Original scientific paper

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 43, No. 2(2013), 131 – 138

Transforming the LSTM training algorithm for efficient FPGA-based adaptive control of nonlinear dynamic systems

Rok Tavčar[1], Jože Dedič[1,2], Drago Bokal[1,3], Andrej Žemva[4]

[1]Cosylab d.d., Control Systems Laboratory, Ljubljana, Slovenia
[2]CO BIK, Solkan, Slovenia
[3]University of Maribor, Faculty of Natural Sciences and Mathematics
[4]University of Ljubljana, Faculty of electrical engineering and CO Namaste, Ljubljana, Slovenia

Abstract: In the absence of high-fidelity analytical descriptions of a given system to be modeled, designers of model-driven control systems rely on empirical nonlinear modeling methods such as neural networks. The particularly challenging task of modeling time-varying nonlinear dynamic systems requires from the modeling technique to capture complex internal system dynamics, dependent of long input histories. Traditional recurrent neural networks (RNNs) can in principle satisfy these demands, but have limitations on retaining long-term input data. Long Short-Term Memory (LSTM) neural networks overcome these limitations.
In applications with strict requirements imposed on the size, power consumption and speed, embedded implementations of control systems based on Field Programmable Gate Array (FPGA) technology are required. However, as neural networks are traditionally a software discipline, direct ports of neural networks and their learning algorithms into hardware give disappointing, often impractical results. To enable efficient hardware implementation of LSTM with on-chip learning, we present a transformation strategy which leads to replacing original LSTM learning algorithm with Simultaneous Perturbation Stochastic Approximation (SPSA). Our experimental results on a protein sequence classification benchmark confirm the efficacy of the presented learning scheme. The use of this scheme streamlines the architecture of on-chip learning phase substantially and enables efficient implementation of both forward phase and learning phase in FPGA based hardware.

Key words: model predictive control, control of nonlinear dynamic systems, recurrent neural networks, hardware neural networks, FPGA, LSTM, SPSA

Prilagoditev učenja nevronskih mrež LSTM za učinkovito realizacijo adaptivne regulacije nelinearnih dinamičnih sistemov v vezjih FPGA

Povzetek: V primerih kjer podroben analitični opis modela ni na voljo, snovalci modelno naravnanih regulacijskih sistemov potrebujejo empirične nelinearne metode modeliranja kot so umetne nevronske mreže. Modeliranje časovno spremenljivih, nelinearnih dinamičnih sistemov zahteva sposobnost posnemanja zapletene notranje dinamike procesa, pri čemer so izhodi modela odvisni od zgodovine vhodnih podatkov, raztezajoče se prek dolgih časovnih intervalov. Tradicionalne rekurentne nevronske mreže (ang. recurrent neural nework) v principu zadostijo tem zahtevam, ampak imajo omejitve pri pomnenju vhodov preko dolgih zakasnitev. Posebej z namenom premagati te omejitve so bile zasnovane mreže z dolgim kratkoročnim spominom (ang. Long Short-Term Memory, LSTM).
Mnoge aplikacije, ki imajo stroge zahteve po velikosti, hitrosti in porabi energije, zahtevajo namensko strojno izvedbo regulacijskega algoritma v polju programirljivih logičnih vrat (ang. Field Programmable Gate Array, FPGA). Ker so nevronske mreže tradicionalno disciplina splošnonamenske programske opreme, neposredna preslikava nevronskih mrež in njihovega algoritma za učenje v strojno opremo običajno prinese nepraktičen rezultat z zmogljivostmi pod pričakovanji. Z namenom učinkovite realizacije mrež LSTM z učenjem v strojni opremi, v tem delu predstavljamo prilagoditveno strategijo, ki motivira zamenjavo izvirnega učnega algoritma z algoritmom Simultaneous Perturbation Stochastic Approximation (SPSA). Učinkovitost delovanja mrež LSTM, učenih z SPSA, potrdimo s poskusi na znanem učnem problemu klasifikacije beljakovin. Nova kombinacija arhitekture nevronske mreže ter algoritma za učenje omogoča izjemne poenostavitve pri izvedbi tako testne faze kot učne faze v namenski strojni opremi, osnovani na tehnologiji FPGA.

