Doctoral Program in Statistics, Optimization and Applied Mathematics

Deep learning and complex systems: Structure, applications and limitations

Adrián Hernández Rodríguez

DOCTORAL DISSERTATION

Director

Prof. Dr. José María Amigó García

Universidad Miguel Hernández de Elche

2025

This Doctoral Thesis, entitled **"Deep learning and complex systems: Structure, applications and limitations"**, is presented under the modality of **conventional thesis**. As indicators of quality, we list below the four **publications** on which it is based:

1. **Multilayer adaptive networks in neuronal processing.** A. Hernández and J.M. Amigó (2018). *The European Physical Journal Special Topics* 227, 1039-1049.

   https://doi.org/10.1140/epjst/e2018-800037-y

2. **The need for more integration between machine learning and neuroscience.** A. Hernández and J.M. Amigó (2021). In: D. Volchenkov, editor, *Nonlinear Dynamics, Chaos, and Complexity*, pp 9-19. Springer, Singapore.

   https://doi.org/10.1007/978-981-15-9034-4_2

3. **Attention mechanisms and their applications to complex systems.** A. Hernández and J.M. Amigó (2021). *Entropy* 23, 283.

   https://doi.org/10.3390/e23030283

4. **Differentiable programming: Generalization, characterization and limitations of deep learning.** A. Hernández, G. Millerioux, and J.M. Amigó (2022). arXiv:2205.06898.

   https://doi.org/10.48550/arXiv.2205.06898

El Prof. Dr. D. José María Amigó García, director de la tesis doctoral titulada **“Deep learning and complex systems: Structure, applications and limitations”,**

INFORMA:

Que D. Adrián Hernández Rodríguez ha realizado bajo mi supervisión el trabajo titulado **“Deep learning and complex systems: Structure, applications and limitations”** conforme a los términos y condiciones definidos en su Plan de Investigación y de acuerdo al Código de Buenas Prácticas de la Universidad Miguel Hernández de Elche, cumpliendo los objetivos previstos de forma satisfactoria para su defensa pública como tesis doctoral.

Lo que firmo para los efectos oportunos, en Elche, febrero de 2025.

Director de la tesis

Prof. Dr. D. José María Amigó García

El Prof. Dr. D. Domingo Morales González, Coordinador del Programa de Doctorado en **Estadística, Optimización y Matemática Aplicada,**

INFORMA:

Que D. Adrián Hernández Rodríguez ha realizado bajo la supervisión de nuestro Programa de Doctorado el trabajo titulado **"Deep learning and complex systems: Structure, applications and limitations"** conforme a los términos y condiciones definidos en su Plan de Investigación y de acuerdo al Código de Buenas Prácticas de la Universidad Miguel Hernández de Elche, cumpliendo los objetivos previstos de forma satisfactoria para su defensa pública como tesis doctoral.

Lo que firmo para los efectos oportunos, en Elche, febrero de 2025.

Prof. Dr. D. Domingo Morales González

Coordinador del Programa de Doctorado en Estadística, Optimización y Matemática Aplicada

This doctoral thesis has been financially supported by:

- Agencia Estatal de Investigación, Spain, grant PID2019-108654GB-I00/AEI/10.13039/501100011033.
- Generalitat Valenciana, Spain, grant PROMETEO/2021/063.

Dedication

I dedicate this thesis to my family for their endless love, support and encouragement throughout my pursuit for research.

Acknowledgments

First of all, I am very grateful to J.M. Amigó. He introduced me to the wonderful world of research and has supported me in all my initiatives and especially in the development of this thesis. I appreciate the support of Gilles Millerioux from CRAN-CNRS for his time and dedication. And I would also like to thank Domingo Morales, Joaquín S. Soriano and Juan Aparicio from CIO-UMH. Their passion, dedication and altruism not only have a great impact on mathematics but also on the education and training of researchers.

Contents

Abbreviations

The following abbreviations are used in this thesis:

CNN	Convolutional Neural Network
FNN	Feed-forward Neural Network
GNN	Graph Neural Network
GPT-X	Generative Pre-Trained Transformer-X
GPU	Graphics Processing Units
LSTM	Long Short-Term Memory
MNN	Multilayer Neural Network
RNN	Recurrent Neural Network
TPU	Tensor Processing Unit

1 | Summary

1.1 Spanish

El desarrollo de los modelos de deep learning y el aumento de las capacidades computacionales han permitido mejorar notablemente el rendimiento de tareas específicas y generales antes atribuidas solo a la inteligencia humana.

Estos modelos, siendo el más básico una red neuronal, son ejemplos específicos de programas diferenciables: Un programa diferenciable es una secuencia de transformaciones de tensores cuyos parámetros son diferenciables y se ajustan para minimizar el error de la tarea a realizar.

La programación diferenciable, como generalización del deep learning, es un marco lo suficientemente expresivo para aproximar cualquier función y lo suficientemente flexible para incorporar la estructura del problema o tarea a resolver. Para resolver un problema concreto es entonces necesario restringir la estructura del programa diferenciable según las características del problema (dependencia temporal, invarianzas, etc.)

En paralelo, el campo de los sistemas complejos (es decir, sistemas que contienen una gran cantidad de variables que interactúan entre sí de formas no triviales) ha producido recientemente muchos avances que pueden contribuir al deep learning, sirviendo de fuente de inspiración de nuevos modelos y estructuras y siendo los candidatos ideales para aplicar las últimas técnicas de la programación diferenciable.

El objetivo de esta Tesis es poder relacionar y avanzar estos dos campos a través de sus sinergias, técnicas y estructuras comunes.

Primero, hemos analizado los modelos de redes complejas usados para describir

el procesamiento de la información y la relación entre las neuronas en el cerebro. Hemos identificado que usar una matriz con las conexiones estructurales entre neuronas ha sido un enfoque revolucionario, pero tiene sus limitaciones.

En el cerebro, o cualquier sistema complejo de procesamiento de información, las relaciones entre sus unidades básicas de información (neuronas), no son solo conexiones estructurales que no dependen del estado de cada neurona. En un sistema complejo de procesamiento de información, sus unidades están relacionadas de diversas formas y hay una continua dependencia entre sus relaciones y el estado de las unidades.

Hemos propuesto las redes adaptativas multicapa, en las que diferentes capas paralelas interactúan de manera adaptativa con los nodos, como el marco apropiado para explicar el procesamiento de la información neuronal. Para ello extendemos el formalismo matricial que relaciona cada par de nodos al formalismo tensorial de rango cuatro, que relaciona un nodo de una capa con uno de cualquier otra capa, teniendo en cuenta también la interacción entre la dinámica de cada nodo y la sus conexiones con otros nodos.

Esta primera contribución de la Tesis nos demuestra cuán rico y variado es el comportamiento de los sistemas complejos y por qué es importante usar los sistemas complejos como fuente de inspiración y aplicación para el deep learning.

Siguiendo con la relación entre deep learning y sistemas complejos, la forma más útil y completa de estudiar los diferentes modelos de deep learning es a través de la programación diferenciable. Definimos formalmente la programación diferenciable a partir de un grafo dirigido acíclico que se implementa en tiempo de ejecución y cuyos parámetros se ajustan para minimizar el error de la tarea.

Dado que la programación diferenciable permite implementar cualquier secuencia de transformaciones diferenciables de tensores, es necesario poder restringir la estructura del programa para adaptarlo a la tarea a realizar. Para ello proponemos varias características como relaciones entre tensores, invarianzas o simetrías, combinación de módulos... y las usamos para explicar los modelos más recientes de deep learning.

Aplicamos estos conceptos a un problema de clasificación de publicaciones científicas relacionadas mediante un grafo usando diferentes modelos de deep learning como redes neuronales, graph neural networks y auto-atención. La conclusión es que, cuando los datos de entrenamiento son limitados, es mejor usar

modelos que incorporen la estructura del problema. Discutimos también una gran limitación inherente de la programación diferenciable, que minimiza el error de predicción de los datos de entrenamiento pero es incapaz de generar nueva información.

Para profundizar en la relación entre deep learning y sistemas complejos, analizamos uno de los modelos más relevantes y recientes de programas diferenciables, los mecanismos de atención, y sus aplicaciones a los sistemas complejos. Para ello describimos los principales aspectos, ventajas y modos de operación de la atención diferenciable. Presentamos las principales técnicas de atención como los modelos seq2seq, los Transformers y las redes de memoria, analizando por qué representan un avance en el deep learning. Finalmente, ilustramos algunos usos interesantes de estas técnicas para modelar sistemas complejos demostrando que los mecanismos de atención permiten modelar ciertas características típicas de los sistemas complejos como la integración de diferentes partes, el razonamiento secuencial o las dependencias temporales de rango largo.

En este punto, en el que hemos definido nuevas estructuras de sistemas y redes complejas y hemos analizado la programación diferenciable, sus características, modelos específicos y aplicación a los sistemas complejos, la pregunta que nos surge es: ¿qué limitaciones inherentes tienen estos modelos y cómo se pueden superar?

Los modelos diferenciables ajustan los parámetros del programa para minimizar el error de predicción de los datos de entrenamiento. Este modo inherente de funcionamiento los hace ideales para modelar estadísticamente los datos de entrenamiento, pero impide que se genere nueva información y conocimiento no visto anteriormente.

Para analizar si se pueden superar estas limitaciones, definimos el marco de las estrategias de aprendizaje y evaluamos si los LLM (grandes modelos de lenguaje), la técnica más reciente y disruptiva basada en los mecanismos de atención, pueden implementar dichas estrategias y generar nuevo conocimiento.

Estas estrategias de aprendizaje consisten en modelos de deep learning que interaccionan con nuevos datos y con el entorno para generar nuevo conocimiento. Definimos una estrategia concreta que consiste en una serie de actividades secuenciales para predecir los precios del petróleo, obteniendo los datos y definiendo, mejorando y complementando un modelo de predicción.

Concluimos que los LLM pueden implementar correctamente actividades típicas con presencia en los datos de entrenamiento, pero no tienen una capacidad general para definir estrategias de aprendizaje de una manera lógica y coherente. De hecho, el rendimiento de los LLM baja significativamente cuando intentan implementar actividades más creativas que no coinciden con actividades típicas de los datos de entrenamiento.

Finalmente, discutimos el compromiso generador-verificador de los LLMs y sugerimos que su solución parcial pasa por la preparación cuidadosa de los datos de entrenamiento, el uso de verificadores externos y la intervención humana. Estas condiciones alejan a los LLMs de la inteligencia artificial general.

1.2 English

The development of deep learning models and the increase in computational capabilities have made it possible to significantly improve the performance of specific and general tasks previously attributed only to human intelligence.

These models, the most basic being a neural network, are specific examples of differentiable programs: A differentiable program is a sequence of tensor transformations whose parameters are differentiable and are adjusted to minimize the error of the performed task.

Differentiable programming, as a generalization of deep learning, is a framework expressive enough to approximate any function and flexible enough to incorporate the structure of the problem or task to be solved. To solve a specific problem it is then necessary to restrict the structure of the differentiable program according to the characteristics of the problem (time dependence, invariances, symmetries, etc.)

In parallel, the field of complex systems (i.e., systems that contain a large number of variables that interact with each other in non-trivial ways) has recently produced many advances that can contribute to deep learning, both as a source of inspiration for new structures and as ideal candidates to apply the latest techniques of differentiable programming.

The objective of this Thesis is to relate and advance these two fields through their synergies, techniques and common structures.

First, we have analyzed the complex network models used to describe infor-

mation processing and the relationship between neurons in the brain. We have identified that using a matrix with the structural connections between neurons has been a revolutionary approach, but it has its limitations.

In the brain, or any complex information processing system, the relationships between its basic information units (neurons) are not just structural connections that do not depend on the state of each neuron. In a complex information processing system, its units or nodes are related in various ways and there is a continuous dependence between their relationships and the state of the units.

We have proposed multilayer adaptive networks, in which different parallel layers adaptively interact with nodes, as the appropriate framework to explain neural information processing. To do this, we extend the matrix formalism that relates each pair of nodes to the rank four tensor formalism, which relates a node from one layer to another node from any other layer, also taking into account the interaction between the dynamics of each node and its connections with other nodes.

This first contribution of the Thesis shows us how rich and varied the behavior of complex systems is and why it is important to use complex systems as a source of inspiration and application for deep learning.

Following with the relationship between deep learning and complex systems, the most useful and complete way to study the different deep learning models is through differentiable programming. We formally define differentiable programming based on an acyclic directed graph that is implemented at runtime and whose parameters are adjusted to minimize the task error.

Since differentiable programming implements any sequence of differentiable tensor transformations, it is necessary to be able to restrict the structure of the program to adapt it to the performed task. To do this, we propose several characteristics such as relationships between tensors, invariances or symmetries, combination of modules... and we use them to explain the most recent deep learning models.

We apply these concepts to a problem of classifying scientific publications using different deep learning models such as neural networks, graph neural networks and self-attention. The bottom line is that, when training data is limited, it is better to use models that incorporate the structure of the problem. We also discuss a major inherent limitation of differentiable programming, namely, that

it minimizes the prediction error of the training data but is unable to generate new information.

To delve deeper into the relationship between deep learning and complex systems, we analyze attention mechanisms, the most relevant and recent models of differentiable programming, and their applications to complex systems. To this end, we describe the main aspects, advantages and modes of operation of differentiable attention. We present the main attention techniques such as seq2seq models, Transformers and memory networks, analyzing why they represent an advance in deep learning. Finally, we illustrate some interesting uses of these techniques to model complex systems by demonstrating that attention mechanisms allow modeling certain typical characteristics of complex systems such as the integration of different parts, sequential reasoning or long-range temporal dependencies.

At this point, where we have defined new structures of complex systems and networks and have analyzed differentiable programming, its characteristics, specific models and application to complex systems, the question that arises is what inherent limitations do these models have and how can they be overcome?

Differentiable models adjust program parameters to minimize the prediction error of the training data. This inherent mode of operation makes them ideal for statistically modeling training data, but prevents the generation of new information and knowledge not previously seen.

To analyze whether these limitations can be solved, we define the framework of learning strategies and evaluate whether LLMs (Large Language Models), the most recent and disruptive techniques based on attention mechanisms, can implement such strategies and generate new knowledge.

These learning strategies consist of deep learning models that interact with new data and the environment to generate new knowledge. We define a concrete strategy that consists of a series of sequential activities to predict oil prices. The activities are to obtain the data, define the model and improve and complement the model.

We conclude that LLMs can correctly implement typical activities with presence in the training data, but do not have a general ability to define learning strategies in a logical and coherent way. In fact, LLM performance drops significantly when you try to implement more creative activities that do not match typical activities in the training data.

Finally, we discuss the generator-verifier trade-off of LLMs and suggest that its partial solution involves the careful preparation of training data, the use of external verifiers and human intervention. These conditions move LLMs away from general artificial intelligence.

2 | Introduction

2.1 Motivation and description of the problem

In the last years, the combination of deep learning models and the computational capabilities of Graphics Processing Units (GPUs) [123] has brought a breakthrough to the field of machine learning and artificial intelligence, improving the performance of several tasks such as image recognition, machine translation, language modelling, time series prediction, language modelling, etc. [69, 105, 100, 45, 92, 20].

These models, the most basic being neural networks, are specific examples of differentiable programs. A differentiable program is a sequence of tensor transformations whose parameters are differentiable and that is trained end-to-end to minimize the task error.

Differentiable programming, as a generalization of deep learning, is a framework expressive enough to contain the space of all functions, and flexible enough to incorporate the priors and the structure of the problem or task when we define a new program. To solve a concrete problem, it is necessary to restrict this hypothesis function space taking into account the structure and priors of the problem. This is what we have seen in recent years with the development of specific models such as multilayer neural networks, RNNs, CNNs, etc.

In parallel, the field of complex systems (i.e., systems that contain a very large number of variables interacting with each other in non-trivial ways [50, 68, 97]) has recently produced many advances that can contribute to differentiable and deep learning models through a double path:

- New structures and models used in complex systems and networks can provide inspiration for differentiable programming and deep learning.

- Complex systems are ideal candidates for applying the latest deep learning models due to their intrinsic difficulty in modeling behaviour.

Given the flexibility of differentiable programming and the varied and rich behavior of complex systems, the following challenges to relate the two fields arise:

i Which architectures and tensor transformations are most appropriate considering the characteristics of the problem? For example, is it useful to use an RNN to model sequences with long temporal dependencies?

ii Can more general and interesting architectures for processing information be found in complex systems and neuroscience?

iii For systems with interdependence between parts such as complex systems, which architectures and deep learning models allow us to reflect their great diversity of behaviors?

iv By its own definition as data-driven programs, does differentiable programming have inherent limitations in generating new knowledge and advancing artificial intelligence?

2.2 Objectives

The objective of this thesis is to try to address these challenges and use complex systems as a source of inspiration and application for deep learning. The specific objectives are:

1. To propose new structures and models in complex systems and networks, such as multilayer adaptive networks, in which different parallel layers interact, as the appropriate framework to explain neuronal and information processing.
2. To define and motivate differentiable programming as a generalization of deep learning and evaluate the application of different types of differentiable programs to a specific complex system.
3. To describe the key aspects and techniques of attention mechanisms, the most innovative models of differentiable programming, and illustrate some important applications of these techniques in the modeling of complex systems.
4. To analyze the inherent limitations of differentiable programs and evaluate

whether Large Language Models, the most disruptive models based on attention mechanisms, can overcome these limitations and generate learning strategies for complex systems.

2.3 Materials and methods

The materials and methods used in the development of the research that is compiled in this thesis are the usual ones in applied mathematics, namely, related literature, computer code, specialized libraries, simulations and public data sets.

The research has been based on a methodology that combines theoretical research and computational simulations. Specifically, the methodology has followed the phases below:

i Reading related literature (books, chapters, articles, etc.)

ii Definition of mathematical and computational models.

iii Code development and simulations.

iv Publication and dissemination of the results.

v Contact with other researchers to exchange and validate ideas.

2.4 Structure of the thesis and main results

This doctoral thesis is structured in six chapters. **Chapter 1** provides a summary of the thesis. **Chapter 2** gives a brief introduction to the subject of this thesis, its motivation and interest, the objectives that have been posed, the materials and methods used in the development of the research, and the structure of the thesis.

Chapter 3 highlights the limitations of the connectome and neurotransmission networks to understand neuronal processing, as well as the need of using a new framework. Then, it describes the characteristics that a complex network framework should have in order to provide a complete description of the different neuronal information dynamics, scales and interactions that occur in the brain. Multilayer adaptive networks, in which different synaptic and chemical layers interact, are the appropriate framework to explain neuronal processing and the emergence of interesting computational capabilities. Finally, it proposes a simplified multilayer adaptive network model that accounts for these extra-layers of interaction and analyses the emergence of interesting computational capabilities.

Recent deep learning models are inspired by complex networks and neuroscience and are specific examples of differentiable programming. **Chapter 4** defines and motivates differentiable programming, as well as specifies some program characteristics to incorporate the structure of a problem into a differentiable program. It analyzes different types of differentiable programs, from more general to more specific. As an application, it evaluates the structure of a classification task in a complex network using several differentiable programs and deep learning models. Finally, it discusses some inherent limitations of deep learning and differentiable programs, which are key challenges in advancing artificial intelligence. It concludes that, for problems with a defined structure and a limited amount of training data, it is better to use specific programs or models adapted to that previous structure than general ones.

Chapter 5 elaborates on the need and importance of attention mechanisms, the most recent and innovative models of differentiable programming. It presents the key aspects, the advantages and the main modes of operation of attention. Then it describes some important attention techniques such as attention in seq2seq models, as well as self-attention and memory networks, emphasizing why they represent significant progress in machine learning. Finally, it illustrates some interesting applications of theses techniques for modeling complex systems.

Although differentiable programming has allowed great advances in recent years, this approach has limitations in generating new knowledge not seen before. Inspired by cognitive science, **Chapter 6** defines learning strategies as a framework to overcome these limitations and evaluates whether Large Language Models (LLMs) can generate such strategies. These strategies define machine learning models and data, and interact with the environment to generate new knowledge. It concludes that LLMs can correctly implement the activities with significance in the training data but do not have a general planning capacity based on principles and logical rules. Furthermore, the performance of LLMs drops greatly when the activities are more creative and do not coincide with the typical activities present in code repositories. Finally, it discusses the generator-verifier trade-off in LLMs and suggests that its solution lies in the careful preparation of training data, the use of external verifiers and human intervention.

Chapter 7, the last chapter, presents the main conclusions of the thesis and future lines of research.

3 Multilayer adaptive networks in neuronal processing

3.1 Introduction

In the last years considerable efforts are being dedicated to provide insights into neural circuits in what has been called the connectome [102, 127, 58, 37]. The connectome is a map of neural connections in the brain and may be thought of as its "wiring diagram".

The connectome can be structural, if it describes anatomical connections between parts of the brain or neurons, or functional, if it describes statistical associations between activities in those parts. If we think of the structural connectome as a road map, then the functional connectome corresponds to the vehicles that travel the roads.

The knowledge of the neural elements and their neural connections can help understand how the cognitive function emerges from the neuronal structure and dynamics. This wiring diagram maps all the neural connections in the brain and, at the cellular level, it provides a map of the neurons and synapses in the brain.

The ideal framework to study and model the dynamics, topology and properties of this type of synaptic connections is that of complex networks and dynamical systems. Different approaches, from artificial neural networks to biophysical models that take into account the biological reality (conductances, response times, properties of synapses and dendrites, ...) have been used to describe the dynamics of neuronal connections and information processing in the brain; see [43] for a comprehensive description of neuronal dynamic models.

However, as some neuroscientists have pointed out [13, 12, 19, 75], knowl-

edge and modelling of the connectome, either structural or functional, are not enough to understand how the brain processes the information, although they contribute in a prominent way. There are other biological mechanisms, such as neuromodulation, that reconfigure the connectome.

In neuromodulation [21], a given neuron uses one or more chemicals to regulate diverse populations of neurons, in contrast to classical synaptic transmission, where one presynaptic neuron directly influences a single postsynaptic neuron. Neuromodulators operate on several time scales, and modulate and configure the connectome so as to determine the processing of information. The connectome provides a minimal structure and, on top of it, neuromodulators configure and specify the functional circuits that give rise to behaviour.

Thus, neuromodulators add new computational and processing capabilities to traditional synaptic transmission. First, they add extra-layers of biochemicals that regulate or tune neuronal processing. Second, the temporal scales of these extra-layers are different from classical ones. Third, these extra-layers and classical synaptic networks interact in a complicated way.

We propose that the appropriate framework for modelling neuronal processing is that of multilayer adaptive networks. Multilayer because in the brain different synaptic and neuromodulatory networks interact to produce behaviour, and adaptive because the topology of these networks changes according to the dynamics.

The first goal of this chapter is to highlight the limitations of the connectome and neurotransmission networks to understand neuronal processing, as well as to point out the need of using a new framework. We review examples in which the same structural network can have different configurations and behaviours.

The second goal is to define the characteristics that a complex network framework must have in order to provide a complete description of the different neuronal information dynamics, scales and interactions that occur in the brain. Multilayer adaptive networks, in which different synaptic and chemical layers interact, are the appropriate framework to explain neuronal processing and the emergence of interesting computational capabilities.

3.2 Beyond neurotransmission networks

The connectome is a wiring diagram mapping all the neural connections in the brain. At the cellular level, it provides a map of the neurons and synapses within

a part or all of the brain of an organism. The structural connectome provides the basis on which functional activity is implemented and therefore shapes the functional connectivity.

On the way to unveil the connectome of the human brain, one of the ultimate goals in neuroscience, some milestones have been achieved, e.g., the complete connectome of the nematode C. elegans [120, 111], which comprises 302 neurons.

In recent years, significant advances have been also made in the study of the connectome via network science and graph theory [1, 3, 2, 11, 14]. At the cellular level, the nodes of the network are the neurons, and the edges correspond to the synapses between those neurons. Therefore, graph theory is the ideal framework to study the topology and dynamics of brain networks. The combination of new tools to map and record neuronal patterns and the computational techniques of network science has provided a new setting for the study of the brain dynamics.

Many studies have been conducted to analyze the topology and dynamics of the neuronal connectome. In [85] the authors studied the growth rules of the neuronal system of C.elegans. They found that the network growth undergoes a transition from an accelerated to a constant increase in the number of synaptic connections as a function of the number of neurons. The transition between different growth regimes may be explained by a dynamic economical model incorporating a trade-off between topology and cost. In [109] graph theory is used to investigate the neuronal connectome of C.elegans. The authors identified a small number of highly connected neurons as a rich club interconnected with high efficiency and high connection distance.

Clearly, connectome modelling and analysis play a salient role when it comes to understand neurotransmission (fast synaptic transmission) networks. However, neurons use other forms of communication as neuromodulation that, instead of conveying excitation or inhibition, change neuronal and synaptic properties.

As described in [83], neuromodulators are released in modes other than fast synaptic transmission and modify the neuronal circuit outputs. They are the main factors in shaping behaviour by changing neuronal excitability and synaptic dynamics and strength. The processes that are subject to modulation include changes in probability of neurotransmitter release, changes in transmitter receptors, modification of synaptic strength, adding or subtracting ionic currents, and changes in voltage and time dependence of channel gating. In doing so, they reconfigure the connectome and provide adaptability of the brain functions.

Although traditionally neurochemical messengers have been classified as small-molecule neurotransmitters, biogenic amines and neuropeptides, for the purposes of this chapter it is more appropriate to differentiate between neurotransmitter (fast synaptic transmission) and modulator functions as in [19].

In neuronal circuits, connectivity alone does not provide enough information to predict circuits outputs [12, 13]. Neuronal processing and behaviour are sensitive to intrinsic channels and electrical properties that vary within and between cell types. Fast synaptic transmission and biochemical processes interact to generate complex dynamics in neurons and circuits.

Studies of neuronal circuits in invertebrate and vertebrate animals [12] have revealed the ability of neuromodulators to reconfigure information processing. They change the composition of neuronal circuits and permit a circuit with a fixed number of neurons to produce many different patterns of activity.

Then, the greatest challenge that we face to understand the brain is to have new models that allow us to explain how the interaction of different layers of neurotransmission, neuromodulators and genetic changes gives rise to information processing. Moreover, without taking into account these different interactions, it is impossible to explain many computational functions observed in the brain.

3.3 Multilayer adaptive networks in neuronal processing

Network science has grown over the last decades to become a relevant conceptual framework for the analysis of many real systems. A tremendous progress has been made in the application of network models in neuroscience. Modelling brain networks as graphs of nodes connected by edges has provided major advances in understanding brain dynamics. From the dynamic analysis of groups of neurons to the topological characterization of large-scale human brain networks, network theory has become a fundamental tool in neuroscience; see [41] and [40] for a comprehensive description of network neuroscience.

Traditionally, the study of dynamical networks has covered either dynamics on networks or dynamics of networks. In the first approach, nodes are dynamical systems coupled through static links. This case includes dynamical systems describing the dynamics in a phase space with no topological changes between the nodes. In the second approach, network topology evolves dynamically in time.

This is the case of traditional complex networks, where the focus has been put on analyzing the statistical properties that arise from exogenous topological transformations.

In recent years, there has been a growing interest in adaptive networks, i.e., networks whose links change adaptively with the states of the nodes in an interplay between node states and network topology [97, 77, 121, 5, 6]. Two facts make adaptive networks specially convenient for the study of natural and social systems. First, dynamical processes on a network are sensitive to the network topology, which influences the states of nodes. Second, the states of the nodes feed back to the network topology creating a dynamical feedback loop between topology and states of the network. In a neuronal network, the firing rate of a neuron depends on the synaptic connections (topology) and, in turn, the evolution of the synaptic connections and weights depends on the neuronal activity. Furthermore, both processes –neuronal activity and synaptic reconfiguration– take place at different timescales.

A typical adaptive, directed network with a fixed set of nodes is composed of the following elements:

1. A set of N nodes $V = \{v_1, v_2, ..., v_N\}$. Abusing notation, node v_i will be denoted by i.
2. Each node $i \in \{1, ..., N\}$ has a state $s_i(t)$.
3. The set of evolving links is encoded in a time-dependent, weighted adjacency matrix $A(t)$ with entries $a_{ij}(t)$. In our case, self-links are excluded, so $a_{ii}(t) = 0$.
4. Each link weight $a_{ij}(t)$ represents the relationship from node i to node $j \neq i$ and is a function of $s_i(t)$ and $s_j(t)$.
5. Node states $s_j(t)$ are a function of the sum of incoming weighted nodes, that is, $\sum_{i=1}^{N} a_{ij}(t)s_i(t)$.

These systems change their states and topologies simultaneously according to their interrelated dynamical rules. In a link removal $a_{ij}(t) \neq 0$ becomes $a_{ij}(t) = 0$ while a new link is established when $a_{ij}(t) = 0$ becomes $a_{ij}(t) \neq 0$.

In addition, until recently the majority of studies have focused on single-layer networks, usually with a single type of node connected via a single type of link. But in most biological systems multiple entities interact with each other in

complicated patterns that include multiple layers of connectivity. Consequently, it became necessary to generalize network theory by developing a new setting to study multilayer systems in a comprehensive fashion [64, 30, 29, 80]. Then, multilayer networks are the suitable framework to study different networks that interact to produce complex activities.

As we have seen, single-layer networks are represented using adjacency matrices which, in the case considered, represent directed and weighted relationships between the nodes of a network. Instead, multilayer networks represent multiple dimensions of connectivity that, in the case of neurons, can stand for different types of communication channels (neurotransmitters and neuromodulators). A typical multilayer network has the following ingredients:

1. A number N of nodes (denoted by Latin letters i, j, ...) and a number L of layers (denoted by Greek letters α, β, ...).
2. Node $i \in \{1, 2, ..., N\}$ in layer $\alpha \in \{1, 2, ..., L\}$ has a state $s_{i\alpha}(t)$.
3. A 4th-order, time-dependent adjacency tensor $\boldsymbol{M}(t)$ with components $m_{i\alpha}^{j\beta}(t)$ which are the weights of the link from any node i in layer α to any node j in layer β in the network.

In multilayer networks (see Fig. 3.1) nodes can be connected by different types of interactions: Intra-layer links connecting nodes within the same layer, inter-layer links between the same nodes in different layers, as well as inter-layer links between different nodes in different layers. Multiplex networks are a special class of multilayer networks such that $m_{i\alpha}^{j\beta} = 0$ if $\alpha \neq \beta$ and $i \neq j$, i.e., different layers are not interconnected except from each node to itself.

