UNIVERSITÀ DEGLI STUDI DI SIENA

Dipartimento di Ingegneria dell'Informazione
e Scienze Matematiche

Local Propagation in Neural Network Learning by Architectural Constraints

Matteo Tiezzi

Ph.D Thesis in Information Engineering and Science

Supervisor
Prof. Marco Maggini

Siena, 2020

Abstract

A crucial role for the success of the Artificial Neural Networks (ANN) processing scheme has been played by the feed-forward propagation of signals. The input patterns undergo a series of stacked parametrized transformations, which foster deep feature extraction and an increasing representational power. Each artificial neural network layer aggregates information from its incoming connections, projects it to another space, and immediately propagates it to the next layer.

Since its introduction in the '80s, BackPropagation (BP) is considered to be the "de facto" algorithm for training neural nets. The weights associated to the connections between the network layers are updated due to the backward pass, that is a straightforward derivation of the chain rule for the computation of the derivatives in a composition of functions. This computation requires to store all the intermediate values of the process. Moreover, it implies the use of non-local information, since the activity of one neuron has the ability to affect all the subsequent units up to the last output layer.

However, learning in the human brain can be considered a continuous, life-long and gradual process in which neuron activations fire, leveraging local information, both in space, e.g neighboring neurons, and time, e.g. previous states.

Following this principle, this thesis is inspired by the ideas of decoupling the computational scheme, behind the standard processing of ANNs, in order to decompose its overall structure into local components. Such local parts

are put into communication leveraging the unifying notion of "constraint". In particular, a set of additional variables are added to the learning problem, in order to store the information on the status of the constrained neural units. Therefore, it is possible to describe the computations performed by the network itself guiding the evolution of these auxiliary variables via constraints. This choice allows us to setup an optimization procedure that is "local", i.e., it does not require (1) to query the whole network, (2) to accomplish the diffusion of the information, or (3) to bufferize data streamed over time in order to be able to compute gradients. The thesis investigates three different learning settings that are instances of the aforementioned scheme: (1) constraints among layers in feed-forward neural networks, (2) constraints among the states of neighboring nodes in Graph Neural Networks, and (3) constraints among predictions over time.

Acknowledgements

. . . , here we go again. Three years after the last time, keeping the tradition of writing overnight. Maybe because it is the most inspiring moment, the silence broken solely by the typewriting on the keyboard. Nothing seems to have changed, but in reality lots of things have been overthrown, revolutionized.

This last part has been tough. I am not referring uniquely to the writing of this dissertation, but the overall final period. Decisions to be taken, deadlines to be respected, audience to be satisfied. But fortunately, this whole path and the people I have met in the last years have trained me to cope with it and have supported me in all the possible (both physically and, recently, *remotely*) ways.

Firstly, this whole experience was possible thanks to my advisor Marco Maggini, to his readiness, the last-hour corrections, his clever comments and his decision to talk to me about that Machine Learning scholarship, almost four years ago, on the fourth floor. It all started in that way, and I was not even remotely imagining how powerful that choice could affect my life. A decision that could not have been possible also without Prof. Trentin, his slides, his ability to impressively teach AI and tell strange stories. I owe to him (and to the *lana caprina*) my interest in this amazing discipline. Other thanks to Marco Gori, to his warm welcome into our SAILab, his stunning speech ability and the insightful talks, the powerful phone calls. Thanks to the other Prof. of the lab, Michelangelo for his multiple incredible abilities, Monica (thanks indeed.) and especially to Franco Scarselli, his kindness, visionary mind and his attention to details in our discussions on GNNs.

I want to reserve a special thanks to Stefano Melacci, without whom many of the work I have done during these three years would not have been possible. Thanks for being always ready to help, to give me insightful comments, to teach me how a researcher must behave, for the post-calls. I always try to follow the suggestion I received from Achille, *learn as much as you can from Stefano*. I will try my best.

I surely will remember these years for the challenges, struggles, late night work (nothing

new), but also for the amazing girls and guys I met in the lab. Thanks to Alessandro Betti for his intuitions, the consequent hardships, the final rewards. Thanks, with absolutely no order, to Lisa for always being there from the first day, to Luca for all the discussions and advises, for being a friend, thanks to Alessandro, Sara, Vincenzo, Simone and Paolo for their availability, courtesy, the tales and the laugh, for the whole time spent together. Thanks to Gabriele, for the questions and insightful doubts and for the amazing trip to NY. Thanks to Enrico for being my new code mate, for the memes, for the adversities and bugs we will defeat together in the next years. Furthermore, I am grateful to have met amazing friends as Francesco, always ready to help, to advise me (also songs), to guide me from the first day at the lab; as Dario, for our laughs, his *attention* to details, for being a charismatic steady point in the lab, for Re Mida and the interview; as Giuseppe, always inspiring for his astonishing extra gear, for all the fruitful discussions, collaborations and fun, for the beautiful experience of his wedding with Maria.

Then I want to thank some old great friends that are travelling with me, since many years, along the strenuous manifold of life. Thanks to Alberto for the many laughs, videos, car travels, nights out, code debugging, for his intuitions and for asking me to work together on GNNs during the first PhD year. Thanks to Marco for always being ready to support me (and I hope to have done the same for him, I always will) even if we are far apart. For always being there for the big decisions. And I want to thank Andrea, an amazing friend, a brilliant researcher, a great roommate. Thanks for the talks, all the messages, the dinners, the laughs and the advises, for all this period. Furthermore, I am not able to express in simple words my appreciation to Giorgia, for being so important to me during this last period. Thanks for the wonderful talks, for the walks and all, for the time spent together. For showing me my weaknesses and for looking after (and to) me. Everyday. Thanks.

Concluding this three years recap, this time I will try not to be too sentimental with my family, hopefully without tear drops. Even if actually all the thanks I can imagine are not enough, given the support, affection (and many other kind words) my parents, my awesome brother and my grandma give me daily (even if I am not always accessible). Thanks for being always with me. Thanks for this wonderful travel.

This PhD experience have brought me many new amazing experiences, discoveries, situations, fun and distress. My main regret is the fact that one of the greatest experiences I had, coincided and made impossible for me to give a proper farewell to an important person of my life. The time just flew away. I will try my best to pursue my path in order to not make that missed greeting worthless.

Ave atque vale.

Siena, *December 2020*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A peculiar characteristic of complex cognitive systems is the ease with which they are able to face challenging tasks, such as humans do. Recognizing an object, understanding the dynamics of a scene or disentangling the factors that identify an item are easy tasks for adult humans, who have been subject to a gradual process of learning and experience accumulation during their lifetimes. Even without the explicit definition of rules that a pattern must follow or other kind of symbolic knowledge, humans are able to classify, recognize and discern the objects and entities in the environment around them, solely by exploring and learning from experience.

Developing artificial cognitive models able to even remotely reproduce such advanced properties is hard. Actually, old-fashioned pattern recognition techniques rely on a careful hand-crafted engineering process, aimed at designing proper feature extractors. The discovery of effective data characteristics, with the purpose of finding an useful representation to discern the data at hand, is solely depending on the designer's experience and predetermined preprocessing techniques. Notwithstanding, the increasing amount of data and complexity in most of the practical applications, along with the need of a good level of generalization to unseen cases, bring a undeniable difficulty in the design of robust and effective features for complex problems.

The recent success of Deep Learning comes also from its capability of learning data representations directly from raw data, automatically discovering useful features to represent data for the task at hand (Representation Learning), while learning a proper predictive model. The previously mentioned difficulties in the design of appropriate data characteristics have been in fact sidestepped by learning the representation itself. Deep Neural Networks [3] represent the most successful example of this direction. Starting from the

raw input, many non-linear modules are stacked in layers, each of which transforms its input to a more abstract and complex representation, trained in an end-to-end fashion. The ubiquity of Deep Learning usages in modern technology is evidence of the impressive power of this approach. Autonomous driving [4], healthcare [5] and medicine [6], intelligent assistants guided by Natural Language [7], even particle physics [8, 9] and many other fields have been hit by this novel promising field of research.

The success of Deep Learning can be ascribed also to the user-friendliness and availability of several Mathematical and Machine Learning frameworks [10, 11, 12, 13, 14], and the advantages brought by Automatic Differentiation, i.e. derivative computation exploiting a view of the problem as a computational graph. These frameworks, and the big tech companies beyond their development, are playing a key role in the widespread diffusion of Deep Learning.

At their core, Deep Learning models leverage the computational power of artificial neural networks, models inspired by previous attempts to find a mathematical representation of the information processing steps occurring in biological neurons [15, 16, 17, 18]. Indeed, since their foundations machine learning and theoretical neuroscience were highly intertwined, often falling under the same umbrella of Cybernetics [19, 20], which formalized biological findings (feedforward processes, negative feedback [21]) exploiting mathematical tools. The Cybernetics view on a common computational framework, held by both animals and machines, frames perception and control in terms of probabilistic models and negative feedback, hence using errors as an input to correct a system.

The computational processing scheme shared by most commonly used artificial neural networks, is the *feed-forward* one. Indeed, the network flow of information can be represented through a directed acyclic (DAG) computational graph, involving the neural units and the synaptical weights connecting them, as shown in Figure 1.1.

Given this representation, first a *forward* flow of information takes place, processing the input pattern starting from the roots (input layer) and ending on the leaves (output layer), going through a series of differentiable operations, giving the final predictions of the network. Learning happens leveraging large

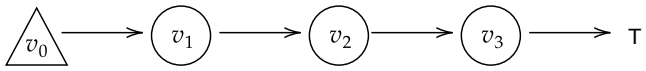

Figure 1.1: The computational graph definition of Artificial Neural Networks.

scale datasets and a procedure that aims at minimizing an error function of the network's prediction with respect to the given target output value. The most commonly used training algorithms share the same pipeline to find a minimum of the error function, defined by a two-stage process [22]. In the first one, the derivatives of the error function with respect to the synaptical weights, the connections among layers, are evaluated. In the latter one, the derivatives are used to compute the adjustment to the architecture parameters. In this phase, starting from simple techniques such as gradient descent [23, 24] several powerful optimization schemes have emerged [3]. Going back to the the first stage, derivatives are computed exploiting Automatic Differentiation. Using the chain rule of calculus, it is possible to undergo a message passing scheme to propagate the error *backward*, differentiating through the computational graph. The instantiation of Automatic Differentiation in artificial neural networks is known as BackPropagation [25, 26, 23](see Figure 1.2).

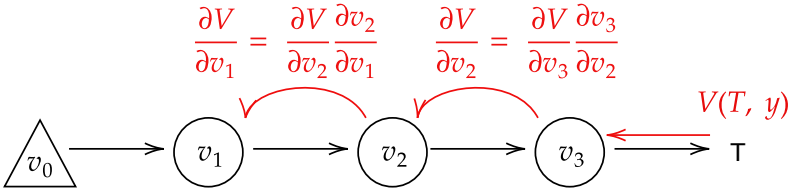

Figure 1.2: BackPropagation on the computational graph of ANNs exploiting the Chain Rule (in red).

Consequently, in BackPropagation the activation of each unit affects all the descendant neurons in the computational graph. Hence, the weight updates are non-local and rely on upstream layers. The downside of these solutions are high memory consumption, to store intermediate values, and the need to

obey to the aforementioned sequential nature of computations.

1.1 Motivations

Despite the previously highlighted promising characteristics, the ability of the human cognitive system is still far away in many aspects. Deep Neural Networks still lack in term of conceptual abstraction, causal or relational reasoning as long as many other characteristics of *intelligent* behaviour.

However, whilst the study of human neurological patterns can foster advances in the computational processes behind artificial intelligence and neural networks, the actual success of deep learning is a direct consequence of mathematical solutions. Indeed, the chance to develop a mathematical superstructure, inspired by intuitions similar to the biological ones, could be capable to grasp the underlying laws of the cognitive processes guiding human-like intelligence. This direction could leverage the advances in computational systems modeling and, likewise, it could be prone to the injection of interesting and beneficial solutions or shortcuts.

Following the previously presented principles, this dissertation is inspired by a recent line of research that describes learning using the unifying mathematical notion of "constraint" [27, 28]. This line of research lays the foundations for a comprehensive theory for the design of intelligent agents, giving a broader view of learning. In particular, the intuition comes from the consideration that just like humans, intelligent agents live and interact with an environment that imposes the fulfillment of several constraints. Therefore, it is possible to build a computational model of learning based on this mathematical notion. This approach foster the model ability to process higher level concepts, and in particular to handle the *injection of external knowledge on the domain* of the task-at-hand.

Complex environments can be modeled as a set of constraints, which the agent is expected to satisfy. As introduced in [29] and then systematically studied in [30], the theory of *Learning from constraints* takes inspiration from principles of cognitive development and stage-based learning. An agent goes through a first training stage considering only the supervised examples, i.e. sub-symbolic information provided as an *input to ouput* point–wise constraint.

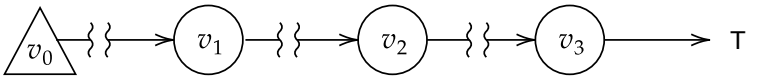

Figure 1.3: The decomposition of the computational graph into local components, connected via constraints.

In this way, the agent is expected to develop some kind of abstract and higher level cognitive capabilities, following a completely data–driven mechanism. Hence, in principle the agent could be capable to process more abstract and symbolic knowledge, injected into the models through a *semantic-based regularization*, in the form of constraints to be fulfilled. The symbolic prior knowledge injected in the second stage via constraints is generally represented by logic clauses [31, 30].

The unifying concept of constraint is broadly general and able to manipulate various kinds of knowledge in different tasks and contexts. For instance, they have been used to enforce the probabilistic normalization when learning density distributions or functions modeling a classification task [32], or to impose consistency of the classifier outputs either in the case of different views of the same object [33] or in the case of correlations in time [34]. As it become clear from these examples, all these methods leverage constraints in order to incorporate domain knowledge into the neural models.

The contributions, presented in this thesis, explore a novel path in the direction of learning via the unifying mathematical notion of constraints. The main intuition is the ability to decompose the neural architectures into *local components*, i.e. subparts constituting the overall architecture, as depicted in Figure 1.3.

Rather than leveraging the constraints to express and inject external knowledge onto the domain of the task-at-hand, the intuition is to exploit constraints to force internal knowledge onto the structure of the neural models. The learning problem is enriched with auxiliary local variables whose evolution is constrained to follow the neural computational scheme. Thus, constraints are the mathematical tool leveraged to put into communication the obtained local structures. In other words, the computation performed by the network is

partially described by *structural constraints*.

This choice allows us to setup an optimization procedure that is "local", i.e., it does not require (1) to query the whole network, (2) to accomplish the diffusion of the information, or (3) to bufferize data streamed over time in order to be able to compute gradients.

Learning is the outcome of the interaction of data and constraints, that describe the dependencies in the neural computation. Hence, the proposed technique can be summarized by the definition *Learning by Constraints*.

1.2 Research questions and contributions

In the following Chapters, given the background overview seen so far, the thesis will explore three different learning settings, seeking to answer the following research questions. These questions are not intended to be self-contained and are characterized by concepts that will be expanded in the next chapters.

In particular, three different learning settings that are instances of the aforementioned scheme will be investigated: (1) constraints among layers in feed-forward neural networks, (2) constraints among the states of neighboring nodes in Graph Neural Networks (GNNs), (3) constraints among predictions over time.

1. *BackPropagation has become the de-facto algorithm for training neural networks. Despite its success, the sequential nature of the performed computation hinders parallelizations capabilities and causes a high memory consumption. Is it possible to devise a novel computational method for a generic Directed Acyclic Graph that gets inspiration and advantages from principles of locality?*

   Constraint-based Neural Networks In the proposed approach, *Local Propagation*, DAGs can be decomposed into local components. The processing scheme of neural architecture is enriched with auxiliary variables corresponding to the neural units, and therefore can be regarded as a set of constraints that correspond with the neural equations. Constraints enforce and encode the message passing scheme among neural units, and in particular the consistency between the input and the output variables by means of the corresponding weights of the synaptic connections. The

proposed scheme leverages completely local update rules, revealing the opportunity to parallelize the computation.

2. *The seminal Graph Neural Networks [35] model uses an iterative convergence mechanism to compute the fixed-point of the state transition function, in order to allow the information diffusion among long-range neighborhoods of a graph. Is it possible to avoid such costly procedure maintaining these powerful aggregation capabilities?*

   Constraining the Information Diffusion in Graph Neural Networks The original GNN model[35] encode the state of the nodes of the graph by means of an iterative diffusion procedure that, during the learning stage, must be computed at every epoch, until the fixed point of a learnable state transition function is reached, propagating the information among the neighbouring nodes. *Lagrangian Propagation GNNs* decompose this costly operation, proposing a novel approach to learning in GNNs, based on constrained optimization in the Lagrangian framework. Learning both the transition function and the node states is the outcome of a joint process, in which the state convergence procedure is implicitly expressed by a constraint satisfaction mechanism, avoiding iterative epoch-wise procedures and the network unfolding.

3. *Unsupervised learning from continuous visual streams is a challenging problem that cannot be naturally and efficiently managed in the classic batch-mode setting of computation. Lifelong learning suffers from the problem of catastrophic forgetting [36]. Hence, the task of transferring visual information in a truly online setting is hard. Is it possible to overcome this issue by devising a local temporal method that forces consistency among predictions over time?*

   Constraint-based Mutual Information Computation in Salient Areas of Video Streams We consider the problem of transferring information from an input visual stream to the output space of a neural architecture that performs pixel-wise predictions. This problem consists in maximizing the Mutual Information (MI) index. Most approaches of learning commonly assume uniform probability density of the input. Actually, devising an appropriate spatio-temporal distribution of the vi-

sual data can foster the information transfer. In the proposed approach, a human-like focus of attention model takes care of filtering the spatial component of the visual information, restricting the analysis on the salient areas. On the other side, various temporal locality criteria can be explored. In particular, the analysis sweeps over the probability estimates obtained in subsequent time instants. Global changes in the entropy of the output space are approximated by introducing a specific constraint. The probability predictions obtained at each time instant can once more be regarded as *local components*, that are put into relation by soft-constraints enforcing a temporal estimate not limited to the current frame.

To sum up, the proposed scheme decomposes neural architectures into local subparts, i.e. neural network neurons (or layers), node states updates in graphs, or time instant predictions, obtained via the introduction of local auxiliary variables. Constraints are the mathematical tool leveraged to put into communication these components, connecting units, aggregating neighboring nodes in graphs, forcing temporal entropy predictions, respectively.

In such a way, several advantages like parallelization and locality of computations are obtained, besides the ability to avoid costly procedures without sacrificing representational capabilities.

1.3 Thesis structure

In the following Chapters the thesis will try to answer the research questions described in Section 1.2, preceded by a general description of the theoretical foundations.

1. Chapter 2 gives a broad introduction to all the concepts this dissertation is based on. Some insight on artificial neural architectures are followed by an introduction to optimization schemes, with a focus on BackPropagation. Moreover, we recall some backgrounds of a theoretical framework for BackPropagation, as long as constrained differential optimization based on the Lagrangian Framework. This principles will establish the foundation for the following Chapters.

2. In Chapter 3, constrains are defined among layers in feed-forward neural networks and potentially over arbitrary Directed Acyclic Graphs, obtaining the *Local Propagation* optimization procedure. *Based on [37].*

3. In Chapter 4, the iterative diffusion mechanism defining the message passing scheme of Graph Neural Networks is defined via constraints. *Based on [38, 39]*

4. Finally, in Chapter 5 a lifelong online learning process in visual streams is empowered by constraints among predictions over time. *Based on [40]*

1.4 List of Publications

In the following, a list of the research contributions produced during the period of this research is provided.

Peer reviewed conference papers

1. **Tiezzi, M.**, Melacci, S., Betti, A., Maggini, M., and Gori, M. (2020). "Focus of Attention Improves Information Transfer in Visual Features". *Advances in Neural Information Processing Systems, 33.* (**NeurIPS 2020**)
   Candidate Contribution: joint definition of the method, joint design and implementation of both the framework and the experiments.

2. Meloni, E., Pasqualini, L., **Tiezzi, M.**, Gori, M., and Melacci, S. (2020). "SAILenv: Learning in Virtual Visual Environments Made Simple". In *International Conference on Pattern Recognition* (**ICPR 2020**)
   Candidate Contribution: definition of the theory, joint definition of the method, design and implementation of the experiments.

3. **Tiezzi, M.**, Marra, G., Melacci, S., Maggini, M., and Gori, M. (2020). "A Lagrangian Approach to Information Propagation in Graph Neural Networks". In *European Conference of Artificial Intelligence* (**ECAI 2020**)

Candidate Contribution: joint definition of the theory, joint definition of the method, design and implementation of both the framework and the experiments.

4. Marra, G., **Tiezzi, M.**, S. Melacci, A. Betti, Maggini M. and M. Gori. (2020) "Local Propagation in Constraint-based Neural Networks". In *International Joint Conference on Neural Networks* (**IJCNN2020**): 1-8.
Candidate Contribution: formulation of the solution, design and development of the experimental campaign

5. **Tiezzi, M.**, Melacci, S., Maggini, M., and Frosini, A. (2018, October). "Video Surveillance of Highway Traffic Events by Deep Learning Architectures". In *International Conference on Artificial Neural Networks* (**ICANN 2018**) (pp. 584-593). Springer, Cham.
Candidate Contribution: joint formulation of the problem, formulation of the solution and design of the experimental campaign.

Workshop papers

1. **Tiezzi, M.**, Marra, G., Melacci, S., Maggini M. (2020) "Deep Lagrangian Propagation in Graph Neural Networks". In *Graph Representation Learning and Beyond* (**GRL+ ICML 2020** Workshop) .
Candidate's contributions: joint definition of the theory, joint definition of the method, design and implementation of both the framework and the experiments.

2. **Tiezzi, M.**, Marra, G., Melacci, S., Maggini M., and Gori M. (2020) "Lagrangian Propagation Graph Neural Networks" In *Deep Learning on Graphs: Methodologies and Applications* (**DGLMA- AAAI 2020** Workshop) .
Candidate's contributions: joint definition of the theory, joint definition of the method, design and implementation of both the framework and the experiments.

3. Zugarini A., **Tiezzi, M.**, Maggini M., (2020) "Vulgaris: Analysis of a Corpus for Middle-Age Varieties of Italian Language", *Seventh Workshop on NLP for Similar Languages, Varieties and Dialects*(**VarDial**

2020),pages:150–159,
Candidate's contributions: corpus creation, joint statistical analysis of the corpus, joint design of the experiments

4. Rossi, A., **Tiezzi, M.**, Dimitri, G. M., Bianchini, M., Maggini, M., and Scarselli, F. (2018, September). "Inductive–transductive learning with graph neural networks". In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition* (**ANNPR 2018**)(pp. 201-212). Springer, Cham.
Candidate Contribution: joint design, development and testing of the overall framework

Papers under review

1. **Tiezzi, M.**, Marra, G., Melacci, S., Maggini M. (2020) "Deep Constraint-based Propagation in Graph Neural Networks". **Submitted at TPAMI**
Candidate's contributions: joint definition of the theory, joint definition of the method, design and implementation of both the framework and the experiments.

Chapter 2

Neural Network Architecture as Constraints

In the context of Deep Learning, artificial neural architectures are characterized by a computational structure that can be generally decomposed into local subcomponents, at different granularity levels. The atomic computational component is the neuron itself, but, at a coarse grain level, appropriate aggregations of neural units, such as layers, can be considered as building blocks of a complex architecture. The idea is to describe the interactions of these blocks thanks to the introduction of local auxiliary variables and the mathematical tool of constraint, which is leveraged in order to define and describe the connections among such submodules. This intuition imply several advantages, allowing to express a local processing scheme in several architectures (Chapter 3), to avoid costly iterative procedures (Chapter 4) or to better estimate certain temporal quantities (Chapter 5). The backbone of this approach is the ability to describe the flow of information happening inside computational models by means of constraints.

In this Chapter we will establish a common notational framework, provide the necessary background and give a comprehensive overview of the needed background methods. In what follows, we will give an introduction to several Artificial Neural Networks (ANNs) in Section 2.1 and the learning procedure commonly used in ANNs in Section 2.2. Afterward, a brief introduction on the Lagrangian formalism for constrained Optimization will be given in Section 2.4, followed by the description of methods to carry on optimization in this context in Section 2.6, and by insights on a Lagrangian derivation of BackPropagation in Section 2.5.

2.1 Artificial Neural Architectures

Artificial Neural networks (ANN) are machine learning models born as a byproduct of studies on the information processes happening in biological neurons [15, 16, 17, 18]. At their core, they are composed of simple interconnected processing units known as *artificial neurons*, connected by *synaptic weights*, which are commonly denoted with the symbol $\boldsymbol{w}$ and represent learnable parameters defining the overall computation. The final goal is the approximation of some function of the input x, a mapping $y = f(x; \boldsymbol{w})$, where the parameters $\boldsymbol{w}$ are learned in order to achieve the best input to output fitting on a set of given examples. Such mapping is obtained via the composition of many simple functions, defined by each artificial neuron. The overall computation emerges from the flow of information in the network of neurons, given the pattern of connections between them and the synaptic weights assigned to such connections.

Generally speaking, the flow of information happening inside ANNs can be described in terms of a computational graph $\mathcal{G} = (V, E)$, a Directed Acyclic Graph that describes any program or computable function as a composition of elementary functions. The computation consists in a data flow through the edges E, undergoing several subsequent numerical transformations. Each node $v \in V$ represents an input (scalar, vector, matrix, tensor, etc.) or an intermediate variable (neural activation) obtained by applying elementary operations (e.g. non-linearities, tensor multiplications) to the values available at another node connected to v by an arc $e \in E$. A set of neurons residing at the same depth – layer – of the data-flow are denoted by the same node v_i (see top of Figure 2.1). Multi-layered ("deep") neural networks are graphs having more than two of such layers.

Several neural architectures have been developed in order to deal with different tasks. In the following, we will briefly review the main models that will be used in the dissertation.

2.1.1 Multi-Layer Perceptrons

MLPs represent a powerful instance of the previously mentioned computational DAG, as depicted in Figure 2.1. In particular, it is a directed feed-

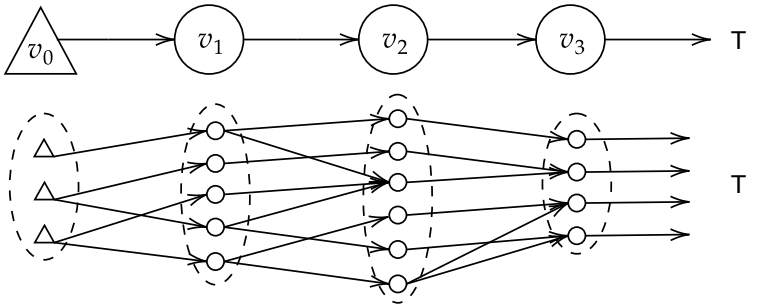

Figure 2.1: Instance of the computational graph (top) as a Multi–Layer Perceptron (bottom). A set of neurons at the same depth – layer – of the data-flow are denoted by the same node v_i. To ease the plot comprehension, we depict an MLP with sparse connections among neurons.

forward data-flow graph. Each artificial neuron receives an input vector $\boldsymbol{x} \in \mathbb{R}^n$, which may be a task related fixed-size external input or the intermediate outputs of the set of neurons in the parent nodes of the computational graph. Given this vector, the neuron computes a linear combination of its components parametrized by a weight vector $w \in \mathbb{R}^n$ and it finally adds a bias value b. This procedure computes the so called *neural activation*, on which an *activation function* $\sigma(\cdot)$ can be applied, in order to obtain the unit output $y \in \mathbb{R}$:

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b) = \sigma(W x^T) \tag{2.1}$$

where W and x collect the connection weights and the input variables, respectively, including also the bias in W in correspondence to an additional input entry in x set to a constant value of 1.

MLPs assume a feedforward structure in which several neural units sharing the same input compose a layer, and several of such layers are stacked, yielding the overall architecture. The term "feedforward" refers to the fact that data flow in the same direction, processed in a synchronous way, starting from the *input layer* up to the *output layer*, through the *hidden layers*. In general each unit populating a layer is connected to all the neural units composing

the following one, in a *fully-connected* topology. Given a vectorial form of the input signal x_0, where the subscript denotes the layer index, the computational graph can be translated into a series of dense matrix multiplications, regarded as *matrix form* of the computation. The output of a generic hidden layer $\ell \in [1, H]$ is indicated with x_ℓ, that is a column vector with a number of components equal to the number of hidden units in such layer. We also have that

$$x_\ell = \sigma(W_{\ell-1} x_{\ell-1}) \tag{2.2}$$

where $\sigma(\cdot)$ is the activation function that is intended to operate element-wise on its vectorial argument (we assume that this property holds in all the following functions). The matrix $W_{\ell-1}$ collects the weights linking layer $\ell - 1$ to ℓ. We avoid introducing bias terms in the argument of $\sigma(\cdot)$, to simplify the notation (they can be included in the considered variables as stated before).

Activation functions

In literature, several activation functions $\sigma(\cdot)$ have been introduced. This dissertation exploits the following mappings, depicted in Figure 2.2:

- **Logistic Sigmoid** – A Monotonic continously differentiable mapping from $\mathbb{R}$ to $[0, 1]$
$$\sigma(a) = \frac{1}{1 + e^{-a}} \tag{2.3}$$
- **Hyperbolic Tangent** – A Sigmoidal shape mapping from $\mathbb{R}$ to $[-1, 1]$
$$\sigma(a) = \tanh(a) \tag{2.4}$$
- **Rectified Linear** – A mapping defined as the positive part of its argument
$$\sigma(a) = \max(0, a) \tag{2.5}$$

2.1.2 Recurrent Neural Networks

Whilst MLPs are designed to process fixed sized inputs, several problems require an architecture able to handle streams of data, potentially having

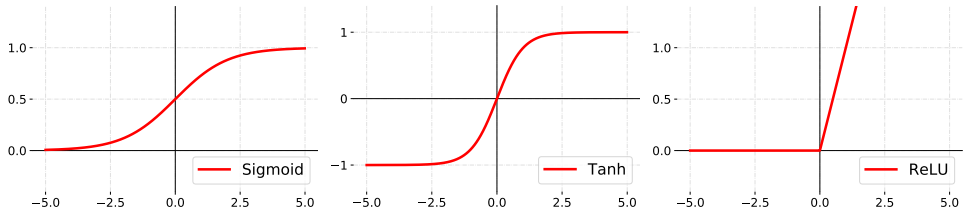

Figure 2.2: The mapping defined by the Sigmoid, Hyperbolic Tangent (Tanh) and Rectified Linear Unit (ReLU) activation functions.

variable-length. The main idea behind Recurrent Neural Networks (RNNs) [41, 42] is to extend neural networks to sequentially structured data, by generating a dynamic computational graph capable to unroll on the length of each individual input sequence. The unfolding of the computational graph allows us to compute a hidden representation for each time step, i.e. node, based on the current input and the node's previous state.