Ključne besede: prediktivno vodenje, vodenje nelinearnih dinamičnih sistemov, rekurentne nevronske mreže, nevronske mreže v strojni opremi, FPGA, LSTM, SPSA

** Corresponding Author's e-mail: rok.tavcar@cosylab.com*

1 Introduction

Control of complex nonlinear dynamical systems demands advanced control methods such as Model Predictive Control (MPC). MPC is particularly challenging in the absence of a high-fidelity analytical description of the modeled system, a frequent reality in control of real-world systems. In such cases, designers must rely on empirical nonlinear modeling methods such as neural networks (NN) [1]. Neural-network-based modeling also plays an important role in control of time-varying systems, where NN learning is used to adapt to system-parameter changes over time [2]. Typical MPC tasks demand the model to capture complex internal system dynamics, dependent on long input histories. The type of neural networks that can satisfy these demands are Recurrent Neural Networks (RNNs) [3]. Because of their enhanced prediction qualities, they have been applied to numerous dynamic system control applications including speech recognition, phoneme recognition and chemical process identification [4]. However, traditional RNN models have limitations on retaining long-term input data. Long Short-Term Memory (LSTM) neural networks have been designed particularly to overcome these limitations by introducing architectural concepts which prevent exponential decay of input information over extended sequence lengths [5]. These concepts enable LSTM networks to learn patterns in sequences longer than 1000 steps, a 2 orders of magnitude improvement over traditional RNNs [6]. Figure 1 shows a basic unit of an LSTM network called a memory block.

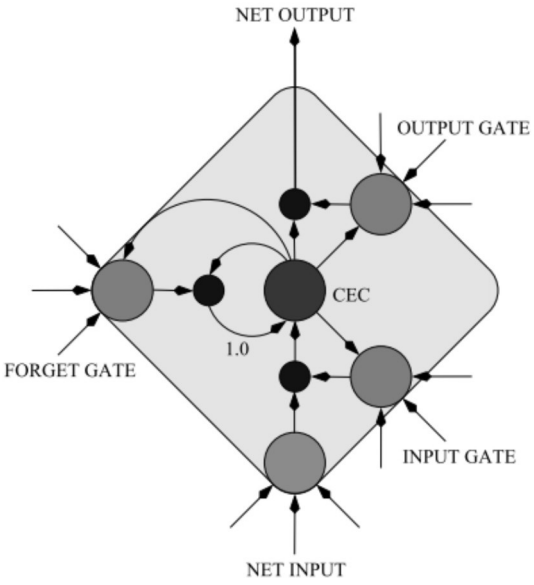

Figure 1: LSTM Memory Block: the memory cell with its constant error carousel (CEC) retains data over long input sequences. The update, output and erasure of this data are controlled by input, output and forget gates, respectively. Image source: [7].

It comprises several neuron-like components (a memory cell 'guarded' by gating units; input, output and forget gates), each with its own set of in- and outcoming weighted connections and a nonlinear activation function.

In control applications with strict requirements imposed on size, power consumption and speed, compact implementations of control systems in dedicated hardware are required. Due to the ceaselessly increasing density of Field Programmable Gate Arrays (FPGAs), along with their high degree of flexibility, they have become the technology of choice in a wide range of control applications [8]. However, NNs being traditionally a software discipline, direct ports of themselves and their learning algorithms into hardware give disappointing, often impractical results. Thus, algorithmic and architectural transformation techniques are crucial when porting neural networks into hardware [9].

In this work, we aim towards hardware-friendly implementation of LSTM with on-chip learning. This paper presents a strategy by which LSTM training is transformed and adapted in a way that reduces overall architectural complexity and allows hardware implementation with explicit exploitation of parallelism, avoiding mechanisms requiring complex circuitry.

Moreover, our proposed implementation strategy enables an independent architectural design of network forward phase and learning phase, leaving wide design freedom in choosing the implementation approach for each phase.

The validity of our approach is confirmed by experiments presented in this paper, showing that our proposed approach, i.e. learning of LSTM with Simultaneous Perturbation Stochastic Approximation (LSTM-SPSA), retains the ability of LSTM to learn sequences in data whilst delivering immense architectural benefits in terms of suitability for hardware implementation.

