ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

Tree-Recursive Computation of Gradient Information for Structures

Andreas Küchler

Dept. of Neural Information Processing, University of Ulm, 89069 Ulm, Germany, Email: kuechler@neuro.informatik.uni-ulm.de

Abstract. Recently, the so-called *Backpropagation Through Structure* (BPTS) gradient calculation algorithm has been developed to capture learning scenarios where data is adequately represented by hybrid continuous-discrete structures (e.g. labeled ordered trees, nodes augmented by continuous information). BPTS can be viewed as an extension of the well-known *Backpropagation Through Time* (BPTT) algorithm for discrete-time dynamical systems and sequence processing. The well-known (functionally equivalent) *Real-time Recurrent Learning* (RTRL) algorithm has to be favored to BPTT if long sequences are processed.
This paper investigates whether and how RTRL can be generalized – while conserving its appealing *algorithmic* properties – to calculate the gradient information for models operating on the domain of rooted labeled ordered trees. The answer is partly negative. It turns out that a *postorder traversal* of the tree has to be obeyed in order to keep the space consumption independent from the size of the input structures. By processing vertices in an *inverse topological ordering* the algorithm can also be applied on *labeled directed ordered acyclic graphs*. However, we show that on this graph domain the memory consumption grows (in the worst case) linearly with the size of the input structure.

1 Introduction

Recently, the "zoo" of analog models of computation and learning has been extended by some inhabitants that are capable to capture structured information (for an introduction based on a probabilistic framework see Frasconi et al. [2]). For the purpose of this paper we focus our attention to the class of *tree-recursive dynamical systems* (TRDS) which can be regarded as *deterministic* machines designed to map rooted labeled ordered trees to the Euclidean space.

Definition 1 (TRDS) Let $\mathcal{T}(U)$ be the set of rooted labeled ordered trees (where nodes carry labels of the type U) with maximum outdegree k and let '$\oplus$' denote the vector concatenation operator. A *tree-recursive dynamical system* (TRDS) (M, nil, f, h) consists of a *state space* $M \subseteq \mathbb{R}^m$, an encoding $nil \in M$ of the "empty tree" and two functions f, h of the type $f : U \times M^k \to M$ and $h : M \to \mathbb{R}^q$ where $q \in \mathbb{N}$. Let $t \in \mathcal{T}(U)$ be an arbitrary tree with $d \leq k$ immediate subtrees $t_1, t_2, \ldots, t_d$. The mapping $g : \mathcal{T}(U) \to M$ is defined by

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

tree-recursion

$$g(t) = f(\lambda(root(t)), g(t_1) \oplus g(t_2) \oplus \cdots \oplus g(t_d) \oplus \underbrace{nil \oplus \cdots \oplus nil}_{k-d}) \qquad (1)$$

where the operator *root* extracts the root node of a tree and λ its label information. Finally the mapping $\Xi : \mathcal{T}(U) \to \mathbb{R}^q$ computed by the TRDS is defined by the composition of g with the *output* function h, i.e. $\Xi := h \circ g$.

The so-called *neural folding architecture* (FA) is an example for "neural network" instances of the TRDS model (Goller and Küchler [3]). The mappings f and h are implemented by standard multi-layer neural networks. Several other neural network architectures (e.g. *cascade correlation*, *neural trees*) have been used in a similar manner (Sperduti and Starita [5]). TRDS are very powerful models of computation. Hammer [4] proved the FA to be a *universal approximator* for mappings of the type $\Xi : \mathcal{T}(\mathbb{R}^n) \to \mathbb{R}^q$.

The basic idea to incorporate adaptivity ("learning") into TRDS models is to choose parameterized functions f, h and to estimate these parameters according to the given task and data. The learning task can often be adequately formulated as the optimization of an *error measure* E (in the parameter space) which depends somehow on the parameter vectors $\mathbf{w}^f, \mathbf{w}^h$ (found in f, h) and on the output $\hat{\Xi}(t)$ that is computed by the given TRDS (under its current parameter settings) for trees t from a given data set $\mathcal{P}$. *Gradient-based* methods [1] are usually taken into account to solve this optimization problem if the error measure is a continuous and differentiable function on the region of interest.