Formally speaking, given an input sequence $\left(x_0^{(0)}, x_0^{(1)}, .., x_0^{(T)}\right)$ composed by T steps, $x_0^{(i)}$ denotes the input at the i-th time step. A layered RNN computes the state vector for layer ℓ at time step t as:

$$x_\ell^{(t)} = \sigma(W_{\ell-1} x_{\ell-1}^{(t)} + U_\ell x_\ell^{(t-1)}) \tag{2.6}$$

where U_ℓ is the matrix of the weights that tune the contribution of the state at the previous time step. The initial state $x_\ell^{(0)}$ is generally set to zero. As can be seen from Eq. (2.6), this model exploits a *temporal weight sharing*, meaning that the same learnable parameters are exploited at the different time steps.

2.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [43] are feed-forward architectures exploiting peculiar operations in one or more sub-parts of the computational graph, in particular *convolution* and *pooling*.

The main reason behind the introduction of the convolution operation is the ability to apply the same parametrized function - called *kernel* - to different areas of the input, in order (1) to deal with high dimensional data

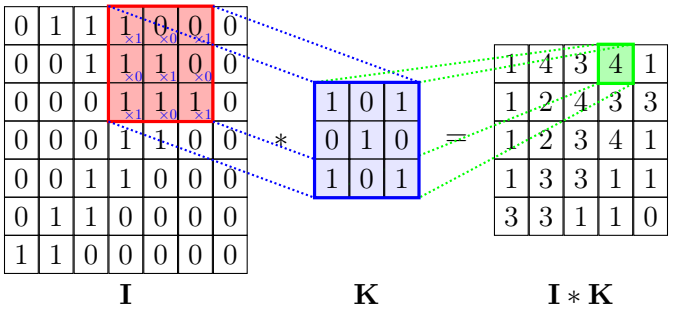

Figure 2.3: 2D convolution of an input tensor I and a kernel K (Figure thanks to Petar Veličković).

and (2) to extract common patterns from heterogeneous locations of the input raster. Indeed, processing an input signal (e.g. an image) encoded by a tensor of dimension $c \times h \times w$ (number of channels, height and width of the image, respectively) is an expensive task for a common MLP, requiring full connectivity between each pixel (input element) and every hidden unit, resulting in a complexity of $\mathcal{O}(c \times h \times w)$ for each hidden unit. Moreover, a fully connected network will be needed to learn translation invariance from the examples, whereas the convolution operation can guarantee this property a priori. In fact, thanks to the kernel sharing property, convolutional layers are *equivariant to translation*, practically meaning that they are able to produce the same output when detecting the same pattern in different locations.

In the simplest case the convolution operation can be defined for an 1-dimensional vector input $x_0 \in \mathbb{R}^n$. In this case, the convolutional kernel is a vector $K \in \mathbb{R}^k$ of learnable parameters, where $k < n$, that slides along the input with a given *stride*. At each position, the kernel is multiplied element-wise with the k input components around the current element, then summed up producing an aggregated scalar value. The newly obtained values compose a *feature map*. Generally, many Kernels slide over the same locations, obtaining an output composed by many channels. This same concept can be easily extended to multi-dimensional tensors, for instance an input image $X \in \mathbb{R}^{c \times h \times w}$[43]. See Figure 2.3 for a graphical representation. Typically, non-linearities are applied after convolutional layers.

Another relevant component of CNNs are *Pooling* layers, that aggregate (summing up, averaging or extracting the maximum) the values computed by the convolution filters in small sub-regions (e.g. 2×2) of the input feature map to yield a single output. Pooling performs a sub-sampling in the feature maps and, hence, helps reducing the potentially redundant information and the memory consumption, allowing to increase the channel size to obtain a higher representational power in higher level features. Moreover, by pooling the CNN gains the property of *translation invariance* to small displacements that is useful if we care more about the presence of some features rather than their exact location.

Residual Networks

With very deep neural networks the increased representational power comes at the cost of a higher difficulty in the training procedures. Indeed, stacking many layers of computation interleaved with sigmoidal activation functions $\sigma(\cdot)$, hinders the backpropagation of error signals, as we will see in Section 2.2, a problem known with the term *vanishing gradient*.

ResNets [44] consist of several stacked residual units, that are robust with respect to the vanishing gradient problem. In particular, this solution has been shown to attain state-of-the art results in Deep Convolutional Neural Nets [44] (without being limited to such networks).

The most generic structure of a residual network [45] is defined by layers whose outputs are computed by

$$x_\ell = z(h(x_{\ell-1}) + f(W_{\ell-1}x_{\ell-1})) \ . \tag{2.7}$$

In [44], $z(\cdot)$ corresponds to a rectifier (ReLu) and $h(\cdot)$ is the identity function, while $f(\cdot)$ is a non-linear function. The role of the residual or skip connection is to create an alternative path, a shortcut $h(x_{\ell-1})$, for the flow of information coming from the previous layer. In this way, this information can directly be propagated, whereas the role of the current layer becomes to solely learn a residual function $f(W_{\ell-1}x_{\ell-1})$.

2.1.4 Graph Neural Networks

The architectures described in the previous sections have been devised to extend the representation capability of ANNs beyond flat vectors of features. Recurrent Neural Networks [41, 42] have been proposed to process sequences, Convolutional Neural Networks to deal with groups of adjacent pixels [46].

However, several tasks require to deal with data that exhibit a complex structure, for instance data can be provided as entities and relations among them. Such data can generally be represented by a graph $G = (V, E)$, where V is a finite set of *nodes* and $E \subseteq V \times V$ collects the *arcs*, representing relations between an ordered pair of nodes $(i, j) \in V \times V$.

Several attempts were made to deal with such Non-Euclidean structures [47], also restricting the problem domain to directed acyclic graphs (Recursive Neural Networks [48, 49]).

The term Graph Neural Network (GNN) refers to a general computational model, that exploits the inference and learning schemes of neural networks to process non Euclidean data, organized as graph structures. GNNs, with a procedure that resembles Recursive networks, dynamically unfold their computational graph on the topology of the whole input structure. They are able to take into account both the local information attached to each node and the whole graph topology. GNNs can implement either a *node-focused* function, where an output is produced for each node of the input graph, or a *graph-focused* function, where the representations of all the nodes are aggregated to yield a single output for the whole input graph.

Message Passing Neural Networks

Inspired by the original model [35], Graph Representation Learning is becoming one of the hottest topic in Deep Learning, thanks to hundreds of works in this direction. The work by Gilmer [50, 51] is an effort to unify this entire movement under the common framework of *Message Passing NNs* (MPNNs). Each node i is characterized by an initial set of features, denoted by l_i (label in the original model). The same holds for an arc connecting node i and j, whose feature, if available, is denoted with $l_{(i,j)}$. Each node i has an associated hidden representation (or state) $x_i \in \mathbb{R}^s$, which in modern models is initialized with the initial features, $x_i = l_i$. Such state is updated through a message

passing scheme among neighboring nodes, denoted by $\mathcal{N}_i$ or $ne[i]$, followed by an aggregation of the exchanged information, to obtain the updated node representation $x_i^{(t+1)}$:

$$x_{(i,j)}^{(t)} = \texttt{MSG}^t(x_i^{(t)}, x_j^{(t)}, l_{(i,j)}) \tag{2.8}$$

$$x_i^{(t+1)} = \texttt{AGG}^t(x_i^{(t)}, \sum_{j \in \mathcal{N}_i} x_{(i,j)}^{(t)}, l_i) \tag{2.9}$$

where $x_{(i,j)}$ is an explicit edge representation, i.e. the exchanged message, computed by a learnable mapping $\texttt{MSG}^t(\cdot)$. The mapping $\texttt{AGG}^t(\cdot)$ aggregates the messages from incoming edges to node i, exploiting also local node information (state and features). The functions $\texttt{MSG}^t(\cdot)$ and $\texttt{AGG}^t(\cdot)$ are typically implemented via MLPs, whose parameters (weights) are learned from data, and shared among nodes in order to gain generalization capability and save memory consumption. Generally these two functions are not shared between message passing steps, hence $\texttt{MSG}^t(\cdot)$ denotes the message function implemented at time step t. This choice entails both a higher representational capability and an increased memory and computational complexity in the prediction. Such steps are commonly referred to as layers, hence an ℓ-step message passing scheme can be seen as a ℓ-layered graph network. Having ℓ-layers, such model is able to directly propagate a message up-to an ℓ-hop distance. The representation of each node at the top layer represents the node-level output of the GNN.

The Graph Neural Network Model

For the purpose of this dissertation we focus our attention on the seminal model by Scarselli et al. [35], which can be seen as a particular instantiation of MPNNs with peculiar characteristics. The node states $x_i \in \mathbb{R}^s$ are an additional variable to the problem, independent from the node initial features $l_i \in \mathbb{R}^m$. The node states are zero-initialized, and the node state update function, denoted with the term *state transition function* $f_a(\cdot)$, is conditioned on neighboring states x_j with $j \in \mathcal{N}_i$ and local node features. As described from the MPNNs guidelines, the state of node i is the outcome of the iterative application of the *shared* state transition function computation, which processes

Method	Implementation of $f_a(\cdot)$
Sum	$f_{a,v}^{(\text{SUM})} = \sum_{u \in ne[v]} h(x_u, l_u, l_v, l_{(u,v)} \mid \theta_h)$
Average	$f_{a,v}^{(\text{AVG})} = \frac{1}{\lvert ne[v] \rvert} \sum_{u \in ne[v]} h(x_u, l_u, l_v, l_{(u,v)} \mid \theta_h)$

Table 2.1: Common implementations of the state transition function $f_a()$. The function $h()$ is implemented by a feedforward neural network with s outputs, whose input is the concatenation of its arguments.

information attached to each node and to its neighborhood. However, differently from *layered*-MPNNs, in this case the same state transition function is used for every node and for every iteration or aggregation step. The final purpose is the computation of a vector embedding for each node

$$x_v^{(t+1)} = f_a(x_{ne[v]}^{(t)}, l_{ne[v]}, l_v, l_{(ne[v],v)} | \theta_{f_a}) \tag{2.10}$$

where ne$[v]$ are the *neighbors* of v and $l_v, l_{ne[v]} \in \mathbb{R}^m$, $l_{(\text{ne}[v],v)} \in \mathbb{R}^d$ encode additional information (sometimes referred as *labels*) on the node v, on its neighbors and on the arcs connecting them. The vectors θ_{f_a} collect the function parameters (i.e. the weights of the neural networks implementing the functions) that are adapted by the learning procedure.

It should be noted that the state transition function $f_a(\cdot)$ may depend on a variable number of inputs, given that the nodes $v \in V$ may have different degrees de$[v] = |\text{ne}[v]|$. It is for this reason that the authors proposed the implementations reported in Table 2.1 and represented in Figure 2.4, that inspired the arc-level message passing scheme of MPNNs.

The $h(\cdot)$ function, commonly implemented by an MLP, acts at arc-level receiving as input the concatenation of its arguments (for example, in the first case the input consists of a vector of $s + 2m + d$ entries, with $l_{(u,v)} \in \mathbb{R}^d$ and $l_u \in \mathbb{R}^m$). The plain aggregation of the arc-level messages yields the node state at the current iteration. Another interesting property of such functions is the invariance with respect to permutations of the nodes in ne$[v]$, unless some predefined ordering is given for the neighbors of each node.

Once the state transition function application has been iterated T times, GNNs employ a *readout* output function, whose nature depends on the task at hand, by leveraging the final node states $x_i^{(T)}$, as

$$y_v = f_r\left(x_v^{(T)} \mid \theta_{f_r}\right), \tag{2.11a}$$

$$y_G = f_r\left(\{x_v^{(T)}, v \in V\} \mid \theta_{f_r}\right), \tag{2.11b}$$

where y_v is the node level output in the node-focused case, whereas y_G is the graph level output in the graph-focused case. The vectors θ_{f_r} collect the weights of the neural networks implementing the readout function. The state transition function $f_a(\cdot)$ is recursively applied on the graph nodes, yielding an information diffusion mechanism whose range depends on T. In fact, by stacking t times the aggregation of 1-hop neighborhoods by $f_a(\cdot)$, a node is able to directly receive messages coming from nodes that are distant at most t-hops.

To draw a parallel with MPNNs, the number t may be seen as the *depth* of the GNN and thus each iteration can be considered a different *layer* of the GNN. In this case, however, all the layers (i.e., iterations) share the same $f_a(\cdot)$. Furthermore, this same function is computed leveraging at each time step the initial node features, potentially avoiding oversmoothing or washing away node feature information [52].

Given these considerations, a sufficient number of layers seems the key to produce a node state informed of the whole graph topology. In the original GNN model [53], Eq. (2.10) is iterated until convergence of the state representation, i.e. until $x_v^{(t)} \simeq x_v^{(t-1)}, \forall v \in V$. This scheme corresponds to the computation of the *fixed point* of the state transition function $f_a(\cdot)$ on the input graph. As stated by the *Banach fixed-point theorem*, in order to guarantee the convergence of this phase, the transition function is required to be a *contraction map* [54].

2.2 Learning by BackPropagation

Deep Neural Nets are usually trained leveraging a procedure that aims at optimizing a differentiable *loss function*, denoted by $V(\cdot)$, which gives a measure

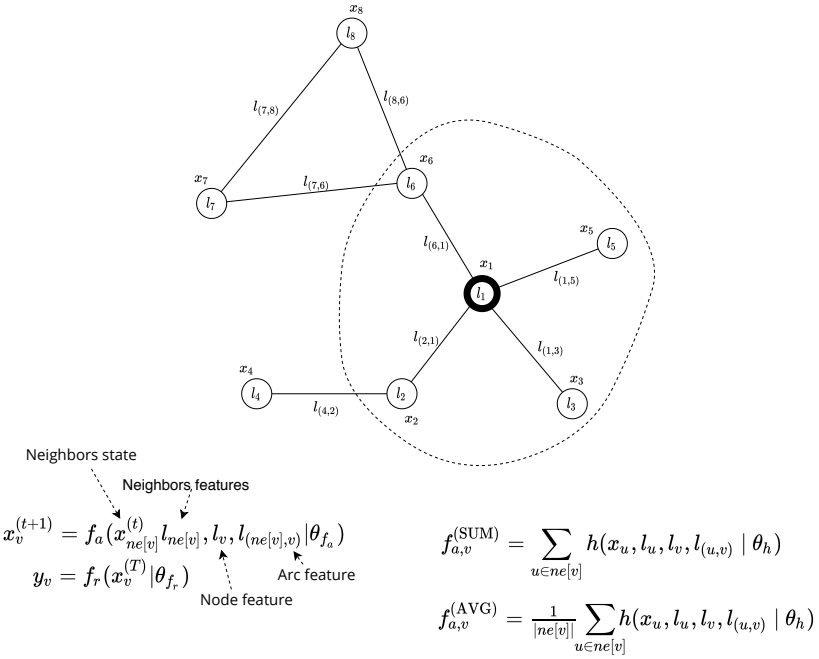

Figure 2.4: The Graph Neural Network model.

of how good are the prediction capabilities of the current model.

As briefly mentioned in Chapter 1, the common training pipeline is composed by a two-stage process aiming at the optimization of the loss function [22]. In the former phase the derivatives of the error function with respect to the weights, which we denote with $\nabla_w V$, are evaluated. In the latter phase, advanced optimization schemes, inspired by gradient descent [23, 24], are employed to adjust the network parameters towards values yielding an improvement of the performances [3].

The first stage of this process, i.e. the parameters derivative computation, leverages Automatic Differentiation techniques and the data-flow computational model of ANNs presented in Section 2.1. The instantiation of Automatic Differentiation in neural networks is known as BackPropagation [25, 26, 23], which has become the "de-facto" standard to compute derivatives (as sketeched in Figure 2.5).

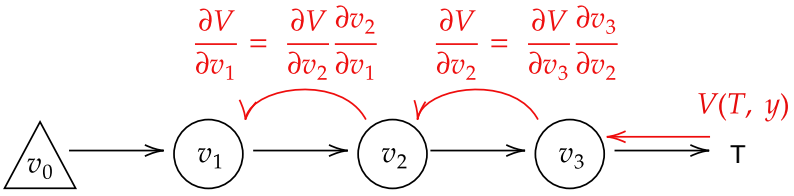

Figure 2.5: BackPropagation on a generic computational graph exploiting the Chain Rule (in red).

In the following, without any loss of generality, we will show the derivation of the BackPropagation algorithm (BP) in a *supervised setting*, where we are given N supervised pairs $(x_{0,i}, y_i)$, $i = 1, \ldots, N$, where $x_{0,i}$ and y_i denote the input features and the target values of the i-th example, respectively. Moreover, we consider a Multi Layer Perceptron (MLP) with H hidden layers as an instantiation of a generic neural architecture described by a Directed Acyclic Graph (DAG), following the notation introduced in Section 2.1.

If we denote the prediction of the model for the pattern i with with $y'_i = \sigma(W_H x_{H,i})$, the function $V(y_i, y'_i)$ computes the loss on the i-th supervised pair, and, when summed up for all the N pairs, it yields the objective function that is minimized by the learning algorithm. In the case of classic neural networks, the variables involved in the optimization are the weights W_ℓ, $\forall \ell$.

Without any loss of generality, we will refer to the Squared Error cost function:

$$V_{\text{SE}} = \frac{1}{2}\sum_{i=1}^{N}(y_i - y'_i)^2 \tag{2.12}$$

The goal is the computation of the derivative of the loss function with respect to the weights. Firstly, we consider the evaluation of the loss function derivative with respect to the weight matrix W_H, collecting the weights linking layer H to the output layer, which will be denoted as $H + 1$.

$$\frac{\partial V}{\partial W_H} = \sum_{i=1}^{N}\frac{\partial(\frac{1}{2}(y_i - y'_i)^2)}{\partial W_H} = -\sum_{i=1}^{N}(y_i - y'_i)\frac{\partial y'_i}{\partial W_H} \tag{2.13}$$

The last term can be computed exploiting the *chain rule*

$$\frac{\partial y_i'}{\partial W_H} = \frac{\partial \sigma(W_H x_H)}{\partial W_H} = \frac{\partial \sigma(W_H x_{H,i})}{\partial W_H x_{H,i}} \frac{\partial W_H x_{H,i}}{\partial W_H} = \sigma'(W_H x_{H,i}) x_{H,i}^T \tag{2.14}$$

Hence, we obtain

$$\frac{\partial V}{\partial W_H} = -\sum_{i=1}^{N} (y_i - y_i') \sigma'(W_H x_{H,i}) x_{H,i}^T \tag{2.15}$$

We introduce the notation

$$\delta_{H+1} = (y_i - y_i') \sigma'(W_H x_{H,i}) \tag{2.16}$$

where δ_i are commonly referred to as *errors*. Substituting it into Eq. (2.15), we obtain

$$\frac{\partial V}{\partial W_H} = -\sum_{i=1}^{N} \delta_{H+1} x_{H,i}^T \tag{2.17}$$

Subsequently, we evaluate the derivative of the loss function with respect to the weights of the underneath layer, W_{H-1}, obtaining:

$$\frac{\partial V}{\partial W_{H-1}} = \sum_{i=1}^{N} \frac{\partial (\frac{1}{2}(y_i - y_i')^2)}{\partial W_{H-1}} = -\sum_{i=1}^{N} (y_i - y_i') \frac{\partial y_i'}{\partial W_{H-1}} \tag{2.18}$$

We evaluate the derivative of the last term using once again the chain rule

$$\frac{\partial y_i'}{\partial W_{H-1}} = \frac{\partial \sigma(W_H x_{H,i})}{\partial W_H x_{H,i}} \frac{\partial W_H x_{H,i}}{\partial W_{H-1,i}} = \sigma'(W_H x_{H,i}) W_H^T \odot \frac{\partial x_{H,i}}{\partial W_{H-1,i}} \tag{2.19}$$

where, exploiting one last time the chain rule:

$$\frac{\partial x_{H,i}}{\partial W_{H-1,i}} = \frac{\partial \sigma(W_{H-1} x_{H-1,i})}{\partial W_{H-1,i}} = \frac{\partial \sigma(W_{H-1} x_{H-1,i})}{\partial W_{H-1} x_{H-1,i}} \frac{\partial W_{H-1} x_{H-1,i}}{\partial W_{H-1}} = \\ = \sigma'(W_{H-1} x_{H-1,i}) x_{H-1,i}^T$$

Substituting into Eq. (2.18) we obtain

$$-\sum_{i=1}^{N}(y_i - y_i')\sigma'(W_H x_{H,i})W_H^T \odot \sigma'(W_{H-1}x_{H-1,i})x_{H-1}^T \tag{2.20}$$

Leveraging Eq. (2.16) we define the *error* of the current layer as

$$\delta_H = (\delta_{H+1}W_H^T) \odot \sigma'(W_{H-1}x_{H-1,i}) \tag{2.21}$$

and finally we get, substituting into Eq. (2.20)

$$\frac{\partial V}{\partial W_{H-1}} = -\sum_{i=1}^{N}\delta_H x_{H-1}^T \tag{2.22}$$

This results can be generalized to the derivative computation of a generic layer ℓ :

$$\begin{aligned} \frac{\partial V}{\partial W_{\ell-1}} &= -\sum_{i=1}^{N}\delta_{\ell,i}\cdot x_{\ell-1,i}^T, \\ \delta_{\ell,i} &= \sigma'(W_{\ell-1}x_{\ell-1,i}) \odot \left(W_\ell^T \delta_{\ell+1,i}\right), \end{aligned} \tag{2.23}$$

that are the popular equations for updating weights and the Backpropagation deltas.

From Eq. (2.23) it is quite clear the dependence of the BP algorithm on the sequential nature of the computations. The weight derivatives are obtained thanks to the backward propagation of the *errors* δ_i from the uppermost layers. In order to compute the derivatives of the loss function with respect to the early layers weights, it is necessary to backpropagate the errors through all the subsequent layers. Thus, in this sense BP is a *Non-local* algorithm.

The derivatives of the network parameters are exploited into the second stage of the learning process mentioned at the beginning of this section, the update of the network parameters. Techniques such as Stochastic Gradient Descent (SGD) operate in learning epochs. In each of them, firstly a *mini-batch*, a subset of training examples, is used to evaluate the loss function and then compute all the weight gradients through BP. Afterwards, the model

parameters are updated moving towards the minima of the loss function, along the direction of the descent of the gradient :

$$\Delta \mathcal{W}^{t+1} = \mathcal{W}^t - \eta \nabla_{\mathcal{W}^t} V(\cdot) \tag{2.24}$$

where η denotes the update pace or *learning rate* [22].

2.2.1 Learning in Structured Networks

The end-to-end differentiation capability characterizing the BackPropagation algorithm fosters its application to several kinds of architectures and methods. While the CNNs data-flow can be simply translated into a feed-forward one and directly dealt with by an extension of BP, that takes into account the propagation of the errors through the pooling operator, the application to structured dynamical architectures such as RNNs or GNNs is not straightforward. In the context of recurrent structures, the recursive definition of the update equation introduces loops into the network computational graph, that is in practice unrolled in time along the input sequential structure to be processed. This method allows us to define the BackPropagation Through Time (BPTT) algorithm [18]. The recurrent layer is unfolded many times as the number of time steps to be processed, with each step sharing the same parameters. For this reason, the total gradient is given by accumulating of the gradients computed at each time step.

The same consideration holds for the GNN model [35], whose convergence procedure up to the fixed point of the state transition function shares a dynamical unrolling procedure similar to that of RNNs, backpropagating the gradient errors on the GNN unfolded following the graph topology. For this reason this model in recent surveys falls under the umbrella of Recurrent GNNs (RecGNNs).

One interesting consideration in the context of learning in very deep networks via BackPropagation is the *vanishing gradient* problem. Applying recurrent architectures to long sequences implies the creation of very deep unrolled networks. When using BPTT, due to the usage of the chain rule and Eq. (2.23), the computation of the gradients at a given time step depends on the propagation of the error through all the subsequent steps, that implies the multiplication by the network output derivative at each step. The usage of

non-linearities, whose output belongs to a limited range $(0, 1)$ with a saturating behaviour, such as the sigmoid, results in the product of many small values, that causes a quick decay of the error signal towards zero. Hence, the term *vanishing gradient* has been used to enlighten the fact that the contribution of early steps of the unrolled computational graph is quite likely to become negligible.

The introduction of the ReLU activation function can alleviate this problem, but also affects very deep networks with the opposite issue of *exploding gradients*, caused by the same gradient product diverging due to the unrestricted function output space.

2.2.2 Complexity Analysis

The success of the Backpropagation algorithm is mostly due to its ability to perform the gradient computation in a very efficient manner. In particular, the forward flow of information inside the model, for a single input pattern, is dominated by the weight matrices products needed for the neural activation computation, resulting in a $\mathcal{O}(|W|)$ complexity.

The same applies for the backward pass, where once again the weight matrices are leveraged in order to backpropagate the δ_ℓ, resulting in the same complexity of $\mathcal{O}(|W|)$. All the competing methods, such as computing derivatives by finite differences, require a greater computational complexity.

In the context of the GNN learning procedure, there are synchronous updates among all nodes and multiple iterations for the node state embedding, with a computational complexity for each parameter update of $\mathcal{O}(T(|V| + |E|))$, where T is the number of iterations, $|V|$ the number of nodes and $|E|$ the number of edges [35].

2.3 Regularization

The usage of mapping functions parametrized by many learnable variables favours the creation of separating hyperplanes that highly adapt to the lower dimensional manifolds populated by the input patterns, somehow memorizing the characteristics of the training data. However, such behaviour may hinder the generalization capabilities of the model. Deep Neural Nets, often

constituted by million of parameters, deeply suffer from this *overfitting* issue, and often the trained models are not capable to generalize satisfactorily their performance to the test data. In order to alleviate overfitting, some *regularization* methods have been devised, that extend the learning problem with additional soft-constraints that guide the optimization process restricting the set of allowed parameters, thus limiting their degrees of freedom.

For the purpose of this Thesis, two methods will be exploited, given the learnable parameter vector w and the loss function $V(\cdot)$:

L_1-norm regularizer Reduces the parameters search space adding a penalty, weighted by $\alpha > 0$, in order to help the model to focus on a smaller number of paths. The loss function is transformed:

$$\hat{V}(\cdot) = V(\cdot) + \alpha\|w\| \tag{2.25}$$

L_2-norm regularizer The parameters are softly enforced to not diverge from zero, avoiding any excessive adaptation to the training data, transforming the loss function:

$$\hat{V}(\cdot) = V(\cdot) + \frac{\alpha}{2}\|w\|^2 \tag{2.26}$$

This regularizer leads to the *weight decay* scheme.

2.4 The Lagrangian formalism

In the previous Sections, various Neural Network architectures and their learning mechanisms have been described via the unifying concept of computational graph model. Each one of the submodules constituting such graph, i.e. each node $v_i \in V$ from $G = (V, E)$, can be seen as an independent component in the data-flow. Therefore, in the following Chapters neural architectures will be treated as a collection of submodules, whose interconnection and processing scheme is defined via constraints. This entails several advantages in the computational, memory and learning point of view. The information data flow happening in the networks computational graph will be defined in the context of constraint satisfaction. In order to deal with constrained optimization, many techniques have been explored [55].

In this section an overview on the Lagrangian formalism of constrained optimization problems will be given. In particular, we will focus on the method of *Lagrange Multipliers*, used to find stationary points of functions defined on variables subject to equality constraints.

Suppose to have a variable $\boldsymbol{x} \in \mathbb{R}^d$ which can be interpreted as a point in a d-dimensional space. A *constraint equation* defined as

$$g(\boldsymbol{x}) = 0 \tag{2.27}$$

represents a $(d-1)$-dimensional surface in the point space. The final goal is to solve the following constrained optimization problem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g(\boldsymbol{x}) = 0 \end{aligned} \tag{2.28}$$

consisting in finding a minimum of a function of the $\boldsymbol{x}$ variable subject to the $g(\boldsymbol{x}) = 0$ constraint. We can derive the Lagrange multiplier method from a geometrical perspective [22], firstly noting that for every point $\boldsymbol{x}$ lying on the surface $g(\boldsymbol{x}) = 0$, the gradient $\nabla g(\boldsymbol{x})$ is orthogonal the surface itself. To prove this, starting from the surface point $\boldsymbol{x}$, we can move along the surface (in a direction parallel to it) by a small step $\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \to 0$. Hence, the new position will be an infinitely close point $\boldsymbol{x} + \boldsymbol{\epsilon}$. We can expand the function by the Taylor series around $\boldsymbol{x}$

$$g(\boldsymbol{x} + \boldsymbol{\epsilon}) \simeq g(\boldsymbol{x}) + \boldsymbol{\epsilon}^T \nabla g(\boldsymbol{x}) \tag{2.29}$$

and considering that both the points lay on the surface, then $g(\boldsymbol{x}) = g(\boldsymbol{x} + \boldsymbol{\epsilon}) = 0$, Eq. (2.29) is reduced to $\boldsymbol{\epsilon}^T \nabla g(\boldsymbol{x}) = 0$. Hence, being $\boldsymbol{\epsilon}$ tangent to the surface, the gradient of the constraint must be orthogonal to it.

The constrained optimization goal is to find a solution to the problem defined in Eq. (2.28), i.e. a point $\boldsymbol{x}$ minimizing $f(\boldsymbol{x})$ while lying on the surface. Suppose that we are on a point $\boldsymbol{x}$ on the constraint surface where the gradient $\nabla f(\boldsymbol{x})$ is not parallel to $\nabla g(\boldsymbol{x})$. This means that there exist an admissible direction where to move in order to minimize the function. Otherwise, if the two gradients share the same direction,

$$\nabla f(\boldsymbol{x}) = \lambda \nabla g(\boldsymbol{x}) \tag{2.30}$$

it is assured that there is no direction causing an improvement in the optimization process. The term λ is known as *Lagrange multiplier* and, if defined over equality constraints, it can assume both positive and negative values.

