The Loading Problem for Recursive Neural Networks

Marco Gori [a] Alessandro Sperduti [b]

[a] *Dipartimento di Ingegneria dell'Informazione*
Università di Siena

[b] *Dipartimento di Matematica Pura ed Applicata*
Università di Padova

Abstract

The present work deals with one of the major and not yet completely understood topics of supervised connectionist models. Namely, it investigates the relationships between the difficulty of a given learning task and the chosen neural network architecture. These relationships have been investigated and nicely established for some interesting problems in the case of neural networks used for processing vectors and sequences, but only a few studies have dealt with loading problems involving graphical inputs.

In this paper, we present sufficient conditions which guarantee the absence of local minima of the error function in the case of learning directed acyclic graphs with recursive neural networks. We introduce topological indices which can be directly calculated from the given training set and that allows us to design the neural architecture with local minima free error function. In particular we conceive a reduction algorithm that involves both the information attached to the nodes and the topology, which enlarges significantly the class of the problems with unimodal error function previously proposed in the literature.

Key words: Recursive Neural Networks, Loading Problem, Local Minima.

1 Introduction

In most real-world problems faced by neural networks, it is common practice for learning algorithms to face the presence of local minima. As a matter of fact, optimal solutions can often be found but, unfortunately, the presence of local minima seems to represent a border to the efficiency of learning. When passing that border, the cost of discovering an optimal solution can explode for large problems regardless of the chosen numerical optimization algorithm.

Minsky, in his epilogue of the expanded edition of Perceptron (Minsky and Papert, 1988) pointed out that

> ... as the field of connectionism becomes more mature, the quest for a general solution to all learning problems will evolve into an understanding of which types of learning processes are likely to work on which classes of learning problems. And this means that, past a certain point, we won't be able to get by with vacuous generalities about hill-climbing. We will really need to know a great deal more about the nature of those surfaces for each specific realm of problems that we want to solve.

This motivated the research carried out in many papers concerning the presence of sub-optimal local minima of the error function (see Bianchini *et al.* (Bianchini et al., 1998; Bianchini and Gori, 1996) for a survey in the field).

Most of the studies of the loading problem has been limited to the case of inputs represented by vectors, whereas the case of structured domains, where the inputs are typically represented by graphs, so far, has received a limited attention.

Like for the case of static data types, the problem of loading the weights of neural networks which process data structures can be seriously plagued by the presence of local minima in the error surface. In this paper, we analyze the efficiency of learning the membership of DAG's (Directed Acyclic Graphs) in terms of local minima of the error surface by relying on the principle that their absence is a guarantee of efficient learning (in the sense defined in (Gori and Meer, 2002)). We give sufficient conditions under which the error surface is local minima free. Based on studies reported in (Frasconi et al., 2000), it turns out that there are no local minima if certain conditions are met on the information attached to the nodes (labels). These conditions, however, hold independently of the network architecture, that is there is no need to increase the number of hidden units in order to face the problem of local minima. When dealing with more complex tasks, however, the hidden units play a fundamental role which is formally shown in the paper, i.e., under the assumption of existence of a solution with zero error and the use of discrete input labels, it is possible to give a data dependent upper bound on the number of hidden units sufficient to guarantee the absence of local minima. We give an algorithm, unfortunately with exponential time complexity in the worst case, to compute this upper bound from the training set. An informal presentation of the key steps used to reach the definition of the upper bound and associated algorithm is given in the following.

1.1 Outline of the main result of the paper

Here we consider Recursive Neural Networks for processing directed acyclic graphs with information attached to vertexes, as defined in (Frasconi et al., 1998). In these models, a structure is processed by defining a recursive function exploiting a suitable set of weights for processing the information attached to the vertexes (labels), and a set of weights for each possible outgoing edge from a generic vertex. We focus on the case of categorical labels. The aim is to show that when processing structural information under these conditions, it is actually possible to compute from the training data an upper bound on the number of hidden units used by a Recursive Neural Network to guarantee absence of local minima in the error function. The key steps to obtain this upper bound are the following:

i) In order to avoid spurious minima, we use target values that are inside the output range of the neurons and an error function (the threshold-LMS error function introduced by Sontang and Sussman (Sontag and Sussman, 1989)) that does not penalize outputs "beyond" the target values.

ii) We show that, provided the conditions defined in step *i)*, if the weight vectors of output neurons have no null component, then the presence/absence of local minima depends on whether matrices defined by collecting all the labels or state vectors associated to k-th outgoing edges, have maximum rank or not.

iii) We show that, in the case of categorical labels, topological information contained in the training set can be "compressed", in the sense that substructures that occur within several distinct structures (and/or several times inside the same structure), can be represented only once. As a consequence of that, gradient matrices can be expressed as the product of matrices with a reduced number of columns. This reduction "generates" constraints on the deltas (i.e. local gradient contributions) which need to be satisfied to guarantee absence of local minima. Specifically, these constraints are in the form of sums of subsets of deltas (i.e., deltas involving state variables that are "topologically" related) equated to 0. This step extends work proposed in (Frasconi et al., 1997b).

iv) Because of steps *i)*-*iii)*, it can be observed that, if a solution with zero error exists, then if it is possible to show that the constraints on the deltas introduced in step *iii)* can be solved only by setting all the individual deltas to 0, then no local minima can exist. A necessary condition for this to be true, is that all the matrices involving the state vectors associated to the k-th outgoing edges (states matrices), are such to jointly span a basis of the deltas space (note that the dimension of the delta space is not obvious because of dependencies in the training data and in the neural network model.) In fact, this would imply that the corresponding gradient matrices can be null if and only if the constraints on (all) deltas are satisfied; It turns

out that, since the state vectors are generated by an analytic function of the weights, a sufficient condition to obtain the desired necessary condition is as follows: if the number of hidden units is equal or larger than the maximum number of columns in the state matrices, then this condition is satisfied.

v) The sufficient condition stated in step *iv)* can however be improved by looking at the following information: the dependence among deltas induced by the topology of the neural network (back-propagation style relationships), and target information. These two type of information can actually be exploited to put additional constraints on the deltas that hopefully lead to simplify the already introduced constraints on deltas so to obtain equality constraints involving single deltas. An algorithm that iteratively exploits these two sources of information is proposed. The output of the algorithm is an upper bound on the number of hidden units sufficient to guarantee that when the gradient matrices are null, then necessarily all the deltas must be zero, i.e., no local minima can exist.

The concrete application of the algorithm proposed in step *v)* is given for an example, where we can appreciate the proposed reduction mechanism for the design of the network architecture.

The paper is organized as follows. In the next section, we review early studies on the problem of local minima in neural networks, while in Section 3 we introduce the basic ideas on recursive neural networks for structured domains and define the formalism which is used in the paper. In Section 4 we give general conditions for local minima free loading problems, while in Section 5 we present the core of the results for the case of graphs with categorical labels. Finally, some conclusions are drawn in Section 6.

2 Early studies on local minima

The problem of loading has been studied in different ways to shed light on its computational complexity. Through reduction to 3-SAT, it has been proven by Judd that loading a given set of examples into multilayer perceptrons is NP-complete (Judd, 1990). Related results have been found by Blum & Rivest (Blum and Rivest, 1992) who proved that training a 3-node neural net is also NP-complete, and by Sima (Sima, 2002), who proved that training a single sigmoidal neuron is hard. As clearly pointed out by Baum, however, this kind of pessimistic conclusions do not fully take the experimental protocol into account (Baum, 1991). The intractability of the loading problem is an interesting theoretical result which clearly points out the limitations of any blind learning from examples approach. Note that the reason of the intractability resides in the arbitrary selection of both the neural network and the data. When neural networks are tailored to the given examples, that is,

when focusing on a specific task, the intractability does not necessarily apply.

Other studies of the loading problem has been based on the associated error function, relying on the principle that the absence of sub-optimal local minima in the error function indicates that the learning task can efficiently be solved (*unimodal problems*). Complexity and local minima have been preliminarily linked by Frasconi *et al.* (Frasconi et al., 1997a) who proved that proper conditions on the error surface, basically related to the absence of critical points, make it possible to conclude that the weights can be learned in polynomial time. Notice that the notion of polynomial time algorithm typically refers to the discrete setting of computation, whereas most of the studies in the field are carried out with real numbers. In particular, Gori and Meer introduce the class of *unimodal problems* (Gori and Meer, 2002), that is very well suited to formalize computational issues of the loading problem. Related investigations on the general issues concerning links between continuous and discrete computation can be found in (Blum et al., 1998).

There have been many attempts to disclose the nature of the error surface and, particularly to investigate the presence of local minima. In the case of autoassociators, Baldi and Hornik (Baldi and Hornik, 1989) proposed an interesting analysis on local minima under the assumption of linear neurons. They proved that the attached cost function has only saddle points and only one global minimum. As the authors pointed out, however, it does not seem easy to extend such an analysis to the case of non-linear neurons. Sontag and Sussman (Sontag and Sussman, 1989) provided other conditions guaranteeing local minima free error surfaces in the case of single-layered networks of sigmoidal neurons. When adopting LMS-threshold cost functions, they proved the absence of local minima for linearly-separable patterns. This is of remarkable interest, in that it allows us to get rid of spurious local minima arising with an unproper joint selection of cost and squashing functions (Brady et al., 1989). Shynk (Shynk, 1990) showed that the perceptron learning algorithm may be viewed as a steepest-descent method by defining an appropriate performance function. In so doing, the problem of optimal convergence in perceptrons turns out to be closely related to that of the shape of such performance function.

However, although interesting, these analyses give no indication in the case of networks with non-linear hidden neurons.

Beginning from investigation on small examples, Hush *et al.* (Hush and Salas, 1988) gave some interesting qualitative indications on the shape of the cost surface. They pointed out that the cost surface is mainly composed of plateaux, that extend to infinity in all directions, and very steep regions. When the number of patterns is "small", they observed "stair-steps" in the cost surface, one for each pattern. When increasing the cardinality of the training set, however, the surface becomes smoother. Careful analyses on the shape of the cost

surface, also supported by a detailed investigation of an example, were proposed by Gouhara *et al.* (Gouhara and Uchikawa, 1993; Gouhara et al., 1993). They introduced the concepts of *memory and learning surfaces*. The learning surface is the surface attached to the cost function, whereas the memory surface is the region in the weight space which represents the solution to the problem of mapping the patterns onto the target values. One of their main conclusions is that the learning process "has the tendency to descend along the memory surfaces because of the valley-hill shape of the learning surface". Gori and Tesi (Gori and Tesi, 1992) provided a general analysis on the problem of local minima for multilayer perceptrons and, in particular, they proved that the error function is local minima free in the case of linearly-separable patterns also in the case of network with hidden units. Bianchini *et al.* (Bianchini et al., 1995) extended the analysis for sigmoidal multilayer networks to the case of radial basis functions, and Bianchini *et al.* (Bianchini et al., 1994) provided also some intriguing conditions guaranteeing the absence of local minima for recurrent networks that process sequences. Related studies concerning the optimal convergence of on-line learning algorithms in the case of linearly-separable patterns were carried out by Rosenblatt (Rosenblatt, 1962) for the PC learning algorithm and by Gori and Maggini (Gori and Maggini, 1996) for multilayer perceptrons.

3 Learning data structures by recursive networks

Before a problem can be solved in a connectionist framework, it is necessary to find a representation of data that is compatible with existing architectures. The large majority of methodological studies, theoretical results, and applications are limited to vector-based representations and (to a lesser extent) to sequential representations. However, recursive or nested representations, as opposed to flat attribute-value representations are needed in different situations. In the last few years connectionist models dealing with this kind of representations have been massively investigated (see e.g. (Frasconi et al., 2001, 2002)).

