Local Propagation in Constraint-based Neural Networks

Giuseppe Marra*, Matteo Tiezzi†, Stefano Melacci†, Alessandro Betti†, Marco Maggini†, Marco Gori†

*Dept. of Information Engineering
University of Florence
Florence, Italy
{g.marra,alessandro.betti}@unifi.it

†Dept. of Information Engineering and Mathematical Sciences
University of Siena
Siena, Italy
{mtiezzi,mela,maggini,marco}@diism.unisi.it

Abstract—In this paper we study a constraint-based representation of neural network architectures. We cast the learning problem in the Lagrangian framework and we investigate a simple optimization procedure that is well suited to fulfil the so-called architectural constraints, learning from the available supervisions. The computational structure of the proposed Local Propagation (LP) algorithm is based on the search for saddle points in the adjoint space composed of weights, neural outputs, and Lagrange multipliers. All the updates of the model variables are locally performed, so that LP is fully parallelizable over the neural units, circumventing the classic problem of gradient vanishing in deep networks. The implementation of popular neural models is described in the context of LP, together with those conditions that trace a natural connection with Backpropagation. We also investigate the setting in which we tolerate bounded violations of the architectural constraints, and we provide experimental evidence that LP is a feasible approach to train shallow and deep networks, opening the road to further investigations on more complex architectures, easily describable by constraints.

I. INTRODUCTION

In the last years, neural networks have become extremely widespread models, due to their role in several important achievements of the Machine Learning community [1]. 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 [2], [3], or of new architectures that are created as alternatives to existing models [4]. Backpropagation [5] is assumed to be the "de facto" algorithm for training neural nets.

In this paper, we are inspired by the ideas of describing learning using the unifying notion of "constraint" [6], [7], and we nicely intercept the work of [8], where a theoretical framework for Backpropagation is studied in a Lagrangian formulation of learning. In particular, we regard the neural architecture as a set of constraints that correspond with the neural equations, which enforce the consistency between the input and the output variables by means of the corresponding weights of the synaptic connections. However, differently from

[8], we do not only focus on the derivation of Backpropation 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 idea of training networks represented with constraints and extending the space of learnable parameters has been originally introduced by [9], using an optimization scheme based on quadratic penalty. The approach of [9] is built on the idea of finding an inexact solution of the original learning problem, and it relies on a post-processing procedure that refines the last-layer connections. The related approach of [10] involves closed-form solutions, but most of the architectural constraints are softly enforced, and further additional variables are introduced to parametrize the neuron activations. Other approaches followed these seminal works to implement constraining schemes for block-wise optimization of neural networks [11].

Differently, in this paper we propose a hard-constraining scheme based on the augmented Lagrangian and on the optimization procedure of [12], in which we search for saddle points in the adjoint space by a differential optimization process. This procedure is very easy to implement.

The obtained results show that constraint-based networks, even if optimized with the proposed simple strategy, can be trained in an effective way. However, the main goal of the paper 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 which 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 architecture search in the deep learning scenario.

Differently from [9], [10], we study the connections with BackPropagation, performing an extended comparison on several benchmarks. Moreover, we do not optimize variables associated to the activation score of each neuron. We use the Lagrangian approach to find a solution of the optimization problem that includes hard constraints. Instead of following a soft optmization procedure, we relax the constraints with ϵ-based functions, where the tolerance is fixed and defined in advance.

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

II. Constraint-based Neural Networks

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, in the context of this paper and without any loss of generality, is 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. The function $V(W_H x_{H,i}, y_i)$ 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 by describing the network architecture with a set of constraints. In particular, all the $x_{\ell,i}$'s become variables of the learning problem, and they are constrained to fulfil the so-called (hard) *architectural*

constraints $x_{\ell,i} = \sigma(W_{\ell-1}x_{\ell-1,i})$, i.e., [1]

$$\text{minimize} \quad \sum_{i=1}^{N} V(W_H x_{H,i}, y_i)$$
$$\text{subject to} \quad \mathcal{G}(x_{\ell,i} - \sigma(W_{\ell-1}x_{\ell-1,i})) = 0, \quad \forall(i, \ell) \quad (1)$$

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 $\sigma(W_{\ell-1}x_{\ell-1,i})$. We will also make use of the notation $\mathcal{G}_{\ell,i}$ to compactly indicate the left-hand side of Eq. (1). In the Lagrangian framework [13], if $\lambda_{\ell,i}$ are the Lagrange multipliers associated to each architectural constraint, then we can write Lagrangian function $\mathcal{L}$ as

$$\mathcal{L}(\mathcal{W}, \mathcal{X}, \Lambda) = \sum_{i=1}^{N} \Big(V(W_H x_{H,i}, y_i) + \sum_{\ell=1}^{H} \lambda_{\ell,i}^T \mathcal{G}_{\ell,i} \Big), \quad (2)$$

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 $x_{\ell,i}$'s variables; Λ collects the Lagrange multipliers (T is the transpose operator). The Lagrangian $\mathcal{L}$ can also be augmented with a squared L_2 norm regularizer on the network weights, scaled by a positive factor c.