Hence, the Lagrangian function is defined as

$$\mathcal{L}(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x}) \tag{2.31}$$

such that its extrema are characterized by the condition of null gradient

$$\begin{aligned} &\nabla f(\boldsymbol{x}) + \lambda \nabla g(\boldsymbol{x}) = 0 \\ &g(\boldsymbol{x}) = 0 \end{aligned} \tag{2.32}$$

that requires the satisfaction of the Eq. (2.30) for a point $\boldsymbol{x}$ laying on the constraint surface[1].

Thanks to this approach, a constrained optimization problem can be cast as an unconstrained one, solved seeking for the stationary points of the Lagrangian with respect to the new problem variables, $\boldsymbol{x}$ and λ.

2.5 A Lagrangian framework for BackPropagation

The Lagrangian formalism introduced in the previous section allows us to attain a very interesting derivation of the BackPropagation algorithm, an approach firstly introduced by LeCun [56]. This work nicely intercepts the concepts exploited in this dissertation. BP is formalized as an optimization problem subject to non-linear constraints. The problem will be cast into the Lagrangian framework, with the goal of optimizing a Lagrangian function composed by a cost function and constraints that describe the network dynamics.

In particular, the optimization process involves as usual the architecture weights $W_\ell \ \forall \ell$, and it is based on the addition of new auxiliary variables to the learning problem, corresponding to the neural states $\mathcal{X}$ of the network.

Given an L-layered feed-forward DAG, the neural learning problem can be written as

[1] In this context the sign of λ is changed with respect to the previous equation.

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} V\big(y_i, y_i'\big) \\ \text{subject to} \quad & x_{\ell,i} - \sigma(W_{\ell-1} x_{\ell-1,i}) = 0, \quad \forall (i,\ell) \end{aligned} \tag{2.33}$$

where the constraints define the network data-flow. The newly added variable are constrained to follow the local neural unit computation dynamics. Once the constraint is fulfilled, the forward propagation rule is obtained. If we consider each layer/unit as a local component, the constraints define the global architecture dynamics connecting such submodules. This constrained optimization problem can be cast, following the Lagrangian formalism introduced in Section 2.4, into the unconstrained Lagrangian function

$$\mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda) = \sum_{i=1}^{N} \Big(V(y_i, y_i') + \sum_{\ell=1}^{L} \lambda_{\ell,i}^{T} \big(x_{\ell,i} - \sigma(W_{\ell-1} x_{\ell-1,i}) \big) \Big), \tag{2.34}$$

where $\mathcal{W}$ is the set of all the network weights, $\mathcal{X}$ is the set of all the $x_{\ell,i}$'s variables and Λ collects the Lagrange multipliers (T is the transpose operator).

Once gain, note that when the constraints are fulfilled, the corresponding term is zero, hence yielding the forward propagation step of BP,

$$x_{\ell,i} = \sigma(W_{\ell-1} x_{\ell-1,i}), \quad \forall (i,\ell) \tag{2.35}$$

In this way, constraints define the forward dynamics in the computational graph and describe the dependencies among the involved variables $\mathcal{X}$.

To solve the unconstrained optimization problem, the author [56] points out that a necessary condition defining a local minimum of the loss function is

$$\nabla \mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda) = 0 \tag{2.36}$$

which can be subdivided into the three conditions

$$\frac{\partial \mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda)}{\partial \Lambda} = 0 \tag{2.37}$$

$$\frac{\partial \mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda)}{\partial \mathcal{X}} = 0 \tag{2.38}$$

$$\frac{\partial \mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda)}{\partial \mathcal{W}} = 0. \tag{2.39}$$

Equation (2.37) is decomposed into $N \times L$ conditions ($i \in [1, ..., N]$ and $\ell \in [1, ..., L]$), obtaining

$$\frac{\partial \mathcal{L}}{\partial \lambda_{\ell,i}} = x_{\ell,i} - \sigma(W_{\ell-1} x_{\ell-1,i}) = 0 \quad \forall (i, \ell) \ , \tag{2.40}$$

that yields the forward pass dynamics.

To evaluate the second partial derivative (Eq. (2.38)), we firstly consider the output layer states, $x_{L,i} = \sigma(W_{L-1} x_{L-1,i})$,

$$\frac{\partial \mathcal{L}}{\partial x_{L,i}} = \frac{\partial V(y_i, y_i')}{\partial x_{L,i}} + \lambda_{L,i}^T = 0 \quad \forall (i, \ell) \ . \tag{2.41}$$

The derivative with respect to layers $\ell \in [1, ..., L-1]$ are

$$\frac{\partial \mathcal{L}}{\partial x_{\ell,i}} = \lambda_{\ell,i} - \lambda_{\ell+1,i} \odot W_\ell^T \sigma'(W_\ell x_{\ell,i}) = 0 \quad \forall (i, \ell) \ . \tag{2.42}$$

It is interesting to note that, substituting $\delta_{\ell,i} = \lambda_{\ell,i} \odot \sigma'(W_{\ell-1} x_{\ell-1})$ into Eq. (2.42), we obtain the backward dynamics of the BP algorithm,

$$\delta_{\ell,i} = \sigma'(W_{\ell-1} x_{\ell-1,i}) \odot \left(W_\ell^T \delta_{\ell+1,i} \right) \ . \tag{2.43}$$

Hence, in this context, the Lagrange multipliers $\lambda_{\ell,i}$ can be identified as the *backpropagated gradients*.

Finally, the third condition (Eq. (2.39)) is evaluated for all the layers $\ell \in [0, ..., L-1]$.

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = -\sum_{i=1}^{N} \left(\lambda_{\ell+1,i} \odot \sigma'(W_\ell x_{\ell,i}) \right) x_{\ell,i}^T \quad \forall \ell \ . \tag{2.44}$$

Once again, exploiting the same change of variables $\delta_{\ell,i} = \lambda_{\ell,i} \odot \sigma'(W_{\ell-1} x_{\ell-1})$, we obtain

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = -\sum_{i=1}^{N} \delta_{\ell+1,i} \cdot x_{\ell,i}^T \tag{2.45}$$

that is the BP equation for the weight update.

Summarizing, starting from a constrained problem describing the flow of information happening inside a neural architecture, the evaluation of the associated unconstrained Lagrangian problem makes it possible the derivation of the BP algorithm. In particular, the three subconditions of Eqs. (2.37),(2.38), and (2.39) yield the **forward pass**, **backward pass** and **weight update** equations of the BP algorithm, respectively.

2.6 Constrained differential optimization

In Section 2.4 and 2.5 we have seen how the Lagrangian formalism can be useful in order to face constrained optimization problems and derive the BP algorithm, respectively. However, solving the unconstrained optimization problem defined by the Lagrangian of Eq. (2.31) is not trivial.

An interesting analysis is given in the work by Platt and Barr [1], that analyses the dynamics of a continuous time constrained neural learning system. As noticed by the authors, techniques such as Gradient Descent (GD) do not work with Lagrangian multipliers. In particular, GD applied on the energy in Eq. (2.31) yields the component-wise differential equations

$$\dot{x_i} = -\frac{\partial \mathcal{L}}{\partial x_i} = -\frac{\partial f}{\partial x_i} - \lambda \frac{\partial g}{\partial x_i} \tag{2.46}$$

$$\dot{\lambda} = -\frac{\partial \mathcal{L}}{\partial \lambda} = -g(\boldsymbol{x}) \ . \tag{2.47}$$

Indeed, GD optimizes the learning parameters aiming at reaching the local minima of the energy function, going towards the direction opposite with respect to the gradient. Hence, the stationary points of the learning problem must be attractors of this dynamics. However, the dual nature of the La-

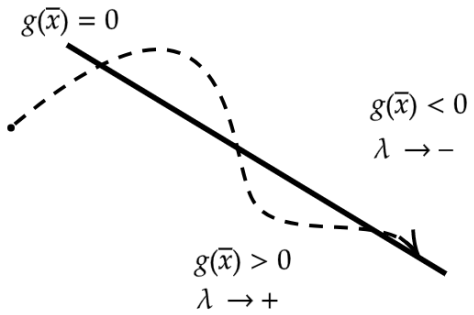

Figure 2.6: In BDMM [1] the state $\boldsymbol{x}$ is attracted toward the constraint subspace. The state slides along the subspace moving to the minima of the function $f(\cdot)$, undergoing damped oscillations dictated by the differential equations.

grangian multipliers causes the critical points to be saddle points, thus hard to reach via GD.

The Basic Differential Multiplier Method (BDMM) [1] is a differential optimization process that can be interpreted as a force that gradually attracts the state $\boldsymbol{x}$ to the subspace $g(\boldsymbol{x}) = 0$, enforcing the constraint satisfaction while reaching the minima of $f(\boldsymbol{x})$ on the subspace itself. A depiction of the process is shown in Fig. 2.6.

The constrained minima become attractors of the differential equations by inverting the sign of Eq. (2.47), as

$$\dot{x_i} = -\frac{\partial \mathcal{L}}{\partial x_i} = -\frac{\partial f}{\partial x_i} - \lambda \frac{\partial g}{\partial x_i} \tag{2.48}$$

$$\dot{\lambda} = +g(\boldsymbol{x}) \tag{2.49}$$

which corresponds to Gradient *Ascent* on the Lagrangian multiplier λ.

The optimization process can be summarized in a minmax problem as

$$\min_{\boldsymbol{x}} \max_{\lambda} \ \mathcal{L}(\boldsymbol{x}, \lambda) \tag{2.50}$$

and easily extended to the case of multiple constraints [1]. Moreover, this differential process can be interpreted, from the physics point of view, as a

damped mass system which, under some conditions, is guaranteed to converge to fulfill the constraints.

Chapter 3

Local Propagation in Neural Networks

Artificial Neural Networks have become extremely popular models, due to their role in several important achievements in the Machine Learning community [57]. If we consider the recent scientific contributions in the field, it is often the case of new neural architectures that are designed to solve the task at hand [58, 59], or of new architectures that are created as alternatives to existing models [60].

The various ANNs models share the same architectural paradigm that can be described by the computational graph scheme and the usage of Automatic Differentiation (BackPropagation) techniques presented in Chapter 2. Such characteristics represent the major components behind their success, but also conceal several drawbacks such as the mandatory sequential nature of computation (both in the forward and the backward phase of BackPropagation) described in Section 2.1, which hinders the parallelization of computations, and causes the problem of vanishing gradients (see Section 2.2.1).

The main idea behind this dissertation naturally deals with such weaknesses, trough the decomposition of the neural architecture processing scheme into local components. In particular, in this Chapter general architectures represented as DAGs are subdivided into local modules, i.e. artificial neurons, whose computational scheme and learning mechanisms are described using the unifying notion of "constraint" [28, 27]. This mathematical tool is leveraged in order to describe the connections among local sub-modules. Moreover, we nicely intercept the work of [56], where a theoretical framework for BackPropagation is studied in a Lagrangian formulation of learning (see Section 2.5). We regard the neural architecture as a set of constraints that correspond with the neural equations and enforce the consistency between the input and the output variables by means of the corresponding weights of the synaptic con-

nections. However, differently from [56], we do not only focus on the derivation of BackPropagation in the Lagrangian framework, and we introduce a novel approach to learning that explores the search in the adjoint space that is characterized by the triple (w, x, λ), i.e., weights, neuron outputs, and Lagrangian multipliers. The locality of the approach is obtained by the introduction of the additional output variables x, that are constrained to be consistent with the neural computation.

The main goal of the proposed approach is not to show improved performance w.r.t. BackPropagation, with which it shares the same Lagrangian derivation, but to propose an optimization scheme for the weights of a neural network, that shows new and promising properties. Indeed, it turns out that the gradient descent w.r.t to the variables (w, x) and the gradient ascent w.r.t to the multipliers λ give rise to a truly *local* algorithm for the variable updates that we refer to as *Local Propagation* (LP). By avoiding long dependencies among variable gradients, this method nicely circumvents the vanishing gradient problem in optimizing neural networks. Moreover, the local nature of the proposed algorithm enables the parallelization of the training computations over the neural units. Finally, by interpreting Lagrange multipliers as the reaction to single neural computations, the proposed scheme opens the door to new methods for automatic architecture design in the deep learning scenario.

After a brief introduction to the related context given in Section 3.1, this Chapter makes three important contributions. First, in Section 3.2 and 3.3 it introduces a local algorithm (LP) for training neural networks described by means of the so-called architectural constraints, evaluating a simple optimization approach, together with the conditions under which we can see the natural connection with BackPropagation. Second, the implementation of popular neural network models is described in the context of LP, in Section 3.4. Third, we investigate the setting in which we tolerate bounded violations of the architectural constraints (Section 3.3.2), and we provide experimental evidence that LP is a feasible approach to train shallow and deep networks in Section 3.5. LP also opens the road to further investigations on more complex architectures, easily describable by means of constraints.

3.1 Related Work

Besides the theoretical analysis carried on in the work by LeCun [56] and presented in Section 2.5, which derives the BackPropagation algorithm via a constrained formulation of learning, several other works share interesting intuitions.

The idea of training ANNs described with constraints and extending the space of learnable parameters has been originally introduced by the method of auxiliary coordinates (MAC) [61], that exploits an optimization scheme based on a quadratic penalty. The approach of [61] is built on the idea of finding an approximate solution of the original learning problem, and it relies on a post-processing procedure that refines the last-layer connections. The related approach of [62] involves closed-form solutions, but most of the architectural constraints are softly enforced, and further additional variables are introduced to parametrize the neuron activations. In particular the authors showed great parallelization capabilities, training in multi-core processors and showing an inverse linear scaling between training time and the number of cores used. Other approaches followed these seminal works to implement constraining schemes for block-wise optimization of neural networks [63]. All these approaches decompose the learning problem into multiple, local subproblems which are efficiently solved without using SGD or Adam. The work by Carreira et al. [61] solves the learning problem via block coordinate descent (BCD), while Taylor et al. [62] leverage the alternating direction method of multipliers (ADMM). These approaches, that introduce auxiliary variables into the learning problem, increase, or *lift*, the dimension of the problem itself. For this reason recent works [64] refer to these models as *Lifted Networks*.

Another interesting line of research tries to find alternative mechanisms for learning in ANNs, often guided by the claim of *biological plausibility*. Undoubtedly, inspiration taken from neuroscience advancements could, in principle, help in developing better artificial architectures. However, the emergence of human intelligence and the brain computational structure can be ascribed also to the need to sustain all the organism essential functionalities, and to a gradual temporal process of learning, that happens in the whole lifetime of each human. It is for these reasons that the claims made by many models in modern literature, commonly consisting in computational graphs trained via

BackPropagation (BP), around *biological plausibility principles* can be considered exaggerated. In this particular context, there are two main problems beyond the biological implausibility of BackPropagation [65, 66].

- **Weight transport problem** – BP relies on identical forward and backwards weights, hence synaptic symmetry on both the paths. This is implausible in biological brains, and known as the weight transport problem [67].

- **Non-local information** – In BackPropagation, each unit activation affects all the descendant neurons of the computational graph. Hence, the weight updates are non-local and rely on upstream layers. It is unclear how this information could be backwardly transmitted throughout the neural synapses. End-to-end propagation of errors is unlikely to occur, implying that local learning is required. In biological neural nets, neurons act solely on the basis of their input neurons activity and on the incoming/outcoming synaptic weights.

With this considerations in mind, some works propose methods dealing with the two aforementioned problems [68, 69]. Several approaches approximate BP using local update rules, such as Target-Prop[70] that estimates backpropagated gradients via parametrized inverse functions. Equilibrium Propagation [71] follows contrastive Hebbian rules in order to be capable to asymptotically estimate BP, with memory and computational drawbacks. Another promising line of research showed that BP in simple MLPs [72] or arbitrary DAGs [66] can be correctly approximated by Predictive Coding, a biologically-plausible theory of the computation happening at cortical level, that relies solely on local and Hebbian updates.

3.2 Constraint-based Neural Networks

In the context of this dissertation, the architectural setting shared by ANNs, that has been introduced in Chapter 2, can be differently expressed by exploiting the mathematical notion of constraints, following the intuition presented in Section 2.5.

As a brief summary of the setting presented in Section 2.1.1, we are given N supervised pairs $(x_{0,i}, y_i)$, $i = 1, \ldots, N$, and we consider a generic neural architecture described by a Directed Acyclic Graph (DAG), that we can instantiate as a Multi Layer Perceptron (MLP) with H hidden layers. The output of a generic hidden layer $\ell \in [1, H]$ for the i-example is indicated with $x_{\ell,i}$ that is a column vector with a number of components equal to the number of hidden units in such layer. We also have that $x_{0,i}$ is the input signal, and $x_{\ell,i} = \sigma(W_{\ell-1}x_{\ell-1,i})$, where $\sigma(\cdot)$ is the activation function that is intended to operate element-wise on its vectorial argument (we assume that this property holds in all the following functions). The matrix $W_{\ell-1}$ collects the weights linking layer $\ell - 1$ to ℓ. We avoid introducing bias terms in the argument of $\sigma(\cdot)$, to simplify the notation. We denote the prediction of the model for the pattern i with $y'_i = \sigma(W_H x_{H,i})$. The function $V(y_i, y'_i)$ computes the loss on the i-th supervised pair, and, when summed up for all the N pairs, it yields the objective function that is minimized by the learning algorithm. In the case of classic neural networks, the variables involved in the optimization are the weights W_ℓ, $\forall \ell$.

We formulate the learning problem decoupling the computation by describing the network architecture with a set of *architectural constraints*. In such a way, each neural computation can be treated as a local component of the overall architecture. The constraints enforce the communication and message exchange among the local neighboring subparts.

In particular, following the same procedure described in Section 2.5, the $x_{\ell,i}$'s become variables of the learning problem, and they are constrained to fulfil the (hard) *architectural constraints* $x_{\ell,i} = \sigma(W_{\ell-1}x_{\ell-1,i})$, i.e., [1]

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} V(y_i, y'_i) \\ \text{subject to} \quad & \mathcal{G}(x_{\ell,i} - \sigma(W_{\ell-1}x_{\ell-1,i})) = 0, \quad \forall (i, \ell) \end{aligned} \tag{3.1}$$

being $\mathcal{G}(\cdot)$ a generic function such that $\mathcal{G}(0) = 0$, and that is only used to differently weight the mismatch between $x_{\ell,i}$ and $\sigma(W_{\ell-1}x_{\ell-1,i})$. The possible choices for the function $\mathcal{G}$ will be discussed in Section 3.3.2. In the following,

[1] Without any loss of generality, we could also introduce the same constraint in the output layer ($\ell = H + 1$).

we will also make use of the notation $\mathcal{G}_{\ell,i}$ to compactly indicate the left-hand side of Eq. (3.1).

In the Lagrangian framework (see Section 2.4), if $\lambda_{\ell,i}$ are the Lagrange multipliers associated to each architectural constraint, then we can write the Lagrangian function $\mathcal{L}$ as

$$\mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda) = \sum_{i=1}^{N} \Big(V(y_i, y_i') + \sum_{\ell=1}^{H} \lambda_{\ell,i}^T \mathcal{G}_{\ell,i} \Big), \tag{3.2}$$

where we only emphasized the dependance on the set of variables that are involved in the learning process: $\mathcal{W}$ is the set of all the network weights; $\mathcal{X}$ is the set of all the introduced $x_{\ell,i}$'s variables; Λ collects the Lagrange multipliers (T is the transpose operator[2]). The Lagrangian $\mathcal{L}$ can also be augmented with a squared L_2 norm regularizer (see Section 2.3) on the network weights, scaled by a positive factor c.

3.3 Local Propagation in Constraint-based Neural Networks

The problem definition described in Eq. (3.2) is the same one outlined in Section 2.5. However, in our approach we do not just exploit the aforementioned problem to derive BP, but we propose a novel local constraint-based optimization process to devise information diffusion in neural architectures. In order to do so, despite the variety of popular approaches that can be used to solve the constrained problem above [55], we decided to exploit the Basic Differential Multiplier Method [1], introduced in Section 2.6. Hence, the Local Propagation (LP) algorithm consists in a "differential optimization" process that enforces the constraints converging towards a saddle point of Eq. (3.2), minimizing it with respect to $\mathcal{W}$ and $\mathcal{X}$, and maximizing it with respect to Λ (see Section 2.6). The whole procedure is very simple, and it consists in performing a gradient-descent step to update $\mathcal{W}$ and $\mathcal{X}$, and a gradient-ascent step to update Λ, until we converge to a stationary point. As it will become

[2]Note that, given the vectorial notation we are using, $\lambda_{\ell,i}$ is a column vector with a number of components equal to the number of hidden units in such layer. Hence, it is constituted by a component for each architectural constraint

clear shortly, each iteration of the optimization algorithm is $\mathcal{O}(|\mathcal{W}|)$ (without considering the number of examples), that is, it exhibits the same optimal asymptotical property of BackPropagation.

We initialize the variables in $\mathcal{X}$ and Λ to zero, while the weights $\mathcal{W}$ are randomly chosen. For this reason, at the beginning of the optimization, the degree of fulfillment is the same for all the architectural constraints $\mathcal{G}_{\ell,i}$, $\ell > 1$, in every unit of all layers and for all examples, while only $V(\cdot,\cdot)$ and $\mathcal{G}_{1,i}$ contribute to the Lagrangian of Eq. (3.2). In fact, only these two local farthest portions of the architecture are influenced by external knowledge, the input pattern and the target, respectively. Afterwards, the message exchange among local components propagates inside the architectures connecting the whole computational structure. In the case of BackPropagation, the outputs of the neural units are the outcome of the classic forward step, while in the training stage of LP the evolution of the variables in $\mathcal{X}$ is dictated by gradient-based optimization. Indeed, in LP the forward propagation itself is the outcome of a constraint-fulfilment process guided by differential optimization. Once LP has converged, the architectural constraints of Eq. (3.1) are satisfied, so that we can easily devise the values in $\mathcal{X}$ with the same forward step of BackPropagation-trained networks. In other words, we can consider LP as an algorithm to train the network weights while still relying on the classic forward pass during inference.

3.3.1 Properties of Local Learning

The LP algorithm is inspired by the main idea underlying this dissertation, that is the locality of computations. The main advantage of such approach, in the context of this Chapter, is reflected on the gradient computations with respect to a certain variable of layer ℓ, that only involve units belonging (at most) to neighbouring layers. This is largely different from the usual case of the BackPropagation (BP) algorithm, where the gradient of the cost function with respect to a certain weight in layer ℓ is computed only after a forward step (that involves all the neural units of all layers), and then by progressively computing the gradients back-propagating layer-wise the errors δ_ℓ (see Eq. (2.23) and Section 2.2) with respect to all the units above ℓ (backward) (see Fig. 3.1 (a)).

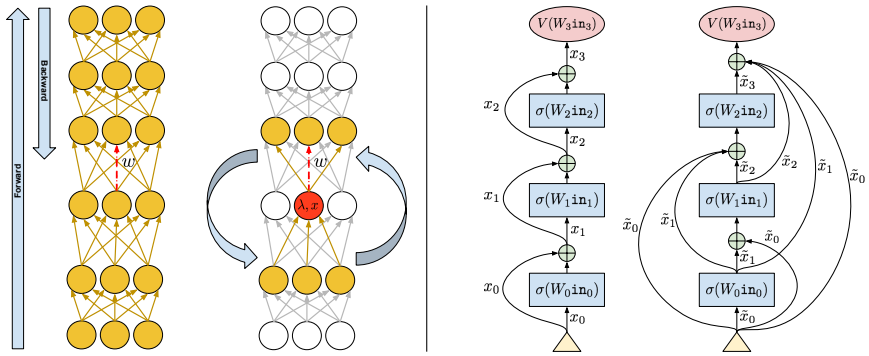

Figure 3.1: Left: the neurons and weights that are involved in the computations to update the red-dotted weight w are highlighted in yellow. (a) BackPropagation; (b) Local Propagation – the computations required to update the variables x, λ (associated to the red neuron) are also considered. Right: (c) ResNet in the case of $H = 3$, and (d) after the change of variables ($x_\ell \to \tilde{x}_\ell$) described in Sec. 3.4. Greenish circles are sums, and the notation $\mathtt{in}_\ell$ inside a rectangular block indicates the block input.

Differently, in the case of LP we have

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = -\sum_{i=1}^{N} \left(\lambda_{\ell+1,i} \odot \mathcal{G}'_{\ell+1,i} \odot \sigma'(W_\ell x_{\ell,i})\right) x_{\ell,i}^T \tag{3.3}$$

$$\frac{\partial \mathcal{L}}{\partial x_{\ell,i}} = \lambda_{\ell,i} \odot \mathcal{G}'_{\ell,i} - W_\ell^T \left(\lambda_{\ell+1,i} \odot \mathcal{G}'_{\ell+1,i} \odot \sigma'(W_\ell x_{\ell,i})\right) \tag{3.4}$$

$$\frac{\partial \mathcal{L}}{\partial x_{H,i}} = \lambda_{H,i} \odot \mathcal{G}'_{H,i} + W_H^T \left(V'(y_i, y'_i)\right) \tag{3.5}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_{\ell,i}} = \mathcal{G}_{\ell,i} \tag{3.6}$$

where $\mathcal{G}'$, σ' and V' are the first derivatives of the respective functions, and $\odot$ denotes the Hadamard product. The equations above hold for all $i \in [1, n]$ and $\ell \in [1, H]$, with the exception of Eq. (3.4) that holds for $\ell \in [1, H-1]$. It is evident that each partial derivative with respect to a variable associated to layer ℓ only involves terms that belong to the same layer (e.g., $\partial\mathcal{L}/\partial\lambda_{\ell,i}$) and also to either layer $\ell - 1$ (as in $\partial\mathcal{L}/\partial W_\ell$, given the presence of

$x_{\ell,i} = \sigma(W_{\ell-1}x_{\ell-1,i})$) or layer $\ell+1$ (the case of $\partial\mathcal{L}/\partial x_{\ell,i}$), that is, gradient computations are *local* (see Fig. 3.1 (b)).

This analysis reveals the full local structure of the algorithm for the discovery of saddle points. The role of the local updates is twofold: first, they project the variables onto the feasible region defined by the $\mathcal{G}_{\ell,i}$ constraints; second, they allow the information attached to the supervised pairs to flow from the loss function $V(\cdot,\cdot)$ through the network. The latter consideration is critical, since the information can flow through a large number of paths, and many iterations could be required to keep the model projected onto the feasible region and efficiently learn the network weights. In Section 3.5 we will also explore the possibility of enforcing a L_1-norm regularizer (weighted by $\alpha > 0$) on each $x_{\ell,i}$, in order to help the model to focus on a smaller number of paths from input to output units, reducing the search space.

Parallel Computations over Layers. As already emphasized, in Back-Propagation we have to perform a set of *sequential* computations over layers to complete the forward stage, and only afterwards we can start to *sequentially* compute the gradients, moving from the top layer down to the currently considered one (backward computations). Modern hardware (GPUs) can benefit by the layer-wise parallelization of the matrix operations, and there are no means to introduce further parallelization over multiple layers. Thanks to the decoupling principles of the proposed approach, in LP the locality in the gradient computation allows us to go beyond that. We can promptly see from Eq. (3.3-3.6) that we can trivially distribute all the computations associated to each layer ℓ in a different computational unit. Of course, the ℓ-th computational unit needs to share the memory where some variables are stored with the $(\ell+1)$-th and $(\ell-1)$-th units (see Eq. (3.3-3.6)).

Introducing Noise via Constraint Satisfaction Learning in the space to which the variables $\mathcal{W}, \mathcal{X}, \Lambda$ belong introduces a particular information flow through the network. If, during the optimization stage, the architectural constraints of Eq. (3.1) are strongly violated, then the updates applied to the network weights are not related to the ground truths that are attached to the loss function $V(\cdot)$, and we can imagine that the gradients are just noise. Differently, when the constraints are fulfilled, the information traverses the

network in a similar way to what happens in BackPropagation, i.e., in a noise-free manner. When the optimization proceeds, we progressively get closer to the fulfilment of the constraints, so that the noisy information is reduced. It is the learning algorithm itself that decides how to reduce the noise, in conjunction with the reduction of the loss on the supervised pairs. It has been shown that introducing a progressively reduced noise contribution to the gradient helps the BackPropagation algorithm to improve the quality of the solution, allowing very deep networks to be trained also when selecting low quality initialization of the weights [73]. The LP natively embeds this property so that, differently from [73], the noise reduction scheme is not a hand-designed procedure. Moreover, the local gradient computations of LP naturally offer a setting that is more robust to the problem of vanishing gradients, which afflicts BackPropagation when training deep neural networks.

Recovering BackPropagation. Similarly to the procedure presented in Section 2.5, the connections between the LP algorithm and BackPropagation become evident when imposing the stationary condition on the Lagrangian $\partial\mathcal{L}/\partial\lambda_{\ell,i} = 0$ and $\partial\mathcal{L}/\partial x_{\ell,i} = 0$. For the purpose of this description, let $\mathcal{G}(\cdot)$ be the identity function. From Eq. (3.6), we can immediately see that the stationary condition $\partial\mathcal{L}/\partial\lambda_{\ell,i} = 0$ leads to the classic expression to compute the outputs of the neural units, $x_{\ell,i} = \sigma(W_{\ell-1}x_{\ell-1,i})$, that is associated to the forward step of BackPropagation. Differently, when imposing $\partial\mathcal{L}/\partial\lambda_{\ell,i} = 0$ and defining $\delta_{\ell,i} = \lambda_{\ell,i} \odot \sigma'(W_{\ell-1}x_{\ell-1,i})$, Eq. (3.3) and Eq. (3.4) can be respectively rewritten as

$$\frac{\partial\mathcal{L}}{\partial W_{\ell-1}} = -\sum_{i=1}^{N} \delta_{\ell,i} \cdot x_{\ell-1,i}^T,$$
$$\delta_{\ell,i} = \sigma'(W_{\ell-1}x_{\ell-1,i}) \odot \left(W_\ell^T \delta_{\ell+1,i}\right),$$

that are the popular equations for updating weights and the BackPropagation deltas (Eq. (2.23)).