An interesting example of a multilayer network can be defined by extending the connectome with synaptic and neuromodulatory layers representing alternative modes of interaction between neurons, along with the corresponding communication links between layers (see Fig. 3.2).

The combination of multilayer and adaptive networks describes networks with different interactions between their nodes, together with a dynamical feedback between network topology and node states.

As we pointed out before, the connectome and single synaptic networks are not enough to understand neuronal information processing. We summarize next the characteristics that make multilayer adaptive networks the right framework:

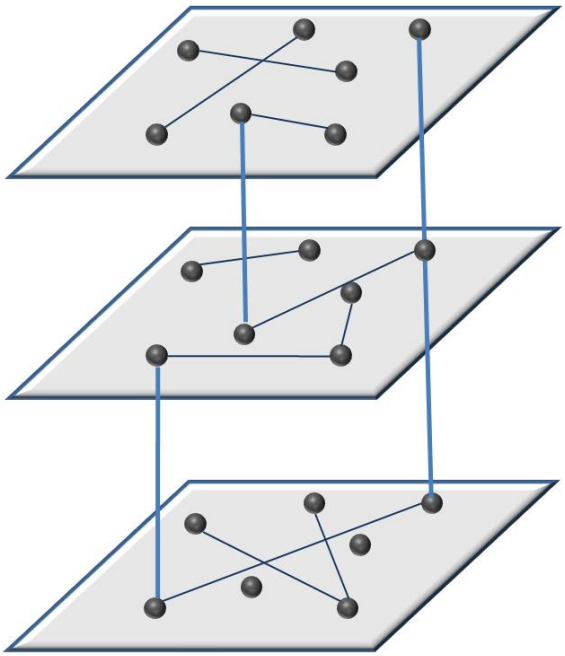

Figure 3.1: A multilayer network with intra-layer edges and inter-layer edges that connect entities with their replicas in other layers.

1. In addition to fast synaptic transmission, neurons use other forms of communication such as neuromodulation that change neuronal and synaptic properties.
2. The extra layers of a multilayer approach regulate or tune neuronal processing and operate on temporal and spatial scales different from the fast synaptic ones.
3. The neuromodulatory layers interact with the fast synaptic transmission layer in a complicated way, changing neuronal excitability and synapses dynamics and strength.
4. The fast neurotransmission layer topology changes according to neuronal and neuromodulatory activity.

3.4 Modeling neurotransmission and neuromodulation

The connectome represents a network of potential neurons and connections (synapses). Function, context and neuromodulatory networks shape and reconfigure the con-

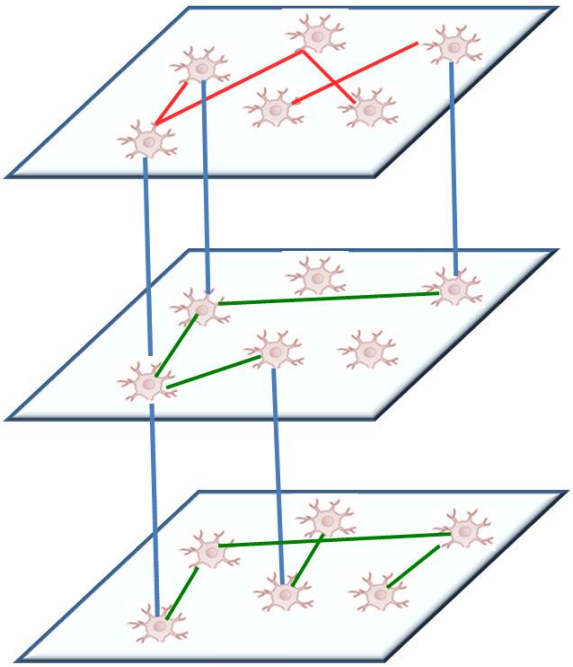

Figure 3.2: A multilayer network with one synaptic layer (red links) and two neuromodulatory layers (green links) representing alternative modes of interaction between neurons, along with the corresponding communication links between layers.

nectome network allowing different paths of information flow.

Neurotransmission is a wiring transmission that targets designated synapses and produces localized responses. On the other hand, neuromodulation is a volume transmission that diffuses through large areas of the neural tissue and affects multiple neurons. As a result, neuromodulatory networks are more complex than neurotransmission ones because neuromodulators act non-locally.

In [65] the authors hold that further understanding of brain function and dysfunction will require an integrated framework that links brain connectivity with brain dynamics. This expanded description is called "dynome" and includes the functional connectome but expands the notion to the mechanisms involved in producing and processing brain signals. Detailed biophysical models of neural activity embedded in an anatomical network may be essential to examine the effects of biological dynamics on functional connectivity.

As described in [28] for the crustacean stomatogastric nervous system, dif-

ferent regulatory mechanisms (synaptic and intrinsic neuronal properties, neuromodulation and gene expression regulation) influence each other to produce stable neuronal circuits.

The C.elegans connectome is considered in [16] as a multiplex network, with each node representing a neuron and with different layers of connection (synaptic and neuromodulatory). The authors found highly significant multilink motifs of interaction between the extrasynaptic and synaptic connectomes, detecting locations in the network where the monoamines and neuropeptides modulate synaptic activity.

A simplified multilayer adaptive network model that accounts for these extralayers of interaction can be represented as follows:

1. A number N of neurons and a number L of layers. Each neuron is replicated in the rest of the layers, but with a different associated dynamics. An adjacency tensor $M(t)$ with components $m_{i\alpha}^{j\beta}(t)$ encodes the relationships from any neuron i in layer α to any neuron j in layer β.
2. The first layer is the neurotransmission layer. Each neuron in this layer has a state $s_{i1}(t)$. An evolving set of synapse weights is represented by $m_{i1}^{j1}(t)$.
3. The remaining layers are neuromodulatory layers. Each neuron in one of this layer has a state $s_{i\alpha}(t)$ with $\alpha \in \{2, ..., L\}$. An evolving set of neuromodulatory link weights is represented by $m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$.
4. The states in the first layer $s_{j1}(t)$ are a function of both the sum of incoming synapses $\sum_{i=1}^{N} m_{i1}^{j1}(t)s_{i1}(t)$ and the sum of incoming interactions from neuromodulatory nodes $\sum_{\alpha=2}^{L} m_{j\alpha}^{j1}(t)s_{j\alpha}(t)$.
5. The synapse weights in the first layer $m_{i1}^{j1}(t)$ are a function of both $s_{i1}(t)$ and neuromodulatory link weights in the remaining layers $\epsilon_\alpha m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, where ϵ_α is a coupling parameter between neuromodulatory and neurotransmission links.
6. The states in the remaining layers $s_{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, are a function of both the sum of incoming neuromodulatory links $\sum_{i=1}^{N} m_{i\alpha}^{j\alpha}(t)s_{i\alpha}(t)$ and $m_{j1}^{j\alpha}(t)s_{j1}(t)$.
7. The neuromodulatory link weights $m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, are a function of $s_{i\alpha}(t)$.

In this directed network, the first layer is a typical neurotransmission layer with adaptive node states and synapses, where the link weights $m_{i1}^{j1}(t)$ represent

classical fast-synaptic connections. The remaining layers contain adaptive neuromodulatory states and links, where $m_{i\alpha}^{j\alpha}(t)$ are neuromodulatory link weights within layers. The interaction between layers is of three types: Node states in the first layer are influenced by the states of the same node in the neuromodulatory layers via $m_{j\alpha}^{j1}$. Second, synapse weights in the first layer are influenced by the same link weights in the neuromodulatory layers via ϵ_α. Third, node states in the neuromodulatory layers are influenced by the state of the same node in the first layer via $m_{j1}^{j\alpha}$. Adjacency weights are different from 0, $m_{i\alpha}^{j\beta} \neq 0$, when $\alpha = \beta$ or when $\alpha \neq \beta$, $i = j$, and $\alpha = 1$ or $\beta = 1$.

Within the above setting, one can formulate detailed biological models (e.g., Hodgkin-Huxley [54]) as well as more abstract models (e.g., McCulloch-Pitts [79]). For example, in an abstract model the output of a neuron in the neurotransmission layer is given by the weighted sum of its inputs. The neuromodulatory layers determine how synaptic weights are updated. In such a model, neuromodulation can change how a circuit function is achieved. The variables of a model are fixed by the level of biological detail (concentration, membrane potential, state of the neuron, etc.).

Regarding the dynamics, the evolution of a network is mostly described by differential equations (Hodgkin-Huxley, Fitzhugh-Nagumo, ...) or by difference equations (logistic maps, ...) depending on the kind and purpose of the model; see, e.g., [70, 78].

As a final remark, by examining basic aspects of the multilayer adaptive models being considered, several insights can be extracted. If neuromodulatory layers do not depend on neurotransmission activity, then we have extrinsic neuromodulation. In intrinsic neuromodulation, neuromodulatory layers are not isolated from neurotransmission activity. It is also interesting to compare the temporal scale of the neurotransmission layer with that of the neuromodulatory layers. In most cases, neuromodulation is a slow process compared to neurotransmission [38].

In the next section we consider the importance of multilayer adaptive models to analyze the computational capabilities of neuromodulators and to provide a complete description of neuronal processing beyond the connectome.

3.5 Computational capabilities of the multilayer connectome

As pointed out above, it is necessary to take into account the different layers of neuronal communication to complete the connectome, both functional and structural. To this end, multilayer adaptive networks build the appropriate framework to analyse how these layers interact to produce neuronal processing. Next we focus on the computational capabilities added by these interactions at the circuit level. The computational capabilities listed below can be materialized with two interacting layers, one for neurotransmission and the other one for neuromodulation. Other works have focused on aspects of neuromodulators more related to behaviour [126, 36, 35]. Among those capabilities, we underline the following:

1. The different time scales between neurotransmission and neuromodulatory layers give rise to interesting phenomena. For example, neuromodulatory activity in a slower process may fine-tune some properties of the neurotransmission layer such as neuronal excitability and synapse dynamics.
2. Because neuromodulation occurs in a diffusive manner, the same neuromodulatory layer can tune different and isolated neuronal circuits (neurotransmission layers).
3. Neuromodulatory layers can improve the functioning of neuronal circuits. For example, in [56] the authors found that the neuromodulators trigger distinct changes in representations (through the synaptic weights) that improve the networks classification performance. In [51] the authors demonstrate that neuromodulation of a single target neuron may dramatically alter the performance of an entire network or have almost no effect depending on the state of the network.
4. Neuromodulatory layers can ensure a reliable neuronal circuit function despite changes in the parameters of the neurotransmission layer. These extra layers regulate the correlation between parameters, providing robustness to their variation [86, 76].
5. Neuromodulatory layers can reconfigure the neuronal circuit and even change the function performed. This provides a very useful circuit adaptability to the environment.
6. Neuromodulators possibly regulate the storage of new information in neuronal networks. Neuromodulatory layers can maintain memory and activity

after reconfiguration of neuronal networks [93, 25].

These facts show once more the limitations of the connectome to predict the output of the circuit and the need to extend the connectome with additional layers of neuromodulation that interact dynamically to reconfigure the connectome at every moment.

This new approach will call for network theory and dynamical systems, along with large computational resources, but it will be essential to understand the complexity of the brain. Moreover, the study of neuronal circuits using multilayer adaptive networks will provide clues and evidence of why the processing of information in the brain is more complex and varied than the one observed in artificial neural networks.

3.6 Conclusions and future work

Important as is the study of structural and functional connectome networks, it is even more important to consider the additional neuromodulatory layers that condition and reconfigure the connectome. In line with the complex cognitive needs of the brain, there is no universal neural coding-decoding scheme but rather different layers of processes that add capabilities to information processing.

As we have seen in this chapter, multilayer adaptive networks are an appropriate framework to study neuronal processing and to take into account all the communication processes that occur beyond neurotransmission (for example neuromodulation). Further work is still needed on concrete biological models of interesting phenomena via the multilayer approach.

Recent research has challenged the current premises in memory formation. The use of new techniques such as optogenetics has made it possible to differentiate between mechanisms of memory retrieval and memory storage [108, 95, 107]. New theoretical and computational models, such as the one described here, will be required to explain these new observations.

More generally, the use of multilayer adaptive networks for the study of neuronal circuits will lead to analyze the computational capabilities added by the additional layers. These computational capabilities (e.g., adaptability, regulation, robustness, degeneracy, memory, recurrency) will be key to understand the particularities of information processing in the brain and relate them to those of computers.

Furthermore, cognitive scientists are calling for greater integration between neuroscience and artificial intelligence. This enlarged framework will allow to establish synergies and points in common between neuronal processing and current techniques such as machine learning. Machine learning needs new approaches to imitate how the brain learns and operates and, therefore, it may be relevant to consider how neuromodulation and biochemical communications condition the processing of information.

4 | Differentiable programming: Generalization, characterization and limitations of deep learning

4.1 Introduction

In recent years, different deep learning models (neural networks, RNNs, CNNs,...) have been developed and successfully applied to cognitive tasks such as natural language processing, gaming, computer vision and more [69]. Although these models were originally biologically inspired, they are specific examples of differentiable programs and the key to their success are the relationships and transformations of the elements (tensors) of the model. A differentiable program stands here for any tensor transformation whose parameters (trained end-to-end) are differentiable.

Differentiable programming is a programming framework that is expressive enough to contain the space of all functions, and flexible enough to incorporate the priors and the structure of the problem when we define a new program. In traditional programming there are certain techniques for constructing algorithms (divide and conquer, recursion, dynamic programming,...) that depend on the problem to be solved. For differentiable programming, it is interesting to have also some characteristics that allow us to incorporate the structure and priors of the problem at hand.

To be more specific, by the structure and priors of a problem we mean the knowledge we have of the problem: temporal dependency, symmetries, relationship between its components, etc. The program characteristics are a series of properties and relations between the elements (tensors) of a differentiable program that allow us to incorporate this prior knowledge and reduce the hypothesis

space in which learning is carried out.

In this chapter we define and motivate differentiable programming, as well as specify some program characteristics that are relevant to construct differentiable programs and adapt them to important classes of problems. Furthermore, we analyze several domains or types of differentiable programs, from more general to more specific, and map the problem knowledge and structure to these programs using such characteristics. This procedure will be exemplified with the CiteSeer dataset [124] and the problem of classifying scientific publications into one of six classes.

We also discuss the limitations of differentiable models and possible solutions, which is a central theme for the future of artificial intelligence. Specifically, we analyze how the characterization and structure of the models allow their combination and extension.

In sum, the topic of this chapter is differentiable programming, which is a programming model that generalizes deep learning. We call the concrete examples or models that it generates differentiable programs, which are represented by acyclic directed graphs. The learning process is done in two steps: In the forward pass, the graph is built; in the backward pass, the parameters are adjusted by gradient descent.

In recent years, the term differentiable programming has begun to be used as a generalization of deep learning. However, although emphasis has been placed on automatic differentiation as the key to differentiable programming [15, 114], a definition and analysis of differentiable programming as a new programming paradigm has not been made. Moreover, specific techniques have not been defined to build and characterize programs and relate them to the problem to be modeled. To overcome these shortcomings, this chapter makes the following contributions:

1. It provides a definition and motivation for differentiable programming, frames it within deep learning, and compares it to traditional programming.
2. It highlights the components of a differentiable program that make it very flexible and adaptable.
3. It defines techniques and characteristics to build differentiable programs and incorporate the structure and priors of the considered problem.
4. We analyze different typical deep learning models using the defined techniques and characteristics.

5. We evaluate the structure and priors of a particular problem (the classification of the nodes of a graph) with various differentiable programs using these characteristics.
6. We discuss the limitations of differentiable programming that are imposed by its very definition as data-defined programs and analyze some solutions.

In the following table we show the different steps and questions in the formalization of differentiable programming and their location in this chapter.

Aspect of differentiable programming	**Section**
Motivation, definition and relation to deep learning and classical programming	Section 2
Expressiveness and flexibility to model problems with different structure	Section 3, first part
Techniques and characteristics to build programs	Section 3.1
Heuristics to relate the programs with the problem to be solved	Section 3.1
Analysis of current deep learning models from the point of view of differentiable programming, its flexibility and characterization	Section 4
Practical experiments. Define and implement different differentiable programs to solve a concrete problem using the previously defined characteristics	Section 5
Inherent limitations of differentiable programming and analysis of possible solutions	Section 6

Table 4.1: Different aspects of differentiable programming and their location in this chapter.

This chapter is organized as follows. In the next section we describe, motivate and define differentiable programming. In Section 4.3 we discuss its flexibility and propose a characterization. In Section 4.4 we study general programs and programs that incorporate very specific priors, while in Section 4.5 we perform a series of experiments to analyze a particular graph problem using the defined characteristics. Finally, in Section 4.6 we describe the limitations of differentiable models and analyze possible solutions. The conclusions winds up this chapter.

4.2 Differentiable programming. Motivation and definition

In the past years we have seen major advances in the field of machine learning. Deep neural networks, along with the computational capabilities of Graphics Processing Units (GPUs) [123] have improved the performance of several tasks, including image recognition, machine translation, language modelling, time series prediction and game playing [69, 105, 100].

Let us recall that, in a feedforward neural network (FNN) composed of multiple layers, the output (without the bias term) at layer l is given by

$$\boldsymbol{x}^{l+1} = \sigma(W^l \boldsymbol{x}^l), \tag{4.1}$$

where W^l is the weight matrix at layer l, σ is the activation function, and $\boldsymbol{x}^{l+1}$ is the output vector at layer l and the input vector at layer $l+1$. The weight matrices for the different layers are the parameters of the model.

Deep learning is a part of machine learning that is based on neural networks and uses multiple layers, where each layer extracts higher level features from the input. Although deep learning can implicitly implement logical reasoning [55, 115, 23], it has limitations that make it difficult to achieve more general intelligence. Among such limitations, let us mention that deep learning only performs perception and does not carry out conscious and sequential reasoning [74].

Simplified, the learning problem can be formulated as follows. Given an input space $\mathcal{X}$ and an output space $\mathcal{Y}$, let the pairs $(X, Y) \in \mathcal{X} \times \mathcal{Y}$ be random variables distributed according to a joint probability mass or density function $\rho(x, y)$. In this setting, construct a function $g : \mathcal{X} \to \mathcal{Y}$ from a hypothesis space of functions $\mathcal{H}$ which predicts Y from X after observing a sequence of n pairs (x_i, y_i) independently and identically distributed according to $\rho(x, y)$.

And a learning algorithm over $\mathcal{H}$ is a computable map from (X, Y) to $\mathcal{H}$, more precisely, an algorithm that takes as input the sequence of training samples and outputs a function $g : \mathcal{X} \to \mathcal{Y} \in \mathcal{H}$ with a low probability of error $P(g(X) \neq Y$.

As a general rule we are interested in having a hypothesis space (set of functions) $\mathcal{H}$ as expressive as possible. However, to solve a concrete problem, it is necessary to restrict this hypothesis space taking into account the structure and priors of the problem. This is what we have seen in recent years with the development of specific models and graph structures. For example, multilayer neural networks are based on the compositional and hierarchical structure of information, RNNs (Recurrent Neural Networks) on temporal dependence, CNNs (Convolutional Neural Networks) on translational invariance, etc.

Therefore, to advance and generalize deep learning, a programming model or framework with the following characteristics is necessary.

i Be expressive enough to define the space of all hypotheses or functions.

ii Be able to incorporate the structure and the priors of the problem we want to model. The incorporation of prior knowledge about the underlying task or class of functions can make the learning process more efficient.

iii Be able to define new primitives to advance deep learning capabilities (reasoning, attention, memory, modeling of physical problems, etc.)

As we are going to see, a natural evolution to move forward in these directions and overcome the limitations of classical models is to generalize deep learning using a framework composed of differentiable blocks. This approach, called differentiable programming, adds new differentiable components to traditional neural networks. For the purposes of this chapter, differentiable programming is defined as follows.

Differentiable programming is a programming model defined by a tuple $\langle n, l, \boldsymbol{E}, f_i, v_i, \alpha_i \rangle$ where:

1. Programs are directed acyclic graphs.
2. Graph nodes are mathematical functions or variables and the edges correspond to the flow of intermediate values between the nodes.
3. n is the number of nodes and l the number of input variables of the graph, with $1 \leq l < n$. v_i for $i \in \{1, ..., n\}$ is the variable associated with node i.
4. $\boldsymbol{E}$ is the set of edges in the graph. For each directed edge $(i, j) \in \boldsymbol{E}$ from node i to node j we have $i < j$, therefore the graph is topologically ordered.
5. f_i for $i \in \{(l+1), ..., n\}$ is the differentiable function computed by node i in the graph; α_i for $i \in \{(l+1), ..., n\}$ contains all input values for node i.
6. The forward algorithm or pass, given input variables $v_1, ..., v_l$ calculates $v_i = f_i(\alpha_i)$ for $i = \{(l+1), ..., n\}$.
7. The graph is dynamically constructed and composed of functions that are differentiable and whose parameters are learned from data.

To learn the parameters of the graph from the data, automatic differentiation is used. Automatic differentiation, in its reverse mode and in contrast to manual, symbolic and numerical differentiation, computes the derivatives in a two-step process [15, 114].

In the last years, deep learning frameworks such as PyTorch have been developed that provide reverse-mode automatic differentiation [88]. The define-by-run philosophy of PyTorch, whose execution dynamically constructs the computa-

tional graph, facilitates the development of general differentiable programs.

Differentiable programming can be seen as a continuation of the deep learning end-to-end architectures that have replaced, for example, the traditional linguistic components in natural language processing [32, 44]. Differentiable programs are composed of classical blocks (feedforward, recurrent neural networks, etc.) along with new ones such as differentiable branching, attention, memories, etc.

Differentiable programming is an evolution of classical (traditional) software programming where, as summarized in Table 4.2:

i Instead of specifying explicit instructions to the computer, an objective is set and an optimizable architecture is defined which allows to search in a subset of possible programs.

ii The program is defined by the input-output data and not predefined by the user.

iii The optimizable elements of the program have to be differentiable, say, by converting them into differentiable blocks.

Classical Programming	**Differentiable Programming**
Sequence of explicit instructions	Sequence of differentiable primitives
Fixed architecture	Optimizable architecture that searchs in a subset of possible programs
User defined programs	Data defined programs
Imperative programming	Declarative programming, specifying the objectives but not how to achieve them
Direct, intuitive and explainable	High level of abstraction

Table 4.2: Differentiable vs classical programming.

RNNs, for example, are an evolution of feedforward networks because they are classical neural networks inside a for-statement (a control flow statement for iteration) which allows the neural network to be executed repeatedly with recurrence. At each time interval, the network integrates the current input and the previous state. However, this for-statement is a predefined feature of the model. Differentiable programming allows to dynamically construct the graph and vary the length of the iteration. The ideal situation would be to augment the neural network with programming primitives (for-statement, if branches, while statements, external memories, logical modules, etc.) that are not predefined by the user but learned with the training data.

But many of these programming primitives are not differentiable and need to be converted into optimizable modules. For instance, if the condition a of an "if" primitive (e.g., if a is satisfied do $y(x)$, otherwise do $z(x)$) is to be learned, it can be the output of a neural network (linear transformation and a sigmoid function) and the conditional primitive will transform into a convex combination of both branches $ay(x) + (1 - a)z(x)$. Similarly, in an attention module, different weights that are learned with the model are assigned to give a different influence to each part of the input.

Figure 4.1 shows an example of a differentiable program. $\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_T}$ are the input nodes, where $\boldsymbol{x_t}$ is the vector associated to node t. The graph is built dynamically by connecting each input node with the output node using an edge. This output node computes a weighted sum of the input node vectors, $\sum_{i=1}^{T} a_i \boldsymbol{x}_i$, where a_i are the weights of the function, which depend on the parameters. In the backward pass the parameters are calculated based on the training data.

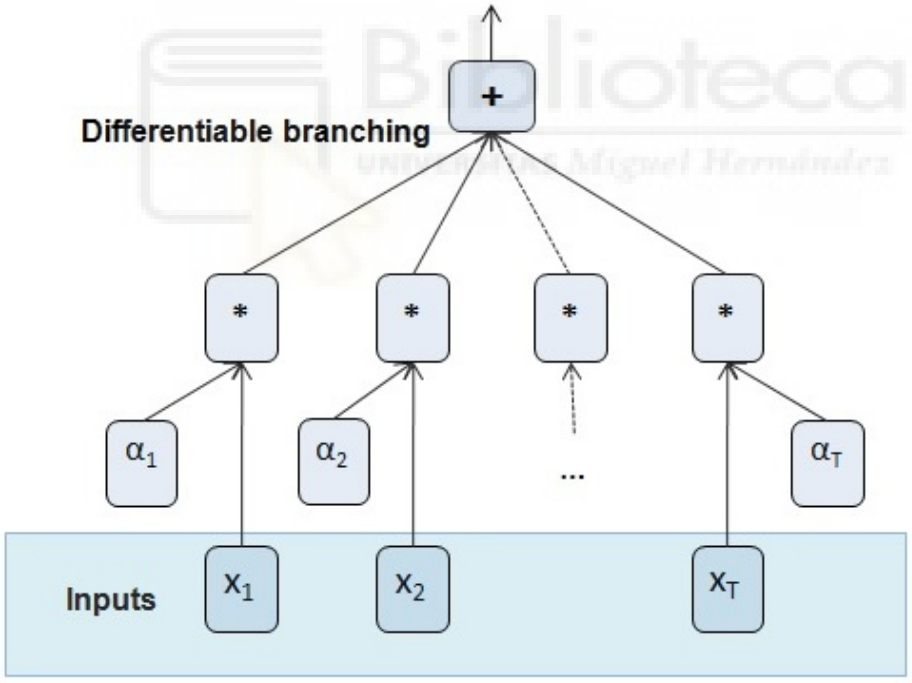

Figure 4.1: Differentiable program (computational graph) of differentiable branching.

4.3 Flexibility and characterization

From the definition provided, it can be seen that differentiable programs are very flexible and give rise to very varied structures, which allows to restrict the hypothesis space in an adequate way.

The following elements of differentiable programs support such flexibility and variability.

i Nodes can be tensors of any rank. E.g. scalars (0-rank tensors) v, vectors (1-rank) v_i, matrices (2-rank tensors) v_i^j, general tensors $v_{i_1 i_2 \ldots}^{j_1 j_2 \cdots}$.

ii The function f_i computed by each node i can be any transformation of the input tensors, as long as the program (composite of functions f_i) can represent any continuous function (universal approximator). If the function f_i has parameters, it has to be differentiable with respect to the parameters. Examples are linear transformation of a vector $\boldsymbol{A}\boldsymbol{v}$, a non-linear function of a vector $\sigma(\boldsymbol{A}\boldsymbol{v})$, a weighted sum of vectors $\sum \alpha_i \boldsymbol{v}_i$, etc.

iii The program not only performs a transformation of the input but can also integrate information in a very flexible way. As we will see in the next subsection, information nodes can be integrated and related using different graph paths.

iv The graph is dynamically constructed for each input at execution. To do this, each of the primitives is executed sequentially, be they classic programming primitives (for, while, if, etc.) or differentiable functions. Therefore, the graph can be different depending on the input.

v Parameters can be set at training time or dynamically calculated. For example, the attention weights α_n in Figure 4.1 usually depend on both a fixed parameter and the input.

Thus, differentiable programming generates, from a general hypothesis space $\mathcal{H}$ and a set of priors $\mathcal{P}_i$, a restricted hypothesis space $\mathcal{H}_1 \subset \mathcal{H}$. The learning algorithm over $\mathcal{H}_1$ takes as input the sequence of training samples, builds the graph, computes the derivatives and outputs a program or model $g \in \mathcal{H}_1$ (see Figure 4.2).

To be more specific, differentiable programming translates the prior knowledge and structure of the problem to the elements of a differentiable program, i.e. to a tuple $\langle n, l, E, f_i, v_i, \alpha_i \rangle$. It is precisely this flexibility that makes it easy to generate

the restricted hypothesis space $\mathcal{H}_1$.

For example, the program defined in Figure 4.1 would be the restricted hypothesis space $\mathcal{H}_1$. The structure of the problem would be a transformation of input vectors, without any vector having more importance a priori than another. The learning algorithm takes the sequence of training samples and outputs the concrete function $g = \sum_{i=1}^{T} a_i \boldsymbol{x}_i$ from $\mathcal{H}_1$ with the learned parameters.

Next, we define some program characteristics that help in this process. They are also instrumental to define differentiable programs and adapt them to important classes of problems.

4.3.1 Program characteristics

In traditional programming there are certain heuristics to construct algorithms depending on the problem to be solved. New structures have also been developed in neuroscience that help explain neural processing (Chapter 3).

For differentiable programming, it is also useful to have some characteristics that allow us to incorporate the structure and priors of the considered problem and generate a restricted hypothesis space $\mathcal{H}_1 \subset \mathcal{H}$, as seen in Figure 4.2.

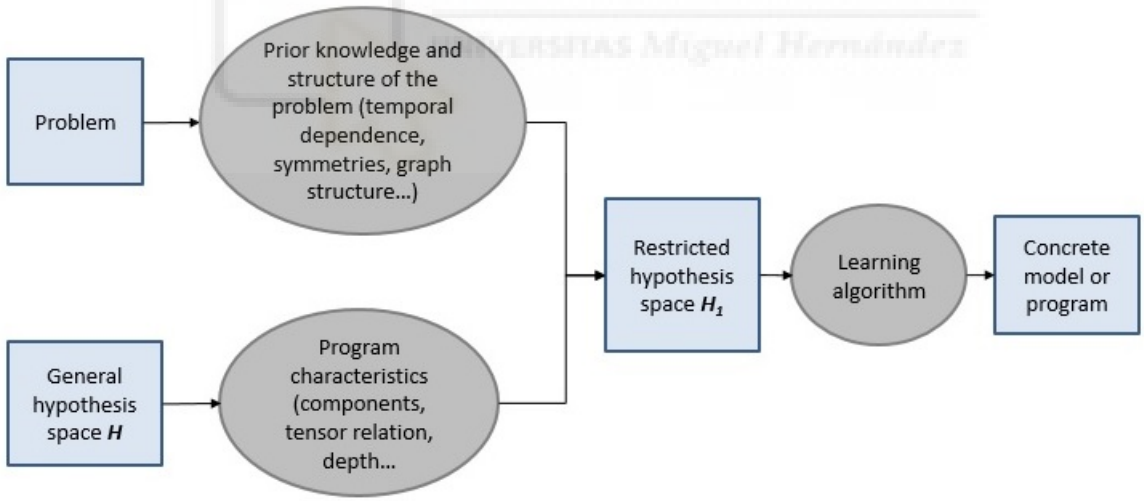

Figure 4.2: Differentiable programming. From the problem to the model

Traditionally, the characteristics of differentiable and deep learning programs (multilayer feed-forward networks, convolutional networks, etc.) were inspired by

the brain. However, as we are going to see, the key to their success has actually been the concrete relationships and transformations of the elements (tensors) of a differentiable program.

