissar. “View from the top: Hierarchies and reverse hierarchies in the visual system”. In: *Neuron* 36.5 (2002), pp. 791–804.

[161] Vladimir K Berezovskii, Jonathan J Nassi, and Richard T Born. “Segregation of feedforward and feedback projections in mouse visual cortex”. In: *J Comp Neurol* 519.18 (2011), pp. 3672–3683.

[162] Charles D Gilbert. “Laminar differences in receptive field properties of cells in cat primary visual cortex”. In: *The Journal of physiology* 268.2 (1977), pp. 391–421.

[163] Daniel J Graham and David J Field. “Sparse coding in the neocortex”. In: *Evolution of Nervous S ystems* 3 (2006), pp. 181–187.

[164] Risto Miikkulainen, James A Bednar, Yoonsuck Choe, et al. *Computational maps in the visual cortex*. Springer Science & Business Media, 2006.

[165] Tom Binzegger, Rodney J Douglas, and Kevan AC Martin. “Topology and dynamics of the canonical circuit of cat V1”. In: *Neural Networks* 22.8 (2009), pp. 1071–1078.

[166] David H Hubel and Torsten N Wiesel. “Sequence regularity and geometry of orientation columns in the monkey striate cortex”. In: *J Comp Neurol* 158.3 (1974), pp. 267–293.

[167] Zheng Liu, James P Gaska, Lowell D Jacobson, et al. “Interneuronal interaction between members of quadrature phase and anti-phase pairs in the cat’s visual cortex”. In: *Vision research* 32.7 (1992), pp. 1193–1198.

[168] David H Hubel and Torsten N Wiesel. “Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex”. In: *The Journal of physiology* 160.1 (1962), pp. 106–154.

[169] Jan Antolik and James A Bednar. “Development of maps of simple and complex cells in the primary visual cortex”. In: *Frontiers in computational neuroscience* 5 (2011), p. 17.

[170] Richard Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* Springer, 2022. URL: `https://szeliski.org/Book/`.

[171] Laurenz Wiskott. “Slow feature analysis: A theoretical analysis of optimal free responses”. In: *Neural Computation* 15.9 (2003), pp. 2147–2177.

[172] Pietro Berkes and Laurenz Wiskott. “Slow feature analysis yields a rich repertoire of complex cell properties”. In: *Journal of vision* 5.6 (2005), pp. 9–9.

[173] Marian Stewart Bartlett, Javier R Movellan, and Terrence J Sejnowski. “Face modeling by information maximization”. In: *Face Processing: Advanced Modeling and Methods* (2002), pp. 219–253.

[174] Hisham E Atallah, Michael J Frank, and Randall C O’Reilly. “Hippocampus, cortex, and basal ganglia: Insights from computational models of complementary learning systems”. In: *Neurobiology of learning and memory* 82.3 (2004), pp. 253–267.

[175] Carlos Gershenson. *Design and control of self-organizing systems*. CopIt Arxives, 2007.

[176] James L McClelland. “How far can you go with Hebbian learning, and when does it lead you astray”. In: *Processes of Change in Brain and Cognitive Development: Attention and Performance XXI* 21 (2006), pp. 33–69.

[177] Wei Luo, Jun Li, Jian Yang, et al. “Convolutional sparse autoencoders for image classification”. In: *IEEE Trans Neural Netw Learn Syst* 29.7 (2017), pp. 3289–3294.

[178] Ah Chung Tsoi and Andrew Back. “Discrete time recurrent neural network architectures: A unifying review”. In: *Neurocomputing* 15.3-4 (1997), pp. 183–223.

[179] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[180] Allen Y Yang, Roozbeh Jafari, S Shankar Sastry, et al. “Distributed recognition of human actions using wearable motion sensor networks”. In: *J Ambient Intell Smart Environ* 1.2 (2009), pp. 103–115. URL: https://people.eecs.berkeley.edu/~yang/software/WAR/.

[181] Zohar Jackson, César Souza, Jason Flaks, et al. *Jakobovski/free-spoken-digit-dataset: v1. 0.8*. Ed. by Zohar Jackson. 2018. URL: https://github.com/Jakobovski/free-spoken-digit-dataset.

