GUESSING CAN OUTPERFORM MANY LONG TIME LAG ALGORITHMS

Technical Note IDSIA-19-96

Jürgen Schmidhuber
IDSIA
Corso Elvezia 36
6900 Lugano, Switzerland
juergen@idsia.ch
http://www.idsia.ch/~juergen

Sepp Hochreiter
Fakultät für Informatik
Technische Universität München
80290 München, Germany
hochreit@informatik.tu-muenchen.de
http://www7.informatik.tu-muenchen.de/~hochreit

May 6, 1996

Abstract

Numerous recent papers focus on standard recurrent nets' problems with long time lags between relevant signals. Some propose rather sophisticated, alternative methods. We show: many problems used to test previous methods can be solved more quickly by random weight guessing.

Introduction. The main problem of gradient-based, recurrent nets (see, e.g., overviews by Williams, 1989; Pearlmutter, 1995) is this: error signals "flowing backwards in time" tend to either blow up or vanish (for the first, detailed, theoretical analysis see Hochreiter, 1991 — the vanishing error case was later also treated by Bengio et al., 1994). That's why standard recurrent nets cannot deal with long time lags between relevant input/error signals. Rather sophisticated, alternative methods were proposed. For instance, Bengio et al. (1994) investigate simulated annealing, multi-grid random search, time-weighted pseudo-Newton optimization, and discrete error propagation. Bengio and Frasconi (1994) also propose an EM approach for propagating targets. Quite a few papers use Bengio et al.'s "2-sequence problem" (and "latch problem") to show the proposed algorithms's superiority, e.g., Bengio et al. (1994), Bengio and Frasconi (1994), El Hihi and Bengio (1995), Lin et al. (1995). For the same purpose, some papers also use the so-called "parity problem", e.g., Bengio et al. (1994), Bengio and Frasconi (1994). Some of Tomita's grammars (1982) are also often used as benchmark problems for recurrent nets (see, e.g., Bengio and Frasconi, 1995; Watrous and Kuhn, 1992; Pollack, 1991; Miller and Giles, 1993; Manolios and Fanelli, 1994). This paper exemplifies: such problems can be solved more quickly by random weight guessing than by the proposed algorithms.

Guessing. With a given architecture, random weight guessing works as follows: *REPEAT randomly initialize the weights UNTIL the resulting net happens to classify all training sequences correctly. Then test on a test set* (for more sophisticated guessing with bias towards nets with low Levin complexity, see Schmidhuber, 1995). In all our experiments, we randomly initialize weights in [-100.0,100.0]. Binary inputs are -1.0 (for 0) and 1.0 (for 1). Targets are either 1.0 or 0.0. All activation functions are sigmoid in [0.0,1.0]. We use two architectures (A1, A2) suitable for many widely used "benchmark" problems: A1 is a recurrent, fully connected net with 1 input, 1 output, and n biased hidden units. A2 is like A1 with $n = 10$, but less densely connected: each hidden unit sees the input unit, the output unit, and itself; the output unit sees all other units; all units are biased. We will indicate where we also use different architectures of other authors. All activations are set to 0 at each sequence begin. All sequence lengths are randomly chosen between 500 and 600 (most other authors actually facilitate their problems by using much shorter training/test sequences). The "benchmark" problems always require to classify two types of sequences. Our training set consists of 100 sequences, 50 from class 1 (target 0) and 50 from class 2 (target 1). Correct sequence classification is defined as "absolute error at sequence end below 0.1". We stop the search once a random weight matrix correctly classifies all training sequences. Then we test on the test set (100 sequences). All results below are averages of 10 trials. **In all our simulations below, guessing finally classified all test set sequences correctly; average absolute test set errors were always below 0.001 — in most cases below 0.0001.**

"2-sequence problem" (and "latch problem", e.g., Bengio et al., 1994; Bengio and Frasconi, 1994; Lin et al., 1995). The task is to observe and classify input sequences. There are two classes. There is only one input unit or input line. Only the first N real-valued sequence elements convey relevant information about the class. Sequence elements at positions $t > N$ (we use $N = 1$) are generated by a Gaussian with mean zero and variance 0.2. The first sequence element is 1.0 for class 1, and -1.0 for class 2. Target at sequence end is 1.0 for class 1 and 0.0 for class 2 (the latch problem is a simple version of the 2-sequence problem that allows for input tuning instead of weight tuning).

Bengio et al.'s results. For the 2-sequence problem, the best method among the six tested by Bengio et al. (1994) was multigrid random search (sequence lengths 50 — 100; no precise stopping criterion mentioned), which solved the problem after 6,400 sequence presentations, with final classification error 0.06. In more recent work (1994), Bengio and Frasconi were able to improve their results: an EM-approach was reported to solve the problem within 2,900 trials.

Results with guessing. Random guessing with architecture A2 (A1, $n = 1$) solves the problem within 718 (1247) trials on average. Using Bengio et al.'s 1994 architecture for the latch problem (only 3 parameters), the problem was solved within 22 (twenty-two) trials on average, due to tiny parameter space. Random guessing outperforms Bengio et al.'s methods in every respect: (1) many fewer trials required, (2) less computation time per trial. Also, in most cases (3) the solution quality is better (less error).