These characteristics are not independent but related to each other. Here we define some of these characteristics.

1. Integration and relation of information tensors (vectors). Given a sequence of input vectors $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$ with $\boldsymbol{x}_t \in \mathbb{R}^n$ and a sequence of intermediate or output vectors $\boldsymbol{y} = (\boldsymbol{y}_1, ..., \boldsymbol{y}_{T\}})$ with $\boldsymbol{y}_t \in \mathbb{R}^m$, a program characteristic is the graph path relationship (shortest path, non-intersecting paths...) between each vector.

   For example, in RNNs the length of the path between the output vector $\boldsymbol{y}_t$ and the past input vector $\boldsymbol{x}_{t-l}$ increases with l.

2. Relationship between the represented problem structure (locality, local or distant relations, node degree, temporal relations, etc.) and the differentiable program characteristics (depth, edges information, temporal evolution, etc.).

   An example of this characteristic is the relationship between the average degree of a graph and the depth (number of layers) of its corresponding graph neural network (GNN).

3. Invariant transformations of data and symmetries. If f represents the function that transforms the data x and $\boldsymbol{T}$ is a function that performs a transformation of the data, f is invariant under $\boldsymbol{T}$ if and only if $f(\boldsymbol{T}(x) = f(x)$.

   For example, convolutional networks (CNNs) have translational invariance, that is, the model produces the same response despite translations of input elements.

   In this way, we restrict the structure of the differentiable program taking into account certain symmetries of the information. Another example would be a transformation of input vectors that is invariant under a change of order or permutation of the vectors. Given a sequence of input vectors $\boldsymbol{x} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$, the transformation $f(\boldsymbol{x})$ does not depend on the ordering of the set of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T$.

4. Combination of modules or tasks, that is, the possibility of combining different modules using classical or differentiable primitives. A model $\boldsymbol{M}$ is composed of different parts or tasks $\boldsymbol{T}_i$ and each task $\boldsymbol{T}_i$ has a direct mean-

ing or some grounding.

Thus, by imposing restrictions on the structure, it is possible to better approximate the real distribution of the problem and be able to carry out subsequent tasks.

For example, different particles that interact with each other in a physical model could correspond to the different nodes of a graph neural network. Also, as in [60], a task can be broken down into a series of reasoning steps, thus learning to perform iterative reasoning.

In Section 4.4 we are going to see several domains or types of differentiable programs in which we analyze and map the problem structure to specific programs using the characteristics defined above.

4.4 From general to specific differentiable programs

In this section we analyze various deep learning models using the defined framework of differentiable programming, its flexibility and its characterization.

4.4.1 Generalization of neural networks. Self-attention

In describing the flexibility of differentiable programming, we stated that the node function f_i can be any transformation of the input tensors, as long as the program can represent any continuous function (universal approximator).

Traditionally and following the inspiration of biological neural networks, a linear transformation (together with the activation function) of a vector has been used, as described in Equation (4.1).

If we want to transform a set of vectors, then we can concatenate the vectors and use the neural network of Equation (4.1) (with a relevant increase in the number of parameters) or use a recurrent neural network (RNN). However, as seen in Figure 4.3, an RNN assumes a one-step temporal dependence, which implies that the length of the path between the output vector $\boldsymbol{y}_{t+1}$ and the past input vector $\boldsymbol{x}_{t-l}$ increases with l. Consequently, the relationship between the output $(\boldsymbol{y}_1, ..., \boldsymbol{y}_T)$ and the input vectors $(\boldsymbol{x}_1, ..., \boldsymbol{x}_T)$ is not symmetric. As a result, this model is biased towards certain problem structures.

Therefore, a more general transformation of the input vectors should have the

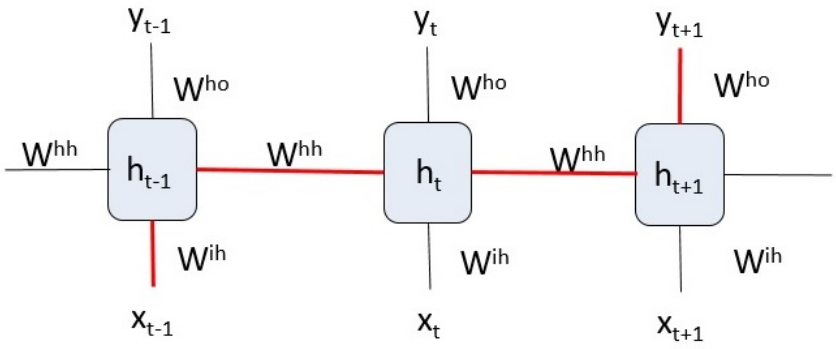

Figure 4.3: Computational graph of a recurrent neural network.

following characteristics.

i It acts on a set of vectors, consciously and dynamically choosing the most important ones.

ii It has to integrate all the input vectors in a direct and symmetric way (as shown in Figure 4.4), with the same path length between an output vector and each of the input vectors.

iii The weights to integrate each input vector should not be fixed but dependent on the context (input).

The structure described above is the basis of self-attention [112], one of the most successful deep learning models developed in recent years. The input vectors are transformed (using learnable matrices) into query (q), key (k) and value (v) vectors and, at each step, the output is a direct integration of the value vectors based on the similarity between the query and each of the keys, $y_t = \sum_{i=1}^{T} similarity(\boldsymbol{q}_t, \boldsymbol{k}_i)\boldsymbol{v}_i$. Furthermore, this model has made it possible to integrate sequential and conscious reasoning into deep learning models, as seen in Chapter 5.

4.4.2 Compositional structures and reasoning

The problems that we want to model using machine learning usually have a certain structure and are made up of a sequence of different tasks or modules. For example, human reasoning sometimes takes place in a series of phases, with each phase focusing on different information and feeding the next phase.

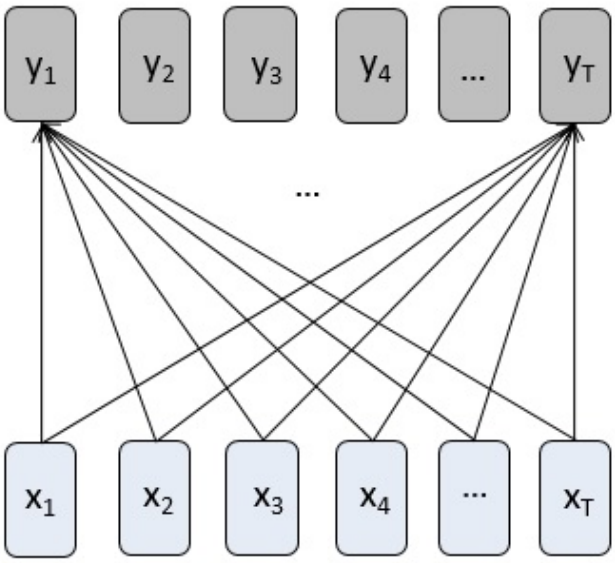

Figure 4.4: A direct and symmetric transformation of input vectors.

To make learning more efficient, the same structure can be transferred to the differentiable program, which will be made up of a sequence of modules. For example, a problem can be decomposed into a sequence of attention tasks. Each task focuses on a set of elements of information (text, image, memory, etc.) and is the input of the next task, as depicted in Figure 4.5.

Thus, a differentiable program can be very flexibly defined as a sequence or combination of tasks or modules. As stated in Chapter 5, an attention module can be added to any deep learning model in multiple ways: in several parts of the model; focusing on different elements; with different attention dimensions (spatial, temporal and input dimension), etc.

Even more, as in Recurrent Independent Mechanisms (RIMs) [46], there can be multiple modules that operate independently and compete with each other at each step to read from the input and update their states, while attending only the parts of the input relevant to that module.

4.4.3 Graph structure restrictions. Graph neural networks

After the success of deep learning in image processing, time series analysis, etc., it is logical to apply it to unstructured data and graphs.

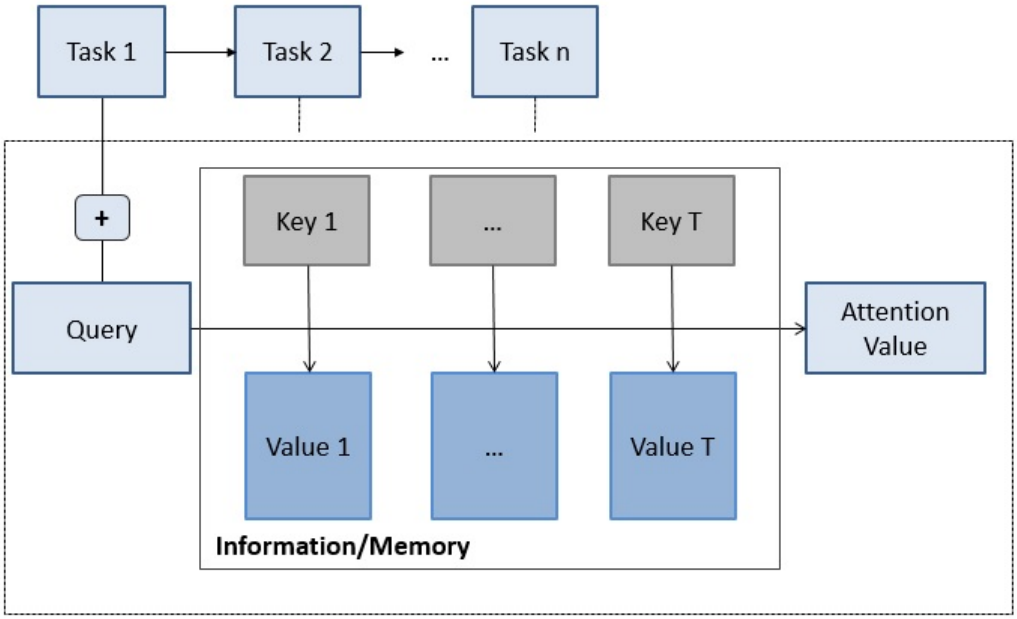

Figure 4.5: A sequence of attention tasks.

Graph neural networks [39, 94] apply neural networks to graphs, based on nodes connected to each other. To do this, each state x of the node v of the graph $G = (V, E)$ is iteratively updated by adding information from the neighboring nodes $\mathcal{N}(v)$:

$$\mathbf{x}_v^k = f_\theta^k(\mathbf{x}_v^{k-1}, \Gamma_{j \in \mathcal{N}(v)} \phi^k(\mathbf{x}_v^{k-1}, \mathbf{x}_j^{k-1}, \mathbf{e}_{jv})), \tag{4.2}$$

with $\mathbf{x}_v^{k-1}$ denoting F-dimensional vector features of node v in layer $k-1$ and $\mathbf{e}_{jk}$ denoting optional edge features from node j to node v. Γ denotes a differentiable, permutation invariant function (e.g. a summation) and f and ϕ denote differentiable functions such as feedforward neural networks (FNN).

In Figure 4.6, where $\mathbf{x}_v^1$ represents the vector features of node v in the original graph, we can see two consecutive updates of information among the nodes of the graph.

Based on that definition, certain relationships can be drawn between the structure of the represented graph and the characteristics of GNNs:

1. Information aggregation in the represented network is performed via the

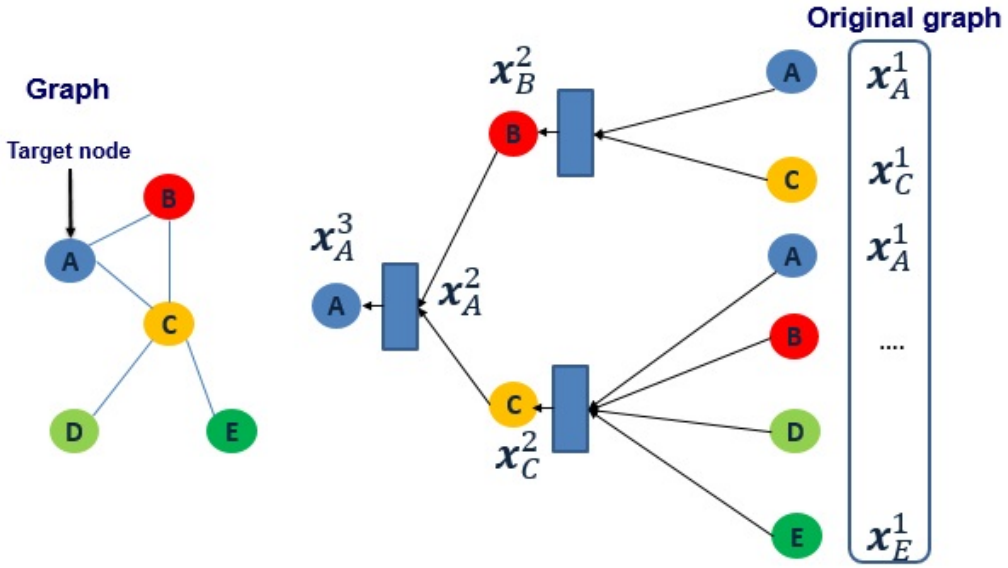

Figure 4.6: Two consecutive updates of information between nodes of a graph.

layers of the GNN that add information from the neighboring nodes. The more layers, the further information travels from a node.

2. Small world networks (most nodes are separated by a short distance) do not need many layers.
3. In theory, in graphs with a high average degree a GNN with few layers is enough to propagate the information of the nodes. However, the high average degree can cause saturation since the GNN nodes receive information from many nodes at the same time.
4. Although increasing the number of layers of the GNN allows information to propagate between nodes, the task we perform at each node (e.g., prediction) may depend on the local neighborhood of a node or on distant information. Hence, two different graphs or problems but with similar structure (similar adjacency matrix) may require different GNNs.
5. There are some invariant transformations of information. For example, the function that integrates the information of the neighboring nodes (e.g. summation) should be permutation invariant.

4.5 Experiments

4.5.1 Materials and Methods

In this section we evaluate the structure and priors of a particular problem with various differentiable programs (models) using the characteristics defined in the previous sections. This problem is the classification of scientific publications into one of six classes with the CiteSeer dataset [124]. We use the PyTorch framework, especially PyTorch Geometric [39] and self-attention libraries.

The description and preparation of the CiteSeer problem are as follows:

i It is a network that contains 3327 nodes that represent publications.

ii Each node (publication) is represented by a vector of 3703 features with the 0/1 value of each feature indicating the absence/presence of the corresponding word from the dictionary.

iii The nodes are connected by links, which represent citations between publications. There are 9104 links or edges, that are undirected (the links appearing twice in the link matrix).

iv The average degree is 2.737, the average clustering coefficient is 0.141 and there are isolated nodes.

v Each node belongs to one of six classes, which represent publication categories.

vi The problem is to categorize each node given the input vector.

vii Initially 120 nodes are used for training, 500 for validation and 1000 for test, although this will change according to the experiment.

viii Node features have been normalized.

Next, the justification and methodology of the experiments are described:

1. We have seen that the classification problem not only has an input-output relationship between the node features and the node classification, but there is more structure such as link information.
2. Furthermore, as the dataset represents a graph, there are measures to characterize the graph such as the clustering coefficient, the mean degree, etc.
3. In sections 3 and 4 we have seen that there are different techniques and characteristics to adapt a differentiable program to the structure of a prob-

lem: using a neural network or self-attention to integrate the input vectors, using composition, taking into account the topology of the graph in the hyper-parameters of the GNN, etc.

4. In the experiments we are going to define and implement different models to solve the problem. Through the use and combination of different building blocks and hyper-parameters, we can incorporate the structure of the problem to a greater or lesser extent. Examples of these building blocks and hyper-parameters are: Neural networks and the number of layers; GNNs [39] and the information of the links; self-attention [112] and the attention mask; etc.
5. Then, the aim of the experiments is not to obtain a model that exceeds state-of-the-art performance but to evaluate various programs (models) with different previous structure.
6. To carry out the experiments, virtual machine instances of Google Colab have been used. For the experiments with the self-attention module, a GPU has been used to speed up the process.

4.5.2 Results

The results are described below, where we have varied the following elements of the model to adapt it to the structure of the problem:

i The base component of the model: Neural networks, GNNs, self-attention, etc.

ii The hyper-parameters of the model: number of layers, learning rate, dropout layers, pre-linear and post-linear layers after the self-attention module, etc.

iii The number of training nodes.

First, we use non-linear transformations (neural networks) of the vectors of each node without taking into account the graph structure (links):

1. Using a neural network (without any relational information) with two layers (input and output layer), 59,366 parameters, with node features as inputs and training with the training mask (120 nodes), the test accuracy is 58.2%. The model suffers overfitting due to only a small amount of training nodes and because it does not take into account the link information. Therefore, it generalizes poorly to unseen node representations.
2. Using a neural network (without any relational information) with three

layers and with node features as inputs, intermediate dimensions 512 and 256 (2,029,318 of parameters) and training with the training mask, the test accuracy is 59%, that is, slightly above the 58.2% of the previous configuration. Using the validation mask (500 nodes) to train the model, the loss decreases more slowly and the test accuracy is around 68%.

Second, we apply graph neural networks by adding information from the neighboring nodes:

1. Using a GNN with two graph convolutional layers and 59,366 parameters, the test accuracy is around 71.4%. If we use the validation mask to train, the test accuracy is 76.2%.
2. Using a GNN with three graph convolutional layers, we see that the test accuracy decreases to 62% as the intermediate dimensions increase, probably because the model is overfitting the training data. If we lower the feature dimensions to 16, then the test accuracy returns to 67.4%. If we use the validation mask to train the model, we get a test accuracy of 73.3% with feature dimensions of 256 and 1,015,558 parameters.

Third, we use a GNN with two graph convolutional layers and 59,366 parameters but modifying the number of training nodes to assess the influence of the training size:

1. When the first 620 nodes are used for training the GCN to avoid overfitting, we get a test accuracy of 77%. We obtain the following training set-test accuracy: 50 nodes-62.1%, 70 nodes-63.6%, 120 nodes-71.4%, 400 nodes-77%, 500 nodes-77.3%, 620 nodes-77%, 800 nodes-76.80%, 1000 nodes-76.2%, 1500 nodes-76.9%. Using the validation mask to train, we get a test accuracy of 76.2%.

Fourth, we use a GNN with two graph convolutional layers and 59,366 parameters but modifying the links (edges) of the original graph:

1. Using random edges (directed edges) between the nodes and training with the training mask, the test accuracy is 16.5% compared to the 71.4% obtained with the correct edges.
2. Using the training mask and a (i) 20% of the edges removed gives a test accuracy 67.7%; (ii) 25% removed, 68.7%; (iii) 33% removed, 67.1%; (iv) 50% removed, 68.1%; (v) 66% removed, 62%; (vi) 75% removed, 61.1%; and (vii) 80% removed, 59.2%; compared to the 71.4% test accuracy obtained

with the correct edges and the 58.2% of the neural network.

Finally, we design a model with a self-attention module between graph nodes, a pre-linear and post-linear layer for adjusting dimensions and a dropout layer. The model has 503,286 parameters and the attention mask is modified to prevent attention to certain positions:

1. Attention is applied to all the nodes of the graph to learn the importance weights between the nodes. Then, training with the training mask the accuracy is 18.10%, while training with the first 1500 nodes the accuracy is 23.10%.
2. Attention is applied only between the same node. Then, training with the training mask the accuracy is 51.30%, while training with the first 1500 nodes the accuracy is 69.10%.
3. Attention is applied only between neighboring nodes, thus respecting the graph structure and learning the weights between neighboring nodes. Then, training with the training mask the accuracy is 65.40%, while training with the first 1500 nodes the accuracy is 73%.

4.5.3 Discussion

As describe in section 4.2, given the training data, the learning algorithm outputs a function $g : \mathcal{X} \to \mathcal{Y}$ from the hypothesis space $\mathcal{H}$ with a low probability of error.

There is an ongoing debate in deep learning between general models or more specific models, whether learning is conditioned and possible by the innate structure of the model or simply with the data and the environment is enough. For example, in the brain the necessary structure has been developed to relate objects, space and time.

Is it better to constrain the hypothesis space $\mathcal{H}$ using the structure of the problem or to use a general model? In recent years, general models such as Transformers have been successfully applied to many tasks but with large amounts of training data. For concrete problems with a limited amount of training data, is better a general model or a model with a prior structure?

The present problem, node classification on CiteSeer, has a lot of prior structure (graph links, graph measures such as the clustering coefficient, the mean degree, etc.) and current best performing models are about 80% in accuracy. Ta-

ble 4.3 summarizes the results of evaluating various models that incorporate to a greater or lesser extent this structure.

Model	Structure	Performance
Neural network with two layers	Node connections are not taken into account	58.2%
Training with the training mask (120 nodes)		
Neural network with tree layers	Node connections are not taken into account	
Training with the training mask		59%
Training with the validation mask (500 nodes)		68%
GNN with two convolutional layers	Link information is taken into account	
Training with the training mask		71.4%
Training with the validation mask		76.2%
GNN with three convolutional layers	Link information is taken into account	
Training with the training mask		67.4%
Training with the validation mask		73.3%
GNN with two convolutional layers	Link information is taken into account	
Training with 50 nodes	The training size is modified	62.1%
Training with 500 nodes		77.3%
Training with 800 nodes		76.8%
Training with 1500 nodes		76.9%
GNN with two convolutional layers and	Using random links	16.5%
training with the training mask	20% of the edges removed	67.7%
	33% of the edges removed	67.1%
	66% of the edges removed	62%
	80% of the edges removed	59.2%
Self-attention module and training with	Attention to all nodes	23.10%
the first 1500 nodes	Attention between the same node	69.10%
	Attention between between neighboring nodes	73%

Table 4.3: Generic vs specific programs. Evaluation of various programs (models) that incorporate to a different degree the structure of the problem (classification of nodes in the CiteSeer database)

When conventional neural networks are used, without incorporating the links of the graph, the performance was acceptable (58.2%). Increasing the number of layers and parameters of the neural network did not significantly increase accuracy.

When GNNs are applied by adding information from the neighboring nodes, the accuracy increased significantly, even more when the number of training nodes is increased (76.2%).

However, when more convolutional layers are added to the GNN, the test accuracy decreased. As described in Section 4.4.3, in the graph-like CiteSeer dataset, with an average degree of 2.74, increasing the number of layers helps to propagate the information between distant nodes. But probably the task that we perform at

each node or document (classification) only depends on the local neighborhood and not on distant nodes, so, in such cases, we do not need any more layers in the GNN.

Also, when we purposely changed the links of the graph when implementing the GNN, the accuracy decreased. When random links are used between the nodes, the accuracy was very bad (16.5%). Removing an increasing percentage of the links lowered the accuracy, approaching the accuracy achieved using neural networks instead of GNNs.

Finally, a more general transformation such as self-attention is evaluated. As stated in 4.4.1, it transforms a set of vectors (nodes), dynamically choosing the most important ones. The model has to learn the graph structure. When we applied attention to all the nodes of the graph to learn the importance weights between the nodes, the accuracy was very low (23.1%). When we applied attention only between neighboring nodes, thus respecting the graph structure, accuracy went up to 73%. Therefore, general differentiable programs such as self-attention, applied to problems with structure and without a great amount of data do not work well.

Regarding the computational complexity, the self-attention layer has a complexity of $O(n^2 \cdot d)$ where n is the number of vector (nodes) and d the dimension of the vectors. If neural network layers are used to transform each node, the total complexity is $O(n \cdot d^2)$.

Therefore we have seen that for a problem with a defined structure and a limited amount of training data, it is better to use specific models adapted to that previous structure than general ones. In addition, it is necessary to emphasize the following limitations in the experiments:

i For the experiments, a problem with a graph structure and few training data has been intentionally chosen instead of a more general problem.

ii Key deep learning models have been used, extending them and modifying the hyper-parameters.

iii The design and analysis of the experiments have been focused on the flexibility and characterization of differentiable programming, proposed in the previous sections.

4.6 The limits of differentiable models and the path to new learning strategies

As we have seen, in differentiable models, the learning algorithm takes as input a sequence of training samples and outputs a function $g : \mathcal{X} \to \mathcal{Y} \in \mathcal{H}$, from the hypothesis space of functions, with a low probability of error $P(g(X) \neq Y$. These models are composed of any transformation of tensors and if the transformation has parameters to be learnt, it has to be differentiable.

We identify the following differences and limitations, summarized in Table 4.4, of differentiable programming compared to human learning:

1. Differentiable models learn the transformations of the input that minimize the error to predict the training data.
2. Differentiable models do not learn certain aspects of the task that are later combined to generate new knowledge. They cannot generate new tasks or information without additional training data.
3. By contrast, humans can correctly interpret novel combinations of existing concepts, even if those combinations have not been seen previously.
4. Differentiable models are only based on input data and are not grounded on other information like humans do.
5. Human learning uses a lot of previous knowledge guided by logic to reason and generate new information.
6. Differentiable models scale well in data (when we increase the input data) but not in novel tasks, while in humans it is the other way around.

Characteristic	Differentiable programming	Human learning
Functional complexity	Functional approximators	Limited functional complexity
New knowledge	No new knowledge without additional training data	Interpret novel concepts
Previous data	Only uses the model training data and the model input	Combination of previous knowledge to generate new information Large base of acquired knowledge
Scalability	Good data scalability	Good scalability on new tasks
Increasing complexity	Only one model	Increasingly complex model of the world based on simpler models

Table 4.4: Evaluation of the different learning characteristics of differentiable programming and human learning

To overcome these limitations we need learning strategies that leverage previous data and models to generate new experiences and knowledge. These new learning strategies should have the following characteristics:

i Be similar to the learning strategies developed by humans but adapted to the machine learning framework. As pointed out in [67], humans have two levels of learning. Level 1, which is more automatic, continuous and unconscious, and level 2, which comprises the deliberate learning strategies that create the experiences for level 1.

ii In the framework of machine learning, level 1 corresponds to differentiable models that approximate a function and level 2 to learning strategies that, based on previous data and models, create new experiences and tasks.

iii These learning strategies are algorithmic transformations of existing models and data. They build an increasingly complex model of the world and can be more logical (some logical combination of models characteristics or outputs) or automatic.

iv Given the training data (x_i, y_i) and task T_j for each model, a learning strategy is an algorithm $A : H_u \rightarrow g_u$, where H_u are transformations of all previous information (training data, model information, etc.) and g_u is a new model that, when presented with an input and a task not seen in the training data, is capable of getting the correct output for this task.

v They can be based on any previous information: Priors of the models, information of the trained models (intermediate variables, output), training data, automatically generated labels, etc.

Next, we analyze some current examples of these strategies.

4.6.1 Large language models

Transformations of training data that scale in tasks. A large amount of data (e.g., text) is collected in the form of $(Task\,1, data)...(Task\,n, data)$ and the differentiable model learns to map a bigger space: $probability(output|task; input; previous\ output)$. The model predicts an output by also conditioning on the task in addition to the input and previous output. In this way, a large amount of data is transformed and adapted to build differentiable models that scale in tasks. GPT-X models [92, 20], based on language models which are trained in an unsupervised manner on web datasets, are a great advance in this line.

4.6.2 Transformation using model priors

We first train one or several differentiable models (e.g., GNNs or attention models) used for some tasks. Each of these models has a series of internal components or variables with a physical meaning, $v_1^i, v_2^i, ..., v_n^i$ for model i. Then we apply some machine learning technique (e.g., another differentiable model, symbolic regression, etc.) to extract some relations $R(v_1^1, ..., v_n^i)$ between the variables of the models. In this way we can generalize and perform new tasks. For example, if we have several attentional convolutional models that perform different tasks on an image, we can relate the attention weights of the different models.

In sum, artificial intelligence models need to build an increasingly complex model of the world. In this section we have discussed the need, characteristics and examples of these new learning strategies. Nevertheless, much work remains to be done to characterize and specify the learning strategies that leverage existing knowledge (models and data), in a similar way as humans do. Furthermore, these strategies should be as automatic as possible and rely less on human engineering.

4.7 Conclusions

Differentiable programs are very flexible and give rise to a wide variety of structures. This flexibility is due to the translation of the prior knowledge and structure of the problem to the elements of a differentiable program, i.e., a tuple $\langle n, l, E, f_i, v_i, \alpha_i \rangle$. As a result, it is easy to generate the restricted hypothesis space.

Similar to classical programming, it is useful to define some characteristics to incorporate the knowledge and structure of the problem to be solved. These characteristics allow us to analyse the differentiable structures that are used in several domains or models and to define new structures with new capabilities.

We have illustrated with numerical experiments that in problems with few training data it is important to incorporate the structure of the problem into the models rather than using very general models.

We have also seen that a shortcoming with differentiable models is that they just minimize an error to predict the data, but they cannot generate new tasks or information without additional training data. We have presented the characteristics and examples of new learning strategies that leverage existing knowledge (models and data) to generate new knowledge, in a similar way as humans. However much work remains to be done in this regard.

5 Attention mechanisms and their applications to complex systems

5.1 Introduction

The combination of deep neural networks and the computational capabilities of Graphics Processing Units (GPUs) [123] has brought a breakthrough to the field of machine learning, improving the performance of several tasks such as image recognition, machine translation, language modelling, time series prediction, etc. [69, 105, 100, 45].

Recurrent neural networks (RNNs) and long short-term memories (LSTMs), which were specially designed for sequence modelling [22, 117, 116, 71], and convolutional neural networks (CNNs) to a lesser extent, have been successfully used to model, analyze and predict complex systems. Indeed, they are able to capture temporal dependencies and nontrivial relationships in complex systems, specifically in the sequential data generated by them. By complex systems we mean, generally speaking, systems that evolve over time in a possibly more general setting than that of dynamic systems.