In this paper we analyze recursive neural networks for directed ordered acyclic graphs (DOAG) (Frasconi et al., 1998). The results reported here, however, can directly be applied to the more general class of directed positional acyclic graphs (DPAG) as well. We mainly restrict the attention to the special case of stationary transductions that, however, are relevant in practical applications. For stationary transductions we mean transductions that do not chance with time. One of the simplest connectionist assumptions consists of extending first-order [1] recurrent neural networks for sequences. The dependence of node

[1] A first order network is a network using standard neurons computing the weighted

v's state variables is expressed by means of function f using matrices (that we will call *pointer matrices*) $[\boldsymbol{A}] \doteq \{\boldsymbol{A}_k \in \mathbb{R}^{n,n},\ k = 1, \ldots, o\}$. Likewise, the information attached to the nodes is propagated by weight matrix $\boldsymbol{B} \in \mathbb{R}^{n,m+1}$ and, finally, matrix $\boldsymbol{C} \in \mathbb{R}^{p,n+1}$ yields the output of the transduction [2]. Hence

$$\boldsymbol{x}_v = \sigma\left(\boldsymbol{net}_v\right) = \sigma\left(\sum_{k=1}^{o} \boldsymbol{A}_k \cdot \boldsymbol{x}_{ch_k[v]} + \boldsymbol{B} \cdot \begin{bmatrix} \boldsymbol{u}_v \\ 1 \end{bmatrix}\right) \tag{1}$$

$$\boldsymbol{y}_v = \sigma\left(\boldsymbol{net}_v\right) = \sigma\left(\boldsymbol{C} \cdot \begin{bmatrix} \boldsymbol{x}_v \\ 1 \end{bmatrix}\right), \tag{2}$$

where $ch_k[v]$ represents the k-th child of node v. The sigmoidal function that we consider in this paper is odd (e.g. $\sigma(x) = \tanh(x)$). The recursive computational scheme behind these equations can be regarded as the forward step taking place in an associated multilayer network referred to as the *encoding network*. The architecture of the encoding network $\mathcal{N}(\mathcal{U})$ is inherited from the given DOAG $\mathcal{U}$ by simple inversion of the graph arrows. Basically, for each node v of the graph, a corresponding layer of n neurons is created and the edges of the graph are associated with the pointer matrices $[\boldsymbol{A}]$. The input and output variables corresponding to the labels of the graph are instead connected to these neurons by means of matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, respectively. The supersource s of each DOAG plays an important role. Since we consider only supersource tranductions, only the output $\boldsymbol{y}_s \in \mathbb{R}^p$ is produced. In general, this output takes on real-valued numbers but, in this paper, most of the attention is restricted to the case of *binary classification* (positive and negative examples). Note that the coding of the output $\boldsymbol{y}_v = \sigma\left(\boldsymbol{C} \cdot \begin{bmatrix} \boldsymbol{x}_v \\ 1 \end{bmatrix}\right)$ can easily be extended by considering a transformation of the state $\boldsymbol{x}_v$ based on a multilayer perceptron (MLP). When considering the computation of different DOAG's we shall also introduce an index l to address the specific structure. Given a DOAG $\mathcal{U}_l,\ l = 1, \ldots, L$, the scalar variables attached to node $v,\ v = 1, \ldots, P_l \doteq |(\mathcal{U}_l)|$ (either input/output or state) will be denoted by $u_{jlv},\ j = 1, \ldots, m$ (input variables), $x_{ilv},\ i = 1, \ldots, n$ (state variables), and $y_{klv},\ k = 1, \ldots, p$ (output variables), respectively. For the special case of the supersource, when considering the task of classification into positive and negative DOAG's, the output degenerates to one value only, which is referred to as y_l.

---

sum of inputs.

[2] Matrix $\boldsymbol{B} \in \mathbb{R}^{n,m+1}$ is chosen with $m+1$ columns in order to incorporate biases. The biases for the output neurons are instead incorporated into the last column of matrix $\boldsymbol{C} \in \mathbb{R}^{p,n+1}$.

The supervised training of the neural network is based on a collection of pairs composed of DOAG's with their own targets. Formally, let $d^-, d^+ \in \mathbb{R}$ be such that $[d^-, d^+] \subset [-1, 1]$ and define the training set $\mathcal{L}^{\#} \doteq \{(\mathcal{U}_l, d_l)\,,\ l = 1, \ldots, L\}$, where $\mathcal{U}_l \in U^{\#}$ is a DOAG and $d_l \in \{d^-, d^+\}$ its corresponding target value. The collection of DOAG's $U_L^{\#} \doteq \{\mathcal{U}_1, \ldots, \mathcal{U}_L\}$ will be useful for an analysis on graph topology. For the sake of notation, the nodes of any graph are totally ordered by any inverse topological order. Let $P_M \doteq \max P_l$ and $\mathcal{M}_{P_M}$ be the space of all matrices in $\mathbb{R}^{m+1,P_l}$, being $l \leq L$. We can associate any DOAG with its corresponding matrix by $\mathcal{V} : U^{\#} \to \mathcal{M}_{P_M} : \mathcal{U}_l \to \boldsymbol{U}_l$, where $\boldsymbol{U}_{lv} \doteq \boldsymbol{u}_{lv}$ and the order of the columns strictly follows the inverse topological order chosen for graph $\mathcal{U}_l$. In the case of two-classes only, the learning environment $\mathcal{L}^{\#}$ can be partitioned into $\mathcal{C}_+^{\#} \doteq \{\mathcal{U}_l \mid (\mathcal{U}_l, d_l) \in \mathcal{L}^{\#}\ and\ d_l = d^+\}$ and $\mathcal{C}_-^{\#} \doteq \{\mathcal{U}_l \mid (\mathcal{U}_l, d_l) \in \mathcal{L}^{\#}\ and\ d_l = d^-\}$, which collect the positive and the negative structures of the learning environment, respectively. In order to address the single graphs of sets $\mathcal{C}_+^{\#}$ and $\mathcal{C}_-^{\#}$, we use the operator $\triangleright$, that is $\triangleright\mathcal{C}_+^{\#} = \{l \mid \mathcal{U}_l \in \mathcal{C}_+^{\#}\}$. Given the pair $\left\{\mathcal{N}(\cdot), \mathcal{L}^{\#}\right\}$, the output-target data fitting is measured by means of the cost function

$$E \doteq \sum_{l=1}^{L} E_l = \sum_{l \in \triangleright\mathcal{C}_+^{\#}} \beta_+(y_l - d^+) + \sum_{l \in \triangleright\mathcal{C}_-^{\#}} \beta_-(y_l - d^-). \tag{3}$$

where

$$\begin{cases} \beta_+(\alpha) = 0 & \text{if } \alpha \geq 0 \\ \beta_+(\alpha) > 0\ \beta'_+(\alpha) < 0 & \text{if } \alpha < 0 \end{cases}, \qquad \begin{cases} \beta_-(\alpha) = 0 & \text{if } \alpha \leq 0 \\ \beta_-(\alpha) > 0\ \beta'_-(\alpha) > 0 & \text{if } \alpha > 0 \end{cases}, \tag{4}$$

$\beta'_+(0) = \beta'_-(0) = 0$, and "$\prime$" stands for differentiation with respect to α. This threshold-LMS error has been introduced by Sontag and Sussman in (Sontag and Sussman, 1989). This cost does not penalize outputs "beyond" the target values. It can be proven that the choice of this type of functions makes it possible to avoid the *spurious* local minima arising when choosing non-asymptotical values for the targets, that is when $[d^-, d^+] \subset [-1, 1]$ and $d^- \neq -1$ and (or) $d^+ \neq 1$. This kind of spurious local minima were shown in (Brady et al., 1989), while Sontag and Sussman (Sontag and Sussman, 1989) proved that they disappear when choosing threshold-LMS functions. Gori and Tesi (Gori and Tesi, 1992) proved that no such spurious local minima arise when $[d^-, d^+] = [-1, 1]$.

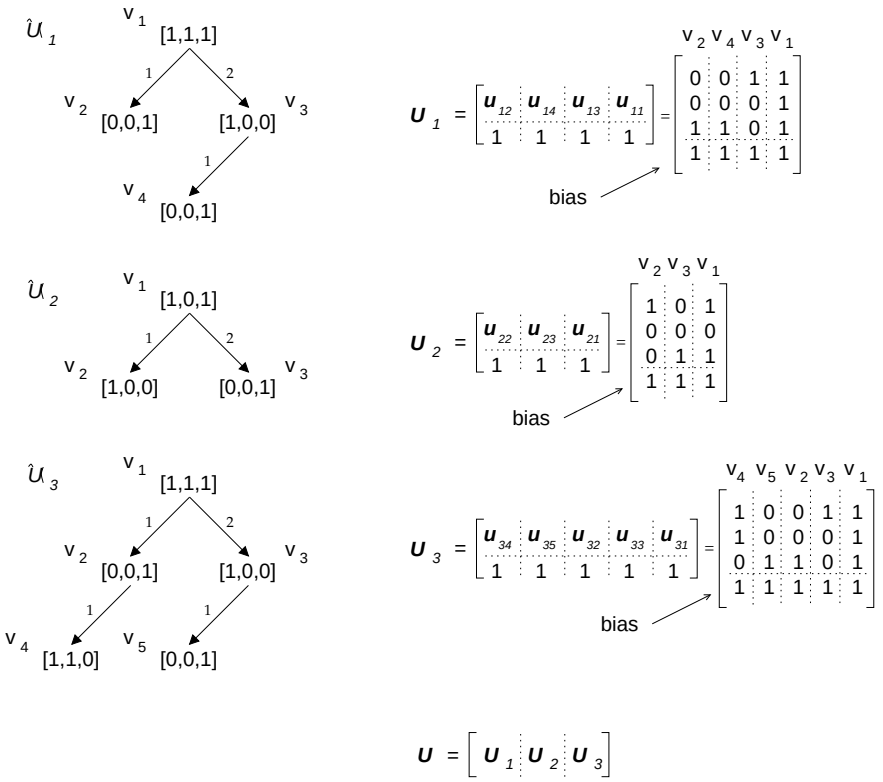

Fig. 1. Example of training set composed of three trees, i.e., $\mathcal{U}_1$, $\mathcal{U}_2$, $\mathcal{U}_3$ (left side), and the corresponding definition of the label matrices $\boldsymbol{U}_1$, $\boldsymbol{U}_2$, $\boldsymbol{U}_3$, and $\boldsymbol{U}$ (right side). Please, note that a row of 1's is added to each matrix so to include the bias. Moreover, the order of presentation of the columns of any $\boldsymbol{U}_i$, $i = 1, 2, 3$, follows an inverse topological order of the vertices of the corresponding structures $\mathcal{U}_i$, $i = 1, 2, 3$.

4 The loading problem for recursive networks

The analysis proposed in this paper is based on the implicit assumption that there exists at least one set of weights $\boldsymbol{\omega} \doteq \{[\boldsymbol{A}], \boldsymbol{B}, \boldsymbol{C}\}$ for which $E(\boldsymbol{\omega}) = 0$, that is we assume that all the DAGs of the learning environment can be classified correctly. In this section we give sufficient conditions for ensuring that a given set of examples can efficiently be loaded into a neural network. Let us introduce some more notation. For each $\mathcal{U}_l$, its vertexes are enumerated according to a chosen inverse topological order as $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_{P_l}$. Moreover, let $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_h$ be the set of vertexes, belonging to any DAG in the training set, for which a target value is defined. For simplicity, here we assume that only the supersources have a target defined (thus $h = L$). Given this notation:

(1) $\boldsymbol{U} \doteq [\boldsymbol{U}_1, \cdots, \boldsymbol{U}_L] \in R^{m+1, P^*}$ collects all the labels of the data structures (including the bias components always to 1), where $P^* \doteq \sum_{l=1}^{L} P_l$

(see Figure 1). Similarly, for each graph $\mathcal{U}_l \in U_L^{\#}$, and each $k \in [1, \ldots, o]$, the matrices $\boldsymbol{X}_l^{(k)} \in \mathbb{R}^{n,P_l}$ and defined as $\boldsymbol{X}_l^{(k)} \doteq [\boldsymbol{x}_{l,ch_k[\tilde{v}_1]}, \ldots, \boldsymbol{x}_{l,ch_k[\tilde{v}_{P_l}]}]$, where $\boldsymbol{x}_{l,ch_k[\tilde{v}_i]} = \boldsymbol{x}_0$ if $ch_k[\tilde{v}_i]$ is missing (we assume $\boldsymbol{x}_0$ to be the null vector), collect the status information for each pointer k. All the information concerning a pointer k is stored into matrices $\boldsymbol{X}^{(k)} \doteq \left[\boldsymbol{X}_1^{(k)}, \ldots, \boldsymbol{X}_L^{(k)}\right] \in \mathbb{R}^{n,P^*}$. Moreover, we define the matrix collecting the information needed to compute the output associated to the training graphs as (see Figure 2)

$$\boldsymbol{X}^{target} = \begin{bmatrix} \boldsymbol{x}_{\hat{v}_1} & \boldsymbol{x}_{\hat{v}_2} & \cdots & \boldsymbol{x}_{\hat{v}_L} \\ 1 & 1 & & 1 \end{bmatrix}.$$

(2) Let us define $\delta_{ilv} \doteq \partial E_l / \partial net_{ilv}$. For node v of a given DOAG $\mathcal{U}_l$, the corresponding delta error δ_{ilv} for the state variables can be collected in vector $\boldsymbol{\delta}_{lv} \doteq [\delta_{1lv}, \ldots, \delta_{nlv}]^T$ and the contributions from all the graph's nodes can be collected in matrix $\boldsymbol{\Delta}_l \doteq [\boldsymbol{\delta}_{l,1}, \ldots, \boldsymbol{\delta}_{l,P_l}] \in \mathbb{R}^{n,P_l}$, where the order of the columns follows the inverse topological order chosen for the graph. Finally, $\boldsymbol{\Delta} \doteq [\boldsymbol{\Delta}_1, \ldots, \boldsymbol{\Delta}_L] \in \mathbb{R}^{n,P^*}$ contains the delta errors for all the graphs of the learning environment, whereas the delta error corresponding to the output unit is denoted by $\delta_{ls}^{out} \doteq \frac{\partial E_l}{\partial net_{ls}}$ and collected into the matrix $\hat{\boldsymbol{\Delta}}$ (see Figure 3). Throughout the paper, the dependence on the network parameters will be denoted compactly as $\delta_{ls}^{out}(\boldsymbol{\omega})$.