The paper is organized as follows. The rest of Chapter 1 briefs our mission statement and reviews related work. Chapter 2 explains the proposed transformation strategy and motivates the search of an alternative algorithm for LSTM learning. This search is laid out in Chapter 3, which also explains the chosen algorithm and emphasizes the advantages and drawbacks of the new learning scheme. Chapter 4 explains and discusses our experiments and results. Chapter 5 provides the conclusion and guidelines for future work.

1.1 Mission statement

In this work, we make the first attempt to transform LSTM and its learning rule to enable their efficient im-

plementation in dedicated hardware. At the time of writing, no account on research aiming at a hardware-native implementation of LSTM and its learning rule has yet been published.

We seek for the optimal strategy for efficient hardware implementation of both LSTM forward pass and on-chip learning.

We stress the importance of early architectural transformations upon porting software algorithms into dedicated hardware. We are led by the idea that an early, educated algorithm transformation will yield superior gains compared to a low-level, partial optimization of a design based on concepts unfit for dedicated hardware.

Investing effort to review alternative implementation options is crucial ground work that enables early architectural decisions that maximize future design fitness.

1.2 Related work

In the last two decades, extensive experimental and theoretical research effort has been aimed towards optimal hardware realization of different NN types and learning algorithms [9,10,11,12].

Systolic arrays [13, ch.5], [14] and stream processing architectures [15] minimize hardware idle-time and optimize dataflow and hardware efficiency. However, these approaches put strong constraints on the kind of neural architectures they can be applied to [13]. Logarithmic multipliers [16,17] spare hardware resources needed to implement expensive multipliers. Such optimization techniques gain priority when the benefits of higher-level algorithmic transformations have already been exploited. Limiting network weight levels to powers of two [18] replaces multipliers altogether with bitwise shifts. However, typical neural networks do not allow such modifications without significant performance loss. Cellular neural networks [19] and RAM-based NNs [9] are specifically designed with efficient hardware implementation in mind. However, their implementation principles (and their learning algorithms, also typically suitable for hardware) cannot be arbitrarily transferred to other network architectures, rendering these implementation principles unsuitable for applications requiring specific architectures. Perturbation algorithms and local learning algorithms [9] generalize well to different network architectures, and are well-suitable for hardware implementations. Perturbation algorithms do not put any assumptions on the neural network architecture, which is particularly beneficial when they are applied to architecturally complex neural networks such as LSTM.

Specifically for LSTM networks, no development on their architecture or their learning algorithm has yet been aimed at improving their suitability for hardware implementation. Improvements of LSTM are mainly focused towards improving their learning capacity [20,21] or convergence [22]. Research has yet to be made towards making LSTM networks and their learning suitable for dedicated hardware.

2 Criteria for selecting the transformation approach

In our search for conceptual transformations to LSTM and its learning on the algorithmic level, alternatives that bring the following benefits are sought for:

- decoupling the implementation of forward and backward pass to reduce implementation complexity, possibly without doubling the necessary hardware resources
- lowering the amount of expensive arithmetic operations
- lowering the complexity of control circuitry
- lowering the data dependencies between different algorithm parts (improving spatial locality)

To keep complexity of the hardware implementation at minimum, the implementation of on-chip learning should affect the implementation of the network's forward phase as little as possible. Ideally, the two phases should be completely decoupled, the only link between them being the data they both need in their operations (e.g. they both access the same memory storing network weights). In such case, the design of each phase can be treated separately, giving the designer more flexibility when choosing design approaches for either of them. However, being based on backpropagation, the LSTM backward pass is in the same order of architectural complexity as the forward pass, thus complete separation of the two phases could mean doubling the amount of required hardware resources. This motivates an architectural design where parts of the hardware are used by both phases; but that complicates the implementation process significantly compared to the case where each phase is treated independently. Consequently, we seek high-level transformations that allow the design of backward pass independently from the forward pass without a significant increase of the overall required hardware resources.

As the first step, we systematically analyze the findings of our literature search laid out in the previous chapter with respect to our criteria. As LSTM's advanced learning abilities stem from its architectural composition, we leave the neural network topology intact and focus on

augmenting the LSTM learning rule. We isolate hardware-friendly learning algorithms that generalize well to different neural network topologies and satisfy our criteria in several points. In subsequent steps of our research, these algorithms are analyzed in further depth.

3 Selecting the alternative training algorithm for LSTM

There are in principle two classes of hardware-friendly training algorithms: a) variations of widely-used but complex training algorithms with some of their core mechanisms altered or replaced and b) training algorithms that apply to hardware-friendly network architectures and are thus, in concept, fit for hardware themselves [11].