However, these classic deep learning models do not perform sequential reasoning [74], a process that is based on perception with attention. In the brain, attention mechanisms allow to focus on one part of the input or memory (image, text, etc) while giving less attention to others, thus guiding the process of reasoning.

Attention mechanisms have provided and will provide a paradigm shift in

machine learning [112, 106]. These mechanisms allow a model to focus only on a set of elements and to decompose a problem into a sequence of attention based reasoning tasks [60]. Moreover, they can be applied to model complex systems in a flexible and promising way. When it comes to their application, information processing in the system and internal structure are crucial.

Here, as shown in Table 5.1, we describe the evolution of machine learning techniques and demonstrate how attention mechanisms, in combination with classic models, allow modeling certain important characteristics of complex systems, e.g., sequential reasoning, integration of different parts and long term dependencies.

Techniques	**Capabilities in modeling complex systems**
Classic models (RNNs, LSTMs...)	Are universal approximators, provide perception, temporal dependence and short memory
Seq2seq with attention	Integrates parts, models long term dependencies, guides a task by focusing on a set of elements (temporal, spatial, features...)
Memory networks	Integrate external data with the current task and provide an explicit external memory
Self-attention	Generalization of neural networks, relates input vectors in a more direct and symmetric way

Table 5.1: Classic deep learning and attention techniques described in this chapter and their capabilities to model complex systems.

In this chapter we review recent progress in attention mechanisms. We focus on differentiable attention, in which the attention weights are learned together with the rest of the model parameters. In Section 5.2 we present a general overview of the use of deep learning in modeling dynamical systems and, more generally, complex systems. We also elaborate on the need for attention mechanisms. In Section 5.3 we present the key aspects, the advantages and the main modes of operation of attention (Section 5.3.1). Then we describe some important attention techniques such as attention in seq2seq models (Section 5.3.2), as well as self-attention and memory networks (Section 5.3.3), emphasizing why they represent significant progress in machine learning. Finally, in Sections 5.4.1-5.4.4 we illustrate some interesting uses of theses techniques to model complex systems and in Section 5.5 we discuss these techniques.

5.2 Traditional deep learning and the need for attention

In recent years, we have seen major advances in the field of artificial intelligence and machine learning. The combination of deep neural networks with the computational capabilities of Graphics Processing Units (GPUs) [123] has improved the performance of several tasks such as image recognition, machine translation, language modelling, time series prediction, game playing and more [69, 105, 100, 45]. Deep learning models have evolved to take into account the computational structure of the problem to be resolved.

In a feedforward neural network (FNN) composed of multiple layers, the output (without the bias term) at layer l, see Figure 5.1, is defined as

$$\boldsymbol{x}^{l+1} = f(W^l \boldsymbol{x}^l), \tag{5.1}$$

W^l being the weight matrix at layer l. f is the activation function and $\boldsymbol{x}^{l+1}$, the output vector at layer l and the input vector at layer $l+1$. The weight matrices for the different layers are the parameters of the model.

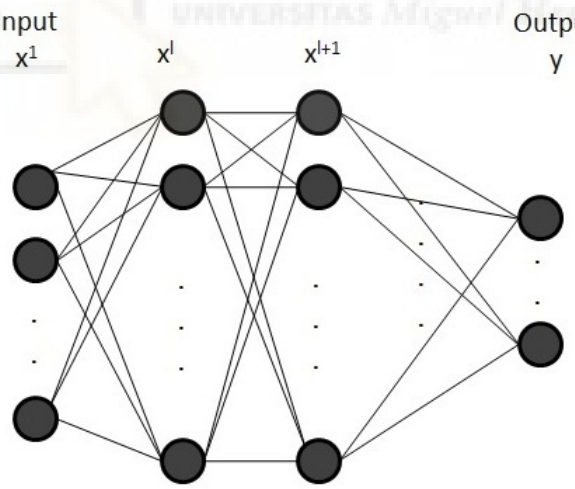

Figure 5.1: Multilayer neural network.

Learning is the mechanism by which the parameters of a neural network are adapted to the environment in the training process. This is an optimization prob-

lem that has been addressed using gradient-based methods, in which given a cost function $f : \mathbb{R}^n \to \mathbb{R}$, the algorithm finds local minima $w^* = \arg\min_w f(w)$ by updating each layer parameter w_{ij} with the rule $w_{ij} := w_{ij} - \eta \nabla_{w_{ij}} f(w)$, where $\eta > 0$ is the learning rate.

Therefore, a deep learning model consists of the forward pass, in which the computational graph with the multiple layers is built, and the backward pass, in which the gradients are calculated and the parameters are updated. Then, all the functions of the parameters used in the model must be differentiable.

RNNs (see Figure 5.2) are a basic component of modern deep learning architectures, especially of encoder-decoder networks. The following equations define the time evolution of an RNN:

$$\boldsymbol{h}_t = f^h(W^{ih}\boldsymbol{x}_t + W^{hh}\boldsymbol{h}_{t-1}), \tag{5.2}$$

$$\boldsymbol{y}_t = f^o(W^{ho}\boldsymbol{h}_t), \tag{5.3}$$

where W^{ih}, W^{hh} and W^{ho} are weight matrices. f^h and f^o are the hidden and output activation functions while $\boldsymbol{x}_t$, $\boldsymbol{h}_t$ and $\boldsymbol{y}_t$ are the network input, hidden state and output.

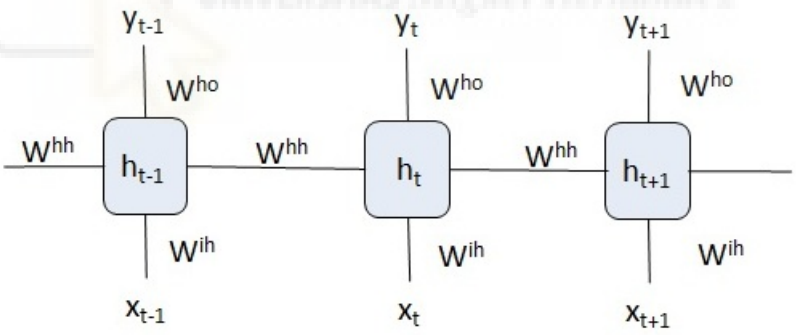

Figure 5.2: Temporal structure of a recurrent neural network.

LSTMs [53] are an evolution of RNNs in that they feature an RNN structure with gated units, i.e. regulators. Specifically, LSTMs are composed of a memory cell, an input gate, an output gate and a forget gate, and allow gradients to flow unchanged. The memory cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

Here we refer to systems that contain a very large number of variables interacting with each other in non-trivial ways as complex systems [50]. Their behaviour is intrinsically difficult to model due to the dependencies and interactions between their parts and they have emergence properties arising from those interactions such as adaptation, evolution, learning, etc. In Section 5.4 we describe the use of attention mechanisms to model the sequential data generated by complex systems.

Dynamical systems are a special class of complex systems. At any given time, a dynamical system has a state that can be represented by a point in a state space (manifold). The evolution equations of the dynamical system describes what future states follow from the current state. This process can be deterministic, if its entire future is uniquely determined by its current state, or non-deterministic otherwise [68] (e.g., a random dynamical system [8]). Furthermore, it can be a continuous-time process, represented by differential equations or a discrete-time process, represented by difference equations or maps. Thus,

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}; \boldsymbol{\theta}) \tag{5.4}$$

for *autonomous* discrete-time deterministic dynamical systems with parameters $\boldsymbol{\theta}$, and

$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{5.5}$$

for *non-autonomous* discrete-time deterministic dynamical systems driven by an external input $\boldsymbol{x}_t$. Dynamical systems with multiple time lags can be rewritten as a higher dimensional dynamical system with time lag 1.

A key aspect in modelling dynamical systems is, of course, temporal dependence. Traditionally, there have been two ways to implement it in the neural network paradigm [66]:

i Classic feedforward neural networks with time delayed states in the inputs but perhaps with an unnecessary increase in the number of parameters.

ii RNNs since, as shown in Equations (5.2) and (5.3), they have a temporal recurrence that makes them appropriate for modelling discrete dynamical systems of the form given in Equations (5.4) and (5.5). As said in the Introduction, RNNs were precisely designed for sequence modelling. [22].

Therefore, RNNs seem the ideal candidates to model, analyze and predict dynamical systems and more generally complex systems. Theoretically, the temporal recurrence of RNNs allows to model and identify dynamical systems described with equations with any temporal dependence.

To learn chaotic dynamics, recurrent radial basis function (RBF) networks [82] and evolutionary algorithms that generate RNNs have been proposed [96]. "Nonlinear Autoregressive model with exogenous input" (NARX) [33] and boosted RNNs [9] have been applied to predict chaotic time series.

LSTMs have also succeeded in various applications to complex systems such as model identification and time series prediction [117, 116, 71]. Another remarkable application of the LSTM is machine translation [105, 27].

Although the classic models above work well, they have limitations that make it difficult to perform sequential reasoning and achieve more general intelligence [74]. Among these limitations, we highlight the following:

i Classic models only perform perception, representing a mapping between inputs and outputs.

ii Classic models follow a hybrid model where synaptic weights perform both processing and memory tasks but do not have an explicit external memory.

iii Classic models do not carry out sequential reasoning. This essential process is based on perception and memory through attention and guides the steps of the machine learning model in a conscious and interpretable way.

In the next section, we present attention mechanisms as an important step to address these limitations.

5.3 Attention mechanisms

5.3.1 Differentiable attention

As explained in Section 5.2, classic deep learning models do not perform sequential reasoning, a process that is based on attention.

In the brain, reasoning is the process of establishing and verifying facts combining attention with new or existing information. The role of the attention mechanisms is to focus on one part of the input or memory (image, text, etc), thus guiding the process of reasoning.

As described in [72], there are several classes of attention in neuroscience: attention as a level of alertness, attention over sensory inputs, attention to select and execute tasks and attention for memory encoding and retrieval. In [31], the authors modeled the interaction between top-down attention and bottom-up stimulus contrast effects and found that external attention inputs bias neurons to move to different parts of their nonlinear activation functions. Insects have been a source of inspiration in intelligence and attention mechanisms. In [61], a multiclass support vector machine with inhibition is inspired by the brain structure of insects. In [7], a multi-layer spiking neural network is presented that models the Mushroom Bodies and their interactions to other key elements of the insect brain, the Central Complex and the Lateral Horns.

Analogously, a learning problem in machine learning can be decomposed into a sequence of tasks, where in each task it is necessary to focus on one part of an input (or transformed input) or a memory. Once again, neural information processing in the brain, in which several layers interact with each other (Chapter 3), has been a source of inspiration for machine learning.

Generally formulated, attention in machine learning is a sequential process in which a learning task is guided by a set of elements of the input source (or memory). This is achieved by integrating the attention value into the task.

Attention mechanisms have provided and will provide a paradigm shift in machine learning. Specifically, this change is from traditional large-scale vector transformations to more conscious processes (i.e., that focus only on a set of elements), e.g. decomposing a problem into a sequence of attention based reasoning tasks [60, 4, 42, 62, 52, 122].

As stated in Section 5.2, to integrate a component into a deep learning model that learns using gradient descent, all the functions of the parameters in the component must be differentiable. One way to make attention mechanisms differentiable is to formulate them as a convex combination of the input or memory. In this case, all the steps are differentiable and can be learned, and the combination weights must add up to one (forcing them to focus on some parts more than others). In this way, the mechanism learns which parts it needs to focus on.

As in [112], this convex combination, shown in Figure 5.3, is described as mapping a query and a set of key-value pairs to an output:

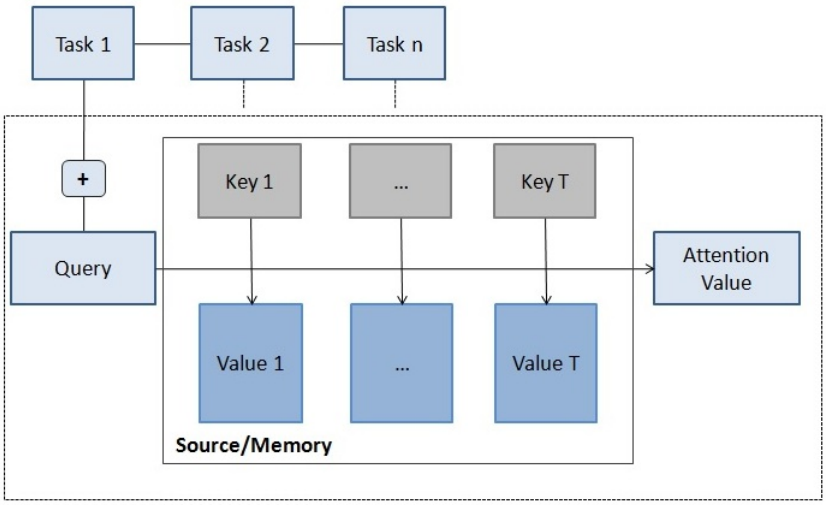

Figure 5.3: Attention diagram. Attention as a sequential process of reasoning in which the task (query) is guided by a set of elements (values) of the source (or memory).

$$att(\boldsymbol{q}, \boldsymbol{s}) = \sum_{i=1}^{T} \alpha_i(\boldsymbol{q}, \boldsymbol{k}_i)\boldsymbol{V}_i, \tag{5.6}$$

where, as seen in Figure 5.3, $\boldsymbol{k}_i$ and $\boldsymbol{V}_i$ are the key and the value vectors from the source/memory $\boldsymbol{s}$, and $\boldsymbol{q}$ is the query vector (task). $\alpha_i(\boldsymbol{q}, \boldsymbol{k}_i)$ is the similarity function between the query and the corresponding key and is calculated by applying the softmax function,

$$\text{softmax}(z_i) = \frac{exp(z_i)}{\sum_{i'} exp(z_{i'})}, \tag{5.7}$$

to the score function $score(\boldsymbol{q}, \boldsymbol{k}_i)$:

$$\alpha_i = \frac{\exp(score(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{i'=1}^{T} \exp(score(\boldsymbol{q}, \boldsymbol{k}_{i'}))}. \tag{5.8}$$

The score function can be computed using a feedforward neural network:

$$score(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{Z}_a \tanh(\boldsymbol{W}_a[\boldsymbol{q}, \boldsymbol{k}_i])), \tag{5.9}$$

as proposed in [10], where $\boldsymbol{Z}_a$ and $\boldsymbol{W}_a$ are matrices to be jointly learned with the rest of the model and $[\boldsymbol{q}, \boldsymbol{k}_i]$ is a linear function or concatenation of $\boldsymbol{q}$ and $\boldsymbol{k}_i$. Also, in [48] the authors use a cosine similarity measure for content-based attention, namely,

$$score(\boldsymbol{q}, \boldsymbol{k}_i) = \cos((\boldsymbol{q}, \boldsymbol{k}_i)), \tag{5.10}$$

where $((\boldsymbol{q}, \boldsymbol{k}_i))$ denotes the angle between $\boldsymbol{q}$ and $\boldsymbol{k}_i$.

Then, attention can be seen as a sequential process of reasoning in which the task (query) is guided by a set of elements of the input source (or memory) using attention.

The attention process can focus on:

1. Temporal dimensions, e.g. different time steps of a sequence.
2. Spatial dimensions, e.g. different regions of an image.
3. Different elements of a memory.
4. Different features or dimensions of an input vector, etc.

Depending on where the process is initiated, we have:

i Top-down attention, initiated by the current task.

ii Bottom-up, initiated spontaneously by the source or memory.

To apply the attention mechanism, it is necessary to break down the learning process into a sequence of attention-guided tasks.

Then, due to its flexibility, an attention mechanism can be added in multiple ways to any deep learning architecture that models a complex system. In Section 5.4 we illustrate this flexibility as follows:

1. Through a conventional attention (the query is different from the key and the value) in Section 5.4.2, with the encoder selecting input features and the decoder selecting time steps.
2. Through a memory network in which a memory of historical data guides

the current prediction task in Section 5.4.3.

3. Through self-attention (the keys, values and queries come from the same source) in Section 5.4.4. Here, to encode a vector of the input sequence, self-attention allows the model to focus in a direct way on other vectors in the sequence.

5.3.2 Attention in seq2seq models

An encoder-decoder model maps an input sequence to a target one with both sequences of arbitrary length [105]. They have applications ranging from machine translation to time series prediction.

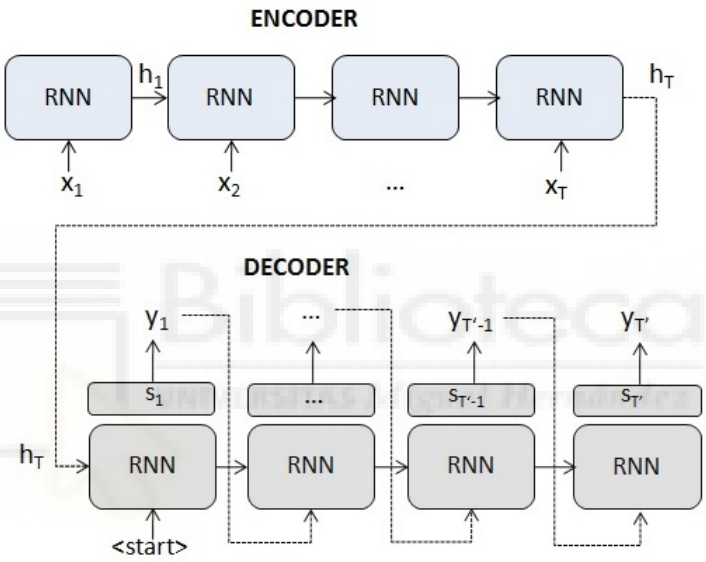

Figure 5.4: An encoder-decoder network.

More specifically, this mechanism uses an RNN (or any of its variants such as an LSTM or a GRU, Gated Recurrent Unit) to map the input sequence to a fixed-length vector, and another RNN (or any of its variants) to decode the target sequence from that vector (see Figure 5.4). Such a seq2seq model typically features an architecture composed of:

1. An encoder which, given an input sequence $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$ with $\boldsymbol{x}_t \in \mathbb{R}^n$, maps $\boldsymbol{x}_t$ to

$$\boldsymbol{h}_t = f_1(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t), \tag{5.11}$$

where $\boldsymbol{h}_t \in \mathbb{R}^m$ is the hidden state of the encoder at time t, m is the size of the hidden state and f_1 is an RNN (or any of its variants).

2. A decoder, where $\boldsymbol{s}_t$ is the hidden state and whose initial state $\boldsymbol{s}_0$ is initialized with the last hidden state of the encoder $\boldsymbol{h}_T$. It generates the output sequence $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_{T'})$, $\boldsymbol{y}_t \in \mathbb{R}^o$ (the dimension o depending on the task), where

$$\boldsymbol{y}_t = f_2(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}), \tag{5.12}$$

and f_2 is an RNN (or any of its variants) with an additional layer depending on the task (e.g. a linear layer for series prediction or a softmax layer for translation).

Because the encoder compresses all the information of the input sequence in a fixed-length vector (the final hidden state $\boldsymbol{h}_T$), the decoder possibly does not take into account the first elements of the input sequence. The use of this fixed-length vector is a limitation to improve the performance of the encoder-decoder networks. Moreover, the performance of encoder-decoder networks degrades rapidly as the length of the input sequence increases [26]. This occurs in applications such as machine translation and time series prediction, where it is necessary to model long time dependencies.

The key to solve this problem is to use an attention mechanism to guide the decoding task. In [10] an extension of the basic encoder-decoder architecture was proposed by allowing the model to automatically search and learn which parts of a source sequence are relevant to predict the target element. Instead of encoding the input sequence in a fixed-length vector, it generates a sequence of vectors, choosing the most appropriate subset of these vectors during the decoding process.

Equipped with the attention mechanism, the encoder is a bidirectional RNN [47] with a forward hidden state $\overrightarrow{\boldsymbol{h}_i} = f_1(\overrightarrow{\boldsymbol{h}}_{i-1}, \boldsymbol{x}_i)$ and a backward one $\overleftarrow{\boldsymbol{h}_i} = f_1(\overleftarrow{\boldsymbol{h}}_{i+1}, \boldsymbol{x}_i)$. The encoder state is represented as a simple concatenation of the two states,

$$\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}_i}; \overleftarrow{\boldsymbol{h}_i}], \tag{5.13}$$

with $i = 1, ..., T$. The encoder state includes both the preceding and following

elements of the sequence, thus capturing information from neighbouring inputs.

The decoder has an output

$$\boldsymbol{y}_t = f_2(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t) \tag{5.14}$$

for $t = 1, ..., \boldsymbol{T}'$. f_2 is an RNN with an additional layer depending on the task (e.g. a linear layer for series prediction or a softmax layer for translation), and the input is a concatenation of $\boldsymbol{y}_{t-1}$ with the context vector $\boldsymbol{c}_t$, which is a sum of hidden states of the input sequence weighted by alignment scores:

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_{ti} \boldsymbol{h}_i. \tag{5.15}$$

Similar to Equation (5.8), the weight α_{ti} of each state $\boldsymbol{h}_i$ is calculated by

$$\alpha_{ti} = \frac{\exp(score(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{T} \exp(score(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))}. \tag{5.16}$$

In this attention mechanism, the query is the state $\boldsymbol{s}_{t-1}$ and the key and the value are the hidden states $\boldsymbol{h}_i$. The score measures how well the input at position i and the output at position t match. α_{ti} are the weights that implement the attention mechanism, defining how much of each input hidden state should be considered when deciding the next state $\boldsymbol{s}_t$ and generating the output $\boldsymbol{y}_t$ (see Figure 5.5).

As we have described previously, the score function can be parametrized using different alignment models such as feedforward networks and the cosine similarity.

An example of a matrix of alignment scores is shown in Figure 5.6. This matrix provides interpretability to the model since it allows to know which part (time-step) of the input is more important to the output.

The attention mechanism then transforms an encoder-decoder sequential model into a non-sequential model in which the attention mechanism guides the decoding task based on the encoded states.

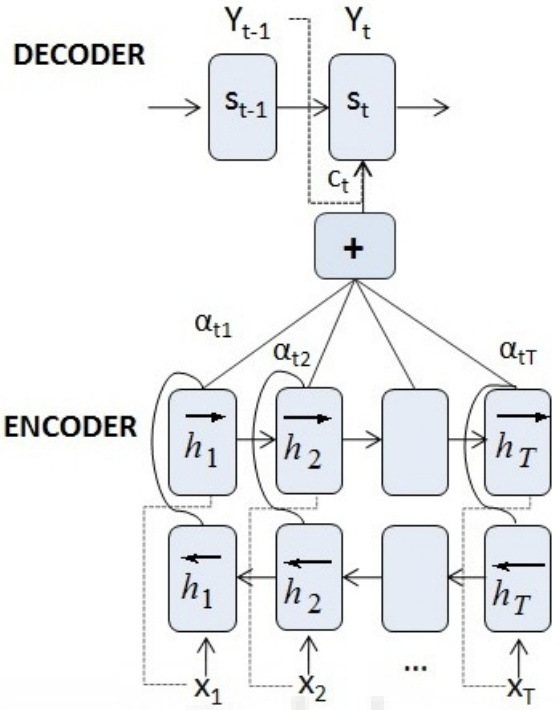

Figure 5.5: An encoder-decoder network with attention.

5.3.3 Self-attention and memory networks

A variant of the attention mechanism is self-attention, in which the attention component relates different positions of a single sequence in order to compute a representation of the sequence. In this way, the keys, values and queries come from the same source. The mechanism can connect distant elements of the sequence more directly than using RNNs [106].

Similar to the description given in [112], for an input sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$, the self-attention process can be implemented by the following steps:

1. For each of the input vectors, create a query $\boldsymbol{Q}_t$, a key $\boldsymbol{K}_t$ and a value vector $\boldsymbol{V}_t$ by multiplying the input vector $\boldsymbol{x}_t$ by three matrices that are trained during the learning process, $\boldsymbol{W}_i^Q \in \mathbb{R}^{d \times d_k}, \boldsymbol{W}_i^K \in \mathbb{R}^{d \times d_k}$ and $\boldsymbol{W}_i^V \in \mathbb{R}^{d \times d_v}$.
2. For each query vector $\boldsymbol{Q}_t$, the self-attention value is computed by mapping the query and all the key-values to an output, $\boldsymbol{Attention}(\boldsymbol{Q}_t, \boldsymbol{K}, \boldsymbol{V}) = \sum_{j=1}^{T} \boldsymbol{\alpha}_j(\boldsymbol{Q}_t, \boldsymbol{K}_j)\boldsymbol{V}_j$, where

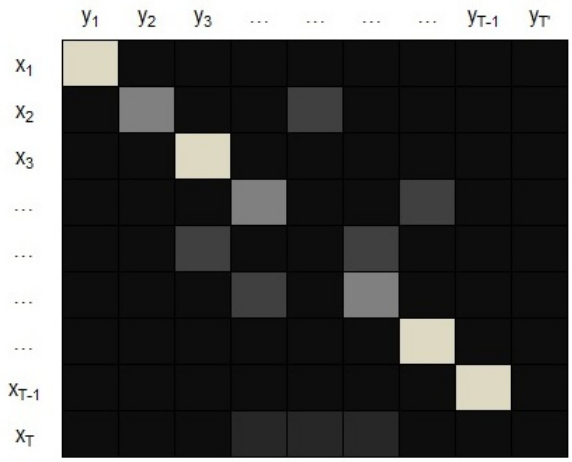

Figure 5.6: A matrix of alignment scores. It represents how much of each input state should be considered when deciding the next state and generating the output.

$$\alpha_j(\boldsymbol{Q}_t, \boldsymbol{K}_j) = \text{softmax}\left(\frac{\boldsymbol{Q}_t \boldsymbol{K}_j^T}{\sqrt{d_k}}\right) \tag{5.17}$$

3. This self-attention process is performed h times in what is called multi-headed attention. Each time, the input vectors are projected into a different query, key and value vector using different matrices $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K$ and $\boldsymbol{W}_i^V$ for $i = 1, ..., h$. On each of these projected queries, keys and values, the attention function is performed in parallel, producing d_v dimensional output values, that are concatenated and once again projected to the final values. This multi-headed attention process allows the model to focus on different positions from different representation subspaces.

When the model is processing a vector of the input sequence, single self-attention allows the model to focus on other vectors in the sequence to get a better representation of this vector. With multi-headed self-attention (see Figure 5.7), each attention head is focusing on a different set of vectors when processing the vector.

The Transformer [112], a network architecture based only on self-attention, is composed of an encoder and a decoder:

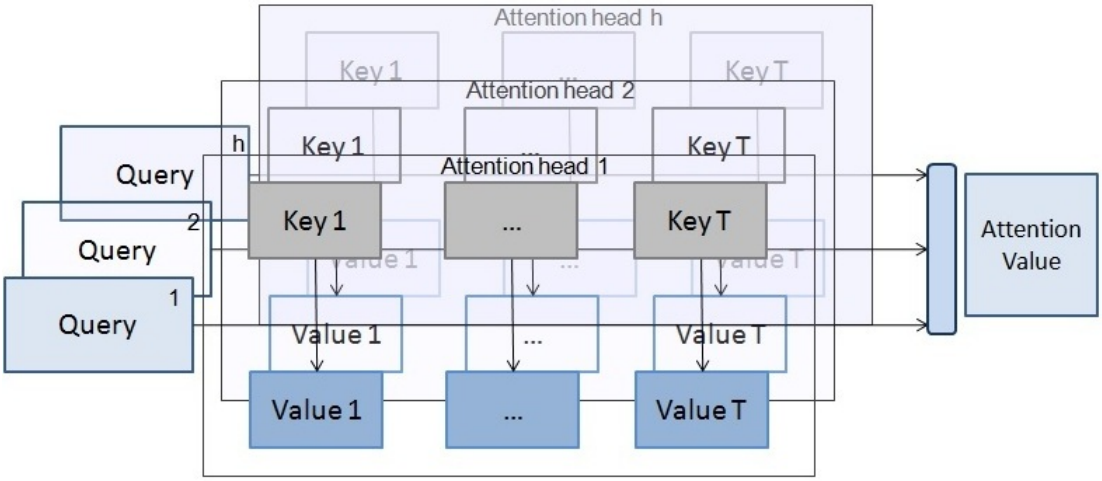

Figure 5.7: Multi-headed attention. Self-attention process performed in parallel h times in different subspaces. The output values are concatenated and projected to a final value

1. Encoder: Composed of a stack of six identical layers, each layer with a multi-head self-attention process and a position-wise fully connected feed-forward network. Around each of the sub-layers, a residual connections followed by layer normalization is employed.
2. Decoder: Is also composed of a stack of six identical layers (with self-attention and a feed-forward network) with an additional third sub-layer to perform attention over the output of the encoder (as in the seq2seq with attention). The self-attention sub-layer is modified to prevent a vector from attending to subsequent vectors in the sequence.

The Transformer allows to replace CNNs and RNNs, improving machine translation tasks while using less training time. The Transformer is also the basic component of GPT-3 (Generative Pre-Trained Transformer-3), a pre-trained language model which achieves good performance in few-shot learning on many Natural Language Processing tasks without fine-tuning [20].

Another variant of attention are end-to-end memory networks [103], which are neural networks with a recurrent attention model over an external memory. The model, trained end-to-end, is described in more detail in Section 5.4.3 and outputs an answer based on a query and a set of inputs $x_1, x_2, ..., x_n$ stored in a

memory.

5.4 Attention mechanisms in complex systems

5.4.1 Where and how to apply attention

In the previous sections we have described various attention mechanisms. These mechanisms allow a task to focus on a set of elements of an input sequence, an intermediate sequence or a memory source.

Due to its flexibility, an attention mechanism can be added to any deep learning model in multiple ways. Therefore, when applying it to model complex systems, it will be necessary to decide the following issues:

i In which part of the model should be introduced?

ii What elements of the model will the attention mechanism relate?

iii What dimension (temporal, spatial, input dimension, etc.) is the mechanism going to focus on?

iv Will self-attention or conventional attention be used?

v What elements will correspond to the query, the key and the value?

In the following sections we describe some illustrative cases of application of attention mechanisms to model complex systems. As we will see, how the information is processed in the system and how the different elements are related will be key when defining the aforementioned issues.

5.4.2 Attention in different phases of a model

In a non-autonomous dynamical system, the current state is a transformation of the previous states and the current input, which contains n dimensions or features. More generally, the dependencies between time steps can be dynamic, i.e., time-changing. In such complex systems, attention mechanisms learn to focus on the most relevant parts of the system input or state.