III. Local Propagation

Despite the variety of popular approaches that can be used to solve the constrained problem above [13], we decided to focus on the optimization procedure studied in the context of neural networks in [12]. In the proposed Local Propagation (LP) algorithm, learning consists in a "differential optimization" process that converges towards a saddle point of Eq. (2), minimizing it with respect to $\mathcal{W}$ and $\mathcal{X}$, and maximizing it with respect to Λ. 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 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 $\mathcal{G}_{1,i}$ (that is, the farthest portions of the architecture) contribute to the Lagrangian of Eq. (2). 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. Once LP has converged, the architectural constraints of Eq. (1) are fulfilled, 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

---

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

the network weights while still relying on the classic forward pass during inference.

One of the key features of LP is the locality in the gradient computations. Before going into further details, we remark that, in the context of this paper, the term *locality* refers to those 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 and a backward steps that involve all the neural units of all layers (see Fig. 1 (a)) Differently, in the case of LP we have

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = -\sum_{i=1}^{N} \left(\lambda_{\ell+1,i} \odot \mathcal{G}'_{\ell+1,i} \odot \sigma'(W_\ell x_{\ell,i}) \right) x_{\ell,i}^T \qquad (3)$$

$$\frac{\partial \mathcal{L}}{\partial x_{\ell,i}} = \lambda_{\ell,i} \odot \mathcal{G}'_{\ell,i} - W_\ell^T \left(\lambda_{\ell+1,i} \odot \mathcal{G}'_{\ell+1,i} \odot \sigma'(W_\ell x_{\ell,i}) \right) \quad (4)$$

$$\frac{\partial \mathcal{L}}{\partial x_{H,i}} = \lambda_{H,i} \odot \mathcal{G}'_{H,i} + W_H^T \left(V'(W_H x_{H,i}, y_i) \right) \qquad (5)$$

$$\frac{\partial \mathcal{L}}{\partial \lambda_{\ell,i}} = \mathcal{G}_{\ell,i} \qquad (6)$$

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

a) Parallel Computations over Layers.: It is well known that in Backpropagation 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 parallelization of the matrix operations within each layer, while in the case of LP, the locality in the gradient computation allows us to go beyond that. We can promptly see from Eq. (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. (3-6)).

b) Deep Learning in the Adjoint Space.: 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. (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 [14]. The LP natively embeds this property so that, differently from [14], 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.

c) Recovering Backpropagation.: 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. (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) and Eq. (4) can be respectively rewritten as