Because LSTM networks are a traditional multilayer network architecture and original LSTM training is based on backpropagation, it is best to look for algorithms close to its principles, focusing thus on the first class of learning algorithms. Their most successful representatives rely on some variety of parameter perturbation.

The general idea of perturbation algorithms is to obtain a direct estimate of the gradients by a slight random perturbation of network parameters, using the forward pass to measure the resulting network error. These on-chip training techniques do not only eliminate the complex backward pass but are also likely to be more robust to non-idealities occurring in hardware, such as a lowered numerical precision [9]. Mainly two variations exist: node perturbation and weight perturbation. Examples of node perturbation algorithms are Madaline-3 and Madaline-2.

We choose weight perturbation algorithms because of the lower complexity of their addressing and routing circuitry compared to node perturbation algorithms. Specifically, we look into two fully parallel versions of weight perturbation algorithms, namely Simultaneous Perturbation Stochastic Approximation (SPSA) [23] and Alopex [24]. Both are local training algorithms which determine weight updates using only locally available information and a global error signal. Both algorithms are closely related, but unlike SPSA, Alopex relies on the principles of simulated annealing, which adds complexity to the calculation of each weight perturbation.

In contrast, SPSA uses a simple random distribution function to perform weight perturbations and then updates all weights using the same absolute value of the update. Neither algorithm makes any assumptions as to the neural network topology, thus both are conceptually fit for direct generalization to LSTM network architecture. Neither have yet been applied to the LSTM architecture, but have been demonstrated to successfully train simpler FFNNs and RNNs [24, 25, 26], which motivates us to research their applicability for LSTM training. Because SPSA uses less parameters and computational steps to determine the update of each weight than Alopex, ultimately allowing a more streamlined hardware description, SPSA was selected as the algorithm of choice in this study.

3.1 LSTM-SPSA: LSTM trained by Simultaneous Perturbation Stochastic Approximation

SPSA [23] is based on a low-complexity, highly efficient gradient approximation that relies on measurements of the objective function, not on the gradient itself. The gradient approximation is based on only two function measurements, regardless of the dimension of the gradient vector, which is especially important in the field of neural networks, where this dimension quickly reaches several thousands. The weight-update scheme of the SPSA learning algorithm is explained by the following equations:

$$\Delta w_t^i = \frac{J(w_t + cs_t) - J(w_t)}{cs_t^i} \tag{1}$$

$$\Delta w_{t+1}^i = \begin{cases} w_{\max} & , \; if\,(w_t^i - a\Delta w_t^i) > w_{\max} \\ -w_{\max} & , if\,(w_t^i - a\Delta w_t^i) < -w_{\max} \\ w_t^i - a\Delta w_t^i , & otherwise \end{cases} \tag{2}$$

Here Δw_t and Δw_t^i denote the weight vector of a network and its i-th element at the t-th iteration, respectively, α is a positive constant and c is the magnitude of the perturbation. Δw^i represents the *i*-th element of the modifying vector.W_{max} is the maximum value of a weight. s_t and s_t^i denote a sign vector and its *i*-th element that is 1 or -1. The sign of s_t^i is determined randomly, with adherence to one of the recommended variable distributions. $J(w_t)$ denotes the criterion function, which is most frequently the Mean Square Error (MSE) between the network's actual and desired output.

From Eqs. 1 and 2 we see that a) during weight update, the *same absolute value* is used to update *all* network weights and b) to compute this value, only *two measurements* of the error function are required, one obtained via forward pass with perturbed weights and one without perturbations. SPSA algorithm flowchart is shown in Figure 2.