From this perspective, the BackPropagation algorithm represents the optimum w.r.t. the stationary conditions connected to the $\lambda_{\ell,i}$ and the $x_{\ell,i}$ when compared with Local Propagation. However, by strictly searching only on the hyperplane where the Lagrangian is stationary w.r.t $\lambda_{\ell,i}$ and $x_{\ell,i}$, Back-

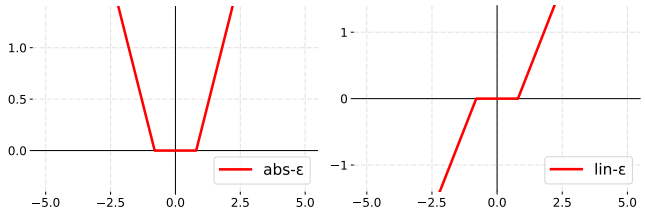

Figure 3.2: The mapping defined by the ε-insensitive functions, abs-ε and lin-ε.

Propagation loses the locality and parallelization properties characterizing our algorithm.

The decomposition of computation into units depending on local quantities allows us to compute gradients relying solely on the variables of neighboring layers, whilst in BP such computation depends on all the variables of the architecture.

3.3.2 Epsilon-insensitive Constraints

In order to facilitate the convergence of the optimization algorithm or to improve its numerical robustness, we can select different classes of $\mathcal{G}(\cdot)$ functions in Eq. (3.1). We focus on the class of ε-insensitive functions, and, in particular, on the following two cases $\mathcal{G} \in \{\text{abs-}\varepsilon, \text{lin-}\varepsilon\}$, depicted in Figure 3.2

$$\begin{aligned} \text{abs-}\varepsilon(a) &= \max(|a| - \varepsilon, 0) \\ \text{lin-}\varepsilon(a) &= \max(a, \varepsilon) - \max(-a, \varepsilon) \ . \end{aligned}$$

Both the functions are continuous, map values in $[-\varepsilon, \varepsilon]$ to zero, and they are linear out of such interval. However, abs-$\varepsilon(\cdot)$ is always positive, while lin-$\varepsilon(\cdot)$ is negative for arguments smaller than $-\varepsilon$. When plugged into Eq. (3.1), they allow the architectural constraints to tolerate a bounded mismatch in the values of $x_{\ell,i}$ and $\sigma(W_{\ell-1}x_{\ell-1,i})$ (ε-insensitive constraints). By way of example, let us consider two different input patterns indexed by i and j, for which we get two similar values $\sigma(W_{\ell-1}x_{\ell-1,i})$ and $\sigma(W_{\ell-1}x_{\ell-1,j})$ in a given layer ℓ. Then, for small values of ε, the same value $x_{\ell,i} = x_{\ell,j}$ can be selected by the optimization algorithm, thus propagating the same signal to

the units of the layer above. In other words, ε-insensitive constraints introduce a simple form of regularization when training the network, that allows the network itself not to be influenced by small changes in the neuron inputs, thus stabilizing the training step. Notice that, at test stage, if we compute the values of $x_{\ell,i}$'s with the classic forward procedure, then the network will not take into account the $\mathcal{G}(\cdot)$ function anymore. If ε is too large, there will be a large discrepancy between the setting in which the weights are learned and the one in which they are used to make new predictions. This could end up in a loss of performances, but it is in line with what happens in the case of the popular Dropout [74] when the selected drop-unit factor is too large.

Augmented Lagrangian

A key difference between abs-$\varepsilon(\cdot)$ and lin-$\varepsilon(\cdot)$ is the effect they have in the development of the Lagrange multipliers. It is trivial to see that, since abs-$\varepsilon(\cdot)$ is always positive, the multipliers $\lambda_{\ell,i}$ can only increase during the optimization (Eq. (3.6)). In the case of lin-$\varepsilon(\cdot)$, the multipliers can both increase or decrease. We found that abs-$\varepsilon(\cdot)$ leads to a more stable learning, where the violations of the constraints change more smoothly that in the case of lin-$\varepsilon(\cdot)$. In such a way, the constrained problem is altered in order to have a region of positive damping surrounding the stationary point, facilitating the convergence. A popular way to improve the numerical stability of the algorithm is to introduce the so called *Augmented Lagrangian* [55], where $\mathcal{L}$ of Eq. (3.2) is augmented with an additive term $\rho\|\mathcal{G}_{\ell,i}\|^2$, for all i, ℓ. This approach combines the penalty methods with the Lagrangian approach, yielding a Modified Differential Method of Multipliers (MDMM) [1], that enforces a local convergence around the constrained minima.

3.4 LP formulation for Neural Models

The described constraint-based formulation of neural networks and the LP algorithm can be easily applied to the most popular neural units, thus offering a generic framework for learning in neural networks. It is trivial to rewrite Eq. (3.1) to model convolutional units and implement Convolutional Neural Networks (CNNs) (see Section 2.1.3), and also the pooling layers can

be straightforwardly described with constraints. We study in detail the cases of Recurrent Neural Networks (RNNs) and of Residual Networks (ResNets). In order to simplify the following descriptions, we consider the case in which we have only $N = 1$ supervised pairs, and we drop the index i to make the notation simpler.

Recurrent Neural Networks. At a first glance, RNNs (see Section 2.1.2) might sound more complicated to implement in the proposed framework. As a matter of fact, when dealing with RNNs and BackPropagation, we have to take care of the temporal unfolding of the network itself (BackPropagation Through Time)[3]. However, we can directly write the recurrence by means of architectural constraints and, we get that, for all time steps t and for all layers $\ell \in [1, H]$,

$$\mathcal{G}\left(x_\ell^{(t)} - \sigma(W_{\ell-1}x_{\ell-1}^{(t)} + U_\ell x_\ell^{(t-1)})\right) = 0 \ ,$$

where U_ℓ is the matrix of the weights that tune the contribution of the state at the previous time step. The constraint-based formulation only requires to introduce constraints over all considered time instants. This implies that also the variables x_ℓ and the multipliers λ_ℓ are replicated over time (superscript t). The optimization algorithm has no differences with respect to what we described so far, and all the aforementioned properties of LP (Sec. 3.3) still hold also in the case of RNNs. While it is very well known that recurrent neural networks can deal only with sequential or DAG inputs structures (i.e. no cycles), LP architectural constraints show no ordering, since we ask for the overall fulfillment of the constraints. This property opens the door to the potential application of the proposed algorithm to problems dealing with generic graphical inputs, which is a very hot topic in the deep learning community [35, 75, 76].

Residual Networks. ResNets (see Section 2.1.3) consist of several stacked residual units, that have been popularized by their property of being robust with respect to the vanishing gradient problem, showing state-of-the art results in Deep Convolutional Neural Nets [44] (without being limited to such

[3]What we study here can be further extended to the case of Long Short-Term Memories (LSTMs) [42]

networks). The most generic form of a single residual unit has been described in Section 2.1.3,

$$x_\ell = z(h(x_{\ell-1}) + f(W_{\ell-1}x_{\ell-1})) \ . \tag{3.7}$$

In the popular paper [44], we have that $z(\cdot)$ is a rectifier (ReLu) and $h(\cdot)$ is the identity function, while $f(\cdot)$ is a non-linear function. On one hand, it is trivial to implement a residual unit as a constraint of LP once we introduce the constraint

$$\mathcal{G}(x_\ell - z(h(x_{\ell-1}) + f(W_{\ell-1}x_{\ell-1}))) = 0 \ . \tag{3.8}$$

However, we are left with the question whether these units still provide the same advantages that they show in the case of backprop-optimized networks. In order to investigate the ResNet properties, we focus on the identity mapping of [45], where $z(\cdot)$ and $h(\cdot)$ are both identity functions, and, for the sake of simplicity, $f(\cdot)$ is a plain neural unit with activation function σ,

$$x_\ell = x_{\ell-1} + \sigma(W_{\ell-1}x_{\ell-1}) \ , \tag{3.9}$$

as sketched in Fig. 3.1 (c). This implementation of residual units is the one where it is easier to appreciate how the signal propagates through the network, both in the forward and backward steps. The authors of [44] show that the signal propagates from layer ℓ to layer $L > \ell$ by means of additive operations, $x_L = x_\ell + \sum_{j=\ell}^{L-1} \sigma(W_j x_j)$, while in common feedforward nets we have a set of products. Such property implies that the gradient of the loss function $V(y, y')$ with respect to x_ℓ is

$$\frac{\partial V}{\partial x_\ell} = \frac{\partial V}{\partial x_H}\left(1 + \frac{\partial}{\partial x_\ell}\sum_{j=\ell}^{H-1}\sigma(W_j x_j)\right) \ , \tag{3.10}$$

that clearly shows that there is a direct gradient propagation from V to layer ℓ (due to the additive term 1). Due to the locality of the LP approach, this property is lost when computing the gradients of each architectural constraint, that in the case of the residual units of Eq. (3.9) are

$$\mathcal{G}(x_\ell - x_{\ell-1} - \sigma(W_{\ell-1}x_{\ell-1})) = 0 \ . \tag{3.11}$$

As a matter of fact, the loss $V(y, y')$, being $y' = \sigma(W_H x_H)$ will only have a role in the gradient with respect to variables x_H and W_H, and no immediate

effect in the gradient computations related to the other constraints/variables. However, we can rewrite Eq. (3.11) by introducing $\tilde{x}_\ell = x_\ell - x_{\ell-1}$, that leads to $x_\ell = \tilde{x}_\ell + x_{\ell-1}$. By repeating the substitutions, we get $x_\ell = \sum_{j=0}^{\ell} \tilde{x}_j$, and Eq. (3.11) becomes[4]

$$\mathcal{G}\Big(\tilde{x}_\ell - \sigma\Big(W_{\ell-1} \cdot \sum_{j=0}^{\ell-1} \tilde{x}_j\Big)\Big) = 0 \ , \tag{3.12}$$

where the arguments of the loss function change to $V\big(y, \sigma(W_H \cdot \sum_{j=0}^{H} \tilde{x}_j)\big)$. Interestingly, this corresponds to a feed-forward network with activations that depend on the sum of the outputs of all the layers below, as shown in Fig. 3.1 (d). Given this new form of the second argument of $V(\cdot, \cdot)$, it is now evident that even if the gradient computations are local, the outputs of all the layers directly participate to such computations, formally

$$\frac{\partial \mathcal{L}}{\partial \tilde{x}_\ell} = \frac{\partial V}{\partial \tilde{x}_\ell} + \frac{\partial}{\partial \tilde{x}_\ell} \sum_{j=\ell}^{H} \mathcal{G}_j \ . \tag{3.13}$$

Differently from Eq. (3.10), $\frac{\partial V}{\partial \tilde{x}_\ell}$ (that is the same $\forall \ell$) does not scale the gradients of the summation, so that the gradients coming from the constraints of all the hidden layers above ℓ are directly accumulated by sum.

3.5 Experiments

We designed a batch of experiments aimed at validating the simple local optimization approach to constraint-based networks presented in this Chapter. Our goal is to show that the approach is feasible and that the learned networks have generalization skills that are in-line with BackPropagation, also when using multiple hidden layers. In other words, we show that the new properties provided by the Local Propagation algorithm (i.e. locality and parallelization) do not correspond to a loss in performance w.r.t. BP, even if the search space has been augmented with the unit activations and the Lagrange multipliers variables.

[4] We set $\tilde{x}_0 = x_0$ ($\tilde{x}_0$ is not a variable of the learning problem).

Table 3.1: Number of patterns, of input features and of output classes in the datasets exploited for benchmarking the LP algorithm.

Dataset	Examples	Dimensions	Classes
Adult	48842	14	2
Ionosphere	351	33	2
Letter	20000	16	26
Pima	768	8	2
Wine	179	13	3
Ozone	2536	72	2
Dermatology	366	34	6
MNIST	70000	784	10

Setting & Data. We performed experiments on 7 benchmarks from the UCI repository [77], and on the MNIST data (Table 3.1). The MNIST is partitioned into the standard training, validation and test sets, while in the case of the UCI data we followed the experimental setup of [78], where the authors used the training and validation partitions of [79] to tune the model parameters, and 4-folds to compute the final accuracy (averaged over the 4 test splits)[5].

Parameters. We evaluated several combinations of the involved parameters, varying them in: $\varepsilon \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$, $\rho \in \{10^{-2}, 10\}$, $c \in \{0, 0.001\}$, dropout keep-rate (BP only) $\in \{0.7, 0.8, 0.9\}$, $\alpha \in \{0, 10^{-8}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-1}\}$. We used the Adam optimizer (TensorFlow), where the learning rate η_w for updating variables $\mathcal{W}$ is $\in \{10^{-4}, 10^{-3}, 10^{-2}\}$, and the learning rate η_z for updating $\mathcal{X}, \Lambda$ is $\in \{0.1 \cdot \eta_w, 10 \cdot \eta_w\}$. We used the same initialization of the weight matrices for BP and LP.

Since similar behaviours are shown by both sigmoid and ReLU activations, in our experiments, we exploited only the former, in order to reduce the hyper-parameter search space. We trained our models for thousands epochs, measuring the accuracy on the validation data (or, if not available, a held-out portion of the training set) to select the best $\mathcal{W}$.

UCI performances. We evaluated the accuracies of BP and LP focussing on the same pair of architectures (sigmoidal activation units), that is composed

[5] In the case of the Adult data we have only 1 test split.

Table 3.2: Performances of the same architectures optimized with BP and LP. Left: $H = 1$ hidden layer (100 units); right: $H = 3$ hidden layers (30 units each). Largest average accuracies are in bold.

	BP ($H = 1$)	LP ($H = 1$)	BP ($H = 3$)	LP ($H = 3$)
Adult	84.66 ±0.00	**85.43** ±0.00	84.91 ±0.00	**85.34** ±0.00
Iono.	91.48 ±0.57	91.48 ±2.95	92.61 ±0.57	**94.60** ±1.86
Letter	94.20 ±0.31	**94.94** ±0.05	**92.27** ±0.19	90.42 ±0.78
Pima	76.17 ±1.62	**77.21** ±2.79	**76.56** ±2.42	75.91 ±1.54
Wine	97.16 ±1.88	**98.86** ±1.14	97.73 ±2.78	**98.86** ±1.97
Ozone	97.04 ±0.26	**97.12** ±0.13	**97.28** ±0.13	97.20 ±0.17
Derma.	95.60 ±1.74	**96.70** ±1.74	97.53 ±1.20	**98.63** ±0.48

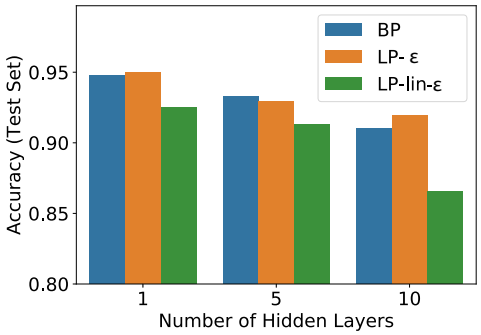

Figure 3.3: Accuracies of BP and LP (with different ε-insensitive functions, abs-ε, lin-ε) on the MNIST data.

by a shallow net with 1 hidden layer of 100 units, and a deeper network with 3 hidden layers of 30 units each, reporting results in Table 3.2. Both algorithms perform very similarly, with LP having some minor overall improvements over BP.

MNIST performances. Similar conclusions can be drawn in the case of the MNIST data, as shown in Fig. 3.3. In this case, we considered deeper networks with up to 10 hidden layers (10 neurons on each layer), and we also evaluated the impact of the different ε-insensitive constraints of Section 3.3.2. We considered 5 different runs, reporting the test accuracy corresponding to the largest result on the validation data. When using lin-$\varepsilon(\cdot)$ as the $\mathcal{G}(\cdot)$

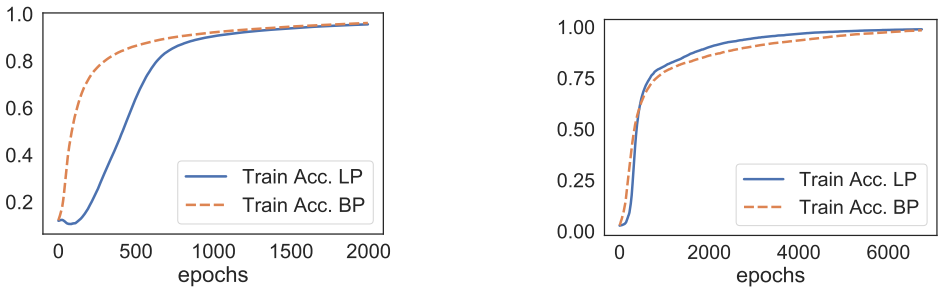

Figure 3.4: Convergence speed of BP and LP in the MNIST dataset (left), and in the Letter data (right).

function, we faced an oscillating behaviour of the learning procedure due to the inherent double-signed violation of the constraints. The Augmented Lagrangian ($\rho > 0$) resulted to be fundamental for the stability and for improving the convergence speed of LP.

Due to local nature of LP and to the larger number of variables involved in the optimization, we usually experimented an initial transitory stage in the optimization process, where the system is still far from fulfilling the available constraints, and the model accuracy is small, as shown in Fig. 3.4. This sometimes implies a larger number of iterations with respect to BackPropagation to converge to a solution (Fig. 3.4 (left) - MNIST), as expected, but it is not always the case (Fig. 3.4 (right) - Letter).

The role of ε-insensitive constraints In order to better understand how LP behaves, we deeply explored the previously discussed results of Table 3.2. First, we evaluated the role of the ε-insensitive constraints, reporting in Table 3.3 (top-half) the results for the case in which $\varepsilon = 0$ and $\varepsilon > 0$. Then, we explored the effect of including the L_1-norm regularization term, as shown in Table 3.3 (*bottom-half*) ($\alpha = 0$ means no-L_1-regularization). In the case of shallow networks, $\varepsilon > 0$ offers performances that, on average, are preferable or on-par to the case in which no-tolerance is considered (both for abs-ε and lin-ε). This consideration is not evident in the case of deeper nets, where a too strong insensitivity might badly propagate the signal from the ground truth to the lower layers. We notice that while this is evident in the case of UCI data, we did not experienced this behaviour in the case of MNIST of the

Table 3.3: Accuracies of LP when using ($\varepsilon > 0$) or not using ($\varepsilon = 0$) ε-insensitive constraints (*top-half*) and when using ($\alpha > 0$) or not using ($\alpha = 0$) L_1-norm-based regularization on the outputs of each layer (*bottom-half*). We report the cases of the abs-ε and lin-ε functions, and we compare architectures with $H = 1$ hidden layer (100 units) and $H = 3$ hidden layers (30 units each).

	$H = 1$			
	$\varepsilon = 0$ (abs-ε)	$\varepsilon > 0$ (abs-ε)	$\varepsilon = 0$ (lin-ε)	$\varepsilon > 0$ (lin-ε)
Adult	85.33	85.33	**85.43**	85.27
Iono	**91.48** ±2.95	91.19 ±2.71	90.34 ±0.98	**91.19** ±2.18
Letter	**94.94** ±0.05	94.85 ±0.06	93.71 ±0.20	**94.00** ±0.27
Pima	75.39 ±2.03	**77.21** ±2.79	75.00 ±1.77	**75.78** ±1.97
Wine	97.73 ±1.61	**98.86** ±1.14	98.30 ±1.88	98.30 ±1.88
Ozone	97.04 ±0.13	97.04 ±0.13	96.96 ±0.30	**97.12** ±0.13
Derma.	**95.60** ±0.78	95.33 ±2.11	95.60 ±1.74	**96.70** ±1.74
	$\alpha = 0$ (abs-ε)	$\alpha > 0$ (abs-ε)	$\alpha = 0$ (lin-ε)	$\alpha > 0$ (lin-ε)
Adult	85.33	85.33	85.27	**85.43**
Iono	90.63 ±0.49	**91.48** ±2.95	**91.19** ±2.18	90.06 ±2.71
Letter	94.85 ±0.27	**94.94** ±0.05	93.78 ±0.38	**94.00** ±0.27
Pima	75.39 ±2.03	**77.21** ±2.79	**75.78** ±1.97	75.00 ±1.77
Wine	98.86 ±1.97	98.86 ±1.14	**98.30** ±1.88	97.73 ±1.61
Ozone	97.00 ±0.11	**97.04** ±0.13	**97.12** ±0.13	96.96 ±0.30
Derma.	95.60 ±1.10	95.60 ±0.78	95.88 ±1.43	**96.70** ±1.74
	$H = 3$			
	$\varepsilon = 0$ (abs-ε)	$\varepsilon > 0$ (abs-ε)	$\varepsilon = 0$ (lin-ε)	$\varepsilon > 0$ (lin-ε)
Adult	**85.34**	85.25	**85.25**	85.23
Iono	**94.60** ±1.86	94.32 ±1.14	**92.61** ±1.27	91.19 ±0.94
Letter	**87.60** ±0.48	87.54 ±0.48	**90.42** ±0.78	90.39 ±0.27
Pima	75.52 ±1.91	**75.91** ±3.34	74.48 ±0.90	**75.91** ±1.54
Wine	97.73 ±3.94	97.73 ±2.78	**98.86** ±1.97	97.73 ±1.61
Ozone	97.08 ±0.14	**97.20** ±0.17	**97.16** ±0.22	97.04 ±0.34
Derma.	**98.63** ±0.48	97.80 ±0.78	97.80 ±0.78	**98.08** ±0.91
	$\alpha = 0$ (abs-ε)	$\alpha > 0$ (abs-ε)	$\alpha = 0$ (lin-ε)	$\alpha > 0$ (lin-ε)
Adult	**85.34**	84.93	85.23	**85.25**
Iono	90.34 ±1.70	**94.60** ±1.86	89.77 ±2.27	**92.61** ±1.27
Letter	**87.60** ±0.48	87.54 ±0.48	**90.42** ±0.78	88.88 ±0.45
Pima	**75.91** ±3.34	75.65 ±3.62	**75.91** ±1.54	75.65 ±1.89
Wine	**97.73** ±2.78	96.59 ±3.77	**98.86** ±1.97	97.73 ±1.61
Ozone	97.08 ±0.14	**97.20** ±0.17	97.08 ±0.14	**97.16** ±0.22
Derma.	97.80 ±0.78	**98.63** ±0.48	**98.08** ±0.91	97.80 ±0.78

aforementioned Fig. 3.3, where the best accuracies where usually associated with $\varepsilon > 0$. This might be due to the smallest redundancy of information in the UCI data with respect to MNIST. When focussing on the effect of the L_1-norm-based regularization (Table 3.3, bottom-half), we can easily see that such regularization helps in several cases, suggesting that it is an useful feature that should be considered in validating LP-based networks. This is due to the

sparsification effect that emphasizes only a few neurons per layer, allowing LP to focus on a smaller number of input-output paths.

Constraints insensitivity and deeper nets The locality of the gradient computations in LP ideally offer a setting that is more robust to the problem of vanishing gradients. Indeed, each computation rely solely on local quantities, avoiding the sequential nature of computation characterizing BP. Such local variables, $x_{\ell,i}$, are the result of a learning dynamic that depends on the constrained optimization mechanism of BDMM. When such constraints are not completely fulfilled, such as in the case of the introduction of ε-regulated insensitivity, the variable update becomes noisy and less representative of the true contributions coming from the neighboring layers. Whilst this behaviour could favour generalization [73], the increase of the depth of the neural network model at hand (see Figure 3.3) foster the injection of an increasing amount of noise in the variable updates. This could hinder the correct propagation of the signal from the ground truth to the lower layers. Solutions that force the constraints to be fulfilled in the long term, such as the Augmented Lagrangian term or higher order dynamics [80], as long as a longer training phase up to the exact constraint satisfaction, could alleviate this drawback.

3.6 Discussion

This Chapter presented the first implementation of the idea of decomposing artificial neural networks into local components to foster the representation of their computational structure by constraints, giving rise to the Local Propagation algorithm. We presented a novel way of interpreting the architecture of neural networks and of the learning process for their parameters, based on the so-called architectural constraints. It has been shown that the Lagrangian formulation in the adjoint space leads to a fully local algorithm, LP, that naturally emerges when searching for saddle points. An experimental analysis on several benchmarks assessed the feasibility of the proposed approach, whose connections with popular neural models has been described. Despite its simplicity and its strongly parallelizable computations, LP introduces additional variables to the learning problem. Some ideas and future works will be presented in Chapter 6.

We analyzed the application of LP to several architectures, but its powerful nature opens the road to further investigations on complex neural architectures, such as the ones that operate on graphs, that will be the subject of the following Chapter.

Chapter 4

Lagrangian Propagation Graph Neural Networks

The neural architectures exploited in the previous Chapter are commonly used to deal with data that exhibit a "flat" structure, that can be easily represented with a vector embedded in an Euclidean space. Conversely, several real-world applications are characterized by data with an underlying structure composed by entities, their properties, and relations among them, naturally represented with the mathematical notion of graphs. The increasing interest [81] in Graph Neural Network models, that have been introduced in Section 2.1.4, testifies the recent trend in dealing with such kind of data.

All the models falling under the umbrella of GNNs (or MPNNs), generally share the same learning mechanism based on error BackPropagation (BP), that when processing structured data is straightforwardly extended by the process of *unfolding*, as described in Section 2.2.1. A network topology based on the current input structure is generated by replicating a base neural network module (e.g. BP Through Time, BP Through Structure).

Bearing in mind the main idea spanning this dissertation, one could think to simply inject the Local Propagation algorithm (Chapter 3) into this class of graph architectures, especially attracted by the local nature of computations, and its applicability to learn any computational structure, both acyclical or cyclical.

However, the main drawback of the Lifted Methods (i.e. weak points shared by LP and related approaches [61, 62]) is that they are quite memory inefficient; in particular, they need to keep extra-variables for each hidden neuron and for each example. This makes them inapplicable to large problems, like the ones available in graph data, where BP is still the only viable option. However, in this Chapter we will introduce a novel mixed method which is still inspired by the principle of locality, which is leveraged to describe the

computational structure of the problem using the unifying notion of constraint.

Recent trends in the field of Graph Representation learning leverage MPNN models (see Section 2.1.4) composed by a limited number of layers. Indeed, stacking T layers of parametrized aggregation fosters layer–wise message propagation inside the T-hop neighborhood of the nodes. Several works highlighted the advantages of such solution, for instance the fact that this models are Turing Universal if their capacity, defined as the product of the dimension of the state vector representation by the network depth (number of layers), is large enough [82]. However, having many layers will wash away node features information [52], an issue that can be overcome with some tricks involving residual connections or particular node aggregation functions.

The seminal Graph Neural Network model [53] represents a more general message passing scheme on graphs. The learning process requires, for each training epoch, an iterative diffusion mechanism that is repeated until it converges to a stable fixed point, requiring a multi-stage optimization procedure that is computationally expensive and less practical than models based on gradient-based optimization. The iterative process can be early stopped to speed-up the computation, but this ends up in limiting the quality of the outcome of the local encoding, virtually reducing the depth of the diffusion along the graph of the information carried by each node. The diffusion mechanism at convergence virtually involves all the graph, not only a T-hop limited neighborhood.

Leveraging the principles guiding this dissertation, we propose a new learning mechanism for GNNs that is based on the decomposition of the unfolded computational graph of the model. We exploit the same Lagrangian formulation characterizing the Local Propagation algorithm, in which we represent the diffusion of information and the relationship between each node and its local neighborhood by a set of constraints. Finding node state representations that fulfill the constraints is a simple way to rethink the computation of the fixed point of the aforementioned diffusion process. In particular, we exploit the same differential optimization process already described in the context of Chapter 3, to devise a learning scheme that is fully based on BP and where the state representations and the weights of the state transition and output functions are jointly optimized without the need of applying any separate iterative

procedures at every training epoch.

Differently from Local Propagation (Chapter 3), both the state transition function and the output function are classic BP-trainable models, that are shared by all the training examples, whereas the only additional variables of the learning problem are associated to the nodes of the graphs. This allows us to find a good trade off between the flexibility introduced by the Lagrangian-based formulation of the graph diffusion and the addition of new variables. We further extend this idea computing multiple representations of each node by means of a pipeline of constraints that very much resembles a multi-layer computational scheme. In particular, the evolving representation of each node is treated as new information attached to the node itself. Another state transition function is introduced, that has the use of such new information, while constraints enforce a parallel diffusion process that leads to the development of another representation of the node. This procedure can be replicated multiple times, thus simulating a deep constraining scheme that augments the representation capabilities of the GNN. Experimental results on several popular benchmarks emphasize the quality of the proposed model, hereafter referred to as Lagrangian Propagation GNN (LP-GNN), that compares favourably with the original GNN model and more recent variants.

This Chapter is organized as follows. Section 4.1 reviews the recent developments in the field of Neural Network models for processing graphs and learning methods based on constraint base formulations. Section 4.2 introduces the basics of the GNN model in the context of our constrained approach, followed by the proposed Lagrangian formulation of GNNs, and the multi-layered model in Section 4.3. Section 4.4 reports an experimental evaluation. Finally, conclusions are drawn in Section 4.5.

4.1 Related Work

Despite the large ubiquity of data collected in Euclidean domains in which each sample is a fixed-length vector, a large number of application domains require to handle data that are characterized by an underlying structure that lays on a non-Euclidean domain, i.e. graphs [83]. Whilst commonly addressed in relational learning, such domains have been initially not taken into account

by popular machine learning techniques, that have been mostly devised for grid–like and Euclidean structured data [83]. We will describe in the following some of the most important facets of this trending topic.

4.1.1 Extending ANNs to graphs

Early machine learning approaches for structured data were designed for directed acyclic graphs [84, 49], while a more generic framework is represented by the aforementioned GNNs [53], able to deal with directed, undirected and cyclic graphs. The iterative encoding procedure up to the fixed point of the state transition function fosters the diffusion among neighboring nodes, but it is a computationally expensive process. Some methods were aimed at simplifying this step, such as the scheme proposed in [85] that exploits gated recurrent units. These models are usually denoted as *Recurrent GNNs*.

