

Evolino for Recurrent Support Vector Machines

Jürgen Schmidhuber,^{1,2} Matteo Gagliolo,¹ Daan Wierstra,¹ and Faustino Gomez¹

1- IDSIA - Galleria 2, 6928 Manno (Lugano) - Switzerland

2- TU Munich - Boltzmannstr. 3, 85748 Garching, München - Germany

{juergen,matteo,daan,tino}@idsia.ch

Abstract. We introduce a new class of recurrent, truly sequential SVM-like devices with internal adaptive states, trained by a novel method called EVolution of systems with KERNel-based outputs (Evoke), an instance of the recent Evolino class of methods. Evoke evolves recurrent networks to detect and represent temporal dependencies while using SVM to produce precise outputs. Evoke is the first SVM-based mechanism learning to classify a context-sensitive language. It also outperforms recent state-of-the-art gradient-based recurrent neural networks (RNNs) on various time series prediction tasks.

1 Introduction

Support Vector Machines (SVMs) [1] are powerful regressors and classifiers that make predictions based on a linear combination of kernel basis functions. A limited way of applying existing SVMs to time series prediction [2, 3] or classification [4] is to build a training set either by transforming the sequential input into some static domain (e.g., a frequency and phase representation), or by considering restricted, fixed time windows of m sequential input values. One alternative presented in [5] is to average kernel distance between elements of input sequences aligned to m points. Of course such approaches are bound to fail if there are temporal dependencies exceeding m steps. In a more sophisticated approach by Suykens and Vandewalle [6], a window of m previous output values is fed back as input to a recurrent model with a fixed kernel. So far, however, there has not been any recurrent SVM that *learns* to create internal state representations for sequence learning tasks involving time lags of arbitrary length between important input events.

Our novel algorithm, EVolution of systems with KERNel-based outputs (Evoke), addresses such problems. It evolves a recurrent neural network (RNN) as a preprocessor for a standard SVM kernel. The combination of both can be viewed as an adaptive kernel learning a task-specific distance measure between pairs of input sequences. Evoke is a special instance of a recent, broader algorithmic framework for supervised sequence learning called Evolino: EVolution of recurrent systems with Optimal LINear Output [7, 8]. Evolino combines neuroevolution (i.e. the artificial evolution of neural networks) and analytical linear methods that are optimal according to various criteria. The underlying idea of Evolino is that often a linear model can account for a large number of properties of a sequence learning problem. Non-linear properties not predictable by the linear model are then dealt with by more general evolutionary optimization.

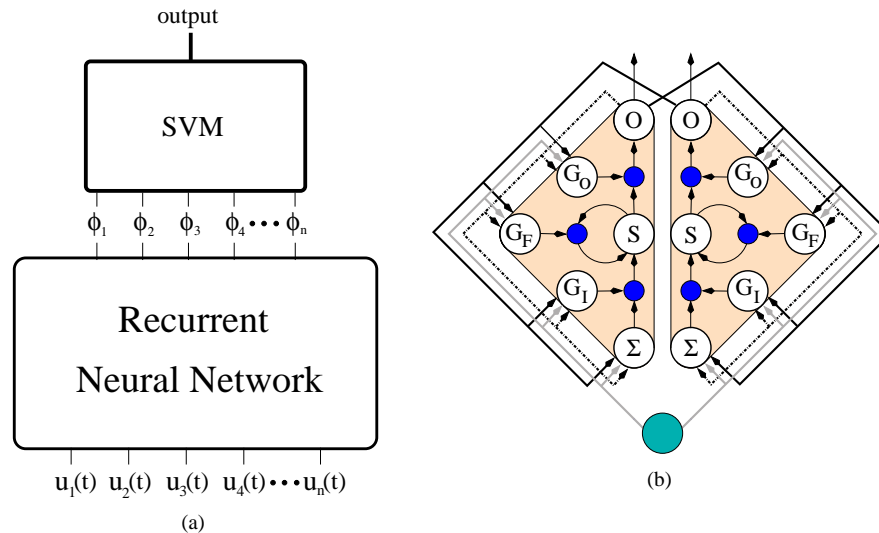


Fig. 1: (a) **Evoke network.** An RNN receives sequential inputs $\mathbf{u}(t)$ and produces neural activation vectors $\phi_1 \dots \phi_n$ at every time step t . These values are fed as input to a Support Vector Machine, which outputs a scalar $y(t)$. While the RNN is evolved, the weights of the SVM module are computed with support vector regression/classification. (b) **Long Short-Term Memory.** The figure shows the LSTM architecture that we use for the RNN module. This example network has one input (lower-most circle), and two *memory cells* (two triangular regions). Each cell has an internal state S together with a Forget gate (G_F) that determines how much the state is attenuated at each time step. The Input gate (G_I) controls access to the cell by the external inputs that are summed into each Σ unit, and the Output gate (G_O) controls when and how much the cell's output unit (O) fires. Small dark nodes represent the multiplication function.

