ESANN 2015 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 22-24 April 2015, i6doc.com publ., ISBN 978-287587014-8. Available from http://www.i6doc.com/en/.

Learning Recurrent Dynamics using Differential Evolution

Sebastian Otte[1]*, Fabian Becker[1], Martin V. Butz[2], Marcus Liwicki[3] and Andreas Zell[1]

1- University of Tuebingen - Cognitive Systems Group
Tuebingen - Germany
2- University of Tuebingen - Cognitive Modelling Group
Tuebingen - Germany
3- University of Kaiserslautern - Multimedia Analysis and Data Mining
Kaiserslautern - Germany

Abstract. This paper presents an efficient and powerful approach for learning dynamics with Recurrent Neural Networks (RNNs). No specialized or fine-tuned RNNs are used but rather standard RNNs with one fully connected hidden layer. The training procedure is based on a variant of Differential Evolution (DE) with a modified mutation schemey that allows to reduce the population size in our setup down to five, but still yields very good results even within a few generations. For several common Multiple Superimposed Oscillator (MSO) instances new state-of-the-art results are presented, which are across the board multiple magnitudes better than the achieved results published so far. Furthermore, for new and even more difficult instances, i.e., MSO9–MSO12, our setup achieves lower error rates than reported previously for the best system on MSO5–MSO8.

1 Introduction

Sequence generation is a key aspect of neural dynamic systems, e.g., generating trajectories for mobile robots, robot arm control, and so on. Most recently, a interesting Recurrent Neural Network (RNN) based system for hand-writing generation was developed by Alex Graves [1]. This and other applications demonstrate great potential of RNNs in terms of performance, generalization and robustness for sequence generation.

In this paper we contribute a simple method for learning dynamics with RNNs that in spite of its simplicity outperforms other more sophisticated approaches. The presented system is benchmarked on several instances of the commonly known Multiple Superimposed Oscillator (MSO). Learning to generate a single sine wave is a relatively easy task. However, learning a superimposition of more than one wave is clearly more difficult. In the past, learning even a two-wave superimposition with RNNs was considered as almost impossible [2]. Even if this was later disproved, the problem remained to be difficult due to the necessity to form multiple independent oscillators in one closed system [3]. Most recently, balanced Echo State Networks (ESNs) [4] were shown to be capable of learning MSO instances up to MSO8 very precisely. Important findings of

*Corresponding author, `sebastian.otte@uni-tuebingen.de` the latter are that the output feedback must be very small. On the other hand, neither the degree of connectivity nor the spectral radius in the reservoir were particularly crucial.

2 Recurrent Neural Networks

For this research we used standard recurrent neural networks. Furthermore, we avoided any topological pre-selection or optimization, e.g., like in Evolino [3], as well as other tuning mechanisms, except, that we scale the driving signal by a given factor. Our RNN architecture has one input cell, one linear output cell and one fully-connected hidden layer consisting of hyperbolic tangent cells. The internal dynamics are only driven by a scaled signal passed through the input cell. This driving signal is either a teaching signal (during washout and training phase) or output feedback (during test phase). How this works in particular is discussed in Section 4.

While the input and hidden weights are given as a candidate solution vector by the optimizer, the output weights are computed for each such vector via linear least squares like in, e.g., ESNs [2] or Evolino [3] thus matching the internal dynamics optimally by solving

$$\mathbf{X}\mathbf{w} = \mathbf{z} \Leftrightarrow \mathbf{X}^{+}\mathbf{X}\mathbf{w} = \mathbf{X}^{+}\mathbf{z} \Leftrightarrow \mathbf{w} = \mathbf{X}^{+}\mathbf{z}, \tag{1}$$

where $\mathbf{z} \in \mathbb{R}^{T}$ is the target sequence, $\mathbf{X} \in \mathbb{R}^{T \times n}$ is a matrix containing all hidden activations x_h^t and $\mathbf{w}$ is a the output weight vector mapping the hidden dynamics to the linear output neuron. In this paper Eq. (1) is solved using a singular value decomposition of $\mathbf{X}$.