More recently, a large number of approaches further extended the aforementioned research direction. They differ in the choice of the neighborhood aggregation method and in the graph level pooling scheme, and they can be categorized into two main areas. *Spectral approaches* exploit particular embeddings of the graph and convolution operations defined in the spectral domain [86]. However, they are characterized by computational drawbacks caused by the eigen–decomposition of the graph Laplacian. Simplified approaches are based on smooth reparametrization [87] or approximation of the spectral filters by a Chebyshev expansion [88]. Finally, in Graph Convolutional Networks (GCNs) [89], filters are restricted to operate in a 1-hop neighborhood of each node. *Spatial methods*, instead, directly exploit the graph topology, without the need of an intermediate representation. These approaches differ mainly in the definition of the aggregation operator used to compute the node states, that must be able to maintain weight sharing properties and to process nodes with different numbers of neighbors. The PATCHY-SAN [90] model converts graph-structured data into a grid-structured representation, extracting and normalizing neighborhoods containing a fixed number of nodes. In [91] the model exploits a weight matrix for each node degree, whereas DCNNs [92] compute the hidden node representation by convolving the input channels with power series of the transition probability matrix, learning weights for each neighborhood degree. GraphSAGE [93] exploits different aggregation

functions to merge the node neighborhood information. In the context of graph classification tasks, SortPooling [94] uses a framework based on DGCNNs with a pooling strategy, that performs pooling by ordering vertices. Finally, MPNN by Gilmer [50, 51] (see Section 2.1.4) tries to unify all the previously described models under a common framework.

4.1.2 The role of aggregation functions

All the models presented in the previous Section compute the node states carrying on an aggregation phase in which every neighbor contributes equally to the state computation. Every direction is treated in the same way, and for this reason these models are denoted as *Isotropic* GNNs. However, it is possible to gain a directional structure adding edge features to the computation, but this solution holds only if such features are available in the task at hand. An interesting alternative solution is to let the model learn *anisotropy* from data, in order to treat neighbors differently, as done in models like Graph Attention Networks [95] and alternatives [96, 97].

A subsequent line of research focuses on the analysis of the expressive capabilities of GNNs and their aggregation functions. The seminal work by Xu et al. [2] paved the road to studies on the expressive power of GNNs through the notion of graph isomorphism. The authors propose a model, the Graph Isomorphism Network (GIN), having the same representational power of the Weisfeiler-Leman Test [98]. The main intuition is based on the usage of an injective aggregation function, e.g. the sum, which is able to distinguish and devise elements belonging to a multiset (a set that allows multiple instances for its elements). Conversely, other types of aggregation functions tend to capture the proportion/distribution of elements of a given type (Mean) or ignore multiplicities, reducing the multiset to a simple set (Max). However, the WL-test itself is not a sufficient condition for two graphs being isomorphic, and for this reason several other works try to capture higher-order graph properties [99, 100].

Bearing in mind these findings, other approaches try to combine several aggregation functions in order to devise a powerful representational scheme [101].

4.1.3 Other recent trends

The are multiple research directions in the context of Graph Representation Learning. One of the biggest weaknesses of the early models was the lack of a common benchmarking setting for performance evaluation [102], that recent work tries to propose [103, 104]. Beside other approaches looking into different mixed learning strategies (i.e. combining inductive and transductive learning [105, 106]), there are other approaches trying to devise methods able to scale to big real-world datasets (social networks, recommender systems) [107, 108]. A very interesting work in this direction, Steady State Embedding (SSE) [109], shares similar intuitions to those described in this Chapter. Indeed, the authors prove that algorithms on graphs can be effectively learned exploiting a constrained fixed-point formulation, solved via the Reinforcement Learning *policy iteration* algorithm. The authors exploit this method for the interleaved evaluation of the fixed point equation and the improvement of the transition and output functions, resorting to ad-hoc moving average updates.

4.2 Constraint-based Propagation in Graph Neural Networks

During the learning stage of the original GNN training algorithm [35], that has been described in Section 2.1.4 and 2.2.1, convergence to the fixed point (or of an approximation of it) of the state transition function is required at *each epoch* of the learning procedure. The gradient computation requires to take into account such relaxation procedure, by a BackPropagation schema that involves the replicas of the state transition network exploited during the iterative fixed point computation (*unfolding*). This is due to the fact that the definition of the computational graph in GNNs is driven both by the input graph topology and by the convergence procedure.

To briefly summarize, we denote the state transition function, applied to a node $v \in V$, with:

$$f_{a,v} = f_a(x^{(t)}_{ne[v]}, l_{ne[v]}, l_v, l_{(ne[v],v)}|\theta_{f_a}) \ . \tag{4.1}$$

Basically, the encoding phase, through the iteration of $f_a()$, finds a solution to the fixed point problem defined by the *equality constraint* between each node

state and the way it is computed by the state transition function

$$\forall v \in V, \quad x_v = f_{a,v}. \tag{4.2}$$

Therefore, we can think at the computational graph of GNNs as defining a set of constraints among the state variables $x_v,\ v \in V$ and the state transition functions.

This view of the learning process paves the way to the introduction of similar intuition to the ones presented in Chapter 3, this time restricted to the node state computation. The iterative encoding procedure itself can be expressed via the unifying notion of constraint, making it possible to decouple the node state computation from the convergence procedure.

We can consider a Lagrangian formulation of the learning problem by adding free variables corresponding to the node states x_v, such that the fixed point is directly defined by the constraints themselves, as

$$\forall v \in V, \quad \mathcal{G}\left(x_v - f_{a,v}\right) = 0, \tag{4.3}$$

where $\mathcal{G}(x)$ is an ε-insensitive function (see Section 3.3.2) characterized by $\mathcal{G}(0) = 0$, such that the satisfaction of the constraints implies the solution of Eq. (4.2). The original formulation of the problem would require $\varepsilon = 0$, but by setting ε to a small positive value it is possible to obtain a better generalization and tolerance to noise. In this way, the convergence procedure is translated into the enforcement of the constraint expressed in Eq. (4.3). Therefore, the fulfillment of such constraints fosters the diffusion of information in the neighborhood of each node.

4.2.1 A Local Learning formulation

In the following, for simplicity, we will refer to a node-focused task, such that for some (or all) nodes $v \in S \subseteq V$ of the input graph G, a target output y_v is provided as a supervision[1]. If $L(f_r(x_v \mid \theta_{f_r}), y_v)$ is the loss function used to

[1]For the sake of simplicity we consider only the case when a single graph is provided for learning. The extension to more graphs is straightforward for node-focused tasks, since they can be considered as a single graph composed by the given graphs as disconnected components.

measure the target fitting approximation for node $v \in S$, the formulation of the learning task is:

$$\begin{aligned} \min_{\theta_{f_a}, \theta_{f_r}, X} \quad & \sum_{v \in S} L(f_r(x_v \mid \theta_{f_r}), y_v) \\ \text{subject to} \quad & \mathcal{G}\left(x_v - f_{a,v}\right) = 0, \quad \forall\, v \in V, \end{aligned} \tag{4.4}$$

where we already defined θ_{f_a} and θ_{f_r} as the weights of the MLPs implementing the state transition function and the output function, respectively, while $X = \{x_v : v \in V\}$ is the set of the newly introduced free state variables.

This problem statement implicitly includes the definition of the fixed point of the state transition function since for each solution in the feasible region the constraints are satisfied, and hence the learned x_v are solutions of Eq. (4.2). The constrained optimization problem of Eq. (4.4) can be faced in the Lagrangian framework (see Section 2.4) by including a Lagrange multiplier λ_v for each constraint, such that the Lagrangian function $\mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda)$ is defined as:

$$\mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda) = \sum_{v \in S} \left[L(f_r(x_v \mid \theta_{f_r}), y_v) + \lambda_v \mathcal{G}\left(x_v - f_{a,v}\right) \right], \tag{4.5}$$

where Λ is the set of the $|V|$ Lagrangian multipliers. Hence, we can find the solution of the learning problem by optimizing an unconstrained optimization index and searching for saddle points in the adjoint space $(\theta_{f_a}, \theta_{f_r}, X, \Lambda)$, following the Basic Differential Multiplier Method [1], introduced in Section 2.6. In detail, we aim at solving

$$\min_{\theta_{f_a}, \theta_{f_r}, X} \max_{\Lambda} \; \mathcal{L}(\theta_{f_a}, \theta_{f_r}, X, \Lambda), \tag{4.6}$$

that can be approached with gradient descent-based optimization with respect to the variables $\theta_{f_a}, \theta_{f_r}, X$, and gradient ascent with respect to the Lagrange multipliers Λ (see Section 2.6). Interestingly, the gradient can be computed *locally* to each node, given the node-related variables and those of the neighboring nodes. In fact, the derivatives of the Lagrangian with respect to the

involved parameters are[2]:

$$\frac{\partial \mathcal{L}}{\partial x_v} = L' f'_{r,v} + \lambda_v \mathcal{G}'_v (1 - f'_{a,v}) - \sum_{w : v \in \text{ne}[w]} \lambda_w \mathcal{G}'_w f'_{a,w} \tag{4.7}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{f_a}} = - \sum_{v \in S} \lambda_v \mathcal{G}'_v f'_{a,v} \tag{4.8}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_{f_r}} = \sum_{v \in S} L' f'_{r,v} \tag{4.9}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_v} = \mathcal{G}_v \tag{4.10}$$

where $f_{a,v}$ is defined in Eq. (4.1), $f'_{a,v}$ is its derivative[3], $f_{r,v} = f_r(x_v \mid \theta_{f_r})$, $f'_{r,v}$ is its derivative (with respect to θ_{f_r}), $\mathcal{G}_v = \mathcal{G}\left(x_v - f_{a,v}\right)$ and $\mathcal{G}'_v$ is its first derivative, and, finally, L' is the first derivative of L. Being f_a and f_r implemented by feedforward neural networks, their derivatives are obtained easily by applying a classic BackPropagation scheme, in order to optimize the Lagrangian function in the descent-ascent procedure, aiming at reaching a saddle point. The decomposition of the convergence procedure into a constraint satisfaction process and the locality of computations represent two big advantages with respect to the original formulation of the problem.

4.2.2 A mixed optimization strategy

A noticeable difference with respect to the Local Propagation algorithm presented in Chapter 3 is that, even if the proposed formulation adds the free state variables x_v and the Lagrange multipliers λ_v, $v \in V$, there is no significant increase in the memory requirements. Indeed, since the state variables also need to be memorized in the original formulation of GNNs, the learning problem is lifted just by a single Lagrange multiplier for each node.

This novel approach in training Graph Neural Network has several interesting properties. The learning algorithm is based on a mixed strategy characterized by the following two properties.

[2]When parameters are vectors, the reported gradients should be considered element-wise.

[3]In Eq. (4.7) and Eq. (4.8) such derivative is computed with respect to the same argument as in the partial derivative on the left side.

1. BackPropagation is used to efficiently update the weights of the neural networks that implement the state transition and output functions.

2. The diffusion mechanism evolves gradually by enforcing the convergence of the state transition function to a fixed point by virtue of the constraints.

This introduces a significant difference with respect to running an iterative procedure at each epoch and, only afterwards, applying the backward stage of BackPropagation to update the weights of f_a and f_r, as done in classic GNNs. In the proposed scheme, both the neural network weights and the node state variables are simultaneously updated by gradient-based rules. The learning proceeds by jointly updating the function weights and by diffusing information among the nodes, through their state, up to a stationary condition where both the objective function is minimized and the state transition function has reached a fixed point of the diffusion process. This also introduces a strong simplification in the way the algorithm can be implemented in modern software libraries that commonly include automatic gradient computation. Given the fact that the Lagrangian function describes both the loss function optimization and the information diffusion inside the processed graphs, we denote our model with the name Lagrangian Propagation GNN (LP-GNN).

As it will be further investigated in Section 4.4.4, in the proposed algorithm, the diffusion process is turned itself into an optimization process that must be carried out both when learning and when making predictions. As a matter of fact, inference itself requires the diffusion of information through the graph, that, in our case, corresponds with satisfying the constraints of Eq. (4.3). For this reason, the testing phase requires a (short) optimization routine to be carried out, that simply looks for the satisfaction of Eq. (4.3) for test nodes, and it is implemented using the same code that is used to optimize Eq. (4.6), avoiding to update the state transition and output functions. This is deeply different both from original GNNs [53], where it is realized by the unrolling of the transition function, and from Graph Convolutional Networks (GCN) [89] or other MPNNs, where it is embedded into the layer-wise projection.

4.2.3 LP-GNN Complexity analysis

Common graph models exploit synchronous updates among all nodes and multiple iterations/layers for the node state embedding, with a computational complexity for each parameter update $\mathcal{O}(T(|V| + |E|))$, where T is the number of iterations, $|V|$ the number of nodes and $|E|$ the number of edges. By simultaneously carrying on the optimization of neural models and the diffusion process, our scheme relies only on 1-hop neighbors for each parameter update, hence showing a computational cost of $\mathcal{O}(|V| + |E|)$. From the memory cost point of view, the persistent state variable matrix requires $\mathcal{O}(|V|)$ space. However, it represents a much cheaper cost than most of GNN models, usually requiring $\mathcal{O}(T|V|)$ space. In fact, those methods need to store all the intermediate state values of all the iterations, for a latter use in back-propagation.

4.3 Deep LP-GNNs

The GNN computation may exploit a Multi-Layer Neural Network with any number of hidden layers to implement the state transition function f_a of Eq. (4.1). By using more layers in this network, the model is able to learn more complex functions to diffuse the information on the graph, but the additional computation is completely local to each node. A different approach to yield a deep structure is to add layers to the state computation mechanism, in order to design a Layered GNN [110]. Basically, a set of K states $\{x_{v,k}, k = 0, \ldots, K-1\}$ is computed for each node $v \in V$. The state of the first layer, $x_{v,0}$, is computed by Eq. (4.1) and it becomes a labeling of node v that can be used to compute the state of the node at the following layer, i.e., $x_{v,1}$. More generally, at layer $k+1$ we have the use of the node label $l_v^{k+1} = x_{v,k}$. A different state transition function f_a^k may be exploited at each layer. Formally, the computation is performed by the following schema for each layer $k > 0$:

$$x_{v,k}^{(t)} = f_a^k \left(x_{\mathrm{ne}[v],k}^{(t-1)}, x_{v,k}^{(t-1)}, x_{v,k-1} \mid \theta_{f_a^k} \right), \tag{4.11}$$

whereas the states of the first layer $x_{v,0}$ are computed by Eq. (4.1). The model outputs are computed by applying the output network f_r to the states $x_{v,K-1}$

available at the last layer. The proposed implementation is a simplification of the more general model that may process the neighboring node and arc labels $l_{\text{ne}[v]}, l_{(v,\text{ne}[v])}$ (and also additional node labels to augment l_v) again at each layer[4].

Following the original GNN computation schema, the states need to be sequentially computed layer-by-layer applying the relaxation procedure to reach the fixed point: when moving to layer k the states computed by the previous layer $k-1$ are considered constant inputs, as shown in Eq. (4.11). As a result, Layered GNNs may be computationally demanding since they require to compute the fixed point for each layer in the forward phase, and to backpropagate the information through the resulting unfoldings in the backward phase.

In order to overcome these issues, we can exploit the locality in the computations of the proposed Lagrangian formulation by considering the state variables $x_{v,k}$ at each node and layer as free variables. Once we introduce new layer-wise constraints with the same structure of Eq. (4.3), the proposed learning problem can be generalized to multiple layers as follows:

$$\begin{aligned} &\min_{\Theta_{f_a},\theta_{f_r},X} \sum_{v\in S} L(f_r(x_{v,K-1} \mid \theta_{f_r}), y_v) \\ &\text{subject to} \quad \mathcal{G}(x_{v,k} - f^k_{a,v}) = 0, \quad \forall\ v \in V,\ \forall\ k \in [0, K-1] \end{aligned} \tag{4.12}$$

where $\Theta_{f_a} = \left[\theta_{f_a^0}, \ldots, \theta_{f_a^{K-1}}\right]$ collects the weights of the neural networks implementing the transition function of each layer, and X collect the states $x_{v,k}$ for each node and layer. The notation $f^k_{a,v}$ is a straightforward extension of the one proposed in Eq. (4.1), taking into account the schema defined by Eq. (4.11), where $f^k_{a,v}$, with $k > 0$, is function of x_v^{k-1}. In this context, $f^0_{a,v}$ corresponds to the original definition. Our approach not only allows us to jointly optimize the weights of the networks and diffuse the information along the graph, but also to propagate the information through the layers, where, for each of them, a progressively more informed diffusion process is carried on and optimized.

[4]The model can be also extended by considering an output function f^k_r for each layer such that the output $y_{n,k-1}$ at layer $k-1$ is concatenated to $x_{v,k-1}$ for each node as input for the following layer. These intermediate outputs can be subject to the available supervisions [110].

4.3.1 Deep LP-GNN Complexity analysis

The introduction of the layered computational structure empowers the representational capability of the model at the expense of an increased complexity. Given a K-layer LP-GNN, the computational complexity for each parameter update is $\mathcal{O}(K(|V|+|E|))$, where $|V|$ the number of nodes and $|E|$ the number of edges. The persistent state variable matrix requires $\mathcal{O}(K|V|)$ space.

4.4 Experiments

We implemented the algorithm described in the previous sections using TensorFlow[5]. The implementation exploits the TensoFlow facilities to compute the gradients (Eq. (4.7)-(4.10)), while we updated the parameters of the problem of Eq. (4.6) using the Adam optimizer [111]. For the comparison with the original GNN model, we exploited the GNN Tensorflow implementation[6] introduced in [106].

4.4.1 Qualitative analysis

When dealing with graphs, a very interesting analysis [89] consists in having a look at how latent representations of nodes (states) evolve during the learning process. Indeed, when the dependence of the state transition function on the available node-attached features (l_v^0) is reduced or completely removed, the algorithm can only rely on the topology of the graph to perform the classification task at hand. In this setting, the states are continuous representations of topological features of the nodes in the graph and the LP-GNN model would implicitly learn a metric function in this continuous space. For this reason, we would expect that nodes belonging to the same class are placed close to each other in the embedding space.

In order to perform this evaluation, we exploit the simple and well-known Zachary Karate Club dataset [112]. The data was collected from the members of a university karate club by Wayne Zachary in 1977. Each node represents

[5] `https://www.tensorflow.org`

[6] The framework is available at `https://github.com/mtiezzi/gnn`. The documentation is available at `http://sailab.diism.unisi.it/gnn/`

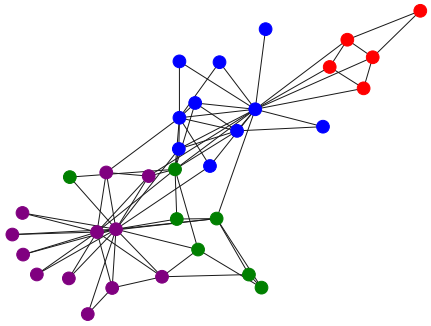

Figure 4.1: The karate club dataset. This is a simple and well-known dataset exploited to perform a qualitative analysis of the behaviour of our model. Nodes have high intra-class connections and low inter-class connections. Each color is associated to a different class (4 classes).

a member of the club, and each edge represents a tie between two members. There are 34 nodes, connected by 154 (undirected and unweighted) edges. Every node is labeled by one out of four classes, obtained via modularity-based clustering (see Figure 4.1).

We trained a layered LP-GNN with three state layers ($K = 3$). In order to visualize the node states and how they change over time, we set the state dimension of the last layer to 2 units and we used a shallow softmax regressor as output function, to force a linear separation among classes. Moreover, node-attached features of the given data were totally removed in order to force the algorithm to exploit only structural properties in the solution of the classification task. Figure 4.2 shows how the node states evolve over time, starting from an initialization composed of zero-only states (i.e. node states are initialized to a zero value). During training, they move progressively toward four distinct areas of the 2D embedding space, one for each class. Since features of nodes were removed, the distinct areas of the space group nodes only with similar topological features.

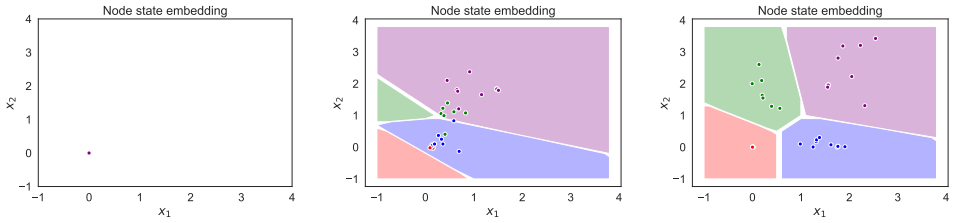

Figure 4.2: Evolution of the node state embeddings at different stages of the learning process: beginning, after 200 epochs and at convergence. We are not exploiting any node-attached features from the available data, so that the plotted node representations are the outcome of the diffusion process only, which is capable of mapping the topology of the graph into meaningful latent representations. Each node is represented with the color of the given corresponding class (ground truth), while the four background colors are the predictions of the output function learned by our model. The model learns node state embeddings that are linearly separated with respect to the four classes.

4.4.2 Artificial Tasks

We consider the two tasks of subgraph matching and clique localization. These tasks represent different challenges for the proposed model. We first describe the main features of the considered tasks, and, afterward, we describe the experimental results.

Table 4.1: The considered variants of the $\mathcal{G}$ function. By introducing ε-insensitive constraint satisfaction, we can inject a controlled amount (i.e. ε) of tolerance of the constraint satisfaction into the hard-optimization scheme.

Function $\mathcal{G}(x) =$	*lin* x	*lin-*ε $\max(x,\varepsilon) - \max(-x,\varepsilon)$	*abs* $\lvert x\rvert$	*abs-*ε $\max(\lvert x\rvert - \varepsilon, 0)$	*squared* x^2
Unilateral	×	×	✓	✓	✓
ε-insensitive	×	✓	×	✓	×

Subgraph Matching Given two graphs G and S such that $|S| \leq |G|$, the subgraph matching problem consists in finding the nodes of a subgraph $\hat{S} \subset G$ which is isomorphic to S. The task is that of learning a function τ_S, such that $\tau_S(G, n) = 1, n \in V$, when the node n belongs to the given subgraph S, otherwise $\tau_S(G, n) = 0$. The target subgraph S is predefined in the learning phase by providing examples of graphs G that contain S (the nodes of G that define the subgraph S have a supervision equal to 1). The problem of finding a given subgraph is common in many practical problems and corresponds, for instance, to finding a particular small molecule inside a greater compound. An example of a subgraph structure is shown in Figure 4.3. The dataset that we considered is composed of 100 different graphs, each one having 7 nodes. The number of nodes of the target subgraph S is instead 3.

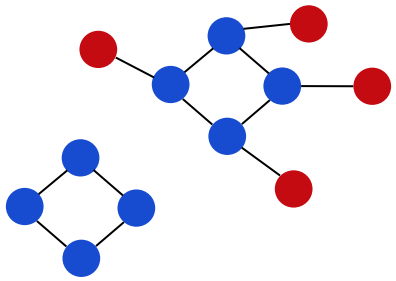

Figure 4.3: An example of a subgraph matching problem, where the graph with the blue nodes is matched against the bigger graph.

Clique localization A clique is a complete graph, i.e. a graph in which each node is connected with all the others. In a network, overlapping cliques (i.e. cliques that share some nodes) are admitted. Clique localization is a particular instance of the subgraph matching problem, with S being complete. However, the several symmetries contained in a clique makes the graph isomorphism test more difficult. Indeed, it is known that the graph isomorphism has polynomial time solutions only in absence of symmetries. A clique example is shown in Figure 4.4. In the experiments, we consider a dataset composed by graphs having 7 nodes each, where the dimension of the maximal clique is 3 nodes.

Table 4.2: Accuracies on the artificial datasets, for the proposed model (Lagrangian Propagation GNN - LP-GNN) and the standard GNN model for different settings.

Model			Subgraph		Clique	
	$\mathcal{G}$	ε	*Acc(avg)*	*Acc(std)*	*Acc(avg)*	*Acc(std)*
LP-GNN	*abs-ε*	0.00	96.25	0.96	88.80	4.82
		0.01	**96.30**	0.87	88.75	5.03
		0.10	95.80	0.85	85.88	4.13
	lin-ε	0.00	95.94	0.91	84.61	2.49
		0.01	95.94	0.91	85.21	0.54
		0.10	95.80	0.85	85.14	2.17
	squared	-	96.17	1.01	**93.07**	2.18
GNN [53]	-	-	95.86	0.64	91.86	1.12

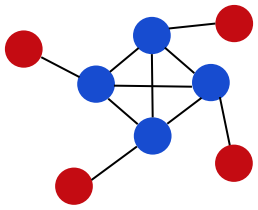

Figure 4.4: An example of a graph containing a clique. The blue nodes represent a fully connected subgraph of dimension 4, whereas the red nodes do not belong to the clique.

We designed a batch of experiments on these two tasks aimed at validating our simple local optimization approach to constraint-based graph networks. In particular, we want to show that our optimization scheme can learn better transition and output functions than the corresponding GNN of [53, 106]. Moreover, we want to investigate the behaviour of the algorithm for different choices of the function $\mathcal{G}(x)$, i.e. when changing how we enforce the state convergence constraints. In particular, we tested functions with different properties: *ε-insensitive functions*, i.e $\mathcal{G}(x) = 0,\ \forall x : -\varepsilon \leq x \leq \varepsilon$, *unilateral functions*, i.e. $\mathcal{G}(x) \in \mathbb{R}^+$, and *bilateral functions*, i.e. $\mathcal{G}(x) \in \mathbb{R}$ (a $\mathcal{G}$ func-

tion is either unilateral or bilateral). Table 4.1 reports the definition of the considered functions showing if they are ε-insensitive, bilateral or unilateral.

Following the experimental setting of [53, 106], we exploited a training, validation and test set having the same size, i.e. 100 graphs each. We tuned the hyperparameters on the validation data, by selecting the node state dimension from the set $\{5, 10, 35\}$, the dropout drop-rate from the set $\{0, 0.7\}$, the state *transition function* from $\{f_{a,v}^{(\text{SUM})}, f_{a,v}^{(\text{AVG})}\}$, where

$$f_{a,v}^{(\text{SUM})} = \sum_{u \in ne[v]} h(x_u, l_u, l_{(v,u)}, l_{(u,v)}, x_v, l_v \mid \theta_h)$$
$$f_{a,v}^{(\text{AVG})} = \frac{1}{|ne[v]|} \sum_{u \in ne[v]} h(x_u, l_u, l_{(v,u)}, l_{(u,v)}, x_v, l_v \mid \theta_h).$$

bearing in mind the various capabilities of the Sum and Mean aggregators, outlined in Section 4.1.2. The number of hidden units was selected from $\{5, 20, 50\}$. The learning rate for parameters θ_{f_a} and θ_{f_r} is selected from the set $\{10^{-5}, 10^{-4}, 10^{-3}\}$, and the learning rate for the variables x_v and λ_v from the set $\{10^{-4}, 10^{-3}, 10^{-2}\}$.

We compared our model with the equivalent GNN in [53], with the same number of hidden neurons in the f_a and f_r functions. Results are presented in Table 4.2. On average, LP-GNN perform favourably than vanilla GNN when the $\mathcal{G}$ function is properly selected. Constraints characterized by *unilateral functions* usually offer better performances than equivalent bilateral constraints. This might be due to the fact that keeping constraints positive (as in unilateral constraints) provides a more stable learning process. Moreover, smoother constraints (i.e *squared*) or ε-insensitive constraints tend to perform slightly better than the other versions, in line with the experimental findings of Local Propagation (Section 3.5). This can be due to the fact that as the constraints move closer to 0 they tend to give a small or null contribution, for *squared* and *abs-*ε respectively, acting as regularizers.

4.4.3 Graph Classification

Graph-focused tasks consists in finding a representation of the current input graph, yielding a single output, as stated by Eq. (2.11b). To extract this unique embedding from the representations encoded by all the states available

Table 4.3: Average and standard deviation of the classification accuracy on the graph classification benchmarks, evaluated on the test set, for different GNN models. The proposed model is denoted as LP-GNN and it is evaluated in two different configurations. LP-GNN-Single exploits only one layer of the diffusion mechanism, while LP-GNN-Multi exploits multiple layers as described in Section 4.3. It is interesting to note that, even if LP-GNN-Single exploits only a shallow representation of nodes, it performs, on average, on-par with respect to other state-of-the-art models. Finally, the LP-GNN-Multi model performs equally to or better than most of the competitors on most of the benchmarks.

Datasets	IMDB-B	IMDB-M	MUTAG	PROT.	PTC	NCI1
# graphs	1000	1500	188	1113	344	4110
# classes	2	3	2	2	2	2
Avg # nodes	19.8	13.0	17.9	39.1	25.5	29.8
DCNN	49.1	33.5	67.0	61.3	56.6	62.6
PatchySan	71.0 ± 2.2	45.2 ± 2.8	**92.6** ± 4.2	75.9 ± 2.8	60.0 ± 4.8	78.6 ± 1.9
DGCNN	70.0	47.8	85.8	75.5	58.6	74.4
AWL	74.5 ± 5.9	51.5 ± 3.6	87.9 ± 9.8	–	–	–
GraphSAGE	72.3 ± 5.3	50.9 ± 2.2	85.1 ± 7.6	75.9 ± 3.2	63.9 ± 7.7	77.7 ± 1.5
GIN	75.1 ± 5.1	**52.3** ± 2.8	89.4 ± 5.6	76.2 ± 2.8	64.6 ± 7.0	**82.7** ± 1.7
GNN	60.9 ± 5.7	41.1 ± 3.8	88.8 ± 11.5	76.4 ± 4.4	61.2 ± 8.5	51.5 ± 2.6
LP-GNN-Single	71.2 ± 4.7	46.6 ± 3.7	90.5 ± 7.0	77.1 ± 4.3	64.4 ± 5.9	68.4 ± 2.1
LP-GNN-Multi	**76.2** ± 3.2	51.1 ± 2.1	**92.2** ± 5.6	**77.5** ± 5.2	**67.9** ± 7.2	74.9 ± 2.4

at each node, we implemented the following version of the *readout* function:

$$y_G = f_r^{(\text{SUM})}\left(\{x_v, v \in V\} \mid \theta_{f_r}\right) = f_r\left(\sum_{v \in V} f_{a,v} \mid \theta_{f_r}\right). \tag{4.13}$$

We selected 6 datasets that are popular for benchmarking GNN models. In particular, four of them are from bioinformatics (MUTAG, PTC, NCI1, PROTEINS) and two from social network analysis (IMDB-BINARY, IMDB-MULTI) [113].