A representative attention mechanism in this context implements a dual-stage attention, namely, an encoder with input features attention and a decoder with temporal attention, as pointed out in [90]. Next we describe this architecture, in which the first stage extracts the relevant input features and the second selects the relevant time steps of the model.

Let $X = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T)$ with $\boldsymbol{x}_t \in \mathbb{R}^n$ be the input sequence. T is the length of the time interval and n the number of input features or dimensions. $\boldsymbol{x}_t = (x_t^1, x_t^2, ..., x_t^n)$ is the input at the time step t and $\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_T^k)$ is the k input feature series.

Encoder with input attention

Given an input sequence X, the encoder maps $\boldsymbol{u}_t$ to

$$\boldsymbol{h}_t = f_1(\boldsymbol{h}_{t-1}, \boldsymbol{u}_t), \tag{5.18}$$

where $\boldsymbol{h}_t \in \mathbb{R}^m$ is the hidden state of the encoder at time t, m is the size of the hidden state and f_1 is an RNN (or any of its variants). $\boldsymbol{x}_t$ is replaced by $\boldsymbol{u}_t$, which adaptively selects the relevant input features as follows:

$$\boldsymbol{u}_t = (\alpha_t^1 x_t^1, \alpha_t^2 x_t^2, ..., \alpha_t^n x_t^n). \tag{5.19}$$

Here

$$\alpha_t^k = \frac{\exp(score(\boldsymbol{h}_{t-1}, \boldsymbol{x}^k))}{\sum_{i=1}^n \exp(score(\boldsymbol{h}_{t-1}, \boldsymbol{x}^i))}, \tag{5.20}$$

is the attention weight measuring the importance of the k input feature at time t, where $\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_T^k)$ is the k input feature series and the score function can be computed using a feedforward neural network, a cosine similarity measure or other similarity functions.

Therefore, this first attention stage extracts the relevant input features with the query, keys and values shown in Figure 5.8.

Decoder with temporal attention

Similar to the attention decoder described in Section 5.3.2, the decoder has an output

$$\boldsymbol{y}_t = f_2(\boldsymbol{s}_{t-1}, \boldsymbol{y}_{t-1}, \boldsymbol{c}_t) \tag{5.21}$$

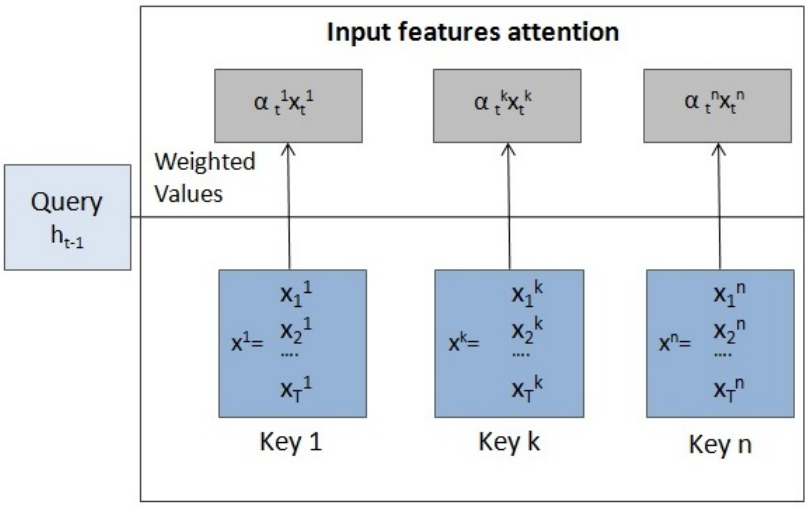

Figure 5.8: Diagram of the input features attention mechanism.

for $t = 1, ..., T'$. f_2 is an RNN (or any of its variants) with an additional linear or softmax layer, and the input is a concatenation of $\boldsymbol{y}_{t-1}$ with the context vector $\boldsymbol{c}_t$, which is a sum of hidden states of the input sequence weighted by alignment scores:

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \beta_t^i \boldsymbol{h}_i. \tag{5.22}$$

The weight β_t^i of each state $\boldsymbol{h}_i$ is computed using the similarity function, $score(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i)$, and applying a softmax function, as described in Section 5.3.2.

This second attention stage selects the relevant time steps, as shown in Figure 5.9 with the corresponding query, keys and values.

Further remarks

In [90], the authors define this dual-stage attention RNN and show that the model outperforms a classical model in time series prediction.

A comparison is made between LSTMs and attention mechanisms for financial time series forecasting in [57]. It is shown that an LSTM with attention performs better than stand-alone LSTMs.

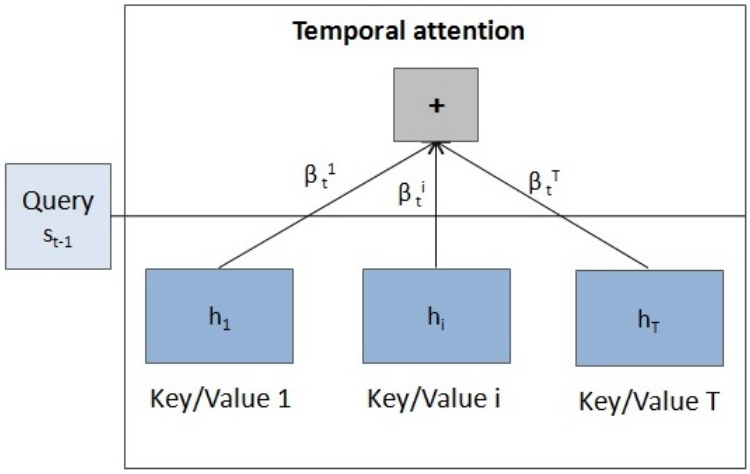

Figure 5.9: Diagram of the temporal attention mechanism.

A temporal attention layer is used in [113] to select relevant information and to provide model interpretability, an essential feature to understand deep learning models. In [99], interpretability is further studied in detail, concluding that attention weights partially reflect the impact of the input elements on model prediction.

5.4.3 Memory networks

Memory networks allow long-term or external dependencies in sequential data to be learned thanks to an external memory component. Instead of taking into account only the most recent states, memory networks also consider the entire list of states or the states of a memory.

Here we define one possible application of memory networks to complex systems, following an approach based on [103]. We are given a time series of historical data $\boldsymbol{n}_1, ..., \boldsymbol{n}_{T'}$ with $\boldsymbol{n}_i \in \mathbb{R}^n$ and the input series $\boldsymbol{x}_1, ..., \boldsymbol{x}_T$ with $\boldsymbol{x}_t \in \mathbb{R}^n$ the current input, which is the query in the attenton mechanism.

The set $\{\boldsymbol{n}_i\}$ are converted into memory vectors $\{\boldsymbol{m}_i\}$ and output vectors $\{\boldsymbol{c}_i\}$ of dimension d. The query $\boldsymbol{x}_t$ is also transformed to obtain a internal state $\boldsymbol{u}_t$ of dimension d. These transformations correspond to a linear transformation: $\boldsymbol{A}\boldsymbol{n}_i = \boldsymbol{m}_i, \boldsymbol{B}\boldsymbol{n}_i = \boldsymbol{c}_i, \boldsymbol{C}\boldsymbol{x}_t = \boldsymbol{u}_t$, where $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ are parameterizable matrices.

A match between $\boldsymbol{u}_t$ and each memory vector $\boldsymbol{m}_i$ is computed by taking the inner product followed by a softmax function:

$$p_t^i = Softmax(\boldsymbol{u}_t^T \boldsymbol{m}_i). \tag{5.23}$$

The final vector from the memory, $\boldsymbol{o}_t$, is a weighted sum over the transformed memory inputs $\{\boldsymbol{c}_i\}$:

$$\boldsymbol{o}_t = \sum_i p_t^i \boldsymbol{c}_i. \tag{5.24}$$

To generate the final prediction $\boldsymbol{y}_t$, a linear layer is applied to the sum of the output vector $\boldsymbol{o}_t$ and the transformed input $\boldsymbol{u}_t$, and to the previous output $\boldsymbol{y}_{t-1}$:

$$\boldsymbol{y}_t = f(W^1(\boldsymbol{o}_t + \boldsymbol{u}_t) + W^2 \boldsymbol{y}_{t-1}) \tag{5.25}$$

A basic diagram of the model is shown in Figure 5.10. This model is differentiable end-to-end by learning the matrices (the final matrices W^i(W^1 and W^2) and the three transformation matrices $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$) to minimize the prediction error.

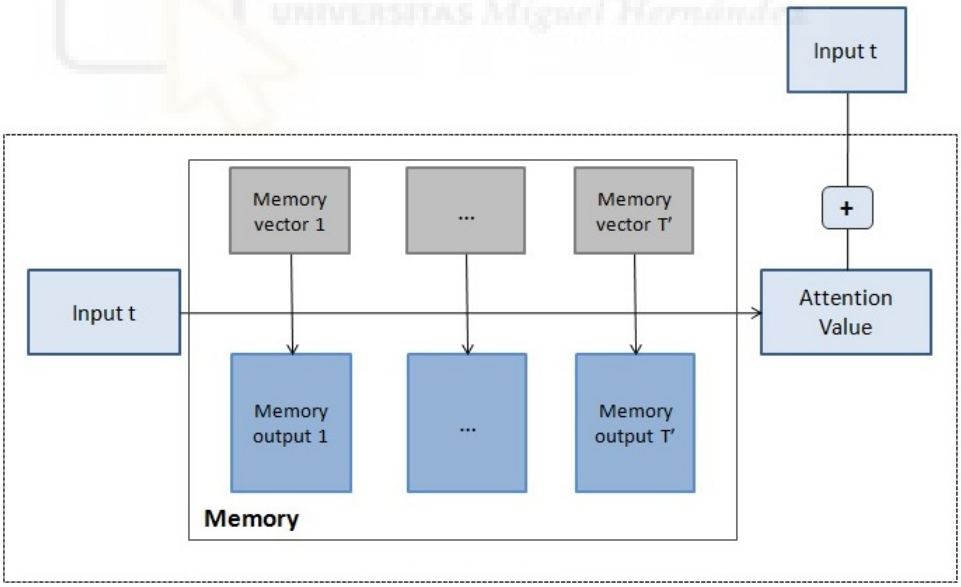

Figure 5.10: Basic diagram of a memory network. For each input, the attention mechanism integrates a weighted sum over the memory vectors.

In [24] the authors propose a similar model based on memory networks with a memory component, three encoders and an autoregressive component for multivariate time-series forecasting. Compared to non-memory RNN models, their model is better at modeling and capturing long-term dependencies and, moreover, it is interpretable.

Differentiable Neural Computers (DNCs) [49] consist of a neural network that uses attention and can read from, and write to, an external memory. Taking advantage of these capabilities, an enhanced DNC for electroencephalogram (EEG) data analysis is proposed in [81]. By replacing the LSTM network controller with a recurrent convolutional network, the potential of DNCs in EEG signal processing is convincingly demonstrated.

5.4.4 Self-attention

An important aspect to model complex systems is to capture the temporal dependence and the relationship between the parts that make up the system.

If we compare the computational graph of an RNN (see Figure 5.2) with the graph of an attention module (Figure 5.11), we observe that even adding a memory unit (LSTM), the attention module relates each of the inputs in a more direct and symmetric way to form the output vector.

The distance, in number of edges in the graph, between an input and an output distant in time, is shorter and is the same for all input vectors in the self-attention module. However, this is at the cost of not prioritizing local interactions, which has a high computational cost for very long sequences.

The Transformer, as we have pointed out, is composed of a stack of multi-headed self-attention components. With multi-headed attention, the input vectors are projected into a different query, key and value vector, performing the self-attention process $\boldsymbol{h}$ times. When processing a vector, each attention head is focusing on a different set of vectors from different representation subspaces.

These mentioned characteristics make self-attention and the Transformer a promising building block in deep learning models for complex systems.

In [59] the authors propose a dual self-attention network for multivariate time (dynamic-period or non-periodic) series forecasting. In [101] the authors utilize attention models for clinical time-series modeling. They employ a masked self-attention mechanism and use positional encoding and dense interpolation for

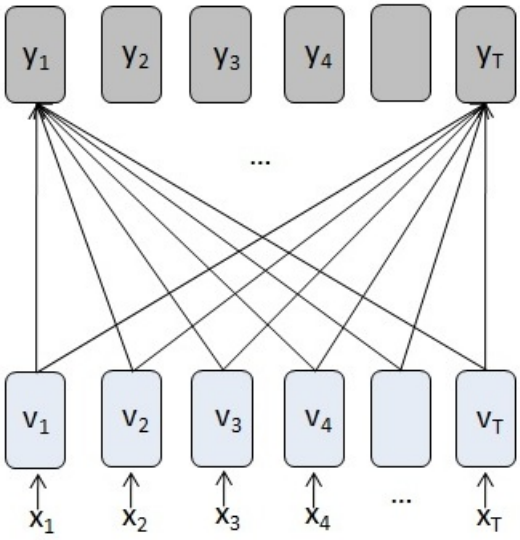

Figure 5.11: Self-attention graph. The self-attention component calculates how much each input vector contributes to form each output vector.

incorporating temporal order.

Further understanding of the Transformer architecture is carried out in [73], where the authors show that the Transformer architecture can be interpreted as a numerical ODE (Ordinary Differential Equation) solver for a convection-diffusion equation in a multi-particle dynamic system. They interpret how words (vectors) in a sentence are abstracted by passing through the layers of the Transformer as approximating the movement of multiple particles in the space using the Lie-Trotter splitting scheme and the Euler's method.

5.5 Discussion

After the success of recent years, one of the most important challenges that deep learning faces is to improve input-output models, adopting new primitives that provide reasoning, abstraction, search and memory capabilities.

Similar to what happens in the brain, attention mechanisms allow the reasoning or cognitive process to be guided in a flexible way. This improvement is important when modeling complex systems due to their temporal dependence and complex relationships.

As we have seen, attention mechanisms have the following benefits in modeling such systems:

i By focusing on a subset of elements, it guides the reasoning or cognitive process.

ii These elements can be tensors (vectors) from the input layer, from the intermediate layer or be external to the model, e.g. a external memory.

iii It can focus on temporal dimensions (different time steps of a sequence), spatial dimensions (different regions of space) or different features of an input vector.

iv It can relate each of the input vectors in a more direct and symmetric way to form the output vector.

More specifically, as shown in Table 5.2, for each of the techniques and applications described, we discuss its application potential, characteristics and advantages.

1. *One stage conventional attention.* The attention mechanism allows guiding any complex system task such as modeling, prediction, identification, etc. To do this, it focuses on a set of elements from the input layer or from an intermediate layer. These elements can be temporal, spatial or feature dimensions. For example, to model a dynamical system with an input of dimension n, one can add an attention mechanism to focus and integrate the different input dimensions. The attention mechanism is combined, as we have seen, with an RNN or an LSTM and allows modeling long temporal dependencies. This technique, like the rest, adds complexity to the model. To calculate the attention weights between a task (query) of T elements and an attended region (key, value) of T elements, it is necessary to perform T^2 multiplications.
2. *Several stages conventional attention.* This case is similar to the previous, one-stage conventional attention but with several attention phases or stages. The attention mechanism is also combined with an RNN or an LSTM and allows modeling long temporal dependencies. As we have seen, the model can focus on a set of feature elements from the input layer and on a set of temporal steps from an intermediate layer. This enables multi-step reasoning. The downside is that more computational cost is added to the model with T^2 multiplications for each attention stage.

3. *Memory networks.* In memory networks, any complex system task such as modeling, prediction or identification is guided by an external memory. Then, memory networks allow long-term or external dependencies in sequential data to be learned thanks to an external memory component. Instead of taking into account only the most recent states, these networks consider the states of a memory or external data as well. Such is the case of time series prediction also based on an external source that can influence the series. To calculate the attention weights between a task (query) of $\boldsymbol{T}$ elements and an attended memory of $\boldsymbol{T'}$ elements, it is necessary to perform $\boldsymbol{TT'}$ multiplications.
4. *Self-attention.* In self-attention, the component relates different positions of a single sequence in order to compute a transformation of the sequence. The keys, values and queries come from the same source. It is a generalization of neural networks, since they perform a direct transformation of the input but the weights are dynamically calculated. The attention module relates each of the inputs in a more direct way to form the output vector but at the cost of not prioritizing local interactions. Their use case is general since they can replace neural networks, RNNs or even CNNs. To calculate the attention weights for a sequence of $\boldsymbol{T}$ elements it is necessary to perform $\boldsymbol{T}^2$ multiplications.
5. *Combination of the above techniques.* It is interesting to combine several of the previous techniques but at the cost of increasing the complexity and adding the computational cost of each of the components. For example, the Transformer, which can be used in a multitude of tasks such as sequence modeling, generative models, predictions, machine translation, multi-tasking, etc. The Transformer combines self-attention with conventional attention. In the encoder, the Transformer has a stack of self-attention blocks. The decoder also has self-attention blocks and an additional layer to perform attention over the output of the encoder.

Techniques	Operation	Use cases	Costs	Applications
One-stage att.	Over the input or an intermediate layer Temporal, spatial...	Integrate parts Long term dependencies	Complexity $\boldsymbol{T}^2$ operations	Modeling Prediction
Several stages	Over the input, over an intermediate layer Temporal, spatial...	Integrate several parts Long term dependencies Multi-step reasoning	Complexity $\boldsymbol{T}^2$ operations each att. stage	Modeling Prediction Sequential reasoning
Memory networks	Over external data Temporal, spatial...	Integrate a memory	Complexity $\boldsymbol{TT'}$ operations	Modeling Reasoning over a memory
Self-attention	Relate elements of the same sequence	General Encode an input	Complexity Non local $\boldsymbol{T}^2$ operations	Replace neural networks
Combination	Combine the above elements	All of the above	Sum of the costs	All of the above

Table 5.2: Characteristics of attention techniques in complex systems described in this chapter.

However, despite the theoretical advantages and some achievements, further studies are needed to verify the benefits of the attention mechanisms over traditional networks in complex systems.

6 Learning strategies using Large Language Models and the generator-verifier trade-off

6.1 Introduction

Recent years have seen great progress in artificial intelligence based on deep learning models and the use of GPUs (Graphics Processing Units) to train large amounts of data [123, 69, 45]. These differentiable and deep learning models learn the transformations of the input that minimize the error to predict the training data, making it difficult to recombine previous functional modules to extract new information. This is the great challenge of current artificial intelligence and the main point of criticism [74, 18, 89].

The recent development of large language models (LLMs), which are autoregressive models trained in a self-supervised manner on a large corpus [84, 34], have raised the possibility of overcoming these limitations and challenges. The ability of LLMs to generate language in response to requests makes them ideal candidates for planning, reasoning and generalizing.

In this chapter, we propose LLMs to generate learning strategies similar to human learning strategies [67]. These strategies consist of a series of activities that define machine learning models and data, and interact with the environment to generate new knowledge.

In the experiments we evaluate whether LLMs are capable of performing those activities autonomously and without syntactic or semantic errors. Moreover, we are interested in knowing if LLMs have a general ability to correctly define the

activities of a learning strategy or if success depends rather on the type of activity and its presence in the training data.

Finally, we characterize the set of learning activities that LLMs can perform in a correct and autonomous manner, and discuss the generator-verifier trade-off.

6.1.1 Related work and contributions

Below we mention research related to: human reasoning, learning strategies and the limitations of deep learning; approaches to structure the operation of LLMs and improve their reasoning capacity; and results that measure that planning and reasoning capacity.

Thus, in [67] the authors propose two levels of learning: level 1 as innate and automatic and level 2 as learning strategies which are managed by the agent's knowledge. In [74, 18, 89], the limitations and challenges of current deep learning AI are analyzed.

A conceptual framework to systematize diverse methods for LLM-based reasoning, grounding, learning and decision making is proposed in [104]. Different approaches have been recently defined to structure the input and improve the reasoning capacity of LLMs, such as tree of thought [125] and chain of thought [119].

Regarding the reasoning and emergent abilities of LLMs, the authors of [118] conclude that additional scaling could further expand the range of performed tasks but in [98] these abilities appear due to the choice of metric rather than to changes in model behavior with scale. In [110] the authors confirm that LLMs ability to generate executable plans autonomously is rather limited and that external verifiers can help provide feedback for better plan generation in heuristic mode.

Our work focuses on defining learning strategies, both human and artificial, and evaluating whether current LLMs are able of implementing them correctly. The main contributions are the following:

i We define a typical human learning framework that can serve as a foundation for extending current artificial intelligence.

ii We use LLMs to develop learning strategies and emulate Level 2 learning.

iii We conduct experiments to evaluate the capabilities of current LLM to emulate the Level 2 learning and generate learning strategies in a logical

and sequential manner.

iv We discuss and characterize the set of learning activities that LLMs can perform in a correct and autonomous manner.

6.2 Limitation of deep learning and differences with human learning

6.2.1 Short description and limitations of differentiable models

As stated in Chapter 4, in differentiable or deep learning models the learning algorithm takes as input a sequence of training samples and outputs a function $g : \mathcal{X} \to \mathcal{Y} \in \mathcal{H}$, from the hypothesis space of functions, with a low probability of error $P(g(X) \neq Y$. Then, differentiable models learn the transformations of the input that minimize the error to predict the training data.

This is what makes it difficult to recombine previous functional modules to extract new information as humans do. This is also the great challenge of current AI as well as the main point of criticism towards its acceptance [74, 18, 89].

Nevertheless, humans can correctly interpret novel combinations of existing concepts, even if those are extremely unlikely under our training distribution, so long as they respect high-level syntactic and semantic patterns.

6.2.2 Human learning versus differentiable learning

Following our work in Chapter 4, we highlight the following differences and limitations of deep learning compared to human learning; see Table 6.1 for a summary.

1. Differentiable or deep learning models learn the transformations of the input that minimize the error to predict the training data.
2. Differentiable models do not learn certain aspects of the task that are later combined to generate new knowledge. They cannot generate new tasks or information without additional training data.
3. By contrast, humans can correctly interpret novel combinations of existing concepts, even if those combinations have not been seen previously.
4. Differentiable models are only based on input data and are not grounded on other information as humans do.
5. Human learning uses a lot of previous knowledge guided by logic to reason

and generate new information.

6. Differentiable models scale well in data (when we increase the input data) but not in novel tasks, while in humans it is the other way around.

Characteristic	Differentiable programming	Human learning
Functional complexity	Functional approximators	Limited functional complexity
New knowledge	No new knowledge without additional training data	Interpret novel concepts
Previous data	Only uses the model training data and the model input	Combination of previous knowledge to generate new information Large base of acquired knowledge
Scalability	Good data scalability	Good scalability on new tasks
Increasing complexity	Only one model	Increasingly complex models of the world based on simpler models

Table 6.1: Evaluation of the different learning characteristics of differentiable programming and human learning

As can be seen in Table 6.1, the most relevant aspects of human learning are the learning strategies that leverage previous data and models to generate new experiences and knowledge.

6.3 Large Language Models to generate learning strategies

6.3.1 Cognitive learning strategies as the base framework

We define a typical cognitive learning framework that can serve as a foundation for extending deep learning.

Similar to [67], two levels of learning are proposed. Level 1 (L1) is unconscious, involuntary and similar to deep learning. Level 2 (L2) selects and executes the strategies that create the experiences and data for the L1 algorithms to learn; for example, to repeat a phone number to memorize it or to relate two different concepts. L2 strategies are natural to humans and combine knowledge from multiple sources explicitly, generalizing that knowledge.

Usually we associate Level 1 with current deep learning and Level 2 with logical and sequential reasoning [63].

In cognitive science, researchers analyze mental architectures that are models of the overall organization of the mind and how information is actually processed

in the different components of the architecture [17]. These architectures can be hybrid and combine symbolic or logical modules (Level 2) with connectionist networks (Level 1).

Learning strategies are an important part of these mental architectures and involve various cognitive functions such as reasoning, imagination, planning, etc.

We define the following framework to learn intuitively:

1. Generate learning strategies using a knowledge base. The knowledge base is a pre-acquired capability in humans or a LLM in artificial intelligence.
2. Apply learning strategies to create the experiences, data and labels for the models.
3. Learn a model using the experiences and data.
4. Generate new knowledge with the new models and previous knowledge.
5. Learn a new, more complex model of the world.

This framework has the relevant special characteristics defined in Section 6.2.2 for human learning: Increasing complexity and generating new knowledge and novel combinations of existing concepts.

For example, when humans perform a task, such as predicting the stock market, they use models that they have previously learned (e.g. economic cycles, previous crises...), current information (current state of the economy, interest rates...) and combine them in a logical and coherent way to generate new knowledge.

These learning strategies are innate capabilities that allow humans to generalize and create knowledge in a data-efficient manner. With training, learning and enough data, Level 2 collapses into Level 1. So, it seems appropriate to suggest that LLMs can emulate and generate learning strategies.

6.3.2 Using LLMs to generate learning strategies

In recent years, LLMs have revolutionized artificial intelligence and generative deep learning. For a sequence of symbols or tokens $(s_1, s_2, ..., s_n)$, these models estimate the conditional probability $p(s_n, ..., s_{n-k}|s_{n-k-1}, ..., s_1)$ and they perform not a single but many different tasks. To do that, they condition not only on the input but also on the task by modelling $p(output|input, task)$ [91, 92, 20, 87] and are trained in a self-supervised manner on a large corpus with some human feedback.

The input text, the task and the examples are entered as the input to the language model in what is known as a prompt. An example of a prompt would be:

"Translate the following sentence into french: {Sentence.} Example: { English sentence, french sentence}".

Recently, different approaches have been defined to structure the input and improve the reasoning capacity of LLMs, such as chain of thought [119] and tree of thought [125]. See [84, 34] for an overview of LLMs, their history, foundations and operation modes.

Here, we propose LLMs to generate learning strategies. These strategies define machine learning models and data, and interact with the environment to generate new knowledge.

A learning strategy in a LLM is a sequence of input-outputs in which the language model dynamically defines learning models and interacts with external tools and data to increase its knowledge. With this new approach, LLMs can emulate the human ability to create knowledge from available data and tools.

We define the following process to implement learning strategies using LLMs:

1. Output a machine learning strategy in a sequence of steps with model code and external data.
2. Implement the strategy and extract the new data and/or model.
3. Improve the learning model.
4. Complement the model to build a physical or abstract model of the world.

6.4 Experiments

6.4.1 Material and methods

In these experiments we want to evaluate the ability of LLMs to generate learning strategies that create new knowledge using external tools and data such as data sources, code interpreters, etc.

The objective is for the LLM to carry out the following activities, which comprise a learning strategy:

i Define a general machine learning strategy to solve a problem: predicting

oil prices.

ii Specify the source of the oil prices data and the code to download, and prepare the data.

iii Define a concrete learning model. Specify the model code, data transformation, learning procedure and meta-parameters.

iv Implement and validate the model.

v With the model (code, parameters, input-output data, conclusions) and data, how can we improve the model?

vi Implement and validate the new model.

vii With the model trained, obtain new data and test the model.

We evaluate whether LLMs are capable of performing these activities autonomously and without syntactic and semantic errors. Furthermore, we are interested in knowing if LLMs have a general ability to correctly define the activities of a learning strategy or if success depends rather on the type of activity and its significance in the training data.

Due to the limited capabilities of current LLMs to run complex code in their sanboxes, we run the code in an external environment, a Jupyter Notebook hosted in Google Colab.

The following LLMs, versions and characteristics are used for the experiments:

1. The OpenAI platform and APIs.
2. Completions API with the model "text-davinci-003" for some initial tests on logical meaning. The input is a free text prompt.
3. Chat Completions API with the model "gpt-3.5-turbo" for the rest of the tests. The main input is the messages parameter and messages must be an array of objects, where each object has a role (either "system", "user", or "assistant") and content.
4. In the "system" object we specify the instructions and guidelines. For example: "You can write and execute Python code..., the user will provide you with a problem and you have to define a machine learning model with those requirements. In the first step you have to search, download and prepare the data."
5. In the "user" object we specify the direct request. For example: "Please give

me data from a public repository as Kaggle."

6.4.2 Results

General ability to define and implement the entire learning strategy

First of all, we have asked the LLM to define and implement the entire learning strategy but the results have not been satisfactory. The output is not precise enough and many actions are specified with text rather than implemented with code.

To overcome these limitations, the strategy has been separated into sequential activities (general description, data search and download, model definition, model improvement...) and the LLM was asked to sequentially execute each activity separately. In each activity, the LLM is provided with the actions of that activity and the context of the previous activities.

General description of the learning strategy

In this first activity we evaluate the general approach of the machine learning strategy without evaluating the syntax and code execution.

```
completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a machine learning
engineer. The user will provide you with a problem and a
requirements field and you have to define a machine learning
strategy with those requirements. \
    A machine learning strategy is a set of instructions to learn
from the data and solve a problem: A set of training data and a
model to apply it.\
    Use the following step-by-step instructions to define the
machine learning strategy:\
    1. Specify data collection with data sources.\
    2. Justify model selection and training strategies.\
    3. Write the Python code of the learning strategy. Write the
detailed code for data loading, model definition and model
training."},
    {"role": "user", "content":" Can you give me a learning
strategy to predict stock prices? \
    Requirements: {Python library: PyTorch. Possible models: Any
machine learning model but preferably neural networks}."}
  ]
)
```

Figure 6.1: Example of a prompt for the learning strategy approach.

Figure 6.1 shows the prompt. The LLM outputs correctly the three sequential

actions (data collection, model selection and training strategies and python code for the learning strategy) with the details of each action. In each of the actions the LLM describes the logical steps. The concept of learning strategy is correctly understood and all the phases of the model (data loading, model definition, model training, model evaluation, etc.), the hyperparameters and external data have been stated or defined and explained.

The results are good since the actions have been specified in the prompt and are very typical actions with presence in the training data and with ambiguous syntactic and semantic validity since the code is not executed.

It is important to highlight that this is one of the most surprising but tricky abilities of LLMs, namely, their ability to emulate that they understand and solve a problem when they are instead recovering a series of steps and actions present in the training data.