The gradient of the cost can be calculated by using Backpropagation in each encoding network (see e.g. Figure 3), that is by propagating the error through the given structure, instead of through time, as typically occurs in recurrent networks processing sequences. The gradient of the cost can be written in a compact form by using the vectorial notation $\boldsymbol{G}_B \doteq \left[\frac{\partial E}{\partial b_{ij}}\right] \in \mathbb{R}^{m+1,n}$ and $\mathbf{G}_{A_k} \doteq \left[\frac{\partial E}{\partial a_{ijk}}\right] \in \mathbb{R}^{n,n}$. Based on these definitions, $\boldsymbol{G}_B$ and $\boldsymbol{G}_{A_k}$ can be computed as follows

$$\begin{aligned} \boldsymbol{G}_B &= \sum_{l=1}^{L} \boldsymbol{G}_{B,l} = \sum_{l=1}^{L} \sum_{v \in |(\mathcal{U}_l)} \boldsymbol{u}_{lv} \boldsymbol{\delta}_{lv}^T = \boldsymbol{U} \boldsymbol{\Delta}^T, \\ \boldsymbol{G}_{A_k} &= \sum_{l=1}^{L} \boldsymbol{G}_{A_k,l} = \sum_{l=1}^{L} \sum_{v \in |(\mathcal{U}_l)} \boldsymbol{x}_{l,ch_k[v]}\ \boldsymbol{\delta}_{lv}^T = \boldsymbol{X}^{(k)} \boldsymbol{\Delta}^T, \end{aligned} \tag{5}$$

where $(\cdot)^T$ is the transposition operator, $\mathbf{G}_{B,l},\ \mathbf{G}_{A_k,l}$ are the gradient contributions corresponding to E_l, that is to DOAG $\mathcal{U}_l$. Let $Q_k(v) \doteq \{u \mid ch_k[u] = v\}$. The *delta-error* δ_{ilv} can be computed recursively according to (see Figure 4)

$$\delta_{ilv} = \sigma'(net_{ilv}) \sum_{k=1}^{o} \sum_{j=1}^{n} a_{kji} \left(\sum_{z \in Q_k(v)} \delta_{jlz} \right) \tag{6}$$

where if $Q_k(v) = \emptyset$ then $\sum_{z \in Q_k(v)} \delta_{jlz} = 0$.

The above equation can be rewritten in compact form as

$$\boldsymbol{\delta}_{lv} = \boldsymbol{J}_{lv} \sum_{k=1}^{o} \boldsymbol{A}_k \left(\sum_{z \in Q_k(v)} \boldsymbol{\delta}_{lz} \right), \tag{7}$$

where $\boldsymbol{J}_{lv}$ is a diagonal matrix with elements $[\boldsymbol{J}_{lv}]_{ii} = \sigma'(net_{ilv})$. Moreover, by applying recursively equation (7), we obtain

$$\boldsymbol{\delta}_{lv} = \left(\sum_{p \in Paths_l(s,v)} \prod_{(u,ch_k[u]) \in p}^{(s \to v)_{left}} \boldsymbol{J}_{l,ch_k[u]} \boldsymbol{A}_k^T \right) \boldsymbol{\delta}_{ls} \tag{8}$$

where $Paths_l(s,v)$ is the set of paths in $\mathcal{U}_l$ from the supersource s to node v, and the product is left-hand starting from the supersource s and ending to node v.

This equation gives rise to the *Back-Propagation Through Structure* ($BPTS$) gradient computational scheme (Goller and Küchler, 1996; Sperduti and Starita, 1997). Without loss of generality, let us assume that $l \in \triangleright \mathcal{C}_+^{\#}$. The end of recursion is defined by means of the output backpropagation equations corresponding to the $\boldsymbol{C}$ matrix (which in our setting is a vector, i.e., $p = 1$), that is

$$\delta_{ls}^{out} = \sigma'(net_{ls}^{out}) \beta_+'(y_l - d^+), \tag{9}$$

$$\delta_{ils} = \sigma'(net_{ils}) c_i \delta_{ls}^{out}. \tag{10}$$

According to these formulas, the gradient for $\boldsymbol{C}$ is:

$$\boldsymbol{G}_C = \sum_{l=1}^{L} \boldsymbol{G}_{C,l} = \sum_{l=1}^{L} \begin{bmatrix} \boldsymbol{x}_{\hat{v}_l} \\ 1 \end{bmatrix} (\boldsymbol{\delta}_{ls}^{out})^T = \boldsymbol{X}^{target} \hat{\boldsymbol{\Delta}}^T \tag{11}$$

and the deltas can be written as

$$\boldsymbol{\delta}_{lv} = \left(\sum_{p \in Paths_l(s,v)} \prod_{(u,ch_k[u]) \in p}^{(s \to v)_{left}} \boldsymbol{J}_{l,ch_k[u]} \boldsymbol{A}_k^T \right) \boldsymbol{J}_{ls} (\delta_{ls}^{out} \boldsymbol{C}^T). \tag{12}$$

Lemma 4.1 *If* $\exists \boldsymbol{\omega} \in \Omega : \ \forall l, s$, *where we recall that* s *is the supersource of* $\mathcal{U}_l$, $\delta_{ls}^{out}(\boldsymbol{\omega}) = 0$ *then* $E(\boldsymbol{\omega}) = 0$.

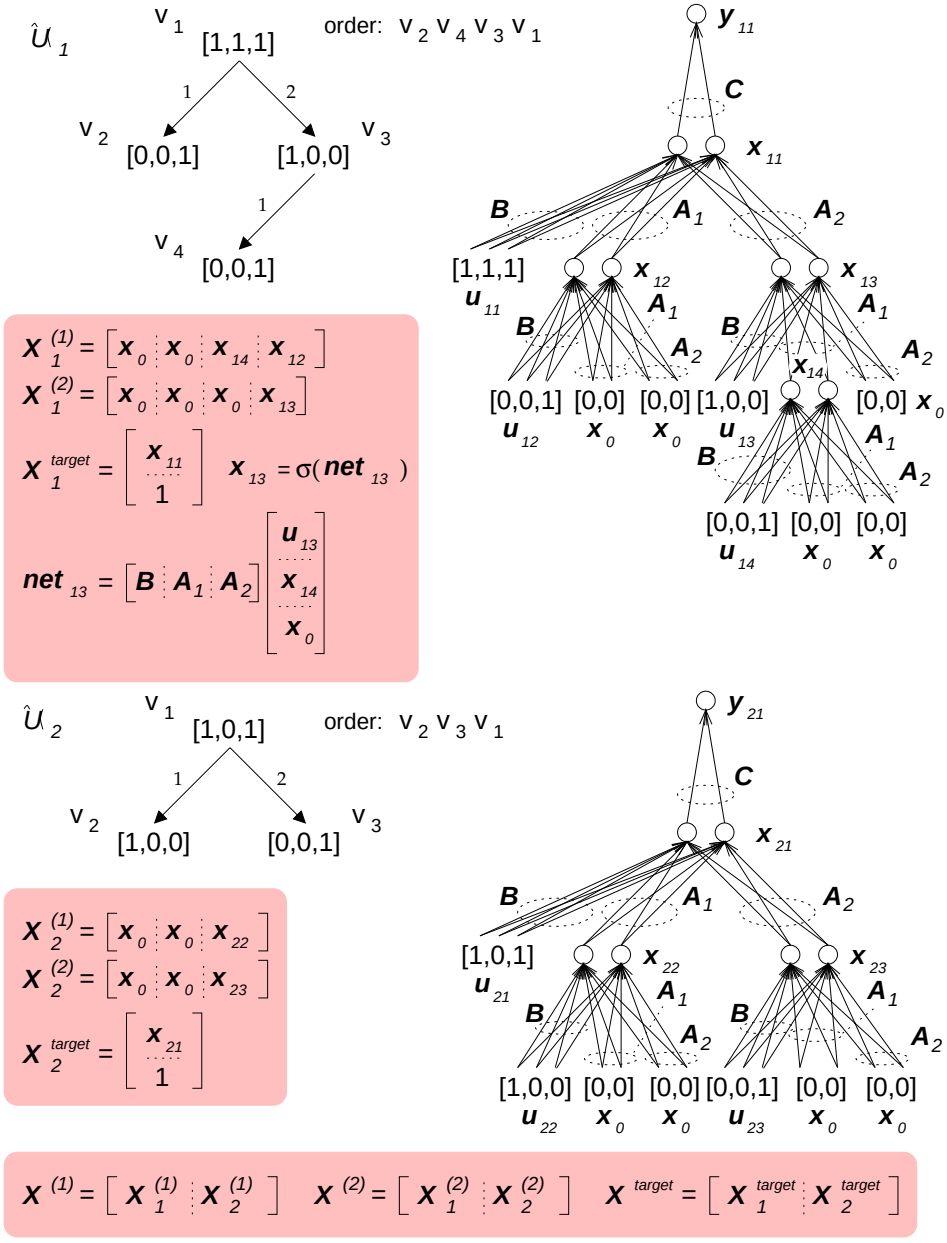

Fig. 2. Example of encoding networks. Two trees, $\mathcal{U}_1$ and $\mathcal{U}_2$ (see Figure 1), are considered. Their corresponding state vectors and state matrices are reported, as well as the corresponding encoding networks generated by using two state variables (i.e., two hidden units).

Proof: Without loss of generality, let us consider any $\mathcal{U}_l \in \mathcal{C}_+^{\#}$.
From the hypothesis we have $\exists \boldsymbol{\omega} \in \Omega : \ \forall l \in {\triangleright}\mathcal{C}_+^{\#}, s, \ \delta_{ls}^{out}(\boldsymbol{\omega}) = 0$. From (9),

$$\sigma'(net_{ls}^{out}(\boldsymbol{\omega}))\beta'_+(y_l(\boldsymbol{\omega}) - d^+) = 0.$$

Since $\sigma'(net_{ls}^{out}(\boldsymbol{\omega})) > 0$, $\beta'_+(y_l(\boldsymbol{\omega}) - d^+) = 0$ follows, which, in turn, implies $y_l(\boldsymbol{\omega}) \geq d^+$. This guarantees that $E_l(\boldsymbol{\omega}) = 0$. Since the property holds $\forall l \in {\triangleright}\mathcal{L}^{\#}$ we conclude that $E(\boldsymbol{\omega}) = 0$. $\square$

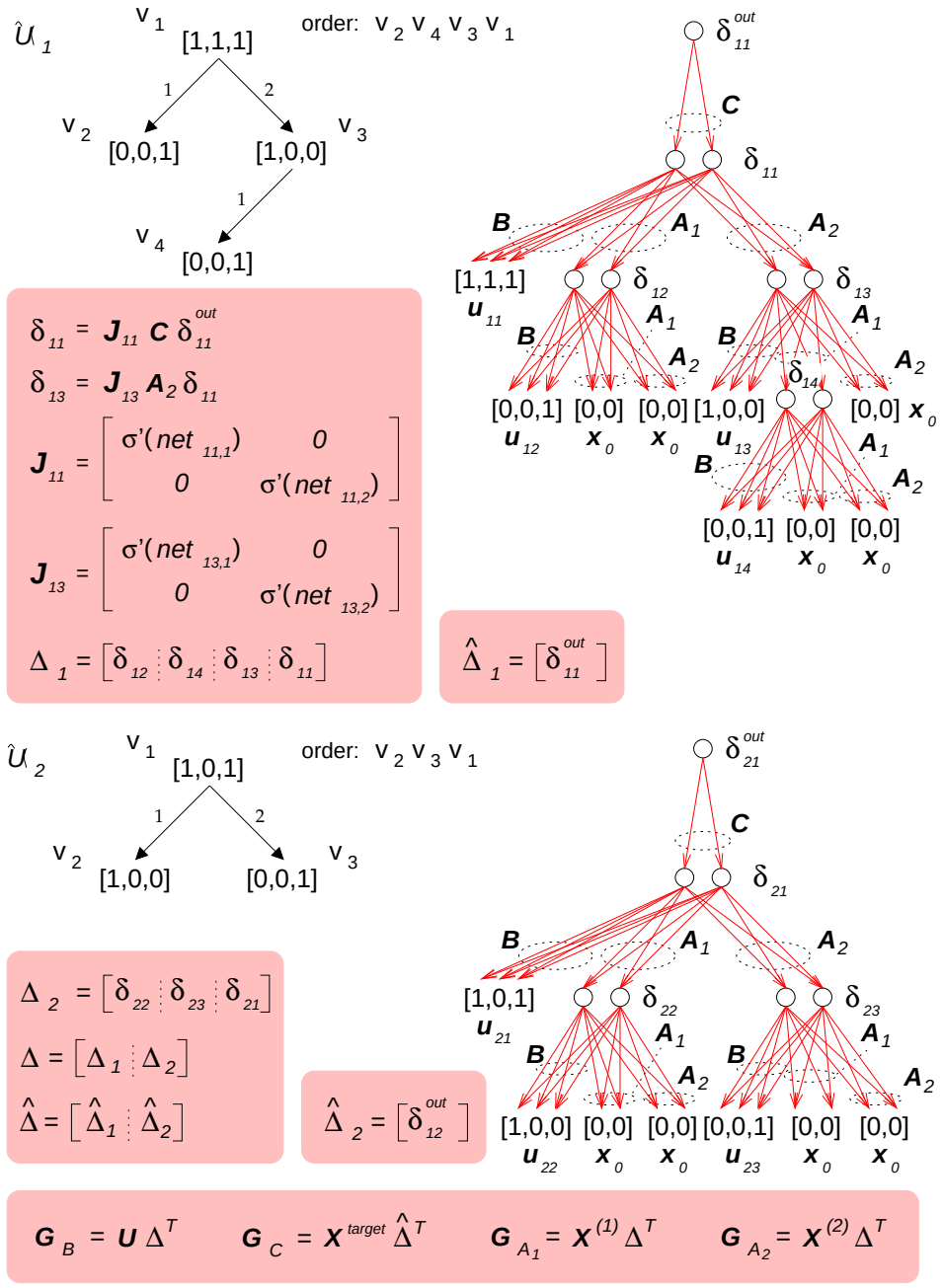

Fig. 3. The gradient vectors and matrices of two trees $\mathcal{U}_1$ and $\mathcal{U}_2$ (also shown in Figure 2) are reported. The corresponding flows of computation of the gradient, when computing it on the corresponding encoding networks reported in Figure 2, are reported as well.

