State of the Art on: Structured Learning

Alberto Archetti, alberto1.archetti@mail.polimi.it

1. Introduction to the research topic

In recent years, deep learning models outperformed state-of-the-art techniques in many machine learning tasks, bringing ground-shaking advances in research and industrial environments. Deep learning models involve powerful function approximators to be trained over huge sets of data. As an example, the review [19] collects deep learning advances in common computer vision tasks, such as motion tracking, action recognition, human pose estimation, and semantic segmentation.

The most common building blocks of deep learning architectures are neural networks. A neural network is a parametric function, built by composition of smaller, possibly non-linear functions. Function composition can be represented by a directed graph. Neural networks are trained with the backpropagation algorithm: it searches for the parameters that minimize an error function, dependent on the available data.

Even if the literature is overflowing with incredibly successful applications of deep learning architectures based on neural networks (Table 1), they come with some structural drawbacks. The first one is explainability [7]: it is very difficult to delve into their complex structure and reconstruct the *reasoning*, in a human-understandable sense, that brings the network to a specific decision. Explainability is mandatory in large-scale, safety-critical and mission-critical applications, as requirements impose hard constraints on model verifiability. The second drawback is the size of the training set necessary for the learning procedure. Since deep neural networks have a huge set of parameters, their tuning requires a high cost in terms of data, computational time, and memory.

Recently, many researchers started experimenting with embedding algorithmic priors directly in the neural network structure. Their aim is to insert algorithmic modules with a predefined behavior in the architecture while maintaining the end-to-end training procedure. This approach increases explainability, as algorithmic priors force the network to adapt to the algorithms' interfaces. It also helps training, because it reduces the variance of the model by lowering the number of tunable parameters. Finally, it provides a stabler solution during training, being it theoretically guided. This emerging field is called Structured Learning: algorithms provide structure and neural networks provide flexibility to learn and adapt from data.

Journal Name	**Impact factor**
IEEE Transactions on Pattern Analysis and Machine Intelligence	17.730
IEEE Transactions on Neural Networks and Learning Systems	11.683
Pattern Recognition	5.898
Journal of Machine Learning Research	4.091
Conference Name	**Google Scholar H5-index**
CVPR: Conference on Computer Vision and Pattern Recognition	240
NIPS: Neural Information Processing Systems	169
ECCV: European Conference on Computer Vision	137
ICML: International Conference on Machine Learning	135

Table 1: List of the most prestigious journals and conferences related to deep learning from `guide2research.com`

1.1. Preliminaries

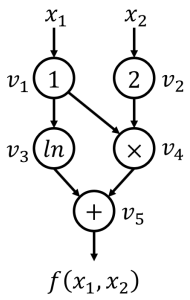


Figure 1

Automatic differentiation (AD) is a family of techniques for evaluating derivatives of numeric functions, expressed as computer programs, efficiently and accurately [3, 14]. Automatic differentiation is fundamentally different from numerical or symbolical differentiation. Any computer program, for a fixed input, can be treated as a trace of operations on that input. An AD procedure builds a computational graph that encodes the program structure, divided into a set of elementary, differentiable operations. The procedure then derivates punctually each operation and combines the intermediate results using the chain rule. AD has two modes: forward and backward. The former is simpler to understand and can be implemented using dual numbers and operator overloading. Its complexity is proportional to the number of inputs of a program. The latter is the generalization of backpropagation, thus it is used in neural networks training. Its complexity is proportional to the number of outputs of a program. Consider, for example, the graph in Figure 1 that represents the function $f(x_1, x_2) = ln(x_1) + 2x_1x_2$. Suppose we want to evaluate $\frac{\partial f(x_1,x_2)}{\partial x_1}$ with forward mode AD. First, for each node we calculate the derivative with respect to the incoming nodes: $\dot{v_1} = \dot{x_1} = 1$, $\dot{v_2} = 2\dot{x_2} = 0$, $\dot{v_3} = \frac{\dot{v_1}}{v_1}$, $\dot{v_4} = \dot{v_1}v_2 + v_1\dot{v_2}$ and $\dot{v_5} = \dot{v_3} + \dot{v_4}$. By substitution we get $\dot{v_5} = \frac{1}{x_1} + 2x_2 = \frac{\partial f(x_1,x_2)}{\partial x_1}$. AD can be applied also to programs with complex control flow. This is possible thanks to the fact that differentiation through elementary operations is carried along the execution trace, without considering non-traversed control paths.