Data search, download and preparation

In this activity we evaluate the capability of the LLM to search, download and load the data for the learning strategy.

```
completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You can write and execute Python
code by enclosing it in triple backticks, \
    e.g. ```code goes here```. Use this to run the code and execute
the model.\
    The user will provide you with a problem and you have to define
a machine learning model with those requirements. \
    You have to define and execute the model step by step. Remember
that you have to execute the Python code.\
    In the first step you have to download and load the training
data. Use any publicly available oil price dataset from Kaggle,
yahoo finance..."},
    {"role": "user", "content":" Please give me the code and path
to download a historical series of oil prices from a public
repository (Kaggle, etc.)"}
  ]
)
```

Figure 6.2: Example of a prompt for the data search and download activity.

Figure 6.2 shows the prompt. The LLM outputs correctly the code for the sequential actions: data url and data downloading, data loading, data preparation and data plotting. The actions follow a logical and coherent order and the code for each one is correct except for some minimal error.

The results are good enough since the actions have necessarily been specified in the prompt and have a presence in the training data. However, it was necessary to review and add some code from a specific library.

Definition and implementation of the model

In this activity we evaluate the ability of the LLM to prepare the data and define and train the machine learning model.

```
completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You can write and execute Python
code by enclosing it in triple backticks, \
    e.g. ```code goes here```. Use this to run the code and execute
the model.\
    The user will provide you with a problem and you have to define
a machine learning model with those requirements. \
    You have to define and execute the model step by step. Remember
that you have to execute the Python code.\
    In a previous step you have downloaded the training data, a
series of oil prices. You have the oil prices loaded in a Pandas \
    dataframe. The first column is the date, the second is the
price. The data contains 9011 rows, each one corresponds to a the
oil price of a day.\
    Write the code to define and train a deep learning model in
PyTorch to predict oil prices. The model can be neural networks,
RNNs, LSTMSs."},
    {"role": "user", "content":" Please give me the code to define
and train a deep learning model to predict oil prices\
    write the code and use PyTorch."}
  ]
)
```

Figure 6.3: Example of a prompt for the model definition and training activity.

Figure 6.3 shows the prompt. In the prompt we specify that we have an oil prices series in a Pandas dataframe and the format of the dataframe (column and rows). We only asked the LLM to define and train a deep learning model in PyTorch to predict oil prices. The key is whether the output of the LLM correctly implements the particularities of the strategy.

Syntactically, the output code needed an intense review since there were errors in model input formats that needed to know specific libraries.

Semantically, in general the output has the necessary actions (data preparation, definition and instantiation of the model, training, ...). These general results are expected since it is a typical activity with significance and presence in the training data. But these actions can have many different implementations and

the results show that the defined model incorrectly uses the entire sequence of prices (more than 9000) in each training step. This produces a model with large errors that does not replicate the training data.

Therefore, it seems that the LLM implements the most common actions and code in this activity and does not understand the particularities of the problem (goal, sequence size).

Model improvement

In this activity we evaluate the LLM's capability to understand a problem in a machine learning model and propose a better strategy and model.

```
completion = openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You can write and execute Python
code by enclosing it in triple backticks,\
    e.g. ```code goes here```. Use this to run the code and execute
the model.\
    The user will provide you with a problem and you have to define
a machine learning model with those requirements.\
    You have to define and execute the model step by step. Remember
that you have to execute the Python code.\
    In a previous step you have defined a deep learning model in
PyTorch to predict oil prices.\
    The model you defined loads the entire sequence in a tensor and
uses the entire sequence of more than 9000 points to train and
update the parameters.\
    It does not seem like the right strategy.\
    We want a better strategy with shorter training sequence
lengths and especially an strategy that fits the data trend
well."},
    {"role": "user", "content":"Please give me the code in PyTorch
to train a deep learning model to predict the trend in oil prices\
    write the code and use PyTorch."}
  ]
)
```

Figure 6.4: Example of a prompt for the model improvement activity.

Figure 6.4 shows the prompt. In this prompt we specify that (i) in a previous step the LLM has defined a deep learning model in PyTorch to predict oil prices, (ii) the model used the entire sequence to train and update the parameters and (iii) this does not seem the right strategy.

The LLM takes into account the problem in the previous model and provides us a better strategy with shorter training sequences to fit the data trend. When we execute the new model, we obtain better results and a much smaller loss. We

continue to see that the LLM has specific errors in parts of the code, such as preparing the appropriate input format for deep learning models (LSTM).

The results are satisfactory since the previous model problems have been detailed and specified in the prompt.

Model complementation

In this activity we want the LLM to test the model with recent data using an external API and to extract conclusions.

```
completion = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You can write and execute Python
code by enclosing it in triple backticks,\
    e.g. ```code goes here```. Use this to run the code and execute
the model.\
    The user will provide you with a problem and you have to define
a machine learning model with those requirements.\
    You have to define and execute the model step by step. Remember
that you have to execute the Python code.\
    In a previous step you have defined a deep learning model in
PyTorch to predict oil prices.\
    We have trained the model. The model is called 'model', you
don't need to load it and you can evaluate it using 'model.
eval()'.\
    The model has been trained with daily prices between May 1987
and November 2022.\
    The model has an input of shape (L,1,7), where L is the length
of the input sequence.\
    The data has been normalized with max_price= '143.95'.\
    We want to test the model with recent data from 2023 using a
external API like Yfinance and extract conclusions."},
    {"role": "user", "content":"Please give me the code in PyTorch
to get the new data an test the model.\
    Write the code and use PyTorch."}
  ]
)
```

Figure 6.5: Example of a prompt for the model complementation activity.

Figure 6.5 shows the prompt. In this prompt we specify a trained model, the method to evaluate the model, the date of the training series and the input shape.

The results are not satisfactory because, instead of using the start of the downloaded data and making a step-by-step prediction, the output model uses the entire sequence. Furthermore, the LLM does not provide any conclusions as requested. The explanation is that, despite having detailed the specifications in the prompt, this is a less common activity that requires a true understanding of the problem (complementing a model).

6.5 Discussion. LLMs as universal generators and the generator-verifier trade-off

In the previous experiments we have evaluated the generation of learning strategies with the following conclusions:

1. The LLM correctly implements the activities with presence and significance in the training data. The model specifies a set of instructions to learn from the data and solve a problem.
2. The code in each phase or activity of the strategy (data collection, model definition, implementation and learning...) is quite acceptable despite some errors that need human review.
3. This success is due to the fact that the strategy has been separated into activities and in each activity specific actions have been requested at the prompt, rathen than to a general planning capacity. In fact, LLMs like ChatGPT have been trained with web code repositories that implement such specific actions.
4. LLMs emulate that they understand and solve a problem when they are instead recovering a series of actions present in the training data.

We have seen that LLMs can perform some sort of specific planning and reasoning, specially for the learning strategies with a significant presence in the training data. Furthermore, these capabilities are complemented by access to external data and code execution environments.

However, LLMs do not have a general planning capacity based on principles and logical rules and, therefore, cannot ensure that the specific strategies generated are coherent, consistent and error-free. In order to talk about a general artificial intelligence that has the unique human capabilities shown in Table 6.1, LLMs should have out of the distribution planning ability and universal verifiers that ensure that the outputs are coherent and correct.

Only in this way could current AI, based on deep learning, be autonomous and reliable regardless of whether it uses methods based on replicating patterns and functions instead of logical reasoning.

Given the lack of universal verifiers, this limitation can be partially solved by either using LLMs to easy-to-evaluate problems or by complementing LLMs with

external verifiers (code error detection, math evaluators), prompt structuring and human refinement. The core of the solution then lies in the careful preparation of training data, the use of external verifiers and human intervention.

From this we can extract a trade-off inherent to current LLMs with technical, business and societal implications: The power of these models is their ability to generate strategies, activities and plans that sound good but are not fundamentally reliable. But if you want their operation to be reliable, you have to restrict them to some specific tasks (present in the training data) and tasks that are easy to verify, as shown in Figure 6.6.

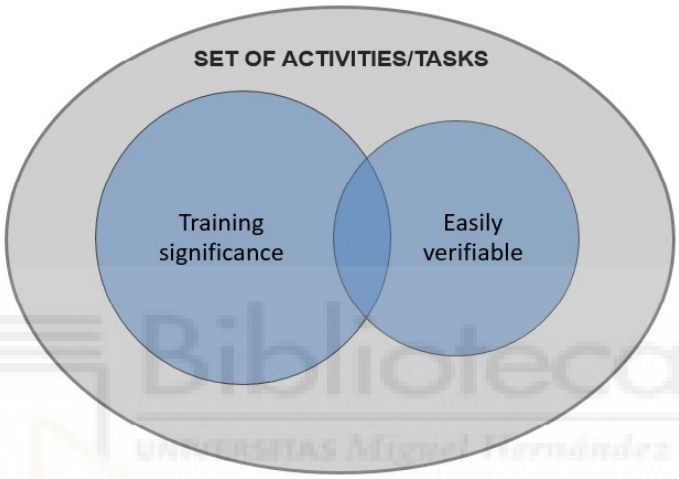

Figure 6.6: Classification of the set of reliable activities/tasks.

In our learning strategies framework you can prompt the LLM many different activities such as finding the data, defining the model, improving the model, combining it with another model, etc. However, for the output to be correct and autonomous you have to focus on the activities that are at the intersection of the two sets in Figure 6.6 (training significance and ease of verification), such as restricted model definition and implementation instead of more creative ones.

6.6 Conclusions

LLMs have good generative power because they have been trained with a large and diverse amount of data.

In the case of generating machine learning strategies, with appropriate instructions, LLMs adequately generate typical activities such as searching and

downloading data and defining and implementing the model. However, LLMs do not have a general planning capacity based on logical rules and therefore cannot ensure that the specific activities generated are coherent, consistent and error-free. Furthermore, the performance of LLMs drops greatly when the activities are more creative and do not coincide with the typical activities present in code repositories.

Due to these limitations, in the future it is necessary to adequately prepare the training data to include these activities and complement the LLMs with human intervention and external verifiers.

7 | Conclusions

7.1 Spanish

De manera general, se ha podido comprobar que los sistemas complejos, debido a sus diversas estructuras e interacciones, pueden contribuir a la programación diferenciable y al deep learning sirviendo de fuente de inspiración de nuevos modelos y estructuras, y siendo los candidatos ideales para aplicar las últimas técnicas de la programación diferenciable.

El estudio y la definición de las redes adaptativas multicapa permiten modelar sistemas en los que hay diversas interacciones en paralelo y influencia entre los nodos y las conexiones. Èste es el caso del procesamiento de la información neuronal, en el que las redes de neurotransmisores y las redes de neuromoduladores permiten a las neuronas conectarse y procesar información.

En cuanto a la programación diferenciable, hemos visto que es un marco general de programación parametrizable en función de los datos de entrenamiento. A través de una serie de características (relaciones entre tensores, invarianzas o simetrías, combinación de módulos...) podemos restringir el modelo diferenciable y adaptarlo al problema a resolver. En problemas con pocos datos de entrenamiento es importante restringir el modelo diferenciable para tener un aprendizaje más eficiente.

De la misma manera que en las redes complejas adaptativas multicapa, la estructura del modelo y las diferentes relaciones entre los vectores son claves para modelar un problema específico.

Al analizar los mecanismos de atención, concluimos que son una estructura diferenciable flexible que permite realizar una selección de vectores secuencial e interpretable. Esto permite modelar ciertas características típicas de los sistemas

complejos como la integración de diferentes partes, el razonamiento secuencial o las dependencias temporales de rango largo.

Respecto a los grandes modelos de lenguaje (LLMs) como solución a las limitaciones inherentes de la programación diferenciable, concluimos que pueden implementar correctamente actividades típicas con presencia en los datos de entrenamiento, pero no tienen una capacidad general para definir estrategias de aprendizaje de una manera lógica y coherente. De hecho, el rendimiento de los LLMs baja significativamente cuando intenta implementar actividades más creativas que no coinciden con actividades típicas de los datos de entrenamiento.

Enunciamos el compromiso generador-verificador de los LLMs: El poder de estos modelos es su capacidad para generar estrategias, actividades y planes que suenan bien pero que no son fundamentalmente confiables. Para que su funcionamiento sea fiable, hay que restringir estas actividades a algunas tareas específicas (presentes en los datos de entrenamiento) y tareas que sean fáciles de verificar.

Una solución parcial pasa por la preparación cuidadosa de los datos de entrenamiento, el uso de verificadores externos y la intervención humana.

Estas condiciones alejan a los LLMs de la inteligencia artificial general. En un futuro, además de usar los LLMs con verificadores externos para aumentar su fiabilidad, es muy importante seguir analizando sus limitaciones inherentes y la forma en la que aprenden sistemas complejos como el cerebro y su entorno. En estos sistemas, a través de la evolución y el aprendizaje, se elaboran modelos complejos de la realidad que interactúan con el entorno y con modelos lógicos para crear nuevo conocimiento.

Por último, para continuar los avances de la tesis y seguir profundizando en la relación entre deep learning y sistemas complejos, es necesario profundizar en las siguientes líneas futuras de investigación:

1. Realizar simulaciones y estudios empíricos de las redes adaptativas multicapa para seguir validando su utilidad práctica y su capacidad explicativa en sistemas complejos como el cerebro. Las capacidades computacionales de estas redes en distintas escalas temporales pueden tener relación con los diferentes tipos de aprendizaje: El sistema 1 más automático y relacionado con la percepción y el sistema 2 relacionado con el razonamiento lógico.
2. Usar técnicas complementarias al deep learning, como la regresión simbólica,

para encontrar nuevas relaciones y dinámicas entre los componentes de los sistemas complejos. Estas técnicas son directamente interpretables y al contrario que el deep learning, tienen una capacidad explicativa de base, lo que la hace ideal para explicar relaciones en fenómenos físicos.

3. Combinar los LLMs con verificadores lógicos, semánticos y experimentales para proponer, validar y modificar modelos matemáticos y computacionales de sistemas complejos. De esta forma, se combinan la capacidad generativa y expresiva de los LLMs con las restricciones lógicas y experimentales del problema a modelar.

7.2 English

In general, it has been proven that complex systems, due to their diverse structures and interactions, can contribute to differentiable programming and deep learning, serving both as a source of inspiration for new structures and as ideal candidates for applying the latest differentiable programming techniques.

The study and definition of multilayer adaptive networks allow modeling systems in which there are various parallel interactions and influence between nodes and connections. This is the case of neural information processing, in which neurotransmitter and neuromodulatory networks allow neurons to connect and process information.

Regarding differentiable programming, we have seen that it is a general programming framework that can be parameterized based on training data. Through a series of characteristics (relations between tensors, invariances or symmetries, combination of modules...) we can restrict the differentiable model and adapt it to the task. This restriction is indeed important in problems with little training data to have a more efficient learning.

In the same way as in complex multilayer adaptive networks, the structure of the model and the different relationships between the vectors are key to modeling a specific problem.

By analyzing attention mechanisms, we conclude that they are a flexible differentiable structure that implements sequential and interpretable vector selection. This allows modeling certain typical characteristics of complex systems such as the integration of different parts, sequential reasoning or long-range temporal dependencies.

Regarding Large Language Models (LLMs) as a solution to the inherent limitations of differentiable programming, we conclude that they can correctly implement typical activities with presence in the training data, but they do not have a general ability to define learning strategies in a logical and coherent way. In fact, LLMs performance drops significantly when you try to implement more creative activities.

We state the generator-verifier trade-off in LLMs: The power of these models is their ability to generate strategies, activities and plans that sound good but are not fundamentally reliable. For them to work reliably, you have to restrict these activities to some specific tasks (present in the training data) and tasks that are easy to verify.

A partial solution involves careful preparation of training data, the use of external verifiers and human intervention. But these conditions move LLMs away from general artificial intelligence. In the future, in addition to using LLMs with external verifiers to increase their reliability, it is very important to continue analyzing their inherent limitations and the way in which complex systems like the brain and its environment learn. In these systems, through evolution and learning, complex world models are created and those models interact with the environment and with logical models to generate new knowledge.

Finally, to continue the progress of the thesis and continue delving into the relationship between deep learning and complex systems, it is necessary to advacance into the following future lines of research:

1. Carry out simulations and empirical studies of multilayer adaptive networks to continue validating their practical usefulness and explanatory ability in complex systems like the brain. The computational capabilities of these networks at different time scales may be related to different types of learning: System 1 being more automatic and related to perception and system 2 related to logical reasoning.
2. Use complementary techniques to deep learning, such as symbolic regression, to find new relationships and dynamics between the components of complex systems. These techniques are directly interpretable and unlike deep learning, they have a basic explanatory ability, which makes it ideal for explaining relationships in physical phenomena.
3. Combine LLMs with logical, semantic and experimental verifiers to propose, validate and modify mathematical and computational models of complex

systems. In this way, the generative and expressive capabilities of LLMs are combined with the logical and experimental restrictions of the problem to be modeled.

References

[1] S. Achard and E. Bullmore. Efficiency and cost of economical brain functional networks. *PLoS Computational Biology*, 3(2):0174–0183, 2007.

[2] A.F. Alexander-Bloch, P.E. Vértes, R. Stidd, F. Lalonde, L. Clasen, J. Rapoport, J. Giedd, E.T. Bullmore, and N. Gogtay. The anatomical distance of functional connections predicts brain network topology in health and schizophrenia. *Cerebral Cortex*, 23(1):127–138, 2013.

[3] A.P. Alivisatos, M. Chun, G.M. Church, R.J. Greenspan, M.L. Roukes, and R. Yuste. The brain activity map project and the challenge of functional connectomics. *Neuron*, 74(6):970–974, 2012.

[4] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang. Bottom-up and top-down attention for image captioning and visual question answering. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6077–6086, 2018.

[5] T. Aoki, L.E.C. Rocha, and T. Gross. Temporal and structural heterogeneities emerging in adaptive temporal networks. *Physical Review E*, 93(4), 2016.

[6] T. Aoki, K. Yawata, and T. Aoyagi. Self-organization of complex networks as a dynamical system. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 91(1), 2015.

[7] P. Arena, L. Patané, and P. S. Termini. Modeling attentional loop in the insect mushroom bodies. *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2012.

[8] L. Arnold. *Random Dynamical Systems*. Springer, 1998.

[9] M. Assaad, R. Boné, and H. Cardot. Predicting chaotic time series by boosted recurrent neural networks. In *Neural Information Processing: 13th International Conference, ICONIP 2006, Hong Kong, China, October 3-6, 2006. Proceedings, Part II 13*, pages 831–840, 10 2006.

[10] D. Bahdanau, K. Cho, and Y Bengio. Neural machine translation by jointly learning to align and translate. *ArXiv*, 1409, 09 2014.

[11] S.T.E. Baker, D.I. Lubman, M. Yücel, N.B. Allen, S. Whittle, B.D. Fulcher, A. Zalesky, and A. Fornito. Developmental changes in brain network hub connectivity in late adolescence. *Journal of Neuroscience*, 35(24):9078–9087, 2015.

[12] C.I. Bargmann. Beyond the connectome: How neuromodulators shape neural circuits. *BioEssays*, 34(6):458–465, 2012.

[13] C.I. Bargmann and E. Marder. From the connectome to brain function. *Nature Methods*, 10(6):483–490, 2013.

[14] L. Barnett, C.L. Buckley, and S. Bullock. Neural complexity and structural connectivity. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 79(5), 2009.

[15] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018.

[16] B. Bentley, R. Branicky, C.L. Barnes, Y.L. Chew, E. Yemini, E.T. Bullmore, P.E. Vértes, and W.R. Schafer. The multilayer connectome of caenorhabditis elegans. *PLoS Computational Biology*, 12(12), 2016.

[17] J. L. Bermúdez. *Cognitive science: An introduction to the science of the mind*. Cambridge University Press, 2014.

[18] J. M. Bishop. Artificial intelligence is stupid and causal reasoning will not fix it. *Frontiers in Psychology*, 11, 2021.

[19] V. Brezina. Beyond the wiring signalling through complex neuromodulator networks. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 365(1551):2363–2374, 2010.

[20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krüger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[21] D. Bucher and E. Marder. Xsnapshot: Neuromodulation. *Cell*, 155(2):482–482.e1, 2013.

[22] B. Chang, M. Chen, E. Haber, and E. H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.

[23] O. Chang, L. Flokas, H. Lipson, and M. Spranger. Assessing satnet's ability to solve the symbol grounding problem. In *NeurIPS*, 2020.

[24] Y.Y. Chang, F.Y. Sun, Y. H. Wu, and S.D Lin. A memory-network based solution for multivariate time-series forecasting. *ArXiv*, abs/1809.02105, 2018.

[25] R. Chaudhuri and I. Fiete. Computational principles of memory. *Nature Neuroscience*, 19(3):394–403, 2016.

[26] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[27] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 06 2014.

[28] N. Daur, F. Nadim, and D. Bucher. The complexity of small circuits: the stomatogastric nervous system. *Current Opinion in Neurobiology*, 41:1–7, 2016.

[29] M. De Domenico. Multilayer modeling and analysis of human brain networks. *GigaScience*, 6(5):1–8, 2017.

[30] M. De Domenico, C. Granell, M.A. Porter, and A. Arenas. The physics of spreading processes in multilayer networks. *Nature Physics*, 12(10):901–906, 2016.

[31] G. Deco and E. Rolls. Neurodynamics of biased competition and cooperation for attention: A model with spiking neurons. *Journal of neurophysiology*, 94:295–313, 08 2005.

[32] L. Deng and Y. Liu. *A Joint Introduction to Natural Language Processing and to Deep Learning*, pages 1–22. Springer Singapore, Singapore, 2018.

[33] E. Diaconescu. The use of narx neural networks to predict chaotic time series. *WSEAS Transactions on Computer Research*, 3, 03 2008.

[34] M. R Douglas. Large language models. *arXiv preprint arXiv:2307.05782*, 2023.

[35] K. Doya. Metalearning and neuromodulation. *Neural Networks*, 15(4-6):495–506, 2002.

[36] K. Doya. Modulators of decision making. *Nature Neuroscience*, 11(4):410–416, 2008.

[37] D.J. Felleman and D.C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex (New York, N.Y. : 1991)*, 1(1):1–47, 1991.

[38] J.M. Fellous and C. Linster. Computational models of neuromodulation. *Neural Computation*, 10(4):771–805, 1998.

[39] M. Fey and J. E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[40] A. Fornito, A. Zalesky, and E.T. Bullmore. *Fundamentals of Brain Network Analysis*. Academic press, 2016.

[41] F. Fröhlich. *Network Neuroscience*. Academic Press, 2016.

[42] Z. Gan, Y. Cheng, A. E. Kholy, L. Li, J. Liu, and J. Gao. Multi-step reasoning via recurrent dual attention for visual dialog. In *ACL*, 2019.

[43] W. Gerstner, W.M. Kistler, R. Naud, and L. Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[44] Y. Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10:1–309, 04 2017.

[45] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[46] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. Recurrent independent mechanisms. *ArXiv*, abs/1909.10893, 2021.

[47] A. Graves, N. Jaitly, and A. Mohamed. Hybrid speech recognition with deep bidirectional lstm. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 273–278, 2013.

[48] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *ArXiv*, abs/1410.5401, 2014.

[49] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwinska, S. Gomez Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. Puigdomènech Badia, K. Moritz Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538:471–476, 2016.

[50] C. Gros. *Complex and adaptive dynamical systems. A primer. 3rd ed*, volume 1. Springer, 08 2008.

[51] G.J. Gutierrez and E. Marder. Modulation of a single neuron has state-dependent actions on circuit dynamics. *eNeuro*, 1(1), 2014.

[52] L. Hahne, T. Lüddecke, F. Wörgötter, and D. Kappel. Attention on abstract visual reasoning. *ArXiv*, abs/1911.05990, 2019.

[53] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[54] A.L. Hodgkin and A.F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117(4):500–544, 1952.

[55] P. Hohenecker and T. Lukasiewicz. Ontology reasoning with deep neural networks. *ArXiv*, abs/1808.07980, 2018.

[56] R. Holca-Lamarre, J. Lücke, and K. Obermayer. Models of acetylcholine and dopamine signals differentially improve neural representations. *Frontiers in Computational Neuroscience*, 11, 2017.

[57] T. Hollis, A. Viscardi, and S. E. Yi. A comparison of lstms and attention mechanisms for forecasting financial time series. *ArXiv*, abs/1812.07699, 2018.

[58] A. Horn, D. Ostwald, M. Reisert, and F. Blankenburg. The structural-functional connectome and the default mode network of the human brain. *NeuroImage*, 102(P1):142–151, 2014.

[59] S. Huang, D. Wang, X. Wu, and A. Tang. Dsanet: Dual self-attention network for multivariate time series forecasting. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 2129–2132, 11 2019.

[60] D. A. Hudson and C. D. Manning. Compositional attention networks for machine reasoning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[61] R. Huerta, S. Vembu, J. M. Amigó, T. Nowotny, and C. Elkan. Inhibition in multiclass classification. *Neural computation*, 24:2473–507, 05 2012.

[62] S. Jetley, N. Lord, N. Lee, and P. Torr. Learn to pay attention. *ArXiv*, abs/1804.02391, 2018.

[63] D. Kahneman. *Thinking, fast and slow.* Macmillan, 2011.

[64] M. Kivelä, A. Arenas, M. Barthelemy, J.P. Gleeson, Y. Moreno, and M.A. Porter. Multilayer networks. *Journal of Complex Networks*, 2(3):203–271, 2014.

[65] N.J. Kopell, H.J. Gritton, M.A. Whittington, and M.A. Kramer. Beyond the connectome: The dynome. *Neuron*, 83(6):1319–1328, 2014.

[66] S N. Kumpati, P. Kannan, et al. Identification and control of dynamical systems using neural networks. *IEEE transactions on neural networks*, 1 1:4–27, 1990.

[67] J. Laird and S. Mohan. Learning fast and slow: Levels of learning in general autonomous intelligent agents. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[68] G.C. Layek. *An Introduction to Dynamical Systems and Chaos*. Springer, 01 2015.

[69] Y. LeCun, Y Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[70] X. Li, Q. Chen, and F. Xue. Biological modelling of a computational spiking neural network with neuronal avalanches. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2096), 2017.

[71] Y.F. Li and H. Cao. Prediction for tourism flow based on lstm neural network. *Procedia Computer Science*, 129:277–283, 01 2018.

[72] G. W. Lindsay. Attention in psychology, neuroscience, and machine learning. *Frontiers in Computational Neuroscience*, 14, 2020.

[73] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang, and T.Y. Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. *ArXiv*, abs/1906.02762, 2019.

[74] G. Marcus. Deep learning: A critical appraisal. *ArXiv*, abs/1801.00631, 2018.

[75] E. Marder. Neuromodulation of neuronal circuits: Back to the future. *Neuron*, 76(1):1–11, 2012.

[76] E. Marder, M.L. Goeritz, and A.G. Otopalik. Robust circuit rhythms in small circuits arise from variable circuit components and mechanisms. *Current Opinion in Neurobiology*, 31:156–163, 2015.

[77] O.V. Maslennikov and V.I. Nekorkin. Adaptive dynamical networks. *Physics-Uspekhi*, 60(7):694–704, 2017.

[78] O.V. Maslennikov, D.S. Shchapin, and V.I. Nekorkin. Transient sequences in a hypernetwork generated by an adaptive network of spiking neurons. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2096), 2017.

[79] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[80] G. Menichetti, D. Remondini, P. Panzarasa, R.J. MondragÃ³n, and G. Bianconi. Weighted multiplex networks. *PLoS ONE*, 9(6), 2014.

[81] Y. Ming, D. Pelusi, C. N. Fang, M. Prasad, Y. K. Wang, D. Wu, and C. T. Lin. Eeg data analysis with stacked differentiable neural computers. *Neural Computing and Applications*, 11 2018.

[82] T. Miyoshi, H. Ichihashi, S. Okamoto, and T. Hayakawa. Learning chaotic dynamics in recurrent rbf network. In *Proceedings of ICNN'95-International Conference on Neural Networks*, pages 588 – 593 vol.1, 12 1995.

[83] F. Nadim and D. Bucher. Neuromodulation of neurons and synapses. *Current Opinion in Neurobiology*, 29:48–56, 2014.

[84] H. Naveed, A. U. Khan, S. Qiu, M. Saqib, S. Anwar, M. Usman, N. Barnes, and A. Mian. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*, 2023.

[85] V. Nicosia, P.E. Vértes, W.R. Schafer, V. Latora, and E.T. Bullmore. Phase transition in the economically modeled growth of a cellular nervous system. *Proceedings of the National Academy of Sciences of the United States of America*, 110(19):7880–7885, 2013.

[86] T. O'Leary, A.H. Williams, J.S. Caplan, and E. Marder. Correlations in ion channel expression emerge from homeostatic tuning rules. *Proceedings of the National Academy of Sciences of the United States of America*, 110(28):E2645–E2654, 2013.

[87] OpenAI et al. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

[88] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[89] J. Pearl. Theoretical impediments to machine learning with seven sparks from the causal revolution. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 3, New York, NY, USA, 2018. Association for Computing Machinery.

[90] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *ArXiv*, abs/1704.02971, 2017.

[91] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[92] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[93] J.A. Reggia, E. Ruppin, and D.L. Glanzman. *Disorders of brain, behavior, and cognition: the neurocomputational perspective*, volume 121. Elsevier, 1999.

[94] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon, and R. Sarkar. *PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models*, page 4564–4573. Association for Computing Machinery, New York, NY, USA, 2021.

[95] T.J. Ryan, D.S. Roy, M. Pignatelli, A. Arons, and S. Tonegawa. Engram cells retain memory under retrograde amnesia. *Science*, 348(6238):1007–1013, 2015.

[96] Y. Sato and S. Nagaya. Evolutionary algorithms that generate recurrent neural networks for learning chaos dynamics. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 144–149, May 1996.

[97] H. Sayama, I. Pestov, J. Schmidt, B.J. Bush, C. Wong, J. Yamanoi, and T. Gross. Modeling complex systems with adaptive networks. *Computers and Mathematics with Applications*, 65(10):1645–1664, 2013.

[98] R. Schaeffer, B. Miranda, and S. Koyejo. Are emergent abilities of large language models a mirage? *Advances in Neural Information Processing Systems*, 36, 2024.

[99] S. Serrano and N. A. Smith. Is attention interpretable? In *ACL*, 2019.

[100] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. R. Baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. F. C.

Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[101] H. Song, D. Rajan, J. J. Thiagarajan, and A. Spanias. Attend and diagnose: Clinical time series analysis using attention models, 2017.