The MUTAG dataset is composed of 188 mutagenic aromatic and heteroaromatic nitro compounds, having 7 discrete labels. PTC is characterized by 344 chemical compounds belonging to 19 discrete labels, reporting the carcinogenicity for male and female rats. The National Cancer Institute (NCI) made publicly available the NCI1 dataset (4100 nodes), consisting of chemical compounds screened for their ability to suppress or inhibit the growth of

a panel of human tumor cell lines, having 37 discrete labels. Nodes in the PROTEINS dataset represent secondary structure elements (SSEs), with an edge connecting them if they are neighbors in the amino-acid sequence or in the 3D space. The set has 3 discrete labels, representing *helix*, *sheet* or *turn*. Regarding the social network datasets, IMDB-BINARY is a movie collaboration dataset collecting actor/actress and genre information of different movies extracted from the popular site IMDB. For each graph, nodes represent actors/actresses, connected with edges if they appear in the same movie. Each graph is derived from a pre-specified movie category, and the task is to classify the genre it is derived from. In the IMDB-B dataset, the collaboration graphs are labelled with the two genres *Action* and *Romance*. The multi-class version of this dataset is IMDB-MULTI, which is composed by a balanced set of graphs belonging to the *Comedy*, *Romance* and *Sci-Fi* labels.

The main purpose of this kind of experiments is to show the ability of the model to strongly exploit the graph topology and structure. In fact, whilst in the bioinformatics graphs the nodes have categorical input labels (l_v^0) (e.g. atom symbol), in the social networks sets there are no input node labels. In this case, we followed what has been recently proposed in [2], i.e. using one-hot encodings of node degrees. Dataset statistics are summarized in Table 4.3.

We compared the proposed Lagrangian Propagation GNN (LP-GNN) scheme with some of the state-of-the-art neural models for graph classification, such as Graph Convolutional Neural Networks. In particular, the models used in the comparison are: Diffusion-Convolutional Neural Networks (DCNN) [92], PATCHY-SAN [90], Deep Graph CNN (DGCNN) [94], AWL [114], GraphSAGE [93], GIN-GNN [2], original GNN [53]. For all the GNN-like models there are a number of layers equal to 5. We compared also two versions of our scheme: LP-GNN-Single, which is a shallow architecture with $K = 1$, and LP-GNN-Multi, which is a layered version of our model, as described in Section 4.3. It is important to notice that differently from LP-GNN-Single, all the convolutional models use a different transition function at each layer. This fact entails that, at a cost of a much larger number of parameters, they have a much higher representational power. Apart from original GNN, we report the accuracy as available in the referred papers.

We followed the evaluation setting of [90]. In particular, we performed 10-fold cross-validation and reported both the average and standard deviation of the accuracies across the 10 folds within the cross-validation. The stopping epoch is selected as the epoch with the best cross-validation accuracy averaged over the 10 folds. We tuned the hyperparameters by searching: (1) the number of hidden units for both the f_a and f_r functions from the set $\{5, 20, 50, 70, 150\}$; (2) the state transition function from $\{f_{a,v}^{(\text{SUM})}, f_{a,v}^{(\text{AVG})}\}$; (3) the dropout ratio from $\{0, 0.7\}$; (4) the size of the node state x_v from $\{10, 35, 50, 70, 150\}$; (5) learning rates for both the θ_{f_a}, θ_{f_r}, x_v and λ_v from $\{0.1, 0.01, 0.001\}$.

Results are reported in Table 4.3. As previously stated and as it will be further discussed in Section 4.4.4, the LP-GNN-Single model offers performances that, on average, are preferable or on-par to the ones obtained by more complex models that exploit a larger amount of parameters. Finally, the LP-GNN-Multi model performs equally to or better than most of the competitors on most of the benchmarks.

4.4.4 Depth vs Diffusion

It is interesting to note that for current convolutional GNN models [92, 90, 94, 114, 2, 93] the role of the architecture depth is twofold. First, as it is common in deep learning, depth is used to perform a multi-layer feature extraction of node inputs, providing more and more representational power as depth increases. Secondly, it allows node information to flow through the graph fostering the realisation of a diffusion mechanism. Conversely, our model strictly splits these two processes. Diffusion is realised by looking for a fixed point of the state transition function, while deep feature extraction is realised by stacking multiple layers of node states, enabling a separate diffusion process at each layer. We believe this distinction to be a fundamental ingredient for a clearer understanding of which mechanism, between diffusion and node deep representation, is concurring in achieving specific performances.

In the previous section, we showed indeed that our diffusion mechanism paired only with a simple shallow representation of nodes (reffered as LP-GNN-Single) is sufficient, in most cases, to match performances of much deeper and complex networks. In this section, we want to investigate further this aspect. In particular, we focused on the IMDB-B dataset. The choice of this dataset

has to be attributed to the fact that it contains no node features. In this way, as done in Section 4.4.1, we can assure that the only information available to solve the task is the topological one. Other datasets, for example the simpler MUTAG, reach high level accuracies even without the structural information of the graph, by only exploiting node features.

We compared our model with the state-of-the-art GIN [2] model. For both models, we tested four architectures with {1,2,3,5} GNN state layers. We want to show that in very shallow GNNs (one layer) our model can still perform fairly well, since the diffusion process is independent from the depth of the network. On the other side, the GIN model, as other graph convolutional networks, needs deep architectures with a larger number of parameters for the diffusion process to take place. We believe this to be a big advantage of our model w.r.t. convolutional architectures in all the cases where high representational power is not required.

Results are shown in Table 4.4. It can be noted that this task can reach the 96% of the top accuracies (75.1 and 76.2, respectively) using only 2 layers of GNN, for both the competitors. The great difference between the two approaches becomes clear in the architecture composed by only 1 layer. In this setting, the GIN model, like all the other convolutional architectures, can only exploit information contained in direct 1-hop neighbors, reaching a 52% accuracy (which is close to random in a binary classification task). On the contrary, our model can reach a 71.2% of accuracy (93% of the maximum accuracy). This is a signal of the fact that convolutional architectures need a second layer (and thus a larger number of parameters) mainly to perform diffusion at 2-hop neighbors rather than for exploiting a higher representational power.

4.5 Discussion

In This Chapter we proposed an approach that simplifies the learning procedure of GNN models, making them more easily implementable and more efficient. The formulation of the learning task is devised following the principles of locality and the computational graph decomposition into local subparts connected via constraints, avoiding the unfolding of the state convergence proce-

Evaluation	**Model**	**Number of State Layers**			
		1	2	3	5
Absolute	GIN [2]	52	72.6	72.7	75.1
	LP-GNN	**71.2**	**73.7**	**73.9**	**76.2**
Relative	GIN [2]	69	96	97	100
	LP-GNN	**93**	96	97	100

Table 4.4: Depth vs Diffusion analysis. Absolute (top rows) and Relative (bottom rows) test accuracies on the IMDB-B dataset when the number of GNN state layers varies from 1 to 5 (i.e. $K \in [1, 5]$. Here, the Relative accuracy represents the percentage of the current accuracy with respect to the maximum obtained performances. The state-of-the-art GIN [2] model and our proposed approach are compared.

dure. Indeed, the defined constrained optimization problem allows us to avoid the explicit computation of the fixed point, that is needed to encode the graph. The proposed framework defines how to jointly optimize the model weights and the state representations without the need of separate optimization stages. For this reason, our model can be easily implemented using modern machine learning libraries since it is completely based on BackPropagation. The constrained representation of the learning procedure can be replicated recursively multiple times, thus introducing different levels of abstraction and different representations, similarly to what happens in multi-layer networks.

We proposed and investigated constraining functions that allow the model to modulate the effects of the diffusion process. Our experiments have shown that the proposed approach leads to results that compare favourably with the ones of related models, and we investigated the effects of the constraining functions. Interestingly, the proposed constraint-based scheme can be extended to all the other methods proposed in the literature that exploit more sophisticated architectures or aggregation schemes. In such a way, the advantages brought by powerful aggregation schemes and the capability to carry on a diffusion process involving the whole graph, could be intermixed achieving performance improvements.

We will highlight some future research lines in Chapter 6.

Chapter 5

Feature learning in video streams

One of the key components of complex cognitive systems is their ability to continuously process streams of data, in a real-time manner. Following a conversation, reading a paragraph of a book or looking at the world surrounding us, are activities linked by their sequential nature and characterized by a unique temporal direction. In experiencing such activities, the human brain processes a continuous flow of information, where the *local* statistics attached to the currently processed pattern rely heavily on the preceding one, and it is highly interleaved with the pattern that comes afterwards, along this temporal direction. The complexity of the cognitive mechanisms derives from a continuous learning process that happens in this scenario. Indeed, the human brain is capable of taking advantage of this context carrying on a lifelong process of learning, where experience and notions are accumulated and enriched over time, leveraging the continuity and coherence present in such data stream. Moreover, intelligent agents do not need an always available oracle ready to supervise and correct their errors. Supervisions are instead sparse in time, and even without any hints or suggestion agents are capable to grasp relevant features. Such consideration holds for various tasks, but here we will consider the human visual system as a case study [115].

Conversely, the standard Deep Learning training procedure in Computer Vision tasks is based on a heavily supervised scenario, where batches of shuffled and unrelated frames (e.g. ImageNet [116]) are exploited in order to train neural models. Despite the fact that the recent literature is moving towards the analysis of video streams [117, 118], or computer simulations of the real world [119, 120, 121], these solutions still heavily rely on densely annotated frames (in the case of Video Object detection or segmentation) or on the evaluation of policies (Navigation, Visual QA).

However, there is an increasing interest in grasping more powerful cogni-

tive capabilities. The lines of research on Continual and Lifelong Learning [122] represent a step towards models able to act in an online manner [123], overcoming difficulties such as the *catastrophic forgetting* [36] of previously learned tasks, solely based on parsimonious principles.

Unsupervised Learning, on the other hand, tackles the problem of the need of massive supervision. A proper artificial cognitive model should have also the ability to reason on the whole input state without the need of any auxiliary supervised information. Several popular methods try to discover an useful representation in an unsupervised manner. Independent component analysis [124], Self-Organizing Maps [125], neural Clustering techniques [126] or self-supervision [127, 128] represent some of the techniques used in Unsupervised Learning. Another interesting approach follows the InfoMax principle [129], a feature learning technique based on the maximization of the transferred information, in terms of Mutual Information (MI), among inputs and outputs of neural networks. The resurgence of such techniques, allowed by scalable approximations [130, 131], is having an impact on this research field.

In this Chapter, we focus on unsupervised learning for transferring visual information in a truly online setting by using a computational model that is inspired to the principle of least action in physics, denoted as the Cognitive Action Laws (CALs) [132]. The maximization of the mutual information is carried out by a temporal process that yields online estimation of the entropy terms. One of the key issues with MI maximization over time is the fact that stochastic updates of the model parameters do not keep track of the entropy of the output space due to the data processed so far, leading to poorly informed updates. We investigate the case in which the global changes in the entropy of the output space are approximated by introducing a specific constraint.

In the context of this dissertation, the entropy approximations yielded at each time instant are regarded as components of the same temporal computational model. The *temporally local* entropy approximations are subcomponents of the overall architecture, put into relation by soft-constraints that enforce a temporal estimate that is not limited to the current frame. We intersect these ideas with the recent human-like attention model of [133], that has shown state-of-the art results in human visual scanpath estimation. The focus of attention implements a filtering procedure on the input video stream, allow-

ing the system to deal only with those areas that would attract the human attention.

5.1 Preliminaries and Related Work

This section will give some background notions on several research directions and related works in the context of unsupervised learning and learning over time.

5.1.1 Lifelong learning over time

Continual lifelong learning [122] represents a long-standing challenge for the creation of intelligent artificial agents. This term refers to the ability to learn over time grasping new knowledge while retaining previously learned notions. The simplest approach to learn new knowledge, i.e. retraining the neural model parameters with the novel data, in fact suffers from the issue of *catastrophic forgetting*. The new information causes the parameters to change in such a way that patterns belonging to the older data may be no more correctly classified. Many techniques have been devised in order to deal with this issue, many of them inspired by the *stability-plasticity dilemma* [134]. In order to be able to learn over time and overcome catastrophic forgetting, a system must be capable to balance between the integration of novel information, hence being *plastic*, and the retention of previously learned knowledge, hence being *stable*.

Machine learning approaches to this task involve dynamical architectural properties, Complementary Learning Systems or Memory Replay (see [122] for a complete review) as well as methods inspired by *parsimonious principles* [134] and regularization approaches [135] (for instance, Slow Feature Analysis [136]).

5.1.2 Unsupervised Learning by Mutual Information Maximization

The goal of Unsupervised Learning is to discover an useful representation of unlabeled data available at-hand, which is often gained employing a learnable function implemented by an ANN. Features developed in an unsupervised

manner can be considered by themselves meaningful and even capable to reduce the burden and ease supervised training in downstream tasks.

In this direction, the information-based topic of Mutual Information (MI) maximization has recently attracted the attention of several researches [130, 131, 137, 138, 139]. These works are inspired by the InfoMax principle [129], that can be briefly described as the search of a representation $g(x)$ of the input pattern x such that the MI between the input and the representation, that can be interpreted as the computed symbols, is maximized. Indeed, the encoder $g(x)$ must act in such a way that the information transfer between its inputs and the generated symbols is maximized. The Mutual Information $I(Y, X)$ between two random variables X and Y measures the reduction of uncertainty about Y by virtue of the observation on the value of X.

Given this definition, we can formally express the MI as

$$I(Y, X) = H(Y) - H(Y|X) \tag{5.1}$$

where $H(\cdot)$ denotes the *entropy* of a random variable, which measures the uncertainty of a probability distribution defined on the realizations x of a random variable X[1], $H(X) = -\sum_x p(x) \log(p(x))$.

Most of the recent works are based upon customized MI-based criteria to learn representations for downstream tasks, in the setting of unsupervised image representation. Moreover, the approaches of [131, 137] are based on surrogate functions that loosely approximate [139] the MI formulation, that is very difficult to compute in the context of continuous random variables and high dimensional settings.

The common approach shared by these works can be conceptually summarized with the following problem setup, as highlighted in [139]. The random variable X represents the input image, while we denote with $X^{(1)}$ and $X^{(2)}$ two different *views* of the image (for instance two subparts or different augmentation/image modalities). The problem consists in maximizing an estimated MI $I_{EST}(\cdot)$ between the representations of such view:

$$\max_{g_1, g_2} \quad I_{EST}\big(g_1(X^{(1)}), g_2(X^{(2)})\big) \tag{5.2}$$

[1]Random variables are denoted using upper case letters (X, Y) while we use lower case letters (x, y) for their realizations.

which is a lowerbound on the InfoMax [129] objective $max_g \quad I(X, g(X))$. Several works can be described under this common setting.

DeepInfomax [131] empirically proves that maximizing the MI between the complete input X and its representation $g(X)$ is often insufficient for learning useful representations. Hence, the authors propose the maximization of an approximated continuous MI, obtained through an approach inspired by MINE [130], between global features extrapolated from the image ($g_1(X^{(1)})$) and *local features* obtained from image patches ($g_2(X^{(2)})$). This encourages the encoder $g(\cdot)$ to grasp the useful information present in all locations of the input image, with the hope of being globally relevant. The MI estimation is made easier by the fact that the views lie on lower dimensional spaces with respect to the original image.

Contrastive Predictive Coding (CPC) [138] maximizes the MI between global and local representations, enriched with an ordered autoregression over local features. In particular, aggregated features are obtained from an ordered sequence of t patches. Then, the MI index between such aggregated statistics and the representation of a local feature not processed yet (at location $t + k$, hence the name *predictive*) is maximized. In reference to Eq. (5.2), here $X^{(1)}$ corresponds to the first t patches, and $X^{(2)}$ to the patch at position $t + k$.

Given the burden of computing the continuous MI index, the methods presented up to now heavily rely on the estimator $I_{EST}(\cdot)$, that deals with lower bounds of the real MI [139]. In the remaining of the Chapter, we will instead consider the discrete MI index, that, for instance, has been previously used as a criterion to relate different views of the input data [140] or in clustering [141].

The information transferred by multi-layer networks is discussed also in the context of the popular *information bottleneck principle* by Naftali Tishby and other authors as a mean to study deep network internal dynamics [142, 143, 144].

5.1.3 Cognitive Action Laws: Learning Over Time

The methods considered in Section 5.1.2 deal with MI maximization on image representation tasks, leveraging batches of data that are commonly processed in an offline manner using stochastic updates of the model parameters, peri-

odically shuffling the available samples.

The main goal of this Chapter is Unsupervised learning from continuous visual streams, that is indeed a challenging problem that cannot be naturally and efficiently managed in the aforementioned batch-mode setting of computation, as highlited in Section 5.1.1.

Recent studies, inspired by *parsimony principles* and information-based statistics, connected learning over time and classical mechanics [132, 145, 146]. The framework proposed in [132] naturally deals with learning problems in which time plays a crucial role, and it is well-suited to learn from streams of visual data in a principled way. The temporal trajectories of the variables of the learning problem are modeled by the so called 4th order Cognitive Action Laws (CALs), that come from the stationary conditions of a functional, as it happens for generalized coordinates in classical mechanics.

The Lagrangian Mechanics Formalism

By and large, the Lagrangian formalism in classical mechanics models a physics system with coordinates $x_t = (q, \dot{q})$, where the vector q is a point in the configuration space of the system and $\dot{q}$ denotes its time derivative. In the time interval $[t_0, t_1]$ the system starts in state x_0 and ends in state x_1. Each path taking from x_0 to x_1 has an associated scalar value called *Action*. In Lagrangian mechanics, such value is given by the functional:

$$\Gamma := \int_{t_0}^{t_1} K(q_t, \dot{q}_t) - V(q_t, \dot{q}_t))dt \tag{5.3}$$

where K is the kinetic energy and V is the potential energy, that compose the arguments to the functional and are referred to by the term *Lagrangian*, denoted with $L = K - V$. Moreover, if the system is defined in such a way, the only path that the system will take is the stationary value of Γ. The enforcement of the stationary condition, i.e. $\partial\Gamma = 0$, is obtained via the definition of the *Euler-Lagrange* constraint equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_j} = \frac{\partial L}{\partial q_j} \tag{5.4}$$

Cognitive Action Laws

We consider the problem of processing a stream of data over time and, in particular, a stream of video frames $t \mapsto u(t)$ from a target source, being $u(t)$ the frame at time t in the time horizon $[0, T]$. The stream is processed by a neural network whose weights and biases at time t are represented by the generic vector variable $w(t)$, while $\dot{w}(t)$, $\ddot{w}(t)$ are respectively its first and second derivatives. In this context, it is possible to draw a link with the principle of least action in analytic mechanics, in such a way that the neural network weights configuration $w(t)$ can be treated as the aforementioned Lagrangian coordinates of a system of particles. In this formulation of learning, the potential energy of the system V can be put into relation with the loss function commonly used in machine learning, while a more general form of the kinetic energy K is leveraged to enforce the temporal evolution of the model parameters.

In particular, commonly the performances of a learning agent are measured using a pattern-related loss function, which here we denote with $v(\cdot)$, with the purpose of evaluating how appropriate the current network weights configuration w is, with respect to the pattern-at-hand. Moreover, learning happens minimizing an *empirical functional risk* where the input patterns x_k are extracted from the sample space, usually exploiting some random sampling strategies.

$$V(w) = \frac{1}{\ell} \sum_{k=1}^{\ell} v(w, x_k) \tag{5.5}$$

In an online learning framework, we are interested in leveraging a different intuition guiding the functional risk, where we assume there is a trajectory $t \mapsto u(t)$ in the pattern space that slides along a continuous temporal manifold. Input patterns are not extrapolated from a random sampling process, but from a temporally coherent signal (video stream, in the considered scenario). In this setting, we denote with $u(t)$ the pattern available at time t. Given this dynamics, the parallel with mechanics becomes more clear, and justifies the usage of the more evocative term *potential*. Hence, $V\big(w(t), u(t)\big)$ is referred to as the potential of the system defined by Lagrangian coordinates w. Following this duality, we look for trajectories of the weights $t \mapsto w(t) \in \mathbb{R}^n$ that

possibly lead to configurations with small potential energy, $V\big(w(t), u(t)\big) \approx 0$. Following the duality with mechanics, the notion of kinetic energy is introduced, mainly acting as a temporal regularization term which minimization leads to develop weights that settle to constant values. In the following, we will describe how such configuration can be obtained.

CAL: problem formulation

In [132], learning is described, in analogy with the classical mechanics setting described by Eq. (5.3), as a variational problem whose objective is to find a stationary point of the following functional $\Gamma^{(\xi)}$ of the maps $t \mapsto w(t) \in \mathbb{R}^n$,

$$\begin{aligned}\Gamma^{(\xi)}(w) &:= \int_0^T L(t, w(t), \dot{w}(t), \ddot{w}(t))\, dt = \\ &= \int_0^T h(t)\big(K(\dot{w}(t), \ddot{w}(t)) \quad - \quad \xi V(w(t), u(t))\big)\, dt. \end{aligned} \tag{5.6}$$

The Lagrangian L is composed of a kinetic energy K and a potential energy V (see Eq. (5.3)), while $h(t)$, when appropriately chosen, is responsible of introducing energy dissipation. In fact, the optimization is generally formulated over large time spans, i.e. lifelong visual streams. The term $h(t)$ is a discount factor that enforces the system to forget very old information, yielding dissipation and defining the time direction via a monotone increasing form. The term $\xi \in \{-1, 1\}$ is selected in function of the way K is implemented (see [145] for details). In particular, in [145, 146, 132] we have $\xi = -1$, $h(t) = e^{\theta t}$, V is composed of the loss function U of the considered problem and a quadratic regularizer on $w(t)$, and K includes the squared norm of the derivatives plus their dot product, leading to

$$\begin{aligned}\Gamma^{(-1)}(w) \equiv \Gamma(w) = \quad & \int_0^T e^{\theta t}\big(\tfrac{\alpha}{2}|\ddot{w}(t)|^2 + \tfrac{\beta}{2}|\dot{w}(t)|^2 + \gamma \dot{w}(t) \cdot \ddot{w}(t) + \\ & + \tfrac{k}{2}|w(t)|^2 + U(w(t), u(t))\big)\, dt \end{aligned} \tag{5.7}$$

where $\theta \in \mathbb{R}$ and α, β, γ, k are custom positive scalars, $|\cdot|$ is the Euclidean norm in $\mathbb{R}^n$ and $\cdot$ is the standard scalar product in $\mathbb{R}^n$, being n the size of $w(t)$.

The regularization terms involving the weights come from the connections with the *parsimony principle* mentioned in Section 5.1.1, with the purpose to

enforce spatial and temporal smoothness. In particular, the spatial smoothness is enforced by penalizing solutions characterized by abrupt spatial changes, hence having high spatial derivatives. Temporal smoothness is gained via the kinetic terms K, that avoid quick temporal transitions in the solutions by penalizing high temporal derivatives. The kinetic energy can be also interpreted as a regularization term which minimization enforces the weights development to settle to constant values. The peculiar choice on the regularization terms (deeply analyzed in [132]) guarantees the existence of a minimum under given conditions.

The Euler-Lagrange (EL) equations of Eq. (5.7) yield the Cognitive Action Laws (CALs), 4th order differential equations that, when integrated, allow w to be updated over time. In particular, they are[2]

$$\alpha w^{(4)} + 2\theta\alpha w^{(3)} + (\theta^2\alpha + \theta\gamma - \beta)\ddot{w} + (\theta^2\gamma - \theta\beta)\dot{w} + kw + \nabla U(w, u) = 0 \quad (5.8)$$

being $w^{(4)}$ and $w^{(3)}$ the fourth and third derivatives of w, respectively, and ∇U is the gradient of U with respect to its first argument. Cauchy's initial conditions can be provided on w and $\dot{w}$, while stationarity conditions of Γ prescribe that Eq. (5.8) must be paired with boundary conditions on the right border ($t = T$).

CAL: Boundary Conditions and Causality

The whole learning process is based upon the idea of solving the problem in a *temporally local* and *causal* way. These two requirements are strongly interleaved, requiring the usage of quantities local in time (relative to the current time step t) to solve the problem of determining $w(t)$. In particular, in order to achieve a truly online update of the model parameters, we need a causal dynamic computational model as long as the requirement of the satisfaction of the boundary conditions.

In fact, we can define a system as *causal* if for any t_0 the state at time t_0 uniquely determines the state at time $t \geq t_0$. Hence, the solution w at time t must not depend on "future" values, belonging to the interval $(t, T]$. Conversely, the Cauchy's initial conditions for the right border impose their

[2]We removed the time index to simplify the notation. We will do it occasionally also in the following.

satisfaction at the end of learning ($t = T$). But the evolution of the involved quantities depends on the whole video signal, i.e. the "future". This issue is due to an inherent incompatibility between Evolution problems and Variational problems for integral functionals. To gain back causality, the fulfilment of the boundary conditions in $t = T$ is approximated in [145] by introducing a mechanism that sets ("resets") to zero all the derivatives up to $w^{(3)}$, whenever their norms become too large. In such a way, it is possible to satisfy the boundary conditions in small portions of the analyzed video stream [132].

The $U(w(t), u(t))$ term in the functional of Eq. (5.7) is a potential that describes the interaction with the environment, i.e. the frame $u(t)$ of the video stream, through its dependence on time. It can be instantiated in various manners, for instance in a supervised setting as a loss function measuring the fitting of the example-target pair, or to enforce other dynamics (visual motion coherence in [132]).

In an unsupervised learning framework, the U term can be exploited to solely leverage information theory-based techniques, that will be presented in the following.

5.2 Introducing Second-Order Laws

Despite their robust principled formulation, the main drawbacks of the 4th order CALs is the difficulty in tuning the parameters (α, β, γ, k) that weigh the contribution of the derivatives, and the computational/memory burden due to the integration of a 4th order ODE. Moreover, the theoretical guarantees on the stability of Eq. (5.8) are experimentally shown to not be necessarily needed, mostly due to the aforementioned derivative reset procedure [145]. For these reasons, we will use the CAL theory in a particular *causal* regime of the parameters for which two important simplifications are attained. First, the dynamics of the weights are described by a 2nd order ODE (instead of Eq. (5.8)). Second, we get direct causality without the need of any reset mechanisms.

The limiting procedure that leads to the 2nd order laws is based on a conjecture by De Giorgi [147] that has been subsequently proved and studied in [148, 149, 150]. By and large, this approach defines a solution for Evolutive

problems, as a limit of the trajectory of a series of a family of functionals.

In detail, we consider a reparametrization in terms of $\varepsilon > 0$ of the Γ functional (Eq. (5.7)), where $\theta \to -1/\varepsilon$, $\alpha \to \varepsilon^2\alpha$, $\beta \to \varepsilon\beta$. This allows us to rewrite Eq. (5.7) in line with De Giorgi's functional,

$$\Gamma_\varepsilon(w) := \int_0^T e^{-t/\varepsilon} \left(\frac{\alpha\varepsilon^2}{2}|\ddot{w}(t)|^2 + \frac{\beta\varepsilon}{2}|\dot{w}(t)|^2 + \frac{k}{2}|w(t)|^2 + U(w(t), u(t)) \right) dt \quad (5.9)$$

where we also chose, for simplicity, $\gamma = 0$. Letting $\varepsilon \to 0$, the minima of the functional Γ_ε with fixed initial conditions on w and $\dot{w}$ converges to the solution of a Cauchy problem based on a 2nd order differential equation, thus gaining full causality, i.e., ε measures the "degree of causality" of the solution. Notice that the factor $e^{-t/\varepsilon}$ in Eq. (5.9) becomes peaked on $t = 0$ as $\varepsilon \to 0$, exponentially suppressing the Lagrangian at later times. Hence, the minimization procedure of Γ_ε will be mainly concerned in the minimization of the loss calculated at $t = 0^+$. At a first glance, this might seem counter-intuitive. However, it becomes a useful feature when considered in conjunction with the properties of the input signal $u(t)$.

Let us indicate with $\tau > 0$ the temporal scale of $u(t)$, that is a small time span under which the variations of the input frame $u(t)$ with respect to the previous considered time instant are semantically negligible.
The whole temporal interval $[0, T]$ can be partitioned into $\lceil T/\tau \rceil$ disjoint intervals $[0, \tau), [\tau, 2\tau), \ldots, [(\lceil T/\tau \rceil - 1)\tau, T)$, in each of which the aforementioned picky behaviour is not critical in evaluating the functional, due to the temporal scale of $u(t)$. Indeed, the time span $\tau > 0$ is so small that the peaked term is able to comprise the whole temporal window of the integration.

The minimization of Eq. (5.9) can be iteratively defined by minimizing Γ_ε in each interval (hence, a series of minima problem), where the conditions on the left boundary are given by the solution of the minimization in the previous interval. In particular, each one of these minimization procedures solves a non causal problem. When $\varepsilon \ll \tau$, the minimization problem can be well interpreted in terms of the value of $U(\cdot, u(\kappa\tau))$, for $\kappa = 0, \ldots, \lceil T/\tau \rceil - 1$. Hence, the condition $\varepsilon \ll \tau$ assures the approximation of a problem that is indeed causal. When $\varepsilon > 0$ the problem is non-causal due to the negative exponential and the sequence of problems, however as $\varepsilon \to 0$ the solutions of this problem will converge (in some sense) to the solution of a single causal

problem.