Today we have many frameworks to choose from in order to experiment with deep learning and automatic differentiation. Their goal is to allow the construction of complex neural network architectures in an accessible and intuitive way, hiding most of the implementation issues and optimization techniques to make the networks fast. They provide an abstract interface, usually written in Python, to design, train and test complex models. They offer AD support for parameter optimization and GPU support for fast training. The most common deep learning libraries, sorted by the number of stars on GitHub, are Tensorflow, Keras, PyTorch and Caffe[1]. In [18], the authors present different approaches taken to AD implementation in the most common deep learning frameworks for Python. The review depicts a topology of the most used implementation techniques for AD, with respective advantages and disadvantages.

1.2. Research topic

In a deep neural architecture, devoted to the solution of a complex task, we can identify several components. Each component is designed to solve an easier sub-task. It may happen that, for some sub-tasks, we already have an algorithm that provides a solution, more efficiently and robustly than a neural network. The issue is that, by directly attaching the algorithm to the network, we lose the possibility of training end-to-end the whole system, as backpropagation requires all components of a neural network to be differentiable. In general, there is no unique way to build the derivative of an algorithm, as it may contain crisp operations and non-differentiable control flow. Nevertheless, end-to-end training is a desirable property, because the system can be trained altogether, with a single dataset. Thanks to the advances in the AD technology (Section 1.1), we can differentiate through programs with complex control flow. Structured Learning can bridge model-free and model-based approaches, by providing a framework in which classic algorithms and deep learning modules can coexist. Explainability is increased, as the algorithmic priors fix the interface between architectural modules with their parameters, that have a clear interpretation. The learning phase is more data efficient, thanks to the strong prior that the algorithms provide. Finally, the solution is constrained by fixed procedures, so it is structured and guided by design.

The research question is the feasibility and range of applicability of Structured Learning. Up to which level can we robustly differentiate through complex algorithms? What are the constraints we are subject to when mixing algorithmic and neural modules? What are the technological issues we must face in order to obtain easy and efficient integration?

[1] `https://github.com/mbadry1/Top-Deep-Learning`

2. Main related works

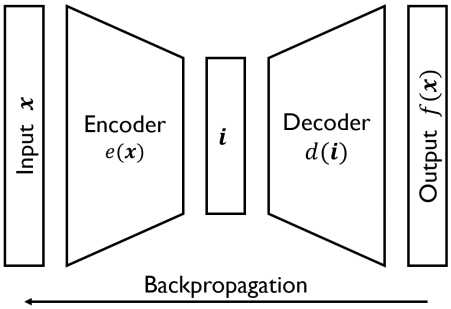
2.1. Classification of the main related works

Algorithmic prior embedding is becoming an important tool in deep learning research. It is commonly involved in computer vision with differentiable renderers for 3D reconstruction. Some other fields in which it started to spread include physics simulations with differentiable physics engines, automation and control, and logical reasoning. In all these fields research brought many important advances and results. The common trend is to embed this prior knowledge into unstructured architectures, in order to give a significant advantage over the learning procedure, and provide a tool for model verifiability.

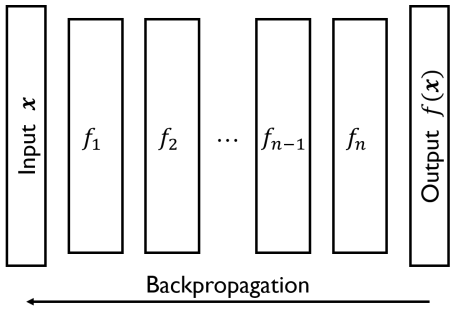
The works that involve Structured Learning ideas are currently scattered and usually exploit AD libraries for algorithm differentiation. All works presented here after empirically show improvements in data efficiency and performance with respect to unstructured deep neural networks. The main issue identified by the experiments is the increase in computational complexity during training. Among the works, we identify two main architectures:

Structured Autoencoder This architecture f is based on deep learning autoencoders (Fig. a). It is composed of two networks. The first is the *encoder* e that takes an input $\mathbf{x}$ and transforms it into an intermediate representation $\mathbf{i} = e(\mathbf{x})$. The second is the *decoder* d that reconstructs the best approximation of the input, given $\mathbf{i}$. The network, $f(\mathbf{x}) = d(e(\mathbf{x}))$, is trained to learn the identity function, that is, the difference between $\mathbf{x}$ and $f(\mathbf{x})$, according to a distance metric. The goal of classical autoencoders is to find the intermediate compact representation $\mathbf{i}$ that contains only meaningful information for the reconstruction process. Conversely, a Structured Autoencoder has an algorithm as a decoder, so $\mathbf{i}$ is fixed and corresponds to the inputs expected by the algorithm. The goal, then, is to produce the encoder e that finds the parameters $\mathbf{i}$ such that the algorithm returns the best approximation of the input. Structured Autoencoders are suited for the solution of inverse problems ($e \approx d^{-1}$) and they don't require external labeling.

Structured Pipeline In order to solve a complex task, this architecture f mixes n modules f_i that can be neural or algorithmic in a single pipeline, such that $f = f_n \circ f_{n-1} \circ \cdots \circ f_2 \circ f_1$ (Fig. b). The idea is that we may have algorithms that can solve some subtasks f_i of the problem soundly and more robustly than a neural network. We can integrate any algorithm in a deep architecture, but, in order to maintain the end-to-end learning, we must define the derivative of the algorithmic modules, as requested by the backpropagation procedure. Structured Pipelines are more general than Structured Autoencoders, but they are not still as common in the literature, as they require supervised training and more computational power. In principle, the modules of a Structured Architecture can be composed according to any topology, including recurrent schemes with loops.



(a) Architecture of a generic autoencoder. In a standard autoencoder every component is neural, while, in a Structured Autoencoder, $d(\mathbf{i})$ is a differentiable algorithm.



(b) Architecture of a Structured Pipeline. The input traverses n modules, each implementing a function f_i, in neural or differentiable algorithm form.

2.2. Brief description of the main related works

2.2.1 3D reconstruction

Rendering is the process of building an image from a compact description of the scene. Several of the presented architectures involve Structured Autoencoders with differentiable renderers as decoders, such as NMR [11], Soft Rasterizer [12] and OpenDR [13].

Inverse Transport Networks (ITN) by Che et al. [5] rely on a custom AD-based differentiable renderer to extract the physical scene parameters $\pi = \{\text{sensor, geometry, material, illumination}\}$ from an image I. Let's call the light transport process of photons towards the camera sensor $\mathcal{T}(\pi)$. The analysis-by-synthesis approach aims at finding $\hat{\pi} = \arg\min_\pi ||I - \mathcal{T}(\pi)||^2$. It is very good for deriving guarantees about the reconstructed parameters $\hat{\pi}$, but it is also computationally costly. The alternative approach is to use supervised learning with a parameter estimating neural network $\mathcal{N}[\mathbf{w}](I)$. With that we can find $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} ||\pi - \mathcal{N}[\mathbf{w}](I)||^2$ and then use $\mathcal{N}[\hat{\mathbf{w}}](I)$ to evaluate $\hat{\pi}$. This approach loses the theoretical guarantees, but is more efficient than the previous one. By following the Structured Learning approach and using the differentiable renderer, the authors propose a Structured Autoencoder, whose loss function is regularized according to the light transport function $\mathcal{T}(I)$:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} ||\pi - \mathcal{N}[\mathbf{w}](I)||^2 + \lambda ||I - \mathcal{T}(\mathcal{N}[\mathbf{w}](I))||^2$$

The name Inverse Transport Network comes from the fact that, given this loss function, we have that $\mathcal{N}[\hat{\mathbf{w}}] \approx \mathcal{T}^{-1}$. The authors experimentally show the performance improvement of ITN over the model-free neural network (no regularization term in the loss function) and the single scattering network (only $\mathcal{T}(I)$ term in the loss function).