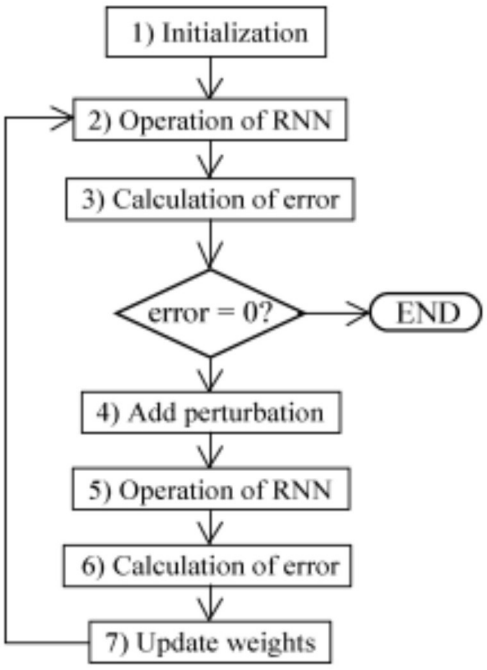

Figure 2: Flowchart of Simultaneous Perturbation Stochastic Approximation applied to Recurrent Neural Network Learning. Image source: [26].

The first advantage of SPSA over the original LSTM learning algorithm is simplification of the gradient estimation, because of the substantial reduction the number of arithmetical operations needed for weight updates. The second advantage, less obvious but equally important, is SPSA's equal treatment of all weights, eliminating in this way the separate error backpropagation paths (with different arithmetic expressions) required by different LSTM weight types, simplifying the algorithm routing circuitry significantly.

This second advantage in simplicity could prove to be a disadvantage in learning performance. For example, error backpropagation paths (set of weighted connections) that lead into forget gates, could have entirely different update dynamics than those leading into input gates. In original LSTM learning, this is accounted for; but not in SPSA. It is thus expected that SPSA algorithm will take longer to converge than original LSTM learning rule, but the increased simplicity of hardware implementation could compensate this by increasing operation speeds and possibilities of parallelization. An added benefit is also a simpler, more easily maintainable hardware description code.

3.2 Improving Learning Performance of LSTM-SPSA

After initial experiments with LSTM-SPSA (on the benchmark presented in the following chapter), possible augmentations to the learning algorithm were explored to maximize learning performance. The underlying idea of our augmentation was that if presented with a more difficult task, the algorithm will also improve on its basic task (minimize mean square error). For classification tasks such as ours, receiver operating characteristics curves (ROC) are better discriminative measures of classification performance than MSE [28]. Furthermore, the classification performance is in our experiments measured by AUC and AUC50 (area under ROC and ROC50 curve, respectively, presented briefly in the next chapter), [6], motivating the idea that the algorithm should also aim to maximize these scores.

To challenge our learning algorithm with a more difficult optimization task, we extended the criterion function by adding the AUC and AUC50 score, getting two new criterion functions. In addition to bringing MSE towards zero, the algorithm thus also had to maximize (bring to value of 1) AUC or AUC50. The two enhanced criterion functions used were:

$$J_{AUC}(w_t) = MSE + y * (1 - AUC) \text{ and}$$

$$J_{AUC50}(w_t) = MSE + y * (1 - AUC50)$$

using *y* as a scaling factor to tune the ratio between the MSE and AUC (or AUC50) in the score.

Because the AUC score can only be calculated at the end of a learning epoch, we needed to implement batch learning, applying cumulative weight updates at each learning epoch end. When using batch learning with the original criterion function, the performance of the learning algorithm did not change significantly compared to online learning. When adding ROC or ROC50 momentum to the criterion function, learning improved only by a few %, not reaching statistical significance.

4 Experimental results

Replacing the learning algorithm considerably interferes with the neural network's native learning scheme. Thus, before actual hardware implementation, the effectiveness of SPSA in training of LSTM has to be experimentally verified.

The most significant property of LSTM networks is their ability to retain temporal information in sequential input data. Therefore, we must test the LSTM-SPSA combination on a learning task that demands this ability. To allow for a back-to-back comparison with the original implementation, our experiments were based on those described in [6]. We implemented SPSA learning for LSTM networks and applied LSTM-SPSA to the

SCOP 1.53 database, which is a standard, widely used sequence-classification benchmark.

The preliminary experiments on a single SCOP 1.53 dataset, described in [27], showed promising learning results, indicating that SPSA-trained LSTM networks are able to learn temporal information over extended lengths of sequences.

For the main experiment, run on the complete SCOP 1.53 benchmark, we used pure SPSA with the original criterion function on an LSTM NN architecture identical to the one described in [6]. We used online learning, meaning that weight updates were computed and applied at the end of each sequence presented to the network within a learning epoch. In the experiment, the two SPSA learning parameters values used were c=0.0015 and $a = \frac{c}{2^3}$. In the generation of SPSA perturbation matrix, a Bernoulli distribution was used, as one of the recommended, optimal distributions for SPSA perturbations [29].