2 The Evoke Algorithm

Evolino systems are based on two cascaded modules: (1) a recurrent neural network that receives the sequence of external inputs, and (2) a parametric function that maps the internal activations of the first module to a set of outputs. In particular, an Evoke network is governed by the following formulas:

$$\phi(t) = f(\mathbf{W}, \mathbf{u}(t), \mathbf{u}(t-1), \dots, \mathbf{u}(0)), \quad (1)$$

$$y(t) = w_0 + \sum_{i=1}^k \sum_{j=0}^{l_i} w_{ij} K(\phi(t), \phi^i(j)), \quad (2)$$

where $\phi(t) \in \mathbb{R}^n$ is the activation at time t of the n units of the RNN, $f(\cdot)$, given the sequence of input vectors $\mathbf{u}(0) \dots \mathbf{u}(t)$, and weight matrix \mathbf{W} . Note that, because the networks are recurrent, $f(\cdot)$ is a function of the entire input history. The output $y(t) \in \mathbb{R}$ of the combined system can be interpreted as a class label, in classification tasks, or as a prediction of the next input $\mathbf{u}(t+1)$, in time-series prediction.

In order to find a \mathbf{W} that minimizes the error between $y(t)$ and the correct output, we use artificial evolution [9]. Starting with random population of real-numbered strings or *chromosomes* representing candidate weight matrices, we evaluate each candidate through the following two-phase procedure.

In the first phase, the aforementioned training set of sequence pairs, $\{\mathbf{u}^i, d^i\}, i = 1..k$, each of length l^i , is presented to the network. For each input sequence \mathbf{u}^i , starting at time $t = 0$, each pattern $\mathbf{u}^i(t)$ is successively propagated through the RNN to produce a vector of activations $\phi^i(t)$ that is stored as a row in a $n \times \sum_i^k l^i$ matrix Φ . Associated with each input sequence is a *target* d^i in D containing the correct output values for each time step. Once all k sequences have been seen, the weights w_{ij} of the kernel model (equation 2) are computed using support vector regression/classification from Φ to D , with $\{\phi^i, d^i\}$ as training set.

In the second phase, a validation set is presented to the network, but now the inputs are propagated through the RNN *and* the newly computed output connections to produce $y(t)$. The error in the classification/prediction or the *residual error*, possibly combined with the error on the training set, is then used as the fitness measure to be minimized by evolution. By measuring error on the validation set rather than just the training set, RNNs will receive better fitness for being able to generalize. Those RNNs that are most fit are then selected for reproduction where new candidate RNNs are created by exchanging elements between chromosomes and possibly mutating them. New individuals replace the worst old ones and the cycle repeats until a sufficiently good solution is found.

In this study, Evoke is instantiated using Enforced SubPopulations (ESP; [10]) to evolve Long Short-Term Memory (LSTM; [11]) networks. ESP differs from standard neuroevolution methods in that, instead of evolving complete networks, it *coevolves* separate subpopulations of network components or *neurons*.

LSTM is an RNN purposely designed to learn long-term dependencies via gradient descent. The unique feature of the LSTM architecture is the *memory cell* that is capable of maintaining its activation indefinitely (figure 1b). Memory cells consist of a linear unit which holds the *state* of the cell, and three gates that can open or close over time. The Input gate “protects” a neuron from its input: only when the gate is open, can inputs affect the internal state of the neuron. The Output gate lets the internal state out to other parts of the network, and the Forget gate enables the state to “leak” activity when it is no longer useful. The gates also receive inputs from neurons, and a function over their input (usually the sigmoid function) decides whether they open or close [11]. Hereafter, the term gradient-based LSTM (G-LSTM) will be used to refer to LSTM when it is trained in the conventional way using gradient-descent.