Tewari et al. [17] propose an autoencoder that involves a model-based decoder. It reconstructs 3D human faces from a semantic vector, encoding face pose, shape, expression, skin reflectance, and scene illumination. From this work, the explainability advantage is huge with respect to a black-box decoder, as the intermediate representation has a clear interpretation. The loss function is composed of photometric alignment (dense and sparse) and a statistical regularizer that takes into account the plausability of the inferred parameters. The authors tested the architecture against several competitors and identify as main drawback the inability of working in presence of facial hair or strong occlusion by external objects.

2.2.2 Physics simulation

Belbute-Peres et al. [6] present a differentiable physics engine that can be integrated into deep learning systems. The physical simulation is expressed as a linear complementary problem, where equality and inequality constraints represent the constraints on the movements of rigid bodies. The authors exploit OptNet [2], the optimization layer by of Amos et al. that is able to solve parametric optimization problems into a neural network layer, returning as output the solution of the optimization problem. They test several systems using OpenAI Gym [4], a library for experimenting with classical reinforcement learning problems, and show some interesting properties that a system gains with integrated physical knowledge. In particular, they highlight that the system can infer physical parameters from observations and perform accurate predictions, all with better sample efficiency than an unstructured network.

The work by Seo et al. [16] underlines the importance of Physics as one of the most important tools for describing how the real world behaves. They introduce an architecture based on graph networks able to incorporate prior knowledge as partial differential equations over time and space into neural networks. First, they show how vector operators (gradient, divergence, laplacian and curl) in Euclidean domain can be analogously defined on a discrete graph domain. Then, they define the update rule to propagate information through the graph, by analyzing some examples from the physics literature. Finally, they provide an autoencoder architecture in which the encoder takes a graph $\mathcal{G}$ and encodes it into a latent space, obtaining another graph $\mathcal{H}$. $\mathcal{H}$ is updated for a custom number of steps, obtaining $\mathcal{H}'$ and finally the decoder returns $\mathcal{H}'$ back to the original domain, obtaining $\hat{\mathcal{G}}'$. The architecture admits two types of loss functions: the first is physics-informed and evaluates the difference between $\mathcal{H}$ and $\mathcal{H}'$ and the second is a supervised loss between $\hat{\mathcal{G}}'$ and the target $\mathcal{G}'$.

2.2.3 Control

Karkus et al. [10] provide a Structured Pipeline of algorithmic modules, built on top of a neural network architecture for robotic motion. The Differentiable Algorithm Network (DAN) is composed of four modules: *vision*, *filter*, *plan*, and *control*. Each of the modules implements a differentiable algorithm with tunable parameters. Experiments show very peculiar properties of the architecture. For example, when the algorithmic modules contain simplifying assumptions that do not correspond to the environment characteristics, the system automatically learns a model that compensates for such mismatch. In general, the authors experimentally show the ability of the system to compensate for approximations in algorithm parameters, model misspecifications, and approximate decomposition of the overall task into submodules.

Amos et al. [1] provide the Model Predictive Control (MPC) algorithm as a differentiable policy class for reinforcement learning. The authors point out the recent prominence of attempting to incorporate planning and control methods into the loop of recurrent deep learning architectures. MPC leverages the model of the controlled system and, at each time steps, solves an optimization problem in order to produce the best sequence of control actions to reach a goal. Their contribution shows how to differentiate the optimization procedure essentially for free, without the need of unrolling through the operations of the optimizer.

2.2.4 Differentiable programming

Many researchers think that great advances in machine learning and scientific computing research areas may come from the possibility of having AD as a basic language feature. Innes presents Zygote [8], a source-to-source automatic differentiation tool for Flux, the machine learning library of Julia. The advantage of Zygote, with respect to AD tools for other programming languages, is that it can efficiently achieve automatic differentiation even in programs with very complex control flow, with just a few hundred lines of code. Proceeding on this line, in a way that is close to the goal of Structured Learning, Innes et al. [9] present a new paradigm for programming languages, differentiable programming (∂P), and propose it as the missing link between scientific computing and machine learning. Among the advantages that a deep interconnection between these two worlds can bring, they point out the possibility of differentiating through simulators for solving inverse problems. The power of this idea is explored in Sections 2.2.1 and 2.2.2.