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

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

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

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 $x_{\ell,i}$, Backpropagation loses the locality and parallelization properties characterizing our algorithm since the gradients cannot rely anymore on the variables of neighboring layers only but depend on all the variables of the architecture.
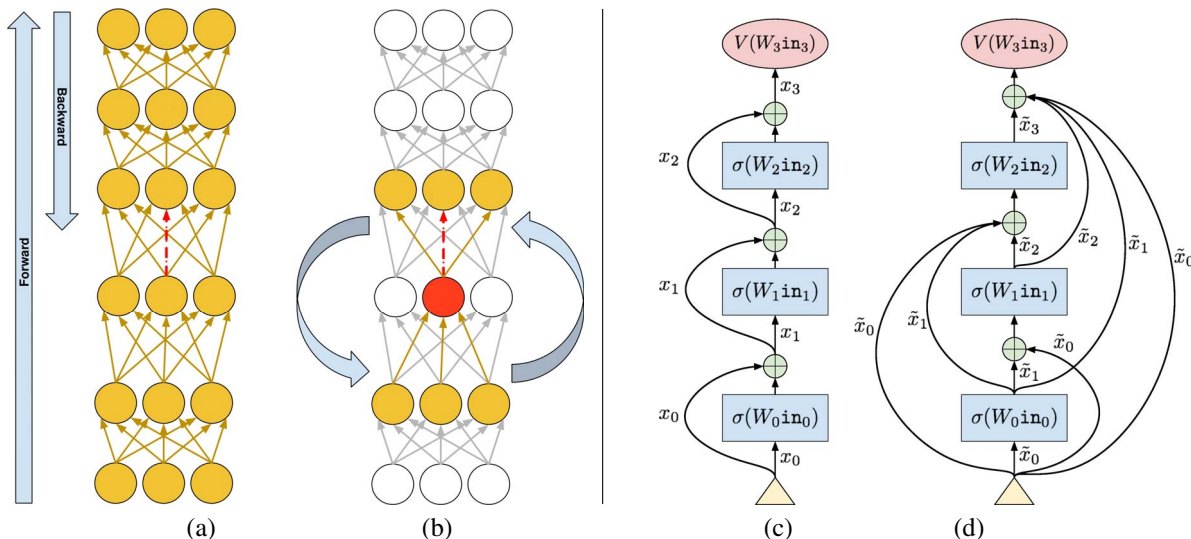
Fig. 1. Left: the neurons and weights that are involved in the computations to update the red-dotted weight w are highlighted in yellow. (a) Backpropagation; (b) Local Propagation – the computations required to update the variables x, λ (associated to the red neuron) are also considered. Right: (c) ResNet in the case of $H = 3$, and (d) after the change of variables $(x_\ell \to \tilde{x}_\ell)$ described in Sec. IV. Greenish circles are sums, and the notation in_ℓ inside a rectangular block indicates the block input.

A. 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. (1). In this paper, we focus on the class of ϵ-insensitive functions, and, in particular, on the following two cases $\mathcal{G} \in \{\text{eps}, \text{lineps}\}$,

$$\text{eps}(a) = \max(|a| - \epsilon, 0)$$
$$\text{lineps}(a) = \max(a, \epsilon) - \max(-a, \epsilon) .$$

Both the functions are continuous, they are zero in $[-\epsilon, \epsilon]$, and they are linear out of such interval. However, $\text{eps}(\cdot)$ is always positive, while $\text{lineps}(\cdot)$ is negative for arguments smaller than $-\epsilon$. When plugged into Eq. (1), they allow the architectural constraints to tolerate a bounded mismatch in the values of $x_{\ell,i}$ and $\sigma(W_{\ell-1}x_{\ell,i})$ (ϵ-insensitive constraints). Let us consider two different examples indexed by i and j, for which we get two similar valus $\sigma(W_{\ell-1}x_{\ell,i})$ and $\sigma(W_{\ell-1}x_{\ell,j})$ in a certain 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 to not 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 [15] when the selected drop-unit factor is too large.

A key difference between $\text{eps}(\cdot)$ and $\text{lineps}(\cdot)$ is the effect they have in the development of the Lagrange multipliers. It is trivial to see that, since $\text{eps}(\cdot)$ is always positive, the multipliers $\lambda_{\ell,i}$ can only increase during the optimization (Eq. (6)). In the case of $\text{lineps}(\cdot)$, the multipliers can both increase or decrease. We found that $\text{eps}(\cdot)$ leads to a more stable learning, where the violations of the constraints change more smoothly that in the case of $\text{lineps}(\cdot)$. As suggested in [12] and as it is also popular in the optimization literature [13], a way to improve the numerical stability of the algorithm is to introduce the so called Augmented Lagrangian, where $\mathcal{L}$ of Eq. (2) is augmented with an additive term $\rho\|\mathcal{G}_{\ell,i}\|^2$, for all i, ℓ.

IV. POPULAR NEURAL UNITS

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. (1) to model convolutional units and implement Convolutional Neural Networks (CNNs) [16], 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.

a) Recurrent Neural Networks.: At a first glance, RNNs [17] 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)[2]. 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 $\ell \in [1, H]$,

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

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. III) 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 [19]–[21]

b) Residual Networks.: ResNets [22] 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 [22] (without being limited to such networks). The most generic form of a single residual unit is described in [23],

$$x_\ell = z(h(x_{\ell-1}) + f(W_{\ell-1}x_{\ell-1})) \ . \tag{7}$$

In the popular paper of [22], 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

$$\mathcal{G}(x_\ell - z(h(x_{\ell-1}) + f(W_{\ell-1}x_{\ell-1}))) = 0 \ . \tag{8}$$

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 [23], 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,

$$x_\ell = x_{\ell-1} + \sigma(W_{\ell-1}x_{\ell-1}) \ , \tag{9}$$

as sketched in Fig. 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 [22] 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