3 Differential Evolution

Differential Evolution (DE) is a population-based evolutionary algorithm for global optimization [5] in continuous space. DE has shown to yield good results on a variety of benchmark functions as well as real world problems [6].

In its simplest form a new candidate solution, a so called donor vector $\mathbf{v}_{i,G}$ is created for each target solution $\mathbf{u}_i$, by randomly selecting three unique vectors $\mathbf{u}_{r_1,G}$, $\mathbf{u}_{r_2,G}$ and $\mathbf{u}_{r_3,G}$ from the current population of size NP in generation G.

$$\mathbf{v}_{i,G} = \mathbf{u}_{r_1,G} + F \cdot (\mathbf{u}_{r_2,G} - \mathbf{u}_{r_3,G}) \tag{2}$$

The differential weight F scales the difference vector, which is added to the first randomly selected vector. Elements of the donor vector enter the trial vector with probability CR. The trial vector is compared to the original target solution regarding their fitness and whichever yields a better result is admitted to the next generation.

However, several different and more sophisticated mutation schemes have been proposed in the past, see [6] for more details. For many problems the

`DE/target-to-best/1` scheme performed best as it is a good trade-off between convergence behavior and number of fitness evaluations.

$$\mathbf{v}_{i,G} = \mathbf{u}_{i,G} + F \cdot (\mathbf{u}_{best,G} - \mathbf{u}_{i,G}) + F \cdot (\mathbf{u}_{r_1,G} - \mathbf{u}_{r_2,G}) \tag{3}$$

Hereby, $\mathbf{u}_{best,G}$ refers to the best solution of the current generation. During our experiments we compared `DE/target-to-best/1` and other popular mutation schemes and studied the influence of the control parameters. Most of them showed sufficiently good results related to our problem, but performed not very efficiently, since they need a high number of problem evaluations (many generations and high population sizes). During this research we came up with the following mutation scheme that can be seen as an adapted version of the `DE/target-to-best/1` mutation.

$$\mathbf{v}_{i,G} = \mathbf{u}_{i,G} + F \cdot (\mathbf{u}_{best,G} - \mathbf{u}_{r_1,G}) + F \cdot (\mathbf{u}_{r_2,G} - \mathbf{u}_{r_3,G}) \tag{4}$$

In contrast to `DE/target-to-best/1` a random solution is subtracted from $\mathbf{u}_{best,G}$, while $\mathbf{u}_{i,G}$ still remains as base vector. Note that in `random-to-best` strategies usually also a random base is used. Surprisingly, this mutation scheme, which is to our knowledge until now not propagated in the common literature as a successful mutation scheme, results in a significantly better and deeper convergence, especially when using very small populations ($NP < 10$). Due to this, DE becomes a powerful tool for optimizing the hidden weights of the RNNs in our problem context.

4 Experiments

The MSO instances used for the experiments are generated using the equation

$$f_n(t) = \sum_{i=1}^{n} \sin(\varphi_i t), \tag{5}$$

where n gives the number of superimposed waves and φ_i each particular frequency. In this paper, MSO instances up to twelve waves are investigated using the following frequencies $\varphi_1 = 0.2$, $\varphi_2 = 0.311$, $\varphi_3 = 0.42$, $\varphi_4 = 0.51$, $\varphi_5 = 0.63$, $\varphi_6 = 0.74$, $\varphi_7 = 0.85$, $\varphi_8 = 0.97$, $\varphi_9 = 1.08$, $\varphi_{10} = 1.19$, $\varphi_{11} = 1.27$, $\varphi_{12} = 1.32$. The first eight frequencies are taken from [4], whereas the remaining four are added by us. For instance, MSO5 contains the frequencies from φ_1 to φ_5, MSO6 contains the frequencies from φ_1 to φ_6 and so on.

The usual MSO benchmark consists of the first 700 time steps of the related dynamics. The very first 100 time steps are used as a washout phase. Here, the target signal of the previous time step is fed into the network but the network output is completely ignored. The next 300 time steps are the training phase, in which the target signal is also injected and the output prediction error is measured. The last 300 time steps are used as test phase, where the network is fed with its own output from each previous time step. An entire fitness evaluation works then as follows. The hidden weights provided by the optimizer are copied into the RNN. Afterwards a washout phase and a training phase is performed. Using the activation dynamics during the training phase the output weights are computed by solving Eq. (1). Then with this output weights copied into the RNN, again a washout phase and a training phase is performed. The performance of the second training phase is then used the fitness signal for the optimizer.