2.3. Discussion

The Structured Learning approach has proven to be very beneficial when facing explainability issues and sample efficiency in deep learning models. However, research up to now still leaves two issues open.

The first concerns the differentiability of control flow. Automatic differentiation is a powerful technique, but it does not provide a smooth derivative of a program *everywhere*. It discards the control flow information such as branches and exit conditions of loops, making derivatives taken from similar inputs that don't follow the same execution path potentially very different. A possible solution is provided by [15]. The authors propose a Turing complete differentiable interpreter, based on a WHILE language. The approach goes in a different direction than differentiable programming, as it is not based on automatic differentiation. They define the probability of execution of an instruction and modify it according to the control flow of the program. For example, in the body of a while loop, the probability of execution of instruction i is $p^{(new)}(i) = p^{(old)}(i) \cdot \phi$, where ϕ is a smoothing function that varies with the number of iterations of the loop. The authors point out computational difficulties, raised in their experiments, that must be investigated further. Moreover, the paper does not address the potential scalability issue that this technique may have in particularly complex programs.

The second issue of Structured Learning is performance. Despite the huge research activity in training time optimization of deep neural networks, the process is still costly in terms of computational resources. By embedding algorithms that must be traversed by the backpropagation procedure, we are just increasing the overall computational cost. The challenge for the future of Structured Learning research is to find the optimal tradeoff between the advantages of data efficiency and the computational burden that structure requires.

References

[1] Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. 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. 8289–8300.

[2] Amos, B., and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. *CoRR abs/1703.00443* (2017).

[3] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. Automatic differentiation in machine learning: a survey, 2015.

[4] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

[5] Che, C., Luan, F., Zhao, S., Bala, K., and Gkioulekas, I. Inverse transport networks, 2018.

[6] de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. 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. 7178–7189.

[7] Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. Explaining explanations: An overview of interpretability of machine learning, 2018.

[8] Innes, M. Don't unroll adjoint: Differentiating ssa-form programs. *CoRR abs/1810.07951* (2018).

[9] Innes, M., Edelman, A., Fischer, K., Rackauckas, C., Saba, E., Shah, V. B., and Tebbutt, W. A differentiable programming system to bridge machine learning and scientific computing. *ArXiv abs/1907.07587* (2019).

[10] Karkus, P., Ma, X., Hsu, D., Kaelbling, L. P., Lee, W. S., and Lozano-Perez, T. Differentiable algorithm networks for composable robot learning, 2019.

[11] Kato, H., Ushiku, Y., and Harada, T. Neural 3d mesh renderer. *CoRR abs/1711.07566* (2017).

[12] Liu, S., Li, T., Chen, W., and Li, H. Soft rasterizer: A differentiable renderer for image-based 3d reasoning, 2019.

[13] Loper, M. M., and Black, M. J. Opendr: An approximate differentiable renderer. In *Computer Vision – ECCV 2014* (Cham, 2014), D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Springer International Publishing, pp. 154–169.

[14] Margossian, C. C. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* (Mar 2019), e1305.

[15] Petersen, F., Borgelt, C., and Deussen, O. Algonet: C^{∞} smooth algorithmic neural networks. *CoRR abs/1905.06886* (2019).

[16] Seo, S., and Liu, Y. Differentiable physics-informed graph networks, 2019.

[17] Tewari, A., Zollhöfer, M., Kim, H., Garrido, P., Bernard, F., Pérez, P., and Theobalt, C. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction, 2017.

[18] van Merrienboer, B., Breuleux, O., Bergeron, A., and Lamblin, P. Automatic differentiation in ml: Where we are and where we should be going. 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. 8757–8767.

[19] Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience 2018* (02 2018), 1–13.