To introduce the EL equations of the newly introduced problem, for simplicity, we will describe the limiting procedure in the interval $[0, T]$, that applies to each of the $\lceil T/\tau \rceil$ previously described intervals. The EL equations for the minimizer of Γ_ε with initial conditions $w(0) = w^0$ and $\dot{w}(0) = w^1$ are

$$\begin{cases} \varepsilon^2 \alpha w^{(4)}(t) - 2\varepsilon\alpha w^{(3)}(t) + (\alpha\epsilon^2 - \varepsilon\beta)\ddot{w}(t) + \beta\dot{w}(t) + kw(t) + \nabla U(w(t), u(t)) = 0; \\ w(0) = w^0, \quad \alpha\dot{w}(0) = \alpha w^1, \quad \alpha\ddot{w}(T) = 0, \quad \alpha\varepsilon w^{(3)}(T) = \beta\dot{w}(T), \end{cases} \tag{5.10}$$

and the following theorem holds:

Theorem 1. *The solution of the problem* (5.10) *converges (weakly in* $H^1\big((0,T), \mathbb{R}^n\big)$ *to the solution of*

$$\begin{cases} \alpha\ddot{w}(t) + \beta\dot{w}(t) + kw(t) + \nabla U(w(t), t) = 0; \\ w(0) = w^0, \qquad \dot{w}(0) = w^1. \end{cases} \tag{5.11}$$

Equations (5.11) are 2nd order CALs. They are simpler than the 4th order CALs of Eq. (5.8), even if they maintain their principled nature[3].

5.2.1 A Temporally local computational model

Thanks to the introduction of the Second Order Laws and the aforementioned limiting procedure, in each interval the learning process carries on the minimization of a functional that is *temporally local*, thanks to the gained causality of the system. This means that in every considered interval $\lceil T/\tau \rceil$, the learning process moves the learnable weights towards the best configuration for the current time span, with boundary conditions depending solely on local quantities. The temporal locality of this approach fosters a computational model view of the problem (see Chapter 2), where this temporal architecture can be easily translated into a DAG. The temporal configurations assumed by the model can be interpreted as nodes of the whole architecture, a DAG defined on the input stream, where each node computation depends solely on the local weight configuration $w(t)$ and the current input frame $u(t)$. This particular intuition will come in handy in the following.

[3]See the Appendix of [40] for formal proofs and further details.

5.3 Mutual Information in Video Streams

We consider the problem of transferring information from a continuous, and potentially lifelong, input visual stream to the output space of a multi-layer convolutional network with ℓ layers.

We denote with the term *Retina* and symbol R the bidimensional plane composed by the pixels of each input frame. The pixel coordinates $x \in R$ identify its position inside the frame. The neural architecture processes each frame and yields m *pixel-wise* predictions, being m the size of the filter bank in layer ℓ. For each pixel, the i-th prediction can be interpreted as the probability of attaching to that particular pixel the i-th symbol (denoted with y_i) of a *discrete vocabulary* composed by m symbols. In order to do so, hyperbolic tangent is used as activation function in each layer $j < \ell$, while the last layer is equipped with a softmax activation, generating m probabilities $p(w, x, u) = (p_1(w, x, u), \ldots, p_m(w, x, u))$, being once again x a pair of pixel coordinates and u the processed frame. Notice that these quantities represent conditional probabilities, i.e. probabilities of the output symbol conditioned on the actual weight configuration, the processed pixel and the frame.

The goal of the learning process is the maximization of the information transfer from the input to the generated symbols, that corresponds to the maximization of the Mutual Information (MI) from the pixels of the input frames to the m-dimensional output space yielded by the m units of the last layer, as depicted in Figure 5.1.

This problem is studied in [132] and related papers [146, 145], where single-layer models (or stacks of sequentially trained single-layer models) are considered, while, in this dissertation, we exploit a deep network trained end-to-end. Previous approaches based on kernel machines can be found in [151, 152].

In order to define the MI index, we consider a generic, time independent weight configuration $\omega \in \mathbb{R}^n$. Moreover, we define the random variables X, associated with the input spatio-temporal probability distribution, while the random variable Y specifies the probability distribution over the output symbols. Since we are dealing with convolutional features, a realization of the random variable X is specified by the coordinates of a point $x \in R$, the value of the temporal instant t and the value of the video $u(t)$, hence with the triple $\big(x, t, u(t)\big)$. The MI between X and Y follows the laws defined in Eq. (5.1),

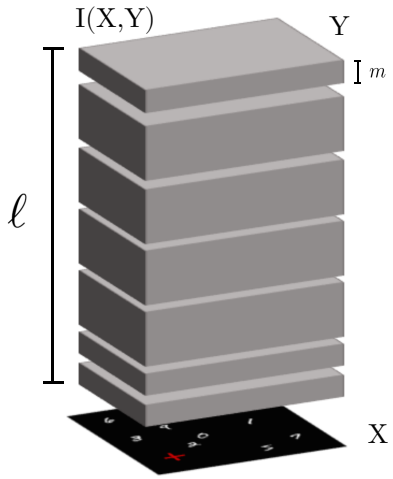

Figure 5.1: The goal of the learning process is the maximization of the information transfer ($I(X,Y)$) from the input visual stream X to the m-dimensional output space Y of a multi-layer convolutional network with ℓ layers.

but in this setting we deal with an estimation of this quantity that comprises a temporal interval, i.e. a continual stream. In order to do so, we introduce the average output activation on the video portion between time instants t_1 and t_2,

$$P(\omega, t_1, t_2) \equiv \int_{t_1}^{t_2} \overline{P}(\omega, t)dt := \int_{t_1}^{t_2} \int_R p(\omega, x, u(t))\mu(x, t)dxdt, \qquad (5.12)$$

where $\mu(x,t)$ is a *spatio-temporal* density and R is the aforementioned Retina. When no further information is available, μ is commonly assumed to be uniform in time and space and it is normalized such that $\int_{t_1}^{t_2} \int_R \mu(x,t)dxdt = 1$. The $\overline{P}(\omega,t)$ symbol denotes an instantaneous probability of the output symbols on the whole Retina, computed via the total probability rule in terms of the conditional probabilities $p(w,x,u) = (p_1(w,x,u), \ldots, p_m(w,x,u))$.

Given this definition, we can obtain the entropy of the output symbols

between time instants t_1 and t_2,

$$H(Y;\omega;t_1,t_2) = -\sum_{j=1}^{m} P_j(\omega,t_1,t_2)\log P_j(\omega,t_1,t_2) \tag{5.13}$$

simply following the definition in the case of discrete random variables.

Moreover, we can define the conditional entropy between time instants t_1 and t_2,

$$H(Y|X;\omega;t_1,t_2) = -\sum_{j=1}^{m}\int_{t_1}^{t_2}\int_{X} p_j(\omega,x,u(t))\log p_j(\omega,x,u(t))\,\mu(x,t)dxdt \tag{5.14}$$

We can then plug Eq.(5.13) and (5.14) into the MI definition (Eq. (5.1)), in order to define the MI index over the video portion $[t_1,t_2]$,

$$I(X,Y;\omega;t_1,t_2) = H(Y;\omega;t_1,t_2) - H(Y|X;\omega;t_1,t_2)$$

In order to better cope with the optimization dynamics, the two entropy terms are commonly weighted by positive scalars λ_e, λ_c, weighing the entropy and conditional entropy, respectively.

5.4 Constraining predictions over-time

Performing maximum-MI-based online learning of w using the CALs in the time horizon $[t_1 = 0, t_2 = T]$ is not straightforward. As highlighted in Section 5.1.3, the online learning mechanics yielded by the second order CALs would require to plug (minus) the MI index as a potential loss U in the Lagrangian. We can restore the dependency of w on time in the MI computation (Eq. (5.15)), by inserting $w(t)$ in place of ω, however we can see that the MI index computation is *non-local* in time. Indeed, formally the computation of the probability of the symbols (needed to compute the entropy) does require information regarding the whole time span $[t_1 = 0, t_2 = T]$ of the video stream. Unfortunately, this requirement breaks both the causality and temporally locality properties. Actually, as noticed in Section 5.2.1, in order to implement online learning dynamics, U must be temporally local, i.e., it should depend on w and u at time t only. For this reason, the authors of [132] enforce time

locality computing the MI index at time t, and not in an interval. With this method it is possible to obtain an approximation of the MI in $[0, T]$, yielded by the outer integration in the functional of Eq. (5.9) (or, equivalently, in the one of Eq. (5.7)).

A drawback of this formulation is that, due to this temporal assumption, it could lead to a loose approximation of the original term $H(Y; \cdot; \cdot, \cdot)$ of Eq. (5.15), for which the inner integration on time (Eq. (5.12)) is lost, and replaced by the outer integration of the functional. This corresponds to an averaging at frame level, hence obtaining a biased view on the symbols probability. Carrying on this kind of update hardly keeps track of the entropy of the output space due to the data processed so far, potentially leading to poorly informed updates.We denote such approach of entropy temporal estimation with PLA.

Following the intuition presented in Section 5.2.1, we explore an alternative criteria to mitigate the impact of time locality. The whole online process that yields the frame probability predictions obtained at each time instant can be treated as a temporal computational model. If the requirements of causality and time locality are met, the computations ideally happen solely leveraging local quantities.

To fulfill this requirements, and similarly to what we presented in the previous Chapters, we decompose the temporal computational model into local subcomponents, which are put into communication through the mathematical notion of constraint. In particular, we introduce an additional auxiliary local variable $s(t)$, that is used to replace P of Eq. (5.12), with the purpose of creating a quantity capable to define a temporal estimate which is not limited to the current frame. In order to do so, we constrain its variation, $\dot{s}(t)$, to be almost equivalent to $\overline{P}(w(t), t)$, the instantaneous output symbol probability. The Lagrangian is augmented with the differential soft-constraint

$$\lambda_s |\dot{s}(t) - \overline{P}(w(t), t)|^2 \tag{5.15}$$

that enforces $s(t)$ to approximate the case in which the probability estimate is not limited to the current frame. The constant λ_s is exploited to weigh the enforcement of the constraint in the total optimization process. Note that probabilistic normalization must be enforced after every update of $s(t)$.

Following this approach, the local auxiliary variable $s(t)$ aggregates temporal information over time, both considering the probability of the output symbol in the current frame ($\overline{P}(w(t), t)$), and the previous time instants, contained in the internal state of the $s(t)$ variable. We denote this approach with the symbol VAR.

This formulation fosters the decomposition of the global entropy estimation through quantities that are able to cope with temporal information, but are indeed *local in time*. In such a way, the requirement of causality is fulfilled, and the online entropy estimation can be injected into learning process (as a potential in the U term) via the causal Second Order CALs.

Inspired by this solution, we propose a second temporal criterion, (AVG), that exploits a similar approach without directly enforcing a constraint into the Lagrangian. We propose to replace P with the outcome ν of an averaging operation that keeps track of the past activation of the output units,

$$\nu(t) = \zeta_s \nu(t') + (1 - \zeta_s)\overline{P}(w(t), t) \tag{5.16}$$

for two consecutive time instants $t > t'$. The intuition is similar to the VAR approach, with the purpose of keeping the *memory* of the past prediction in order to produce a more informed entropy temporal estimation. In this case, it is this averaging operation that indirectly connects the temporal subcomponents of the overall computational model.

5.5 Restricting Computations on Salient Areas

While the previous Section deals with defining a proper temporal probability distribution of the learning process, other interesting considerations can be carried out in terms of the spatial distribution. In fact, the way video data is commonly processed by machines usually lacks a key property of the human visual perception, that is the capability of exploiting eye movements to perform shifts in selective visual attention. High visual acuity is restricted to a small area in the center of the retina (fovea), and the purpose of the Focus Of Attention (FOA) is to selectively orient the gaze toward relevant areas with high information, filtering out irrelevant information from cluttered visual scenes [153, 154, 133] (on the right in Figure 5.2). As a result, the

Figure 5.2: On the left, a depiction of the spatial uniform probability density commonly exploited in image procesing – all the pixels equally contribute to the learning process. On the right side, a human-like focus of attention, denoted with the red cross, filters relevant information in the visual scene.

FOA is a natural *spatial filtering* scheme that humans apply to the visual stimuli, and that allows them to explore and understand also complex scenes in an efficient way, discarding the unneeded visual information. Conversely, the common image processing pipeline implicitly assumes that all the pixels equally contribute to the learning process, hence considering a uniform spatial probability distribution of their coordinates over the retina, as depicted on the left in Figure 5.2.

In the context of Section 5.3, we consider a visual stream and a neural architecture with m output dimensions (per pixel), and we aim at developing the network weights w such that the MI index is maximized as strongly as possible with respect to the model capacity. Of course, restricting the attention to a subset of the spatio-temporal coordinates of the video, due to a FOA mechanism, seems to inherently carry less information than when considering the whole video. However, in the latter case, the processed data will be characterized by a larger variability, mixing up noisy/background information with what could be more useful to understand the video, as pointed out also in recent literature [131]. Such mixture of data could be harder to disentangle by a learning model than well-selected information coming from a human-like FOA trajectory, leading to a worse MI estimate. Curiously, the learning process restricted to the FOA trajectory could end-up in facilitating the development of the weights. In so doing, even the information transfer measured on the whole frame by a model learned solely on the focus trajectory, could be larger than the one obtained by a model learned without restrictions – hence having

a uniform spatial probability density over the Retina.

Following the notation of Eqs. (5.12, 5.14), the MI maximization, for each t, is based on the spatial distribution $\mu(x,t)$. This term models the relevance of each coordinate x when learning from frame $u(t)$. In [145, 146], $\mu(x,t)$ is assumed to be uniform over the frame area, while in [132] it is also described the idea of considering μ (f in [132]) as the most natural candidate for implementing a FOA-based mechanism. Let us assume that $a(t)$ are the spatial coordinates of the FOA at time t, then we define

$$\mu(x,t) := g(x - a(t)), \tag{5.17}$$

being g a function that is peaked on $a(t)$. Following this parametrization of μ, we borrow a state-of-the art model for scanpath $a(t)$ prediction defined in [133], that shares a physics-inspired formulation as CALs. Several recent approaches deal with an attention estimation mechanism which outputs a saliency map, generally producing aggregated statistics on the most relevant areas of analyzed visual stream [155, 156], processed in offline manner. In the context of our online temporal computational model, our choice falls upon a FOA predictor [133] that is specifically designed to generate real-time scanpaths, i.e. temporal sequences of fixation points, from a state-of-the art ODE-based formulation, making it well paired with our learning scheme.

This FOA model has been proven to be strongly human-like in free-viewing conditions [157]. It is based on the intuition that the attention emerges as a gravitational process, in which both low-level (gradient, contours, motion) or high-level features (objects, context) may act as gravitational masses. In particular, given the gravitational field $E(t, a(t))$, the law that drives the attention is

$$\ddot{a}(t) + \rho\dot{a}(t) - E(t, a(t)) = 0, \tag{5.18}$$

that is indeed another 2nd order model as the one we proposed in Section 5.2 (see [133] for more details). The dissipation is controlled by $\rho > 0$, and the importance of each mass can also be tuned. Interestingly, Eq. (5.18) describes the dynamics of the FOA, and it is not based on pre-computed or given saliency maps. In this paper, following [133], we consider two basic (low-level) perceptive features as masses, the spatial gradient of the brightness and the strength of the motion field. The trajectories simulated by the model show the same

patterns of movement characteristic of human eyes: *fixations*, when the gaze remains still in a location of interest; *saccades*, rapid movement to reallocate attention on a new target; *smooth pursuit*, slow movements performed in the presence of a visual feedback with the purpose of tracking a stimulus.

Different choices on g are possible. In Section 5.6 we will consider the extreme case in which $g(x - a(t))$ is a Dirac delta on the coordinates $a(t)$ (we will refer to it as FOA), so that $\mu(x, t)$ is essentially a mono-dimensional signal. A less extreme setting is the one in which g is a squared window centered in $a(t)$ that covers a small fraction of the frame (FOAW), while the most-relaxed setting is when g is simply uniform on the whole frame (UNI), i.e., $a(t)$ is not used.

5.6 Experimental Results

In order to prove the capabilities of the proposed unsupervised learning mechanism, we evaluated the amount of information transferred from different video streams with the 2nd order laws of Section 5.2, using multiple instances of the deep convolutional network described in Section 5.3. We analyzed how the joint contribution carried by the proposed constrained temporal MI estimation presented in Section 5.4 and the spatial filtering induced by the FOA (Section 5.5) is reflected into the information transfer.

Models. We experimented with three different Convolutional Architectures, referred to as S (Small), D (Deeper), DL (Deeper and with a Larger number of neurons), and they are based on 5×5 filters (except for the last layer – 7×7 filters), $\ell = 3$ (S) or $\ell = 7$ (D, DL) layers, and either $m = 10$ (S, D) or $m = 32$ (DL) filters in layer ℓ. Networks S and D are composed of 20 filters in each hidden layer, while DL has 32 filters in each hidden layer. Following Section 5.5, we compared 3 potential terms based on 3 different input probability densities $\mu(x, t)$, named UNI, FOA, FOAW (uniform, foa-restricted, and foa-window-restricted, respectively – window edge is 15% of the min frame dimension). For each of them, we tested the 3 criteria of Section 5.4 to extend the temporal locality, PLA, VAR, AVG (fully local, variable-based, average). We integrated the differential equations using the Euler method.

Setting & Data. We considered three visual streams with $105k$ frames

each. The first $100k$ frames are the ones on which learning is performed, by integrating the CALs. Then, the developed weights $w(T)$ are used to measure the MI index over the following $5k$ frames, directly applying the MI formulation of Eq. (5.15), i.e., $I(X, Y; w(T); 100000, 105000)$, that is what we report in the results of this section. For all the models, independently on the probability density used in their potentials, we measured the MI index using $\mu(x, t)$ in the UNI, FOA, FOAW cases. This means that, for example, a model

Figure 5.3: Sample frames taken from the SparseMNIST, Carpark, Call streams, left-to-right.

trained following the FOA trajectory is then evaluated in the 5k test frames either considering the whole frame area, the FOA trajectory, or the window-based FOA trajectory. The three streams (Fig. 5.3), have different properties. The first one, SparseMNIST, is composed of a static frame (280×280) in which 10 digits from the MNIST data are sparsely located over a dark background. The second video, Carpark, is taken from a fixed camera monitoring a car parking area in front of a building. The last video, Call, is a recording taken from a webcam during a video call. Videos are repeated until the target number of frames is reached. The last two videos are processed at 240×180 pixels per frame, grayscale, ≈ 30 frames per second. We selected videos that naturally represent repetitive contents, so that they can be repeated without introducing evident scene changes. Contents and events are heterogeneous among the videos.

Parameters. The FOA trajectory was generated by weighing the two gravitational masses 0.1 (frame details) and 1.0 (motion), respectively, and adjusting $\rho \in [0.1, 0.5]$ in order to adapt it to the each video. We analyze the behaviour of the FOA trajectories in Fig. 5.4. In particular for each stream we show a representative frame, a heatmap obtained from the FOA scanpath as well as a scatter plot of the FOA fixations and velocity.

After a first experimentation in which we qualitatively observed the behaviour of the 2nd order laws, we set $\alpha = 0.01$, $\beta = 0.1$, $k = 10^{-8}$. For each model we considered multiple weighing schemes of the parameters $\lambda_c \in \{10, 100, 200, 1000\}$, $\lambda_e \in \{20, 200, 400, 2000, 4000\}$, $\lambda_s \in \{10, 100, 1000\}$, $\zeta_s \in \{0.01, 0.05, 0.07\}$, selecting the ones that returned the largest MI during the learning stage. As a general rule of thumb, using a lower value of the conditional entropy weighing term λ_c w.r.t. the entropy weight λ_e, helps the model to exploit all the available output symbols. The network weights $w(0)$ were randomly initialized, enforcing the same initialization to all the compared model.

Spatial filtering. The analysis on the contributions of the various Spatial filtering approaches (UNI,FOA, FOW) are reported in Tab. 5.1. Each column, starting from the third one, is about a model, defined by the pair (*architecture*, *density used in the training potential*). For each model, the MI index is reported when measured using different spatio-temporal densities (they are labeled in column "Test"). We used the temporal locality criterion that led to the best results. Overall, the models trained on FOA-based densities (columns FOA, FOAW) usually perform better than the ones that were

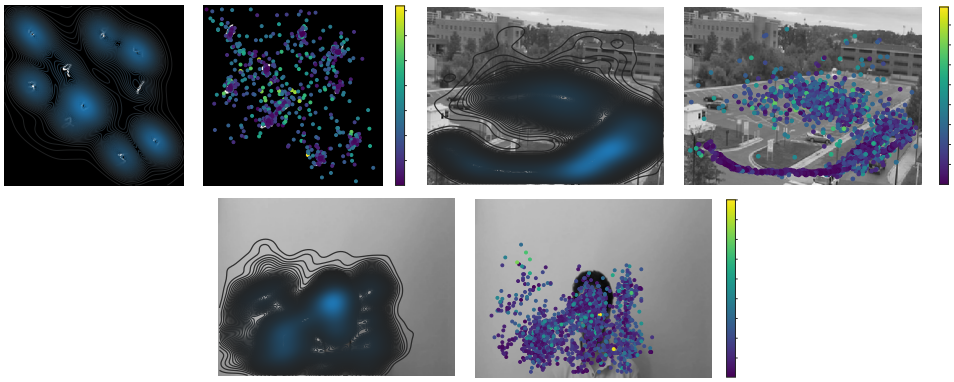

Figure 5.4: For each stream, we show (*left*) the areas mostly covered by FOA (blue: largest attention), and (*right*) the scatterplots of the fixation points, with hue denoting the magnitude of the FOA velocity (blue: slower; yellow: faster). Low-speed movements happen on the most informative areas (e.g., digits, busy roads, human presence/movement, respectively).

Table 5.1: Spatial filtering. Mutual Information (MI) index in three video streams, considering three neural architectures (S, D, DL). Each column (starting from the third one) is about the results of network trained using an input probability density taken from {UNI, FOA, FOAW}, and tested measuring the MI index in all the three density cases (labeled in column "Test").

		S			D			DL		
Stream	Test	UNI	FOA	FOAW	UNI	FOA	FOAW	UNI	FOA	FOAW
SparseMNIST	UNI	0.017	**0.112**	0.078	0.004	**0.144**	0.020	0.012	**0.132**	0.026
	FOA	0.239	**0.486**	0.391	0.103	**0.431**	0.229	0.146	**0.350**	0.194
	FOAW	0.154	**0.209**	0.197	0.144	**0.255**	0.157	0.117	**0.215**	0.131
Carpark	UNI	**0.776**	0.601	0.695	0.653	0.556	**0.745**	0.445	0.292	**0.496**
	FOA	**0.742**	0.675	0.694	0.678	0.639	**0.768**	0.477	0.315	**0.529**
	FOAW	**0.719**	0.629	0.671	0.653	0.601	**0.721**	0.501	0.357	**0.532**
Call	UNI	**0.329**	0.314	0.315	0.339	**0.556**	0.350	0.208	**0.304**	0.218
	FOA	**0.405**	**0.405**	0.371	0.430	**0.582**	0.492	0.246	**0.365**	0.270
	FOAW	**0.429**	0.420	0.413	0.442	**0.566**	0.457	0.304	**0.374**	0.310

exposed to a uniform $\mu(x,t)$ over the frame area (columns UNI). This is particularly noticeable in the SparseMNIST and Call streams, characterized by a still and not-much-detailed background and few regions of interest, i.e. the digits or the moving speaker, respectively. The filtering approach induced by the attention in the training stage highly improves the information transfer over most of the considered test measurements, with just a few exceptions. These considerations hold at a lesser degree also in the Carpark stream, in which frames are more detailed. The focus is attracted by a busy road or by people parking their cars. However, also the immediate surroundings of those regions contain much information, so that training with FOAW density achieves the best results in architectures D and DL, while the more extreme FOA approach do not compete with models trained considering the whole frame (UNI). In both the Carpark and Call streams, the S architecture does not benefit from learning over the attention trajectory. We motivate this result by considering that S is a shallower model, that inherently learns lower level features than the other ones. These features are more common to different frame locations, making the impact of attention less evident. In the case of SparseMNIST, the dark-uniform background dominates the frame, and learning over $a(t)$ induces a largest information transfer also in the network S.

Temporal locality. In order to evaluate the impact of the temporal locality criteria (PLA, AVG, VAR), we restrict our analysis to models trained with a FOA-restricted probability density. In this case, we describe each model by the pair (*architecture*, *temporal locality criterion*), and we report results in Tab. 5.2. In general, the moving average criterion (AVG) achieves the best performances in all settings, with some exceptions. The CARPARK stream has temporal dynamics that are pretty repetitive and periodic (e.g., cars crossing the same crossroad etc.). Hence, the addition of a criterion able to keep better track of the temporal information turns out to be less necessary. We notice higher values of the MI index in the fully temporally local case (PLA) in the architecture DL. This may be due to the fact that DL has a larger number of parameters and units than the other nets, and it has intrinsically more capacity to memorize the temporal information. The MI index is lower than for the other architectures due to the largest size of the output space. Despite the fact that the AVG criterion shares the same intuition with the constrained one (VAR), the latter one performs worse. We believe that this is mostly due to the fact that enriching the already complex Lagrangian with the differential soft-constraint, together with the introduction of a novel auxiliary variable, potentially hinders the optimization process as a whole. However, the very same intuition of a "local" variable that keeps an internal state of the previous entropy predictions proved to be effective thanks to the AVG criterion.

Random scanpaths. We are left with the open question on whether the largest information transfer we experienced when learning on the spatio-temporal focus trajectory is due to the state-of-the art attention model we are using or it is only due to the reduction of the size of the input data. We compared models trained on the FOA trajectories used so far with the same networks trained with a fully-local spatial density defined over a random scanpath – obtained by randomly sampling $a(t)$ from a uniform distribution over the retina. Afterwards, both the learned models are tested exploiting the FOA measure on a regular (human-like) scanpath.

The results of Fig. 5.5 show that the human-like trajectory estimated by the selected attention model has a clear positive impact in the information transfer. Interestingly, in the CARPARK case we sometimes observe that fixations, that explore random coordinates, highly foster information transfer.

Table 5.2: Temporal locality. Mutual Information (MI) index in three video streams, considering three neural architectures (S, D, DL). Each column (starting from the third one) is about the results of the network trained using the FOA trajectory with a temporal locality criterion taken from {PLA, AVG, VAR}, and tested measuring the MI index in all the three density cases (labeled in column "Test").

		S			D			DL		
Stream	Test	PLA	AVG	VAR	PLA	AVG	VAR	PLA	AVG	VAR
SparseMNIST	UNI	0.071	**0.112**	0.102	0.006	0.054	**0.144**	0.028	**0.132**	0.080
	FOA	0.425	**0.486**	0.298	0.149	0.321	**0.431**	0.119	**0.350**	0.184
	FOAW	0.183	**0.209**	0.208	0.146	0.184	**0.255**	0.127	**0.215**	0.176
Carpark	UNI	**0.601**	0.486	0.371	0.422	**0.556**	0.315	**0.292**	0.289	0.204
	FOA	**0.675**	0.521	0.401	0.458	**0.639**	0.326	**0.315**	0.307	0.209
	FOAW	**0.629**	0.548	0.447	0.489	**0.601**	0.389	**0.357**	**0.357**	0.277
Call	UNI	0.289	**0.314**	0.267	0.259	**0.556**	0.369	**0.304**	0.189	0.200
	FOA	0.326	**0.405**	0.265	0.328	**0.582**	0.459	**0.365**	0.214	0.260
	FOAW	0.383	**0.420**	0.373	0.368	**0.566**	0.443	**0.374**	0.274	0.275

This confirms our previous statements regarding the large amount of information in the whole frame area.

Learning dynamics. We investigate the behaviour of the models during the training stage, restricting our analysis in the case of architecture D and a single training/test probability density, FOA. The plots of Fig. 5.6, for each value t of the x-axis, show the MI index computed in the interval $[0, t]$ along the FOA trajectory, for different temporal criteria (PLA, AVG, VAR). The constraint-based (VAR) model tends to quickly find a stationary condition of the estimated MI index value. Both PLA and AVG incur in an initial stage with evident fluctuations before becoming more stable, usually in larger values than VAR. The highly different learning dynamics confirms our claims regarding the high influence on the optimization process caused by the introduction of the soft-constraint into the Lagrangian. The spikes visible in the first stages of learning are due to the fact that learning happens along the FOA trajectory. In this case, the spatio-temporal distribution $\mu(x, t)$, conditioning the output symbols, is defined over a single pixel. This means that the models have to deal with pretty varied conditions, which is

limited to a single focus location in each frame, struggling to find a stationary condition. As long as time passes and a largest portion of stream is processed, fluctuations are mitigated reaching more stable configurations. Remarkably, the VAR constraint-based (VAR) model does not suffer from this issue and reaches faster a stable configuration.

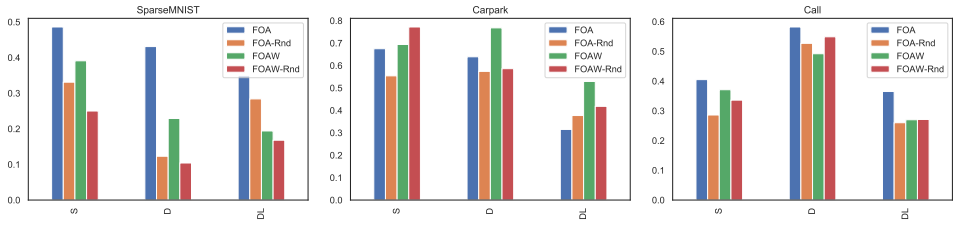

Figure 5.5: Comparison between models trained on a regular trajectory of the attention and on a random trajectory (suffix -RND), for architectures S, D, DL. Each bar is about a different training probability density, and the height of the bar is the test MI index along the regular FOA trajectory.

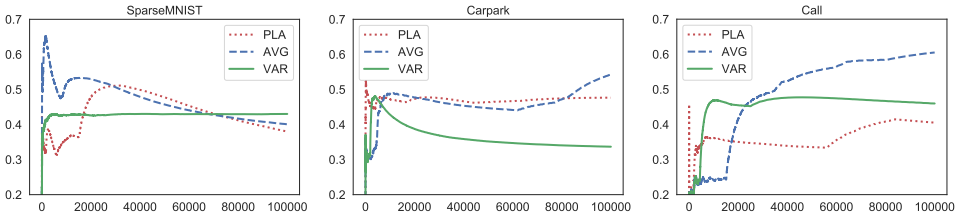

Figure 5.6: Learning dynamics (model D-FOA, different temporal criteria). The MI index is shown at different time instants. The index at time t is evaluated along the FOA trajectory in the interval $[0, t]$.