[182] Monzilur Rahman, Ben DB Willmore, Andrew J King, et al. “Simple transformations capture auditory input to cortex”. In: *Proceedings of the National Academy of Sciences* 117.45 (2020), pp. 28442–28451.

[183] Andrew Ng. *Deep Learning and Unsupervised Feature Learning Handouts*. 2011. URL: https://web.stanford.edu/class/cs294a/handouts.html.

[184] Yanmin Sun, Andrew KC Wong, and Mohamed S Kamel. “Classification of imbalanced data: A review”. In: *International journal of pattern recognition and artificial intelligence* 23.04 (2009), pp. 687–719.

[185] Grace W Lindsay. “Convolutional neural networks as a model of the visual system: Past, present, and future”. In: *Journal of cognitive neuroscience* 33.10 (2021), pp. 2017–2031.

[186] Gary G Blasdel. “Orientation selectivity, preference, and continuity in monkey striate cortex”. In: *Journal of Neuroscience* 12.8 (1992), pp. 3139–3161.

[187] Dario L Ringach, Robert M Shapley, and Michael J Hawken. “Orientation selectivity in macaque V1: diversity and laminar dependence”. In: *Journal of neuroscience* 22.13 (2002), pp. 5639–5651.

[188] Se-Bum Paik and Dario L Ringach. “Retinal origin of orientation maps in visual cortex”. In: *Nature neuroscience* 14.7 (2011), pp. 919–925.

[189] Taylor Chomiak and Bin Hu. “Mechanisms of hierarchical cortical maturation”. In: *Frontiers in cellular neuroscience* 11 (2017), p. 272.

[190] Silvan C Quax, Michele D’Asaro, and Marcel AJ van Gerven. “Adaptive time scales in recurrent neural networks”. In: *Scientific reports* 10.1 (2020), pp. 1–14.

[191] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, et al. “Dropout: a simple way to prevent neural networks from overfitting”. In: *J Mach Learn Res* 15.1 (2014), pp. 1929–1958.

[192] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. “Improving deep neural networks for LVCSR using rectified linear units and dropout”. In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 8609–8613.

[193] Justin Bayer, C Osendorfer, Daniela Korhammer, et al. "On Fast Dropout and its Applicability to Recurrent Networks". In: *Proceedings of the International Conference on Learning Representations*. 2014, p. 14. URL: http://arxiv.org/abs/1311.0701.

[194] Xiao Liu, Fanjin Zhang, Zhenyu Hou, et al. "Self-supervised learning: Generative or contrastive". In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[195] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning". In: *arXiv preprint arXiv:1605.09782* (2016).

[196] Shen Li and Bryan Hooi. "Neural PCA for Flow-Based Representation Learning". In: *arXiv preprint arXiv:2208.10753* (2022).

[197] Madeline C Schiappa, Yogesh S Rawat, and Mubarak Shah. "Self-supervised learning for videos: A survey". In: *arXiv preprint arXiv:2207.00419* (2022).

[198] George Zerveas, Srideepika Jayaraman, Dhaval Patel, et al. "A transformer-based framework for multivariate time series representation learning". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2021, pp. 2114–2124.

[199] Brian Kenji Iwana and Seiichi Uchida. "An empirical survey of data augmentation for time series classification with neural networks". In: *Plos one* 16.7 (2021), e0254841.

[200] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation learning with contrastive predictive coding". In: *arXiv preprint arXiv:1807.03748* (2018).

[201] Emadeldeen Eldele, Mohamed Ragab, Zhenghua Chen, et al. "Time-Series Representation Learning via Temporal and Contextual Contrasting". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 2352–2359. DOI: 10.24963/ijcai.2021/324.

[202] Ling Yang and Shenda Hong. "Unsupervised time-series representation learning with iterative bilinear temporal-spectral fusion". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 25038–25054.

[203] Yann LeCun, Leon Bottou, Genevieve B Orr, et al. "Efficient BackProp". In: *Lecture Notes in Computer Science* 1524 (1998), pp. 5–50.

[204] Ceyuan Yang, Yinghao Xu, Jianping Shi, et al. "Temporal pyramid network for action recognition". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 591–600.