Note that, viceversa, if $\exists \boldsymbol{\omega}: \ E(\boldsymbol{\omega}) = 0$ then $E_l(\boldsymbol{\omega}) = 0$, and we can promptly see that $\delta_{ls}^{out}(\boldsymbol{\omega}) = 0$. In fact, this follows from the definition of $\beta_+(\cdot)$, which yields $y_{ls}^{out}(\boldsymbol{\omega}) \geq d^+$. As a consequence $\beta'_+(y_l(\boldsymbol{\omega}) - d^+) = 0$ which, in turn, implies $\delta_{ls}^{out}(\boldsymbol{\omega}) = 0$.

Definition 4.1 *Given a recursive network $\mathcal{N}$, assume that the weight vector*

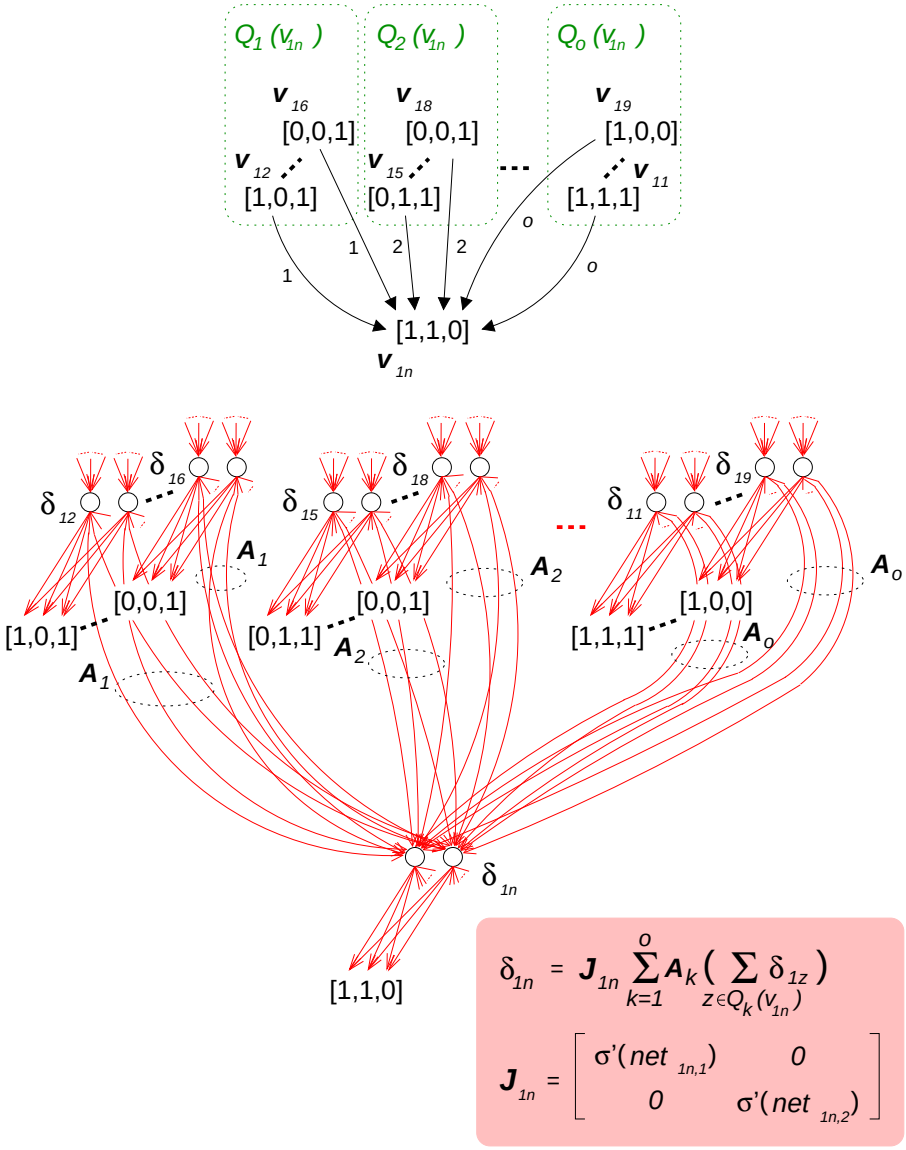

Fig. 4. Example of definition of the sets $Q_k(v)$, $k = 1, \ldots, o$. The corresponding flow of computation on a network with two hidden units is also reported.

$\boldsymbol{C}$ associated to the output unit has no-null component. Whenever this condition holds we say that $\mathcal{N}$ has no-null coding.

Lemma 4.2 *Let $\mathcal{N}$ be a no-null coding recursive network and, for any given $\mathcal{U}_l \in U_L^{\#}$, let us consider the associated delta errors δ_{ls}^{out} and $\boldsymbol{\delta}_{ls}$, corresponding to the output unit and to the supersource state, respectively. Let $\Omega_{ls}^{out} \doteq \{\boldsymbol{\omega} : \delta_{ls}^{out}(\boldsymbol{\omega}) = 0\}$ and for each hidden unit $i = 1, \ldots, n$ consider $\Omega_{ils} \doteq \{\boldsymbol{\omega} : \delta_{ils}(\boldsymbol{\omega}) = 0\}$. Then there exists at least one hidden unit i in the state layer such that $\Omega_{ls}^{out} = \Omega_{ils}$.*

Proof: We need to prove that $\Omega_{ils} \supseteq \Omega_{ls}^{out}$ and $\Omega_{ls}^{out} \subseteq \Omega_{ils}$ for at least one hidden unit i in the state layer.
$\Omega_{ils} \supseteq \Omega_{ls}^{out}$: From equation 10, it follows that if $\delta_{ls}^{out}(\boldsymbol{\omega}) = 0$ then $\forall i, \delta_{ils}(\boldsymbol{\omega}) = 0$
$\Omega_{ls}^{out} \subseteq \Omega_{ils}$: Assume that there exist i such that $\delta_{ils}(\boldsymbol{\omega}) = 0$, then from equation 10, and the fact that both $\sigma'(net_{ils}) > 0$ and $c_i \neq 0$, $\delta_{ls}^{out}(\boldsymbol{\omega}) = 0$ follows. □

Theorem 4.1 *(Perfect loading) Let us consider a no-null coding recursive network. If* $\exists i \in [1, \ldots, n]$ *and* $\exists \boldsymbol{\omega} \in \Omega$ *such that* $\delta_{ils}(\boldsymbol{\omega}) = 0$ *then* $E_l(\boldsymbol{\omega}) = 0$.

Proof: Straightforward consequence of Lemma 4.1 and Lemma 4.2 □

Of course, if $\exists \boldsymbol{\omega} \in \Omega$ such that $\forall \mathcal{U}_l \in U_L^{\#}, \exists i \in [1, \ldots, n] : \ \delta_{ils}(\boldsymbol{\omega}) = 0$ then we can conclude that $E(\boldsymbol{\omega}) = 0$. This theorem opens the doors to the investigation of the problem of perfect loading of the training set. The analysis of the next sections is based on the investigation of the critical points of $\boldsymbol{G}_B(\boldsymbol{\omega}) = \mathbf{0}$ and $\boldsymbol{G}_{A_k}(\boldsymbol{\omega}) = \mathbf{0}$. We shall prove that under proper conditions the presence of critical points yields $\delta_{ils} = 0$ and, in most cases, $\boldsymbol{\delta}_{ls} = \mathbf{0}$.

5 Graphs with categorical labels

When considering DOAG's with labels taking on categorical labels the training set can efficiently be represented by a reduction process generating a single graph (Goller, 1997; Sperduti and Starita, 1997). Here we refer to this graph as the *minimal DOAG*. The basic idea for creating the minimal DOAG is that of using all the "distinct" nodes of $|(U^{\#})$ only once. An example of how the minimal DOAG is produced from three DOAG's is given in Figure 5.

The reduction of graphs can formally be expressed by associating together nodes which are the supersource of the same graph. Basically, we observe that because of the computational model, which is causal and stationary, a node v is fully characterized by the label $\boldsymbol{u}_v$ and the children set ch$[v]$ associated to it. Specifically, the representation $\boldsymbol{x}_v$ of v is generated by just looking at $\boldsymbol{u}_v$ and $\boldsymbol{x}_{\mathrm{ch}_1[v]}$, .., $\boldsymbol{x}_{\mathrm{ch}_o[v]}$. When considering a leaf, the children set is void, and the representation $\boldsymbol{x}_0$ for the void state is used for any $k = 1, .., o$. Thus, the representation for a leaf only depends on the label attached to it. From this consideration it is not difficult to understand that graphs which are equal (with respect to the structure and labels attached to each node) will get the same representation. This fact can be formalized as follows

Definition 5.1 *Given a collection of DOAG's* $U^{\#}$*, let* $V(\#)$ *be the associated*

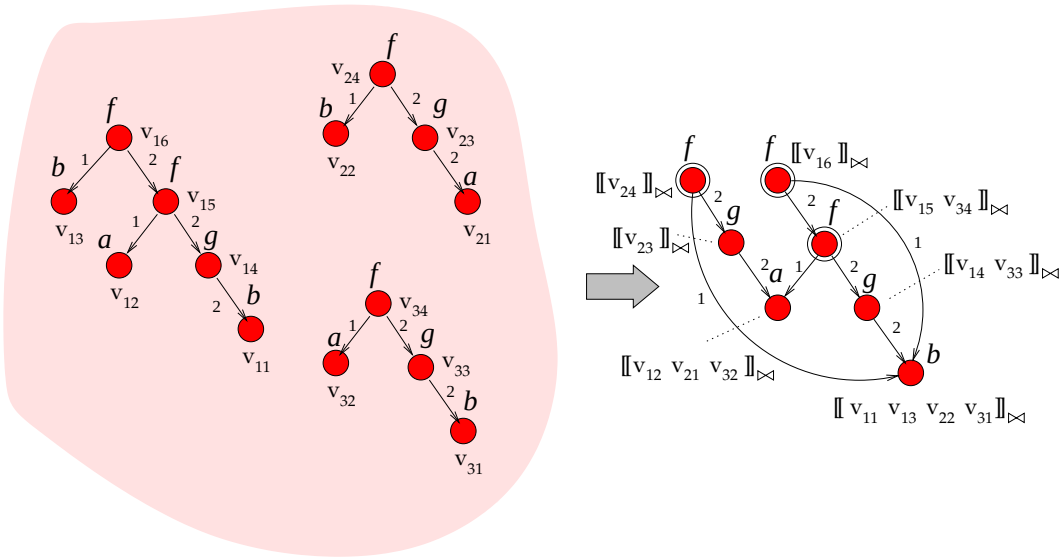

Fig. 5. Different DOAGs can be collapsed to one DOAG by using all the nodes of $|(U^{\#})$ only once. Supersource nodes in the original set of DOAGs are represented in the "collapsed" DOAGs as circled nodes. Each circled node represents a class of equivalence of the relation $\bowtie$.

set of nodes and define the relation $\bowtie \subset V(\#) \times V(\#)$ *as follows:*

$$v_{hl} \bowtie v_{km} \Longleftrightarrow DOAG(v_{hl}) = DOAG(v_{km}), \tag{13}$$

where $DOAG(v)$ *denotes the DOAG (in* $U^{\#}$*) having node* v *as a supersource.*

Definition 5.2 *Given a collection of DOAG's* $U^{\#}$*, consider the quotient set* $U^{\#}/_{\bowtie}$*. The number* $\Uparrow U^{\#} \doteq |\, U^{\#}/_{\bowtie} \,|$*, referred to as the* generation number *of* $U^{\#}$*, is the number of distinct state vectors generated by* $U^{\#}$*. The relation* $\bowtie$ *can be extended to the training set* $\mathcal{L}^{\#}$*, and, therefore, we can also consider the quotient set* $\mathcal{L}^{\#}/_{\bowtie}$ *and the corresponding generation number* $\Uparrow \mathcal{L}^{\#}$*.*

For instance, given the DOAG's of Figure 5, the corresponding quotient set $U^{\#}/_{\bowtie}$ turns out to be

$$\begin{aligned} U^{\#}/_{\bowtie} = \{ &[\![\, v_{12}, v_{21}, v_{32} \,]\!]_{\bowtie},\ [\![\, v_{11}, v_{13}, v_{22}, v_{31} \,]\!]_{\bowtie},\ [\![\, v_{14}, v_{33} \,]\!]_{\bowtie},\ [\![\, v_{23} \,]\!]_{\bowtie}, \\ &[\![\, v_{15}, v_{34} \,]\!]_{\bowtie},\ [\![\, v_{24} \,]\!]_{\bowtie},\ [\![\, v_{16} \,]\!]_{\bowtie} \}, \end{aligned} \tag{14}$$

and consequently, $\Uparrow U^{\#} = 7$, i.e., the number of nodes belonging to the minimal DOAG.