Recently, the well-known *Backpropagation Through Time* (BPTT) algorithm for discrete-time dynamical systems and sequence processing has been generalized to the so-called *Backpropagation Through Structure* (BPTS) scheme which allows to compute the first-order gradient for functional expressions where rooted labeled ordered trees (mapped by the FA) are embedded [3]. Both BPTT and BPTS are time-efficient algorithms, however their memory consumption grows *linearly* with the size of the input structures. The well-known *Real-time Recurrent Learning* (RTRL) algorithm works with *constant* space and is to be favored to (the functionally equivalent) BPTT if long sequences are going to be processed (we refer to Williams and Zipser [6]). Furthermore, it is applicable for adapting models to an infinite stream of data, while BPTT is not.

In this paper we investigate whether and how RTRL can be generalized – while conserving its appealing *algorithmic* properties – to operate for TRDS on the tree domain. We present the mathematical derivation of a RTRL-style gradient calculation for TRDS in Section 2 (for the special case of neural network instances, see Sperduti and Starita [5]). The algorithmic perspective is elaborated in Section 3 and Section 4. Section 5 is to summarize the major implications.

[1] See Baldi [1] for a survey from a dynamical systems perspective.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

2 Tree-Recursive Calculation of the Gradient

For reasons of simplicity we assume for a given TRDS (M, nil, f, h) with $M = \mathbb{R}^m$, $f : \mathbb{R}^{n+km} \to \mathbb{R}^m$, $h : \mathbb{R}^m \to \mathbb{R}^q$, $nil \in \mathbb{R}^m$ the mappings f, h and the error function $E : \Omega_f \times \Omega_h \to \mathbb{R}$ to be continuous and differentiable on the whole parameter space $\Omega_f \times \Omega_h$. Let the data be given by $\mathcal{P} = \{(s_1, \Xi(s_1), (s_2, \Xi(s_2), \ldots, (s_p, \Xi(s_p))\}$ where $s_i \in \mathcal{T}(\mathbb{R}^n)$ and $\Xi(s_i) \in \mathbb{R}^q$, let $\hat{\Xi}_j$ $(j = 1, 2, \ldots, q)$ denote the i-th function $\hat{\Xi}_j : \mathcal{T}(\mathbb{R}^n) \to \mathbb{R}$ of the field $\hat{\Xi}$ computed by the TRDS under the current parameter assignment. Further, let $\mathbf{w}^f$, $\mathbf{w}^h$ be the are $n_f = \dim(\Omega_f)$, $n_h = \dim(\Omega_h)$-dimensional parameter vectors belonging to f, h. Without loss of generality the calculation of the gradient is exemplified by supervised learning and the popular mean squared error function

$$E = \frac{1}{2}\sum_{i=1}^{p} E(s_i) = \frac{1}{2}\sum_{i=1}^{p}\sum_{j=1}^{q}(\Xi_j(s_i) - \hat{\Xi}_j(s_i))^2$$

Thus, the objective is to calculate $\mathbf{e}^f(t) := \partial E(t)/\partial \mathbf{w}^f$ and $\mathbf{e}^h(t) := \partial E(t)/\partial \mathbf{w}^h$ ($\mathbf{e}^f(t) \in \mathbb{R}^{n_f}$ and $\mathbf{e}^h(t) \in \mathbb{R}^{n_h}$) for a given input tree $t \in \mathcal{T}(\mathbb{R}^n)$. We define the following functional matrices as compact representations of partial derivatives.

$$\begin{array}{rcll} G_{ij}(t) & := & \partial g_j(t)/\partial \mathbf{w}^f & \mathbf{G} \in \mathbb{R}^{n_f \times m} \\ J^h_{ij}(g(t)) & := & h_{j,x_i}(g(t)) & \mathbf{J}^h \in \mathbb{R}^{m \times q} \\ H_{ij}(g(t)) & := & \partial h_j(g(t))/\partial \mathbf{w}^h & \mathbf{H} \in \mathbb{R}^{n_h \times q} \\ J^{f(r)}_{ij}(\mathbf{x}(t)) & := & f_{j,x_{n+(r-1)m+i}}(\mathbf{x}(t)) & \mathbf{J}^{f(r)} \in \mathbb{R}^{m \times m}, 1 \le r \le k \\ F_{ij}(\mathbf{x}(t)) & := & \partial f_j(\mathbf{x}(t))/\partial \mathbf{w}^f & \mathbf{F} \in \mathbb{R}^{n_f \times m} \end{array}$$