[205] Qianli Ma, Zhenxi Lin, Enhuan Chen, et al. "Temporal pyramid recurrent neural network". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5061–5068.

[206] Dongha Lee, Seonghyeon Lee, and Hwanjo Yu. "Learnable dynamic temporal pooling for time series classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 9. 2021, pp. 8288–8296.

[207] Peng Wang, Yuanzhouhan Cao, Chunhua Shen, et al. "Temporal pyramid pooling-based convolutional neural network for action recognition". In: *IEEE Transactions on Circuits and Systems for Video Technology* 27.12 (2016), pp. 2613–2622.

[208] Maha Elbayad, Jiatao Gu, Edouard Grave, et al. “Depth-adaptive transformer”. In: *arXiv preprint arXiv:1910.10073* (2019).

[209] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. “Generating diverse high-fidelity images with vq-vae-2”. In: *Advances in neural information processing systems* 32 (2019).

[210] Mortimer Mishkin, Leslie G Ungerleider, and Kathleen A Macko. “Object vision and spatial vision: two cortical pathways”. In: *Trends in neurosciences* 6 (1983), pp. 414–417.

[211] Franziska Krebs, Andre Meixner, Isabel Patzer, et al. “The kit bimanual manipulation dataset”. In: *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2021, pp. 499–506.

[212] Friedrich Niemann, Christopher Reining, Fernando Moya Rueda, et al. “Lara: Creating a dataset for human activity recognition in logistics using semantic attributes”. In: *Sensors* 20.15 (2020), p. 4083.

[213] Jonathan Tompson, Ross Goroshin, Arjun Jain, et al. “Efficient object localization using convolutional networks”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 648–656.

[214] Richard Zhang, Phillip Isola, and Alexei A Efros. “Colorful image colorization”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*. Springer. 2016, pp. 649–666.

[215] Nahian Siddique, Sidike Paheding, Colin P Elkin, et al. “U-net and its variants for medical image segmentation: A review of theory and applications”. In: *Ieee Access* 9 (2021), pp. 82031–82057.

[216] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, et al. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.

[217] Michael Pfeiffer and Thomas Pfeil. “Deep learning with spiking neurons: Opportunities and challenges”. In: *Frontiers in neuroscience* 12 (2018), p. 774.

[218] Joshua Bassey, Lijun Qian, and Xianfang Li. “A survey of complex-valued neural networks”. In: *arXiv preprint arXiv:2101.12249* (2021).

[219] Eduard Kuriscak, Petr Marsalek, Julius Stroffek, et al. “Biological context of Hebb learning in artificial neural networks, a review”. In: *Neurocomputing* 152 (2015), pp. 27–35.

[220] Tobias Schlosser, Michael Friedrich, and Danny Kowerko. “Hexagonal image processing in the context of machine learning: Conception of a biologically inspired hexagonal deep learning framework”. In: *2019 18th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2019, pp. 1866–1873.

[221] Antonio Lieto, Mehul Bhatt, Alessandro Oltramari, et al. *The role of cognitive architectures in general artificial intelligence*. 2018.

[222] Ron Sun and Selmer Bringsjord. “Cognitive systems and cognitive architectures”. In: *Wiley Encyclopedia of Computer Science and Engineering* (2007).

[223] Andre Meixner, Franziska Krebs, Noémie Jaquier, et al. “An Evaluation of Action Segmentation Algorithms on Bimanual Manipulation Datasets”. In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 4912–4919.

[224] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. “Learning 3-d object orientation from images”. In: *2009 IEEE International conference on robotics and automation*. IEEE. 2009, pp. 794–800.

[225] Yi Zhou, Connelly Barnes, Jingwan Lu, et al. “On the continuity of rotation representations in neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 5745–5753.

[226] Afshin Rahimi, Timothy Baldwin, and Trevor Cohn. “Continuous representation of location for geolocation and lexical dialectology using mixture density networks”. In: *arXiv preprint arXiv:1708.04358* (2017).

[227] Gianni Brauwers and Flavius Frasincar. “A general survey on attention mechanisms in deep learning”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[228] Bryan Lim and Stefan Zohren. “Time-series forecasting with deep learning: a survey”. In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[229] Sam Bond-Taylor, Adam Leach, Yang Long, et al. “Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models”. In: *IEEE transactions on pattern analysis and machine intelligence* (2021).