[102] O. Sporns, G. Tononi, and R. Kötter. The human connectome: A structural description of the human brain. *PLoS Computational Biology*, 1(4):0245–0251, 2005.

[103] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-to-end memory networks. In *NIPS*, 2015.

[104] T. Sumers, S. Yao, K. Narasimhan, and T. L. Griffiths. Cognitive architectures for language agents. *arXiv preprint arXiv:2205.06898*, 2023.

[105] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[106] G. Tang, M. Müller, A. Rios, and R. Sennrich. Why self-attention? a targeted evaluation of neural machine translation architectures. In *EMNLP*, 2018.

[107] H.K. Titley, N. Brunel, and C. Hansel. Toward a neurocentric view of learning. *Neuron*, 95(1):19–32, 2017.

[108] S. Tonegawa, M. Pignatelli, D.S. Roy, and T.J. Ryan. Memory engram storage and retrieval. *Current Opinion in Neurobiology*, 35:101–109, 2015.

[109] E.K. Towlson, P.E. Vértes, S.E. Ahnert, W.R. Schafer, and E.T. Bullmore. The rich club of the c. elegans neuronal connectome. *Journal of Neuroscience*, 33(15):6380–6387, 2013.

[110] K. Valmeekam, S. Sreedharan, M. Marquez, A. Olmo Hernandez, and S. Kambhampati. On the planning abilities of large language models (a critical investigation with a proposed benchmark). *ArXiv*, abs/2302.06706, 2023.

[111] L.R. Varshney, B.L. Chen, E. Paniagua, D.H. Hall, and D.B. Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *PLoS Computational Biology*, 7(2), 2011.

[112] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NIPS*, 2017.

[113] P. Vinayavekhin, S. Chaudhury, A. Munawar, D. J. Agravante, G. De Magistris, D. Kimura, and R. Tachibana. Focusing on what is relevant: Time-series learning and understanding using attention. *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2624–2629, 2018.

[114] F. Wang, D. Zheng, X. Wu, G. M. Essertel, J. M. Decker, and T. Rompf. Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *ArXiv*, abs/1803.10228, 2018.

[115] P. W. Wang, P. Donti, B. Wilder, and Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6545–6554. PMLR, 09–15 Jun 2019.

[116] Y. Wang. A new concept using lstm neural networks for dynamic system identification. In *2017 American control conference (ACC)*, pages 5324–5329, 05 2017.

[117] Z. Wang, D. Xiao, F. Fang, R. Govindan, C.C. Pain, and Y. Guo. Model identification of reduced order fluid dynamics systems using deep learning. *International Journal for Numerical Methods in Fluids*, 86:255–268, 07 2017.

[118] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022.

[119] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[120] J.G. White, E. Southgate, J.N. Thomson, and S. Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, 314:1–340, 1986.

[121] M. Wiedermann, J.F. Donges, J. Heitzig, W. Lucht, and J. Kurths. Macroscopic description of complex adaptive networks coevolving with dynamic node states. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 91(5), 2015.

[122] T. Xiao, Q. Fan, D. Gutfreund, M. Monfort, A. Oliva, and B. Zhou. Reasoning about human-object interactions through dual attention networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3918–3927, 2019.

[123] O. Yadan, K. Adams, Y. Taigman, and M. A. Ranzato. Multi-gpu training of convnets. *CoRR*, abs/1312.5853, 2013.

[124] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[125] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

[126] A. J. Yu. *Computational Models of Neuromodulation*, pages 1–6. Springer New York, New York, NY, 2013.

[127] A.M. Zador, J. Dubnau, H.K. Oyibo, H. Zhan, G. Cao, and I.D. Peikon. Sequencing the connectome. *PLoS Biology*, 10(10), 2012.

Appendices

A | Source Code

To carry out the experiments, Google Colab environment and Python programming language have been used. Google Colab (`https://colab.research.google.com/`) is a hosted Jupyter Notebook service that provides access to computing resources, including GPUs and TPUs.

The repository that contains the experiments with different deep learning and differentiable models (experiments in Chapter 4 and foundations of Chapter 5) is `https://github.com/adrianhernr/DPPaperExperiments`.

The repository that contains the experiments with LLMs and OpenAI API (Chapter 6) is `https://github.com/adrianhernr/LLM-Strategies`.

B | The need for more integration between machine learning and neuroscience

The Need for More Integration Between Machine Learning and Neuroscience

Adrián Hernández and José M. Amigó

Abstract Neuroscience and machine learning are two interrelated fields with obvious synergies. In recent years we have seen profound advances in these fields and in the integration between them. However, there remain challenges for a greater integration between the two disciplines. In this chapter we describe two areas, different but related, that can serve as a guide in the future for this integration. On the one hand, it is necessary to have more explanatory algorithms in order to understand information processing in the brain. The combination of multilayer and adaptive networks can be the right framework to understand this processing and analyse the interesting computational capabilities that occur in the brain. On the other hand, machine learning algorithms should have, similar to brain processing, more innate structure. This prior structure could make the process of learning more efficient and intuitive, and support artificial intelligence.

1 Introduction

In recent years we have seen profound advances in the fields of neuroscience and machine learning. The use of deep neural networks together with the parallel-processing capabilities of GPUs (Graphics Processing Unit) has allowed to improve the performance in problems of machine learning (image recognition, language processing, game playing, etc.) [1, 2]. In neuroscience, a revolution is taking place thanks to technologies such as optogenetics that allow neuronal control, and to open initiatives to share neural data [3–5].

Neuroscience and machine learning are two interrelated fields with obvious synergies [6–8]. Three traditional relationships between machine learning and neuroscience are the following.

The first is the use of machine learning techniques to analyze and predict data in neuroscience. Several projects, such as the connectome, are producing large

A. Hernández · J. M. Amigó ( )
Centro de Investigación Operativa, Universidad Miguel Hernández, Av. de la Universidad s/n, 03202 Elche, Spain
e-mail: jm.amigo@umh.es

amounts of observations on the anatomical and functional connections of the brain [4]. Through the use of machine learning algorithms it is possible to classify, predict and extract conclusions from that wealth of data.

The second is the development of machine learning algorithms inspired by the brain. Artificial neural networks and deep learning emerged by emulating (and at the same time simplifying) the way in which the brain processes information [1, 9]. Evolution has made many biological processes an example of efficiency and robustness.

The third, which is a very important field of application, is the use of machine learning techniques to theorize and explain the functioning of the brain. The use of algorithms and ideas from machine learning can contribute in a prominent way to understand how information is processed in the brain, both at the biophysical and system level.

Despite the important developments in these three areas, a tighter integration between the two disciplines is necessary. The two disciplines must evolve together by adopting a common approach in which recent advances in machine learning can inspire new research in neuroscience and, the other way around, a better understanding of the brain could guide the development of new architectures, models and algorithms.

In this chapter we describe two areas, different but related, that can serve as a guide in the future for a greater integration between machine learning and neuroscience.

Recently, multilayer adaptive networks have been proposed as the appropriate framework for modelling neuronal processing [10, 11]. Multilayer because in the brain different synaptic and neuromodulatory layers interact to produce behaviour, and adaptive because the topology of these networks changes according to the dynamics. Within this framework, it is possible to identify and analyse interesting computational capabilities that occur in the brain, many of them similar to those used in machine learning.

Furthermore, the extra-synaptic (neuromodulatory) layers configure and specify the structure of neuronal circuits, adapting them to the environment and to the cognitive needs at each time.

In recent years, though, calls have been made for machine learning algorithms to assume more innate structure [12]. Only by assuming more innate structure, similar to what happens in brain processing, will it be possible to achieve more efficient and intuitive learning. This prior structure incorporates specific aspects of the target function.

2 Machine Learning and Neuroscience

Of the three traditional relationships between neuroscience and machine learning mentioned above, the use of machine learning to analyse data in neuroscience is a promising field with many challenges.

As pointed out in [13], the increasing data acquisition in neuroscience has created the need to leverage that information deluge to better understand how the brain works.

A large number of neuroscience databases have been compiled with information regarding neurons, gene expression, and microscopic and macroscopic brain structure. Although machine learning techniques have helped to analyze this data, much work remains to be done in this area [14].

In [15], the authors were able to define depression subtypes by clustering subjects according to patterns of functional connectivity. Even more, these biotypes were also informative in predicting which patients responded to a specific treatment.

However, in [16] the authors take a classical microprocessor as a model organism to see if data analysis techniques used in neuroscience can decode the way it processes information. They show that those techniques reveal interesting structure in the data but no relevant hierarchy of information processing. More importantly, the study suggests that availability of unlimited data is not sufficient to understand the functioning of the brain.

The design and development of bio-inspired machine learning algorithms is the second relationship between neuroscience and machine learning considered in the Introduction.

Machine learning has been influenced from the beginning by the biological processes that occur in the brain. For example, neural networks have become a key element in the development of machine learning in recent years.

Deep learning models use multiple levels of representation obtained by transforming the representation at one level into a representation at a higher, more abstract level [1]. These models try to emulate the activity in the neural layers of the neocortex, a part of the mammalian brain involved in higher-order brain functions.

Specific architectures of deep learning, such as convolutional neural networks [17], have also been inspired by biological processes. They replicate the hierarchical visual processing in the visual cortex. Furthermore, reinforcement learning, a technique inspired by behaviourist psychology in which an agent interacts with its environment to maximize some reward, has seen important progress in recent years with the use of deep neural networks [18].

Finally, the third relationship between neuroscience and machine learning that we mentioned before is the use of machine learning techniques to understand and model how information is processed in the brain. Computation, machine learning and applied mathematics are indeed a common source of interesting ideas and models for studying neuroscience.

Hodgkin-Huxley model [19] was a key achievement because it allowed to approximate the electrical characteristics of neurons and describe how action potentials in neurons are initiated and propagated. This model is relevant because it explains in a quantitative way some biological processes of the brain. However, it is necessary to go further and find algorithms that describe the underlying computation carried out in the brain. These models, similar to what is indicated in [13], should have some common characteristics:

(i) They must take into account all the biological components relevant to the process studied.
(ii) They must provide a description of how the observed processes and data are generated from the biological components.
(iii) A key aspect is to understand how the model works and how its components interact.
(vi) The model must be generalizable to similar processes in other subjects or organisms.

As we have seen, important advances have been made in the three traditional areas of relationship between machine learning and neuroscience. However, there remain challenges for a greater integration between the two disciplines. On the one hand it is necessary to have explanatory algorithms to better understand and model information processing in the brain. On the other hand, machine learning algorithms should have more innate structure, similar to brain processing. In the following sections we will see two promising approaches that try to address these challenges.

3 Multilayer Adaptive Networks in Neuronal Processing

The connectome is a wiring diagram mapping all the neural connections in the brain. It maps, at the cellular level, the neurons and synapses within a part or all of the brain and can be structural, if it describes anatomical connections, or functional, if it describes statistical associations.

The study of the connectome using graph theory has resulted in a significant advance in neuroscience [20–22]. Neurons are the nodes of the network and synapses correspond to the edges between those nodes. The use of new techniques for recording neural activity combined with network theory has allowed to link topological and dynamic patterns with mental disorders.

Modelling and analysing the connectome play a prominent role to understand neurotransmission (fast synaptic transmission) networks, in which neurotransmitters influence the postsynaptic neuron producing an inhibitory or excitatory response. However, knowledge and study of the connectome are not enough to understand how the brain processes information because neurons use other forms of communication as neuromodulation [23–25].

Neuromodulators modify neuronal circuit outputs by changing neuronal excitability and synaptic dynamics and strength [26]. They reconfigure the connectome and add new computational and processing capabilities to traditional synaptic transmission. Therefore, the connectome provides a minimal structure and neuromodulators shape the functional circuits that give rise to behaviour. Moreover, without taking into account these neuromodulatory layers, it is impossible to explain information processing in the brain.

Then, the question that arises is what models allow us to explain how the interaction of different layers (neurotransmission and neuromodulators) gives rise to information processing [11]?

In the last years, there has been a growing interest in adaptive and multilayer networks. Commonly, the study of dynamical networks has covered either dynamics on networks, in which nodes are dynamical systems coupled through static links, or dynamics of networks, where network topology evolves dynamically in time. But more recently adaptive networks have gained increasing interest [27–29]. In adaptive networks links change with the states of the nodes in an interplay between node states and network topology. An example of adaptive network is a neuronal network in which the firing rates and the synaptic connections interact with each other.

On the other hand, all the research has focused on single-layer networks, in which a single type of node is connected via a single type of link. But in most biological systems multiple entities interact with each other via multiple layers of connectivity, thus making it necessary to generalize network theory to study multilayer systems [30–32].

A typical multilayer network has the following components:

(i) A number N of nodes (denoted by Latin letters $i, j, \ldots$) and a number L of layers (denoted by Greek letters $\alpha, \beta, \ldots$).
(ii) Node $i \in \{1, 2, \ldots, N\}$ in layer $\alpha \in \{1, 2, \ldots, L\}$ has a state $s_{i\alpha}(t)$.
(iii) A 4th-order, time-dependent adjacency tensor $M(t)$ with components $m_{i\alpha}^{j\beta}(t)$ which are the weights of the link from any node i in layer α to any node j in layer β in the network. Self-links are excluded, so $m_{i\alpha}^{i\alpha}(t) = 0$.

In multilayer networks there can be different types of interactions between nodes: Intra-layer links within the same layer, inter-layer links between the same nodes in different layers and inter-layer links between different nodes in different layers. Multiplex networks, in which different layers are not interconnected except from each node to itself, are a special class of multilayer networks (see Fig. 1).

The combination of multilayer and adaptive networks is the right framework to understand neuronal information processing because:

(i) Neuromodulatory layers reconfigure the connectome by changing neuronal and synaptic properties.
(ii) These extra layers operate on temporal and spatial scales different from the fast synaptic ones and regulate neuronal processing.
(iii) The neuromodulatory layers interact with the neurotransmission layer in a complicated way, adding interesting computational capabilities.

In [33] the authors hold that further understanding of brain function and dysfunction will require an integrated framework, called “dynome”, that links brain connectivity with brain dynamics. As pointed out in [34] for the crustacean stomatogastric nervous system, different regulatory mechanisms (synaptic and intrinsic neuronal properties, neuromodulation and gene expression regulation) influence each other to produce robustness and flexibility to circuit outputs. In [10] the C. elegans

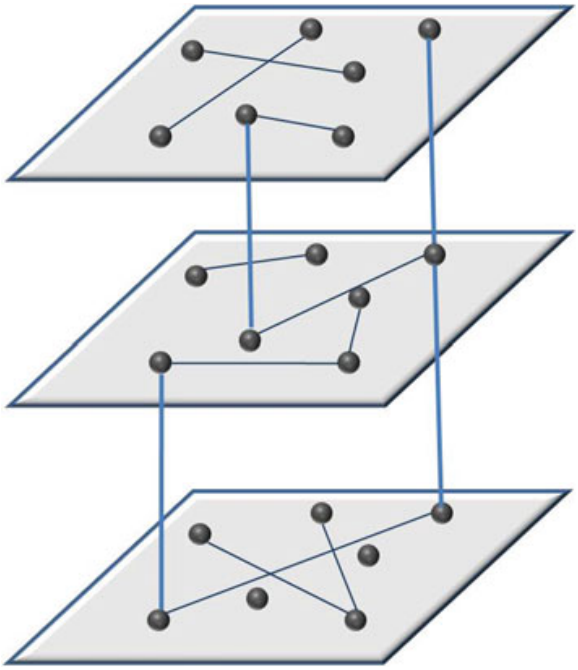

Fig. 1 A multiplex network, with multiple layers of the same set of nodes and different types of relationship between them

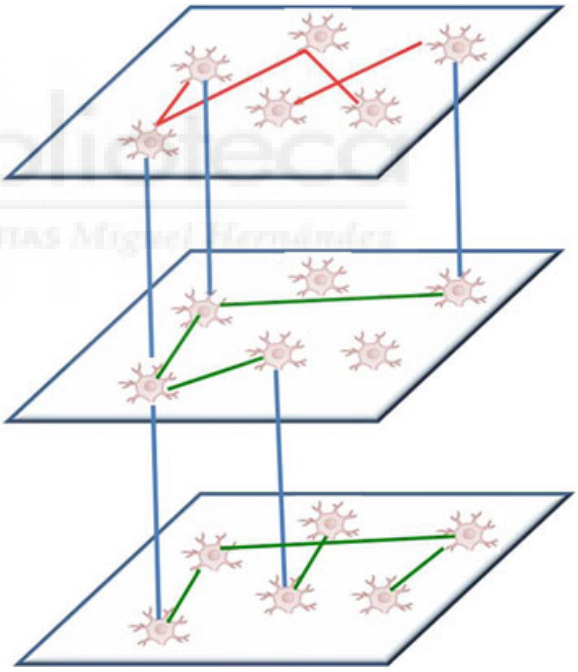

Fig. 2 A multilayer network with synaptic and neuromodulatory layers along with the corresponding communication links between layers

connectome is mapped as a multiplex network with synaptic, gap junction, and neuromodulatory layers representing alternative modes of interaction between neurons. The authors identified locations in the network where aminergic and neuropeptide signalling modulate synaptic activity.

We may conclude then that extending the connectome with synaptic and neuromodulatory layers seems the natural pathway to follow (see Fig. 2).

A simplified multilayer adaptive network model that accounts for these extra-layers of interaction is defined in [11]. The first layer is a typical neurotransmission

layer with adaptive node states and synapses. The remaining layers contain adaptive neuromodulatory states and links. The interaction between layers is of three types:

(i) Node states in the first layer are influenced by the states of the same node in the neuromodulatory layers.
(ii) Synapse weights in the first layer are influenced by the same link weights in the neuromodulatory layers.
(iii) Node states in the neuromodulatory layers are influenced by the state of the same node in the first layer.

Interesting insights can be extracted by examining basic aspects of this type of model. We can have extrinsic neuromodulation (i.e., neuromodulatory layers do not depend on neurotransmission activity) or intrinsic (i.e., neuromodulatory layers are not isolated from neurotransmission activity). Also, in most cases neuromodulation process is slower than the neurotransmission one.

Therefore, multilayer adaptive networks are a promising approach to take into account the different layers of neuronal communication and complete the connectome. Even more, they allow to analyze some computational capabilities added by neuromodulators such as circuit performance improvement [35], robustness [36, 37], circuit reconfiguration [38] and memory storage regulation.

4 More Innate Structure in Machine Learning Algorithms

The development of language is one of the most important cognitive processes in the human brain. Understanding how language is acquired and what biological preconditions are needed for language development are relevant subjects [39].

Traditionally, there have been two competing theories about language acquisition [40]:

(i) Nativist theory: Represented by Noam Chomsky, the theory argues that the learner possesses an innate capacity for dealing with language. The learner is biologically predisposed to learn languages. Chomsky says that all children have an innate language acquisition device (LAD).
(ii) Empiricist theory: According to the theory, there is enough information in the linguistic input and therefore there is no need to assume an innate structure. Learning begins with no knowledge of language but with the ability to learn it. In this model the determinant factor is the environment.

Although currently the language is not seen as the effect of the environment on a blank slate, there is a debate about the nature of the prior structure in language acquisition.

In a similar way, there is at present a constructive debate about the need of incorporating more structure into deep learning systems [41]. Assuming that a structure is necessary, the two points of view here differ in the role that this structure must have in the algorithm.

Possibly, the path towards artificial intelligence involves using more efficient algorithms that allow learning in a more intuitive way. For the following reasons, this is only possible by assuming more structure:

(i) As we have seen in the past, incorporating prior knowledge about the underlying function has achieved great results. Convolutional neural networks [17] make the assumption that the processing of an image should be translationally invariant. The success of long short-term memories [42] comes from modelling temporal dependencies and structure, especially in language processing.
(ii) As pointed out in [12], humans and other animals are born with a lot of innate machinery.
(iii) The human brain develops its cognitive functions thanks to prior structures adapted to each function. Also, alterations in brain connectivity have been linked to mental disorders [43].
(iv) Only with more structure it is possible to design algorithms that can learn more from less data.

Then, how much structure is convenient to incorporate into algorithms in order to improve learning? According to [44], a good start is to understand the innate structure in humans minds and try to analyse if these mechanisms add value to artificial intelligence.

Also, in [12] the author proposes a list of innate machinery and structure to support artificial general intelligence (e.g., object representation, structured representations, causality, translational invariance, …).

Although the debate will continue, we must face it from an open perspective and use evolution and the human brain as a source of inspiration. Millions of years of human evolution have provided us with valuable structures and mechanisms for learning.

5 Conclusion and Outlook

Despite the relevant advances that have been made in machine learning, neuroscience and in the integration between both, there remain challenges for a greater integration.

In neuroscience it is fundamental to find algorithms that provide a description of how the observed processes and data are generated from the biological components. This algorithms must take into account all the biological components relevant to the process studied. Multilayer adaptive networks, in which different synaptic and neuromodulatory layers interact to produce behaviour, is a promising approach in that sense. They allow to analyse some interesting computational capabilities added by neuromodulators. Furthermore, in this models the extra-synaptic (neuromodulatory) layers configure and specify the structure of neuronal circuits, adapting them to the environment.

Regarding future research, we indicate next promising lines of application of multilayer adaptive networks:

(i) Concrete biological models of interesting phenomena via the multilayer approach could provide a description of the observed processes and how they are generated from the biological components.
(ii) Use of multilayer adaptive networks for the study of the computational capabilities added by the additional chemical layers (e.g., adaptability, regulation, robustness, degeneracy, memory, recurrency).
(iii) New theoretical and computational models, such as multilayer adaptive networks, will be required to explain new observations made with optogenetics. This technique has made it possible to differentiate between mechanisms of memory retrieval and memory storage [3, 45, 46].

There is currently a constructive debate about the need of more structure into deep learning systems. Probably, algorithms should have, similar to brain processing, more innate structure to learn in a more intuitive way. As seen in the past, incorporating prior knowledge has achieved good results, and in the future it can support artificial general intelligence. One way to obtain this innate machinery is to analyse what kind of mechanisms and structure have allowed the brain to perform such diverse functions so efficiently.

Looking forward, and as pointed out in [44], a possible roadmap to incorporate more prior knowledge in machine learning algorithms and support artificial intelligence is:

(i) For the different functions implemented by the brain, analyse what architectures, constraints and assumptions facilitate the learning process.
(ii) Try to incorporate this structure into algorithms that perform similar functions but in machine learning.
(iii) In general, use results on information processing in the brain to design new machine learning models.

With these two approaches, neuroscience and machine learning can benefit from each other, eliciting a positive contribution of mutual value:

(i) Advances in machine learning and mathematical modelling can be harnessed in neuroscience to describe the underlying computation carried out in the brain.
(ii) A better understanding of the brain, especially of its structure and mechanisms for information processing, could guide the development of new architectures, models and algorithms in machine learning.

This work was supported by the Spanish Ministry of Science and Innovation, grant PID2019-108654GB-I00.

References

1. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521:436–444
2. Yadan O, Adams K, Taigman Y, Ranzato M (2014) Multi-GPU training of ConvNets. arXiv:1312.5853

3. Tonegawa S, Pignatelli M, Roy DS, Ryan TJ (2015) Memory engram storage and retrieval. Curr Opin Neurobiol 35:101–109
4. Sporns O, Tononi G, Kötter R (2005) The human connectome: a structural description of the human brain. PLoS Comput Biol 1:e42
5. Horn A, Ostwald D, Reisert M, Blankenburg F (2014) The structural-functional connectome and the default mode network of the human brain. NeuroImage 102:142–151
6. Hinton G (2011) Machine learning for neuroscience. Neural Syst Circuits 1:12
7. Marblestone AH, Wayne G, Kording KP (2016) Toward an integration of deep learning and neuroscience. Front Comput Neurosci 10:94
8. Velde F (2010) Where artificial intelligence and neuroscience meet: the search for grounded architectures of cognition. Adv Artif Intell 2010:918062
9. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5:115
10. Bentley B, Branicky R, Barnes CL, Chew YL, Yemini E, Bullmore ET, Vértes PE, Schafer WR (2016) The multilayer connectome of Caenorhabditis elegans. PLoS Comput Biol 12:e1005283
11. Hernández A, Amigó JM (2018) Multilayer adaptive networks in neuronal processing. Eur Phys J Spec Top 227:1039–1049. https://doi.org/10.1140/epjst/e2018-800037-y
12. Marcus G (2018) Innateness, AlphaZero, and artificial intelligence. arXiv:1801.05667
13. Vu MT, Adal T, Ba D, Buzsáki G, Carlson D, Heller K, Liston C, Rudin C, Sohal VS, Widge AS, Mayberg HS, Sapiro G, Dzirasa K (2018) A Shared vision for machine learning in neuroscience. J Neurosci 38:1601–1607
14. Helmstaedter M (2015) The mutual inspirations of machine learning and neuroscience. Neuron 86:25–28
15. Drysdale AT, Grosenick L, Downar J, Dunlop K, Mansouri F, Meng Y, Fetcho RN, Zebley B, Oathes DJ, Etkin A, Schatzberg AF, Sudheimer K, Keller J, Mayberg HS, Gunning FM, Alexopoulos GS, Fox MD, Pascual-Leone A, Voss HU, Casey BJ, Dubin MJ, Liston C (2017) Resting-state connectivity biomarkers define neurophysiological subtypes of depression. Nat Med 23:28–38
16. Jonas E, Kording KP (2017) Could a neuroscientist understand a microprocessor? PLoS Comput Biol 13:e1005268
17. Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86:2278–2324
18. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA (2017) A brief survey of deep reinforcement learning. IEEE Signal Process Mag
19. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117:500–544
20. Achard S, Bullmore E (2007) Efficiency and cost of economical brain functional networks. PLoS Comput Biol 3:e17
21. Baker ST, Lubman DI, Yücel M, Allen NB, Whittle S, Fulcher BD, Zalesky A, Fornito A (2015) Developmental changes in brain network hub connectivity in late adolescence. J Neurosci 35:9078–9087
22. Barnett L, Buckley CL, Bullock S (2009) Neural complexity and structural connectivity. Phys Rev E 79:051914
23. Bargmann C, Marder E (2013) From the connectome to brain function. Nat Methods 10:483–490
24. Bargmann C (2012) Beyond the connectome: how neuromodulators shape neural circuits. BioEssays 34:458–465
25. Brezina V (2010) Beyond the wiring diagram: signalling through complex neuromodulator networks. Philos Trans R Soc B 365:2363–2374
26. Nadim F, Bucher D (2014) Neuromodulation of neurons and synapses. Curr Opin Neurobiol 29:48–56
27. Sayama H, Pestov I, Schmidt J, Bush BJ, Wong C, Yamanoi J, Gross T (2013) Modeling complex systems with adaptive networks. Comput Math Appl 65:1645–1664

28. Maslennikov OV, Nekorkin VI (2017) Adaptive dynamical networks. Physics-Uspekhi 60:694–704
29. Aoki T, Rocha LEC, Gross T (2016) Temporal and structural heterogeneities emerging in adaptive temporal networks. Phys Rev E 93:040301
30. Kivelä M, Arenas A, Barthelemy M, Gleeson JP, Moreno Y, Porter MA (2014) Multilayer networks. J Complex Netw 2:203–271
31. De Domenico M, Granell C, Porter MA, Arenas A (2016) The physics of spreading processes in multilayer networks. Nat Phys 12:901–906
32. De Domenico M (2017) Multilayer modeling and analysis of human brain networks. Gigascience 6:1–8
33. Kopell N, Gritton HJ, Whittington MA, Kramer MA (2014) Beyond the connectome: the dynome. Neuron 83:1319–1328
34. Daur N, Nadim F, Bucher D (2016) The complexity of small circuits: the stomatogastric nervous system. Curr Opin Neurobiol 41:1–7
35. Holca-Lamarre R, Lücke J, Obermayer K (2017) Models of acetylcholine and dopamine signals differentially improve neural representations. Front Comput Neurosci 11:54
36. O'Leary T, Williams AH, Caplan JS, Marder E (2013) Correlations in ion channel expression emerge from homeostatic tuning rules. PNAS 110:E2645–54
37. Marder E, Goeritz ML, Otopalik AG (2015) Robust circuit rhythms in small circuits arise from variable circuit components and mechanisms. Curr Opin Neurobiol 31:156–163
38. Gutierrez GJ, Marder E (2014) Modulation of a single neuron has state-dependent actions on circuit dynamics. eNeuro 1. ENEURO.0009-14.2014
39. Mueller JL, Männel C, Friederici AD (2015) Biological preconditions for language development. In: Wright JD (ed) International encyclopedia of the social & behavioral sciences, 2nd edn. Elsevier Press, Oxford, pp 650–655
40. da Silva RB (2007) A brief discussion on the biological factors in the acquisition of language. Revista do GEL S J do Rio Preto 4:153–169
41. Deep Learning, Structure and Innate Priors. A Discussion between Yann LeCun and Christopher Manning. http://www.abigailsee.com/2018/02/21/deep-learning-structure-and-innate-priors.html
42. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9:1735–1780
43. Zeng K, Kang J, Ouyang G, Li J, Han J, Wang Y, Sokhadze EM, Casanova MF, Li X (2017) Disrupted brain network in children with autism spectrum. Sci Rep 7:16253
44. Marcus G (2018) Deep learning: a critical appraisal. arXiv:1801.00631
45. Ryan TJ, Roy DS, Pignatelli M, Arons A, Tonegawa S (2015) Memory. Engram cells retain memory under retrograde amnesia. Science 348:1007–1013
46. Titley HK, Brunel N, Hansel C (2017) Toward a neurocentric view of learning. Neuron 95:19–32

C | Multilayer adaptive networks in neuronal processing

Regular Article

Multilayer adaptive networks in neuronal processing

Adrián Hernández and José M. Amigó[a]

Centro de Investigación Operativa, Universidad Miguel Hernández, Av. de la Universidad s/n, 03202 Elche, Spain

Received 31 March 2018 / Received in final form 28 June 2018
Published online 12 December 2018

Abstract. The connectome is a wiring diagram mapping all the neural connections in the brain. At the cellular level, it provides a map of the neurons and synapses within a part or all of the brain of an organism. In recent years, significant advances have been made in the study of the connectome via network science and graph theory. This analysis is fundamental to understand neurotransmission (fast synaptic transmission) networks. However, neurons use other forms of communication as neuromodulation that, instead of conveying excitation or inhibition, change neuronal and synaptic properties. This additional neuromodulatory layers condition and reconfigure the connectome. In this paper, we propose that multilayer adaptive networks, in which different synaptic and neurochemical layers interact, are the appropriate framework to explain neuronal processing. Then, we describe a simplified multilayer adaptive network model that accounts for these extra-layers of interaction and analyse the emergence of interesting computational capabilities.