[2]What we study here can be further extended to the case of Long Short-Term Memories (LSTMs) [18]

implies that the gradient of the loss function $V(W_H x_H, y)$ with respect to x_ℓ is

$$\frac{\partial V}{\partial x_\ell} = \frac{\partial V}{\partial x_H}\left(1 + \frac{\partial}{\partial x_\ell}\sum_{j=\ell}^{H-1}\sigma(W_j x_j)\right) \ , \tag{10}$$

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. (9) are

$$\mathcal{G}(x_\ell - x_{\ell-1} - \sigma(W_{\ell-1}x_{\ell-1})) = 0 \ . \tag{11}$$

As a matter of fact, the loss $V(W_H x_H, y)$ 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. (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. (11) becomes[3]

$$\mathcal{G}\left(\tilde{x}_\ell - \sigma\left(W_{\ell-1}\cdot\sum_{j=0}^{\ell-1}\tilde{x}_j\right)\right) = 0 \ , \tag{12}$$

where the arguments of the loss function change to $V(W_H \cdot \sum_{j=0}^H \tilde{x}_j, y)$. 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. 1 (d). Given this new form of the first 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

$$\frac{\partial \mathcal{L}}{\partial \tilde{x}_\ell} = \frac{\partial V}{\partial \tilde{x}_\ell} + \frac{\partial}{\partial \tilde{x}_\ell}\sum_{j=\ell}^H \mathcal{G}_j \ . \tag{13}$$

Differently from Eq. (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.

V. EXPERIMENTS

We designed a batch of experiments aimed at validating our simple local optimization approach to constraint-based networks. 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 our algorithm (i.e. locality and parallelization) does not correspond to a loss in performance w.r.t. Backpropagation, even if the search space has been augmented with unit activations and Lagrange multipliers variables.

We performed experiments on 7 benchmarks from the UCI repository [24], and on the MNIST data (Table I). The MNIST is partitioned into the standard training, validation and test sets, while in the case of the UCI data we followed the

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

TABLE I
NUMBER OF PATTERNS, OF INPUT FEATURES AND OF OUTPUT CLASSES OF
THE DATASETS EXPLOITED FOR BENCHMARKING OUR 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
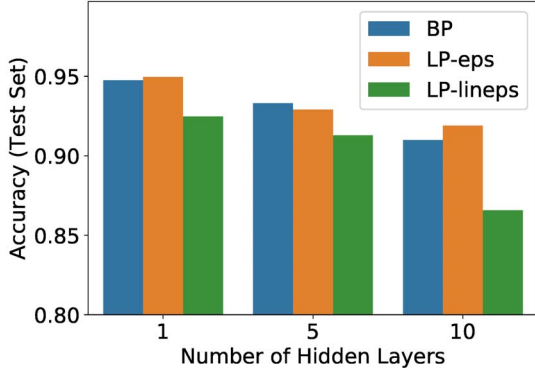
Fig. 2. Accuracies of BP and LP (with different ϵ-insensitive functions, eps, lineps) on the MNIST data.

Fig. 3. Convergence speed of BP and LP in the MNIST dataset (top), and in the Letter data (bottom).

experimental setup of [25], where the authors used the training and validation partitions of [26] to tune the model parameters, and 4-folds to compute the final accuracy (averaged over the 4 test splits)[4].

We evaluated several combinations of the involved parameters, varying them in: $\epsilon \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$, $\rho \in \{10^{-2}, 10\}$, $c \in \{0, 0.001\}$, 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 to BP and LP.

Since similar behaviours are shown by both sigmoid and ReLU activations, we exploited in our experiments 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 $\mathcal{W}$.

We evaluated the accuracies of BP and LP focussing on the same pair of architectures (sigmoidal activation units), that is composed 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 II. Both algorithms perform very similarly, with LP having some minor overall improvements over BP.

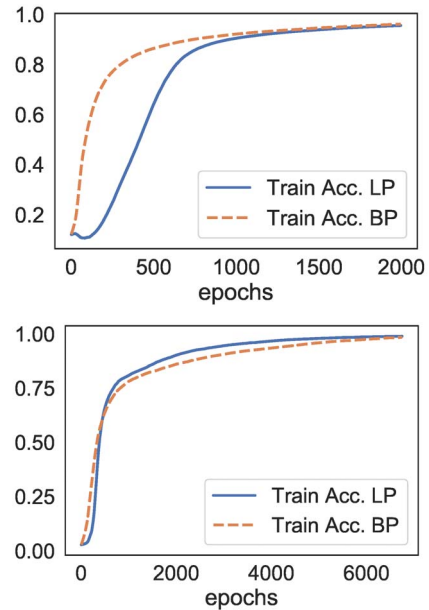[4]In the case of the Adult data we have only 1 test split.