ESP and LSTM are combined by coevolving subpopulations of memory cells instead of standard recurrent neurons. Each chromosome is a string containing the external input weights and the Input, Output, and Forget gate weights, for a total of $4*(I+H)$ weights in each memory cell chromosome, where I is the number of external inputs and H is the number of memory cells in the network. There are four sets of $I + H$ weights because the three gates and the cell itself receive input from outside the cell and the other cells. ESP normally uses crossover to recombine neurons. However, for Evoke, where fine local search is desirable, ESP uses only mutation. The top quarter of

| Training data | G-LSTM | PI-Evolino | Evoke |
|---------------|--------|------------|--------|
| 1..10 | 1..29 | 1..53 | 1..257 |
| 1..20 | 1..67 | 1..95 | 1..374 |

Table 1: **Generalization results for the $a^n b^n c^n$ language.** The left column shows the set of legal strings used to train each method. The other columns show the set of strings that each method was able to accept after training. The results for G-LSTM are from [14], and for Evolino from [7, 8]. Average of 20 runs.

the chromosomes in each subpopulation are duplicated and the copies are mutated by adding Cauchy distributed noise to all of their weight values.

The support vector method used to compute the weights (w_{ij} in equation 2) is a large scale approximation of the quadratic constrained optimization, as implemented in [12]. For continuous function generation, *backprojection* (or *teacher forcing* in standard RNN terminology) is used, where the predicted outputs are fed back as inputs in the next time step, such that 1 becomes: $\phi(t) = f(\mathbf{W}, \mathbf{u}(t), y(t-1), \mathbf{u}(t-1), \dots, y(0), \mathbf{u}(0))$. During training and validation, the correct target values are backprojected, in effect “clamping” the network’s outputs to the right values. During testing, the network backprojects its own predictions.

3 Experimental Results

Experiments were carried out on two test problems: context-sensitive languages, and multiple superimposed out-of-phase sine waves. These tasks were chosen to highlight Evoke’s ability to perform well in both discrete and continuous domains. The first task is of the type standard SVMs cannot deal with at all; the second is of the type even the recent ESNs [13] cannot deal with.

3.1 Context-Sensitive Grammars

Standard SVMs, or any approach based on a fixed time window, cannot learn to recognize context-sensitive languages where the length of the input sequence is arbitrary and unknown in advance. For this reason we focus on the simplest such language, namely, $a^n b^n c^n T$ (i.e. strings of n *as*, followed by n *bs*, followed by n *cs*, and ending with the termination symbol T). Classifying exemplars of this language entails counting symbols and remembering counts until the whole string has been read. Since traditional SVMs cannot solve this task at all, we compare Evoke to the pseudoinverse-based Evolino, and the only pre-2005 subsymbolic learning machine that has satisfactorily solved this problem, namely, gradient-based LSTM [14].

Symbol strings were presented to the networks, one symbol at a time. The networks had 4 input units, one for each possible symbol: S for start, a , b , and c . An input is set to 1.0 when the corresponding symbol is observed, and -1.0 when it is not present. The network state was fed as input to four distinct SVM classifiers, and each was trained to predict one of the possible following symbols a , b , c and T .

Two sets of 20 simulations were run each using a different training set of legal strings, $\{a^n b^n c^n\}$, $n = 1..N$, where N was 10 and 20. The second half of each set was used for validation, and the fitness of each individual was evaluated as the sum of training and validation error, to be minimized by evolution.

LSTM networks with 5 memory cells were evolved, with random initial values for the weights between -5.0 and 5.0 . The Cauchy noise parameter α for both mutation and burst mutation was set to 0.1 , i.e. 50% of the mutations is kept within this bound. In keeping with the setup in [14], we added a bias unit to the Forget gates and Output gates with values of $+1.5$ and -1.5 , respectively. The SVM parameters were chosen heuristically: a Gaussian kernel with standard deviation 2.0 and capacity 100.0 . Evolution was terminated after 50 generations, after which the best network in each simulation was tested. The results are summarized in Table 3.1. While being superior for $N = 10$ and $N = 20$, the performance of Evoke degraded for larger values of N , for which both PI-Evolino and G-LSTM achieved better results.

3.2 Multiple Superimposed Sine Waves

In [15], the author reports that Echo State Networks [13] are unable to learn functions composed of multiple superimposed oscillators. Specifically, functions like $\sin(0.2x) + \sin(0.311x)$, in which the individual sines have the same amplitude but their frequencies are not multiples of each other. In [16], a pre-wired network of oscillators inspired by [17] learns the components of a such a function, but its performance degrades for closer frequencies. G-LSTM also has difficulties in solving such tasks quickly.