Note that $h_{j,x_i} := \partial h_j/\partial x_i$ is a shorthand for the partial derivative of h_j with respect to its i-th argument, i.e. $\mathbf{J}^{f(r)}$ and $\mathbf{J}^h$ are the transposed *Jacobian* matrices of the mappings f and h at the locations $\mathbf{x}(t) \in \mathbb{R}^{n+km}$ and $g(t) \in \mathbb{R}^m$. The value k denotes the maximum outdegree found in the tree domain $\mathcal{T}(\mathbb{R}^n)$ and $\mathbf{z}(t) := (\hat{\Xi}(t) - \Xi(t)) \in \mathbb{R}^q$ the difference between computed and desired output for a given input tree. By applying the well-known *generalized chain rule* for differentials on the the composition $\hat{\Xi} := h \circ g$ (see Definition 1) we get

$$\mathbf{e}^h(t) = \mathbf{H}(g(t)) \cdot \mathbf{z}(t) \qquad \mathbf{e}^f(t) = \mathbf{G}(t) \cdot (\mathbf{J}^h(g(t)) \cdot \mathbf{z}(t)) \tag{2}$$

Now, let $t \in \mathcal{T}(\mathbb{R}^n)$ be an arbitrary tree with $d \le k$ immediate subtrees $t_1, t_2, \ldots, t_d$. The *tree-recursive* dynamics (see Equation 1) can be utilized to directly evolve the partial derivative $\mathbf{G}(t)$ to

$$\mathbf{G}(t) = \mathbf{F}(\mathbf{x}(t)) + \sum_{i=1}^{d} \mathbf{G}(t_i) \cdot \mathbf{J}^{f(i)}(\mathbf{x}(t)) \tag{3}$$

where $\mathbf{x}(t) = \lambda(root(t)) \oplus f(\mathbf{x}(t_1)) \oplus \cdots \oplus f(\mathbf{x}(t_d)) \oplus \underbrace{nil \oplus \cdots \oplus nil}_{k-d}$

Equation 3 measures the total impact of the variables $\mathbf{w}^f$ on the target g, i.e. the direct effects ($\mathbf{F}(\mathbf{x}(t))$, via f) and the indirect effects (via the tree-recursive composition of f on the immediate subtrees). One can easily verify that the recursion is well-founded.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

3 The Algorithmic Perspective

Let $u = |t|$ denote the size (number of nodes) of the given input tree. One can observe that the formulation of Equation 3 is *not tail-recursive*. Thus, a direct algorithmic translation would generate a tree-recursive process (at runtime) allocating worst case $\Theta(u(n_f m + km))$ memory, i.e. equivalent to store u times the matrix $\mathbf{G}(t)$ and the vector $\mathbf{x}(t)$. Can this behavior be improved?

Algorithm 1 TRGC$(t, \Xi(t))$

1: $mem \leftarrow$ ALLOCATE-MEMORY$(\, 2k, (n_f \times m + m)\,)$
2: $(\mathbf{y}, \mathbf{G}) \leftarrow$ TRGC-POSTORDER-TREE-WALK$(root(t), mem)$
3: $\mathbf{z} \leftarrow h(\mathbf{y}) - \Xi(t)$
4: $\mathbf{e}^f \leftarrow \mathbf{G} \cdot (\mathbf{J}^h(\mathbf{y}) \cdot \mathbf{z})$; $\mathbf{e}^h \leftarrow \mathbf{H}(\mathbf{y}) \cdot \mathbf{z}$
5: FREE-MEMORY(mem) ; **return** $(\mathbf{e}^f, \mathbf{e}^h)$

Algorithm 1 (TRGC) expects a pair of input tree $t \in \mathcal{T}(\mathbb{R}^n)$ and desired target value $\Xi(t)$ as input. We assume that the matrices of partial derivatives $\mathbf{H}$, $\mathbf{J}^h$, $\mathbf{F}$ and $\mathbf{J}^{f(r)}$ $(1 \leq r \leq k)$ can be determined in an analytical way and are (together with the mappings f and h) compiled a priori into the program code. An alternative solution might be to numerically approximate the partial derivatives by a finite Taylor series expansion of the original mappings [2]. The interface to function TRGC-POSTORDER-TREE-WALK is provided by allocating $2k$ blocks of memory equivalent to store the gradient matrix $\mathbf{G}$ and the state vector $\mathbf{y} = f(\mathbf{x}(t))$ in advance (line 1).