Similar conclusions can be drawn in the case of MNIST data, as shown in Fig. 2. 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 III-A. We considered 5 different runs, reporting the test accuracy corresponding to the largest result in the validation data. When using lineps($\cdot$) as $\mathcal{G}(\cdot)$ 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 (a,b). This sometimes implies a larger number of iterations with respect to BackPropagation to converge to a solution (Fig. 3 (a) - MNIST), as expected, but it is not always the case (Fig. 3 (b) - Letter).

In order to better understand how LP behaves, we deeply explored the previously discussed results of Table II. First, we evaluated the role of the ϵ-insensitive constraints, reporting in Table III (top-half) differentiated results for the case in which $\epsilon =$ and $\epsilon > 0$. Then, we explored the effect of including the L_1-norm regularization term, as shown in Table III (*bottom-half*) ($\alpha = 0$ means no-L_1-regularization). In the case of shallow networks, $\epsilon > 0$ offers performances that, on average, are preferable or on-par to the case in which no-tolerance is considered (both for eps and lineps). This consideration is not evident in the case of deeper nets, where a too strong

TABLE II
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

TABLE III
ACCURACIES OF LP WHEN USING ($\epsilon > 0$) OR NOT USING ($\epsilon = 0$) 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 eps AND lineps FUNCTIONS, AND WE COMPARE ARCHITECTURES WITH $H = 1$ HIDDEN LAYER (100 UNITS) AND $H = 3$ HIDDEN LAYERS (30 UNITS EACH).

	$H = 1$				$H = 3$			
	$\epsilon = 0$ (eps)	$\epsilon > 0$ (eps)	$\epsilon = 0$ (lineps)	$\epsilon > 0$ (lineps)	$\epsilon = 0$ (eps)	$\epsilon > 0$ (eps)	$\epsilon = 0$ (lineps)	$\epsilon > 0$ (lineps)
Adult	85.33	85.33	**85.43**	85.27	**85.34**	85.25	**85.25**	85.23
Iono	**91.48** ±2.95	91.19 ±2.71	90.34 ±0.98	**91.19** ±2.18	**94.60** ±1.86	94.32 ±1.14	**92.61** ±1.27	91.19 ±0.94
Letter	**94.94** ±0.05	94.85 ±0.06	93.71 ±0.20	**94.00** ±0.27	87.60 ±0.48	87.54 ±0.48	**90.42** ±0.78	90.39 ±0.27
Pima	75.39 ±2.03	**77.21** ±2.79	75.00 ±1.77	**75.78** ±1.97	75.52 ±1.91	**75.91** ±3.34	74.48 ±0.90	**75.91** ±1.54
Wine	97.73 ±1.61	**98.86** ±1.14	98.30 ±1.88	98.30 ±1.88	97.73 ±3.94	97.73 ±2.78	**98.86** ±1.97	97.73 ±1.61
Ozone	97.04 ±0.13	97.04 ±0.13	96.96 ±0.30	**97.12** ±0.13	97.08 ±0.14	**97.20** ±0.17	**97.16** ±0.22	97.04 ±0.34
Derma.	**95.60** ±0.78	95.33 ±2.11	95.60 ±1.74	**96.70** ±1.74	**98.63** ±0.48	97.80 ±0.78	97.80 ±0.78	**98.08** ±0.91

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

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 aforementioned Fig. 2, where the best accuracies where usually associated with $\epsilon > 0$. 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 III, bottom-half), we can easily see that such regularization helps in several cases, suggesting that it is a 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.

VI. CONCLUSIONS AND FUTURE WORK

This paper presented an uncommn way of interpreting the architecture of neural networks and learning 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. We are currently studying an online implementation that is expected to strongly reduce the number of involved variables. LP opens the road to further investigations on other neural architectures, such as the ones that operate on graphs.

REFERENCES

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] 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.

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

[10] 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.

[11] 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.

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

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

[14] 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.

[15] 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.

[16] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber *et al.*, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," 2001.

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

[19] 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.

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

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

[22] 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.

[23] ——, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.

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

[25] 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

[26] 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.