For this task, networks with 10 memory cells were evolved for 50 generations to predict 400 time steps of the above function, excluding the first 100 as washout time; fitness was evaluated summing the error over the training set (points 101..400) and a validation set (points 401..700), and then tested on another set of data points from time-steps 701..1000. This time the weight range was set to $[-1.0, 1.0]$, and a Gaussian kernel with standard deviation 2.0 and capacity 10.0 was used for the SVM.

On 20 runs with different random seeds, the average summed squared error over the test set (300 points) was 0.021 . On the same problem, though, pseudoinverse-based Evolino reached a much better value of 0.003 . Experiments with three superimposed waves, as in [7, 8], gave unsatisfactory results.

4 Conclusion

We introduced the first kernel-adapting, truly sequential SVM-based classifiers and predictors. They are trained by the Evoke algorithm: EVolution of systems with KERnel-based outputs. Evoke is a special case of the recent Evolino class of algorithms [7, 8] in which a supervised learning module (SVM in this case) is employed to assign fitness to the evolving recurrent systems that pre-process inputs. Our particular Evoke implementation uses the ESP algorithm to coevolve the hidden nodes of an LSTM RNN.

This versatile method can deal with long time lags between discrete events as well as with continuous time-series prediction. It is able to solve a context-sensitive grammar task that standard SVMs cannot solve even in principle. It also outperforms ESNs

and previous state-of-the-art RNN algorithms for such tasks (G-LSTM) in terms of generalization. Finally, Evoke also quickly solves a task involving multiple superimposed sine waves on which ESNs fail, and where G-LSTM is slow.

The present work represents a pilot study of evolutionary recurrent SVMs. As for its performance, Evoke was generally better than gradient-based LSTM, but worse than the pseudoinverse-based Evolino [7, 8]. One possible reason for this could be that the kernel mapping of the SVM component induces a more rugged fitness landscape that makes evolutionary search harder. Future work will further explore Evoke's limitations, and ways to circumvent them, including the co-evolution of SVM kernel parameters.

References

- [1] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.
- [2] S. Mukherjee, E. Osuna, and F. Girosi. Nonlinear prediction of chaotic time series using support vector machines. In J. Principe, L. Giles, N. Morgan, and E. Wilson, editors, *IEEE Workshop on Neural Networks for Signal Processing VII*, page 511. IEEE Press, 1997.
- [3] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In W. Gerstner et al., editor, *Artificial Neural Networks - ICANN 1997*, pages 999–1004. Springer, 1997.
- [4] Jesper Salomon, Simon King, and Miles Osborne. Framewise phone classification using support vector machines. In *Proceedings International Conference on Spoken Language Processing*, Denver, 2002.
- [5] H. Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama. Dynamic time-alignment kernel in support vector machine. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.
- [6] J. A. K. Suykens and J. Vandewalle. Recurrent least squares support vector machines. *IEEE Transactions on Circuits and Systems-I*, 47(7):1109–1114, 2000.
- [7] J. Schmidhuber, D. Wierstra, and F. J. Gomez. Evolino: Hybrid neuroevolution / optimal linear search for sequence prediction. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 853–858. Morgan Kaufman, 2005.
- [8] D. Wierstra, F. J. Gomez, and J. Schmidhuber. Modeling non-linear dynamical systems with Evolino. In *Proc. GECCO 2005, Washington, D. C.*, pages 1795–1802, New York, 2005. ACM Press.
- [9] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.
- [10] F. J. Gomez. *Robust Nonlinear Control through Neuroevolution*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, 2003.
- [11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [12] R. Collobert, S. Bengio, and J. Marithoz. Torch: a modular machine learning software library. Technical Report 02-46, IDIAP-RR, 2002.
- [13] H. Jaeger. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304:78–80, 2004.
- [14] F. A. Gers and J. Schmidhuber. LSTM recurrent networks learn simple context free and context sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.
- [15] H. Jaeger. The echo state approach to recurrent neural networks, 2004. — seminar slides, available at <http://www.faculty.iu-bremen.de/hjaeger/courses/SeminarSpring04/ESNStandardSlides.pdf>.
- [16] Vlad Trifa. Associative memory models based on coupled oscillators, 2005. — Student project. See <http://birg.epfl.ch/page57466.html>.
- [17] L. Righetti, J. Buchli, and A.J. Ijspeert. Dynamic Hebbian learning in adaptive frequency oscillators. *Physica D*, 2005. In Press.