Table 1 shows the performance of different algorithms applied to SCOP 1.53 benchmark, showing that LSTM NNs outperform traditional algorithms for protein sequence classification in terms of classification quality, speed or both [6]. The quality of a ranking of test set examples for each protein family is evaluated by the area under the ROC curve. Being a more discriminiative quality measure, the area under ROC50 is also used; this is the area under the ROC curve up to 50 false positives, essentially rescaling the false positive rate of the ROC curve [6]. ROC and ROC50 scores for LSTM-SPSA show competitive learning performance of LSTM-SPSA towards other protein sequence classification algorithms. Because the forward phases of LSTM and LSTM-SPSA are identical, their test times, (Table 1, column 3) when run on software, are equal. Results in the table confirm that after replacing the original LSTM learning algorithm with SPSA, the learning ability of the LSTM NN architecture is preserved to a high degree. Because of the computational and architectural advantages of SPSA, explained in chapter 3, this motivates the use of LSTM-SPSA in hardware implementations of solutions that require the unique learning abilities of LSTM NN architecture.

Table 1: Results of remote homology detection on the SCOP benchmark database. The second and third column report the average area under the receiver operating curve ('ROC') and the same value for maximally 50 false positives ('ROC50'). The fourth column reports the time required to classify 20 000 test protein sequences (equivalent to one genome) into one superfamily. Performance data for solutions other than LSTM-SPSA sourced from [6].

Method	ROC	ROC50	Time
PSI-BLAST	0.693	0.264	5.5 s
FPS	0.596	-	6800 s
SAM-T98	0.674	0.374	200 s
Fisher	0.887	0.250	> 200 s
Mismatch	0.872	0.400	380 s
Pairwise	0.896	0.464	> 700 s
SW	0.916	0.585	> 470 s
LA	0.923	0.661	550 h
Oligomer	0.919	0.508	2000 s
HMMSTR	-	0.640	> 500 h
Mismatch-PSSM	0.980	0.794	> 500 h
SW-PSSM	0.982	0.904	> 620 h
LSTM	0.932	0.652	20 s
LSTM-SPSA	0.900	0.392	20 s

Figure 3 and Figure 4 show the total number of families for which a given algorithm exceeds a ROC or ROC50 threshold, respectively. Because of the rescaling of false positives in ROC50 score, giving it a higher discriminative value, the difference in performance between LSTM and LSTM-SPSA is more evident in Figure 4.

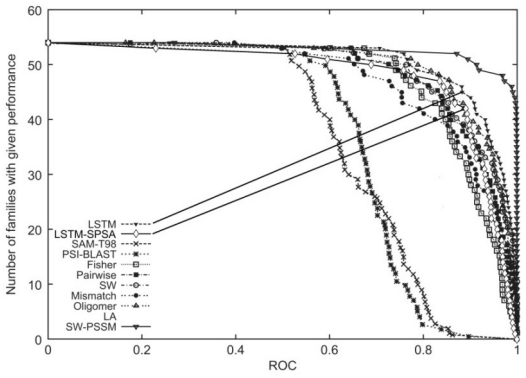

Figure 3: Comparison of homology detection methods for the SCOP 1.53 benchmark dataset. The total number of families for which a given method exceeds a ROC threshold is plotted. Performance data for solutions other than LSTM-SPSA sourced from [6].

Performance figures show that LSTM-SPSA exhibits competitive results compared to other protein classification techniques and compares to the original learning algorithm. This confirms that LSTM-SPSA retains the ability of LSTM networks to learn long sequences in data and, due to its substantial architectural advantages, that it is a viable scheme for implementing LSTM network abilities in dedicated hardware.