To measure the test phase performance as well as the fitness we compute the Normalized Root Mean Square Error (NRMSE). Given a target sequence $\mathbf{z}$ and the generated output sequence $\mathbf{x}$ the NRMSE is then calculated through

$$E(\mathbf{x}, \mathbf{z}) = \sqrt{\frac{\sum_{1 \le t \le T} (z^t - x^t)^2}{T\sigma_z^2}}. \tag{6}$$

In order to get more objective results we used the same training setup, except the hidden layer size, for all MSO instances without any particular tuning. The input for the RNNs is generally scaled with 10^{-12}, matches findings of [4]. The DE population is initialized with values on the default interval $[-0.1, 1.0]$. Further, we used $F = 0.2$, $CR = 0.4$ and $NP = 5$, whereas the optimization was aborted after 1.000 generations. All experiments are performed with the JANNLab Toolkit [7].

5 Results and Discussion

Table 1 contains the error achieved with our system for various MSO instances (average over 10 runs) as well as previously reported results in the literature. For each instance we used an RNN with $5 \cdot n$ hidden neurons (n gives the number of frequencies). The results show very clearly, that the RNNs in our setup outperform the previous methods significantly on all listed dynamics. For example, on MSO8 we achieved an NRMSE of $6.14 \cdot 10^{-8}$ compared to $2.73 \cdot 10^{-4}$ achieved with a balanced ESN. In fact the NRMSE we yielded on MSO12 is still magnitudes better than the best NRMSE for MSO5 reported so far. Note that good

Table 1: Prediction error NRMSE on test phase for various MSO instances

Dynamics	This paper	[4]	[8]	[9]	[3]
MSO5	$4.16 \cdot 10^{-10}$	$1.06 \cdot 10^{-6}$	$2.54 \cdot 10^{-2}$	$\approx 8 \cdot 10^{-5}$	$1.66 \cdot 10^{-1}$
MSO6	$9.12 \cdot 10^{-9}$	$8.43 \cdot 10^{-5}$	–	–	–
MSO7	$2.39 \cdot 10^{-8}$	$1.01 \cdot 10^{-4}$	–	–	–
MSO8	$6.14 \cdot 10^{-8}$	$2.73 \cdot 10^{-4}$	$4.96 \cdot 10^{-3}$	–	–
MSO9	$1.11 \cdot 10^{-7}$	–	–	–	–
MSO10	$1.12 \cdot 10^{-7}$	–	–	–	–
MSO11	$1.22 \cdot 10^{-7}$	–	–	–	–
MSO12	$1.73 \cdot 10^{-7}$	–	–	–	–