1 Introduction

In the last years considerable efforts are being dedicated to provide insights into neural circuits in what has been called the connectome [1–4]. The connectome is a map of neural connections in the brain and may be thought of as its "wiring diagram".

The connectome can be structural, if it describes anatomical connections between parts of the brain or neurons, or functional, if it describes statistical associations between activities in those parts. If we think of the structural connectome as a road map, then the functional connectome corresponds to the vehicles that travel the roads.

The knowledge of the neural elements and their neural connections can help understand how the cognitive function emerges from the neuronal structure and dynamics. This wiring diagram maps all the neural connections in the brain and, at the cellular level, it provides a map of the neurons and synapses in the brain.

[a] e-mail: jm.amigo@umh.es

The ideal framework to study and model the dynamics, topology and properties of this type of synaptic connections is that of complex networks and dynamical systems. Different approaches, from artificial neural networks to biophysical models that take into account the biological reality (conductances, response times, properties of synapses and dendrites, ...), have been used to describe the dynamics of neuronal connections and information processing in the brain; see [5] for a comprehensive description of neuronal dynamic models.

However, as some neuroscientists have pointed out [6–9], knowledge and modelling of the connectome, either structural or functional, are not enough to understand how the brain processes the information, although they contribute in a prominent way. There are other biological mechanisms, such as neuromodulation, that reconfigure the connectome.

In neuromodulation [10], a given neuron uses one or more chemicals to regulate diverse populations of neurons, in contrast to classical synaptic transmission, where one presynaptic neuron directly influences a single postsynaptic neuron. Neuromodulators operate on several time scales, and modulate and configure the connectome so as to determine the processing of information. The connectome provides a minimal structure and, on top of it, neuromodulators configure and specify the functional circuits that give rise to behaviour.

Thus, neuromodulators add new computational and processing capabilities to traditional synaptic transmission. First, they add extra-layers of biochemicals that regulate or tune neuronal processing. Second, the temporal scales of these extra-layers are different from classical ones. Third, these extra-layers and classical synaptic networks interact in a complicated way.

We propose that the appropriate framework for modelling neuronal processing is that of multilayer adaptive networks. Multilayer because in the brain different synaptic and neuromodulatory networks interact to produce behaviour, and adaptive because the topology of these networks changes according to the dynamics.

The first goal of this paper is to highlight the limitations of the connectome and neurotransmission networks to understand neuronal processing, as well as to point out the need of using a new framework. We review examples in which the same structural network can have different configurations and behaviours.

The second goal is to define the characteristics that a complex network framework must have in order to provide a complete description of the different neuronal information dynamics, scales and interactions that occur in the brain. Multilayer adaptive networks, in which different synaptic and chemical layers interact, are the appropriate framework to explain neuronal processing and the emergence of interesting computational capabilities.

2 Beyond neurotransmission networks

The connectome is a wiring diagram mapping all the neural connections in the brain. At the cellular level, it provides a map of the neurons and synapses within a part or all of the brain of an organism. The structural connectome provides the basis on which functional activity is implemented and therefore shapes the functional connectivity.

On the way to unveil the connectome of the human brain, one of the ultimate goals in neuroscience, some milestones have been achieved, e.g., the complete connectome of the nematode *Caenorhabditis elegans* [11,12], which comprises 302 neurons.

In recent years, significant advances have also been made in the study of the connectome via network science and graph theory [13–17]. At the cellular level, the nodes of the network are the neurons, and the edges correspond to the synapses between those neurons. Therefore, graph theory is the ideal framework to study the topology and dynamics of brain networks. The combination of new tools to map and

record neuronal patterns and the computational techniques of network science has provided a new setting for the study of the brain dynamics.

Many studies have been conducted to analyze the topology and dynamics of the neuronal connectome. In [18], the authors studied the growth rules of the neuronal system of *C. elegans*. They found that the network growth undergoes a transition from an accelerated to a constant increase in the number of synaptic connections as a function of the number of neurons. The transition between different growth regimes may be explained by a dynamic economical model incorporating a trade-off between topology and cost. In [19], graph theory is used to investigate the neuronal connectome of *C. elegans*. The authors identified a small number of highly connected neurons as a rich club interconnected with high efficiency and high connection distance.

Clearly, connectome modelling and analysis play a salient role when it comes to understand neurotransmission (fast synaptic transmission) networks. However, neurons use other forms of communication as neuromodulation that, instead of conveying excitation or inhibition, change neuronal and synaptic properties.

As described in [20], neuromodulators are released in modes other than fast synaptic transmission and modify the neuronal circuit outputs. They are the main factors in shaping behaviour by changing neuronal excitability and synaptic dynamics and strength. The processes that are subject to modulation include changes in probability of neurotransmitter release, changes in transmitter receptors, modification of synaptic strength, adding or subtracting ionic currents, and changes in voltage and time dependence of channel gating. In doing so, they reconfigure the connectome and provide adaptability of the brain functions.

Although traditionally neurochemical messengers have been classified as small-molecule neurotransmitters, biogenic amines and neuropeptides, for the purposes of this paper it is more appropriate to differentiate between neurotransmitter (fast synaptic transmission) and modulator functions as in [8].

In neuronal circuits, connectivity alone does not provide enough information to predict circuits outputs [6,7]. Neuronal processing and behaviour are sensitive to intrinsic channels and electrical properties that vary within and between cell types. Fast synaptic transmission and biochemical processes interact to generate complex dynamics in neurons and circuits.

Studies of neuronal circuits in invertebrate and vertebrate animals [7] have revealed the ability of neuromodulators to reconfigure information processing. They change the composition of neuronal circuits and permit a circuit with a fixed number of neurons to produce many different patterns of activity.

Then, the greatest challenge that we face to understand the brain is to have new models that allow us to explain how the interaction of different layers of neurotransmission, neuromodulators and genetic changes gives rise to information processing. Moreover, without taking into account these different interactions, it is impossible to explain many computational functions observed in the brain.

3 Multilayer adaptive networks in neuronal processing

Network science has grown over the last decades to become a relevant conceptual framework for the analysis of many real systems. A tremendous progress has been made in the application of network models in neuroscience. Modelling brain networks as graphs of nodes connected by edges has provided major advances in understanding brain dynamics. From the dynamic analysis of groups of neurons to the topological characterization of large-scale human brain networks, network theory has become a fundamental tool in neuroscience; see [21] and [22] for a comprehensive description of network neuroscience.

Traditionally, the study of dynamical networks has covered either dynamics on networks or dynamics of networks. In the first approach, nodes are dynamical systems coupled through static links. This case includes dynamical systems describing the dynamics in a phase space with no topological changes between the nodes. In the second approach, network topology evolves dynamically in time. This is the case of traditional complex networks, where the focus has been put on analyzing the statistical properties that arise from exogenous topological transformations.

In recent years, there has been a growing interest in adaptive networks, i.e., networks whose links change adaptively with the states of the nodes in an interplay between node states and network topology [23–27]. Two facts make adaptive networks specially convenient for the study of natural and social systems. First, dynamical processes on a network are sensitive to the network topology, which influences the states of nodes. Second, the states of the nodes feed back to the network topology creating a dynamical feedback loop between topology and states of the network. In a neuronal network, the firing rate of a neuron depends on the synaptic connections (topology) and, in turn, the evolution of the synaptic connections and weights depends on the neuronal activity. Furthermore, both processes – neuronal activity and synaptic reconfiguration – take place at different timescales.

A typical adaptive, directed network with a fixed set of nodes is composed of the following elements:

(i) A set of N nodes $V = \{v_1, v_2, ..., v_N\}$. Abusing notation, node v_i will be denoted by i.

(ii) Each node $i \in \{1, ..., N\}$ has a state $s_i(t)$.

(iii) The set of evolving links is encoded in a time-dependent, weighted adjacency matrix $A(t)$ with entries $a_{ij}(t)$. In our case, self-links are excluded, so $a_{ii}(t) = 0$.

(iv) Each link weight $a_{ij}(t)$ represents the relationship from node i to node $j \neq i$ and is a function of $s_i(t)$ and $s_j(t)$.

(v) Node states $s_j(t)$ are a function of the sum of incoming weighted nodes, that is, $\sum_{i=1}^{N} a_{ij}(t)s_i(t)$.

These systems change their states and topologies simultaneously according to their interrelated dynamical rules. In a link removal $a_{ij}(t) \neq 0$ becomes $a_{ij}(t) = 0$ while a new link is established when $a_{ij}(t) = 0$ becomes $a_{ij}(t) \neq 0$.

In addition, until recently the majority of studies have focused on single-layer networks, usually with a single type of node connected via a single type of link. But in most biological systems multiple entities interact with each other in complicated patterns that include multiple layers of connectivity. Consequently, it became necessary to generalize network theory by developing a new setting to study multilayer systems in a comprehensive fashion [28–31]. Then, multilayer networks are the suitable framework to study different networks that interact to produce complex activities.

As we have seen, single-layer networks are represented using adjacency matrices which, in the case considered, represent directed and weighted relationships between the nodes of a network. Instead, multilayer networks represent multiple dimensions of connectivity that, in the case of neurons, can stand for different types of communication channels (neurotransmitters and neuromodulators). A typical multilayer network has the following ingredients:

(i) A number N of nodes (denoted by Latin letters i, j, ...) and a number L of layers (denoted by Greek letters α, β, ...).

(ii) Node $i \in \{1, 2, ..., N\}$ in layer $\alpha \in \{1, 2, ..., L\}$ has a state $s_{i\alpha}(t)$.

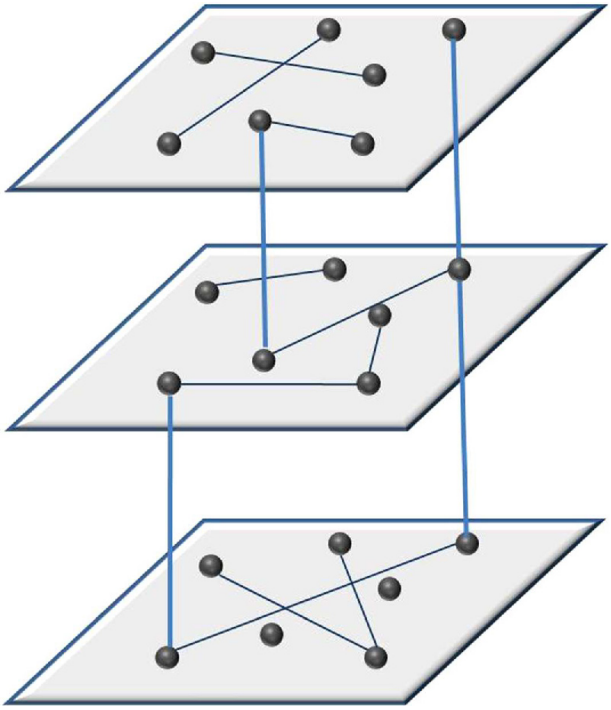

Fig. 1. A multilayer network with intra-layer edges and inter-layer edges that connect entities with their replicas in other layers.

(iii) A 4th-order, time-dependent adjacency tensor $M(t)$ with components $m_{i\alpha}^{j\beta}(t)$ which are the weights of the link from any node i in layer α to any node j in layer β in the network.

In multilayer networks (see Fig. 1), nodes can be connected by different types of interactions: intra-layer links connecting nodes within the same layer, inter-layer links between the same nodes in different layers, as well as inter-layer links between different nodes in different layers. Multiplex networks are a special class of multilayer networks such that $m_{i\alpha}^{j\beta} = 0$ if $\alpha \neq \beta$ and $i \neq j$, i.e., different layers are not interconnected except from each node to itself.

An interesting example of a multilayer network can be defined by extending the connectome with synaptic and neuromodulatory layers representing alternative modes of interaction between neurons, along with the corresponding communication links between layers (see Fig. 2).

The combination of multilayer and adaptive networks describes networks with different interactions between their nodes, together with a dynamical feedback between network topology and node states.

As we pointed out before, the connectome and single synaptic networks are not enough to understand neuronal information processing. We summarize next the characteristics that make multilayer adaptive networks the right framework:

(i) In addition to fast synaptic transmission, neurons use other forms of communication such as neuromodulation that change neuronal and synaptic properties.

(ii) The extra layers of a multilayer approach regulate or tune neuronal processing and operate on temporal and spatial scales different from the fast synaptic ones.

(iii) The neuromodulatory layers interact with the fast synaptic transmission layer in a complicated way, changing neuronal excitability and synapses dynamics and strength.

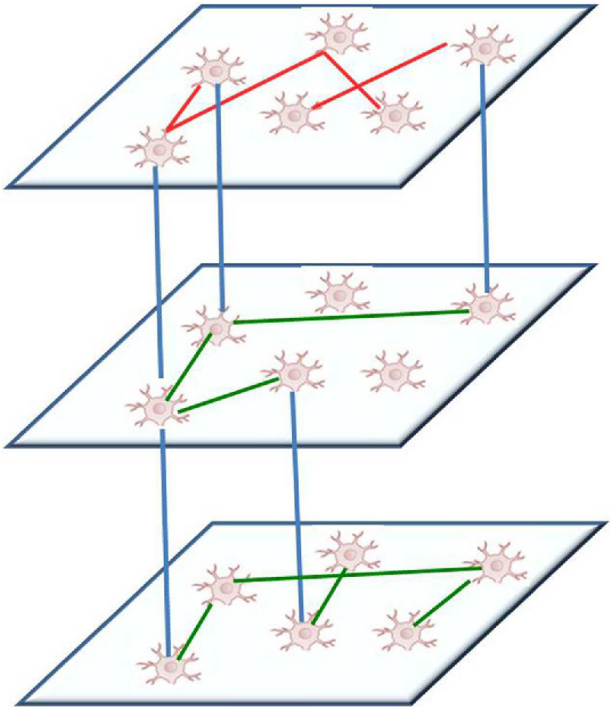

Fig. 2. A multilayer network with one synaptic layer (top layer, red links online) and two neuromodulatory layers (middle and bottom layers, green links online) representing alternative modes of interaction between neurons, along with the corresponding communication links between layers.

(iv) The fast neurotransmission layer topology changes according to neuronal and neuromodulatory activity.

4 Modeling neurotransmission and neuromodulation

The connectome represents a network of potential neurons and connections (synapses). Function, context and neuromodulatory networks shape and reconfigure the connectome network allowing different paths of information flow.

Neurotransmission is a wiring transmission that targets designated synapses and produces localized responses. On the other hand, neuromodulation is a volume transmission that diffuses through large areas of the neural tissue and affects multiple neurons. As a result, neuromodulatory networks are more complex than neurotransmission ones because neuromodulators act non-locally.

In [32], the authors hold that further understanding of brain function and dysfunction will require an integrated framework that links brain connectivity with brain dynamics. This expanded description is called "dynome" and includes the functional connectome but expands the notion to the mechanisms involved in producing and processing brain signals. Detailed biophysical models of neural activity embedded in an anatomical network may be essential to examine the effects of biological dynamics on functional connectivity.

As described in [33] for the crustacean stomatogastric nervous system, different regulatory mechanisms (synaptic and intrinsic neuronal properties, neuromodulation and gene expression regulation) influence each other to produce stable neuronal circuits.

The *C. elegans* connectome is considered in [34] as a multiplex network, with each node representing a neuron and with different layers of connection (synaptic and neuromodulatory). The authors found highly significant multilink motifs of

interaction between the extrasynaptic and synaptic connectomes, detecting locations in the network where the monoamines and neuropeptides modulate synaptic activity.

A simplified multilayer adaptive network model that accounts for these extra-layers of interaction can be represented as follows:

(i) A number N of neurons and a number L of layers. Each neuron is replicated in the rest of the layers, but with a different associated dynamics. An adjacency tensor $M(t)$ with components $m_{i\alpha}^{j\beta}(t)$ encodes the relationships from any neuron i in layer α to any neuron j in layer β.

(ii) The first layer is the neurotransmission layer. Each neuron in this layer has a state $s_{i1}(t)$. An evolving set of synapse weights is represented by $m_{i1}^{j1}(t)$.

(iii) The remaining layers are neuromodulatory layers. Each neuron in one of this layer has a state $s_{i\alpha}(t)$ with $\alpha \in \{2, ..., L\}$. An evolving set of neuromodulatory link weights is represented by $m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$.

(iv) The states in the first layer $s_{j1}(t)$ are a function of both the sum of incoming synapses $\sum_{i=1}^{N} m_{i1}^{j1}(t)s_{i1}(t)$ and the sum of incoming interactions from neuromodulatory nodes $\sum_{\alpha=2}^{L} m_{j\alpha}^{j1}(t)s_{j\alpha}(t)$.

(v) The synapse weights in the first layer $m_{i1}^{j1}(t)$ are a function of both $s_{i1}(t)$ and neuromodulatory link weights in the remaining layers $\epsilon_\alpha m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, where ϵ_α is a coupling parameter between neuromodulatory and neurotransmission links.

(vi) The states in the remaining layers $s_{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, are a function of both the sum of incoming neuromodulatory links $\sum_{i=1}^{N} m_{i\alpha}^{j\alpha}(t)s_{i\alpha}(t)$ and $m_{j1}^{j\alpha}(t)s_{j1}(t)$.

(vii) The neuromodulatory link weights $m_{i\alpha}^{j\alpha}(t)$, $\alpha \in \{2, ..., L\}$, are a function of $s_{i\alpha}(t)$.

In this directed network, the first layer is a typical neurotransmission layer with adaptive node states and synapses, where the link weights $m_{i1}^{j1}(t)$ represent classical fast-synaptic connections. The remaining layers contain adaptive neuromodulatory states and links, where $m_{i\alpha}^{j\alpha}(t)$ are neuromodulatory link weights within layers. The interaction between layers is of three types: node states in the first layer are influenced by the states of the same node in the neuromodulatory layers via $m_{j\alpha}^{j1}$. Second, synapse weights in the first layer are influenced by the same link weights in the neuromodulatory layers via ϵ_α. Third, node states in the neuromodulatory layers are influenced by the state of the same node in the first layer via $m_{j1}^{j\alpha}$. Adjacency weights are different from 0, $m_{i\alpha}^{j\beta} \neq 0$, when $\alpha = \beta$ or when $\alpha \neq \beta$, $i = j$, and $\alpha = 1$ or $\beta = 1$.

Within the above setting, one can formulate detailed biological models (e.g., Hodgkin–Huxley [35]) as well as more abstract models (e.g., McCulloch–Pitts [36]). For example, in an abstract model the output of a neuron in the neurotransmission layer is given by the weighted sum of its inputs. The neuromodulatory layers determine how synaptic weights are updated. In such a model, neuromodulation can change how a circuit function is achieved. The variables of a model are fixed by the level of biological detail (concentration, membrane potential, state of the neuron, etc.).

Regarding the dynamics, the evolution of a network is mostly described by differential equations (Hodgkin–Huxley, Fitzhugh–Nagumo, ...) or by difference equations (logistic maps, ...) depending on the kind and purpose of the model; see, e.g., [37,38].

As a final remark, by examining basic aspects of the multilayer adaptive models being considered, several insights can be extracted. If neuromodulatory layers do not depend on neurotransmission activity, then we have extrinsic neuromodulation. In intrinsic neuromodulation, neuromodulatory layers are not isolated from neurotransmission activity. It is also interesting to compare the temporal scale of the neurotransmission layer with that of the neuromodulatory layers. In most cases, neuromodulation is a slow process compared to neurotransmission [39].

In the next section, we consider the importance of multilayer adaptive models to analyze the computational capabilities of neuromodulators and to provide a complete description of neuronal processing beyond the connectome.

5 Computational capabilities of the multilayer connectome

As pointed out above, it is necessary to take into account the different layers of neuronal communication to complete the connectome, both functional and structural. To this end, multilayer adaptive networks build the appropriate framework to analyse how these layers interact to produce neuronal processing. Next we focus on the computational capabilities added by these interactions at the circuit level. The computational capabilities listed below can be materialized with two interacting layers, one for neurotransmission and the other one for neuromodulation. Other works have focused on aspects of neuromodulators more related to behaviour [40–42]. Among those capabilities, we underline the following:

(i) The different time scales between neurotransmission and neuromodulatory layers give rise to interesting phenomena. For example, neuromodulatory activity in a slower process may fine-tune some properties of the neurotransmission layer such as neuronal excitability and synapse dynamics.

(ii) Because neuromodulation occurs in a diffusive manner, the same neuromodulatory layer can tune different and isolated neuronal circuits (neurotransmission layers).

(iii) Neuromodulatory layers can improve the functioning of neuronal circuits. For example, in [43] the authors found that the neuromodulators trigger distinct changes in representations (through the synaptic weights) that improve the networks classification performance. In [44] the authors demonstrate that neuromodulation of a single target neuron may dramatically alter the performance of an entire network or have almost no effect depending on the state of the network.

(iv) Neuromodulatory layers can ensure a reliable neuronal circuit function despite changes in the parameters of the neurotransmission layer. These extra layers regulate the correlation between parameters, providing robustness to their variation [45,46].

(v) Neuromodulatory layers can reconfigure the neuronal circuit and even change the function performed. This provides a very useful circuit adaptability to the environment.

(vi) Neuromodulators possibly regulate the storage of new information in neuronal networks. Neuromodulatory layers can maintain memory and activity after reconfiguration of neuronal networks [47,48].

These facts show once more the limitations of the connectome to predict the output of the circuit and the need to extend the connectome with additional layers of

neuromodulation that interact dynamically to reconfigure the connectome at every moment.

This new approach will call for network theory and dynamical systems, along with large computational resources, but it will be essential to understand the complexity of the brain. Moreover, the study of neuronal circuits using multilayer adaptive networks will provide clues and evidence of why the processing of information in the brain is more complex and varied than the one observed in artificial neural networks.

6 Conclusions and future work

Important as is the study of structural and functional connectome networks, it is even more important to consider the additional neuromodulatory layers that condition and reconfigure the connectome. In line with the complex cognitive needs of the brain, there is no universal neural coding-decoding scheme but rather different layers of processes that add capabilities to information processing.

As we have seen in this paper, multilayer adaptive networks are an appropriate framework to study neuronal processing and to take into account all the communication processes that occur beyond neurotransmission (for example neuromodulation). Further work is still needed on concrete biological models of interesting phenomena via the multilayer approach.

Recent research has challenged the current premises in memory formation. The use of new techniques such as optogenetics has made it possible to differentiate between mechanisms of memory retrieval and memory storage [49–51]. New theoretical and computational models, such as the one described here, will be required to explain these new observations.

More generally, the use of multilayer adaptive networks for the study of neuronal circuits will lead to analyze the computational capabilities added by the additional layers. These computational capabilities (e.g., adaptability, regulation, robustness, degeneracy, memory, and recurrency) will be key to understand the particularities of information processing in the brain and relate them to those of computers.

Furthermore, cognitive scientists are calling for greater integration between neuroscience and artificial intelligence. This enlarged framework will allow to establish synergies and points in common between neuronal processing and current techniques such as machine learning. Machine learning needs new approaches to imitate how the brain learns and operates and, therefore, it may be relevant to consider how neuromodulation and biochemical communications condition the processing of information.

We thank very much our referees for their constructive criticism. This work was financially supported by the Spanish Ministry of Economy, Industry and Competitivity, grant MTM2016-74921-P (AEI/FEDER, EU).

Author contribution statement

A.H wrote the draft. J.M.A. revised the draft. Both authors discussed the contents and agreed on the final version.

References

1. O. Sporns, G. Tononi, R. Kötter, PLoS Computat. Biol. **1**, e42 (2005)
2. A.M. Zador, J. Dubnau, H.K. Oyibo, H. Zhan, G. Cao, I.D. Peikon, PLoS Biol. **10**, e1001411 (2012)

3. A. Horn, D. Ostwald, M. Reisert, F. Blankenburg, NeuroImage **102**, 142 (2014)
4. D.J. Felleman, D.C. Van Essen, Cereb. Cortex **1**, 1 (1991)
5. W. Gerstner, W. Kistler, R. Naud, L. Paninski, *Neuronal Dynamics* (Cambridge University Press, Cambridge, UK, 2014)
6. C. Bargmann, E. Marder, Nat. Methods **10**, 483 (2013)
7. C. Bargmann, BioEssays **34**, 458 (2012)
8. V. Brezina, Philos. Trans. R. Soc. B **365**, 2363 (2010)
9. E. Marder, Neuron **76**, 1 (2012)
10. D. Bucher, E. Marder, Cell **155**, 482 (2013)
11. J.G. White, E. Southgate, J.N. Thomson, S. Brenner, Philos. Trans. R. Soc. Lond. B **314**, 1 (1986)
12. L.R. Varshney, B.L. Chen, E. Paniagua, D.H. Hall, D.B. Chklovskii, PLoS Computat. Biol. **7**, e1001066 (2011)
13. S. Achard, E. Bullmore, PLoS Computat. Biol. **3**, e17 (2007)
14. A.P. Alivisatos, M. Chun, G.M. Church, R.J. Greenspan, M.L. Roukes, R. Yuste, Neuron **74**, 970 (2012)
15. A.F. Alexander-Bloch, P.E. Vértes, R. Stidd, F. Lalonde, L. Clasen, J. Rapoport, J. Giedd, E.T. Bullmore, N. Gogtay, Cereb. Cortex **23**, 127 (2013)
16. S.T. Baker, D.I. Lubman, M. Yücel, N.B. Allen, S. Whittle, B.D. Fulcher, A. Zalesky, A. Fornito, J. Neurosci. **35**, 9078 (2015)
17. L. Barnett, C.L. Buckley, S. Bullock, Phys. Rev. E **79**, 051914 (2009)
18. V. Nicosia, P.E. Vértes, W.R. Schafer, V. Latora, E.T. Bullmore, PNAS **110**, 7880 (2013)
19. E.K. Towlson, P.E. Vértes, S.E. Ahnert, W.R. Schafer, E.T. Bullmore, J. Neurosci. **33**, 6380 (2013)
20. F. Nadim, D. Bucher, Curr. Opin. Neurobiol. **29**, 48 (2014)
21. F. Fröhlich, *Network Neuroscience* (Academic Press, London, 2016)
22. A. Fornito, A. Zalesky, E. Bullmore, *Fundamentals of Brain Network Analysis* (Academic Press, London, 2016)
23. H. Sayama, I. Pestov, J. Schmidt, B.J. Bush, C. Wong, J. Yamanoi, T. Gross, Comput. Math. Appl. **65**, 1645 (2013)
24. O.V. Maslennikov, V.I. Nekorkin, Physics-Uspekhi **60**, 694 (2017)
25. M. Wiedermann, J.F. Donges, J. Heitzig, W. Lucht, J. Kurths, Phys. Rev. E **91**, 052801 (2015)
26. T. Aoki, L.E.C. Rocha, T. Gross, Phys. Rev. E **93**, 040301 (2016)
27. T. Aoki, K. Yawata, T. Aoyagi, Phys. Rev. E **91**, 012908 (2015)
28. M. Kivelä, A. Arenas, M. Barthelemy, J.P. Gleeson, Y. Moreno, M.A. Porter, J. Complex Netw. **2**, 203 (2014)
29. M. De Domenico, C. Granell, M.A. Porter, A. Arenas, Nat. Phys. **12**, 901 (2016)
30. M. De Domenico, Gigascience **6**, 1 (2017)
31. G. Menichetti, D. Remondini, P. Panzarasa, R.J. Mondragón, G. Bianconi, PLoS One **9**, e97857 (2014)
32. N. Kopell, H.J. Gritton, M.A. Whittington, M.A. Kramer, Neuron **83**, 1319 (2014)
33. N. Daur, F. Nadim, D. Bucher, Curr. Opin. Neurobiol. **41**, 1 (2016)
34. B. Bentley, R. Branicky, C.L. Barnes, Y.L. Chew, E. Yemini, E.T. Bullmore, P.E. Vértes, W.R. Schafer, PLoS Computat. Biol. **12**, e1005283 (2016)
35. A.L. Hodgkin, A.F. Huxley, J. Physiol. **117**, 500 (1952)
36. W.S. McCulloch, W. Pitts, Bull. Math. Biophys. **5**, 115 (1943)
37. X. Li, Q. Chen, F. Xue, Philos. Trans. R. Soc. A **375**, 20160286 (2017)
38. O.V. Maslennikov, D.S. Shchapin, V.I. Nekorkin, Philos. Trans. R. Soc. A **375**, 20160288 (2017)
39. J.M. Fellous, C. Linster, Neural Comput. **10**, 771 (1998)
40. A.J. Yu, Computational models of neuromodulation, in *Encyclopedia of Computational Neuroscience*, edited by D. Jaeger, R. Jung (Springer, New York, 2014)
41. K. Doya, Nat. Neurosci. **11**, 410 (2008)
42. K. Doya, Neural Netw. **15**, 495 (2002)

43. R. Holca-Lamarre, J. Lücke, K. Obermayer, Front. Comput. Neurosci. **11**, 54 (2017)
44. G.J. Gutierrez, E. Marder, eNeuro **1**, ENEURO.0009-14 (2014)
45. T. O'Leary, A.H. Williams, J.S. Caplan, E. Marder, PNAS **110**, E2645 (2013)
46. E. Marder, M.L. Goeritz, A.G. Otopalik, Curr. Opin. Neurobiol. **31**, 156 (2015)
47. J.A. Reggia, E. Ruppin, D. Glanzman (eds.), *Disorders of Brain, Behavior and Cognition: the Neurocomputational Perspective* (Elsevier Science, Amsterdam, 1999)
48. R. Chaudhuri, I. Fiete, Nat. Neurosci. **19**, 394 (2016)
49. S. Tonegawa, M. Pignatelli, D.S. Roy, T.J. Ryan, Curr. Opin. Neurobiol. **35**, 101 (2015)
50. T.J. Ryan, D.S. Roy, M. Pignatelli, A. Arons, S. Tonegawa, Science **348**, 1007 (2015)
51. H.K. Titley, N. Brunel, C. Hansel, Neuron **95**, 19 (2017)