"Parity problem" (Bengio et al., 1994; Bengio and Frasconi, 1994). The task requires to classify sequences consisting of 1's and -1's according to whether the number of 1's is even or odd. The target at sequence end is 1.0 for odd and 0.0 for even.

Bengio et al.'s results. For sequences with only 25-50 steps, among the six methods tested by Bengio et al. (1994), only simulated annealing was reported to achieve final classification error of 0.000 (within about 810,000 trials — the authors did not mention the precise stopping criterion). A method called "discrete error BP" took about 54,000 trials to achieve final classification error 0.05. In Bengio and Frasconi's more recent work (1994), for sequences with 250-500 steps, their EM-approach took about 3,400 trials to achieve final classification error 0.12.

Results with guessing. Guessing with A1 ($n = 1$, identical to Bengio et al.'s 1994 architecture) solved the problem within 2906 trials on average. Guessing with A2 solved it within 2797 trials. We also ran another experiment with architecture A2, but without self-connections for the hidden units. Guessing solved the problem within 250 trials on average.

Tomita grammars. Many authors also use Tomita's grammars (1982) to test their algorithms. See, e.g., Bengio and Frasconi (1995), Watrous and Kuhn (1992), Pollack (1991), Miller and Giles (1993), Manolios and Fanelli (1994). Since we already tested parity problems above, we now focus on a few "parity-free" Tomita grammars (nr.s #1, #2, #4). Previous work facilitated the problems by restricting sequence length. E.g., Miller and Giles' maximal test (training) sequence length is 15 (10). Miller and Giles (1993) report the number of sequences required for convergence (for various first and second order nets with 3 to 9 units): Tomita #1: 23,000 – 46,000; Tomita #2: 77,000 – 200,000; Tomita #4: 46,000 – 210,000. Guessing, however, clearly outperforms Miller and Giles' methods. The average results are: Tomita #1: 182 (A1, $n = 1$) and 288 (A2), Tomita #2: 1,511 (A1, $n = 3$) and 17,953 (A2), Tomita #4: 13,833 (A1, $n = 2$) and 35,610 (A2).

Flat minima. It should be mentioned that successful guessing typically hits flat minima of the error function (Hochreiter and Schmidhuber, 1996).

Feedforward nets. It should also be mentioned that solutions to many well-known, simple, *nontemporal* tasks such as XOR can be guessed within less than 100 trials on numerous standard feedforward architectures.

Limitations of guessing. There are many tasks that require either many free parameters (e.g., input weights) or high weight precision, such that random search becomes completely infeasible (e.g., Schmidhuber's task, 1992). For such problems, we recommend to try a novel method called "Long Short Term Memory", or LSTM for short (Hochreiter and Schmidhuber, 1995). LSTM does not suffer from the above-mentioned problems of other gradient-based approaches. It can solve non-trivial, complex long time lag problems involving distributed, high-precision, continuous-valued representations.

Acknowledgments

This work was supported by *DFG grant SCHM 942/3-1* from "Deutsche Forschungsgemeinschaft".

References

Bengio, Y. and Frasconi, P. (1994). Credit assignment through time: Alternatives to backpropagation. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 75–82. San Mateo, CA: Morgan Kaufmann.

Bengio, Y. and Frasconi, P. (1995). An input output HMM architecture. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, pages 427–434. MIT Press, Cambridge MA.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

El Hihi, S. and Bengio, Y. (1995). Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems 8*. To appear.

Hochreiter, J. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München.

Hochreiter, S. and Schmidhuber, J. (1995). Long short term memory. Technical Report FKI-207-95, Fakultät für Informatik, Technische Universität München. Revised version submitted to Neural Computation, 1996.

Hochreiter, S. and Schmidhuber, J. (1996). Flat minima. *Neural Computation*. In press.

Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1995). Learning long-term dependencies is not as difficult with NARX recurrent neural networks. Technical Report UMIACS-TR-95-78 and CS-TR-3500, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742.

Manolios, P. and Fanelli, R. (1994). First-order recurrent neural networks and deterministic finite state automata. *Neural Computation*, 6:1155–1173.

Miller, C. B. and Giles, C. L. (1993). Experimental comparison of the effect of order in recurrent neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(4):849–872.

Pearlmutter, B. A. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269.

Pollack, J. B. (1991). The induction of dynamical recognizers. *Machine Learning*, 7:227–252.

Schmidhuber, J. H. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242.

Schmidhuber, J. H. (1995). Discovering solutions with low Kolmogorov complexity and high generalization capability. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA.

Tomita, M. (1982). Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Cognitive Science Conference*, pages 105–108. Ann Arbor, MI.

Watrous, R. L. and Kuhn, G. M. (1992). Induction of finite-state languages using second-order recurrent networks. *Neural Computation*, 4:406–414.

Williams, R. J. (1989). Complexity of exact gradient computation algorithms for recurrent neural networks. Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern University, College of Computer Science.