For any two nodes v_{hl}, v_{km}, $v_{hl} \bowtie v_{km}$ yields the same state in the representation level, that is if $v_{hl} \bowtie v_{km}$ then $\boldsymbol{x}_{hl} = \boldsymbol{x}_{km}$. Basically, we can attach an index ρ to the state representing all nodes in a class, that is $\boldsymbol{x}_{\rho}$ can be

regarded as the state which represents class $[\![\ \boldsymbol{x}_{l(\rho),v(\rho)}\ ]\!]_{\bowtie}$. Each equivalence class will correspond to a node of the minimal DOAG.

Relation $\bowtie$ can be specialized to $\bowtie_k$ when placing the restriction that nodes are associated only in the case in which they are $k-th$ children.

Definition 5.3 *Given a collection of DOAG's $U^{\#}$, let $U \doteq \{v_{lv}\}$ be the associated set of nodes and $U^{(k)} \subset U$ be the set of nodes v_{rl} such that $\exists v_{Rl} \in U$: $ch_k[v_{Rl}] = v_{rl}$. The relation $\bowtie_k$ is the restriction of relation $\bowtie$ to $U^{(k)} \times U^{(k)}$.*

For example, given the DOAG's of Figure 5, the corresponding quotient set $U^{\#}/_{\bowtie_1}$ and $U^{\#}/_{\bowtie_2}$ are

$$U^{\#}/_{\bowtie_1} = \{\ [\![\ v_{12}, v_{32}\ ]\!]_{\bowtie_1},\ [\![\ v_{13}, v_{22}\ ]\!]_{\bowtie_1}\ \}, \tag{15}$$

$$U^{\#}/_{\bowtie_2} = \{\ [\![\ v_{21}\ ]\!]_{\bowtie_2},\ [\![\ v_{11}, v_{31}\ ]\!]_{\bowtie_2},\ [\![\ v_{14}, v_{33}\ ]\!]_{\bowtie},\ [\![\ v_{23}\ ]\!]_{\bowtie},\ [\![\ v_{15}\ ]\!]_{\bowtie_2}\ \}. \tag{16}$$

Note that the cardinality of the quotient sets $U^{\#}/_{\bowtie_1}$ and $U^{\#}/_{\bowtie_2}$ can be readily computed on the minimal DOAG by counting how many distinct nodes are pointed by pointers of type 1 and pointers of type 2, respectively. In general,

Definition 5.4 *Given $U^{\#}$ with categorical variables, the number $\models U_k^{\#} \doteq |\ U^{\#}/_{\bowtie_k}\ |$, referred to as the* k-child generation number *of $U^{\#}$, is the number of distinct state vectors representing k-children generated by $U^{\#}$.*

In the example above, $\models U_1^{\#} = 2$ and $\models U_2^{\#} = 5$. Of course, $\models$ can be applied to $\mathcal{L}^{\#}$ with the obvious meaning.

Specifically, we are interested in the set of nodes obtained by applying $Q_k(\cdot)$ to all the members of the equivalence class $[\![\ \boldsymbol{x}_{ch_k[\rho]}\ ]\!]_{\bowtie_k}$, i.e., $Q_k([\![\ \boldsymbol{x}_{ch_k[\rho]}\ ]\!]_{\bowtie_k}) = \bigcup_{u \in [\![\ \boldsymbol{x}_{ch_k[\rho]}\ ]\!]_{\bowtie_k}} Q_k(u)$.

In fact, let us assume that $\boldsymbol{x}_{lv} = \boldsymbol{0}\ \ \forall lv \in F$ (frontier state). Consequently, the gradient equations (5) can be reduced to

$$\boldsymbol{G}_{A_k}(\omega) = \sum_{l=1}^{L} \boldsymbol{G}_{A_k,l}(\omega) \tag{17}$$

$$= \sum_{l=1}^{L} \sum_{v \in |(\mathcal{U}_l)} \boldsymbol{x}_{l,ch_k[v]}\ \boldsymbol{\delta}_{lv}^T \tag{18}$$

$$= \sum_{\rho=1}^{\models \mathcal{L}_k^{\#}} \boldsymbol{x}_{ch_k[\rho]} \sum_{u \in Q_k([\![\ \boldsymbol{x}_{ch_k[\rho]}\ ]\!]_{\bowtie_k})} \boldsymbol{\delta}_u^T. \tag{19}$$

If we define

$$\tilde{\boldsymbol{\delta}}_{\rho}^{(k)} \doteq \sum_{u \in Q_k([\![\, \boldsymbol{x}_{ch_k[\rho]} \,]\!]_{\bowtie_k})} \boldsymbol{\delta}_u^T$$

then

$$\boldsymbol{G}_{A_k}(\omega) = \sum_{\rho=1}^{\models \mathcal{L}_k^{\#}} \boldsymbol{x}_{ch_k[\rho]} \tilde{\boldsymbol{\delta}}_{\rho}^{(k)} \doteq \tilde{\boldsymbol{X}}^{(k)}(\omega) \tilde{\boldsymbol{\Delta}}_k^T(\omega) \tag{20}$$

where $\tilde{\boldsymbol{X}}^{(k)}(\omega) \in \mathbb{R}^{n, \models \mathcal{L}_k^{\#}}$ and $\tilde{\boldsymbol{\Delta}}_k(\omega) \in \mathbb{R}^{n, \models \mathcal{L}_k^{\#}}$.

A similar "restricted" equation can be derived for the gradient $\boldsymbol{G}_B(\omega)$. In fact, note first of all that nodes in $[\![\, \boldsymbol{x}_{ch_k[\rho]} \,]\!]_{\bowtie}$ necessarily share the same label. Moreover, distinct equivalence classes, may have the same associated label. So, we can merge these classes in a single class:

Definition 5.5 *Given a collection of DOAG's $U^{\#}$, the relation $\bowtie_B$ is obtained by relation $\bowtie$ by merging equivalence classes of $\bowtie$ whose members share the same label $\boldsymbol{u} \in U$.*

In the example above, we have

$$\begin{aligned} U^{\#}/_{\bowtie_B} = \{ & [\![\, v_{12}, v_{21}, v_{32} \,]\!]_{\bowtie_B}, \; [\![\, v_{11}, v_{13}, v_{22}, v_{31} \,]\!]_{\bowtie_B}, \; [\![\, v_{14}, v_{33}, v_{23} \,]\!]_{\bowtie_B}, \\ & [\![\, v_{15}, v_{34}, v_{24}, v_{16} \,]\!]_{\bowtie_B} \}, \end{aligned} \tag{21}$$

Using the equivalence classes generated by $\bowtie_B$, we can write

$$\boldsymbol{G}_B(\omega) = \sum_{l=1}^{L} \boldsymbol{G}_{B,l}(\omega) \tag{22}$$

$$= \sum_{l=1}^{L} \sum_{v \in |(\mathcal{U}_l)} \boldsymbol{u}_{l,v} \; \boldsymbol{\delta}_{lv}^T \tag{23}$$

$$= \sum_{\xi=1}^{|U|} \boldsymbol{u}_{\xi} \sum_{z \in [\![\, v_{\xi} \,]\!]_{\bowtie_B}} \boldsymbol{\delta}_z^T. \tag{24}$$

If we define

$$\tilde{\boldsymbol{\delta}}_{\xi} \doteq \sum_{z \in [\![\, v_{\xi} \,]\!]_{\bowtie_B}} \boldsymbol{\delta}_z^T$$

then

$$\boldsymbol{G}_B(\omega) = \sum_{\xi=1}^{|U|} \boldsymbol{u}_\xi \tilde{\boldsymbol{\delta}}_\xi \doteq \tilde{\boldsymbol{U}} \cdot \tilde{\boldsymbol{\Delta}}^T(\omega) \tag{25}$$

where $\tilde{\boldsymbol{U}} \in \mathbb{R}^{m+1,|U|}$ and $\tilde{\boldsymbol{\Delta}}(\boldsymbol{\omega}) \in \mathbb{R}^{n,|U|}$.

Let us now consider some interesting results involving the state matrices $\tilde{\boldsymbol{X}}^{(k)}(\omega)$.

Lemma 5.1 *If $n \geq \models \mathcal{L}_k^{\#}$ then matrix $\tilde{\boldsymbol{X}}^{(k)}(\omega)$, is full rank with probability 1.*

Proof: Under the hypothesis of the lemma $\tilde{\boldsymbol{X}}^{(k)}(\omega)$ is a matrix which is an analytic function of ω. $\square$

Lemma 5.2 *Given a loading problem $\Pi \sim \left\{\{[\boldsymbol{A}], \boldsymbol{B}, \boldsymbol{C}\}, \mathcal{L}^{\#}, E(\cdot)\right\}$ let us assume that for all critical points $\boldsymbol{\omega} \in \Omega$, $\forall l \in \triangleright\mathcal{L}^{\#}$ $\boldsymbol{\delta}_{ls}(\boldsymbol{\omega}) = \mathbf{0}$ holds with probability 1. Then Π is unimodal.*

Proof: First, let us consider any critical point $\boldsymbol{\omega} \in \Omega$ such that $\boldsymbol{\delta}_{ls}(\boldsymbol{\omega}) = \mathbf{0}$. From Theorem 4.1 we immediately conclude that Π is unimodal. Second, we prove that there exists no critical point $\hat{\boldsymbol{\omega}} \in \Omega : \quad \boldsymbol{\delta}_{ls}(\hat{\boldsymbol{\omega}}) \neq \mathbf{0}$. The proof is given by contradiction. Denote $\mathcal{S}(\hat{\boldsymbol{\omega}}, \rho)$ a ball with center in $\hat{\boldsymbol{\omega}}$ and radius ρ. Since $\boldsymbol{\delta}_{ls}(\hat{\boldsymbol{\omega}}) \neq \mathbf{0}$ holds with probability 1 then, regardless of $\rho > 0$ we can always find $\boldsymbol{\omega} \in \mathcal{S}(\hat{\boldsymbol{\omega}}, \rho)$ such that $\boldsymbol{\delta}_{ls}(\boldsymbol{\omega}) = \mathbf{0}$. From Theorem 4.1 we conclude that $E(\boldsymbol{\omega}) = 0$, whereas $E(\hat{\boldsymbol{\omega}}) \neq 0$. If $\rho \to 0$ then $\boldsymbol{\omega} \to \hat{\boldsymbol{\omega}}$, and we end up with the contradiction that $E(\cdot)$ is not a continuous function. $\square$

The above results suggest how (local) topological information can be exploited in order to estimate how many hidden units we need to guarantee absence of local minima. Besides to this information, we have another source of topological information that can be exploited to this purpose. In fact, if the delta variables corresponding to a vertex of a graph are null, then, under some conditions, the null values can be propagated to the delta variables of all its descendent vertexes. Such conditions are formally specified below, resorting to the definition of *forbidden vertexes* and *fully-dependent subgraph.*

Definition 5.6 *Given a sub-graph (with supersource) $\mathcal{Z}$ of $\mathcal{U}_l$, a vertex $z \in |(\mathcal{Z})$ is forbidden with respect to a set of equations $\mathcal{E}q$, if for some value of k, $\exists w \in Q_k(z), w \notin |(\mathcal{Z})$ s.t. $[\boldsymbol{\delta}_{lw} = 0] \notin \mathcal{E}q$.*

In the above definition, an equation is meant to be in the form $\boldsymbol{\delta}_{i_1 j_1} + \boldsymbol{\delta}_{i_2 j_2} + \ldots + \boldsymbol{\delta}_{i_t j_t} = \mathbf{0}$, $t \in \mathbb{N}^+$.