[230] Axel Brando Guillaumes. “Mixture density networks for distribution and uncertainty estimation”. PhD thesis. Universitat Politècnica de Catalunya. Facultat d’Informàtica de Barcelona, 2017.

[231] Anurag Ajay, Yilun Du, Abhi Gupta, et al. “Is conditional generative modeling all you need for decision-making?” In: *arXiv preprint arXiv:2211.15657* (2022).

[232] Anthony Hu, Lloyd Russell, Hudson Yeo, et al. “Gaia-1: A generative world model for autonomous driving”. In: *arXiv preprint arXiv:2309.17080* (2023).

[233] Zihang Dai, Zhilin Yang, Yiming Yang, et al. “Transformer-xl: Attentive language models beyond a fixed-length context”. In: *arXiv preprint arXiv:1901.02860* (2019).

[234] Cecilia Heyes and Caroline Catmur. “What happened to mirror neurons?” In: *Perspectives on Psychological Science* 17.1 (2022), pp. 153–168.

[235] Martin Peniak, Davide Marocco, Jun Tani, et al. “Multiple time scales recurrent neural network for complex action acquisition”. In: *Proceedings of ICDL-Epirob* (2011).

[236] Kahneman Daniel. *Thinking, fast and slow*. 2017.

[237] H Freyja Ólafsdóttir, Daniel Bush, and Caswell Barry. “The role of hippocampal replay in memory and planning”. In: *Current Biology* 28.1 (2018), R37–R50.

[238] Kevin Black, Mitsuhiko Nakamoto, Pranav Atreya, et al. “Zero-shot robotic manipulation with pretrained image-editing diffusion models”. In: *arXiv preprint arXiv:2310.10639* (2023).

[239] Stephen Grossberg. “The resonant brain: How attentive conscious seeing regulates action sequences that interact with attentive cognitive learning, recognition, and prediction”. In: *Attention, Perception, & Psychophysics* 81 (2019), pp. 2237–2264.

[240] Alex Graves. “Adaptive computation time for recurrent neural networks”. In: *arXiv preprint arXiv:1603.08983* (2016).

[241] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. “Query by committee”. In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 287–294.

[242] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *international conference on machine learning*. PMLR. 2016, pp. 1050–1059.

[243] Chitwan Saharia, William Chan, Huiwen Chang, et al. “Palette: Image-to-image diffusion models”. In: *ACM SIGGRAPH 2022 conference proceedings*. 2022, pp. 1–10.

[244] John Patrick Aggleton. *The amygdala: a functional analysis*. Oxford University Press, 2000.

[245] Maël Donoso, Anne GE Collins, and Etienne Koechlin. “Foundations of human reasoning in the prefrontal cortex”. In: *Science* 344.6191 (2014), pp. 1481–1486.

[246] Masataro Asai and Alex Fukunaga. “Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 32. 1. 2018.

[247] Adam Santoro, David Raposo, David G Barrett, et al. “A simple neural network module for relational reasoning”. In: *Advances in neural information processing systems* 30 (2017).

[248] Ishika Singh, Valts Blukis, Arsalan Mousavian, et al. “Progprompt: Generating situated robot task plans using large language models”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 11523–11530.

[249] Danny Driess, Fei Xia, Mehdi SM Sajjadi, et al. “Palm-e: An embodied multimodal language model”. In: *arXiv preprint arXiv:2303.03378* (2023).

[250] Charles D Gilbert and Wu Li. “Top-down influences on visual processing”. In: *Nature Reviews Neuroscience* 14.5 (2013), pp. 350–363.

[251] Richard A Andersen. “Multimodal integration for the representation of space in the posterior parietal cortex”. In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 352.1360 (1997), pp. 1421–1428.

[252] Michael Zhu and Suyog Gupta. “To prune, or not to prune: exploring the efficacy of pruning for model compression”. In: *arXiv preprint arXiv:1710.01878* (2017).

[253] Guusje Collin and Martijn P Van Den Heuvel. “The ontogeny of the human connectome: development and dynamic changes of brain connectivity across the life span”. In: *The Neuroscientist* 19.6 (2013), pp. 616–628.