ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Implicitly and Explicitly Constrained Optimization Problems for Training of Recurrent Neural Networks

Carl-Johan Thore

Linköping University - Division of Mechanics
581 83 Linköping - Sweden

Abstract. Training of recurrent neural networks is typically formulated as unconstrained optimization problems. There is, however, an implicit constraint stating that the equations of state must be satisfied at every iteration in the optimization process. Such constraints can make a problem highly non-linear and thus difficult to solve. A potential remedy is to reformulate the problem into one in which the parameters and state are treated as independent variables and all constraints appear explicitly. In this paper we compare an implicitly and an explicitly constrained formulation of the same problem. Reported numerical results suggest that the latter is in some respects superior.

1 Introduction

Recurrent neural networks (RNNs) in various forms have been subject to much research for at least 30 years. RNNs have been applied to many different tasks, including control of robots, solving combinatorial optimization problems, and time-series prediction [1]. The author's interest in RNNs stems from work on design optimization of a type of mechatronic systems where motion is controlled by RNNs in limit-cycle oscillation [2]. For this reason, the paper focuses on learning of state space trajectories following Pearlmutter [3] and others, but we believe that it bears relevance for other applications as well.

The problem of training RNNs to generate desired state space trajectories dates back to the 1980s [3]. In 1999, Galicki et al. [4] extended the problem to include time-varying weights and solved it using a sequential linear programming algorithm, but the basic idea has mostly remained unchanged [5]. Training is formulated as an unconstrained optimization problem with the objective to minimize some error functional, and the problem is typically treated by variants of the gradient descent algorithm. However, as the state of the RNN is taken to be a function of the parameters to be optimized, there is an implicit constraint that the equations of state must be satisfied for each set of parameters. This makes the problem highly non-linear and hard to solve. In particular, difficulties with learning long-term time dependencies, which are explained by so-called vanishing or exploding gradients, have been known for quite some time but has not yet been completely resolved [6]. Perhaps the most promising approach is due to Martens and Sutskever [7].

In this paper we reformulate a classic problem by treating the parameters and the state as independent optimization variables while making all constraints

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

explicit. This approach is common in optimal control [8] but appears not to have been tried for training of RNNs. We shall refer to the reformulated problem as the *simultaneous* problem, and the traditional problem as the *nested* problem.

2 The state problem

We consider an RNN of additive type, governed by a non-linear, ordinary differential equation (ODE). The state problem comprises the following initial value problem:

$$\dot{\boldsymbol{v}} = \boldsymbol{W}(\boldsymbol{w})\boldsymbol{s}(\boldsymbol{v}) - \boldsymbol{C}\boldsymbol{v} + \boldsymbol{u} \qquad t \in (0, T] \tag{1a}$$

$$\boldsymbol{v}(0) = \boldsymbol{v}^0. \tag{1b}$$

Here, a superposed dot denotes time-derivative. $\boldsymbol{W} \in \mathbb{R}^{m\times m}$ is called the weight matrix, and its entries are collected in a vector $\boldsymbol{w} \in \mathbb{R}^p$, $p = m^2$. The matrix $\boldsymbol{C}$ is positive definite and diagonal, and $\boldsymbol{u} = \boldsymbol{u}(t)$ is an external input signal. The hyperbolic tangent is used as activation function for each neuron, so $\boldsymbol{s}(\boldsymbol{v}) = [\tanh(v_i)]$.

It can be shown that every solution to (1) is bounded, so there exists a unique solution $\boldsymbol{v} = \boldsymbol{v}(t, \boldsymbol{w}, \boldsymbol{v}^0)$ for every T [2]. If the input signal $\boldsymbol{u}$ is of class C^q, $q \geq 0$, it follows that the solution is of class C^q [2].

3 The optimization problem

The objective of our problem is to find network weights, i.e., $\boldsymbol{w}$, that minimize the difference between the actual and desired state-space trajectories for a subset of the neurons in the network over some time interval $[T_0, T]$ for a given input $\boldsymbol{u}$. This difference is quantified by a functional

$$\int_{T_0}^{T} ||\tilde{\boldsymbol{v}} - \boldsymbol{v}_*||^2 \, \mathrm{d}t,$$

where $||\cdot||$ is the ℓ_2-norm, $\tilde{\boldsymbol{v}}$ is a subvector of $\boldsymbol{v}$, and $\boldsymbol{v}_* = \boldsymbol{v}_*(t)$ defines the target trajectory.