Definition 5.7 *Given a graph $\mathcal{U}_l$ and a vertex $u \in |(\mathcal{U}_l)$, the fully-dependent*

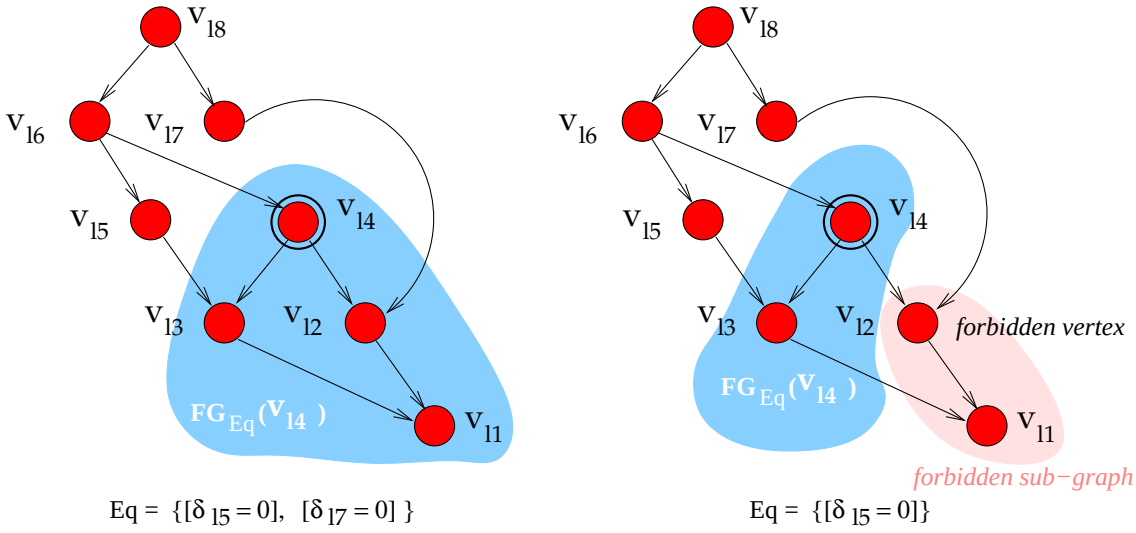

Fig. 6. Two examples of definition of the set $FG_{\mathcal{E}q}(u)$. In the example to the left side, the set of forbidden nodes is void, while in the example to the right side it is constituted by two nodes. Please, note that the second example differs from the first one only because the equation $[\delta_{l7} = 0]$ is not present in the set $\mathcal{E}q$.

subgraph of u with respect to a set of equations $\mathcal{E}q$, denoted $FG_{\mathcal{E}q}(u)$, is defined as the sub-graph of $\mathcal{U}$ rooted in u, $SG(u)$, where the vertexes belonging to sub-graphs of $\mathcal{U}$ rooted in forbidden vertexes of $SG(u)$ (and thus called forbidden sub-graphs) with respect to $\mathcal{E}q$ have been removed.

A couple of examples of $FG_{\mathcal{E}q}(\cdot)$ graphs are shown in Figure 6, were the graph $\mathcal{U}_l$ and vertex v_{l4} are kept fixed while varying the composition of the set of equations $\mathcal{E}q$.

Lemma 5.3 *Given a set of equations $\mathcal{E}q$, a graph $\mathcal{U}_l \in \mathcal{L}^{\#}$, and a vertex $u \in |(\mathcal{U}_l)$, if $[\boldsymbol{\delta}_{lu} = \mathbf{0}] \in \mathcal{E}q$ then $\forall v \in |(FG_{\mathcal{E}q}(u))$, $[\boldsymbol{\delta}_{lv} = \mathbf{0}]$.*

Proof: By definition, vertexes belonging to $FG_{\mathcal{E}q}(u)$ are such that eq. (8) can be applied by substituting $\boldsymbol{\delta}_{lu}$ to $\boldsymbol{\delta}_{ls}$. This is possible since for any $v \in |(FG_{\mathcal{E}q}(u))$, and for any vertex in a path from u to v within $FG_{\mathcal{E}q}(u)$, there is no parent $p \notin FG_{\mathcal{E}q}(u)$ of the considered vertex for which $[\boldsymbol{\delta}_{lp} \neq \mathbf{0}]$. Thus, since $[\boldsymbol{\delta}_{lu} = \mathbf{0}]$, this implies that $[\boldsymbol{\delta}_{lv} = \mathbf{0}]$. □

The above lemma justifies the definition of the following function.

Definition 5.8 *Given a set of equations $\mathcal{E}q$, the function*

$$\tau_{\mathcal{E}q} : EQ \rightarrow 2^{EQ}$$

is defined as the function that, given as input an equation in the form $[\boldsymbol{\delta}_{lu} = \mathbf{0}]$, for some vertex $u \in |(\mathcal{U}_l)$, returns the set of equations $\{[\boldsymbol{\delta}_{lv} = \mathbf{0}] | v \in |(FG_{\mathcal{E}q}(u))\}$.

Notice that, in the worst case $\{[\boldsymbol{\delta}_{lv} = \mathbf{0}] | v \in |(FG_{\mathcal{E}q}(u))\} \equiv \{[\boldsymbol{\delta}_{lu} = \mathbf{0}]\}$, i.e., $FG_{\mathcal{E}q}(u)$ is just composed of u.

Another source of information is coming from the target associated to every graph. Below, we formalize in which case target information can be fruitfully exploited.

Definition 5.9 *An equation in the form* $[\boldsymbol{\delta}_{l_{j_1}s} + \ldots + \boldsymbol{\delta}_{l_{j_d}s} = \mathbf{0}]$, $d > 0$, *is said to be target-pure if all the terms involved in the sum:*

i) refer to supersources;
ii) have an associated target;
iii) all the targets are the same (either d^- *or* d^+*).*

Given the above definition, it is not difficult to prove the following lemma.

Lemma 5.4 *Given an equation in the form* $[\boldsymbol{\delta}_{l_{j_1}s} + \ldots + \boldsymbol{\delta}_{l_{j_d}s} = \mathbf{0}]$, $d > 0$, *if it is target-pure then for* $i = 1, \ldots, d$, $[\boldsymbol{\delta}_{l_{j_i}s} = \mathbf{0}]$.

Proof: Since all the targets of the involved vertexes are the same, and because of eqs. (9) and (10), the deltas have all the same sign. Thus, in order for the equation to be satisfied, all its terms must be 0. □

It is thus useful to define the function

Definition 5.10 *The function*

$$\lambda : EQ \rightarrow 2^{EQ}$$

is defined as the function that, given as input an equation in the form $[\boldsymbol{\delta}_{l_{j_1}s} + \ldots + \boldsymbol{\delta}_{l_{j_d}s} = \mathbf{0}]$, $d > 0$, *returns the input equation itself if the equation is not target-pure, otherwise it returns the set of equations* $\{[\boldsymbol{\delta}_{l_{j_i}s} = \mathbf{0}] | i = 1, \ldots, d\}$.

In the following, we will also need a function that, given two set of equations, $\mathcal{E}q_1$ and $\mathcal{E}q_2$, involving delta variables, returns a single simplified joint set of equations, i.e., a set of equations that is equivalent to the set $\mathcal{E}q_1 \cup \mathcal{E}q_2$ and contains the minimum number of equations, each involving the minimum possible number of delta variables. For example, the two sets $\mathcal{E}q_1 = \{\boldsymbol{\delta}_{14} + \boldsymbol{\delta}_{23} = \mathbf{0}\}$ and $\mathcal{E}q_2 = \{\boldsymbol{\delta}_{23} = \mathbf{0}\}$, can be simplified into the set $\mathcal{E}q_{1\cup 2} = \{\boldsymbol{\delta}_{14} = \mathbf{0}, \boldsymbol{\delta}_{23} = \mathbf{0}\}$.

Definition 5.11 *The function*

$$\rho : 2^{EQ} \times 2^{EQ} \rightarrow 2^{EQ}$$

is defined as the function that, given as input two sets of equations returns a simplified joint set of equations.

The implementation of this function is not of particular interest in this context, however efficient algorithms, based on a lexicographic ordering on the variables and equations, are not difficult to define.

We are now ready to define a function that, given a set of equations involving delta variables, updates this set by a new set of equations (that, in our case, are derived by one of the delta matrices $\tilde{\mathbf{\Delta}}_k^T$). This update, besides to merge and simplify the two set of equations, must also consider the topological relationships among delta variables and the information coming from the targets in order to further simplify the resulting set of equations. Thus, the desired function

$$\texttt{update} : 2^{EQ} \times 2^{EQ} \rightarrow 2^{EQ}$$

can be obtained as composition of the already defined functions $\tau_{\mathcal{E}q}(\cdot)$, $\lambda(\cdot)$, and $\rho(\cdot)$:

Algorithm (`update`)

Input:
Two set of equations $\mathcal{E}q_1$ and $\mathcal{E}q_2$
Output:
A new set of equations $\mathcal{E}q$
`begin`
$\mathcal{E}q \leftarrow \rho(\mathcal{E}q_1, \mathcal{E}q_2)$;
while (there is a change in $\mathcal{E}q$)
`begin`
$\forall[\delta_u = 0] \in \mathcal{E}q,\ \mathcal{E}q \leftarrow \mathcal{E}q \cup \tau_{\mathcal{E}q}([\delta_u = 0])$; /* use topology info */
$\forall eq \in \mathcal{E}q,\ \mathcal{E}q \leftarrow \mathcal{E}q \cup \lambda(eq)$; /* use target info */
$\mathcal{E}q \leftarrow \rho(\mathcal{E}q, \emptyset)$; /* simplify $\mathcal{E}q$ */
`end`
`return` $\mathcal{E}q$;
`end`

Notice that the introduction of new equations due to $\tau_{\mathcal{E}q}(\cdot)$ and/or $\lambda(\cdot)$ requires to check for the possibility to further simplify the set of equations (this is done by applying $\rho(\cdot)$), and then to repeat the attempt to exploit topological and target information to introduce new equations from the resulting set of equations (done by the `while` statement).

As said above, the set of equations used to update the current set of equations is derived from one of the delta matrices $\tilde{\mathbf{\Delta}}_k^T$. Specifically, given a value for k, we assume that the number of hidden units, i.e. n, is such to render the rank of $\tilde{\boldsymbol{X}}^{(k)}$ full, so that the system $\tilde{\boldsymbol{X}}^{(k)}\tilde{\mathbf{\Delta}}_k^T = \mathbf{0}$ has a unique (null) solution. Of course, we are interested in the minimum value of n for which $\tilde{\boldsymbol{X}}^{(k)}$ has full rank. Finally, before considering the system $\tilde{\boldsymbol{X}}^{(k)}\tilde{\mathbf{\Delta}}_k^T = \mathbf{0}$, we have to check

whether the current set of equations already implies that some rows of $\tilde{\boldsymbol{\Delta}}_k^T$ are null. If this is the case, we can discard the null variables and consider a reduced $\tilde{\boldsymbol{X}}^{(k)}$. This is done by the following function.

Definition 5.12 *The function*

$$reduce : Matrix^* \times 2^{EQ} \rightarrow Matrix^*$$

where $Matrix^*$ is the set of matrices of all dimensions, is defined as the function that, given as input a delta matrix $\tilde{\boldsymbol{\Delta}}_k^T$, for some k, and a set of equations $\mathcal{E}q$, returns a reduced delta matrix $\tilde{\boldsymbol{\Delta}}'_k$ where any row $\tilde{\boldsymbol{\delta}}_\rho^{(k)^T} \in \tilde{\boldsymbol{\Delta}}_k^T$ for which $\mathcal{E}q \Rightarrow [\tilde{\boldsymbol{\delta}}_\rho^{(k)^T} = \mathbf{0}]$ is removed.

Notice that the above definition implies that either the input matrix $\tilde{\boldsymbol{\Delta}}_k^T \in \mathbb{R}^{n,\models\mathcal{L}_k^\#}$ stays the same if there is no value of ρ for which $\mathcal{E}q \Rightarrow [\tilde{\boldsymbol{\delta}}_\rho^{(k)} = \mathbf{0}]$, or $\tilde{\boldsymbol{\Delta}}'_k \in \mathbb{R}^{n,\models\mathcal{L}_k^\# - p}$, where $p = |\{\rho | 1 \leq \rho \leq \models \mathcal{L}_k^\#, \mathcal{E}q \Rightarrow [\tilde{\boldsymbol{\delta}}_\rho^{(k)} = \mathbf{0}]\}|$.

Let now assume that $p > 0$, and let denote with $\tilde{\boldsymbol{X}}'^{(k)}$ the k-pointer matrix obtained by $\tilde{\boldsymbol{X}}^{(k)}$ by removing the columns with the same ρ indexes of rows removed from $\tilde{\boldsymbol{\Delta}}_k$. Because of that, the set of solutions for the delta variables is not going to change if we consider the reduced system $\tilde{\boldsymbol{X}}'^{(k)}\tilde{\boldsymbol{\Delta}}'^T_k = \mathbf{0}$ instead of the full system $\tilde{\boldsymbol{X}}^{(k)}\tilde{\boldsymbol{\Delta}}_k^T = \mathbf{0}$. However, in order to have a unique (null) solution, the reduced system $\tilde{\boldsymbol{X}}'^{(k)}\tilde{\boldsymbol{\Delta}}'^T_k = \mathbf{0}$ just requires the matrix $\tilde{\boldsymbol{X}}'^{(k)}$ to have full rank. So, in consideration of the fact that also $\tilde{\boldsymbol{X}}'^{(k)}$ is an analytic function of ω, it is enough to have $n \geq (\mathcal{L}_k^\# - p)$ to guarantee that $\tilde{\boldsymbol{X}}'^{(k)}$ is full rank with probability 1. This fact will be used in the following.

In order to compute an "upper bound" on the number of hidden units, we also need a function that, given a matrix, returns the corresponding set of equations.

Definition 5.13 *The function*

$$eqs : Matrix^* \rightarrow 2^{EQ}$$

is defined as the function that, given as input a delta matrix $\tilde{\boldsymbol{\Delta}}_k^T$, for some k, returns the set of equations $\mathcal{E}q$ defined by the system $\tilde{\boldsymbol{\Delta}}_k^T = \mathbf{0}$.

We are now ready to build the "upper bound" by searching for the optimal sequence of "pointer" updates starting from the initial set of equations derived by the label matrix. In order to do that we need to introduce some ancillary concepts.