5 Conclusion

The work presented in this paper is the first attempt in transforming LSTM and its learning rule with the aim of

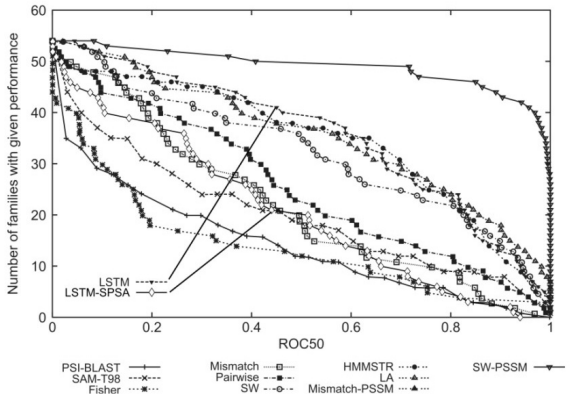

Figure 4: Comparison of homology detection methods for the SCOP 1.53 benchmark dataset. The total number of families for which a given method exceeds a ROC50 threshold is plotted. Performance data for solutions other than LSTM-SPSA sourced from [6].

improving its suitability for hardware implementation. Our transformation strategy is based on the premise that most gains can be achieved by high-level transformations of the algorithm on a conceptual level, which can mean completely replacing its vital parts with alternatives known to be suitable for hardware.

In our particular case, we have refrained from a naive direct port of a LSTM learning algorithm from software to hardware platform, bound to give disappointing results. Instead, we have replaced LSTM's backpropagation-based learning algorithm with Simultaneous Perturbation Stochastic Approximation, which fits our criteria for suitability for hardware implementation.

Our experiments confirm that LSTM-SPSA retains its ability to learn patterns in sequential data, which is the main characteristic of the LSTM network architecture. Due to promising results on a classification task, we expect that LSTM-SPSA could also demonstrate regression abilities. Our results show that LSTM-SPSA yields competitive results to the original learning algorithm, while enabling a cleaner implementation, lower resource utilization, simpler logical circuitry and increased parallelization of LSTM with on-chip learning.

Our strategy yields a solution which enables the designer to treat the forward phase and learning phase circuitry separately and to seek implementation strategies for each independently, giving a broader set of possibilities. Moreover, as SPSA is significantly less complex than the original algorithm, this decoupling does not bring a large increase of FPGA fabric consumption.

We conclude that because of the ability of SPSA in training LSTM on sequential data and because of its substantial advantages in suitability for hardware implementation, LSTM-SPSA is the recommended approach for dedicated hardware implementations of LSTM networks with on-chip learning.

In our future work, the effects of precision loss due to fixed-point arithmetic used in hardware will be studied. Preliminary experiments show that different fixed-point scaling should be used for different parts of the NN. Regression abilities of LSTM-SPSA will be explored. An attempt will be made to improve LSTM-SPSA learning either by using a modified SPSA which uses smoothed gradient or by using an adaptive learning rate. Independently from the learning phase, transformation techniques for LSTM forward phase will be reviewed.

Acknowledgements

Our research is in part funded by the European Union, European Social Fund. CO BIK, the Centre of Excellence for Biosensors, Instrumentation and Process Control and CO Namaste, Institute for research and development of Advanced Materials and Technologies for the Future, are operations funded by the European Union, European Regional Development Fund and Republic of Slovenia, Ministry of Education, Science, Culture and Sport.

References

1 T. Hayakawa, W. M. Haddad and N. Hovakimyan, "Neural network adaptive control for a class of nonlinear uncertain dynamical systems with asymptotic stability guarantees", Neural Networks, IEEE Transactions on, vol.19, no.1, pp. 80-89 2008.

2 H. Chaoui, P. Sicard and W. Gueaieb, "Ann-based adaptive control of robotic manipulators with friction and joint elasticity", Industrial Electronics, IEEE Transactions on, vol.56, no.8, pp. 3174-3187 2009.

3 R. K. Al Seyab and Y. Cao, "Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation", Journal of Process Control, vol.18, no.6, pp. 568-581 2008.

4 P. A. Mastorocostas and J. B. Theocharis, "A recurrent fuzzy-neural model for dynamic system identification", Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol.32, no.2, pp. 176-190 2002.

5 S. Hochreiter and J. Schmidhuber, "Long short-term memory", Neural Computation, vol.9, no.8, pp. 1735-1780, 1997/11/01 1997.

6 S. Hochreiter, M. Heusel and K. Obermayer, "Fast model-based protein homology detection without alignment", Bioinformatics, vol.23, no.14, pp. 1728-1736, July 15, 2007 2007.

7 A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures", Neural Networks, vol.18, no.5-6, pp. 602-610 2005.

8 E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan and M. W. Naouar, "Fpgas in industrial control applications", Industrial Informatics, IEEE Transactions on, vol.7, no.2, pp. 224-243 2011.