A common ingredient in neural network training is some form of regularization term. The ℓ_1-norm of $\boldsymbol{w}$, which tends to yield sparse weight matrices [2] is an attractive choice. A simple way of achieving sparsity while avoiding non-differentiability is to first replace the original variable $\boldsymbol{w}$ by two sets of variables $\boldsymbol{w}^+$ and $\boldsymbol{w}^-$ having non-negative entries, such that $\boldsymbol{w} = \boldsymbol{w}^+ - \boldsymbol{w}^-$. A penalty term

$$\phi(\boldsymbol{w}^+, \boldsymbol{w}^-) = \sum_{i=1}^{p} \left(w_i^+ + w_i^- \right)$$

with the same effect as an ℓ_1-norm penalty can then be added to the objective of our problem. An obvious drawback of this approach is that the number of parameters is doubled, but on the other hand we get smooth optimization problems.

For use below we introduce the set $\mathbb{R}^n_+ = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, x_i \geq 0,\ i = 1, \ldots, n\}$.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

3.1 Simultaneous formulation

The simultaneous formulation is the following, infinite-dimensional, optimization problem:

$$\begin{aligned} &\underset{\boldsymbol{v}\in C^q,\,(\boldsymbol{w}^+,\boldsymbol{w}^-)\in\mathbb{R}^{2p}_+}{\text{minimize}} \quad \int_{T_0}^{T} ||\tilde{\boldsymbol{v}} - \boldsymbol{v}_*||^2 \,\mathrm{d}t + \gamma\phi(\boldsymbol{w}^+,\boldsymbol{w}^-) \\ &\text{subject to} \quad \begin{cases} \dot{\boldsymbol{v}} = \boldsymbol{W}(\boldsymbol{w})\boldsymbol{s}(\boldsymbol{v}) - \boldsymbol{C}\boldsymbol{v} + \boldsymbol{u}, \quad t \in (0,T] \\ \boldsymbol{v}(0) = \boldsymbol{v}^0, \end{cases} \end{aligned} \tag{2}$$

where γ is a positive constant, and $\boldsymbol{w} = \boldsymbol{w}^+ - \boldsymbol{w}^-$.

3.2 Nested formulation

Treating the state variables as (implicit) functions of the parameters, i.e., $\boldsymbol{v} = \boldsymbol{v}(\boldsymbol{w})$, yields the nested version of our optimization problem:

$$\underset{(\boldsymbol{w}^+,\boldsymbol{w}^-)\in\mathbb{R}^{2p}_+}{\text{minimize}} \quad \int_{T_0}^{T} ||\tilde{\boldsymbol{v}}(\boldsymbol{w}) - \boldsymbol{v}_*||^2 \,\mathrm{d}t + \gamma\phi(\boldsymbol{w}^+,\boldsymbol{w}^-), \tag{3}$$

where $\tilde{\boldsymbol{v}}(\boldsymbol{w})$ is obtained by solving the state problem (1) for each $\boldsymbol{w}$. Problem (3) represents the "standard" formulation in the literature on RNNs.

4 Finite-dimensional approximations

Approximate solutions to (2) and (3) are obtained by solving discrete versions of these problems.

Let the interval $[0,T]$ be divided into n segments, each of length Δt. The number of grid points is $M = n+1$. If $\boldsymbol{f}(\cdot,\cdot)$ denotes the right-hand side of (1a), the Euler forward method now yields an iterative scheme

$$\begin{aligned} &\boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \Delta t \boldsymbol{f}_i(\boldsymbol{v}_i, \boldsymbol{w}), \quad i = 1,\ldots,n, \\ &\boldsymbol{v}_0 = \boldsymbol{v}^0 \end{aligned} \tag{4}$$

for computing $O(\Delta t)$-approximations $\boldsymbol{v}_i$ of $\boldsymbol{v}(t_i)$ – the solution to (1) evaluated at t_i – for all i. The integral in the objective of, say (2), is approximated as

$$\int_{T_0}^{T} ||\tilde{\boldsymbol{v}} - \boldsymbol{v}_*(t)||^2 \,\mathrm{d}t \approx \sum_{i=k}^{n} \Delta t ||\tilde{\boldsymbol{v}}_i - \boldsymbol{v}_*(t_i)||^2,$$

where the index $k = \arg\min_i ||t_i - T_0||$.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