Let $\pi \equiv (p_1, \ldots, p_o)$ be a permutation of integers from 1 to o (i.e., all the possible "pointer" updates). Given a training set $\mathcal{L}^{\#}$, we define two related sequences of sets of equations $Eq_i,\ i = 0, \ldots, o$, and matrices $M_i,\ i = 1, \ldots, o$:

$$
\begin{array}{ll}
 & Eq_0 = \texttt{update}(eqs(\tilde{\boldsymbol{\Delta}}^T), \emptyset) \\
M_1 = reduce(\tilde{\boldsymbol{\Delta}}^T_{p_1}, Eq_0) & Eq_1 = \texttt{update}(Eq_0, eqs(M_1)) \\
M_2 = reduce(\tilde{\boldsymbol{\Delta}}^T_{p_2}, Eq_1) & Eq_2 = \texttt{update}(Eq_1, eqs(M_2)) \\
\ldots & \ldots \\
M_i = reduce(\tilde{\boldsymbol{\Delta}}^T_{p_i}, Eq_{i-1}) & Eq_i = \texttt{update}(Eq_{i-1}, eqs(M_i)) \\
\ldots & \ldots \\
M_o = reduce(\tilde{\boldsymbol{\Delta}}^T_{p_o}, Eq_{o-1}) & Eq_o = \texttt{update}(Eq_{o-1}, eqs(M_o))
\end{array}
$$

Finally, we can define the "upper bound".

Definition 5.14 *Given a training set $\mathcal{L}^{\#}$, the* reduction pointer $\models^{red}$ *of $\mathcal{L}^{\#}$ is defined as*

$$\models^{red} \mathcal{L}^{\#} \doteq \min_{\pi \in \mathcal{P}} \max_{p_i \in \pi} \{\texttt{num\_row}(M_i)\}, \tag{26}$$

where $\mathcal{P}$ is the set of permutations of integers from 1 to o, and $\texttt{num\_row}(M_i)$ *is the number of rows of M_i.*

We now state under which conditions the *reduction pointer* is a valid "upper bound". Given a training set $\mathcal{L}^{\#}$, let S be the set of supersource vertexes of graphs in $\mathcal{L}^{\#}$.

Theorem 5.1 *Given $\Pi \sim \left\{\{[\boldsymbol{A}], \boldsymbol{B}, \boldsymbol{C}\}, \mathcal{L}^{\#}, E(\cdot)\right\}$, assume that $\mathcal{L}^{\#}$ is composed of graphs with labels having categorical values. If there exists at least one point $\boldsymbol{\omega} \in \Omega$ such that $E(\boldsymbol{\omega}) = 0$ then, function $E(\cdot)$ has no local minima different from $E = 0$, provided that for each $s \in S$, $[\delta_s = 0] \in Eq_o$ and $n \geq \models^{red} \mathcal{L}^{\#}$.*

Proof: First of all, notice that if for each $s \in S$, $[\delta_s = 0] \in Eq_o$, then all the deltas are 0. This is due to the application of eq. 8 and the fact that every vertex in the training set is reachable starting from the supersource of the graph it belongs to.

Let now consider $\pi^* \equiv (p_1^*, \ldots, p_o^*) = \arg\min_{\pi \in \mathcal{P}} \max_{p_i \in \pi} \{\texttt{num\_row}(M_i)\}$. π^* may not be unique, so let just choose one of the optimal permutations, i.e., one of the permutations for which the maximum dimension of matrices M_i is minimized. Given π^* let now consider $n^* = \max_{p_i \in \pi^*} \{\texttt{num\_row}(M_i)\}$. By definition n^* will thus correspond to the maximum rank matrices $\tilde{\boldsymbol{X}}^{(k)}$ need to have such that $\forall s \in S,\ [\delta_s = 0] \in Eq_o$ with probability 1. In fact, the way

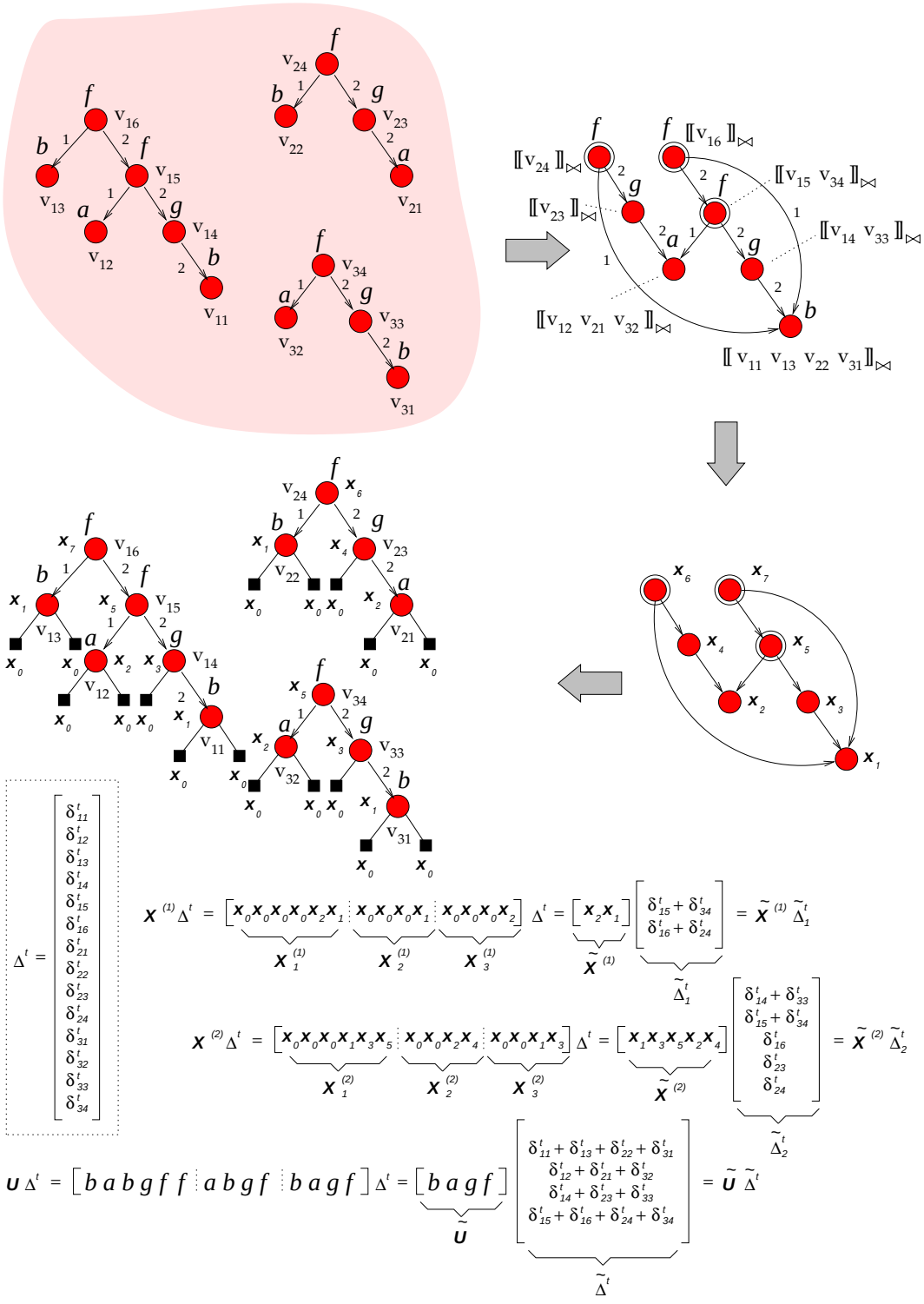

Fig. 7. Example of definition of label, state, and gradient matrices starting from a dataset with three structures. The equivalence classes of $\bowtie$ are first computed and then used to define the distinct vector states $\boldsymbol{x}_i$, $i = 1, \ldots, 7$, which in turn are used to define the relevant matrices.

we have generated matrices $M_0, \ldots, M_o$ implies that matrices $\tilde{\boldsymbol{X}}'^{(1)}, \ldots, \tilde{\boldsymbol{X}}'^{(o)}$ will be full rank ($\leq n^*$) with probability 1. Since $\forall s \in S,\ [\delta_s = 0] \in Eq_o$ with probability 1, then by using Lemma 5.2 the thesis is demonstrated. $\square$

5.1 An example of application

In order to gain a better understanding of the application of the proposed concepts and procedures let us consider the three structures (binary trees) shown at the top left of Figure 7. In this case $L = 3$ and $o = 2$. Moreover, let assume that the trees with root v_{16} and v_{24} have associated target d^+, while the tree with root v_{34} has target d^-.

We already saw (see Figure 5) that the generator number of this set of structures is 7. In fact, at the top right of Figure 7 we have reported the resulting quotient set, where all the 7 equivalence classes are shown. Since each equivalence class will generate a corresponding state vector, we can introduce 7 different state vectors, namely, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_7$ (see middle right of Figure 7). These state vectors can then be associated to the corresponding vertexes in the input structures (see middle left of Figure 7) and this information used to build up the matrices $\boldsymbol{\Delta}^T$, $\tilde{\boldsymbol{\Delta}}^T$, $\tilde{\boldsymbol{\Delta}}_1^T$, $\tilde{\boldsymbol{\Delta}}_2^T$, $\boldsymbol{X}^{(1)} \equiv [\boldsymbol{X}_1^{(1)},\ \boldsymbol{X}_2^{(1)},\ \boldsymbol{X}_3^{(1)}]$, $\boldsymbol{X}^{(2)} \equiv [\boldsymbol{X}_1^{(2)},\ \boldsymbol{X}_2^{(2)},\ \boldsymbol{X}_3^{(2)}]$, $\tilde{\boldsymbol{X}}^{(1)}$, $\tilde{\boldsymbol{X}}^{(2)}$, $\tilde{\boldsymbol{U}}$. Specifically, we have

$$\tilde{\boldsymbol{\Delta}}^T = \begin{bmatrix} \boldsymbol{\delta}_{11}^T + \boldsymbol{\delta}_{13}^T + \boldsymbol{\delta}_{22}^T + \boldsymbol{\delta}_{31}^T \\ \boldsymbol{\delta}_{12}^T + \boldsymbol{\delta}_{21}^T + \boldsymbol{\delta}_{32}^T \\ \boldsymbol{\delta}_{14}^T + \boldsymbol{\delta}_{23}^T + \boldsymbol{\delta}_{33}^T \\ \boldsymbol{\delta}_{15}^T + \boldsymbol{\delta}_{16}^T + \boldsymbol{\delta}_{24}^T + \boldsymbol{\delta}_{34}^T \end{bmatrix},$$

and

$$\begin{aligned} Eq_0 &= \texttt{update}(eqs(\tilde{\boldsymbol{\Delta}}^T), \emptyset) \\ &= eqs(\tilde{\boldsymbol{\Delta}}^T) \\ &= \Big\{ \boldsymbol{\delta}_{11}^T + \boldsymbol{\delta}_{13}^T + \boldsymbol{\delta}_{22}^T + \boldsymbol{\delta}_{31}^T = \boldsymbol{0}, \\ &\qquad \boldsymbol{\delta}_{12}^T + \boldsymbol{\delta}_{21}^T + \boldsymbol{\delta}_{32}^T = \boldsymbol{0}, \\ &\qquad \boldsymbol{\delta}_{14}^T + \boldsymbol{\delta}_{23}^T + \boldsymbol{\delta}_{33}^T = \boldsymbol{0}, \\ &\qquad \boldsymbol{\delta}_{15}^T + \boldsymbol{\delta}_{16}^T + \boldsymbol{\delta}_{24}^T + \boldsymbol{\delta}_{34}^T = \boldsymbol{0} \Big\}. \end{aligned}$$

Note that, $\texttt{update}(eqs(\tilde{\boldsymbol{\Delta}}^T), \emptyset) = eqs(\tilde{\boldsymbol{\Delta}}^T)$ since: $eqs(\tilde{\boldsymbol{\Delta}}^T)$ is already simplified; no topology information can be used since there is no equation in the

form $[\boldsymbol{\delta}^T_{ij} = \mathbf{0}]$; no equation is target-pure.