Algorithm 2 specifies a *postorder* left-to-right tree traversal which is combined with an explicit memory management. The operators CLAIM-MEMORY and RELEASE-MEMORY take $O(1)$ time if implemented as two double-linked lists (of claimed and released memory cells) and only a pointer to a released cell is returned by a claim operation. Further, the data structure representing a tree node v has to be augmented by two entries to keep the memory pointers $v.\mathbf{G}$ and $v.\mathbf{y}$ ($O(1)$ space). By a careful analysis of Equation 3 one can observe that both the computation of the gradient information $\mathbf{G}(t)$ and the computation of the state $f(\mathbf{x}(t))$ are defined by tree-recursion on t. Algorithm 2 makes use of this fact by interlacing these computations into one tree-recursive process.

Proposition 1 ***a)*** *The gradient information* $\mathbf{e}^h(t)$ *and* $\mathbf{e}^f(t)$ *is correctly computed by Algorithm 1 (and Algorithm 2) for any given (compatible) pair of input tree* t *and target value* $\Xi(t)$. ***b)*** *The memory consumption is* independent *from the size of the input tree, i.e. is calculated to* $\Theta(2kn_f m + n_h + q + n)$.

Proof (Sketch) **a)** The postorder traversal guarantees that each node of the input tree will be touched exactly once. Secondly, all subtrees (if there are any) of the node under consideration have already been visited before. Furthermore, Alg. 1 (lines 3–4) directly implements Equation 2, lines 5-11 of Alg. 2 are a direct translation of Equation 3. **b)** For each node v of the input tree memory

[2] However, this requires additional evaluations of f and h in the environment of $\mathbf{x}(t)$ and is source of potential errors.

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

Algorithm 2 TRGC-POSTORDER-TREE-WALK(v, mem)

1: $d \leftarrow$ OUTDEGREE(v) ; $\mathbf{x} \leftarrow \lambda(v)$
2: **for** $i \leftarrow 1$ to d **do**
3: TRGC-POSTORDER-TREE-WALK(v_i, mem)
4: $(v.\mathbf{G}, v.\mathbf{y}) \leftarrow$ CLAIM-MEMORY($mem, n_f \times m + m$)
5: **for** $i \leftarrow 1$ to d **do**
6: $\mathbf{x} \leftarrow \mathbf{x} \oplus v_i.\mathbf{y}$; RELEASE-MEMORY($mem, v_i.\mathbf{y}$)
7: **for** $i \leftarrow d+1$ to k **do**
8: $\mathbf{x} \leftarrow \mathbf{x} \oplus nil$
9: $v.\mathbf{y} \leftarrow f(\mathbf{x})$; $v.\mathbf{G} \leftarrow \mathbf{F}(\mathbf{x})$
10: **for** $i \leftarrow 1$ to d **do**
11: $v.\mathbf{G} \leftarrow v.\mathbf{G} + v_i.\mathbf{G} \cdot \mathbf{J}^{f(i)}(\mathbf{x})$; RELEASE-MEMORY($mem, v.\mathbf{G}$)
12: **return** $(v.\mathbf{y}, v.\mathbf{G})$

will be claimed only once (see Alg. 2, line 4). The postorder traversal ensures that after each node has been processed the release operation is applied on each successor node. It can easily be shown that at most $2k$ memory cells will be claimed at any point of time. The worst case becomes apparent when a rightmost node (of k siblings) rooted itself to k subtrees is going to be visited. ∎

4 Extending the Tree Domain

The BPTS algorithm is known to work on *rooted labeled directed ordered acyclic graphs* (RLDOAGs) if a *topological ordering* is obeyed while visiting the vertices [3]. Thus, the expressive power of TRDS models can be slightly enriched. On the other hand, (a set of) purely symbolic trees (i.e. from the domain $\mathcal{T}(\Sigma)$) can be "compressed" to RLDOAGs by applying a *common subexpression elimination* (CSE) technique which in turn leads to a considerable speed-up in gradient computation. Algorithm 3 describes a variant of tree-recursive gradient computation for RLDOAGs. By processing the given vertices V in an inverse topological ordering it is guaranteed that the gradient information $v_i.\mathbf{G}$ and the state information $v_i.\mathbf{y} = f(\mathbf{x})$ is already calculated for the successors v_i of the vertex v to be visited next.

Again, an explicit memory management is used to achieve an optimal memory consumption. The data structure representing a vertex is augmented by a set of vertices that are incident to it. The operator INCIDENT returns this set. Memory kept for the successors of a vertex v (under consideration) is freed if all vertices incident to it are larger or equal according to the chosen topological ordering (lines 12,13).