5.7 Discussion and Future Work

In this Chapter we delved into a novel approach to Mutual Information (MI) maximization rising from the conjunction of online entropy estimation mechanisms, obtained through a temporal computational model which leverages a constraint-based approach and a human-like focus of attention. We introduced

a 2nd order differential model capable to fulfill the requirements of temporal locality and causality. These two characteristics come in handy when decoupling the temporal computational model into local subcomponents. Doing so, we are able to devise an online temporal estimation of the output entropy exploiting the mathematical notion of contraints (VAR), or a similar intuition that averages the output predictions over time (AVG).

We provide insightful experimental results to support the intuition that using the spatio-temporal density of the focus of attention to drive the learning dynamics fosters an increment of the globally transferred information from the input stream. Models trained exploiting the proposed attention trajectory and temporal estimation are also able to increase the information transfer not only over the focused areas, but, in some cases, over the whole frame area. Future work will be devoted to enforcing coherence over the predictions performed on the focus trajectory to develop high-level representations.

Chapter 6

Conclusions and Future Works

Throughout this dissertation we investigated a novel path into the direction of learning in neural networks. We devise a framework capable of decomposing neural architectures into *local components*, i.e. subparts constituting the overall computational model. The introduction of auxiliary variables and the unifying mathematical notion of constraint are leveraged to force internal knowledge onto the structure of the neural models, enforcing the communication among local components. Indeed, in this way it is possible to partially describe the computation performed by the network using structural constraints.

This heterogeneous optimization scheme is based on "locality" principles, that allow us to express a local learning procedure in several DAG architectures (Chapter 3), to avoid costly iterative procedures (Chapter 4) or to better estimate given temporal quantities (Chapter 5) without the need to bufferize data.

In the following, a brief summary of the key contributions of the Thesis will precede some intuitions on relevant directions for further work.

6.1 Summary of Contributions

- Chapter 3 describes the first instance of the aforementioned intuition. The computational graph describing the flow of information inside a generic DAG is decomposed into local components. In this way, it is possible to express the computational structure by the so-called architectural constraints, giving rise to the Local Propagation algorithm [37]. It has been shown that the Lagrangian formulation in the adjoint space leads to a fully local algorithm, LP, that naturally emerges when searching for saddle points. The optimization process can be interpreted as

a force that gradually enforces the constraint satisfaction, fostering the flow of information and leveraging updates supported by local variables. Finally, the experimental analysis carried on several benchmarks confirmed the feasibility of the proposed approach.

- In Chapter 4 we exploited a similar idea in order to avoid the unfolding of the convergence procedure in the original model of GNNs. The learning task is formulated as a constrained optimization problem devised following the principles of locality, i.e. the computational graph decomposition into local subparts connected via constraints. The problem definition allows us to avoid the explicit computation of the fixed point of the state transition function, that is needed to encode the graph, whilst the constraints allow the model to modulate the effects of the diffusion process. We proposed a mixed strategy that jointly optimizes the model weights and the state representations without the need of separate optimization stages, referred to as Lagrangian Propagation GNNs [39]. Moreover, the constrained representation of the learning procedure can be replicated multiple times, simulating a layered approach, introducing an increased representational power [38].

- Chapter 5 outlined a novel promising approach to maximize the information transfer from a continuous visual stream to the output space of a Convolutional Neural Network. The introduction of Second Order Cognitive Action Laws defines a learning process on a temporal computational model that is causal and local in time. These two characteristics are leveraged to decompose the temporal estimations into local contributions, put into communication by softly-enforced constraints, or similar local averaging ideas, that allow us to produce an online estimation of the entropy terms. Moreover, the input spatio-temporal probability distribution is modelled by a human-like focus of attention (FOA) trajectory, obtained by a SOTA predictor. The FOA scanpath drives the learning dynamics, filtering the visual stream and fostering an increment of the globally transferred information from the input stream [40].

6.2 Issues and future research directions

6.2.1 Local Propagation

Parallel Computations Chapter 3 proved the feasibility of the LP algorithm and its optimization scheme. In order to test our claims, we exploited a commonly used Machine Learning framework (i.e., Tensorflow) leveraging its ready-to-use libraries and resources. This kind of frameworks have been developed bearing in mind the common pipeline of the sequential computational graphs characterizing MLPs and feedforward networks, as long as learning algorithms such as SGD. However, as highlighted in [62], stochastic gradient methods are characterized by a limited scalability, given the fact that they focus on many lightweight minimization steps computed on small amount of data (i.e., mini-batches). Whilst in a serial setting such computational structure represents a big advantage (GPU-based computation, with shared low memory spaces accessible by many cores), in a parallel multi-processor setting each one of such computational step is too inexpensive to be split over multiple cores. Moreover, every mini-batch based computation in BP involves the forward propagation of the data-at-hand along the whole network architecture in a sequential manner, followed by the sequential backward computations of the gradients. Modern hardware (GPUs) can benefit by the parallelization of the matrix operations within each layer, but parallelizing the computations over layers is hard.

Thanks to the decomposition of the learning problem into local components, LP is capable to go beyond the aforementioned limitations. Indeed, in order to make full use of LP, an highly parallel implementation of the proposed algorithm should be exploited. The overall computation performed by LP could be easily scaled up thanks to its local nature, exploiting data parallelization strategies (e.g. many worker nodes). Each computational unit needs to store the variables (auxiliary variable for neural units, $x_{\ell,i}$, and the related Lagrange Multipliers, $\lambda_{\ell,i}$) belonging to a particular subset of layers/neurons and corresponding to a subset of the data, alongside the variables and quantities needed by the local optimization step. In particular, the ℓ-th computational unit needs to share the memory where some variables are stored with the $(\ell+1)$-th and $(\ell-1)$-th units. Following this scheme, the computations

could be parallelized both on the example dimension i and on the layers ℓ dimension. The constraint satisfaction values could be computed at node-level and than transmitted to a central node in order to be summed up with a reduction operation into the global loss function. Afterwards, the central node could broadcast the information needed for the parameter update. This step could represent, however, a bottleneck for the depicted distributed scenario.

Such highly distributed solution could, in principle, scale linearly with the number of working nodes, as proved in [62], which investigates the ability of a similar local constraint-based approach to scale with the number of available computing cores. Such an approach would require an ad-hoc code implementation (which goes beyond the commonly available ML frameworks) as long as a particular focus on the optimization of the communication protocol among nodes (e.g., reduction of exchanged information).

In particular, we can think of pursuing the intuitions behind the research field of Federated Learning [158] and its guiding principles on decentralization and distributed optimization mechanisms to overcome the aforementioned communications problems. Recent works [159] are capable to train deep models leveraging millions of devices with a reduced network communication, using compressed updates with efficiency, security and scalability in the aggregation of the model updates. For all the aforementioned reasons, the application of such solution on a potentially parallelizable algorithm such as LP could bring important advantages.

Memory scalability issues The local nature of the LP algorithm is achieved thanks to the introduction of additional variables to the learning problem. In particular, for each input pattern i the procedure introduces an auxiliary variable for each neural unit of each layer ℓ ($x_{\ell,i}$) and the related Lagrange Multipliers ($\lambda_{\ell,i}$). This fact could hinder its applicability to tasks characterized by huge datasets, especially if the training procedure exploits a full-batch mechanism, i.e. processing all the data at once. While we did not experience any memory complexity issues in the experimental campaign, we believe that scaling up the size of the datasets-at-hand could preclude the usage of the algorithm in the current available Tensorflow implementation. In order to solve this problem, several solutions can be proposed, besides a simple mini-batch SGD-like approach. Certainly, as mentioned in the previous

paragraph, the proposed locality of the computations and their parallelization could represent an immediate solution and highly alleviate this problem.

Moreover, another interesting research path is the reduction of the needed auxiliary variables. The Lagrange multiplier variables (λ_ℓ) could be reduced defining constraints on whole layers instead of single neurons, or decoupling them from the pattern dependence.

Predicting multipliers & optimization The most promising direction of research is the possibility to completely remove the need to collect such variables. An ANN can be used to *predict* the Lagrange multipliers, given the local context of each neuron (connected units, its activation, the previous value of the neuron variable). The optimization process follows the same rules, but in this case instead of the Lagrange Multipliers Λ, the gradient ascent step is carried on considering the derivative of the Lagrangian with respect to the learnable parameters of the ANN implementing the multipliers predictor.

This novel intuition could be analyzed also from the optimization point of view, substituting the BDMM algorithm with alternatives or evolutions [80].

Architecture Search The learning process may be enriched by the enforcement of a L_1-norm regularizer (weighted by $\alpha > 0$) on each $x_{\ell,i}$. In such a way, the model could in principle focus on a smaller number of paths from input to output units, reducing the search space.

This interesting approach could lead to findings in the direction of architecture search. The variables x_ℓ associated to the neural units, that are not contributing to the information flow (i.e. their value is close to zero due to the regularization), could be in fact pruned from the architecture, together with the weight connecting them with the neighboring layers.

6.2.2 Lagrangian Propagation GNNs

The optimization process carried on in LP-GNNs is the same as the one exploited in LP, even if the computational and memory burden is solely due to the node state computation (i.e. not depending on each neural unit of the state transition and output functions, which are learned via BP). However, the considerations on parallelization could bring benefits also in the case of this approach.

Predicting multipliers In the context of GNNs, the idea to use an ANN

to predict the value of the Lagrange multipliers conditioned on the neighboring nodes states and/or their features, highly resembles the isotropic models (see Section 4.1) like GAT [95]. Indeed, learning the Lagrange multiplier in this case can be seen as weighting the contribution of a particular node (the corresponding constraint) as done in attention models.

Other future works Due to the originality of what we described in Chapter 4, several aspects of our model can be the subject of further investigations. We plan to extend the experimental evaluation to verify the algorithm behaviour with respect to either the characteristics of the input graphs, such as the graph diameter, the variability in the node degrees, the type of node and arc features or to the model architecture (f.i. type of the state transition function, of the constraint function, etc.). Furthermore, the proposed constraint-based scheme can be extended to all the other methods proposed in the literature that exploit more sophisticated architectures.

6.2.3 Constraining Predictions over time

In Chapter 5 we gave evidence of the fact that the conjunction of a spatio-temporal input density distribution and an online local entropy estimation mechanism foster the information transfer in visual streams. Future works will be devoted to leverage the approach into downstream tasks. Our findings could, in principle, help in developing more informed features. The proposed method is indeed an unsupervised learning approach that can be considered as a pre-training step for supervised tasks.

FOA coherence Since the model for the focus of attention [133] mostly tracks moving objects, we are currently working on the injection of *motion-coherence* into the potential term [132]. The injection of motion invariance can play a crucial role in reducing the needed supervisions to learn features, since the model dynamics and the FOA trajectory are able to propagate few supervisions along the temporal trajectory of the object.

A Lifelong evolving Agent The contribution presented in Chapter 5 is a step towards the creation of a unifying theory of vision. The final goal is the design of a visual agent that is able to experience a lifelong visual stream and progressively learn to recognize objects, disentangle their components and develop human-like visual cognitive capabilities.

This agent could highly benefit from the ability to interact and explore a visual environment. Unfortunately, it is not so easy to collect and store real-world video streams (e.g. from Video Surveillance cameras [160]) as long as to supervise each video frame for the task at-hand.

Bearing these issues in mind, we are working on the injection of the proposed method into a simulated Virtual environment [121] that we developed, capable to completely overcome the need of handmade supervisions. We believe that this favourable environment can prove the advantages of our lifelong learning approach, and that a visual agent empowered with our learning mechanism can be capable to learn very useful and informed features by navigating and moving its gaze trough the simulated scenes.

Bibliography

[1] J. C. Platt and A. H. Barr, "Constrained differential optimization," in *Neural Information Processing Systems*, 1988, pp. 612–621.

[2] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *CoRR*, vol. abs/1810.00826, 2018. [Online]. Available: http://arxiv.org/abs/1810.00826

[3] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1, no. 2.

[4] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 621–11 631.

[5] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean, "A guide to deep learning in healthcare," *Nature medicine*, vol. 25, no. 1, pp. 24–29, 2019.

[6] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackerman *et al.*, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, 2020.

[7] A. Ram, R. Prasad, C. Khatri, A. Venkatesh, R. Gabriel, Q. Liu, J. Nunn, B. Hedayatnia, M. Cheng, A. Nagar *et al.*, "Conversational ai: The science behind the alexa prize," *arXiv preprint arXiv:1801.03604*, 2018.

[8] J. Shlomi, P. Battaglia *et al.*, "Graph neural networks in particle physics," *Machine Learning: Science and Technology*, 2020.

[9] J. Brehmer, F. Kling, I. Espejo, and K. Cranmer, "Madminer: Machine learning-based inference for particle physics," *Computing and Software for Big Science*, vol. 4, no. 1, pp. 1–25, 2020.

[10] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: a modular machine learning software library," Idiap, Tech. Rep., 2002.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 675–678.

[12] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[13] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[14] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[16] B. Widrow and M. E. Hoff, "Adaptive switching circuits," Stanford Univ Ca Stanford Electronics Labs, Tech. Rep., 1960.

[17] F. Rosenblatt, "Principles of neurodynamics: Perceptions and the theory of brain mechanisms," 1962.

[18] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization*. Springer, 1982, pp. 762–770.

[19] N. Wiener, *Cybernetics or control and communication in the animal and the machine*. Technology Press, 1948.

[20] W. R. Ashby, *An introduction to cybernetics*. Chapman & Hall Ltd, 1961.

[21] J. Marino, "Predictive coding, variational autoencoders, and biological connections," 2019.

[22] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[24] D. E. Rumelhart, G. E. Hinton, R. J. Williams *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[25] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[26] Y. Lecun, "Une procedure dśapprentissage pour reseau a seuil assymetrique cog'nitiva 85: A la frontiere de lqintelligence artificielle des sciences de la connais' sance des neurosciences," *Paris, France*, 1985.

[27] G. Gnecco, M. Gori, S. Melacci, and M. Sanguineti, "Foundations of support constraint machines," *Neural computation*, vol. 27, no. 2, pp. 388–480, 2015.

[28] M. Gori, *Machine Learning: A Constraint-based Approach*. Morgan Kaufmann, 2017.

[29] M. Gori, "Semantic-based regularization and piaget's cognitive stages." *Neural Networks*, vol. 22, no. 7, pp. 1035–1036, 2009.

[30] M. Diligenti, M. Gori, M. Maggini, and L. Rigutini, "Bridging logic and kernel machines," *Machine learning*, vol. 86, no. 1, pp. 57–88, 2012.

[31] G. Marra, F. Giannini, M. Diligenti, and M. Gori, "Lyrics: a general interface layer to integrate logic inference and deep learning," in *ECML*, 2019.

[32] S. Melacci and M. Gori, "Semi-supervised multiclass kernel machines with probabilistic constraints," in *Congress of the Italian Association for Artificial Intelligence*. Springer, 2011, pp. 21–32.

[33] S. Melacci, M. Maggini, and M. Gori, "Semi–supervised learning with constraints for multi–view object recognition," in *International Conference on Artificial Neural Networks*. Springer, 2009, pp. 653–662.

[34] L. Graziani, S. Melacci, and M. Gori, "Coherence constraints in facial expression recognition," *Intelligenza Artificiale*, vol. 13, no. 1, pp. 79–92, 2019.

[35] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[36] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[37] G. Marra, M. Tiezzi, S. Melacci, A. Betti, M. Maggini, and M. Gori, "Local propagation in constraint-based neural network," *arXiv preprint arXiv:2002.07720*, 2020.

[38] M. Tiezzi, G. Marra, S. Melacci, and M. Maggini, "Deep lagrangian constraint-based propagation in graph neural networks," *arXiv preprint arXiv:2005.02392*, 2020.

[39] M. Tiezzi, G. Marra, S. Melacci, M. Maggini, and M. Gori, "A lagrangian approach to information propagation in graph neural networks," *ArXiv*, vol. abs/2002.07684, 2020.

[40] M. Tiezzi, S. Melacci, A. Betti, M. Maggini, and M. Gori, "Focus of attention improves information transfer in visual features," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[41] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[42] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[43] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[44] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

[46] Y. LeCun and Y. Bengio, "The handbook of brain theory and neural networks," in *The Handbook of Brain Theory and Neural Networks*, M. A. Arbib, Ed. Cambridge, MA, USA: MIT Press, 1998, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. [Online]. Available: http://dl.acm.org/citation.cfm?id=303568.303704

[47] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.

[48] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, June 1996, pp. 347–352 vol.1.

[49] P. Frasconi, M. Gori, and A. Sperduti, "A general framework for adaptive processing of data structures," *Trans. Neur. Netw.*, vol. 9, no. 5, pp. 768–786, Sep. 1998. [Online]. Available: https://doi.org/10.1109/72.712151

[50] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[51] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Message passing neural networks," in *Machine Learning Meets Quantum Physics*. Springer, 2020, pp. 199–214.

[52] Q. Li, Z. Han, and X.-M. Wu, "Deeper insights into graph convolutional networks for semi-supervised learning," *arXiv preprint arXiv:1801.07606*, 2018.

[53] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605

[54] P. Agarwal, M. Jleli, and B. Samet, "Banach contraction principle and applications," in *Fixed Point Theory in Metric Spaces*. Springer, 2018, pp. 1–23.

[55] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[56] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski, "A theoretical framework for back-propagation," in *Proceedings of the 1988 connectionist models summer school*, vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988, pp. 21–28.

[57] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[58] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola, "Stacked attention networks for image question answering," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[59] H. Xu and K. Saenko, "Ask, attend and answer: Exploring question-guided spatial attention for visual question answering," in *European Conference on Computer Vision*. Springer, 2016, pp. 451–466.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[61] M. Carreira-Perpinan and W. Wang, "Distributed optimization of deeply nested systems," in *Artificial Intelligence and Statistics*, 2014, pp. 10–19.

[62] G. Taylor, R. Burmeister, Z. Xu, B. Singh, A. Patel, and T. Goldstein, "Training neural networks without gradients: A scalable admm approach," in *International conference on machine learning*, 2016, pp. 2722–2731.

[63] A. Gotmare, V. Thomas, J. Brea, and M. Jaggi, "Decoupling backpropagation using constrained optimization methods," in *Workshop on Efficient Credit Assignment in Deep Learning and Deep Reinforement Learning, ICML 2018.*, 2018, pp. 1–11.

[64] F. Gu, A. Askari, and L. El Ghaoui, "Fenchel lifted networks: A lagrange relaxation of neural network training," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 3362–3371.

[65] J. Launay, I. Poli, F. Boniface, and F. Krzakala, "Direct feedback alignment scales to modern deep learning tasks and architectures," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[66] B. Millidge, A. Tschantz, and C. L. Buckley, "Predictive coding approximates backprop along arbitrary computation graphs," *arXiv preprint arXiv:2006.04182*, 2020.

[67] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive science*, vol. 11, no. 1, pp. 23–63, 1987.

[68] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, no. 1, pp. 1–10, 2016.

[69] M. Akrout, C. Wilson, P. Humphreys, T. Lillicrap, and D. B. Tweed, "Deep learning without weight transport," in *Advances in neural information processing systems*, 2019, pp. 976–984.

[70] D.-H. Lee, S. Zhang, A. Fischer, and Y. Bengio, "Difference target propagation," in *Joint european conference on machine learning and knowledge discovery in databases*. Springer, 2015, pp. 498–515.

[71] B. Scellier and Y. Bengio, "Equilibrium propagation: Bridging the gap between energy-based models and backpropagation," *Frontiers in computational neuroscience*, vol. 11, p. 24, 2017.

[72] J. C. Whittington and R. Bogacz, "An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity," *Neural computation*, vol. 29, no. 5, pp. 1229–1262, 2017.

[73] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach, and J. Martens, "Adding gradient noise improves learning for very deep networks," *arXiv preprint arXiv:1511.06807*, 2015.

[74] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[75] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[76] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[77] D. Dua and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[78] M. Olson, A. Wyner, and R. Berk, "Modern neural networks generalize on small data sets," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds. Curran Associates, Inc., 2018, pp. 3619–3628. [Online]. Available: http://papers.nips.cc/paper/7620-modern-neural-networks-generalize-on-small-data-sets.pdf

[79] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?" *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3133–3181, 2014.

[80] A. Stooke, J. Achiam, and P. Abbeel, "Responsive safety in reinforcement learning by pid lagrangian methods," in *International Conference on Machine Learning*. PMLR, 2020, pp. 9133–9143.

[81] P. Veličković, "The resurgence of structure in deep neural networks," Ph.D. dissertation, University of Cambridge, 2019.

[82] A. Loukas, "What graph neural networks cannot learn: depth vs width," in *8th International Conference on Learning Representations, ICLR 2020,*

Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020. [Online]. Available: https://openreview.net/forum?id=B1l2bp4YwS

[83] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, 2017. [Online]. Available: https://doi.org/10.1109/MSP.2017.2693418

[84] A. Sperduti and A. Starita, "Supervised neural networks for the classification of structures," *Trans. Neur. Netw.*, vol. 8, no. 3, pp. 714–735, May 1997. [Online]. Available: http://dx.doi.org/10.1109/72.572108

[85] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05493

[86] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6203

[87] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *CoRR*, vol. abs/1506.05163, 2015. [Online]. Available: http://arxiv.org/abs/1506.05163

[88] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 3837–3845. [Online]. Available: http://papers.nips.cc/paper/6081-convolutional-neural-networks-on-graphs-with-fast-localized-spectral-filtering

[89] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=SJU4ayYgl

[90] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY,*

USA, June 19-24, 2016, 2016, pp. 2014–2023. [Online]. Available: http://jmlr.org/proceedings/papers/v48/niepert16.html

[91] D. K. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, 2015, pp. 2224–2232. [Online]. Available: http://papers.nips.cc/paper/5954-convolutional-networks-on-graphs-for-learning-molecular-fingerprints

[92] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 1993–2001. [Online]. Available: http://papers.nips.cc/paper/6212-diffusion-convolutional-neural-networks

[93] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[94] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 2018, pp. 4438–4445. [Online]. Available: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17146

[95] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[96] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5115–5124.

[97] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[98] B. Weisfeiler and A. A. Lehman, "A reduction of a graph to a canonical form and an algebra arising during this reduction," *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16, 1968.

[99] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4602–4609.

[100] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," *arXiv preprint arXiv:1812.09902*, 2018.

[101] G. Corso, L. Cavalleri, D. Beaini, P. Liò, and P. Veličković, "Principal neighbourhood aggregation for graph nets," *arXiv preprint arXiv:2004.05718*, 2020.

[102] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, "Pitfalls of graph neural network evaluation," *arXiv preprint arXiv:1811.05868*, 2018.

[103] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv preprint arXiv:2003.00982*, 2020.

[104] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," *arXiv preprint arXiv:1912.09893*, 2019.

[105] H. Wang and J. Leskovec, "Unifying graph convolutional neural networks and label propagation," *arXiv preprint arXiv:2002.06755*, 2020.

[106] A. Rossi, M. Tiezzi, G. M. Dimitri, M. Bianchini, M. Maggini, and F. Scarselli, "Inductive–transductive learning with graph neural networks," in *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*. Springer, 2018, pp. 201–212.

[107] A. Bojchevski, J. Klicpera, B. Perozzi, A. Kapoor, M. Blais, B. Rózemberczki, M. Lukasik, and S. Günnemann, "Scaling graph neural networks with approximate pagerank," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2464–2473.

[108] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti, "Sign: Scalable inception graph neural networks," *arXiv preprint arXiv:2004.11198*, 2020.

[109] H. Dai, Z. Kozareva, B. Dai, A. Smola, and L. Song, "Learning steady-states of iterative algorithms over graphs," in *International Conference on Machine Learning*, 2018, pp. 1114–1122.

[110] M. Bianchini, G. M. Dimitri, M. Maggini, and F. Scarselli, *Deep Neural Networks for Structured Data*. Cham: Springer International Publishing, 2018, pp. 29–51. [Online]. Available: https://doi.org/10.1007/978-3-319-89629-8_2

[111] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[112] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[113] P. Yanardag and S. Vishwanathan, "Deep graph kernels," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1365–1374.

[114] S. Ivanov and E. Burnaev, "Anonymous walk embeddings," *arXiv preprint arXiv:1805.11921*, 2018.

[115] D. Pathak, R. Girshick, P. Dollár, T. Darrell, and B. Hariharan, "Learning features by watching objects move," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2701–2710.

[116] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[117] J. Pont-Tuset, F. Perazzi, S. Caelles, P. Arbeláez, A. Sorkine-Hornung, and L. Van Gool, "The 2017 davis challenge on video object segmentation," *arXiv preprint arXiv:1704.00675*, 2017.

[118] D.-P. Fan, W. Wang, M.-M. Cheng, and J. Shen, "Shifting more attention to video salient object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019, pp. 8554–8564.

[119] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi, "Ai2-thor: An interactive 3d environment for visual ai," *arXiv preprint arXiv:1712.05474*, 2017.

[120] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik *et al.*, "Habitat: A platform for embodied ai research," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 9339–9347.

[121] E. Meloni, L. Pasqualini, M. Tiezzi, M. Gori, and S. Melacci, "Sailenv: Learning in virtual visual environments made simple," *arXiv preprint arXiv:2007.08224*, 2020.

[122] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, vol. 113, pp. 54–71, 2019.

[123] D. Sahoo, Q. Pham, J. Lu, and S. C. Hoi, "Online deep learning: Learning deep neural networks on the fly," *arXiv preprint arXiv:1711.03705*, 2017.

[124] A. J. Bell and T. J. Sejnowski, "An information-maximization approach to blind separation and blind deconvolution," *Neural computation*, vol. 7, no. 6, pp. 1129–1159, 1995.

[125] T. Kohonen, *Self-organizing maps*. Springer Science & Business Media, 2012, vol. 30.

[126] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, 2016, pp. 478–487.

[127] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.

[128] A. Kolesnikov, X. Zhai, and L. Beyer, "Revisiting self-supervised visual representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2019, pp. 1920–1929.

[129] R. Linsker, "Self-organization in a perceptual network," *Computer*, vol. 21, no. 3, pp. 105–117, 1988.

[130] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, A. Courville, and D. Hjelm, "Mutual information neural estimation," in *International Conference on Machine Learning*, 2018, pp. 531–540.

[131] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," *ICLR arXiv preprint arXiv:1808.06670*, 2019.

[132] A. Betti, M. Gori, and S. Melacci, "Learning visual features under motion invariance," *Neural Networks*, vol. 126, pp. 275 – 299, 2020.

[133] D. Zanca, S. Melacci, and M. Gori, "Gravitational laws of focus of attention," *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[134] S. Grossberg, "Associative and competitive principles of learning and development," in *Competition and Cooperation in Neural Nets*. Springer, 1982, pp. 295–341.

[135] M. K. Benna and S. Fusi, "Computational principles of synaptic memory consolidation," *Nature neuroscience*, vol. 19, no. 12, pp. 1697–1706, 2016.

[136] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural computation*, vol. 14, no. 4, pp. 715–770, 2002.

[137] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," *arXiv preprint arXiv:1906.05849*, 2019.

[138] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[139] M. Tschannen, J. Djolonga, P. K. Rubenstein, S. Gelly, and M. Lucic, "On mutual information maximization for representation learning," *ICRL arXiv preprint arXiv:1907.13625*, 2020.

[140] W. Hu, T. Miyato, S. Tokui, E. Matsumoto, and M. Sugiyama, "Learning discrete representations via information maximizing self-augmented training," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1558–1567.

[141] S. Melacci and M. Gori, "Unsupervised learning by minimal entropy encoding," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 12, pp. 1849–1861, 2012.

[142] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *2015 IEEE Information Theory Workshop (ITW)*, 2015, pp. 1–5.

[143] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[144] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox, "On the information bottleneck theory of deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124020, 2019.

[145] A. Betti, M. Gori, and S. Melacci, "Cognitive action laws: The case of visual features," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 938–949, 2020.

[146] A. Betti, M. Gori, and S. Melacci, "Motion invariance in visual environments," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 2009–2015. [Online]. Available: https://doi.org/10.24963/ijcai.2019/278

[147] L. Ambrosio, G. Dal Maso, M. Forti, M. Miranda, and S. Spagnolo, *Ennio De Giorgi-Selected papers*. Springer-Verlag, 2006.

[148] U. Stefanelli, "The de giorgi conjecture on elliptic regularization," *Mathematical Models and Methods in Applied Sciences*, vol. 21, no. 6, p. 1377Ą1394, 2011.

[149] E. Serra and P. Tilli, "Nonlinear wave equations as limits of convex minimization problems: proof of a conjecture by de giorgi," *Annals of Mathematics*, pp. 1551–1574, 2012.

[150] M. Liero and U. Stefanelli, "A new minimum principle for lagrangian mechanics," *Journal of nonlinear science*, vol. 23, no. 2, pp. 179–204, 2013.

[151] M. Gori, M. Lippi, M. Maggini, and S. Melacci, "Semantic video labeling by developmental visual agents," *Computer Vision and Image Understanding*, vol. 146, pp. 9–26, 2016.

[152] M. Gori, S. Melacci, M. Lippi, and M. Maggini, "Information theoretic learning for pixel-based visual agents," in *Computer Vision – ECCV 2012*, A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 864–875.

[153] S. A. McMains and S. Kastner, *Visual Attention*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 4296–4302. [Online]. Available: https://doi.org/10.1007/978-3-540-29678-2_6344

[154] E. Kowler, "Attention and eye movements," in *Encyclopedia of neuroscience*. Elsevier Ltd, 2010, pp. 605–616.

[155] M. Cornia, L. Baraldi, G. Serra, and R. Cucchiara, "Predicting human eye fixations via an lstm-based saliency attentive model," *IEEE Transactions on Image Processing*, vol. 27, no. 10, pp. 5142–5154, 2018.

[156] N. Bruce and J. Tsotsos, "Saliency based on information maximization," in *Advances in neural information processing systems*, 2006, pp. 155–162.

[157] D. Zanca, S. Melacci, and M. Gori, "Toward improving the evaluation of visual attention models: a crowdsourcing approach," *CoRR*, vol. abs/2002.04407, 2020. [Online]. Available: https://arxiv.org/abs/2002.04407

[158] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[159] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," 2017.

[160] M. Tiezzi, S. Melacci, M. Maggini, and A. Frosini, "Video surveillance of highway traffic events by deep learning architectures," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 584–593.