Since we have $o = 2$, there are only two possible permutations to consider, namely, $\pi_a \equiv (1, 2)$ and $\pi_b \equiv (2, 1)$. Let start by considering permutation π_a, i.e., $p_1 = 1$ and $p_2 = 2$. In this case we have,

$$\tilde{\boldsymbol{\Delta}}^T_1 = \begin{bmatrix} \boldsymbol{\delta}^T_{15} + \boldsymbol{\delta}^T_{34} \\ \boldsymbol{\delta}^T_{16} + \boldsymbol{\delta}^T_{24} \end{bmatrix},$$

and

$$M_1 = reduce(\tilde{\boldsymbol{\Delta}}^T_1, Eq_0) = \tilde{\boldsymbol{\Delta}}^T_1,$$

since there is no equation in the form $[\boldsymbol{\delta}^T_{ij} = \mathbf{0}]$ in Eq_0. Consequently, we have

$$eqs(M_1) = \left\{ \begin{array}{l} \boldsymbol{\delta}^T_{15} + \boldsymbol{\delta}^T_{34} = \mathbf{0}, \\ \boldsymbol{\delta}^T_{16} + \boldsymbol{\delta}^T_{24} = \mathbf{0} \end{array} \right\}.$$

Let us now compute

$$Eq_1 = \texttt{update}(Eq_0, eqs(M_1)).$$

This time, the application of the `update` algorithm produces a non trivial result. In fact, first of all it simplifies the two set of equations in input, i.e. $Eq \leftarrow \rho(Eq_0, eqs(M_1))$, where

$$Eq = \left\{ \begin{array}{l} \boldsymbol{\delta}^T_{11} + \boldsymbol{\delta}^T_{13} + \boldsymbol{\delta}^T_{22} + \boldsymbol{\delta}^T_{31} = \mathbf{0}, \\ \boldsymbol{\delta}^T_{12} + \boldsymbol{\delta}^T_{21} + \boldsymbol{\delta}^T_{32} = \mathbf{0}, \\ \boldsymbol{\delta}^T_{14} + \boldsymbol{\delta}^T_{23} + \boldsymbol{\delta}^T_{33} = \mathbf{0}, \\ \boldsymbol{\delta}^T_{15} + \boldsymbol{\delta}^T_{34} = \mathbf{0}, \\ \boldsymbol{\delta}^T_{16} + \boldsymbol{\delta}^T_{24} = \mathbf{0} \end{array} \right\}.$$

Then, it enters the body of the `while` statement, skipping the application of function τ_{Eq} since there is no equation in Eq in the form $[\boldsymbol{\delta}^T_{ij} = \mathbf{0}]$. However, since equation $[\boldsymbol{\delta}^T_{16} + \boldsymbol{\delta}^T_{24} = \mathbf{0}]$ is target-pure (both deltas have associated the same d^+ target), it computes

$$\lambda([\boldsymbol{\delta}^T_{16} + \boldsymbol{\delta}^T_{24} = \mathbf{0}]) = \left\{ \boldsymbol{\delta}^T_{16} = \mathbf{0}, \boldsymbol{\delta}^T_{24} = \mathbf{0} \right\}$$

So, after using the target information and simplifying the resulting set of equations, it obtains

$$\begin{aligned} Eq = \Big\{ & \boldsymbol{\delta}_{11}^T + \boldsymbol{\delta}_{13}^T + \boldsymbol{\delta}_{22}^T + \boldsymbol{\delta}_{31}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{12}^T + \boldsymbol{\delta}_{21}^T + \boldsymbol{\delta}_{32}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{14}^T + \boldsymbol{\delta}_{23}^T + \boldsymbol{\delta}_{33}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{15}^T + \boldsymbol{\delta}_{34}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{16}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{24}^T = \mathbf{0} \Big\} \end{aligned}$$

Since Eq has been modified, it cannot exit from the `while` statement, and has to execute again its body. This time topology information can be exploited. In fact, we have

$$\tau_{Eq}([\boldsymbol{\delta}_{16}^T = \mathbf{0}]) = \left\{ \boldsymbol{\delta}_{16}^T = \mathbf{0}, \boldsymbol{\delta}_{13}^T = \mathbf{0}, \boldsymbol{\delta}_{15}^T = \mathbf{0}, \boldsymbol{\delta}_{12}^T = \mathbf{0}, \boldsymbol{\delta}_{14}^T = \mathbf{0}, \boldsymbol{\delta}_{11}^T = \mathbf{0} \right\},$$

and

$$\tau_{Eq}([\boldsymbol{\delta}_{24}^T = \mathbf{0}]) = \left\{ \boldsymbol{\delta}_{24}^T = \mathbf{0}, \boldsymbol{\delta}_{22}^T = \mathbf{0}, \boldsymbol{\delta}_{23}^T = \mathbf{0}, \boldsymbol{\delta}_{21}^T = \mathbf{0} \right\}.$$

The resulting set of equations $Eq \cup \tau_{Eq}([\boldsymbol{\delta}_{16}^T = \mathbf{0}]) \cup \tau_{Eq}([\boldsymbol{\delta}_{16}^T = \mathbf{0}])$ is not modified by the $\lambda(\cdot)$ function and its simplification leads to

$$\begin{aligned} Eq = \Big\{ & \boldsymbol{\delta}_{16}^T = \mathbf{0}, \boldsymbol{\delta}_{13}^T = \mathbf{0}, \boldsymbol{\delta}_{15}^T = \mathbf{0}, \boldsymbol{\delta}_{12}^T = \mathbf{0}, \boldsymbol{\delta}_{14}^T = \mathbf{0}, \boldsymbol{\delta}_{11}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{24}^T = \mathbf{0}, \boldsymbol{\delta}_{22}^T = \mathbf{0}, \boldsymbol{\delta}_{23}^T = \mathbf{0}, \boldsymbol{\delta}_{21}^T = \mathbf{0}, \\ & \boldsymbol{\delta}_{34}^T = \mathbf{0}, \boldsymbol{\delta}_{32}^T = \mathbf{0}, \boldsymbol{\delta}_{33}^T = \mathbf{0}, \boldsymbol{\delta}_{31}^T = \mathbf{0} \Big\}. \end{aligned}$$

Of course, this set of equations cannot be further modified, so eventually the execution can exit from the body of the `while` statement and the algorithm returns $Eq_1 = Eq$. Then, trivially $M_2 = reduce(\tilde{\boldsymbol{\Delta}}_2^T, Eq_1) = \emptyset$, i.e., the empty matrix. Thus

$$\max\{\texttt{num\_row}(M_1), \texttt{num\_row}(M_2)\} = \max\{2, 0\} = 2.$$

Let now consider π_b, i.e., $p_1 = 2$ and $p_2 = 1$. In this case we have,

$$\tilde{\mathbf{\Delta}}_2^T = \begin{bmatrix} \boldsymbol{\delta}_{14}^T + \boldsymbol{\delta}_{33}^T \\ \boldsymbol{\delta}_{15}^T + \boldsymbol{\delta}_{34}^T \\ \boldsymbol{\delta}_{16}^T \\ \boldsymbol{\delta}_{23}^T \\ \boldsymbol{\delta}_{24}^T \end{bmatrix},$$

and

$$M_1 = reduce(\tilde{\mathbf{\Delta}}_2^T, Eq_0) = \tilde{\mathbf{\Delta}}_2^T.$$

Since $\texttt{num\_row}(M_1) = 5$, it is soon clear that we will have

$$\max\{\texttt{num\_row}(M_1), \texttt{num\_row}(M_2)\} \geq 5,$$

and thus we can conclude that

$$\models^{red} \mathcal{L}^{\#} \doteq \min_{\pi \in \{\pi_a, \pi_b\}} \max_{p_i \in \pi} \{\texttt{num\_row}(M_i)\} = 2.$$

6 Conclusions

In this paper we have proposed an analysis for understanding the class of local minima free loading problems in the case of recursive neural networks operating on directed acyclic graphs. Like for previous studies in the field, the analysis carried out in the paper gives rise to sufficient conditions to guarantee the absence of local minima in the error function. The identification of classes of loading problems for which such a property holds makes it possible to draw a clear picture to represent easily solvable problems from a computational point of view.

The most important result given in the paper involves the notion of *reduction pointer*, which turns out to be very useful for designing recursive networks with corresponding local minima free error function. The results given in this paper extend those previously published in (Frasconi et al., 1997b, 2000). As a result the condition of absence of local minima is typically gained with a smaller number of hidden units.

It is worth mentioning that the given bounds on the number of hidden units that are needed to avoid local minima are not necessarily useful for the network design, since our condition neglect generalization issues. The given bound,

however, seems to be very interesting from a theoretical point of view, especially when compared with related results given for multilayer networks ((Poston et al., 1991; Yu, 1992; Yu and Chen, 1995)). In particular it turns out that the presence of a structure in the training set plays an important role in the definition of sufficient conditions that are significantly sharper than in the more classic case of inputs represented by vectors.

Acknowledgements

We thank Monica Bianchini, Paolo Frasconi, Marco Maggini, and Ah Chung Tsoi for their useful comments and suggestions on a earlier draft of this paper.

References

Baldi, P., Hornik, K., 1989. Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks 2, 53–58.

Baum, E., 1991. Neural network design and the complexity of learning, s. judd, book review. IEEE Trans. on Neural Networks 2 (1), 181–182.

Bianchini, M., Frasconi, P., Gori, M., May 1995. Learning without local minima in radial basis function networks. IEEE Trans. on Neural Networks 6 (3), 749–756.

Bianchini, M., Frasconi, P., Gori, M., Maggini, M., 1998. Optimal learning in artificial neural networks: A theoretical view. In: Leondes, C. (Ed.), Neural Network Systems Techniques and Applications. Academic-Press, Ch. 1, pp. 1–51.

Bianchini, M., Gori, M., 1996. Optimal learning in artificial neural networks: A review of theoretical results. Neurocomputing 13, 313–346.

Bianchini, M., Gori, M., Maggini, M., March 1994. On the problem of local minima in recurrent neural networks. IEEE Trans. on Neural Networks 5 (2), 167–177, special Issue on Recurrent Neural Networks.

Blum, E., Rivest, R., 1992. Training a 3-node neural network is NP-complete. Neural Networks 5, 117–127.

Blum, L., Cucker, F., Shub, M., Smale, S., 1998. Complexity and Real Computation. Springer.

Brady, M., Raghavan, R., Slawny, J., 1989. Back-propagation fails to separate where perceptrons succeed. IEEE Trans. on Circuits and Systems 36, 665–674.

Frasconi, P., amd F. Kurfess, M. G., Sperduti, A., June 2002. Special issue on integration of symbolic and connectionist systems. Cognitive Systems Research 3 (2).

Frasconi, P., Fanelli, S., Gori, M., Protasi, M., 9–12 June 1997a. Suspiciousness

of loading problems. In: IEEE International Conference on Neural Networks. IEEE Press, pp. II 1240–1245.

Frasconi, P., Gori, M., Sperduti, A., 1997b. On the efficient classification of data structures by neural networks. In: International Joint Conference on Artificial Intelligence, Nagoya. pp. 1066–1171.

Frasconi, P., Gori, M., Sperduti, A., 1998. A general framework for adaptive processing of data structures. IEEE Trans. on Neural Networks 9 (5), 768.

Frasconi, P., Gori, M., Sperduti, A., 2000. Learning efficiently with neural networks: A theoretical comparison between structured and flat representations. In: European Conference on Artificial Intelligence, Berlin. pp. 301–305.

Frasconi, P., Gori, M., Sperduti, A., March/April 2001. Special issue on connectionist models for learning in structured domains. IEEE Trans. on Knowledge and Data Engineering 13 (2).

Goller, C., 1997. A Connectionist Approach for Learning Search-Control Heuristics for Automated Deduction Systems. Ph.D. thesis, Technical University Munich, Computer Science.

Goller, C., Küchler, A., 1996. Learning task-dependent distributed structure-representations by backpropagation through structure. In: IEEE International Conference on Neural Networks. pp. 347–352.

Gori, M., Maggini, M., 1996. Optimal convergence of on-line backpropagation. IEEE Trans. on Neural Networks 7 (1), 251–253.

Gori, M., Meer, K., 2002. A step towards a complexity theory for analog systems. Mathematical Logic Quarterly 48, 45–58.

Gori, M., Tesi, A., January 1992. On the problem of local minima in backpropagation. IEEE Transactions on Pattern Analysis and Machine Intelligence 14 (1), 76–86.

Gouhara, K., Kanai, N., Uchikawa, Y., 1993. Experimental learning surface and learning process in multilayer neural networks. Tech. rep., Nagoya University, Nagoya, Japan.

Gouhara, K., Uchikawa, Y., 1993. Memory surface and learning surfaces in multilayer neural networks. Tech. rep., Nagoya University, Nagoya, Japan.

Hush, D., Salas, J., 1988. Improving the learning rate of back-propagation with the gradient reuse algorithm. In: IEEE Int. Conf. on Neural Networks. Vol. 1. IEEE, New York, San Diego 1988, pp. 441–447.

Judd, J., 1990. Neural Network Design and the Complexity of Learning. The MIT Press, Cambridge, London.

Minsky, M., Papert, S., 1988. Perceptrons - Expanded Edition. MIT Press, Cambridge.

Poston, T., Lee, C., Choie, Y., Kwon, Y., July 1991. Local minima and backpropagation. In: International Joint Conference on Neural Networks. Vol. 2. IEEE Press, Seattle, (WA), pp. 173–176.

Rosenblatt, F., 1962. Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanism. Spartan Books, Washington D.C.

Shynk, J., 1990. Performance surfaces of a single-layer perceptron. IEEE

Trans. on Neural Networks 1 (3), 268–274.
Sima, J., 2002. Training a single sigmoidal neuron is hard. Neural Computation 14, 2709–2728.
Sontag, E., Sussman, H., June 1989. Backpropagation separates when perceptrons do. In: International Joint Conf. on Neural Networks. Vol. 1. IEEE Press, Washington DC, pp. 639–642.
Sperduti, A., Starita, A., 1997. Supervised neural networks for classification of structures. IEEE Trans. on Neural Networks 8 (3), 714–735.
Yu, X., 1992. Can backpropagation error surface not have local minima? IEEE Trans. on Neural Networks 3 (6), 1019–1020.
Yu, X., Chen, G., 1995. On the local minima free condition of backpropagation learning. IEEE Trans. on Neural Networks 6 (6), 1300–1303.