Introducing $\boldsymbol{x} = (\boldsymbol{v}_0, \ldots, \boldsymbol{v}_M)^\mathsf{T} \in \mathbb{R}^{mM}$, the finite-dimensional approximation to (2) can be written as

$$\begin{aligned} &\underset{\boldsymbol{x}\in\mathbb{R}^{mM},\, (\boldsymbol{w}^+,\boldsymbol{w}^-)\in\mathbb{R}^{2p}_+}{\text{minimize}} \quad \sum_{i=k}^{n} \Delta t ||\tilde{\boldsymbol{v}}_i - \boldsymbol{v}_*(t_i)||^2 + \gamma\phi(\boldsymbol{w}^+, \boldsymbol{w}^-) \\ &\text{subject to} \quad \begin{cases} \boldsymbol{v}_{i+1} = \boldsymbol{v}_i + \Delta t \boldsymbol{f}_i(\boldsymbol{v}_i, \boldsymbol{w}), \quad i = 1, \ldots, n, \\ \boldsymbol{v}_0 = \boldsymbol{v}^0. \end{cases} \end{aligned} \tag{5}$$

The discrete version of (3) is

$$\underset{(\boldsymbol{w}^+,\boldsymbol{w}^-)\in\mathbb{R}^{2p}_+}{\text{minimize}} \quad \sum_{i=k}^{n} \Delta t ||\tilde{\boldsymbol{v}}(\boldsymbol{w})_i - \boldsymbol{v}_*(t_i)||^2 + \gamma\phi(\boldsymbol{w}^+, \boldsymbol{w}^-), \tag{6}$$

where $\tilde{\boldsymbol{v}}(\boldsymbol{w})_i$, $i = k, \ldots, n$, are obtained from (4).

5 Complexity

The overall complexity of the optimization process depends on the number of iterations times the cost of each iteration times the number of optimization runs (from different initial points) needed to find an acceptable solution.

With the adjoint method (backpropagation through time) for gradient calculations used here, the cost per iteration for the nested formulation is at least $O(pM)$ [9]. The Hessian (of the Lagrangian) for the nested problem is dense, but its formation and solution of linear systems is avoided in variants of gradient descent.

The simultaneous formulation is treated by an interior-point (IP) solver based on Newton's method. Search directions are computed by solving a linear (KKT-) system [10, Eq. 13], an operation which here has worst-case cost $\kappa O([mM+p]^2)$, where κ depends primarily on the sparsity of the constraint Jacobian and the Hessian. The competitiveness of (5) thus hinges on the sparsity of the derivative matrices. Fortunately, these are typically very sparse for local collocation methods such as the Euler forward scheme used here [8].

6 Numerical results

A fully connected network of five ($m = 5$) neurons is used. Given a constant input signal, the time histories of the states of the output neurons plotted against each other should form a circle; cf. [3]. We let the target function be $\boldsymbol{v}^* = (\sin(5\pi t/8), \cos(5\pi t/8))$ and choose time intervals $[T_0, T]$ such that the error functionals are evaluated over two laps of the circle [3]; just one lap is often not enough to ensure that the shape of the limit-cycle persists for $t > T$. To ensure a persistent shape, T_0 should also be set to some relatively large value; see Table 1 below. We take $\boldsymbol{C}$ to be an identity matrix, $\boldsymbol{u} = (1, 0, 0, 0, 0)$ and let $\tilde{\boldsymbol{v}}$ contain the fourth and fifth component of $\boldsymbol{v}$. γ is set to 10^{-7}, mainly because a small positive value tended to improve numerical performance slightly.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

As an optimization solver we used Ipopt v. 3.11.2 [10] with the direct linear solver MA57 [11]; the update strategy for the barrier parameter set to "adaptive"; the maximum number of iterations set to 3000; and default settings for the other options. Exact first and second order derivatives were computed with automatic differentiation using ADOL-C [12]. For the Hessian, however, we used mainly Ipopt's limited-memory quasi-Newton approximation (lbfgs), with "limited_memory_max_history" set to 60. The overall framework was implemented in Matlab R2012a running on an Intel Core I5-520m.