Proposition 2 *The memory consumption of Algorithm 3 grows* linearly *with the number of vertices in the given (worst case) input RLDOAG.*

Proof (Sketch) Consider the RLDOAG built of $\tilde{u} = 2(n+1)$ vertices $V = \{v_1, v_2, \ldots, v_{2n+2}\}$ ($n \in \mathbb{N}$) and the edges (in adjacency description): $\text{succ}(v_i) = (v_{2n+3-i}, v_{i+1})$ if $1 \leq i \leq n+1$, $\text{succ}(v_i) = (v_{i+1}, v_{i+1})$ if $n+2 \leq i \leq 2n+1$,

ESANN'1999 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 21-23 April 1999, D-Facto public., ISBN 2-600049-9-X, pp. 39-44

Algorithm 3 TRGC-RLDOAG(V)

1: Choose a topological ordering $<_{\text{TOP}}$ on V
2: **for** each $v \in V$ in inverse topological ordering $>_{\text{TOP}}$ **do**
3: $d \leftarrow \text{OUTDEGREE}(v)$; $\mathbf{x} \leftarrow \lambda(v)$
4: $(v.\mathbf{G}, v.\mathbf{y}) \leftarrow \text{ALLOCATE-MEMORY}(n_f \times m + m)$
5: **for** $i \leftarrow 1$ to d **do**
6: $\mathbf{x} \leftarrow \mathbf{x} \oplus v_i.\mathbf{y}$
7: **for** $i \leftarrow d+1$ to k **do**
8: $\mathbf{x} \leftarrow \mathbf{x} \oplus nil$
9: $v.\mathbf{G} \leftarrow \mathbf{F}(\mathbf{x})$; $v.\mathbf{y} \leftarrow f(\mathbf{x})$
10: **for** $i \leftarrow 1$ to d **do**
11: $v.\mathbf{G} \leftarrow v.\mathbf{G} + v_i.\mathbf{G} \cdot \mathbf{J}^{f(i)}(\mathbf{x})$
12: **if** $\forall w \in \text{INCIDENT}(v_i) : (v <_{\text{TOP}} w) \vee (v = w)$ **then**
13: FREE-MEMORY$(v_i.\mathbf{G}, v_i.\mathbf{y})$
14: **return** $(v.\mathbf{y}, v.\mathbf{G})$ where v is the root vertex in V

and $\text{succ}(v_i) = (\,,\,)$ else. Due to the linear chain structure of the edges ($v_{i+1} \in \text{succ}(v_i)$) the unique topological ordering on the vertices is $v_1, v_2, \ldots, v_{2n+2}$. Alg. 3 starts according to the inverse ordering. FREE-MEMORY is not executed until the vertex v_{n+1} is reached. Thus, $O(\tilde{u})$ blocks of memory equivalent to store the matrix $\mathbf{G}$ and the state $\mathbf{y}$ have to allocated. ∎

5 Conclusion

RTRL for sequence processing has the appealing property that the memory consumption is independent from the size of the input structure. This property can be conserved by tree-recursive gradient computation for finite trees iff a postorder traversal is obeyed, while it is lost in the domain of finite RLDOAGs. It is hard to imagine a reasonable definition of TRDS and the TRGC algorithm on the domain of infinite trees (RLDOAGs).

References

[1] P. Baldi. Gradient Descent Learning Algorithm Overview: A General Dynamical Systems Perspective. *IEEE Trans. on Neural Networks*, 6(1):182–195, 1995.

[2] P. Frasconi, M. Gori, and A. Sperduti. A General Framework for Adaptive Processing of Data Structures. *IEEE Trans. on Neural Networks*, 9(5):768–786, 1998.

[3] C. Goller and A. Küchler. Learning Task-Dependent Distributed Representations by Backpropagation Through Structure. In *Proc. IEEE Intl. Conf. on Neural Networks (ICNN'96)*, pp. 347–352, 1996.

[4] B. Hammer. On the Approximation Capability of Recurrent Neural Networks. In *Proc. Intl. Symp. on Neural Computation (NC'98)*, pp. 512–518, 1998.

[5] A. Sperduti and A. Starita. Supervised Neural Networks for the Classification of Structures. *IEEE Trans. on Neural Networks*, 8(3):714–735, 1997.

[6] R. J. Williams and D. Zipser. Gradient-Based Learning Algorithms for Recurrent Networks and Their Computational Complexity. In *Backpropagation: Theory, Architectures and Applications*, pp. 433–486. LEA, 1994.