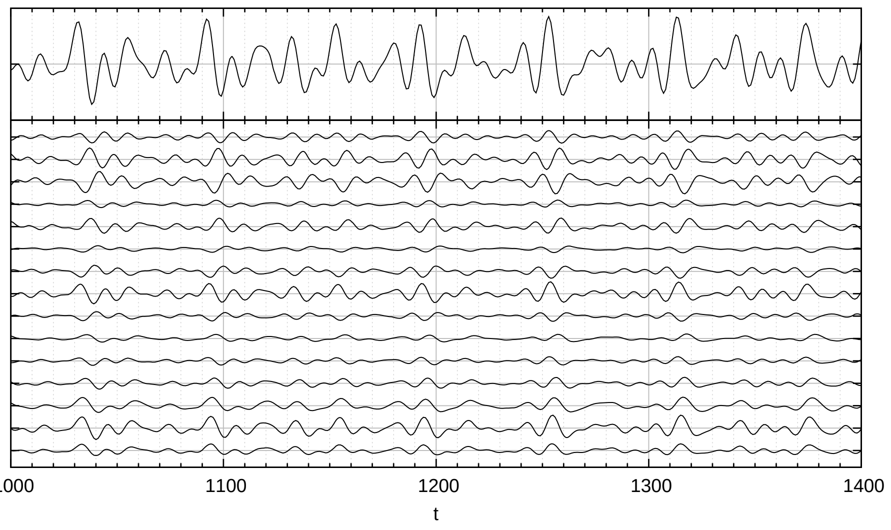

Fig. 1: Visualization of the output signal (upper curve) and the hidden dynamics in a exemplary RNN with 15 hidden neurons learned an MSO5 after oscillating for already 1.000 time steps. The output signal has a variance of ≈ 2.5 the internal dynamics have a variance of $\approx 3 \cdot 10^{-25}$.

error rates ($< 10^{-5}$) are reached even after only a few generations ($G \approx 20$), thus only 1–2 seconds are required to learn MSO12. When plotting the network output against ground truth, there is still no visual difference noticeable even after hundreds of thousands time steps (verified on MSO12), although the network was stimulated with the target signal only during the washout phase for this verification.

To get an idea of the interplay of the hidden neurons generating the output signal, Fig. 1 gives an insight to an exemplary RNN with 15 hidden neurons that learned MSO5. Cleary, individual groups of neurons do not reflect particular frequencies and amplitudes, but the output signal is intertwined in the 15 hidden neurons.

Furthermore, we made several other observations that are shortly outlined in the following. Using the network output as feedback signal during the training phase results in more stable networks concerning long-term generalization. Then training and test error differ not so much (only 1–2 magnitudes) as with teacher forced input (2–4 magnitudes). Note that our setup also works with linear hidden units. Hereby, the relatively small scale of the output-feedback is not required and can be omitted.

With a higher number of hidden neurons ($> 10 \cdot n$) very often networks occur that perform well without any optimization only with the randomly initialized weights, which can thus be seen as ESNs with 100 % connectivity. Their performance is not as high as with DE optimization (3–4 magnitudes lower), but noticeable better than known for ESNs on the MSO benchmark so far. On the other hand, using the same initialization as for DE the best randomly chosen RNN out of 1,000 yielded a poorly NRMSE of ≈ 1.77 in the test phase for MSO12. This confirms the optimization performance in our setup.

6 Conclusion

We presented a way that enables standard Recurrent Neural Networks (RNNs) without any specialized structural optimization or topological dependent fine tuning to learn dynamics very efficiently and precisely using a variant of Differential Evolution (DE) equipped with a modified mutation scheme. An interesting aspect of the proposed method is that the DE optimizer requires only very few individuals, namely five, throughout all of our experiments.

In our system, training the hidden weights is done via DE, whereas the output weights are determined using a least squares solution. Hereby, we achieved the best known results so far for various instances of the common Multiple Superimposed Oscillator (MSO) benchmark with up to twelve waves (more waves are possible as well). In fact our results are magnitudes better than previously reported values.

References

[1] A. Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850 [cs]*, August 2013.

[2] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. Technical Report GMD Report, 148, Fraunhofer Institute for Analysis and Information Systems AIS, Sankt Augustin, Germany, 2001.

[3] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent neural networks by Evolino. *Neural Computation*, 19:757–779, 2007.

[4] D. Koryakin, J. Lohmann, and M. V. Butz. Balanced echo state networks. *Neural Networks*, 36:35–45, 2012.

[5] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[6] S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31, Feb 2011.

[7] S. Otte, D. Krechel, and M. Liwicki. JANNLab Neural Network Framework for Java. In *Poster Proceedings Conference MLDM 2013*, pages 39–46, New York, USA, 2013. ibai-publishing.

[8] B. Roeschies and C. Igel. Structure optimization of reservoir networks. *Logic Journal of IGPL*, 18(5):635–669, 2010.

[9] G. Holzmann and H. Hauser. Echo state networks with filter neurons and a delay & sum readout. *Neural Networks*, 23(2):244 – 256, 2010.

[10] D. Koryakin and M. V. Butz. Reservoir sizes and feedback weights interact non-linearly in echo state networks. *Artificial Neural Networks and Machine Learning - ICANN 2012*, 7552:499–506, 2012.