Table 1 shows a comparison between the simultaneuous (Sim) formulation (5) and the nested (Nes) formulation (6). The numbers are based on optimiza-

Problem	# iterations min, med, max	f^* min, med, max	Δt	I	Var.	CPU [s] med
Nes (l)	476, 3000, 3000	4e-6, 0.5, 3e3	0.1	S	50	264
Nes (e)	3000, 3000, 3000	1e-5, 3, 3	0.1	S	50	222
Sim (l)	136, 319, 1960	3e-6, 1e-5, 8e-4	0.1	S	465	47
Sim (e)	73, 158, 455	4e-6, 6e-6, 3e-5	0.1	S	465	21
Nes (l)	132, 3000, 3000	9e-6, 3, 5e6	0.1	L	50	345
Sim (l)	112, 326, 3000	2e-6, 1e-5, 4	0.1	L	465	44
Nes (l)	37, 3000, 3000	4.2e-6, 2.94, 7e4	0.01	L	50	283
Sim (l)	152, 369, 3000	3.7e-6, 1e-5, 2	0.01	L	8250	350

Table 1: An "l" (lbfgs) or an "e" (exact) in parenthesis in the first column marks the type of Hessian used. f^* denotes the objective function value at the final point returned by the optimization solver. med = median. Var. = Number of variables. I = Integration interval $[T_0, T]$; S = [5.0 8.2], L = [13.2 16.4]. CPU = CPU time.

tions using 30 different initial guesses for $(\boldsymbol{w}^+, \boldsymbol{w}^-)$ generated from a uniform distribution over $(0, 1)$. The state variables in the simultaneous problems were initialized to zero, except those of the output neurons that were initialized to their target values. Loose artificial upper and lower bounds were imposed on the state variables, and upper bounds was added on the parameters (none of these constraints were active at the solutions).

As can be seen in Table 1, rows 1, 2, 5 and 7, the nested formulation is very difficult to solve. For a majority of the initial guesses, Ipopt failed to find an acceptable solution within the maximum number of iterations (note that the few acceptable solutions were always obtained after 3000 iterations). Using an exact Hessian or decreasing the time step (see rows 2 and 7 in the table) had no impact on the results. For the simultaneous formulation, however, the situation is entirely opposite: good solutions was found for almost every initial guess, and with shorter CPU time compared to the nested problem.

The problems were also solved using the IP- and SQP-solvers in `fmincon` from the Matlab Optimization Toolbox, but they performed worse than Ipopt. This suggests that the difficulties with the nested formulation stem from the problem formulation itself rather than some particular property of Ipopt's algorithm.

ESANN 2014 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 23-25 April 2014, i6doc.com publ., ISBN 978-287419095-7.
Available from http://www.i6doc.com/fr/livre/?GCOI=28001100432440.

7 Concluding remarks

Two different formulations of an optimization problem for training of RNNs have been compared. The numerical results confirmed that the nested formulation, which is standard in the literature, is difficult to handle even for state-of-the-art optimization solvers. In contrast, the simultaneous formulation could, despite its greater size, be solved with relative ease — convergence within high accuracy to local optima with small objective function values was almost always obtained. The results suggest that it is possible to avoid difficulties in RNN-training by using an explicitly constrained formulation of the problem.

It should be noted that a time step $\Delta t = 0.1$ or 0.01 is often to long for the discrete solution to be a good approximation of the solution to the underlying continuous-time problem. If the plan is to implement the discrete-time system directly this does not matter, but otherwise one should always check the validity of the solution using an ODE-solver with local error control and strict tolerances. In practise, solutions to the continuous-time problem would likely be sought using higher order discretization schemes together with mesh refinement algorithms to reduce the number of iterations run on the finest grids [8].

References

[1] B Hammer, B Schrauwen, and JJ Steil. Recent advances in efficient learning of recurrent networks. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 213–226, 2009.

[2] C-J Thore. Optimal design of neuro-mechanical oscillators. *Computers & Structures*, 119:189–202, 2013.

[3] BA Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[4] M Galicki, L Leistritz, and H Witte. Learning continuous trajectories in recurrent neural networks with time-dependent weights. *IEEE Transactions on Neural Networks*, 10:741–756, 1999.

[5] AF Atiya and AG Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11:697–709, 2000.

[6] R Pascanu, T Mikolov, and Y Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th Int. Conference on Machine Learning*, 2013.

[7] J Martens and I Sutskever. Learning recurrent neural networks with Hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[8] JT Betts. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. SIAM, 2009.

[9] P Baldi. Gradient descent learning algorithm overview: A general dynamical systems perspective. *IEEE Transactions on Neural Networks*, 6, 1995.

[10] A Wächter and LT Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–27, 2006.

[11] I Duff. MA57—A code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30:118–144, 2004.

[12] A Walter and A Griewank. Getting started with ADOL-C. In U Naumann and O Schenk, editors, *Combinatorial Scientific Computing*, pages 181–202. 2012.