9 P. Moerland and E. Fiesler, "Neural network adaptations to hardware implementations", in Handbook of Neural Computation, 1997.

10 D. G. Bailey, "Invited paper: Adapting algorithms for hardware implementation", in Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on, pp. 177-184, 2011.

11 P. Moerland and E. Fiesler, "Hardware-friendly learning algorithms for neural networks: An overview", in Proceedings of the Fifth International Conference on Microelectronics for Neural Networks and Fuzzy Systems: MicroNeuro'96, IEEE Computer Society Press, Lausanne, Switzerland, 1996.

12 J. Misra and I. Saha, "Artificial neural networks in hardware: A survey of two decades of progress", Neurocomputing, vol.74, no.1-3, pp. 239-255 2010.

13 A. Omondi and J. C. Rajapakse, Fpga implementations of neural networks, Springer Netherlands, 2006.

14 R. G. Gironés, R. C. Palero, J. C. Boluda and A. S. Cortés, "Fpga implementation of a pipelined on-line backpropagation", The Journal of VLSI Signal Processing, vol.40, no.2, pp. 189-213 2005.

15 C. Farabet, C. Poulet and Y. LeCun, "An fpga-based stream processor for embedded real-time vision with convolutional networks", in Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on, pp. 878-885, 2009.

16 U. Lotrič and P. Bulić, "Logarithmic multiplier in hardware implementation of neural networks", Adaptive and Natural Computing Algorithms, pp. 158-168 2011.

17 U. Lotrič and P. Bulić, "Applicability of approximate multipliers in hardware neural networks", Neurocomputing, vol.96, no.0, pp. 57-65 2012.

18 M. Marchesi, G. Orlandi, F. Piazza and A. Uncini, "Fast neural networks without multipliers", Neural Networks, IEEE Transactions on, vol.4, no.1, pp. 53-62 1993.

19 L. Fortuna, P. Arena, D. Balya and A. Zarandy, "Cellular neural networks: A paradigm for nonlinear spatio-temporal processing", Circuits and Systems Magazine, IEEE, vol.1, no.4, pp. 6-21 2001.

20 F. A. Gers, J. Schmidhuber and F. Cummins, "Learning to forget: Continual prediction with lstm", Neural Computation, vol.12, no.10, pp. 2451-2471, 2000/10/01 2000.

21 F. A. Gers, N. N. Schraudolph and J. Schmidhuber, "Learning precise timing with lstm recurrent networks", Journal of Machine Learning Research, vol.3, pp. 115-143 2002.

22 J. Schmidhuber, D. Wierstra, M. Gagliolo and F. Gomez, "Training recurrent networks by evolino", Neural Computation, vol.19, no.3, pp. 757-779 2007.

23 J. C. Spall, "Implementation of the simultaneous perturbation algorithm for stochastic optimization", Aerospace and Electronic Systems, IEEE Transactions on, vol.34, no.3, pp. 817-823 1998.

24 K. Unnikrishnan and K. Venugopal, "Alopex: A correlation-based learning algorithm for feed-forward and recurrent neural networks", Neural Computation, vol.6, no.3, pp. 469-490 1994.

25 Y. Maeda and R. J. P. De Figueiredo, "Learning rules for neuro-controller via simultaneous perturbation", Neural Networks, IEEE Transactions on, vol.8, no.5, pp. 1119-1130 1997.

26 Y. Maeda and M. Wakamura, "Simultaneous perturbation learning rule for recurrent neural networks and its fpga implementation", Neural Networks, IEEE Transactions on, vol.16, no.6, pp. 1664-1672 2005.

27 R. Tavčar, "Design of neural networks for dedicated hardware implementation", in Proceedings of Microelectronics, Devices and Materials (MIDEM), International Conference on, MIDEM, 2012, Otočec, Slovenia, 2012.

28 J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms", Knowledge and Data Engineering, IEEE Transactions on, vol.17, no.3, pp. 299-310 2005.

29 P. Sadegh and J. C. Spall, "Optimal random perturbations for stochastic approximation using a simultaneous perturbation gradient approximation", Automatic Control, IEEE Transactions on, vol.43, no.10, pp. 1480-1484 1998.

Arrived: 07. 03. 2013
Accepted: 15. 05. 2013